THE UNIVERSITY OF MANITOBA

ASPECTS OF SEQUENTIAL MACHINE SYNTHESIS

by

R. KNISPEL

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF MANITOBA

WINNIPEG, MANITOBA

October, 1977

ASPECTS OF SEQUENTIAL MACHINE SYNTHESIS

BY

RAY EDWARD KNISPEL

A dissertation submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

DOCTOR OF PHILOSOPHY
© 1977

ACKNOWLEDGMENTS

I would like to acknowledge Dr. Jon C. Muzio, my supervisor, for his valuable advice and encouragement and Dr. Douglas Lewin for his initial guidance and continued interest. The support provided by University of Manitoba Fellowships is also gratefully acknowledged.

DEDICATION

This thesis is dedicated to my parents for their support and to my wife, Deb, for her patience and understanding.

ABSTRACT

Sequential machine synthesis is the realization of finite state sequential machine behaviour in terms of hardware components. Two aspects of sequential machine synthesis are investigated in this thesis. The first aspect treats the decomposition of sequential machines, while the second explores possible medium and large scale integrated circuit realizations of sequential machines.

Algebraic structure theory, which provides the theoretical basis for the decomposition of sequential machines, is examined. A method for systematically generating the nonredundant sets of substitution property partitions (decompositions) is presented; the method is suitable for computer implementation. Two procedures for evaluating the decompositions generated are presented. Algebraic structure theory is also extended to incompletely specified machines, the synthesis of multiple machines, and asynchronous machines.

The use of transition matrices to synthesize a sequential machine as a single integrated circuit is considered. Synthesis algorithms for transition matrices and a new transition matrix are presented. Sequential machines can be realized as a cascade of combinational logic blocks. The choice of blocks affects the length of the resulting realizations and the complexity of the synthesis algorithm. Machine synthesis using a new set of combinational logic blocks is investigated.

TABLE OF CONTENTS

Chapter 1 Introduction

1.1 Thesis Outline

This thesis presents various methods of sequential machine synthesis. In particular, an algorithmic method of deriving all the decompositions for a machine is developed. (Sequential machines can be realized by a network of interconnected submachines. A network of submachines which realizes a machine M, and for which the connections between the submachines do not form a loop is called a loop-free realization or decomposition of M.) As the number of decompositions for a machine can be large, it is necessary to be able to determine the "best" decomposition, in some sense. Two different device independent methods of evaluating decompositions, uneconomic pairs and read only memory (ROM) evaluation, are presented. The uneconomic pairs technique eliminates decompositions which are uneconomical with relation to the other decompositions, while ROM evaluation determines the size of a ROM realization of each decomposition. ROM evaluation is preferred because of the greater choice it provides the logic designer.

Decomposition theory has only been developed for completely specified machines. As most practical machines are incompletely specified, we extend decomposition theory to the incompletely specified case. A theoretical approach is developed to overcome some of the problems encountered and a decomposition presented by Hartmanis [27] is adapted to incompletely specified machines. Application of decomposition theory to asynchronous machines and multiple machine synthesis is also examined.

Traditionally, machine realization has been done with discrete components, logic gates and memory elements. New technology, however, requires the use of complex, functional modules for realizations. Thus, functional modules which can independently realize machines, or can be interconnected to realize machines, must be developed. Two possible modules, derived from the transition matrix and generator sequence concept, are presented and synthesis techniques for each are developed.

The first module, a cellular array, is based on the transition matrix concept for sequential machines. We develop a universal cellular array which is derived from a new type of transition matrix. The new array can be used as a building block in realizing large sequential machines.

A novel concept in sequential machine realization, is the use of a set of elementary generators to construct a realization of a sequential machine. A synthesis algorithm for a generator set must be able to derive for any machine the most economical sequence of generators. A new generator set is proposed and a heuristic algorithm for obtaining generator sequences is presented. The new generator set, and its synthesis method, provide insight into the use of a large generator set for machine realization.

1.2 Sequential Machines

The effective design and implementation of sequential circuits (circuits with "memory") has become dependent on sequential machine theory. The advantage provided by sequential machine theory is that it allows a logic designer to view a sequential circuit as an abstract mathematical concept, independent of the physical realization of the circuit. This mathematical model provides the designer with a convenient means of specifying the behaviour of the sequential circuit he wishes to build. At the same time, the formalization of the problem as a sequential machine lends itself to various synthesis methods.

The use of sequential machine theory to facilitate the design of sequential circuits was first proposed by Huffman [36]. Prior to this, design methods for sequential circuits were applicable only to developing special purpose circuits with specified logic components. These techniques were not suited for designing the circuitry required by complex, digital systems. Huffman used the flow table as a means of stating a design problem and synthesizing the required circuit.

At the same time, Moore [56] developed an abstract formulation of a sequential machine. This work provided the basis for the development of sequential machine theory. Mealy [52] extended the work of Huffman and further generalized the theory of Moore.

The sequential machine, M, formulated by Moore can be characterized by the following:

(1) a finite set of states, $S = \{s_1,\ldots,s_n\}$;
(2) a finite set of input symbols, $IP = \{i_1,\ldots,i_q\}$;
(3) a finite set of output symbols, $OP = \{0_1,\ldots,0_r\}$;
(4) a mapping δ, called the next-state function, from $S \times IP \rightarrow S$;
(5) a mapping λ, called the output function, from $S \rightarrow OP$.

This model of sequential machine behaviour applies to both synchronous and asynchronous machines. The operation of a synchronous machine is synchronized by a clock pulse; changes in the input values are withheld from the machine until a clock pulse occurs. Thus, it appears to the machine that all inputs change on the clock pulse. An asynchronous machine, however, is not controlled by a clock pulse. Any changes in the input values are immediately received by the machine. The asynchronous mode of operation is potentially faster than the synchronous mode, but additional sophistication is required in the realization to handle the uncontrolled input changes. Throughout most of this thesis we shall be referring to synchronous sequential machines.

Associated with the formalization of the design process was the problem of deriving a near minimal realization of a machine. The synthesis technique used, that based on state assignment, consisted of assigning a set of binary variables to represent the set of states, and then computing a set of logic equations which realized the next-state and output functions. (It was assumed, in most cases, that the input and output values were fixed.)

The complexity of the resulting equations would vary, depending on the particular state assignment used. A measure of the complexity of a state assignment was obtained by counting the number of diodes and/or logic gates required by the state assignment. A minimal assignment could be obtained by enumerating all possible assignments. For any machine with more than a couple of states, this is not a feasible approach because of the vast number of possible assignments.

Some researchers, have subsequently taken advantage of "intuitive" properties of the state table representation of sequential machines, to obtain "good" state assignments. Armstrong [3] uses various types of adjacency relationships between states to develop a state assignment technique. This method, while providing reasonably economical realizations, fails to generate for some machines a truly simple realization, when such a realization is possible. A second, more comprehensive method, also due to Armstrong [4], produces better results. However, this method is more tedious to execute than the former. Dolotta and McCluskey [13] develop a state assignment technique based on codable columns, a subset of the possible columns for a machine. The method is effective for machines with 8, or less, states. For machines with more than 8 states, the number of codable columns is large and methods of reducing their number must be introduced. With the reduction, however, the method also loses a certain amount of efficiency.

These methods were oriented to finding "good" sum of products or product of sums, equations with delay elements for memory. Consequently, the methods must be modified if a different type of hardware is desired.

Another approach to state assignment, which will be introduced in Section 1.4, was based on the formal algebraic structure properties of sequential machines.

1.3 Algebraic Structure Theory

The algebraic structure theory of sequential machines was developed by Hartmanis [24],[25],[26],[27],[28], Hartmanis and Stearns [29],[30],[31], and Stearns and Hartmanis [63]. Certain aspects of structure theory, namely serial and parallel decompositions were obtained independently by Yoeli [72],[73]. (As the research of Hartmanis and Stearns has been consolidated in Hartmanis and Stearns [31], all subsequent references will be [31].)

Just as most tasks can be broken down into a collection of subtasks, some machines can be decomposed into a collection of submachines. The order of execution of the subtasks is important to the successful execution of the main task. Similarly, the order in which the submachines are connected is important to the realization of the initial machine.

Algebraic structure theory supplies a set of techniques and results which enable the designer to determine the submachines of a machine, how they must be connected, and also the "information flow" between the interconnected submachines. Essentially, structure theory provides a means of dividing a large sequential machine synthesis problem into a collection of smaller synthesis problems, which can be solved separately.

The benefits of this approach are:

(1) Machine synthesis tecnhiques can be applied with less difficulty to several, small machines than one large one. In particular state assignment techniques are more viable and efficient when applied to small machines.

(2) If complex functional modules are to be used to realize machines, it is more economical to use several small machines, than one large one.

(3) Testing several small machines for correct operation is simpler than testing one large machine.

(4) The designer gains a better understanding of how a machine functions when it is realized as a collection of separate, functional units.

The benefits of algebraic structure theory for machine realization are comparable to the benefits of modular programming techniques for computer programming. Structure theory has been developed independently of any particular hardware implementation. Thus, changes in the type of logic gates and memory elements available do not affect the application of structure theory.

The submachines of a machine correspond to partitions on the set of states, S, of a machine. (A partition on S is a collection of disjoint subsets of S whose union is S.) A submachine which can operate, receiving state information only from itself, corresponds to what is referred to as a substitution property (S.P.) partition. Hartmanis and Stearns have provided an exhaustive algorithm that generates all the S.P. partitions for a machine. Papers by Farr [14] and Yang [71] propose more systematic methods for generating S.P. partitions. Both methods have a similar purpose, namely that of eliminating, where possible, the enumerative techniques of Hartmanis and Stearns and substituting simpler and more efficient techniques. Sacco [60] presents a technique which simplifies the determination of whether or not a partition has the substitution property

Using S.P. partitions, loop-free realizations or decompositions can be constructed for sequential machines. The necessary conditions for a collection of S.P. partitions to form a decomposition of machine M are defined by structure theory.

Two submachines, one of which is dependent on state information from the other, correspond to what is known as a partition pair. A realization for a machine M using partition pairs must of necessity contain state information loops. In Chapter 2, algebraic structure theory and the background for determining loop-free realizations (decompositions) and non loop-free realizations are presented.

Although, structure theory provides a definition of what constitutes a decomposition, no rigorous algorithm for deriving the decompositions of a machine has been developed. Hartmanis and Stearns indicate with examples, how decompositions may be obtained by "inspection". Thus, the derivation of a decomposition is dependent on the designer's ability to apply structure theory. For small machines, minimal decompositions can be found quite readily. For large machines, however, derivation of a minimal decomposition by inspection is tedious, time consuming and prone to error. An attempt at developing an algorithm for deriving decompositions, Curtis [8], is critically examined in Chapter 2. Several inherent limitations of the algorithm are investigated. The major problems associated with the algorithm are:

(1) it is not universally applicable to all machines with S.P. partitions;

(2) decompositions derived by the algorithm may contain superfluous components.

Structure theory, as developed by Hartmanis and Stearns is not amenable to the development of a decomposition algorithm. Consequently, it has been necessary to extend structure theory in Chapter 3 in order to develop such an algorithm. The algorithm developed is simple to use and ensures the generation of all the decompositions which do not contain superfluous components. In addition, the algorithm is applicable to all sequential machines. Some suggestions for a computer implementation of the algorithm are provided in Chapter 4.

A method of evaluating decompositions is vital as there may be a great number generated for any machine. In Chapter 4, two methods of decomposition evaluation are presented. The first method eliminates pairs of S.P. partitions which are uneconomical with relation to other pairs of partitions. Thus, the decompositions generated, after the uneconomic pairs have been deleted, are the most economical decompositions for the machine.

The second method is based on the fact that read only memories (ROMs) can be used to realize sequential machines, Kvamme [48] and Howard [32]. A measure of the usefulness of any decomposition can thus be obtained by calculating the size of a ROM realization. Even though this method is based on determining the size of a ROM implementation, it is demonstrated that this evaluation technique is device independent.

Both methods agree as to which are the most economical decompositions of a machine. However, ROM evaluation is preferred, as the logic designer has a greater number of decompositions from which to select a realization.

1.4 Structure Theory Applications

Algebraic structure theory has been applied and extended to the study of sequential machines in many ways. An obvious application was the use of S.P. partitions and partition pairs for state assignment, Hartmanis [25] and Stearns and Hartmanis [63]. Kohavi [41] has introduced the implication graph as a means of producing economical state assignments using S.P. partitions. Partition pair theory has been used as a basis for state assignment by Curtis [8][9] and Weiner and Smith [70]. Karp [38] defines a critical pair (the partition pair concept extended to incompletely specified machines) and develops algorithms to derive state assignments using critical pairs.

Farr [14] has investigated various properties of common sequential machines using structure theory. In addition structure theory has been extended to the decomposition of stochastic automata, Bacon [6], and used for designing fault-detection experiments for sequential machines, Das [11].

Originally formulated for synchronous machines, structure theory has been applied to the decomposition of asynchronous machines, Tan et al [65] and Kinney [39], and also to the state assignment of asynchronous machines, Saucier [61]. In Chapter 5 we develop properties that relate S.P. partitions on asynchronous machines to the state assignment requirements of Tracey [67]. The properties developed have relevence to the work of Tan, et al [65].

A collection of p machines may have common component submachines. Obvious savings result if these machines can be realized as an interconnection of machines, where the common submachine is realized once, rather than p times. This possibility has been investigated for the case where the machines have the same inputs by Kohavi and Smith [44] and Smith and Kohavi [62] and for different inputs by Gestri [19]. The work of Kohavi and Smith has been formalized in Chapter 5. The properties of multiple sequential machines developed are used to provide a theoretical basis for Kohavi and Smith's work.

Hartmanis and Stearns have formulated conditions for an autonomous clock (input independent machine) to be a submachine of a sequential machine. Gryzmala-Busse [22] and Hwang [37] further examine this possibility. The decomposition of linear sequential circuits has also been examined by Ae and Yoshida [1] and Marino [51]

Although Hartmanis and Stearns [31] have examined the partition pair concept for incompletely specified machines, they have not done so for S.P. partitions. In Chapter 5, two methods by which S.P. partitions can be extended to incompletely specified machines are developed. A new way of connecting two S.P. partitions, using a decomposition introduced by Hartmanis [27], is presented. While this connection is applicable to completely specified machines, it is of more practical value when used with incompletely specified machines.

1.5 Cellular Synthesis

State assignment techniques for machine synthesis attempt to minimize the number of logic gates and memory elements. However, with present medium and large scale integrated (MSI and LSI) techniques the use of discrete components is not economical. The new technologies dictate the fabrication of more complex modules, Minnick [55]. The reasons for this are:

1) Less expensive; by combining multiple logic components on a single module the cost per component decreases.
2) Greater reliability; as the components in the module are already interconnected, the amount of intermodule wiring is reduced, resulting in fewer wiring errors. In addition, the reduction in wiring permits a miniaturization of circuits.

The availability of functional modules limits the application of minimal state assignment techniques, Lewin [49],[50]. (Clearly, these techniques are still important for the design of MSI and LSI modules.) Structure theory, however, remains relevent for synthesis with the new functional modules. In fact, for a large machine, savings result if instead of having to realize the machine as one large module, it can be realized as an interconnection of smaller modules.

Some of the research into sequential machine synthesis using the new technology concerns the determination of a module that is capable of realizing any sequential machine. If multiple copies of the module are to be fabricated on a single integrated circuit, it is desirable that for any machine realization the modules are connected in a regular pattern.

A synthesis method that can be used to realize a machine with a collection of uniform 2-state component machines is developed by Newborn [58]. A problem with this method is the irregular connection pattern of the modules. Using the same module, and techniques developed by Friedman [16], Arnold, Tan, and Newborn [5] present a synthesis method which enables a machine to be realized as a regular connection of the basic modules. A new module is developed in Newborn and Arnold [59], such that the fan out of any signal in the circuit is bounded. The synthesis method for this module also produces realizations with regular connection patterns. The modules in the above [5][58][59] are intended for binary input-binary output machines and are fairly simple. However, for machines with multiple inputs and/or multiple feedback realizations, the necessary modules become complex. To counteract this, Kukreja and Chen [46] introduce a new module which can be used to realize any machine, without the complexity of the module increasing.

Studies on machine realization using a uniform cellular array have been done by Hu [33] and Ferrarri and Grasselli [15]. Hu proposes a cellular array based on the transition matrix of a sequential machine. The array is uniform with regular interconnections between the individual cells. In Chapter 6 we review Hu's paper and develop some techniques which can be used to minimize the number of cellular arrays required for machine realization. A new cellular array is also presented which overcomes some of the problems associated with Hu's array.

Haring [23] and Menger [53] have developed a unique method for synthesizing sequential machines. The procedure is based on the fact that the transformation on a set of states required by a particular input can be implemented by a series of simple transformations or generators. Haring uses a set of 3 generators from which any input column in a sequential machine can be realized. Unfortunately, the method has the disadvantage that the number of generators required increases extremely rapidly as the number of states increases.

Menger overcomes this problem by using 2(n-1) generators, where n is the number of states. The result is that the number of generators required now becomes a linear function of n. The algorithm for deriving generator realizations obtains realizations which are minimal to within 2 generators.

In Chapter 7, following Menger's example, we expand the generator sets to n(n-1) generators and examine the properties of the new set. Expanding the generator set creates problems with determining absolute minimal realizations. However, a heuristic algorithm is developed which produces "good" realizations.

Huang [34][35] and Krishnan and Smith [45] present possible cellular implementations of Haring's and Menger's generators, respectively. The cellular array proposed by Huang has some desirable characteristics (identical cells and uniform interconnections) lacking in Krishnan and Smith's array. A new array, which implements Menger's generators and has the desirable characteristics mentioned, is developed in Chapter 7.

1.6 Notation and Definitions

The following general notation and definitions will be used throughout the thesis. More specific notation and definitions will be introduced when necessary.

A partition on a set S will be represented using bars and semicolons, rather than the usual set notation. For example, for set $S = \{1,2,3,4,5,6,7,8,9,10\}$, the partition τ consisting of the subsets $\{1,2,3\}$, $\{4\}$, $\{5,6,7,8\}$, and $\{9,10\}$ will be denoted as $\tau = \{\overline{1,2,3};\ \overline{4};\ \overline{5,6,7,8};\ \overline{9,10}\}$. The partition with only one element in each block and the partition with all elements in one block are the <u>0 and I partitions</u>, respectively. Collectively they will be referred to as the <u>trivial partitions</u>. For $S = \{1,2,3,4,5,6,7,8,9,10\}$, $0 = \{\overline{1};\ \overline{2};\ \overline{3};\ \overline{4};\ \overline{5};\ \overline{6};\ \overline{7};\ \overline{8};\ \overline{9};\ \overline{10}\}$ and $I = \{\overline{1,2,3,4,5,6,7,8,9,10}\}$.

Two elements $u, v \in S$ are defined to be <u>equivalent under partition τ</u>, <u>$u \equiv v(\tau)$</u>, if and only if u and v are contained in the same block of τ. Note that every partition defines an equivalence relation on S, and every equivalence relation defines a partition on S.

Partition multiplication and addition operations are defined as follows for τ_1 and τ_2, partition on S:

(i) $\tau_1 \cdot \tau_2$ is the partition on S such that $u \equiv v(\tau_1 \cdot \tau_2)$ if and only if $u \equiv v(\tau_1)$ and $u \equiv v(\tau_2)$.

(ii) $\tau_1 + \tau_2$ is the partition on S such that $u \equiv v(\tau_1 + \tau_2)$ if and only if there exists a sequence in S, $u = u_0, u_1, \ldots, u_n = v$ for which either $u_i = u_{i+1}(\tau_1)$ or $u_i \equiv u_{i+1}(\tau_2)$, for $0 \le i \le n-1$.

$\tau_1 \cdot \tau_2$ can be easily computed by intersecting the blocks of τ_1 and τ_2, while $\tau_1 + \tau_2$ requires an iterative process of successively combining blocks of τ_1 and τ_2.

For example, let $\tau_1 = \{\overline{1,2};\ \overline{3,4};\ \overline{5,6,7};\ \overline{8,10};\ \overline{9}\}$ and

$\tau_2 = \{\overline{1,5,7};\ \overline{2,6,9};\ \overline{3,4,10};\ \overline{8}\}$. Then

$\tau_1 \cdot \tau_2 = \{\overline{1};\ \overline{2};\ \overline{3,4};\ \overline{5,7};\ \overline{6};\ \overline{8};\ \overline{9};\ \overline{10}\}$ and

$\tau_1 + \tau_2 = \{\overline{1,2,5,6,7,9};\ \overline{3,4,8,10}\}$.

A partition τ_1 is said to be greater than or equal to partition τ_2, $\tau_1 \ge \tau_2$, if and only if every block of τ_2 is contained in a block of τ_1. Thus, $\tau_1 \ge \tau_2$ if and only if $\tau_1 \cdot \tau_2 = \tau_2$ if and only if $\tau_1 + \tau_2 = \tau_1$

For example, if $\tau_1 = \{\overline{1,2,3,4};\ \overline{5,6,7,8}\}$ and $\tau_2 = \{\overline{1,2};\ \overline{3,4};\ \overline{5,6};\ \overline{7,8}\}$ then $\tau_1 \ge \tau_2$. Also, $\tau_1 \cdot \tau_2 = \tau_2$ and $\tau_1 + \tau_2 = \tau_1$.

A binary relation R on the set S is a partial ordering of S if and only if R is:

(i) reflexive: that is, uRu for all $u \in S$;

(ii) antisymmetric: that is, uRv and vRu implies $u = v$;

(iii) transitive: that is, uRv and vRw implies uRw.

For a partial ordering, we shall use the more appropriate term, "$\ge$", to represent the relation R. A set with a partial ordering, $(S,\ge)$, is referred to as a partially ordered set. The set, U, of all partitions on S with the relation $\ge$ is a partially ordered set.

For a partially ordered set $(S,\ge)$ and V, a subset of S, v is the least upper bound (l.u.b) of V if and only if

(i) $v \ge x$ for all $x \in V$;

(ii) $v' \ge x$ for all $x \in V$ implies $v' \ge v$.

Similarly, v is the greatest lower bound (g.l.b.) of V, if and only if

(i) $v \le x$ for all $x \in V$;

(ii) $v' \le x$ for all $x \in V$ implies $v' \le v$.

A lattice is a partially ordered set $(S,\geq)$, which has a least upper bound and a greatest lower bound for every pair of elements in S. Clearly, the partially ordered set of all subsets of S, $(U,\geq)$, is a lattice where:

$$\text{lub } (\tau_1,\tau_2) = \tau_1+\tau_2 \text{ and}$$
$$\text{glb } (\tau_1,\tau_2) = \tau_1\cdot\tau_2.$$

We will denote the number of blocks in a partition τ by $\#(\tau)$ and the number of states in the largest block of τ by $\epsilon(\tau)$. Let $b(\tau) = [\log_2 \#(\tau)]$ and $e(\tau) = [\log_2 \epsilon(\tau)]$, where $[x]$ represents the smallest integer greater than or equal to x. For $\tau_i \geq \tau_j$, $\epsilon(\tau_i|\tau_j)$ represents the largest number of blocks of τ_j contained in a block of τ_i. Let $e(\tau_i|\tau_j) = [\log_2 \epsilon(\tau_i|\tau_j)]$. Note that Curtis [8] uses the notation $k\tau$, $\mu\tau$, and $k\tau_i,\tau_j$ to represent $b(\tau)$, $e(\tau)$, and $e(\tau_i|\tau_j)$, respectively.

Chapter 2 Sequential Machine Decomposition

2.1 Introduction

In this chapter we present the basic structure theory of sequential machines and examine an early attempt at state assignment using structure theory. A mathematical model of a sequential machine is given and it is shown how a sequential machine can be realized by a network of combinational logic and memory elements.

For some sequential machines, a subfunction of the machine's behaviour can be realized by a smaller machine or submachine. The submachine is defined by a special type of partition, known as a substitution property partition, on the set of states of the machine. If a machine has one or more substitution property (S.P.) partitions, it is possible to synthesize the machine as an interconnection of the submachines defined by the S.P. partitions.

The two basic connections of submachines, serial and parallel, are used in various combinations to produce networks of submachines. A network of submachines which imitates the behaviour of a machine M is said to be a decomposition of M. One class of networks can be characterized as not having any state information loops. In a loop-free network the information flow between machines is never circular.

The conditions which Hartmanis and Stearns have established for determining a loop-free network are presented. The only method provided by Hartmanis and Stearns to derive loop-free networks, is that of inspection of the lattice generated by the S.P. partitions.

The partition pair concept is introduced. Using partition pairs, networks with information loops can be constructed. The application of partition pairs to incompletely specified machines is also examined. An incompletely specified machine is a sequential machine for which some of the next-state and output conditions are not specified.

There has been only one attempt, Curtis [8], at developing a state assignment algorithm using the S.P. partition theory of

Hartmanis and Stearns. Curtis' method attempts to produce assignments which use the minimal number of state variables. Thus, only S.P. partitions which satisfy certain minimality requirements are retained for a state assignment. As a result, when an assignment is derived for a machine, many of the S.P. partitions are not considered. In fact, for many of the machines none of the S.P. partitions satisfy Curtis' requirements. Consequently, the algorithm is not universal for all machines with S.P. partitions.

Curtis also establishes conditions to test whether sets of S.P. partitions can be used in conjunction with each other to derive state assignments. These conditions ensure that a set of S.P. partitions will never require more than the minimal number of state variables.

However, as will be shown later, the algorithm Curtis derives from these conditions does not test for redundant partitions. Thus, sets of S.P. partitions containing superfluous partitions are derived by the algorithm. The state assignments resulting from these sets consequently require more than the minimal number of state variables. These and other problems associated with Curtis' method will be discussed in Section 2.6. In Chapter 3 an algorithm which overcomes the problems associated with Curtis' method is presented.

Sections 2.3, 2.4, and 2.5 are results from Hartmanis and Stearns [31] and as such will simply be stated without proof.

2.2. Machines and Substitution Property Partitions

Finite sequential machine theory provides an abstract mathematical model of logical computing circuits that have memory. The main characteristics of these circuits are:

(i) a finite set of inputs;
(ii) a memory which is used to control the operation of the machine;
(iii) a finite set of outputs.

The machine definition, which will be used to represent the circuits, uses a finite set of internal states in order to implement the machine memory.

Definition 2.1 (Hartmanis and Stearns) A Moore finite sequential machines is a quintuple,

$M = (S, IP, OP, \delta, \lambda)$, where

S is a finite nonempty set of states,

IP is a finite nonempty set of inputs,

OP is a finite nonempty set of outputs,

$\delta: S \times IP \to S$ is called the next-state function,

$\lambda: S \to OP$ is called the output function.

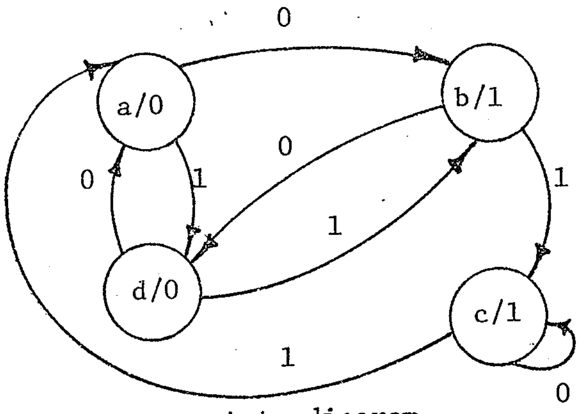
There is another type of sequential machine, the Mealy machine. The only difference between the two machines is the output function. For the Mealy machine the output function, $\lambda: S \times IP \to OP$, is a function of both state and input. The theory presented in this and following chapters deals with the next-state behaviour of sequential machines and as such applies equally to Mealy as to Moore machines. However for consistency we will use the Moore model.

Example 2.1 Let $M = (S, IP, OP, \delta, \lambda)$, where $S = \{a,b,c,d\}$, $IP = \{0,1\}$, and $OP = \{0,1\}$, be a Moore machine. The functions δ and λ can be represented using either a flow table or a state diagram.

	0	1	λ
a	b	d	0
b	d	c	1
c	c	a	1
d	a	b	0

M

flow table



state diagram

Columns 0 and 1 of the flow table represent the next-state function of M under inputs 0 and 1, respectively. The third column, λ, represents the output function values associated with each state.

The state diagram uses arrows, with input labels, connecting the various states to represent the next-state function. The output function is represented by associating an output value with a state.

From the state diagram and flow table representation, we can see that $\delta(c,1) = a$, $\lambda(c) = 1$, $\delta(b,0) = d$, $\lambda(b) = 1$, and so forth.

In some cases we will only be concerned with the next-state function of a machine.

Definition 2.2 (Hartmanis and Stearns) A State machine is a triplet,

$$M = (S, IP, \delta), \text{ where}$$

S is a finite nonempty set of states,

IP is a finite nonempty set of inputs,

$\delta: S \times IP \to S$ is called the next-state function.

The flow table representation for a state machine will be referred to as a state table.

Algebraic structure theory for sequential machines becomes useful for sequential circuit design when the abstract machine can be related to a physical circuit. The following definitions describe how one machine can be used to imitate another machine with only a renaming of the inputs and outputs.

Definition 2.3 (Hartmanis and Stearns) If M and M' are two machines, then the triple (α, β, γ) is said to be an assignment of M into M' if and only if

α is a mapping of S into subsets of S',

β is a mapping of IP into IP',

γ is a mapping of OP' into OP, and the mappings satisfy the conditions:

(i) $\delta'[\alpha(s), \beta(i)] \subseteq \alpha[\delta(s,i)]$, $\forall\ s \in S$ and $i \in IP$;

(ii) $\gamma[\lambda'(s')] = \lambda(s)$, $\forall\ s' \in \alpha(s)$.

Definition 2.4 (Hartmanis and Stearns) A machine M' is said to be a realization of machine M if and only if there is an assignment (α,β,γ) of M into M'. If M and M' are state machines, then we require mappings α and β satisfying condition (i).

Let M' be a realization of M through the assignment (α,β,γ). It can be shown that M' started in a state of $\alpha(s)$, behaves like M, under the interpretation of α and β, started in state s.

A sequential machine can be defined using binary variables.

Definition 2.5 (Hartmanis and Stearns) Input binary variables x_i, for $1 \le i \le m$
state binary variables y_j, for $1 \le j \le n$,
output binary variables z_k, for $1 \le k \le r$,
transition functions $Y_j: \{(y_1,\ldots,y_n,x_1,\ldots,x_m)\} \to (0,1)$,
and output functions $z_k: \{(y_1,\ldots,y_n)\} \to (0,1)$ define the binary sequential machine, $M = (S,IP,OP,\delta,\lambda)$ where

(i) $S = \{(y_1,\ldots y_n)\}$, the set of all n-tuples on (0,1). The present state of M is represented by the vector $\bar{y} = (y_1,\ldots,y_n)$ and the next state by $\bar{Y} = (Y_1,\ldots,Y_n)$.

(ii) $IP = \{(x_1,\ldots,x_m)\}$, the set of all m-tuples on (0,1). The input vector is $\bar{x} = (x_1,\ldots,x_m)$.

(iii) $OP = \{(z_1,\ldots,z_r)\}$, the set of all r-tuples on (0,1).

(iv) The next-state function is given by $\delta(\bar{y},\bar{x}) = Y(\bar{y},\bar{x})$, where $Y(\bar{y},\bar{x}) = (Y_1(\bar{y},\bar{x}),\ldots,Y_n(\bar{y},\bar{x}))$.

(v) The output function is given by $\lambda(\bar{y}) = Z(\bar{y})$, where $Z(\bar{y},\bar{x}) = (z_1(\bar{y}),\ldots,z_k(\bar{y}))$.

For any sequential machine, M, a binary sequential machine, M', can be devised such that M' is a realization of M. From M', it is easy to construct a physical sequential circuit, of logic gates and memory elements, which imitates the behaviour of M.

Example 2.2

	a	b	λ
1	3	2	0
2	1	2	1
3	3	1	1

D

Machine D' with the mappings (α,β,γ) is a realization for M.

y_1y_2	x=0	x=1	
00	01	11	0
10	01	11	0
11	00	11	1
01	01	10	1

D'

α(1) = {00,10}	β(a) = 0
α(2) = {11}	β(b) = 1
α(3) = {01}	γ(0) = 0
	γ(1) = 1

Machine D' can be realized by the circuit in Figure 2.1.

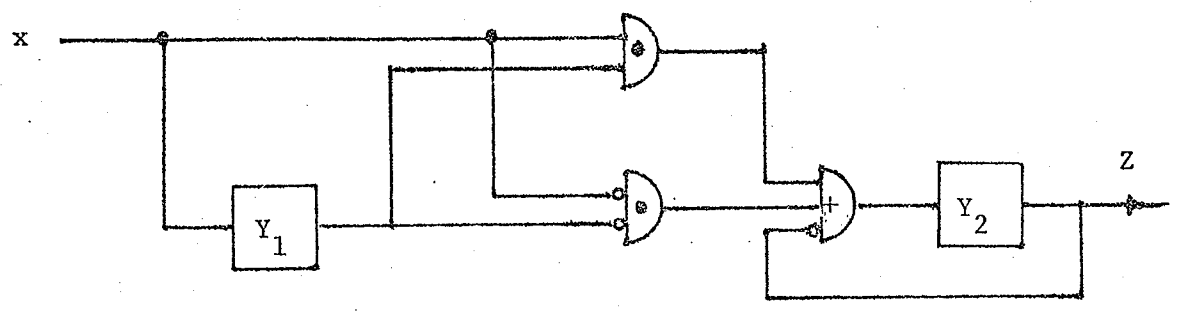


Figure 2.1

The Boolean equations for the circuit are

$$Y_1 = x$$
$$Y_2 = \bar{y}_2 + \bar{x}\,\bar{y}_1 + xy_1$$
$$Z = y_2$$

Of special interest for realizing sequential machines is the state behaviour realization.

<u>Definition 2.6</u> (Hartmanis and Stearns) Machine M' is said to <u>realize the state behaviour</u> of machine M if and only if M' realizes M with an assignment (α,β,γ), such that α maps each state of M onto a single state of M' and α is one-to-one. The mappings satisfy the following relations:

(i) $\delta'[\alpha(s), \beta(x)] = \alpha[\delta(s,x)], \forall s \in S$ and $x \in IP$;

(ii) $\gamma[\lambda'(\alpha(s))] = \lambda(s), \forall s \in S$.

State behaviour realizations ensure that only the minimal number of states will be required in the realizations.

<u>Substitution Property Partitions</u>

For some machines, a partition on the set of states can be used to define a new machine, which performs a subfunction of the original machine. The following definition provides a means of characterizing this type of partition.

<u>Definition 2.7</u> (Hartmanis and Stearns) A partition π on the set of states of the machine $M = (S,IP,OP,\delta,\lambda)$ is said to have the <u>substitution property</u> (S.P.) if and only if for states $s,t \in S$

$$s \equiv t(\pi)$$

implies that

$$\delta(s,i) \equiv \delta(t,i)(\pi), \quad \forall i \in IP .$$

Obviously, the trivial partitions 0 and I are S.P. partitions.

The submachine defined by an S.P. partition π, is represented as a state machine.

<u>Definition 2.8</u> (Hartmanis and Stearns) Let π be an S.P. partition on the set of states of the machine $M = (S, IP, OP, \delta, \lambda)$ Then the <u>π-image</u> of M is the state machine

$$M\pi = (\{B\pi\}, IP, \delta\pi), \text{ where}$$

$B\pi$ are the blocks of partition π and δ $(B\pi, i) = B\pi'$ if and only if $\delta(B\pi, i) \subseteq B\pi'$.

<u>Example 2.3</u>

	i_0	i_1	λ
1	3	7	0
2	4	8	0
3	1	6	0
4	2	5	0
5	2	4	0
6	1	3	1
7	4	4	1
8	3	3	0

M

$\pi_1 = \{\overline{1,2}; \overline{3,4}; \overline{5,6}; \overline{7,8}\}$ is an S.P. partition on machine M. Letting

$B_1 = \{1,2\}$, $B_2 = \{3,4\}$, $B_3 = \{5,6\}$, and $B_4 = \{7,8\}$,

the state machine $M\pi_1$ is defined as

	i_0	i_1
B_1	B_2	B_4
B_2	B_1	B_3
B_3	B_1	B_2
B_4	B_2	B_2

$M\pi_1$

For S.P. partitions, the operations of partition addition and partition multiplication are closed.

Theorem 2.1 (Hartmanis and Stearns) If π_1 and π_2 are S.P. partitions on the set of states of a sequential machine M, then so are the partitions $\pi_1 \cdot \pi_2$ and $\pi_1 + \pi_2$.

For Example 2.3, partitions $\pi_2 = \{\overline{1,2}; \overline{3,4,5,6}; \overline{7,8}\}$ and $\pi_3 = \{\overline{1,2,3,4}; \overline{5,6,7,8}\}$ are S.P. partitions.

$$\pi_2 \cdot \pi_3 = \{\overline{1,2}; \overline{3,4}; \overline{5,6}; \overline{7,8}\}$$
$$= \pi_1 \text{ is also an S.P. partition.}$$

$\pi_4 = \{\overline{1}; \overline{2}; \overline{3,6}; \overline{4,5}; \overline{7}; \overline{8}\}$ is an S.P. partition for M.

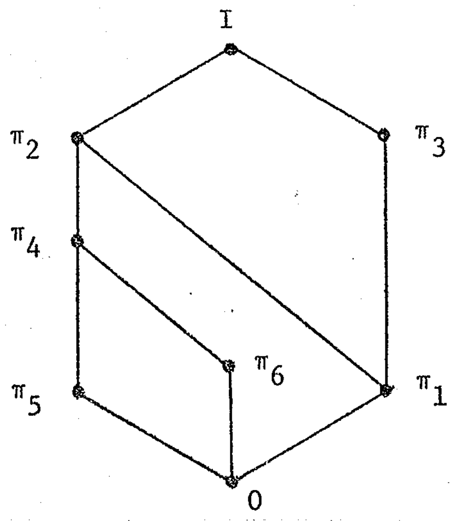
$$\pi_1 + \pi_4 = \{\overline{1,2}; \overline{3,4,5,6}; \overline{7,8}\}$$
$$= \pi_2 \text{ is also an S.P. partition.}$$

The binary relation "≤" forms a partial ordering of the set of all S.P. partitions for a machine M. Since $\pi_1 \cdot \pi_2 = \text{g.l.b.}(\pi_1,\pi_2)$ and $\pi_1+\pi_2 = \text{l.u.b.}(\pi_1,\pi_2)$, the partially ordered set $(P,\leq)$ is a lattice.

Example 2.4 The S.P. partitions for machine M of Example 2.3 are:

$$\pi_1 = \{\overline{1,2}; \overline{3,4}; \overline{5,6}; \overline{7,8}\}$$
$$\pi_2 = \{\overline{1,2}; \overline{3,4,5,6}; \overline{7,8}\}$$
$$\pi_3 = \{\overline{1,2,3,4}; \overline{5,6,7,8}\}$$
$$\pi_4 = \{\overline{1}; \overline{2}; \overline{3,6}; \overline{4,5}; \overline{7}; \overline{8}\}$$
$$\pi_5 = \{\overline{1}; \overline{2}; \overline{3,6}; \overline{4}; \overline{5}; \overline{7}; \overline{8}\}$$
$$\pi_6 = \{\overline{1}; \overline{2}; \overline{3}; \overline{4,5}; \overline{6}; \overline{7}; \overline{8}\}$$

The lattice formed by the S.P. partitions is



The S.P. partitions for a machine M can be calculated using an iterative procedure:

(1) For every pair of states, x and y, derive the smallest S.P. partition $\pi_{x,y}$ which identifies x and y.

(2) Obtain all possible pair-wise sums of the $\pi_{x,y}$ S.P. partitions.

(3) If any new S.P. partitions are produced, obtain all possible pair-wise sums of the augmented set of S.P. partitions.

(4) Repeat step (3) until no new S.P. partitions are produced.

The $\pi_{x,y}$ partitions are derived by identifying the states x and y and then identifying states linked by the next-state function δ. This process is demonstrated in the following example.

Example 2.5

	i_0	i_1	i_2
1	6	2	4
2	4	3	5
3	5	1	6
4	3	5	1
5	1	6	2
6	2	4	3

M

Calculate $\pi_{1,2}$

$\delta(1,i_0) = 6$ and $\delta(2,i_0) = 4$, implies states 4 and 6 are identified under input i_0. Similarly, under input i_1, states 2 and 3 are identified; under i_2, 4 and 5 are identified.

The sets {1,2} and {2,3} are not disjoint. Therefore, the next-state function identifies states 1,2, and 3. ($\{1,2\} \cup \{2,3\} = \{1,2,3\}$). Similarly, states 4,5, and 6 are identified.

Further states are identified by examining the states implied by the sets {1,2,3} and {4,5,6}. Since {1,2,3} implies

{4,5,6} and {1,2,3} and set {4,5,6} implies {1,2,3} and {4,5,6}, no further states are identified.

The resulting partition, $\pi_1 = \{\overline{1,2,3}; \overline{4,5,6}\}$ is an S.P. partition for M.

Identifying states 1 and 4 produces the S.P. partition $\pi_2 = \{\overline{1,4}; \overline{2,5}; \overline{3,6}\}$. No other nontrivial S.P. partitions can be produced by identifying the states of S.

Adding π_1 and π_2 produces the trivial S.P. partition I. Thus, the only nontrivial S.P. partitions for M are π_1 and π_2.

2.3. Abstract Networks and Decomposition

In this section we will look at some of the ways state machines can be interconnected to form larger, more complex machines. Conditions necessary for a large machine to be decomposed into a network of smaller submachines are also presented.

Definition 2.9 (Hartmanis and Stearns) An abstract network, N, of machines consists of:

(i) $\{M_i = (S_i, IP_i, \delta_i)\}$, i=1,...,n, a set of state machines referred to as component machines;

(ii) IP - a non-empty finite set of inputs;

(iii) OP - a non-empty finite set of outputs;

(iv) δ_i: $(\times S_j)\times IP \to IP_i$, $i \geq 1$, $j \leq n$, machine connecting rules;

(v) $g = (\times S_j) \to OP$, the output function, where $\times S_j$ represents a vector of states from the state sets of the M_i.

There are two basic ways of connecting state machines in an abstract network, serial and parallel.

Definition 2.10 (Hartmanis and Stearns) For two state machines $M_1 = (S_1, IP_1, \delta_1)$ and $M_2 = (S_2, IP_2, \delta_2)$, where $IP_2 = IP_1 \times S_1$, a set of output symbols OP with an output function λ: $S_1 \times S_2 \to OP$, the serial connection of M_1 and M_2, $M_1 \circledS M_2$, is the machine

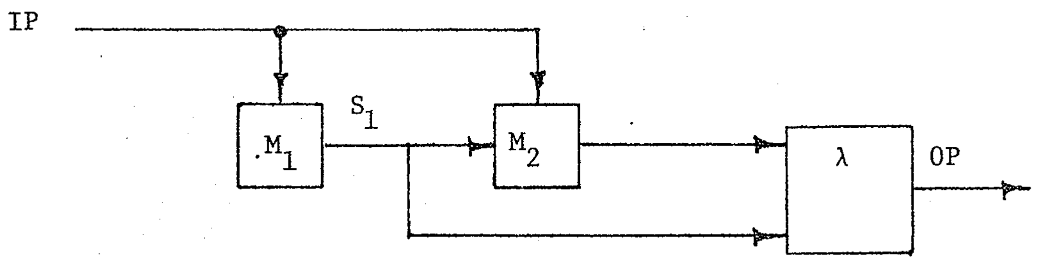
$M = (S_1 \times S_2, IP_1, OP, \delta, \lambda)$ where

$\delta[(s_1,s_2),i] = [\delta_1(s_1,i), \delta_2(s_2,(i,s_1))]$ for

all $i \in IP_1$, $s_1 \in S_1$, and $s_2 \in S_2$,

and $\lambda: S_1 \times S_2 \to OP$.

($\delta_1(s_1,i)$ and $\delta_2(s_2,(i,s_1))$ are the machine connecting rules.)



Serial connection of state machines M_1 and M_2

Figure 2.2

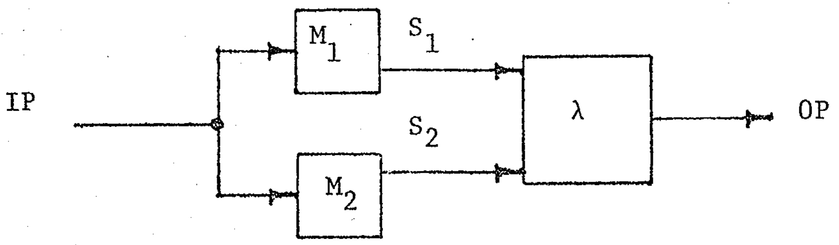
<u>Definition 2.11</u> (Hartmanis and Stearns) For two state machines $M_1 = (S_1, IP, \delta_1)$ and $M_2 = (S_2, IP, \delta_2)$, a set of output symbols OP with an output function $\lambda: S_1 \times S_2 \to OP$, the <u>parallel connection</u> of M_1 and M_2, $M_1 || M_2$, is the machine

$M = (S_1 \times S_2, IP, OP, \delta, \lambda)$, where

$\delta[(s_1,s_2),i] = [\delta_1(s_1,i), \delta_2(s_2,i)]$, for

all $i \in IP$, $s_1 \in S_1$, and $s_2 \in S_2$,

and $\lambda: S_1 \times S_2 \to OP$.



Parallel connection of state machine M_1 and M_2

Figure 2.3

A machine M has a serial decomposition of its behaviour if $M_1 \textcircled{$\rightarrow$} M_2$ is a state behaviour realization of M. Similarly, if $M_1 || M_2$ is a state behaviour realization of M, M has a parallel decomposition of its behaviour. The serial and parallel decompositions of a machine are related to the S.P. partitions for the machine.

Theorem 2.2 (Hartmanis and Stearns) A sequential machine M has a nontrivial serial decomposition of its state behaviour if and only if there exists a nontrivial S.P. partition π on the set of states S of M.

Example 2.6

	i_0	i_1
1	4	1
2	3	4
3	2	3
4	1	2

M

$\pi = \{\overline{1,3};\ \overline{2,4}\}$ is an S.P. partition on M.

Representing {1,3} by a and {2,4} by b, the π-image of M is

	i_0	i_1
a	b	a
b	a	b

$M\pi$

In order to realize M as a serial decomposition using π, another partition τ such that $\pi \cdot \tau = 0$ is required.

$\pi \cdot \tau = 0 = \{\bar{1};\bar{2};\bar{3};\bar{4}\}$ implies that the partition information contained in π and τ is sufficient to identify each state of M uniquely.

To realize τ, since τ is not an S.P. partition, state information from π is required as input.

Let $\tau = \{\overline{1,2}; \overline{3,4}; = \{\bar{c}; \bar{d}\}$

	i_0a	i_0b	i_1a	i_1b
c	d	d	c	d
d	c	c	d	c

$M\tau$

Machines $M\pi$ and $M\tau$ connected in serial realize M.

Theorem 2.3 (Hartmanis and Stearns) A sequential machine M has a nontrivial parallel decomposition of its state behaviour if and only if there exist two nontrivial S.P. partitions π_1 and π_2 on M such that $\pi_1 \cdot \pi_2 = 0$.

Example 2.7

	i_0	i_1
1	6	4
2	5	3
3	2	6
4	1	5
5	4	2
6	3	1

M

$\pi_1 = \{\overline{1,3,5}; \overline{2,4,6}\}$ and $\pi_2 = \{\overline{1,2}; \overline{3,4}; \overline{5,6}\}$ are S.P. partitions for state machine M.

Let $a = \{1,3,5\}$, $b = \{2,4,6\}$, $c = \{1,2\}$, $d = \{3,4\}$ and $e = \{5,6\}$.

The two π-image machines are

	i_0	i_1
a	b	b
b	b	a

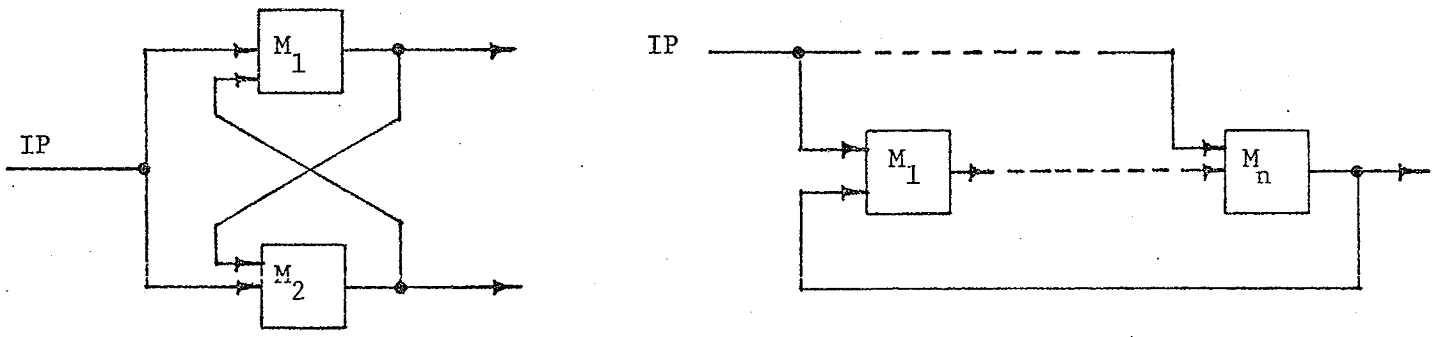
$M\pi_1$

	i_0	i_1
c	e	d
d	c	e
e	d	c

$M\pi_2$

Since $\pi_1 \cdot \pi_2 = 0$, $M\pi_1$ and $M\pi_2$ connected in parallel realize M.

These basic connections, serial and parallel, can be used as building blocks to construct combinations of machine connections. One class of machine connections can be characterized as being "loop-free". That is, the flow of state information between the machines never forms a loop, either directly or indirectly. The machine connecting rules represent the state information flow between machines.



direct loop indirect loop

Machine connections with state information loops

Figure 2.4

Hartmanis and Stearns define a class of sets of S.P. partitions which do not contain superfluous partitions.

Definition 2.12 (Hartmanis and Stearns) A set of partitions, $T = \{\pi_i\}$, defined on S is nonredundant if and only if the π_i are distinct and for all $T' \subseteq T$ and $\pi_k \in T$,

$$\Pi\{\pi_j | \pi_j \in T'\} \le \pi_k \text{ implies } \pi_k \ge \pi_i, \text{ for some } \pi_i \in T'.$$

A nonredundant set of S.P. partitions can be shown to correspond to a loop-free connection of component machines.

Example 2.8

	i_0	i_1	i_2	i_3	i_4
1	2	1	5	8	3
2	1	2	6	7	4
3	4	3	6	6	1
4	3	4	5	5	2
5	5	6	3	4	7
6	6	5	4	3	8
7	7	8	4	2	5
8	8	7	3	1	6

M

$\pi_1 = \{\overline{1,2,7,8}; \overline{3,4,5,6}\}$,

$\pi_2 = \{\overline{1,2,3,4}; \overline{5,6,7,8}\}$, and

$\pi_3 = \{\overline{1,4}; \overline{2,3}; \overline{5,8}; \overline{6,7}\}$ are S.P. partitions for M.

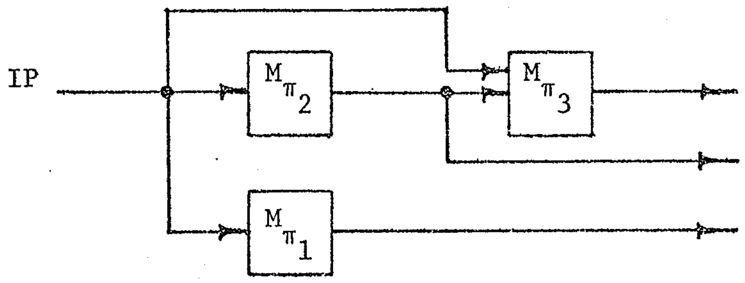
The set $T = \{\pi_1,\pi_2,\pi_3\}$ is nonredundant since Definition 2.12 is satisfied for all subsets of T.

e.g. $\Pi\{\pi_2,\pi_3\} = \pi_2 \cdot \pi_3$
$= \pi_3$

$\pi_3 \geq \Pi\{\pi_2,\pi_3\}$ and $\pi_3 \geq \pi_3$,
$\pi_2 \geq \Pi\{\pi_2,\pi_3\}$ and $\pi_2 \geq \pi_3$.

Since $\pi_1 \not\geq \Pi\{\pi_2,\pi_3\}$, it is not necessary to determine whether $\pi_1 \geq \pi_2$ or $\pi_1 \geq \pi_3$.

The connection of the machines is loop-free.



Since $\pi_1 \cdot \pi_2 \cdot \pi_3 = 0$, the connected machines realize M.

2.4. Partition Pairs and Incompletely Specified Machines

Another possible way of interconnecting state machines involves the use of information loops. Decompositions of this type can be found using the partition pair concept.

Definition 2.13 (Hartmanis and Stearns) A partition pair (π,π') on the machine $M = (S, IP, OP, \delta, \lambda)$ is an ordered pair of partitions on S such that

$$s \equiv t(\pi) \text{ implies } \delta(s,i) \equiv \delta(t,i)(\pi') \text{ for all } i \in IP.$$

Theorem 2.4 (Hartmanis and Stearns) If (π,π') and (τ,τ') are partition pairs on M, then

(i) $(\pi\cdot\tau,\ \pi'\cdot\tau')$ is a partition pair on M

(ii) $(\pi+\tau,\ \pi'+\tau')$ is a partition pair on M.

Example 2.9

	i_0	i_1	i_2	i_3	z
1	1	2	3	4	1
2	3	4	1	2	1
3	2	1	4	3	0
4	4	3	2	1	0

M

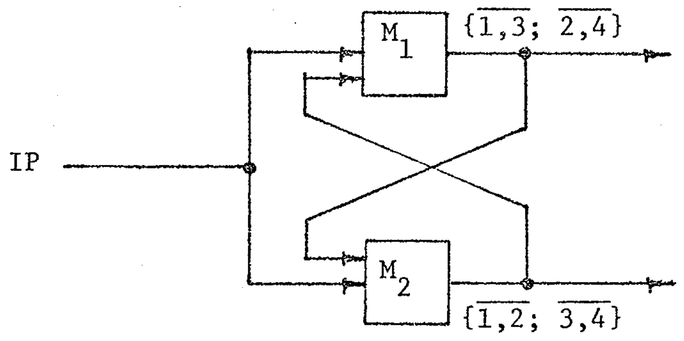
For M, $(\pi_1,\pi_2) = (\{\overline{1,2};\ \overline{3,4}\},\ \{\overline{1,3};\ \overline{2,4}\})$ and

$(\pi_2,\pi_1) = (\{\overline{1,3};\ \overline{2,4}\},\ \{\overline{1,2};\ \overline{3,4}\})$ are

partition pairs.

Since $\pi_1\cdot\pi_2 = \{\overline{1,2};\ \overline{3,4}\} \cdot \{\overline{1,3};\ \overline{2,4}\} = 0$,

M can be realized as a non loop-free connection of M_1 and M_2.


IP
M_1
M_2
$\{\overline{1,3};\ \overline{2,4}\}$
$\{\overline{1,2};\ \overline{3,4}\}$

M_1 produces state partition $\pi_2 = \{\overline{1,3};\ \overline{2,4}\}$ from partition $\pi_1 = \{\overline{1,2};\ \overline{3,4}\}$ and M_2 produces π_1 from π_2.

For a partition pair (π,π') there is a maximal front partition, $M(\pi')$, such that $M(\pi') \geq \pi$ and $(M(\pi'), \pi')$ is a partition pair. Similarly, there is a minimal second partition $m(\pi)$, such that $m(\pi) \leq \pi'$ and $(\pi, m(\pi))$ is a partition pair.

Definition 2.14 (Hartmanis and Stearns) If π is a partition on S of M, let

$$m(\pi) = \Pi \{\pi_i \mid (\pi,\pi_i) \text{ is a partition pair on } M\} \text{ and}$$

$$M(\pi) = \Sigma \{\pi_i \mid (\pi_i,\pi) \text{ is a partition pair on } M\} .$$

The minimal partition, $m(\pi)$, provides a means of justifying the method of generating S.P. partitions presented in Section 2.2.

Notation: For a partition τ let $m^o(\tau) = \tau$ and let $m^i(\tau) = m(m^{i-1}(\tau))$, for $i \geq 1$.

Theorem 2.5 (Hartmanis and Stearns) Given a machine M with a set of states S, there exists an integer K such that for all partitions τ on S and $k \geq K$,

$$\min\{\pi \mid \pi \geq \tau \text{ has S.P.}\} = \sum_{i=0}^{k} m^i(\tau) .$$

For some applications of sequential machines, some of the next-state and output values may not have to be specified. Values which do not have to be specified are known as don't-care conditions. A machine which has don't-care conditions is said to be incompletely specified.

Hartmanis and Stearns have applied the partition pair concept to incompletely specified machines using two different approaches. The first approach leaves thedon't-care conditions blank while the second arbitrarily gives each don't-care condition a unique name.

Definition 2.15 (Hartmanis and Stearns) If $M = (S,IP,OP,\delta,\lambda)$ is a machine with don't-care conditions and π and τ are partitions on S, we say that (π,τ) is a weak partition pair if and only if

$$s \equiv t(\pi) \text{ implies } \delta(s,i) \equiv \delta(t,i)(\tau) \text{ for all } i \in IP$$

such that $\delta(s,i)$ and $\delta(t,i)$ are specified.

Weak partition pairs are closed under partition pair multiplication but not partition pair addition.

Example 2.10

	i_0	i_1	i_2	z
1	4	1	3	-
2	-	2	4	0
3	1	3	4	1
4	2	3	-	0

M

$(\pi_1,\tau) = (\{\overline{1}; \overline{2,4}; \overline{3}\}, \{\overline{1}; \overline{2,3}; \overline{4}\})$ and

$(\pi_2,\tau) = (\{\overline{1}, \overline{2,3}; \overline{4}\}, \{\overline{1}; \overline{2,3}; \overline{4}\})$ are weak partition pairs.

However, $(\pi_1,\tau) + (\pi_2,\tau) = (\{\overline{1}; \overline{2,3,4}\}, \{\overline{1}; \overline{2,3}; \overline{4}\})$ is not a weak partition pair for M.

The second approach uses named don't-care conditions C and D, where next-state don't-care conditions are labelled with elements of C and output don't-care conditions with elements of D.

Definition 2.16 (Hartmanis and Stearns) Given a machine $M = (S,IP,OP,\delta,\lambda)$ with named don't-care conditions C and D, a partition π on S and partition τ on $S \cup C$, then (π,τ) is called an extended partition pair if and only if $s \equiv t(\pi)$ implies $\delta(s,i) \equiv \delta(t,i)(\tau)$ for all $i \in IP$.

Extended partition pairs are closed under both partition pair multiplication and addition.

Example 2.11 Naming the don't-care conditions of machine M in Example 2.10 with elements of C and D, where $C = \{c_1,c_2\}$ and $D = \{d_1\}$, gives

1	4	1	3	d_1
2	c_1	2	4	0
3	1	3	4	1
4	2	3	c_2	0

M'

$(\pi_1,\tau_1) = (\{\overline{1}; \overline{2,4}; \overline{3}\}, \{\overline{1}; \overline{2,3,c_1}; \overline{4,c_2}\})$ and
$(\pi_2,\tau_2) = (\{\overline{1}; \overline{2,3}; \overline{4}\}, \{\overline{1,c_1}; \overline{2,3}; \overline{4}; \overline{c_2}\})$ are extended partition partition pairs.

$(\pi_1,\tau_1) + (\pi_2,\tau_2) = (\{\overline{1}; \overline{2,3,4}\}, \{\overline{1,2,3,c_1}; \overline{4,c_2}\})$ is also an extended partition pair for M'.

2.5 State Assignment Using Curtis' Algorithm

Curtis [8] develops an algorithm to derive state assignments for sequential machines using the S.P. partitions of the machine. The algorithm attempts to obtain an assignment that uses the smallest number of state variables, with the least interdependence between variables. In order to formulate rules for the algorithm, Curtis proves a theorem which relates the ordering properties of self-dependent subsets of states and S.P. partitions. Before stating the theorem, some notation is introduced.

Notation: For a machine $M = (S,IP,OP,\delta,\lambda)$, we will use n to indicate the number of states in S. Thus, the minimal number of binary state variables needed to realize M is $s = [\log_2 n]$.

Theorem 2.6 (Curtis) Let $M = (S,IP,OP,\delta,\lambda)$ be a finite state sequential machine with n internal states. There is an assignment of the $s = [\log_2 n]$ binary state variables of M with two self-dependent subsets S_1 and S_2, having the covering property $S_2 \subset S_1$ if and only if there exists for M, partitions π_1 and π_2 with S.P., satisfying the conditions:

(1) $k\pi_1 + \mu\pi_1 = s;$

(2) $k\pi_2 + \mu\pi_2 = s;$

(3) $\pi_2 > \pi_1;$

(4) $k\pi_1 = k\pi_2,\pi_1 + k\pi_2.$

The rules derived from this theorem are essentially a logical extension of the theorem to sets containing two or more S.P. partitions. They are intended to ensure that only the minimal number of state variables are used in any state assignment derived by the algorithm. However, as will be demonstrated, this is not the case.

To obtain assignments with the minimal number of state variables, Curtis' algorithm first discards all S.P. partitions, π, for which $k\pi + \mu\pi > s$. That is, partitions which will require more than the minimal number of state variables are not used for state assignment.

Next, the retained partitions are divided into groups depending upon their k-value. The S.P. partitions with the lowest k-values are termed the maximal partition candidates. Partitions with the next lowest k-value are the first level submaximal partition candidates, and so forth. The algorithm then selects a state assignment from the various levels of partition candidates.

A brief description of the algorithm is now presented.

(i) Derive the set of S.P. partitions for a machine M;

(ii) Discard each partition for which $k\pi + \mu\pi > s$;

(iii) If there are two or more retained partitions, order them according to the size of their k-values;

(iv) From this ordered set determine the maximal partition candidates;

(v) Form a set of pairs of maximal partition candidates. Include each pair (π_i, π_j) whose l.u.b. is I, whose g.l.b. is a member of the retained set of partitions, and satisfies condition (4) of Theorem 2.6,

$$k\pi_i \cdot \pi_j = k\pi_i, \pi_i \cdot \pi_j + k\pi_i = k\pi_j, \pi_i \cdot \pi_j + k\pi_j \tag{1}$$

(vi) Form a set of triples of maximal partition candidates. Include a triple (π_i, π_j, π_k) in the set if the pairs $(\pi_i, \pi_j), (\pi_i, \pi_k)$, and (π_j, π_k) have been retained and the g.l.b. of (π_i, π_j, π_k) is a member of the retained set of partitions and satisfies condition (4) of Theorem 2.6,

$$\begin{aligned} k\pi_i \cdot \pi_j \cdot \pi_k &= k\pi_\ell, \pi_i \cdot \pi_j \cdot \pi_k + k\pi_\ell, \quad \ell = i,j,k \\ &= k\pi_i \cdot \pi_j, \pi_i \cdot \pi_j \cdot \pi_k + k\pi_i \cdot \pi_j \\ &= k\pi_i \cdot \pi_k, \pi_i \cdot \pi_j \cdot \pi_k + k\pi_i \cdot \pi_k \\ &= k\pi_j \cdot \pi_k, \pi_i \cdot \pi_j \cdot \pi_k + k\pi_j \cdot \pi_k \end{aligned} \tag{2}$$

Quadruples, quintuples, etc., if they exist, are found in a similar manner.

(vii) When no more combinations of maximal partitions can be found, the first level submaximal partitions are determined.

Each pair of first level submaximal partition candidates (π_i, π_j) is retained if its l.u.b. is a maximal partition candidate and satisfies condition (4) of Theorem 2.6,

$$k\pi_i = k\pi_i{+}\pi_j,\pi_i + k\pi_i{+}\pi_j$$
$$k\pi_j = k\pi_i{+}\pi_j,\pi_j + k\pi_i{+}\pi_j ,$$

its g.l.b. belongs to the set of retained partitions and satisfies (1).

Triples, quadruples, etc. of first level submaximal partition candidates are determined in a similar manner.

(viii) Further levels of submaximal partition candidates are found using rules (v), (vi), and (vii), with obvious modifications.

The k-value of a partition is a measure of its intervariable dependence. Thus, bv selecting the maximum number of maximal partitions, the algorithm derives state assignments which have the least amount of intervariable dependence. When no more maximal partitions can be included in an assignment, S.P. partitions with the next, least amount of variable dependence are considered for assignment, and so on. In this way, variables with greater variable dependence are only added if necessary to obtain a state assignment.

Example 2.12

	i_0	i_1	i_2	z
0	4	2	1	0
1	5	2	0	1
2	4	0	0	0
3	4	5	1	0
4	4	3	1	0
5	4	3	0	1

A

The S.P. partitions are

$$0 = \{\overline{0};\ \overline{1};\ \overline{2};\ \overline{3};\ \overline{4};\ \overline{5}\}$$
$$\pi_1 = \{\overline{0,1};\ \overline{2};\ \overline{3};\ \overline{4,5}\}$$
$$\pi_2 = \{\overline{0,1,2};\ \overline{3};\ \overline{4,5}\}$$
$$\pi_3 = \{\overline{0,3};\ \overline{1};\ \overline{2,5};\ \overline{4}\}$$
$$\pi_4 = \{\overline{0,1,4,5};\ \overline{2,3}\}$$
$$\pi_5 = \{\overline{0,1,3};\ \overline{2,4,5}\}$$
$$\pi_6 = \{\overline{0,1};\ \overline{2};\ \overline{3,4,5}\}$$
$$\pi_7 = \{\overline{0,1,2};\ \overline{3,4,5}\}$$
$$I = \{\overline{0,1,2,3,4,5}\}$$

S.P. partitions, π_2 and π_6 are not retained since,

$$k\pi_2 + \mu\pi_2 = k\pi_6 + \mu\pi_6 > 3 = s$$

retained partitions	k
I	0
π_4	1
π_5	1
π_7	1
π_1	2
π_3	2
0	3

The maximal partition candidates are π_4, π_5, and π_7. Since $\pi_4 + \pi_5 = I$, $\pi_4 \cdot \pi_5 = \pi_1$, and $k\pi_4{\cdot}\pi_5 = k\pi_4,\pi_4{\cdot}\pi_5 + k\pi_4 = k\pi_5,\pi_4{\cdot}\pi_5 + k\pi_5 = 2$, (π_4,π_5) is a valid pair of maximal partition candidates. Similarly, (π_4,π_7) and (π_5,π_7) are valid pairs.

The only possible triple of maximal partition candidates is (π_4,π_5,π_7). However, Curtis states that (π_4,π_5,π_7) is not a valid triple because

$$k\pi_4{\cdot}\pi_5{\cdot}\pi_7 = k\pi_1 = 2 \neq k\pi_4{\cdot}\pi_5,\pi_4{\cdot}\pi_5{\cdot}\pi_7 + k\pi_4{\cdot}\pi_5 = 1$$

But $\pi_4{\cdot}\pi_5 = \pi_1$ and $\pi_4{\cdot}\pi_5{\cdot}\pi_7 = \pi_1$

Therefore,

$$\begin{aligned} k\pi_4{\cdot}\pi_5,\pi_4{\cdot}\pi_5{\cdot}\pi_7 + k\pi_4{\cdot}\pi_5 &= k\pi_1,\pi_1 + k\pi_1 \\ &= 0 + 2 \\ &= 2 \end{aligned}$$

Thus, $k\pi_4{\cdot}\pi_5{\cdot}\pi_7 = k\pi_4{\cdot}\pi_5,\pi_4{\cdot}\pi_5{\cdot}\pi_7 + k\pi_4{\cdot}\pi_5$, contrary to Curtis' calculations.

It can easily be shown that (π_4,π_5,π_7) satisfies condition (2) and the other requirements of rule (vi)

$$\begin{aligned} k\pi_4{\cdot}\pi_5{\cdot}\pi_7 &= k\pi_4,\pi_4{\cdot}\pi_5{\cdot}\pi_7 + k\pi_4 = 1 + 1 = 2 \\ &= k\pi_5,\pi_4{\cdot}\pi_5{\cdot}\pi_7 + k\pi_5 = 1 + 1 = 2 \\ &= k\pi_7,\pi_4{\cdot}\pi_5{\cdot}\pi_7 + k\pi_7 = 1 + 1 = 2 \\ &= k\pi_4{\cdot}\pi_5,\pi_4{\cdot}\pi_5{\cdot}\pi_7 + k\pi_4{\cdot}\pi_5 = 0 + 2 = 2 \\ &= k\pi_4{\cdot}\pi_7,\pi_4{\cdot}\pi_5{\cdot}\pi_7 + k\pi_4{\cdot}\pi_7 = 0 + 2 = 2 \\ &= k\pi_5{\cdot}\pi_7,\pi_4{\cdot}\pi_5{\cdot}\pi_7 + k\pi_5{\cdot}\pi_7 = 0 + 2 = 2 \end{aligned}$$

No further sets of maximal partition canditates can be formed.

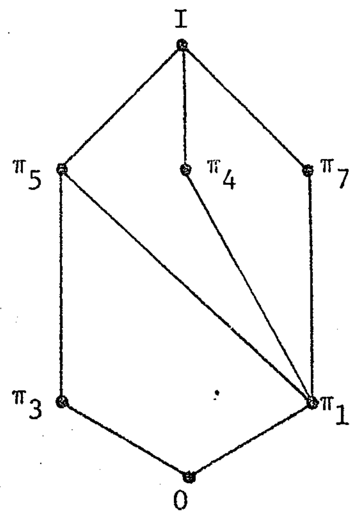
The only first level submaximal partition candidates are π_1 and π_3. The l.u.b. of π_1 and π_3 is π_5, a retained partition. Since,

$$\begin{aligned} k\pi_1 &= 2 = 1 + 1 = k\pi_1{+}\pi_3,\pi_1{+}k\pi_1{+}\pi_3 \quad \text{and} \\ k\pi_3 &= 2 = 1 + 1 = k\pi_1{+}\pi_3,\pi_3{+}k\pi_1{+}\pi_3, \end{aligned}$$

(π_1,π_3) is a valid pair of first level submaximal partitions.

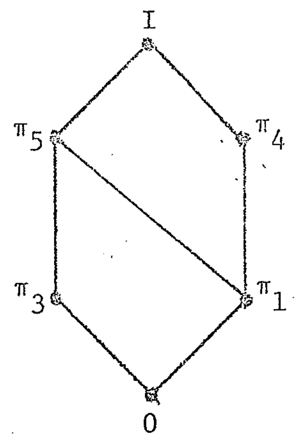
As there are no more levels of partitions, the algorithm ends producing a state assignment using the partitions $\{(\pi_4,\pi_5,\pi_7)(\pi_1,\pi_3)\}$.

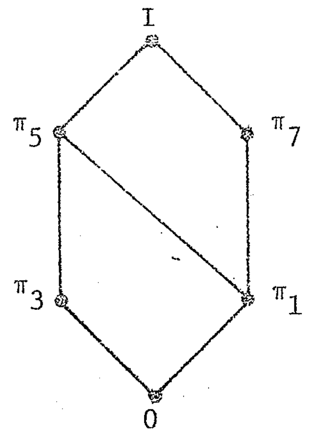
The lattice diagram characterizing this assignment is



Due to the miscalculation, Curtis derives two possible assignments $\{(\pi_4,\pi_5),(\pi_1,\pi_3)\}$ and $\{(\pi_5,\pi_7),(\pi_1,\pi_3)\}$.

The lattice diagrams corresponding to these assignments are:





The assignment $\{(\pi_4,\pi_5,\pi_7),(\pi_1,\pi_3)\}$ is not as economical as either of the above assignments.

Thus, for this example, Curtis' method actually produces an assignment with an extra partition. In the next section we will prove that this problem is not isolated to the above example.

2.6. A Critique of Curtis' Algorithm

In Section 2.5 we indicated a problem with Curtis' algorithm. In this section, this problem will be examined formally and limitations of the algorithm will be presented. The limitations of the algorithm can be itemized as follows:

(1) The method cannot be applied to all machines with partitions;

(2) The grouping of partitions is not straightforward;

(3) Some assignments produced contain superfluous partitions;

(4) All possible assignments for a machine are not produced.

The first problem results from the selection criterion, $k\pi + \mu\pi = s$, used to retain S.P. partitions. For some machines, some or all of the S.P. partitions violate $k\pi + \mu\pi = s$.

Example 2.13

	i_0	i_1	i_2	i_3
1	2	3	4	1
2	1	3	4	1
3	2	1	4	1
4	2	3	1	1

M

The S.P. partitions for M are

$$\pi_1 = \{\overline{1,2,3};\ \overline{4}\}$$
$$\pi_2 = \{\overline{1,3,4};\ \overline{2}\}$$
$$\pi_3 = \{\overline{1,2,4};\ \overline{3}\}$$
$$\pi_4 = \{\overline{1,2};\ \overline{3};\ \overline{4}\}$$
$$\pi_5 = \{\overline{1,3};\ \overline{2};\ \overline{4}\}$$
$$\pi_6 = \{\overline{1,4};\ \overline{2};\ \overline{3}\}$$

Since, $k\pi_i + \mu\pi_i > 2 = [\log_2 4]$, $i = 1,\ldots,6$, none of the partitions can be used to derive an assignment for M. However, the set of partitions $\{\pi_1,\pi_2,\pi_3\}$ provides an assignment for M, since $\pi_1 \cdot \pi_2 \cdot \pi_3 = 0$.

Using this assignment requires three state variables, rather than the minimal two. However, the ability to decompose M into three separate, smaller machines compensates for the use of an extra state variable.

Curtis' algorithm depends upon dividing the S.P. partitions into various groups and then selecting an optimum number of partitions from each group. The use of the k-value of a partition to determine its group causes the second problem with the algorithm.

In formulating Rule 4 of the Algorithm, Curtis notes

".....that no partition can contain a partition whose k is of the same size."

However, a simple counter-example demonstrates that this is not always true.

Example 2.14 For a machine $M = (S, IP, OP, \delta, \lambda)$, where $S = \{1,2,3,4,5,6,7,8,9\}$, assume that $\pi_1 = \{\overline{1,2,3,9};\ \overline{4,5,6};\ \overline{7,8}\}$ and $\pi_2 = \{\overline{1,2,3};\ \overline{4,5,6};\ \overline{7,8};\ \overline{9}\}$ are S.P. partitions.

$k\pi_1 = k\pi_2 = 2$, but π_2 is contained in π_1. As π_1 and π_2 are retained partitions, there is a problem as to which groups π_1 and π_2 belong.

Since, π_1 and π_2 have the same k-value, they should be in the same group. However, $\pi_1 > \pi_2$, which implies that π_1 should be in a group, one level higher than π_2.

Thus, new criteria are required to group the retained partitions of a machine.

In Section 2.5 it was shown that for the example chosen by Curtis, the algorithm produced a decomposition with an extra S.P. partition. Thus an assignment realized from this decomposition would require more state variables than necessary.

Actually, it can be proved that whenever $(\pi_i,\pi_j),(\pi_i,\pi_k)$, and (π_j,π_k) are valid pairs, then (π_i,π_j,π_k) will also be a valid triple.

Theorem 2.7 For a set of maximal partition candidates, the triple (π_i,π_j,π_k) is a valid triple of maximal partitions if

(π_i,π_j), (π_i,π_k), and (π_j,π_k) are valid pairs

and

$$\pi_i \cdot \pi_j = \pi_i \cdot \pi_k = \pi_j \cdot \pi_k = \pi .$$

Proof For (π_i,π_j,π_k) to be a valid triple the following conditions must be satisfied:

$$k\pi_i \cdot \pi_j \cdot \pi_k = k\pi_\ell, \pi_i \cdot \pi_j \cdot \pi_k + k\pi_\ell , \quad \ell = i,j,k \tag{1}$$

$$\left.\begin{aligned} &= k\pi_i \cdot \pi_j, \pi_i \cdot \pi_j \cdot \pi_k + k\pi_i \cdot \pi_j \\ &= k\pi_i \cdot \pi_k, \pi_i \cdot \pi_j \cdot \pi_k + k\pi_i \cdot \pi_k \\ &= k\pi_j \cdot \pi_k, \pi_i \cdot \pi_j \cdot \pi_k + k\pi_j \cdot \pi_k \end{aligned}\right\} \tag{2}$$

(i) $\pi_i \cdot \pi_j \cdot \pi_k = \pi_i \cdot \pi_j \cdot \pi_j \cdot \pi_k$

$= \pi \cdot \pi$

$= \pi$

$\therefore\ k\pi_i \cdot \pi_j \cdot \pi_k = k\pi$

Since (π_i, π_j) is a valid pair, we have

$$k\pi_i \cdot \pi_j = k\pi_i, \pi_i \cdot \pi_j + k\pi_i = k\pi_j, \pi_i \cdot \pi_j + k\pi_j\ .$$

However, this can be replaced by

$$k\pi = k\pi_i, \pi + k\pi_i = k\pi_j, \pi + k\pi_j\ ,$$

which in turn can be replaced by

$$k\pi_i \cdot \pi_j \cdot \pi_k = k\pi_i, \pi_i \cdot \pi_j \cdot \pi_k + k\pi_i = k\pi_j, \pi_i \cdot \pi_j \cdot \pi_k + k\pi_j\ .$$

Similarly,it can be shown that

$$k\pi_i \cdot \pi_j \cdot \pi_k = k\pi_k, \pi_i \cdot \pi_j \cdot \pi_k + k\pi_k.$$

Therefore, $k\pi_i \cdot \pi_j \cdot \pi_k = k\pi_\ell, \pi_i \cdot \pi_j \cdot \pi_k + k\pi_\ell,\quad \ell = i,j,k.$

(ii) $\pi_i \cdot \pi_j = \pi = \pi_i \cdot \pi_j \cdot \pi_k$

Since, $k\pi, \pi = 0$

$$k\pi_i \cdot \pi_j, \pi_i \cdot \pi_j \cdot \pi_k = 0.$$

Thus,

$$k\pi_i \cdot \pi_j, \pi_i \cdot \pi_j \cdot \pi k + k\pi_i \cdot \pi_j = k\pi_i \cdot \pi_j = k\pi$$

and

$$k\pi_i \cdot \pi_j \cdot \pi_k = k\pi_i \cdot \pi_j, \pi_i \cdot \pi_j \cdot \pi_k + k\pi_i \cdot \pi_j\ .$$

Similarly, $k\pi_i \cdot \pi_j \cdot \pi_k = k\pi_i \cdot \pi_k, \pi_i \cdot \pi_j \cdot \pi_k + k\pi_i \cdot \pi_k$

$= k\pi_j \cdot \pi_k, \pi_i \cdot \pi_j \cdot \pi_k + k\pi_j \cdot \pi_k\ .$

Conditions (1) and (2) are satisfied and (π_i, π_j, π_k) is a valid triple of maximal partitions.

Since $\pi_i \cdot \pi_j = \pi_i \cdot \pi_k = \pi_j \cdot \pi_k = \pi_i \cdot \pi_j \cdot \pi_k = \pi$, the triple (π_i, π_j, π_k) realizes the same partition as the pairs (π_i, π_j), (π_i, π_k), and (π_j, π_k), but at the expense of an extra partition.

Sets of S.P. partitions that do not contain extra partitions can be characterized by the definition of nonredundant sets, Definition 2.12. The following theorem proves that valid triples of maximal partition candidates of the type described in the preceding theorem are not nonredundant.

Theorem 2.8 A valid triple of maximal partition candidates,

$T = (\pi_i, \pi_j, \pi_k)$, for which

$\pi_i \circ \pi_j = \pi_i \circ \pi_k = \pi_j \circ \pi_k = \pi$, is not a nonredundant set of S.P. partitions.

Proof

Assume that T is a nonredundant set of maximal partition candidates.

Therefore, for (π_j, π_k)

if $\pi_i \geq \pi_j \cdot \pi_k$, then $\pi_i \geq \pi_j$ or $\pi_i \geq \pi_k$.

Since, $\pi_i \cdot \pi_j \cdot \pi_k = \pi$, $\pi_i \geq \pi = \pi_j \cdot \pi_k$.

Thus, $\pi_i \geq \pi_j$ or $\pi_i \geq \pi_k$.

However, the partitions π_i, π_j, and π_k are distinct,

so $\pi_i > \pi_j$ or $\pi_i > \pi_k$.

Without loss of generality let $\pi_i > \pi_j$.

Therefore, $\pi_i \cdot \pi_j = \pi_j$

$$k\pi_i, \pi_i \cdot \pi_j = k\pi_i, \pi_j > 0$$
$$k\pi_j, \pi_i \cdot \pi_j = k\pi_j, \pi_j = 0$$
$$\therefore \quad k\pi_i, \pi_i \cdot \pi_j > k\pi_j, \pi_i \circ \pi_j .$$

Since $k\pi_i = k\pi_j = k\pi_k$,

$$k\pi_i + k\pi_i, \pi_i \cdot \pi_j > k\pi_j + k\pi_j, \pi_i \cdot \pi_j ,$$

contradicting condition (1) in rule (v).

Therefore, (π_i, π_j) is not a valid pair of maximal partitions, contradicting our assumption that $T = (\pi_i, \pi_j, \pi_k)$ is a nonredundant set of partitions.

The above two theorems prove that Curtis' algorithm will generate assignments with redundant partitions. In addition, it can he shown that the algorithm fails, in a trivial way, to determine all the valid nonredundant decompositions for a machine.

Example 2.14

	i_0	i_1	i_2
1	2	2	3
2	3	4	5
3	6	5	4
4	7	2	8
5	4	9	7
6	9	4	10
7	8	1	11
8	2	11	12
9	7	11	1
10	1	8	2
11	12	9	6
12	6	6	9

M

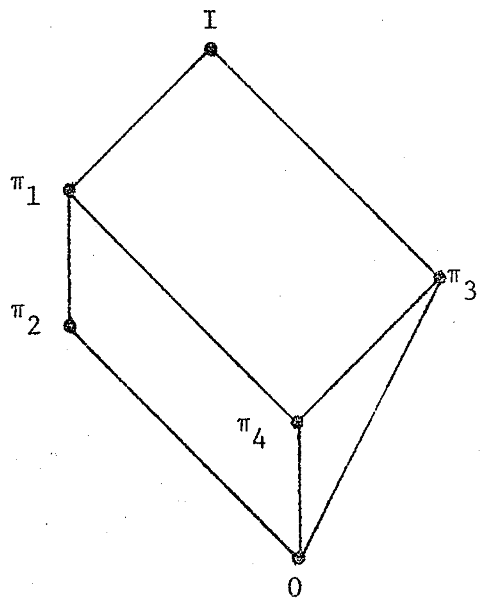
The nontrivial S.P. partitions for M are:

$\pi_1 = \{\overline{1,3,4,8,9,12};\ \overline{2,5,6,7,10,11}\}$

$\pi_2 = \{\overline{1,4,12};\ \overline{3,8,9};\ \overline{2,6,7};\ \overline{5,10,11}\}$

$\pi_3 = \{\overline{1,5,6,8};\ \overline{2,4,9,11};\ \overline{3,7,10,12}\}$

$\pi_4 = \{\overline{1,8};\ \overline{2,11};\ \overline{3,12};\ \overline{4,9};\ \overline{5,6};\ \overline{7,10}\}$



S.P. lattice for M

retained partitions	k
I	0
π_1	1
π_2	2
π_3	2
π_4	3
0	4

π_1 is the only maximal partition candidate. The first level submaximal partition candidates are π_2 and π_3.

However, π_2 and π_3 do not form a first level submaximal pair since the l.u.b. of π_2 and π_3 is not a maximal partition candidate (rule (vii)).

That is, $\pi_2 + \pi_3 = I$ and I is not a maximal partition candidate.

Thus, the decomposition (π_1, π_2, π_3) is rejected, even though it is a nonredundant set of S.P. partitions. Actually, (π_1, π_2, π_3) is the most economical decomposition that can be obtained for machine M.

2.7 Summary and Remarks

In this chapter we have formalized the definition of a sequential machine and demonstrated how a machine may be realized by a sequential circuit. The sequential machine concept is useful as it allows us to examine the behaviour of a sequential circuit, independently of the physical device.

Hartmanis and Stearns' algebraic structure theory of sequential machines is an examination of the structural properties of machines. The properties of interest to this thesis relate to machine decompositions, and, in particular, to loop-free decompositions. Consequently, we have presented S.P. partition theory in detail.

An S.P. partition on a machine performs a subfunction of the machine's behaviour. An S.P. partition can easily be realized as a

π-image, or submachine, of the machine. The submachines, corresponding to S.P. partitions, can be interconnected in either a serial or parallel fashion. Using these two basic connections, a large sequential machine can be decomposed as a network of interconnected submachines.

The problem of determining which S.P. partitions to use to decompose a machine has been examined by Hartmanis and Stearns. They have proved that decomposition without redundant machines can be obtained by selecting a nonredundant set of the S.P. partitions. However, the only method they provide for deriving nonredundant sets is inspection of the S.P. lattice. For machines with a small number of S.P. partitions, this is a satisfactory method. But, as the number of S.P. partitions increase, inspection becomes a tedious and error-prone task. Also, the inspection method does not lend itself to a computer implementation.

Curtis' attempt at developing a method for decomposing sequential machines has been presented. A critical examination has revealed that problems with the algorithm restrict its usefulness for obtaining decompositions. The difficulties in the algorithm arise from the following sources:

(1) An attempt to obtain absolutely minimal decompositions.

(2) The imposition of an artificial structure on the S.P. partition lattice.

In trying to derive state assignments with the minimal number of variables, S.P. partitions which violate the condition $k\pi + \mu\pi = s$ are rejected by Curtis' algorithm. As demonstrated in the previous section, this severely limits the generality of the algorithm.

The condition, $k\pi_1 = k\pi_2, \pi_1 + k\pi_2$, where $\pi_2 > \pi_1$, of Theorem 2.6 is intended to ensure the minimal number of state variables when realizing two S.P. partitions together. However, this is not a sufficient condition when expanded to include sets of three or more partitions. Consequently, exclusive reliance on this condition results in increasing the number of state variables, rather than ensuring the minimal number.

Grouping S.P. partitions based on their k-value is an artificial ordering of the S.P. partition lattice. As demonstrated in Section 2.6, there are partitions which cannot be classified properly using this ordering. Similarly, the rule that pairs of first level submaximal partitions are retained only if their l.u.b. is a maximal partition, assumes a symmetrical lattice. However, the unsymmetrical structure of most lattices limits the application of this rule.

It is possible to make modifications to Curtis' algorithm to handle some of the above problems. However, it is felt that this would further complicate an already unwieldy algorithm.

In the following chapter, a detailed algorithm for decomposing a machine with S.P. partitions is developed. The algorithm provides a systematic method for generating nonredundant sets of S.P. partitions. As a result, the algorithm can be applied to machines with a large number of S.P. partitions and is easily automated. In addition, the problems associated with Curtis' algorithm do not occur.

Chapter 3 Nonredundant Sets of Substitution Property Partitions

3.1 Introduction

Hartmanis and Stearns [31] have proved that a nonredundant set of substitution property (S.P.) partitions, $T = \{\pi_1,\ldots,\pi_n\}$ for a machine M such that $\prod_{\pi_i \in T} \pi_i = 0$, provides a decomposition of M that does not contain any redundant submachines. The only method given by Hartmanis and Stearns for constructing a nonredundant set of S.P. partitions is "inspection" of the S.P. lattice. This is an adequate method for a logic designer dealing with a small S.P. lattice. However, as the number of S.P. partitions in a lattice increases, an algorithmic method becomes necessary.

In the following sections we derive various properties of nonredundant sets. These properties are then used to derive an algorithm for obtaining the nonredundant sets of S.P. partitions of a machine. The application of the method to sequential machine decomposition is demonstrated with examples.

3.2 Properties of Nonredundant Sets of S.P. Partitions

The definition Hartmanis and Stearns give for nonredundancy (Definition 2.12) is more general than is required. Testing for non-redundancy using their definition entails some unnecessary computations. Consider the case where $\Pi\{\pi_j | \pi_j \in T'\} \leq \pi_k$ and $\pi_k \in T'$. Hartmanis and Stearns' definition requires that tests be performed to determine whether $\pi_k \geq \pi_i$, for some $\pi_i \in T'$. Clearly, this test is not necessary since $\pi_k \in T'$.

The following theorem allows us to restate the definition of the nonredundant sets more rigorously.

Theorem 3.1 A set of partitions,

$$T = \{\pi_i\} ,$$

defined on S for machine $M = (S,IP,OP,\delta,\lambda)$ is nonredundant if and only if the π_i are distinct and for all $T' \subseteq T$, $\pi_k \in T$ and $\pi_k \notin T'$, $\Pi\{\pi_j | \pi_j \in T'\} < \pi_k$ implies $\pi_k > \pi_i$, for some $\pi_i \in T'$.

Proof (a) Assume $T = \{\pi_i\}$ is nonredundant. Therefore, for all $T' \subseteq T$, $\pi_k \in T$ and $\pi_k \notin T'$ $\Pi\{\pi_j | \pi_j \in T'\} \leq \pi_k$ implies $\pi_k \geq \pi_i$, for some $\pi_i \in T'$.

Since $\pi_k \notin T'$ and the $\{\pi_i\}$ are distinct

$$\pi_k \neq \pi_i \text{ for some } \pi_i \in T'.$$

Therefore, $\Pi\{\pi_j | \pi_j \in T'\} \leq \pi_k$ implies $\pi_k > \pi_i$, for some $\pi_i \in T'$.

Assume $\Pi\{\pi_j | \pi_j \in T'\} = \pi_k$

$\therefore$ $\pi_i \cdot \Pi\{\pi_j | \pi_j \in T'\} = \pi_i \cdot \pi_k$

$\Pi\{\pi_j | \pi_j \in T'\} = \pi_i$

$\pi_k = \pi_i \cdot \pi_k$

$\therefore$ $\pi_k \leq \pi_i$, which contradicts $\pi_k > \pi_i$.

Therefore, for all $T' \subseteq T$, $\pi_k \in T$ and $\pi_k \notin T'$, $\Pi\{\pi_j | \pi_j \in T'\} < \pi_k$ implies $\pi_k > \pi_i$, for some $\pi_i \in T'$.

(b) Assume the $\{\pi_i\}$ are distinct and for all $T' \subseteq T$, $\pi_k \in T$, and $\pi_k \notin T'$, $\Pi\{\pi_j | \pi_j \in T'\} < \pi_k$ implies $\pi_k > \pi_i$ for some $\pi_i \in T'$. Trivially, for all $T' \subseteq T$, $\pi_k \in T$, and $\pi_k \notin T'$ $\Pi\{\pi_j | \pi_j \in T'\} \leq \pi_k$ implies $\pi_k \geq \pi_i$ for some $\pi_i \in T'$.

We must show, that for all $T' \subseteq T$, $\pi_k \in T$ and $\Pi_k \in T'$ $\Pi\{\pi_j | \pi_j \in T'\} \leq \pi_k$ implies $\pi_k \geq \pi_i$, for some $\pi_i \in T'$.

This can be easily shown by observing that $\pi_k = \pi_i$ for some $\pi_i \in T'$ (i.e. π_k).

Therefore, the set $T = \{\pi_i\}$ is nonredundant.

Definition 3.1 A set of partitions $T = \{\pi_1, \ldots, \pi_n\}$ which is not nonredundant is said to be redundant.

Hartmanis' and Stearns' definition of nonredundancy will be used to prove the following theorem, which establishes a sufficient condition for a set to be redundant. In future, however, the definition of nonredundancy established in Theorem 3.1 will be used in proving theorems and testing for nonredundancy.

Theorem 3.2 A set of partitions,

$$T = \{\pi_i\},$$

defined on S for machine $M = (S, IP, OP, \delta, \lambda)$ is redundant if for one $T' \subseteq T$, $\pi_k \in T$, and $\pi_k \notin T'$,

$$\Pi\{\pi_j | \pi_j \in T'\} = \pi_k .$$

Proof If the set $T = \{\pi_i\}$ is to be nonredundant, then $\pi_k \geq \pi_i$, for some $\pi_i \in T'$.
Since $T = \{\pi_i\}$ is a set of distinct partitions, we must have $\pi_k > \pi_i$, for some $\pi_i \in T'$.

$$\pi_i \cdot \Pi\{\pi_j | \pi_j \in T'\} = \pi_i \cdot \pi_k$$

$$\Pi\{\pi_j | \pi_j \in T'\} = \pi_i \cdot \pi_k$$

$$\therefore \pi_k = \pi_i \cdot \pi_k$$

Therefore, $\pi_i > \pi_k$, which contradicts $\pi_k > \pi_i$. Thus, the set $T = \{\pi_i\}$ is redundant.

As will be shown later, nonredundant sets of partitions can be derived from pairs of nonredundant partitions. Initially, however, we need a lemma establishing conditions for a pair of partitions to be nonredundant.

Lemma 3.1 A set of two partitions $T = \{\pi_i, \pi_j\}$, for machine M is nonredundant if and only if the partitions π_i and π_j are distinct.

Proof The proof is obvious.

An essential result for our algorithm is proven next.

Theorem 3.3 For a set of partitions,

$$T = \{\pi_i\} ,$$

which is nonredundant, any subset $T'' \subseteq T$ is also a nonredundant set of partitions.

Proof   Let T' be a subset of T'' such that

$$T' \subseteq T'' \subseteq T .$$

Since T is nonredundant for $\pi_k \in T$ and $\pi_k \notin T'$,

$\Pi\{\pi_j | \pi_j \in T'\} < \pi_k$ implies $\pi_k > \pi_i$ for some $\pi_i \in T'$. (Theorem 3.1).

Since $T'' \subseteq T$, for $T' \subseteq T''$, $\pi_k \in T''$ ($\pi_k \in T$) and $\pi_k \notin T'$ then

$\Pi\{\pi_j | \pi_j \in T'\} < \pi_k$ implies $\pi_k > \pi_i$ for some $\pi_i \in T'$.

Therefore the set T'' is nonredundant.

The following theorem provides the basis for an algorithmic method to generate nonredundant sets. Essentially, the theorem proves that it is possible to obtain nonredundant sets by combining non-redundant sets which satisfy certain conditions.

Before the theorem is presented some notation is necessary.

Notation   For a set $T = \{\pi_i, \ldots, \pi_n\}$, we will use T_i to denote a subset of T such that,

$$T_i = \{\pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_n\} .$$

Theorem 3.4   A set of distinct partitions,

$$T = \{\pi_1, \ldots, \pi_n\} ,$$

is nonredundant if and only if the subsets T_i, for $i = 1, \ldots, n$, are nonredundant and for each T_i,

$$\Pi\{\pi_j | \pi_j \in T_i\} < \pi_i \text{ implies } \pi_i > \pi_j \text{ for some } \pi_j \in T_i .$$

Proof   If T is nonredundant, then obviously each subset T_i, for $i = 1, \ldots, n$, is nonredundant and

$$\Pi\{\pi_j | \pi_j \in T_i\} < \pi_i \text{ implies } \pi_i > \pi_j, \text{ for some } \pi_j \in T_i .$$

If the T_i, $i = 1, \ldots, n$ are nonredundant, then by Theorem 3.3, all the subsets of the T_i are nonredundant. Thus, for all subsets $A \subseteq T$, which consist of $n - 2$ partitions or less, $\pi_a \in T$ and $\pi_a \notin A$, then

$\Pi\{\pi_j | \pi_j \in A\} < \pi_a$ implies $\pi_a > \pi_j$ for some $\pi_j \in A$.

Thus, if we have n nonredundant sets of $n - 1$ partitions $T_1, \ldots T_n$, such that for $i=1,\ldots,n$,

$\Pi\{\pi_j | \pi_j \in T_i\} < \pi_i$ implies $\pi_i > \pi_j$ for $\pi_j \in T_i$,

then the set $T = T_1 \cup T_2 \cup \ldots \cup T_n$ is nonredundant.

3.3. A Method to Generate Nonredundant Sets

Since, from Lemma 3.1, all sets of two distinct partitions are nonredundant, the nonredundant sets of three partitions can be obtained by applying Theorem 3.4. Similarly, from the nonredundant sets of three partitions, we can derive the nonredundant sets of four partitions, and so on. A basic algorithmic method for generating nonredundant sets is now presented.

Algorithm

(1) For a set of partitions P, first obtain all the products of pairs of partitions.

(2) Each set of three partitions, T, is tested for nonredundancy by testing whether

$\Pi\{\pi_j | \pi_j \in T_i\} < \pi_i$ implies $\pi_i > \pi_j$ for some $\pi_j \in T_i$, for $i = 1,2,3$.

There are four possible situations that can occur in performing the above test:

(i) $\Pi\{\pi_j | \pi_j \in T_i\} = \pi_k$ for some $\pi_k \in T_i$; obviously, if $\pi_i > \Pi\{\pi_j | \pi_j \in T_i\} = \pi_k$, then $\pi_i > \pi_\ell$ for some $\pi_\ell \in T_i$ (i.e., $\pi_\ell = \pi_k$).

(ii) If $\Pi\{\pi_j | \pi_j \in T_i\} > \pi_i$, then the above test need not be performed.

(iii) If $\Pi\{\pi_j | \pi_j \in T_i\} = \pi_i$, then by Theorem 3.2, the set T is redundant.

(iv) Thus, we need to test whether $\pi_i > \pi_j$ for some $\pi_j \in T_i$ only if $\Pi\{\pi_j | \pi_j \in T_i\} < \pi_i$.

(3) The sets of four partitions, T, are tested for nonredundancy by:

(i) Ensuring that all subsets of three partitions are nonredundant.

(ii) Determining whether
$\Pi\{\pi_j | \pi_j \in T_i\} < \pi_i$ implies $\pi_i > \pi_j$ for some $\pi_j \in T_i$, for $i = 1,2,3,4$.
(The four possible situations mentioned in Step (2) apply equally in this case).

(4) The sets of $5,6,\ldots,n$ partitions are tested for nonredundancy in a manner similar to Step (3).

(5) When no further nonredundant sets can be generated, the algorithm ends.

The algorithm given above is applied to the machine used by Hartmanis and Stearns ([31] pp. 102-103) to demonstrate nonredundancy.

<u>Example 3.1</u>

	i_1	i_2	i_3	i_4	i_5
1	2	1	5	8	3
2	1	2	6	7	4
3	4	3	6	6	1
4	3	4	5	5	2
5	5	6	3	4	5
6	6	5	4	3	8
7	7	8	4	2	5
8	8	7	3	1	6

A

The S.P. partitions for state machine A are:

$$0 = \pi_1 = \{\overline{1}; \overline{2}; \overline{3}; \overline{4}; \overline{5}; \overline{6}; \overline{7}; \overline{8}\}$$

$$\pi_2 = \{\overline{1,4}; \overline{2,3}; \overline{5,8}; \overline{6,7}\}$$

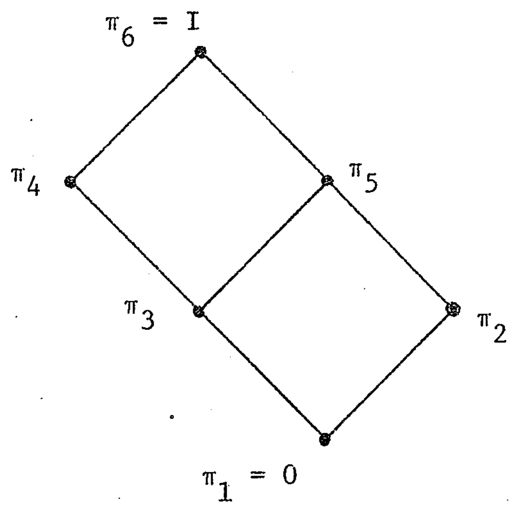
$$\pi_3 = \{\overline{1,2}; \overline{3,4}; \overline{5,6}; \overline{7,8}\}$$

$$\pi_4 = \{\overline{1,2,7,8}; \overline{3,4,5,6}\}$$

$$\pi_5 = \{\overline{1,2,3,4}; \overline{5,6,7,8}\}$$

$$I = \pi_6 = \{\overline{1,2,3,4,5,6,7,8}\}$$

The S.P. lattice for A follows.



All sets of two partitions are nonredundant. Test whether $\{\pi_1, \pi_2, \pi_3\}$ is nonredundant

$$\pi_1 \cdot \pi_2 = \pi_1$$

By step 2(ii), if $\pi_3 > \pi_1 \cdot \pi_2 = \pi_1$, then $\pi_3 > \pi_k$ for some $\pi_k \in \{\pi_1, \pi_2\}$.

Similarly, for $\pi_1 \cdot \pi_3 = \pi_1$.

However, for $\{\pi_2, \pi_3\}$, we have that

$$\pi_2 \cdot \pi_3 = \pi_1 \ .$$

Therefore, by Theorem 3.2, the set $\{\pi_1, \pi_2, \pi_3\}$ is redundant.

Test whether $\{\pi_2, \pi_3, \pi_5\}$ is nonredundant

$$\pi_2 \cdot \pi_3 = \pi_1 < \pi_5 \quad \text{and} \quad \pi_5 > \pi_3 \quad \text{and} \quad \pi_5 > \pi_2 \ .$$

$$\pi_2 \cdot \pi_5 = \pi_2 \quad \text{and} \quad \pi_3 \cdot \pi_5 = \pi_3 .$$

Therefore, the set $\{\pi_2, \pi_3, \pi_5\}$ is nonredundant and is retained in order to generate the nonredundant sets containing four partitions. The determination of all the nonredundant sets for machine A is illustrated below.

$\pi_1\pi_2 = \pi_1$
$\pi_1\pi_3 = \pi_1$
$\pi_1\pi_4 = \pi_1$
$\pi_1\pi_5 = \pi_1$
$\pi_1\pi_6 = \pi_1$

$\pi_2\pi_3 = \pi_1$
$\pi_2\pi_4 = \pi_1$
$\pi_2\pi_5 = \pi_2$
$\pi_2\pi_6 = \pi_2$

$\pi_3\pi_4 = \pi_3$
$\pi_3\pi_5 = \pi_3$
$\pi_3\pi_6 = \pi_3$

$\pi_4\pi_5 = \pi_3$
$\pi_4\pi_6 = \pi_4$

$\pi_5\pi_6 = \pi_5$

X $\pi_1\pi_2\pi_3$
X $\pi_1\pi_2\pi_4$
$\pi_1\pi_2\pi_5 = \pi_1$
$\pi_1\pi_2\pi_6 = \pi_1$

$\pi_1\pi_3\pi_4 = \pi_1$
$\pi_1\pi_3\pi_5 = \pi_1$
$\pi_1\pi_3\pi_6 = \pi_1$

$\pi_1\pi_4\pi_5 = \pi_1$
$\pi_1\pi_4\pi_6 = \pi_1$

$\pi_1\pi_5\pi_6 = \pi_1$

X $\pi_2\pi_3\pi_4$
$\pi_2\pi_3\pi_5 = \pi_1$
$\pi_2\pi_3\pi_6 = \pi_1$

$\pi_2\pi_4\pi_5 = \pi_1$
$\pi_2\pi_4\pi_6 = \pi_1$

$\pi_2\pi_5\pi_6 = \pi_2$

X $\pi_3\pi_4\pi_5$
$\pi_3\pi_4\pi_6 = \pi_3$
$\pi_3\pi_5\pi_6 = \pi_3$
$\pi_4\pi_5\pi_6 = \pi_3$

$\pi_1\pi_2\pi_5\pi_6 = \pi_1$

X $\pi_1\pi_3\pi_4\pi_5$
$\pi_1\pi_3\pi_4\pi_6 = \pi_1$

$\pi_1\pi_3\pi_5\pi_6 = \pi_1$

$\pi_1\pi_4\pi_5\pi_6 = \pi_1$

$\pi_2\pi_3\pi_5\pi_6 = \pi_1$

$\pi_2\pi_4\pi_5\pi_6 = \pi_1$

Sets of partitions without an "X" beside them are nonredundant. As can be seen there are a great number of nonredundant sets of S.P. partitions. Thus, to pick an assignment for machine A based on these nonredundant sets would be difficult.

Fortunately, many of these sets can be discarded. For example, the set $\{\pi_2,\pi_3,\pi_6\}$ does not give us more information than the set $\{\pi_2,\pi_3\}$. In fact, all the sets which contain either $\pi_1 = 0$ or $\pi_6 = I$ are superfluous and need not be considered. The theorems given in the following section establish this.

3.4 Further Properties of Nonredundant Sets

In this section some properties of nonredundant sets are developed which further simplify the generation of nonredundant sets. The theorem given below proves that a partition π can be added to a nonredundant set, T, and the augmented set $T \cup \{\pi\}$ be nonredundant, if π is greater than all the partitions in T.

Theorem 3.5 Let $P = \{\pi_1,\ldots,\pi_n\}$ be a set of distinct S.P. partitions. If for a nonredundant subset, $T_0 \subset P$, there exists an S.P. partition $\pi_a, \pi_a \in P$ and $\pi_a \notin T_0$, such that $\pi_a > \pi_i$, $\forall \pi_i \in T_0$, then the subset $T = T_0 \cup \{\pi_a\}$ is also nonredundant.

Proof We must show that for any subset T', $T' \subseteq T$, that if $\pi_k \in T$ and $\pi_k \notin T'$, then

$$\Pi\{\pi_j | \pi_j \notin T'\} < \pi_k \text{ implies } \pi_k > \pi_i, \text{ for some } \pi_i \in T'.$$

(1) For $T' \subseteq T$ and $\pi_a \notin T'$ (i.e. $T' \subseteq T_0$)

(a) For $\pi_k \neq \pi_a$, then $T' \cup \{\pi_k\} \subseteq T_0$. Since T_0 is nonredundant, the condition holds

(b) For $\pi_k = \pi_a$, since $\pi_k = \pi_a > \pi_j$, $\forall \pi \in T_0$, $\pi_k = \Pi\,\{\pi_j | \pi_j \in T'\}$ and $\pi_k > \pi_j$ for some $\pi_j \in T'$.

(2) For $T' \subseteq T$ and $\pi_a \in T'$.

Assume

$$\Pi\{\pi_j | \pi_j \in T'\} < \pi_k \text{ for } \pi_k \in T \text{ and } \pi_k \notin T' .$$

$\Pi\{\pi_j | \pi_j \in T'\} = \Pi\,\{\pi_j | \pi_j \in (T'-\{\pi_a\})\}$, since $\pi_a > \pi_i$, $\forall \pi_i \in T_0$.

Since $(T' - \{\pi_a\}) \subseteq T_0$ and T_0 is nonredundant, there exists a $\pi_i \in (T' - \{\pi_a\}) \ni \cdot\, \pi_k > \pi_i$.

Therefore, there exists a $\pi_i \in T' \ni \cdot\, \pi_k > \pi_i$.

Thus, the definition of nonredundancy holds for all cases, and the set $T_0 \cup \{\pi_a\}$ is nonredundant.

Corollary 3.5.1 For any nonredundant set of partitions, $T = \{\pi_1,\ldots,\pi_n\}$, where $I \notin T$, the set $T_I = T \cup \{I\}$ is also nonredundant.

Theorem 3.5 is used to generate nonredundant sets in the next example.

Example 3.2

	i_0	i_1
1	3	7
2	4	8
3	1	6
4	2	5
5	2	4
6	1	3
7	4	4
8	3	5

M

S.P. partitions for m

$\pi_1 = \{\overline{1,2}; \overline{3,4}; \overline{5;6}; \overline{7,8}\}$

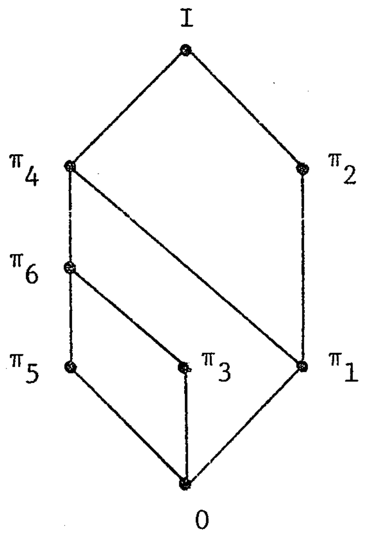
$\pi_2 = \{\overline{1,2,3,4}; \overline{5,6,7,8}\}$

$\pi_3 = \{\overline{1}; \overline{2}; \overline{3}; \overline{4,5}; \overline{6}; \overline{7}; \overline{8}\}$

$\pi_4 = \{\overline{1,2}; \overline{3,4,5,6}; \overline{7,8}\}$

$\pi_5 = \{\overline{1}, \overline{2}, \overline{3,6}; \overline{4}; \overline{5}; \overline{7}; \overline{8}\}$

$\pi_6 = \{\overline{1}; \overline{2}; \overline{3,6}; \overline{4,5}; \overline{7}; \overline{8}\}$



S.P. lattice

The set $T_0 = \{\pi_3, \pi_5, \pi_6\}$ is nonredundant.

Since $\pi_4 > \pi_i$, $\forall \pi_i \in T_0$, the set

$T_0 \cup \{\pi_4\} = \{\pi_3, \pi_4, \pi_5, \pi_6\}$ is also nonredundant.

In this case, the standard tests for nonredundancy need not be performed as Theorem 3.5 applies.

In addition, the tests for nonredundancy need not be performed when the partition to be added to a nonredundant set is less than the product of all the partitions in the nonredundant set.

Theorem 3.6 Let $P = \{\pi_1, \ldots, \pi_n\}$ be a set of distinct S.P. partitions. If for a nonredundant set $T_0 \subset P$, there exists an S.P. partition π_a, $\pi_a \in P$ and $\pi_a \notin T_0$ such that

$$\Pi\{\pi_j | \pi_j \in T_0\} > \pi_a ,$$

then the subset $T = T_0 \cup \{\pi_a\}$ is also nonredundant.

Proof (1) For all subsets of T, $T' \subset T$, such that $\pi_a \notin T'$ (i.e. $T' \subset T_0$)

Assume

$$\Pi\{\pi_j | \pi_j \in T'\} < \pi_k \quad \text{for} \quad \pi_k \in T \text{ and } \pi_k \notin T'.$$

(Obviously $\pi_k \neq \pi_a$ since $\Pi\{\pi_j | \pi_j \in T_0\} > \pi_a$)

Since T_0 is nonredundant, $\pi_k > \pi_i$ for some $\pi_i \in T'$.

(2) For $T' \subset T$ such that $\pi_a \in T'$

Assume

$$\Pi\{\pi_j | \pi_j \in T'\} < \pi_k$$

Since $\pi_a \in T'$ and $\Pi\{\pi_j | \pi_j \in T_0\} > \pi_a$

$$\pi_a = \Pi\{\pi_j | \pi_j \in T'\} < \pi_k$$

$\therefore$ $\pi_k > \pi_i$, for some $\pi_i \in T'$. i.e., π_a.

Thus, the set $T = T_0 \cup \pi_a$ is nonredundant.

Corollary 3.6.1 For a nonredundant set

$$T = \{\pi_1, \ldots, \pi_n\},$$

such that $\Pi\{\pi_j | \pi_j \in T\} \neq 0$, the set

$$T \cup \{0\} \text{ is also nonredundant.}$$

Theorem 3.6 is applied in the example given below.

Example 3.3

	i_0	i_1
1	3	7
2	4	8
3	1	6
4	2	5
5	2	4
6	1	3
7	4	4
8	3	3

M

S.P. partitions for machine M

$\pi_1 = \{\overline{1,2};\ \overline{3,4};\ \overline{5,6};\ \overline{7,8}\}$

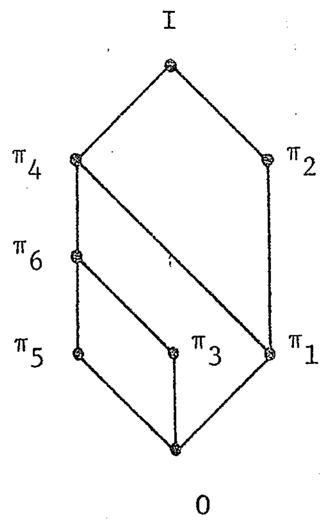
$\pi_2 = \{\overline{1,2,3,4};\ \overline{5,6,7,8};$

$\pi_3 = \{\overline{1};\ \overline{2};\ \overline{3};\ \overline{4,5};\ \overline{6};\ \overline{7};\ \overline{8}\}$

$\pi_4 = \{\overline{1,2};\ \overline{3,4,5,6};\ \overline{7,8}\}$

$\pi_5 = \{\overline{1};\ \overline{2};\ \overline{3,6};\ \overline{4};\ \overline{5};\ \overline{7};\ \overline{8}\}$

$\pi_6 = \{\overline{1};\ \overline{2};\ \overline{3,6};\ \overline{4,5};\ \overline{7};\ \overline{8}\}$



S.P. lattice

The set $T_0 = \{\pi_4, \pi_6\}$ is nonredundant.
Since $\pi_3 < \Pi\{\pi_j | \pi_j \in T_0\}$, the set

$$T_0 \cup \{\pi_3\} = \{\pi_3, \pi_4, \pi_6\} \text{ is also nonredundant.}$$

Obviously it is not necessary to consider the nonredundant pairs containing the trivial partitions, 0 and I, when deriving the nonredundant sets for a machine. Removing the partitions 0 and I, our new derivation of the nonredundant sets for machine A, Example 3.1, follows.

Example 3.4

$\pi_2\pi_3 = \pi_1$ X $\pi_2\pi_3\pi_4 = \pi_1$

$\pi_2\pi_4 = \pi_1$ $\pi_2\pi_3\pi_5 = \pi_1$

$\pi_2\pi_5 = \pi_2$

$\pi_2\pi_4\pi_5 = \pi_1$

$\pi_3\pi_4 = \pi_3$

$\pi_3\pi_5 = \pi_3$ X $\pi_3\pi_4\pi_5 = \pi_3$

$\pi_4\pi_5 = \pi_3$

Adding $0(\pi_1)$ to those sets whose product is not 0, we obtain:

$\{\pi_1,\pi_2,\pi_5\}$, $\{\pi_1,\pi_3,\pi_4\}$, $\{\pi_1,\pi_3,\pi_5\}$, $\{\pi_1,\pi_4,\pi_5\}$, $\{\pi_1,\pi_2\}$, $\{\pi_1,\pi_3\}$, $\{\pi_1,\pi_4\}$, and $\{\pi_1,\pi_5\}$.

Thus, the determination of all the decompositions of machine A has been considerably simplified.

The following theorem permits further simplification of the algorithm.

Theorem 3.7 Let $T = \{\pi_1,\ldots,\pi_n\}$ be a set of n distinct partitions such that the subsets T_i, $i=1,\ldots,n$, are nonredundant. If

$$\Pi\{\pi_j | \pi_j \in T_i\} = \pi, \quad \text{for} \quad i=1,\ldots,n,$$

then the set T is redundant.

Proof Assume that $T = \bigcup_{i=1}^{n} T_i = \{\pi_1,\ldots,\pi_n\}$ is nonredundant.

Since $\Pi\{\pi_j | \pi_j \in T_i\} = \pi$, for $i=1,\ldots,n$

then

$$\pi_i \geq \pi \quad \text{for} \quad i=1,\ldots,n$$

Assume $\pi_i = \pi$

$$\Pi\{\pi_j | \pi_j \in T_i\} = \pi = \pi_i$$

Therefore, by Theorem 3.2, the set $T = \{\pi_1,\ldots,\pi_n\}$ is redundant.

$\therefore$ $\pi_i > \pi$, for $i=1,\ldots,n$.

For $\pi_1 > \pi = \{\pi_j | \pi_j \in T_1\}$, then $\pi_1 > \pi_j$ for some $\pi_j \in T_1$. Renaming the partitions $\pi_2,\ldots,\pi_n$ such that $\pi_1 > \pi_2$. Since $\pi_2 > \pi$, we must also have $\pi_2 > \pi_j$ for some $\pi_j \in T_2$. The partitions $\pi_3,\ldots,\pi_n$ in T_2 can be renamed so that $\pi_2 > \pi_3$.

This process can be continued for all $\pi_i, i=1,\ldots,n$. At no point can there be two partitions, π_ℓ and π_m, such that

$$\pi_\ell > \pi_m, \text{ where } \ell > m$$

since, for $m < \ell$, it has been established that $\pi_m > \pi_{m+1}$ for $\pi_{m+1} \in T_m$.

Thus,

$$\pi_m > \pi_{m+1} > \ldots > \pi_{m+k} = \pi_\ell, \text{ if } m < \ell$$

Therefore, we have that $\pi_m > \pi_\ell$ and $\pi_\ell > \pi_m$, which is contradictory to our assumption that the π_i are distinct.

Thus, for all π_i

$$\pi_i > \pi_{i+1} \text{ for } \pi_{i+1} \in T_i .$$

However, for $i = n$, we cannot have

$$\pi_n > \pi_{n+1} .$$

Thus, at this point $\pi_n > \pi_m$, where $m < n$. As shown above, when this occurs, our assumption is contradicted.

Thus, the set of partitions

$$T = \{\pi_1,\ldots,\pi_n\} \text{ is not nonredundant.}$$

Note that Theorem 3.7 is an extension of Theorem 2.8 to the case where $n \geq 3$ and the assumption of valid $n\overline{\text{tuples}}$ is not made. The usefulness of Theorem 3.7 is demonstrated in the following example.

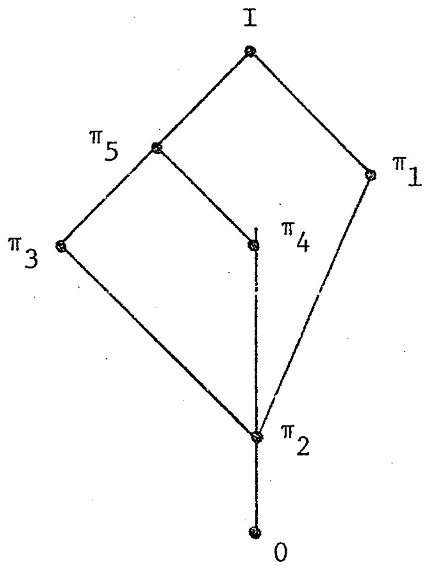
Example 3.5

	i_0	i_1
1	8	2
2	6	1
3	7	4
4	5	3
5	1	3
6	3	4
7	2	1
8	4	2

C

The S.P. partitions are

$\pi_1 = \{\overline{1,2,3,4};\ \overline{5,6,7,8}\}$
$\pi_2 = \{\overline{1,4};\ \overline{2,3};\ \overline{5,8};\ \overline{6,7}\}$
$\pi_3 = \{\overline{1,4,5,8};\ \overline{2,3};\ \overline{6,7}\}$
$\pi_4 = \{\overline{1,4};\overline{2,3,6,7};\overline{5,8}\}$
$\pi_5 = \{\overline{1,4,5,8};\ \overline{2,3,6,7}\}$



S.P. lattice for C

$\pi_1\pi_2 = \pi_2$	X	$\pi_1\pi_2\pi_3$
$\pi_1\pi_3 = \pi_2$	X	$\pi_1\pi_2\pi_4$
$\pi_1\pi_4 = \pi_2$	X	$\pi_1\pi_2\pi_5$
$\pi_1\pi_5 = \pi_2$		
	X	$\pi_1\pi_3\pi_4$
$\pi_2\pi_3 = \pi_2$	X	$\pi_1\pi_3\pi_5$
$\pi_2\pi_4 = \pi_2$		
$\pi_2\pi_5 = \pi_2$	X	$\pi_1\pi_4\pi_5$
$\pi_3\pi_4 = \pi_2$	X	$\pi_2\pi_3\pi_4$
$\pi_3\pi_5 = \pi_3$	✓	$\pi_2\pi_3\pi_5 = \pi_2$
$\pi_4\pi_5 = \pi_4$	✓	$\pi_2\pi_4\pi_5 = \pi_2$
	✓	$\pi_3\pi_4\pi_5 = \pi_2$

For $T = \{\pi_1,\pi_3,\pi_4\}$ since

$\Pi\{\pi_j | \pi_j \in T_i\} = \pi_2$ for $i = 1,3,4$, the set T is redundant. Similarly, we can show that $\{\pi_1,\pi_2,\pi_3\}$, $\{\pi_1,\pi_2,\pi_4\}$, $\{\pi_1,\pi_2,\pi_5\}$, and $\{\pi_2,\pi_3,\pi_4\}$ are redundant.

3.5 Discussion and Further Examples

In this section the universality of the algorithm developed in Section 3.3 is examined. Curtis' algorithm was severely restricted, as partitions for which $k\pi + \mu\pi > s$ were not considered. Our algorithm, however, includes all S.P. partitions when deriving loop-free decompositions and does not make this restriction.

Retaining all the partitions produces a greater number of non-redundant sets. However, as demonstrated in the example below, it is sometimes necessary to retain all the partitions in order to produce a decomposition.

Example 3.6

$$\begin{array}{c|c|c|c|c|} & i_0 & i_1 & i_2 & i_3 \\ \hline 1 & 2 & 3 & 4 & 1 \\ \hline 2 & 1 & 3 & 4 & 1 \\ \hline 3 & 2 & 1 & 4 & 1 \\ \hline 4 & 2 & 3 & 1 & 1 \\ \hline \end{array}$$

M

The S.P. partitions and lattice for M are:

$\pi_1 = \{\overline{1,2,3};\ \overline{4}\}$

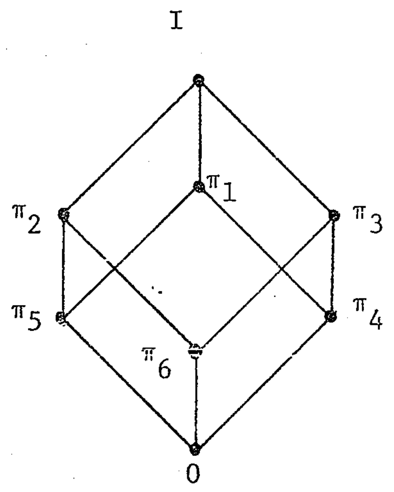
$\pi_2 = \{\overline{1,3,4};\ \overline{2}\}$

$\pi_3 = \{\overline{1,2,4};\ \overline{3}\}$

$\pi_4 = \{\overline{1,2};\ \overline{3};\ \overline{4}\}$

$\pi_5 = \{\overline{1,3};\ \overline{2};\ \overline{4}\}$

$\pi_6 = \{\overline{1,4};\ \overline{2};\ \overline{3}\}$



The generation of the nonredundant sets follows (only nonredundant sets will be listed):

$\pi_1\pi_2 = \pi_5$
$\pi_1\pi_3 = \pi_4$
$\pi_1\pi_4 = \pi_4$
$\pi_1\pi_5 = \pi_5$
$\pi_1\pi_6 = 0$

$\pi_2\pi_3 = \pi_6$
$\pi_2\pi_4 = 0$
$\pi_2\pi_5 = \pi_5$
$\pi_2\pi_6 = \pi_6$

$\pi_3\pi_4 = \pi_4$
$\pi_3\pi_5 = 0$
$\pi_3\pi_6 = \pi_6$

$\pi_4\pi_5 = 0$
$\pi_4\pi_6 = 0$

$\pi_5\pi_6 = 0$

$\pi_1\pi_2\pi_3 = 0$
$\pi_1\pi_2\pi_4 = 0$
$\pi_1\pi_2\pi_6 = 0$

$\pi_1\pi_3\pi_5 = 0$
$\pi_1\pi_3\pi_6 = 0$

$\pi_1\pi_4\pi_5 = 0$

$\pi_2\pi_3\pi_4 = 0$
$\pi_2\pi_3\pi_5 = 0$

$\pi_2\pi_5\pi_6 = 0$

$\pi_3\pi_4\pi_6 = 0$

For machine M there are no decompositions which can be realized with the minimal number of state variables. For example, the decompositions $\{\pi_1,\pi_2,\pi_3\}$ and $\{\pi_4,\pi_5\}$ require 3 and 4 state variables, respectively. However, if the partitions for which $k\pi + \mu\pi > s$ were not retained, no decompositions could be found.

Next the algorithm is applied to the example used by Curtis [8] and presented in Example 2.12.

Example 3.7

The S.P. partitions for the machine are as follows:

$\pi_1 = \{\overline{0,1}; \overline{2}; \overline{3}; \overline{4,5}\}$

$\pi_2 = \{\overline{0,1,2}; \overline{3}; \overline{4,5}\}$

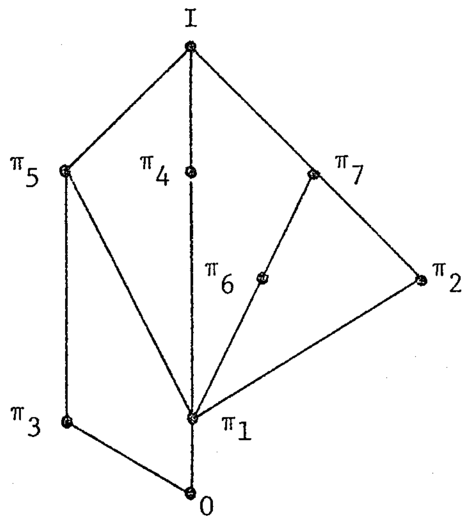
$\pi_3 = \{\overline{0,3}; \overline{2,5}; \overline{1}; \overline{4}\}$

$\pi_4 = \{\overline{0,1,4,5}; \overline{2,3}\}$

$\pi_5 = \{\overline{0,1,3}; \overline{2,4,5}\}$

$\pi_6 = \{\overline{0,1}; \overline{2}; \overline{3,4,5}\}$

$\pi_7 = \{\overline{0,1,2}; \overline{3,4,5}\}$



S.P. lattice

$\pi_1\pi_2 = \pi_1$	$\pi_1\pi_2\pi_7 = \pi_1$
$\pi_1\pi_3 = 0$	
$\pi_1\pi_4 = \pi_1$	$\pi_1\pi_3\pi_5 = 0$
$\pi_1\pi_5 = \pi_1$	
$\pi_1\pi_6 = \pi_1$	$\pi_1\pi_6\pi_7 = \pi_1$
$\pi_1\pi_7 = \pi_1$	
$\pi_2\pi_3 = 0$	$\pi_2\pi_3\pi_5 = 0$
$\pi_2\pi_4 = \pi_1$	
$\pi_2\pi_5 = \pi_1$	$\pi_2\pi_6\pi_7 = \pi_1$
$\pi_2\pi_6 = \pi_1$	
$\pi_2\pi_7 = \pi_2$	
$\pi_3\pi_4 = 0$	$\pi_3\pi_4\pi_5 = 0$
$\pi_3\pi_5 = \pi_3$	
$\pi_3\pi_6 = 0$	$\pi_3\pi_5\pi_6 = 0$
$\pi_3\pi_7 = 0$	$\pi_3\pi_5\pi_7 = 0$

$\pi_4\pi_5 = \pi_1$
$\pi_4\pi_6 = \pi_1$
$\pi_4\pi_7 = \pi_1$

$\pi_5\pi_6 = \pi_1$
$\pi_5\pi_7 = \pi_1$

$\pi_6\pi_7 = \pi_6$

Nonredundant sets whose product is not 0 can be augmented with 0 to produce a decomposition for M.

The two solutions Curtis lists have been found by our algorithm. These sets are $\{\pi_3,\pi_4,\pi_5\}$ and $\{\pi_3,\pi_5,\pi_7\}$. The partition π_1 is not included in these sets as its inclusion would make the sets redundant. But $\pi_4\cdot\pi_5 = \pi_1$ and $\pi_5\cdot\pi_7 = \pi_1$. Thus, π_1 is effectively included in $\{\pi_3,\pi_4,\pi_5\}$ and $\{\pi_3,\pi_5,\pi_7\}$.

In Examples 3.6 and 3.7 there were many possible decompositions for each machine. The choice of the best decomposition requires an exhaustive examination of all decompositions. In the following chapter two possible algorithmic methods of evaluating decompositions are examined.

In deriving a decomposition for a sequential machine, two additional problems should be considered. These are the derivation of output functions and the enhancement of the S.P. structure of sequential machines. These problems and their relation to the decomposition algorithm of this chapter are discussed briefly below.

Algebraic structure theory provides techniques whereby the state variable dependency of the output functions can be reduced, Hartmanis and Stearns [31], Kohavi [42]. To do this it is necessary to find partitions on the set of states, such that the outputs for all states in a block are the same. Partitions of this type are referred to as <u>output-consistent partitions.</u>

Example 3.8

$$\begin{array}{c|c|c||c|}
 & 0 & 1 & z \\
\hline
1 & 2 & 4 & 1 \\
\hline
2 & 1 & 3 & 0 \\
\hline
3 & 4 & 1 & 0 \\
\hline
4 & 3 & 2 & 1 \\
\hline
\end{array}$$

M

For M, $\tau = \{\overline{1,4}; \overline{2,3}\}$ is an output-consistent partition since the outputs for 2 and 3 are the same. The partition $\pi = \{\overline{1,2}; \overline{3,4}\}$ is an S.P. partition. Since $\pi \cdot \tau = 0$, M can be realized using π and τ. Thus, a decomposition which reduces both state and output variable dependence can be obtained.

Unfortunately for many machines there is either a conflict between the reduction of state variable dependence and the reduction of output variable dependence, or no output variable dependence reduction is possible. Because of this conflict, the decomposition method has concentrated on obtaining state variable dependence reduction and ignored the output functions. However, once the decompositions have been derived, output variable dependence reduction may be obtained using the techniques of Kohavi.

For some sequential machines S.P. partitions do not exist, or if they do their structure does not permit economical decompositions. One reason for this is that structure can be lost as the machine is reduced. That is, as redundant states are deleted, the structure of the machine is destroyed. Examples illustrating this can be found in Hartmanis and Stearns [31] and Kohavi [43].

Subsequently, to add structure to a machine it may be necessary to split some of the existing states. The set systems of Hartmanis and Stearns enable us to work with multiple copies of a state.

Definition 3.2 A collection of subsets $A = \{B_i\}$ of S is called a set system on S if and only if

(i) $\cup B_i = S$;

(ii) $B_i \subseteq B_j$ implies $i = j$.

For a machine with no S.P. partitions, set systems which have S.P. can be derived. Thus, the machine can be given a structure and subsequently decomposed. Essentially, set systems with S.P. indicate the states of the machine that have to be split in order to induce structure. Once the S.P. set systems are found, the states which appear more than once in the set system can be split to give a machine with unique states. The decomposition method presented in this chapter can then be applied to the new machine.

Example 3.9

	0	1	z
1	1	2	1
2	3	2	0
3	1	3	1

M

For machine M, $A = \{\overline{1,3}; \overline{2,3}\}$ is a set system with S.P. By splitting state 3, an equivalent machine, M', with 4 unique states can be defined

	0	1	z
1	1	2	1
2	3'	2	0
3'	1	3"	1
3"	1	3"	1

M'

For M', $\pi = \{\overline{1,3'}; \overline{2,3''}\}$ is an S.P. partition.

Chapter 4 Programming and Evaluation Techniques

4.1. Introduction

This chapter is concerned with two aspects of the method for generating nonredundant sets presented in the last chapter; implementation of the method and evaluation of the nonredundant sets produced.

One of the problems of implementation involves determining an efficient method for representing the relationships between S.P. partitions. This is because the S.P. lattice representation becomes complicated when a large number of S.P. partitions are considered. A tabular representation is introduced and demonstrated to be a convenient means of representing the ">" relationships between partitions. In addition, other useful properties of the table are derived.

The usual method of evaluating a state assignment consists of:

(i) finding a minimal realization in terms of AND, OR, and NOT gates;

(ii) counting the number of diodes necessary and using this as a measure of the assignment.

There are several problems associated with this evaluation technique.

(1) It is device dependent. An assignment that is minimal with respect to AND, OR, and NOT gates may not be a minimal assignment when using NAND gates.

(2) The Boolean minimization technique will not necessarily guarantee the minimal expression. Thus, the diode count is not just a measure of the assignment, but also a measure of the minimization technique.

(3) Evaluating more than a couple of assignments is a long process. Because of this, not all possible assignments would be explored.

We present alternate methods for obtaining the "best" decomposition for a machine. The first method eliminates pairs of S.P. partitions which are "uneconomical" with regard to other pairs of partitions. While eliminating many of the possible decompositions, this technique is not able to evaluate the decompositions not eliminated.

The second solution is the development of a fast evaluation method, which is also device independent. The method presented, ROM (Read-Only-Memory) evaluation determines the number of bits necessary for a ROM realization of a decomposition. The evaluation involves simple arithmetic calculations which are performed as the nonredundant sets are derived.

The number of bits necessary for a ROM implementation provides an efficient measure of the variable dependence of a decomposition. Since ROM evaluation measures variable dependence, it is a device independent evaluation technique.

ROM evaluation is also able to detect the "uneconomic" pairs of partitions mentioned in our first evaluation technique. Consequently, it will only be necessary to use one evaluation method rather than two. In addition, decompositions which involve subtle "redundancy", as described by Hartmanis and Stearns [31], have this "redundancy" reflected in their ROM sizes.

4.2 A Tabular Representation for the S.P. Lattice

In the previous chapters we have been using the lattice structure to represent the greater than-less than relationships between S.P. partitions. For a small lattice it is relatively easy to "read" the relationships directly from the lattice. However, for a larger lattice, obtaining the relationships requires a great deal of tedious searching. For this reason a tabular representation of the S.P. lattice has been developed.

Another motivation for using a tabular representation becomes apparent when it is desired to automate the algorithm presented in the previous chapter. A computer representation of the S.P. lattice takes the form of a list structure, while a table is represented as an array. Searching an array is much faster and more efficient than searching a list of comparable size.

The tabular representation of an S.P. lattice that we will use shows the greater than or equal relationships between the non-trivial S.P. partitions. For a machine with n S.P. partitions, an

$n \times n$ table is used. A '1' at the intersection of row i and column j, indicates that partition π_i is greater than or equal to partition π_j. A '0' indicates that π_i is not greater than or equal to π_j. Clearly, the main diagonal of the table is all 1's. This table will be referred to as the greater than or equal (G.E.) table.

Notation The value at the intersection of row i and column j will be represented as $G.E\ (\pi_i,\pi_j)$. The vector of values for row i will be denoted

$$G.E.(\pi_i) = (G.E.(\pi_i,\pi_1),\ldots,G.E.(\pi_i,\pi_n)).$$

In addition, to indicating greater than or equal relationships, the G.E. table can also be used to determine the product of two partitions.

Theorem 4.1 For the G.E. table, $G.E.(\pi_i)\cdot G.E.(\pi_j) = G.E.(\pi_i\cdot\pi_j)$.

Proof We must prove that

$$G.E.(\pi_i,\pi_\ell)\cdot G.E.(\pi_j,\pi_\ell) = G.E.(\pi_i\cdot\pi_j,\pi_\ell),\quad \ell = 1,\ldots,n\ .$$

(i) If $G.E.(\pi_i,\pi_\ell)\cdot G.E.(\pi_j,\pi_\ell) = 1$, then
$G.E.(\pi_i,\pi_\ell) = 1$ and $G.E.(\pi_j,\pi_\ell) = 1$
That is, $\pi_i \geq \pi_\ell$ and $\pi_j \geq \pi_\ell$
$\therefore$ $\pi_i\cdot\pi_j \geq \pi_\ell$ and $G.E.(\pi_i\cdot\pi_j,\pi_\ell) = 1$
Similarly, if $G.E.(\pi_i\cdot\pi_j,\pi_\ell) = 1$, then $G.E.(\pi_i,\pi_\ell)\cdot G.E(\pi_j,\pi_\ell) = 1$.

(ii) If $G.E.(\pi_i,\pi_\ell)\cdot G.E.(\pi_j,\pi_\ell) = 0$ then

$$G.E.(\pi_i,\pi_\ell) = 0 \text{ or } G.E.(\pi_j,\pi_\ell) = 0$$

That is, $\pi_i \ngeq \pi_\ell$ or $\pi_j \ngeq \pi_\ell$.
$\therefore$ $\pi_i\cdot\pi_j \ngeq \pi_\ell$ and $G.E.(\pi_i\cdot\pi_j,\pi_\ell) = 0$.
Similarly, if $G.E.(\pi_i\cdot\pi_j,\pi_\ell) = 0$ then

$$G.E.(\pi_i,\pi_\ell)\cdot G.E.(\pi_j,\pi_\ell) = 0.$$

Thus, $G.E.(\pi_i\cdot\pi_j,\pi_\ell) = G.E.(\pi_i,\pi_\ell)\cdot G.E.(\pi_j,\pi_\ell), \ell=1,\ldots,n.$

Corollary 4.1.1 $G.E.(\pi_i)\cdot G.E.(\pi_j) = G.E.(\pi_i)$ if and only if $\pi_i \leq \pi_j$.

Corollary 4.1.2 $G.E.(\pi_i)\cdot G.E.(\pi_j) = (0,\ldots,0)$ if and only if $\pi_i\cdot\pi_j = 0$.

By indexing the table inversely with G.E.$(\pi_i)\cdot$G.E.(π_j), the product of the partitions π_i and π_j can be easily obtained.

Example 4.1

	i_0	i_1
1	8	2
2	6	1
3	7	4
4	5	3
5	1	3
6	3	4
7	2	1
8	4	2

M

The S.P. partitions for M are

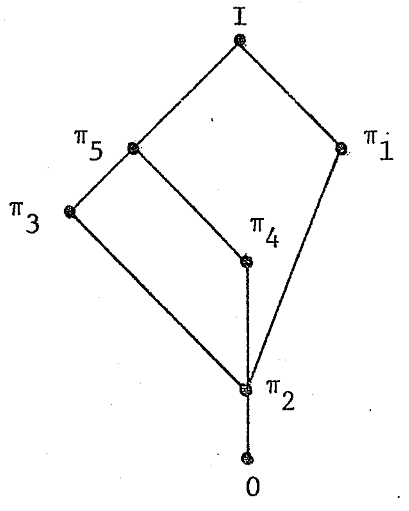
$$\pi_1 = \{\overline{1,2,3,4};\ \overline{5,6,7,8}\}$$

$$\pi_2 = \{\overline{1,4};\ \overline{2,3};\ \overline{5,8};\ \overline{6,7}\}$$

$$\pi_3 = \{\overline{1,4,5,8};\ \overline{2,3};\ \overline{6,7}\}$$

$$\pi_4 = \{\overline{1,4};\ \overline{2,3,6,7};\ \overline{5,8}\}$$

$$\pi_5 = \{\overline{1,4,5,8};\ \overline{2,3,6,7}\}$$



S.P. lattice

	π_1	π_2	π_3	π_4	π_5
π_1	1	1	0	0	0
π_2	0	1	0	0	0
π_3	0	1	1	0	0
π_4	0	1	0	1	0
π_5	0	1	1	1	1

G.E. table

The greater than or equal relationship between any partitions can easily be found by finding their row and column intersections. For example, $\pi_5 \geq \pi_4$ since there is a '1' at the intersection of row 5 and column 4.

From the S.P. lattice the product of π_1 and π_4 is π_2. Using the G.E. table

$$\begin{aligned} G.E.(\pi_1)\cdot G.E.(\pi_4) &= (1,1,0,0,0)\cdot(0,1,0,1,0) \\ &= (0,1,0,0,0) \\ &= G.E.(\pi_2). \end{aligned}$$

Thus from the G.E. table we also obtain $\pi_1 \cdot \pi_4 = \pi_2$.

The G.E. table can also be used as a less than or equal (L.E.) table. Because of the construction of the G.E. table, the intersection of column i and row j indicates whether a less than or equal relationship holds between π_i and π_j. If the intersection is equal to 1, then $\pi_i \leq \pi_j$, If the intersection is 0, then $\pi_i \nleq \pi_j$.

In order to facilitate the description of the properties of the G.E. table when used to represent the less than or equal relationships, the following notation is introduced.

Notation $L.E.(\pi_i) = (G.E.(\pi_1,\pi_i),\ldots\ldots,G.E.(\pi_n,\pi_i))$

$L.E.(\pi_i)\cdot L.E.(\pi_j) = (G.E.(\pi_1,\pi_i)\cdot G.E.(\pi_1,\pi_j),\ldots,G.E.(\pi_n,\pi_i)\cdot G.E.(\pi_n,\pi_j))$.

While the G.E. table can be used to find the product of two partitions, the L.E. table can be used to find the sum of two partitions. The following theorem can be proved in a similar manner to Theorem 4.1.

Theorem 4.2 For the L.E. table, $L.E.(\pi_i)\cdot L.E.(\pi_j) = L.E.(\pi_j) = L.E.(\pi_i+\pi_j)$.

Corollary 4.2.1 $L.E.(\pi_i)\cdot L.E.(\pi_j) = L.E.(\pi_i)$ if and only if $\pi_i \geq \pi_j$.

Corollary 4.2.2 $L.E.(\pi_i)\cdot L.E.(\pi_j) = (0,\ldots,0)$ if and only if $\pi_i + \pi_j = I$.

The use of Theorem 4.2 is demonstrated in the following example.

Example 4.2

	i_0	i_1
1	3	7
2	4	8
3	1	6
4	2	5
5	2	4
6	1	3
7	4	4
8	3	3

M

The S.P. partitions for M are

$\pi_1 = \{\overline{1,2};\ \overline{3,4};\ \overline{5,6};\ \overline{7,8}\}$

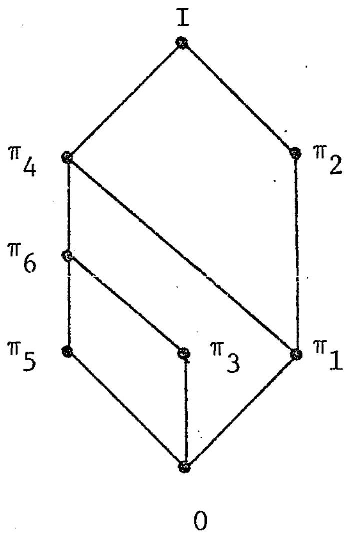
$\pi_2 = \{\overline{1,2,3,4};\ \overline{5,6,7,8}\}$

$\pi_3 = \{\overline{1};\ \overline{2};\ \overline{3};\ \overline{4,5};\ \overline{6};\ \overline{7};\ \overline{8}\}$

$\pi_4 = \{\overline{1,2};\ \overline{3,4,5,6};\ \overline{7,8}\}$

$\pi_5 = \{\overline{1};\ \overline{2};\ \overline{3,6};\ \overline{4};\ \overline{5};\ \overline{7};\ \overline{8}\}$

$\pi_6 = \{\overline{1};\ \overline{2};\ \overline{3,6};\ \overline{4,5};\ \overline{7};\ \overline{8}\}$



	π_1	π_2	π_3	π_4	π_5	π_6
π_1	1	0	0	0	0	0
π_2	1	1	0	0	0	0
π_3	0	0	1	0	0	0
π_4	1	0	1	1	1	1
π_5	0	0	0	0	1	0
π_6	0	0	1	0	1	1

L.E. (G.E.) table

The less than or equal relationships can be determined from the L.E. table. For example, $\text{G.E.}(\pi_4,\pi_6) = 1$ indicates that $\pi_6 \le \pi_4$.

From the S.P. lattice, the sum of π_3 and π_5 is π_6. From the L.E. table

$$\begin{aligned}\text{L.E.}(\pi_3)\cdot\text{L.E.}(\pi_5) &= (\text{G.E.}(\pi_1,\pi_3)\cdot\text{G.E.}(\pi_1,\pi_5),\ldots,\text{G.E.}(\pi_6,\pi_3)\cdot\text{G.E.}(\pi_6,\pi_5))\\ &= (0,0,1,1,0,1)\cdot(0,0,0,1,1,1)\\ &= (0,0,0,1,0,1)\\ &= \text{L.E.}(\pi_6).\end{aligned}$$

Thus, the L.E. table also gives $\pi_3+\pi_5 = \pi_6$.

4.3 Uneconomic Pairs

In an attempt to reduce the number of S.P. partitions involved when deriving a decomposition for a sequential machine, we introduce the concept of uneconomic pairs. Essentially, uneconomic pairs identify pairs of partitions which are costly in comparison with other pairs.

For some pairs of partitions (π_i,π_j) where $\pi_i > \pi_j$, the information transfer from π_i to π_j is not as efficient as is possible. That is, in realizing π_j from π_i, not all the information available in π_i is used. (To realize π_j from π_i, a partition τ_j such that $\pi_i \cdot \tau_j = \pi_j$ is required.) However, there may be another partition π_k such that $\pi_i > \pi_j > \pi_k$, and the partition τ_k which realizes π_k from π_i has as many blocks as partition τ_j.

Thus, the serial connection of (π_i,π_k) "costs" as much to construct as the serial connection of (π_i,π_j). However, (π_i,π_k) provides more information than (π_i,π_j), since $\pi_k < \pi_j$. A pair of partitions which is uneconomical in comparison with another pair is demonstrated next.

Example 4.3 Consider the pairs of partitions (π_5,π_4) and (π_5,π_2) from Example 4.1, where $\pi_2 = \{\overline{1,4}; \overline{2,3}; \overline{5,8}; \overline{6,7}\}$, $\pi_4 = \{\overline{1,4}; \overline{2,3,6,7}; \overline{5,8}\}$, and $\pi_5 = \{\overline{1,4,5,8}; \overline{2,3,6,7}\}$. To realize π_4 from π_5, a partition τ_4 such that $\pi_5 \cdot \tau_4$ is required.

Let $\tau_4 = \{\overline{1,2,3,4,6,7}; \overline{5,8}\}$

$$\pi_5 \cdot \tau_4 = \{\overline{1,4,5,8}; \overline{2,3,6,7} \cdot \{\overline{1,2,3,4,6,7}; \overline{5,8}\} .$$
$$= \pi_4$$

Let $\tau_2 = \{\overline{1,4,6,7}; \overline{2,3,5,8}\}$ Thus

$$\pi_5 \cdot \tau_2 = \{\overline{1,4,5,8}; \overline{2,3,6,7}\} \cdot \{\overline{1,4,6,7}; \overline{2,3,5,8}\} .$$
$$= \pi_2$$

Since τ_2 and τ_4 have the same number of blocks and the same inputs, the cost of realizing τ_2 is identical to the cost of realizing τ_4. However, π_2 contains more information than π_4. To realize the trivial partition 0 from π_2 requires a two-block partition, while realizing 0 from π_4 requires a four-block partition. Consequently, it is more economical to use the pair (π_5,π_2) in a decomposition than the pair (π_5,π_4).

The number of nonredundant sets for a machine can be reduced by deleting all the uneconomic pairs before generating the nonredundant sets. To facilitate this, we present a definition of uneconomic pairs and a method for determining uneconomic pairs.

Definition 4.1 An uneconomic pair (U.P.) of S.P. partitions is a pair of partitions (π_i,π_j) where $\pi_i > \pi_j$ such that there exists another pair of partitions (π_k,π_ℓ), $\pi_k > \pi_\ell$ and either

(i) $\pi_i=\pi_k$, $\pi_j > \pi_\ell$ and $e(\pi_i|\pi_j) = e(\pi_i|\pi_\ell)$;

or

(ii) $\pi_j = \pi_\ell$, $\pi_i < \pi_k$ and $e(\pi_i|\pi_j) = e(\pi_k|\pi_j)$.

Trivially, the pair (π_i,π_j) is uneconomic if $e(\pi_i|\pi_j) = e(\pi_i|0)$, since $\pi_j > 0$. Similarly, (π_i,π_j) is uneconomic if $e(\pi_i|\pi_j) = e(I|\pi_j)$, since $I > \pi_i$.

Applying Definition 4.1 to the pairs of partitions in Example 4.3:

$(\pi_i,\pi_j) = (\pi_5,\pi_4)$ and $(\pi_k,\pi_\ell) = (\pi_5,\pi_2)$. As $\pi_i = \pi_k$, condition (i) is applicable.

$$e(\pi_i|\pi_j) = e(\pi_5|\pi_4) = 1 \text{ and } e(\pi_k|\pi_\ell) = e(\pi_5|\pi_2) = 1.$$

Therefore, by condition (i), the pair (π_5,π_4) is a U.P.

In determining the uneconomic pairs, separate procedures are required for conditions (i) and (ii).

A Method for Determining Uneconomic Pairs

Uneconomic Pairs of type (i)

(1) Consider each S.P. partition π in turn and make a list of the S.P. partitions that it is greater than.

(2) Separate the list into blocks of partitions that have the same $e(\pi|\tau)$ value.

(3) If partitions π_i and π_j are in the same $e(\pi|\tau)$ block of a list and $\pi_j > \pi_i$, then the pair (π,π_j) is a U.P.

Uneconomic Pairs of type (ii)

(1) Consider each S.P. partition π in turn and make a list of the S.P. partitions that are greater than π.

(2) Separate each list into blocks of partitions that have the same $e(\tau|\pi)$ value.

(3) If partitions π_i and π_j are in the same $e(\tau|\pi)$ block of a list and $\pi_i > \pi_j$, then the pair (π_j, π) is a U.P.

The algorithm is used below to determine U.P.'s.

Example 4.4

	i_0	i_1	i_2
1	1	4	2
2	1	4	2
3	1	4	2
4	5	1	3
5	5	1	3

M

The S.P. partitions for M:

$\pi_1 = \{\overline{1,2}; \overline{3}; \overline{4}; \overline{5}\}$

$\pi_2 = \{\overline{1,3}; \overline{2}; \overline{4}; \overline{5}\}$

$\pi_3 = \{\overline{1,4,5}; \overline{2,3}\}$

$\pi_4 = \{\overline{1}; \overline{2,3}; \overline{4}; \overline{5}\}$

$\pi_5 = \{\overline{1}; \overline{2}; \overline{3}; \overline{4,5}\}$

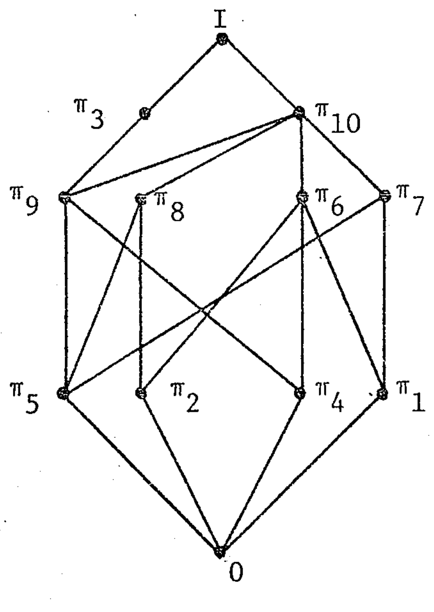
$\pi_6 = \{\overline{1,2,3}; \overline{4,5}\}$

$\pi_7 = \{\overline{1,2}; \overline{3}; \overline{4,5}\}$

$\pi_8 = \{\overline{1,3}; \overline{2}; \overline{4,5}\}$

$\pi_9 = \{\overline{1}; \overline{2,3}; \overline{4,5}\}$

$\pi_{10} = \{\overline{1,2,3}; \overline{4,5}\}$



S.P. lattice

In order to derive the U.P.'s we will make use of the G.E. table.

	π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9	π_{10}
π_1	1	0	0	0	0	0	0	0	0	0
π_2	0	1	0	0	0	0	0	0	0	0
π_3	0	0	1	1	1	0	0	0	1	0
π_4	0	0	0	1	0	0	0	0	0	0
π_5	0	0	0	0	1	0	0	0	0	0
π_6	1	1	0	1	0	1	0	0	0	0
π_7	1	0	0	0	1	0	1	0	0	0
π_8	0	1	0	0	1	0	0	1	0	0
π_9	0	0	0	1	1	0	0	0	1	0
π_{10}	1	1	0	1	1	1	1	1	1	1

G.E. Table

A detailed description of the derivation of U.P.'s of type (i), for π_{10} is given. For partition π_{10} derive the list of partitions π_i such that $\pi_{10} > \pi_i$. The list is then divided into blocks with the same $e(\pi_{10}|\pi_i)$ value.

$$L_{10} = (\{\pi_9,\pi_8,\pi_7,\pi_6,\pi_4,\pi_2,\pi_1,\}, \{\pi_5,0\})$$

Since π_9 and π_4 are in the same block of L_{10} and $\pi_9 > \pi_4$ from the G.E. table, the pair (π_{10},π_9) is a U.P. Similarly, since π_5 and 0 are in the same block and $\pi_5 > 0$, the pair (π_{10},π_5) is a U.P. All the type (i) U.P.'s can be found in a similar fashion.

(i) $L_{10} = (\{\pi_9,\pi_8,\pi_7,\pi_6,\pi_4,\pi_2,\pi_1\}, \{\pi_5,0\})$
U.P.'s: (π_{10},π_9), (π_{10},π_8), (π_{10},π_7), (π_{10},π_6), (π_{10},π_5)
$L_9 = (\{\pi_5,\pi_4,0\})$
U.P.'s: (π_9,π_5), (π_9,π_4)
$L_8 = (\{\pi_5,\pi_2,0)\}$
U.P's: (π_8,π_5), (π_8,π_2)
$L_7 = (\{\pi_5,\pi_1,0\})$
U.P.'s: (π_7,π_5), (π_7,π_1)
$L_6 = (\{\pi_4,\pi_2,\pi_1\}, \{0\})$, $L_5 = (\{0\})$, $L_4 = (\{0\})$
no U.P.'s.
$L_3 = (\{\pi_9,\pi_5\}, \{\pi_4,0\})$
U.P.'s: (π_3,π_9), (π_3,π_4)
$L_2 = (\{0\})$, $L_1 = (\{0\})$
no U.P.'s.

Some of the uneconomic pairs of type (ii), involving partition π_1, will be derived in detail to illustrate the process. For π_1 derive the list of partitions π_i, such that $\pi_i > \pi_1$ and divide the list into block depending on the $e(\pi_i|\pi_1)$ values.

$G_1 = (\{\pi_6,\pi_7,\pi_{10}\}, \{I\})$

Since π_6 and π_{10} are in the same block of G_1 and $\pi_{10} > 6$, the pair (π_1,π_6) is an uneconomic pair.

All the uneconomic pairs of type (ii) are given below.

(ii) $G_1 = (\{\pi_6,\pi_7,\pi_{10}\}, \{I\})$
U.P.'s: $(\pi_1,\pi_6),(\pi_1,\pi_7)$

$G_2 = (\{\pi_6,\pi_8,\pi_{10}\}, \{I\})$
U.P.'s: $(\pi_2,\pi_6),(\pi_2,\pi_8)$

$G_3 = (\{I\})$
no U.P.'s.

$G_4 = (\{\pi_6,\pi_9,\pi_{10}\}, \{\pi_3,I\})$
U.P.'s: $(\pi_4,\pi_6),(\pi_4,\pi_9),(\pi_3,\pi_4)$

$G_5 = (\{\pi_7,\pi_8,\pi_9,\pi_3\}, \{\pi_{10},I\})$
U.P.'s: $(\pi_5,\pi_9),(\pi_5,\pi_{10})$

$G_6 = (\{\pi_{10}\},\{I\})$, $G_7 = (\{\pi_{10}\},\{I\})$, $G_8 = (\{\pi_{10}\},\{I\})$,
$G_9 = (\{\pi_3,\pi_{10}\},\{I\})$, $G_{10} = (\{I\})$.
no U.P.'s.

Consolidating the type (i) and type (ii) uneconomic pairs, we obtain: (π_1,π_6), (π_1,π_7), (π_2,π_6), (π_2,π_8), (π_3,π_4), (π_3,π_9), (π_4,π_6), (π_4,π_9), (π_5,π_7), (π_5,π_8), (π_5,π_9), (π_5,π_{10}), (π_6,π_{10}), (π_7,π_{10}), (π_8,π_{10}), (π_9,π_{10}).

Normally, for machine M, $\frac{10\cdot 9}{2} = 45$ pairs of S.P. partitions would be used to generate the nonredundant sets. By removing the 16 uneconomic pairs, this number is reduced considerably. The generation of nonredundant sets using only the economic pairs follows.

$\pi_1\pi_2 = 0$

$\pi_1\pi_3 = 0$

$\pi_1\pi_4 = 0$

$\pi_1\pi_5 = 0$

$\pi_1\pi_8 = 0$

$\pi_1\pi_9 = 0$

$\pi_1\pi_{10} = \pi_1$

$\pi_1\pi_2\pi_3 = 0$

$\pi_1\pi_3\pi_{10} = 0$

$\pi_1\pi_4\pi_{10} = 0$

$\pi_2\pi_3 = 0$

$\pi_2\pi_4 = 0$

$\pi_2\pi_5 = 0$

$\pi_2\pi_7 = 0$

$\pi_2\pi_9 = 0$

$\pi_2\pi_{10} = \pi_2$

$\pi_2\pi_3\pi_{10} = 0$

$\pi_2\pi_4\pi_{10} = 0$

$\pi_3\pi_5 = \pi_5$

$\pi_3\pi_6 = \pi_4$

$\pi_3\pi_7 = \pi_5$

$\pi_3\pi_8 = \pi_5$

$\pi_3\pi_{10} = \pi_9$

$\pi_3\pi_5\pi_6 = 0$

$\pi_3\pi_6\pi_7 = 0$

$\pi_4\pi_5 = 0$

$\pi_4\pi_7 = 0$

$\pi_4\pi_8 = 0$

$\pi_4\pi_{10} = \pi_4$

$\pi_5\pi_6 = 0$

$\pi_6\pi_7 = \pi_1$

$\pi_6\pi_8 = \pi_2$

$\pi_6\pi_9 = \pi_4$

$\pi_6\pi_7\pi_8 = 0$

$\pi_6\pi_7\pi_9 = 0$

$\pi_6\pi_8\pi_9 = 0$

$\pi_7\pi_8 = \pi_5$

$\pi_7\pi_9 = \pi_5$

$\pi_8\pi_9 = \pi_5$

Deleting uneconomic pairs noticeably reduces the number of decompositions derived. However, there still remains the problem of selecting the "best" decomposition from those remaining. In the following section an evaluation procedure is presented which provides a measure of the variable dependence of any decomposition.

4.4 Read-Only-Memory (ROM) Evaluation

Introduction

With medium scale integrated circuits (M.S.I.), Read-only-memories (ROMs) have become available for use in combinational logic [48]. Using ROMs considerably simplifies the realization of combinational logic. The truth table for the desired functions simply has to be entered into the ROM. Thus, the Boolean function does not have to be simplified, saving considerable time and effort for the logic designer.

ROMs can also be used in constructing sequential machines, by inserting a ROM for the combinational logic [32]. Consequently, the complexity of the combinational logic is no longer a problem in state assignment for sequential machines. However, as the size of a machine increases, the size of the ROM necessary to realize the machine becomes very large.

Thus, the decomposition of a machine into a series-parallel arrangement of submachines becomes important, not only for state assignment, but also for constructing a large machine from small to medium-sized ROMs.

ROM Size and Decomposition

The number of bits in a ROM used to realize a state machine $M = (S,I,\delta)$, can be easily calculated from the number of state variables and the number of input variables [32].

$$R = S \times 2^{P+S} \qquad (1),$$

where S = number of state variables

P = number of input variables.

One of the objectives of the structure theory of Hartmanis and Stearns is to reduce the interdependence between the state variables used to realize a state machine. By examining (1), we see that the term 2^{P+S} is a measure of the dependence of the state variables upon themselves and upon the P external input variables.

Thus, the ROM evaluation formula provides us with a simple, numerical procedure to evaluate the size, in terms of bits, of any decomposition. By evaluating all the nonredundant decompositions for a machine, the most economical decomposition can be determined. Intuitively, the ROM size of a decomposition should provide an accurate evaluation of the decomposition, regardless of the technology used for implementation.

To demonstrate the effectiveness of ROM size in evaluating decompositions, we will compare ROM size with the diode counting technique used by Hartmanis and Stearns. The example below is taken from p.33 of Hartmanis and Stearns [31].

Example 4.5

	i_0	i_1
1	4	3
2	6	3
3	5	2
4	2	5
5	1	4
6	3	4

E

$0 = \{\overline{1}; \overline{2}; \overline{3}; \overline{4}; \overline{5}; \overline{6}\}$

$\pi_1 = \{\overline{1,2,3}; \overline{4,5,6}\}$

$\pi_2 = \{\overline{1,6}; \overline{2,5}; \overline{3,4}\}$

$I = \{\overline{1,2,3,4,5,6}\}$

S.P. partitions for machine E

For machine E, Hartmanis and Stearns give two assignments. The first assignment, α, is random, while the second assignment, β, is based on the S.P. partitions π_1 and π_2.

		y_1	y_2	y_3
1	→	0	0	0
2	→	0	0	1
3	→	0	1	0
4	→	0	1	1
5	→	1	0	0
6	→	1	0	1

Assignment α

		y_1	y_2	y_3
1	→	1	1	0
2	→	1	0	1
3	→	1	0	0
4	→	0	0	0
5	→	0	0	1
6	→	0	1	0

Assignment β

The next-state equations derived from assignment α require 38 diodes, while those derived from assignment β require only 10 diodes.

For machine E, $S = 3$ and $P = 1$. To realize E as a single ROM, would require a ROM with

$$\begin{aligned} R &= S \times 2^{P+S} \\ &= 3 \times 2^4 \\ &= 48 \text{ bits} . \end{aligned}$$

Machine E could be realized by two ROMS which realize partitions π_1 and π_2.

For π_1, $S_1 = 1$ and $P_1 = 1$

$$\begin{aligned} R_1 &= S_1 \times 2^{P_1+S_1} \\ &= 1 \times 2^{1+1} \\ &= 4 . \end{aligned}$$

For π_2, $S_2 = 2$ and $P_2 = 1$

$$\begin{aligned} R_2 &= S_2 + 2^{P_2+S_2} \\ &= 2 \times 2^{1+2} \\ &= 16 . \end{aligned}$$

The combined size of the two ROMs is

$$\begin{aligned} R_{\pi},\pi_2 &= R_1 + R_2 \\ &= 20 \text{ bits.} \end{aligned}$$

The percentage saving indicated by the diode counting techique is 74%, while ROM size shows a saving of 58%. The difference occurs because the equations produced by Hartmanis and Stearns take into consideration the reduced dependence resulting from using the partition pair $(\{\overline{1,3,4,6};\ \overline{2,5}\},\ \{\overline{1,6};\ \overline{2,3,4,5}\})$ in the assignment. Since we have not considered this partition pair in evaluating ROM size, our results are not as accurate.

However, ROM size evaluation can be applied faster and easier than can diode counting, which requires a Boolean simplification step. Except in the case of partition pairs, ROM evaluation provides a good general measure of the variable dependence of a decomposition. As exact minimization is no longer as important as previously, ROM evaluation is a useful evaluation technique.

Example 4.6 illustrates ROM evaluation for serial decompositions. Before presenting the example, some notation and explanation is given.

Notation: Let $n(IP)$ denote the number of inputs in IP. Then $N(IP) = \lceil \log_2 n(IP) \rceil$ gives the minimal number of input variables necessary to represent the inputs.

For a single partition π, the number of state variables is given by $e(\pi)$ and the number of input variables is given by $N(IP)$. Substituting into (1), the ROM size for a single partition is given by

$$R_\pi = e(\pi)\cdot 2^{N(IP)+e(\pi)} \qquad (2)$$

When two S.P. partitions, π_i and π_j, are realized together, the ROM size may be less than the ROM size for a separate realization of the two partitions. A reduction will generally occur if either of the partitions is greater than the other. If $\pi_i > \pi_j$, then π_j may be realized from π_i by a partition τ_j, where $\pi_i \cdot \tau_j = \pi_j$.

The value $\epsilon(\pi_i|\pi_j)$ is the minimum number of blocks required in the partition τ_j. The number of state variables required to realize τ_j, by a state machine M_j, is $e(\pi_i|\pi_j)$.

The machine M_j also requires input information from π_i. The number of input variables required from π_i is $e(\pi_i)$. Thus, the total number of input variables required for M_j is $e(\pi_i) + N(IP)$. Substituting into (1), the ROM size to realize π_j from π_i is

$$R_{\tau_j} = e(\pi_i|\pi_j)\cdot 2^{e(\pi_i)+N(IP)+e(\pi_i|\pi_j)} \qquad (3)$$

Equation (3) is used to evaluate serial decompositions in the following example.

Example 4.6

	i_0	i_1	i_2	i_3	i_4
1	2	1	5	8	3
2	1	2	6	7	4
3	4	3	6	6	1
4	3	4	5	5	2
5	5	6	3	4	7
6	6	5	4	3	8
7	7	8	4	2	5
8	8	7	3	1	6

M

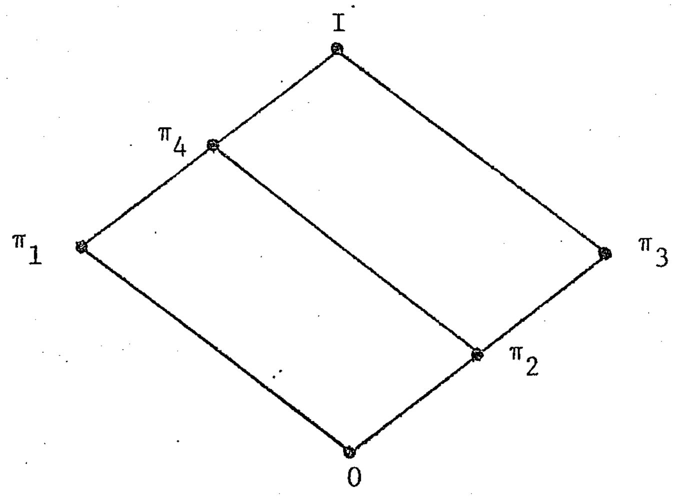
$\pi_i = \{\overline{1,4};\ \overline{2,3};\ \overline{5,8};\ \overline{6,7}\}$

$\pi_2 = \{\overline{1,2};\ \overline{3,4};\ \overline{5,6};\ \overline{7,8}\}$

$\pi_3 = \{\overline{1,2,7,8};\ \overline{3,4,5,6}\}$

$\pi_4 = \{\overline{1,2,3,4};\ \overline{5,6,7,8}\}$

S.P. partitions for M



S.P. lattice

For M, $S = 3$ and $P = 3$. To realize M with a single ROM requires a ROM with

$$\begin{aligned} R &= S \times 2^{P+S} \\ &= 3 \cdot 2^{3+3} \\ &= 192 \text{ bits.} \end{aligned}$$

One possible serial decomposition for M is $\{\pi_4, \pi_1, 0\}$, where $\pi_4 > \pi_1 > 0$

$$e(\pi_4) = 1 \text{ and } N(IP) = 3.$$

Substituting into (2)

$$\begin{aligned} R_{\pi_4} &= e(\pi_4) \cdot 2^{N(IP)+e(\pi_4)} \\ &= 1 \cdot 2^{3+1} \\ &= 16 \text{ bits.} \end{aligned}$$

To realize π_1 we can use state information from π_4, In order to do this a partition τ_i, such that $\pi_4 \cdot \tau_1 = \pi_1$, is required. Let $\tau_1 = \{\overline{1,4,5,8}; \overline{2,3,6,7}\}$.

$$e(\tau_1) = 1 \text{ and } e(\pi_4 | \pi_1) = 1.$$

Substituting into (3)

$$\begin{aligned} R_{\tau_1} &= e(\pi_4 | \pi_1) \cdot 2^{e(\pi_4)+N(IP)+e(\pi_4|\pi_1)} \\ &= 1.2^{1+3+1} \\ &= 32 \text{ bits.} \end{aligned}$$

Similarly, the trivial partition 0 can be realized from π_1 at a cost of

$$\begin{aligned} R_{\tau_0} &= e(\pi_1 | 0) \cdot 2^{e(\pi_1)+N(IP)+e(\pi_1|0)} \\ &= 1 \cdot 2^{2+3+1} \\ &= 64 \text{ bits.} \end{aligned}$$

The total cost for the decomposition $\{\pi_4, \pi_1, 0\}$ is $16+32+64 = 112$ bits. This decomposition gives a saving of 80 bits compared with the ROM size required when realizing M as a single ROM.

The ROM size of a decomposition consisting of 3,...,n partitions could be calculated in a manner similar to that used above for pairs. This would involve considering the information transfers between partitions for each set examined. Obviously, a detailed evaluation of this type is time consuming. To overcome this problem, we present a heuristic method for evaluating the size of a nonredundant set as it is being constructed. The algorithm presented calculates the ROM size of a nonredundant $n^{\underline{\text{tuple}}}$ from the ROM sizes of its constituent, nonredundant $n\text{-}1^{\underline{\text{tuples}}}$. Before the algorithm is stated, a modification is necessary to the method by which the ROM size of a pair of partitions is calculated.

It was stated that, generally, there was a decrease in the ROM size required for a partition π_j, if π_j was realized with π_i, where $\pi_i > \pi_j$. There is a minor exception to this rule which occurs when (π_i,π_j) is an uneconomic pair.

For example 4.4, the S.P. partitions $\pi_5 = \{\overline{1}; \overline{2}; \overline{3}; \overline{4,5}\}$ and $\pi_7 = \{\overline{1,2,3}; \overline{4,5}\}$ are an uneconomic pair. The ROM cost of π_5, by itself, is

$$\begin{aligned} R_{\pi_5} &= e(\pi_5)\cdot 2^{N(IP)+e(\pi_5)} \\ &= 2\cdot 2^{2+2} \\ &= 32 \text{ bits.} \end{aligned}$$

To realize π_5 from π_7 a partition τ_5, such that $\pi_7\cdot\tau_5 = \pi_5$, is required. The ROM cost of τ_5 is

$$\begin{aligned} R_{\tau_5} &= e(\pi_7|\pi_5)\cdot 2^{e(\pi_7)+N(IP)+e(\pi_7|\pi_5)} \\ &= 2\cdot 2^{1+2+2} \\ &= 64 \text{ bits.} \end{aligned}$$

Thus, for the pair $\{\pi_5,\pi_7\}$ it is more economical to realize π_5 by itself, without any information from π_7.

Two possible ways of overcoming this problem are:

(i) delete the uneconomic pairs;

(ii) compare the size of π_i when realized by itself and the size when realized with another partition, and choose the smaller.

The method presented below uses the latter solution.

Notation: The following symbols are used to state the algorithm.

R_i	- ROM size of π_i realized by itself;
$R_{i,j}$	- ROM size of π_i realized in conjunction with π_j;
$R_{i,1\ldots,n}$	- ROM size of π_i realized in conjunction with $\pi_1,\ldots,\pi_n$.

Method for Calculating ROM Sizes

(i) Calculate the ROM size of each S.P. partition when realized by itself using (2).

(ii) Calculate the ROM size for each partition in conjunction with each of the remaining partitions,

(a) if the relation "<" does not hold between π_i and π_j or π_j and π_i, then the ROM sizes of π_i and π_j have already been calculated in (i);

(b) if $\pi_i > \pi_j$, then the ROM size of π_j is calculated by (3) and the ROM size of π_i has been calculated in (i);

(c) if for any pair $\{\pi_i,\pi_j\}$, $R_{i,j} > R_i$ or $R_{j,i} > R_j$, then replace $R_{i,j}$ by R_i or replace $R_{j,i}$ by R_j.

(iii) (a) If for any nonredundant triple, $\{\pi_i,\pi_j,\pi_k\}$, $\pi_j \cdot \pi_k \nleq \pi_i$ or $\pi_j \cdot \pi_k = \pi_\ell$, $\ell = j$ or k, then $R_{i,j,k} = \min(R_{i,\ell})$, $\ell = j,k$,

(b) if $\pi_j \cdot \pi_k = \pi_m > \pi_i$, then

$$R_{i,j,k} = R_{i,m}.$$

(iv) Step (iii) is extended to handle nonredundant sets of $4,\ldots,n$ partitions;

(v) When no further nonredundant sets can be derived, sum the ROM sizes of the partitions in each set to obtain a measure of the variable dependence of each set.

The algorithm is used in the following example to evaluate possible decompositions.

Example 4.7

	i_0	i_1	i_2
1	5	3	2
2	6	3	1
3	5	1	1
4	5	6	2
5	5	4	2
6	5	4	1

M

S.P. partitions for M

$\pi_1 = \{\overline{1,2}; \overline{3}; \overline{4}; \overline{5,6}\}$

$\pi_2 = \{\overline{1,2,3}; \overline{4}; \overline{5,6}\}$

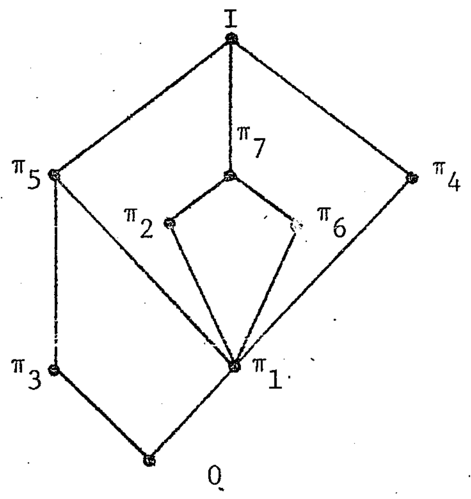
$\pi_3 = \{\overline{1,4}; \overline{2}; \overline{3,6}; \overline{5}\}$

$\pi_4 = \{\overline{1,2,5,6}; \overline{3,4}\}$

$\pi_5 = \{\overline{1,2,4}; \overline{3,5,6}\}$

$\pi_6 = \{\overline{1,2}; \overline{3}; \overline{4,5,6}\}$

$\pi_7 = \{\overline{1,2,3}; \overline{4,5,6}\}$



S.P. lattice

The ROM sizes for the individual partitions are $R_1 = 32$, $R_2 = 32$, $R_3 = 32$, $R_4 = 8$, $R_5 = 8$, $R_6 = 32$, and $R_7 = 8$.

Calculation of ROM size for a nonredundant triple is given in detail for $\{\pi_1,\pi_2,\pi_7\}$. For the pairs $\{\pi_1,\pi_2\}$, $\{\pi_1,\pi_7\}$, and $\{\pi_2,\pi_7\}$ the ROM sizes are

$$(R_{1,2},\ R_{2,1}) = (32,32)$$
$$(R_{1,7},\ R_{7,1}) = (16,8)$$
$$(R_{2,7},\ R_{7,2}) = (16,8)$$

Calculate the ROM size of each partition in the triple.

Since $\pi_1 \cdot \pi_7 = \pi_1 \nleq \pi_2$, $R_{2,1,7} = \min\ (R_{2,\ell})$, $\ell = 1,7$

$\therefore \quad R_{2,1,7} = 16.$

Similarly, $R_{7,1,2} = 8$

$\pi_2 \cdot \pi_7 = \pi_2$ and $R_{1,2,7} = R_{1,7}$. That is, $R_{1,2,7} = 16$.

The size of the triple is

$$(R_{1,2,7}, R_{2,1,7}, R_{7,1,2}) = (16,16,8)$$

Applying the evaluation technique to all the nonredundant sets in M gives:

(32,32)	$\pi_1\pi_2 = \pi_1$	(16,16,8)	$\pi_1\pi_2\pi_7 = \pi_1$
(32,32)	$\pi_1\pi_3 = 0$	(16,16,8)	$\pi_1\pi_3\pi_5 = 0$
(16,8)	$\pi_1\pi_4 = \pi_1$		
(16,8)	$\pi_1\pi_5 = \pi_1$		
(32,32)	$\pi_1\pi_6 = \pi_1$	(16,16,8)	$\pi_1\pi_6\pi_7 = \pi_1$
(16,8)	$\pi_1\pi_7 = \pi_1$		
(32,32)	$\pi_2\pi_3 = 0$	(32,16,8)	$\pi_2\pi_3\pi_5 = 0$
(32,8)	$\pi_2\pi_4 = \pi_1$		
(32,8)	$\pi_2\pi_5 = \pi_1$		
(32,32)	$\pi_2\pi_6 = \pi_1$	(16,16,8)	$\pi_2\pi_6\pi_7 = \pi$
(16,8)	$\pi_2\pi_7 = \pi_2$		
(32,8)	$\pi_3\pi_4 = 0$	(16,8,8)	$\pi_3\pi_4\pi_5 = 0$
(16,8)	$\pi_3\pi_5 = \pi_3$	(16,8,32)	$\pi_3\pi_5\pi_6 = 0$
(32,32)	$\pi_3\pi_6 = 0$	(16,8,8)	$\pi_3\pi_5\pi_7 = 0$
(32,8)	$\pi_3\pi_7 = 0$		
(8,8)	$\pi_4\pi_5 = \pi_1$		
(8,32)	$\pi_4\pi_6 = \pi_1$		
(8,8)	$\pi_4\pi_7 = \pi_1$		
(8,32)	$\pi_5\pi_6 = \pi_1$		
(8,8)	$\pi_5\pi_7 = \pi_1$		
(16,8)	$\pi_6\pi_7 = \pi_6$		

As can be readily seen from inspection of the ROM sizes, the sets $\{\pi_3,\pi_4,\pi_5\}$ and $\{\pi_3,\pi_5,\pi_7\}$ are the smallest decompositions at 32 bits each.

The necessity for step iii(b) is illustrated in the following example.

Example 4.8 For a machine M with the S.P. partitions

$$\pi_1 = \{\overline{1,2,3,4,5,6,7,8};\ \overline{9,10,11,12,13,14,15,16}\},$$

$$\pi_2 = \{\overline{1,3,5,7,9,11,13,15};\ \overline{2,4,6,8,10,12,14,16}\},$$

$$\pi_3 = \{\overline{1,3,5,7};\ \overline{2,4,6,8};\ \overline{9,11,13,15};\ \overline{10,12,14,16}\}, \text{ and}$$

$$\pi_4 = \{\overline{1,5};\ \overline{2,6};\ \overline{3,7};\ \overline{4,8};\ \overline{9,13};\ \overline{10,14};\ \overline{11,15};\ \overline{12,16}\}$$

calculate the ROM size of the triple $\{\pi_1,\pi_2,\pi_4\}$.

The ROM sizes for the pairs making up the triple are

$$(R_{1,2}, R_{2,1}) = (16,16)$$

$$(R_{1,4}, R_{4,1}) = (16,128)$$

$$(R_{2,4}, R_{4,2}) = (16,128)$$

If the ROM size $R_{4,1,2}$ was chosen by selecting the smallest ROM size $R_{4,1}$ and $R_{4,2}$, then $R_{4,1,2} = 128$. However, $\pi_1 \cdot \pi_2 = \pi_3 > \pi_4$. In this case information from π_3 can be used to realize π_4. Thus, using step iii(b),

$$R_{4,1,2} = R_{4,3} = 64 \text{ bits.}$$

In the previous section it was shown that some pairs of S.P. partitions may be uneconomical in comparison with other pairs. A method was presented to determine the uneconomic pairs, which could then be deleted prior to deriving the nonredundant sets. As demonstrated below, ROM evaluation is also able to detect uneconomic pairs.

Example 4.9

	i_0	i_1
1	8	2
2	6	1
3	7	4
4	5	3
5	1	3
6	3	4
7	2	1
8	4	2

M

$\pi_1 = \{\overline{1,2,3,4};\ \overline{5,6,7,8};$

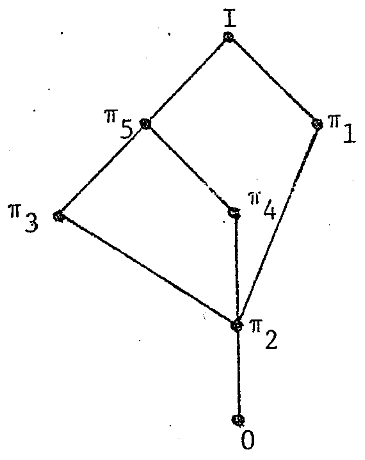
$\pi_2 = \{\overline{1,4};\ \overline{2,3};\ \overline{5,8};\ \overline{6,7}\}$

$\pi_3 = \{\overline{1,4,5,8};\ \overline{2,3};\ \overline{6,7,8}\}$

$\pi_4 = \{\overline{1,4};\ \overline{2,3,6,7};\ \overline{5,8}\}$

$\pi_5 = \{\overline{1,4,5,8};\ \overline{2,3,6,7}\}$

S.P. partitions



S.P. lattice

The uneconomic pairs for M are $\{\pi_2,\pi_3\}$, $\{\pi_2,\pi_4\}$, $\{\pi_3,\pi_5\}$, and $\{\pi_4,\pi_5\}$. Deleting the uneconomic pairs and deriving the nonredundant sets gives:

(4,8)	$\pi_1\pi_2 = \pi_2$
(4,16)	$\pi_1\pi_3 = \pi_2$
(4,16)	$\pi_1\pi_4 = \pi_2$
(4,4)	$\pi_1\pi_5 = \pi_2$
(8,4)	$\pi_2\pi_5 = \pi_2$
(16,16)	$\pi_3\pi_4 = \pi_2$

No nonredundant triples can be derived using the above pairs. From the ROM size, the best decompositions are $\{\pi_1,\pi_5\}$, $\{\pi_1,\pi_2\}$, and $\{\pi_2,\pi_5\}$.

Alternatively, let us derive the nonredundant sets without first deleting the uneconomic pairs.

(4,8)	$\pi_1\pi_2$		
(4,16)	$\pi_1\pi_3$		
(4,16)	$\pi_1\pi_4$		
(4,4)	$\pi_1\pi_5$		
(16,16)	$\pi_2\pi_3$	(16,8,4)	$\pi_2\pi_3\pi_5 = \pi_2$
(16,16)	$\pi_2\pi_4$	(16,8,4)	$\pi_2\pi_4\pi_5 = \pi_2$
(8,4)	$\pi_2\pi_5$		
(16,16)	$\pi_3\pi_4$	(8,8,4)	$\pi_3\pi_4\pi_5 = \pi_2$
(8,4)	$\pi_3\pi_5$		
(8,4)	$\pi_4\pi_5$		

In this case nonredundant triples can be derived. However, the product of each triple is only π_2. Since, the product of the pair $\{\pi_1,\pi_5\}$ is also π_2 and the ROM size of $\{\pi_1,\pi_5\}$ is smaller than any other nonredundant set, $\{\pi_1,\pi_5\}$ is again the best decomposition.

The uneconomic pairs are not specifically indicated by the ROM evaluation technique. Their detrimental effect, however, appears in subsequent decompositions. Further application of ROM evaluation are considered in the next section.

4.5 Subtle Redundancy

Hartmanis and Stearns [31] have provided examples of non-redundant decompositions which contain what they call subtle forms of "redundancy". Even though superfluous components have been eliminated from a nonredundant set, it is still possible that a partition could be replaced or supplemented to produce a more economical decomposition.

The more subtle forms of redundancy that Hartmanis and Stearns describe can be characterized as having an excessive number of state variables. This in turn indicates an excessive amount of variable dependence.

In this section we name the different types of subtle redundancy and prove that ROM evaluation is able to detect them. In this regard our discussion is limited to partitions π_i, where $\pi_i > \pi_j$ implies $e(\pi_j) = e(\pi_i) + e(\pi_i|\pi_j)$.

It is possible to characterize other types of subtle redundancy. As ROM evaluation measures variable dependence, however, it is able to detect the new types of subtle redundancy.

<u>Factor Redundancy</u>

The first form of subtle redundancy considered by Hartmanis and Stearns, concerns a decomposition where the computations of two machines overlap. In this case, part of the computations of both machines can be factored out and computed in a separate machine. A machine which exhibits factor redundancy is given in the following example.

<u>Example 4.9</u>

	i_0	i_1	i_2	i_3	i_4
1	2	1	5	8	3
2	1	2	6	7	4
3	4	3	6	6	1
4	3	4	5	5	2
5	5	6	3	4	7
6	6	5	4	3	8
7	7	8	4	2	5
8	8	7	3	1	6

J

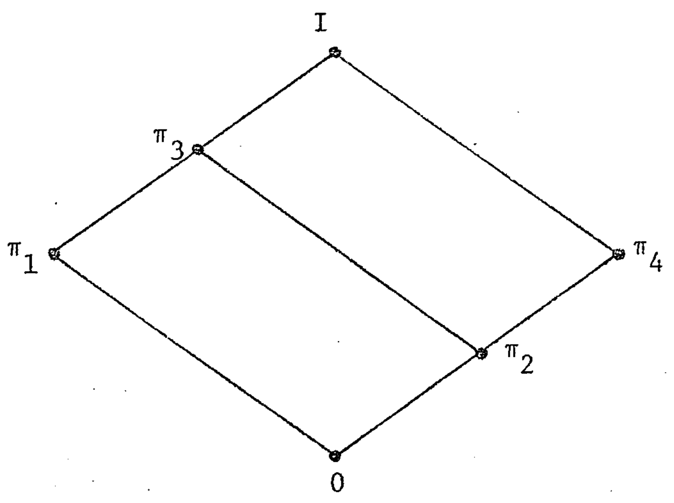
S.P. partitions for J

$\pi_1 = \{\overline{1,4};\ \overline{2,3};\ \overline{5,8};\ \overline{6,7}\}$

$\pi_2 = \{\overline{1,2};\ \overline{3,4};\ \overline{5,6};\ \overline{7,8}\}$

$\pi_3 = \{\overline{1,2,3,4};\ \overline{5,6,7,8}\}$

$\pi_4 = \{\overline{1,2,7,8};\ \overline{3,4,5,6}\}$



S.P. lattice

Two nonredundant decompositions for machine J are $\{\pi_1,\pi_2\}$ and $\{\pi_1,\pi_2,\pi_3\}$.

Since, $\pi_3 = \pi_1 + \pi_2$, π_3 performs a subfunction of both π_1 and π_2. Factoring out π_3 and computing it separately produces a reduction in variable dependency.

The following lemma proves that factor redundancy can be detected by ROM evaluation.

Lemma 4.1 Let $X = \{\pi_i,\pi_j\}$ and $Y = \{\pi_i,\pi_j,\pi_k\}$ be nonredundant sets of S.P. partitions, such that $\pi_k = \pi_i + \pi_j$. Then

$$\pi_i \cdot \pi_j = \pi_i \cdot \pi_j \cdot \pi_k = \pi \quad \text{and}$$

$R_x \gtrless R_y$, where R_x and R_y are the ROM sizes required to realize π using the nonredundant sets X and Y, respectively.

Proof

$$R_x = \sum_{\ell=i,j} e(\pi_\ell)\cdot 2^{N(IP)+e(\pi_\ell)}$$

$$= 2^{N(IP)} \sum_{\ell=i,j} \left[e(\pi_k)+e(\pi_k|\pi_\ell)\right]\cdot 2^{e(\pi_k)+e(\pi_k|\pi_\ell)}$$

$$= e(\pi_k)\cdot 2^{N(IP)+e(\pi_k)} \sum_{\ell=i,j} 2^{e(\pi_k|\pi_\ell)} + 2^{N(IP)+e(\pi_k)} \sum_{\ell=i,j} e(\pi_k|\pi_\ell)\cdot 2^{e(\pi_k|\pi_\ell)}$$

$$R_y = e(\pi_k)\cdot 2^{N(IP)+e(\pi_k)} + \sum_{\ell=i,j} e(\pi_k|\pi_\ell)\cdot 2^{N(IP)+e(\pi_k)+e(\pi_k|\pi_\ell)}$$

$$= e(\pi_k)\cdot 2^{N(IP)+e(\pi_k)} + 2^{N(IP)+e(\pi_k)} \sum_{\ell=i,j} e(\pi_k|\pi_\ell)\cdot 2^{e(\pi_k|\pi_\ell)}$$

$$\therefore R_x - R_y = e(\pi_k)\cdot 2^{N(IP)+e(\pi_k)} \left[\sum_{\ell=i,j} 2^{e(\pi_k|\pi_\ell)} - 1\right]$$

since $\sum_{\ell=i,j} 2^{(\pi_k|\pi_\ell)} > 1$, $R_x > R_y$.

Therefore, the reduction in variable dependence that results from factoring out a common machine is indicated by the ROM size calculations.

For machine J of Example 4.9, the decomposition $X = \{\pi_1,\pi_2\}$ requires a ROM size of

$$R_x = 2.2^{3+2} + 2.2^{3+2}$$
$$= 128 \text{ bits}$$

for the decomposition $Y = \{\pi_1,\pi_2,\pi_3\}$,

$$R_y = 1.2^{3+1} + 1.2^{3+1+1} + 1.2^{3+1+1}$$
$$= 80 \text{ bits}$$

$$R_x - R_y = 48 \text{ bits.}$$

Calculating $R_x - R_y$ using the result of Lemma 4.1, with $\pi_k = \pi_3$, gives

$$R_x - R_y = e(\pi_k)\cdot 2^{N(IP)+e(\pi_k)}\left[\sum_{\ell=i,j} 2^{e(\pi_k|\pi_\ell)} - 1\right]$$

$$= 1\cdot 2^{3+1}\left[2^{e(\pi_3|\pi_1)} + 2^{e(\pi_3|\pi_2)} - 1\right]$$

$$= 16[2 + 2 - 1]$$

$$= 48 \text{ bits.}$$

Submachine Redundancy

Another form of subtle redendancy is submachine redundancy. This redundancy is typified by the use of a machine in decomposition, when a submachine is cheaper and just as effective. Submachine redundancy is demonstrated by the following example.

Example 4.10

	i_0	i_1	i_2	i_3
1	1	5	7	4
2	2	6	8	3
3	1	7	5	2
4	2	8	6	1
5	2	1	2	8
6	2	2	1	7
7	1	3	4	6
8	1	4	3	5

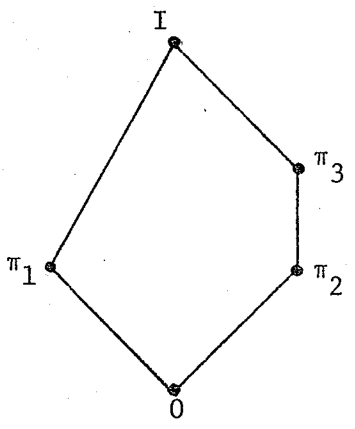
K

$$\pi_1 = \{\overline{1,8};\ \overline{2,6};\ \overline{3,7};\ \overline{4,5}\}$$

$$\pi_2 = \{\overline{1,2};\ \overline{3,4};\ \overline{5,6};\ \overline{7,8}\}$$

$$\pi_3 = \{\overline{1,2,3,4};\ \overline{5,6,7,8}\}$$

S.P. partitions for K



S.P. lattice

Two nonredundant decompositions for machine K are $X = \{\pi_1, \pi_2\}$ and $Y = \{\pi_1, \pi_3\}$.

Since $\pi_1 \cdot \pi_2 = \pi_1 \cdot \pi_3 = 0$, either X or Y realizes machine K. However, π_3 is a submachine of π_2. Consequently, it is more economical to realize machine K using Y. Lemma 4.2 proves that submachine redundancy is detected by the ROM size formula.

We will use the notation $\pi_i||\pi_j$ to indicate that $\pi_i \nmid \pi_j$ and $\pi_j \nmid \pi_i$.

<u>Lemma 4.2</u> If $X = \{\pi_i,\pi_j\}$ and $Y = \{\pi_i,\pi_k\}$ are nonredundant sets of S.P. partitions such that $\pi_k > \pi_j$, $\pi_i||\pi_j$, $\pi_i||\pi_k$, and

$$\pi_i \cdot \pi_j = \pi_i \cdot \pi_k = \pi, \text{ then } R_x > R_y.$$

Proof.

$$R_x = e(\pi_i)\cdot 2^{N(IP)+e(\pi_i)} + e(\pi_j)\cdot 2^{N(IP)+e(\pi_j)}$$

$$R_y = e(\pi_i)\cdot 2^{N(IP)+e(\pi_i)} + e(\pi_k)\cdot 2^{N(IP)+e(\pi_k)}$$

$$R_x-R_y = 2^{N(IP)}\left[e(\pi_j)\cdot 2^{e(\pi_j)} - e(\pi_k)\cdot 2^{e(\pi_k)}\right]$$

$$= 2^{N(IP)}\left([e(\pi_k) + e(\pi_k|\pi_j)]\cdot 2^{e(\pi_k)+e(\pi_k|\pi_j)} - e(\pi_k)\cdot 2^{e(\pi_k)}\right)$$

$$= 2^{N(IP)}\left[e(\pi_k|\pi_j)\cdot 2^{e(\pi_k)+e(\pi_k|\pi_j)} + e(\pi_k)\cdot 2^{e(\pi_k)}\left(2^{e(\pi_k|\pi_j)}\right) - 1\right]$$

Since $\pi_k > \pi_j$, $e(\pi_k|\pi_j) > 1$ and $2^{e(\pi_k|\pi_j)} > 1$

Therefore, $R_x > R_y$.

For machine K in Example 4.10,

$$R_x = 2\cdot 2^{2+2} + 2\cdot 2^{2+2}$$
$$= 64 \text{ bits}$$

$$R_y = 2.2^{2+2} + 1\cdot 2^{2+1}$$
$$= 40 \text{ bits}$$

$$R_x-R_y = 24 \text{ bits.}$$

Calculating R_x-R_y using the formula derived in Lemma 4.2, with $\pi_j=\pi_2$ and $\pi_k=\pi_3$, we obtain

$$R_x-R_y = 2^{N(IP)}\left[e(\pi_3|\pi_2)\cdot 2^{e(\pi_3)+e(\pi_3|\pi_2)} + e(\pi_3)\cdot 2^{e(\pi_3)}\left(2^{e(\pi_3|\pi_2)} - 1\right)\right]$$

$$= 2^2[1\cdot 2^{1+1} + 1\cdot 2\cdot(2 - 1)]$$
$$= 4[4 + 2]$$
$$= 24 \text{ bits.}$$

<u>Alternate Machine Redundancy</u>

The third type of subtle redundancy, alternate machine redundancy, occurs when an S.P. partition, π_j, in a nonredundant decomposition can be replaced by another S.P. partition, π_k, which requires fewer state variables. This redundancy is similar to submachine redundancy. However, for alternate machine redundancy, $\pi_j || \pi_k$.

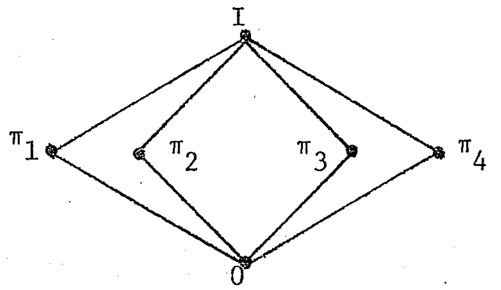
<u>Example 4.11</u>

	i_0	i_1
1	5	2
2	6	1
3	1	6
4	2	5
5	3	4
6	4	3

L

$$\pi_1 = \{\overline{1,2};\ \overline{3,4};\ \overline{5,6}\}$$
$$\pi_2 = \{\overline{1,4};\ \overline{2,5};\ \overline{3,6}\}$$
$$\pi_3 = \{\overline{1,6};\ \overline{2,3};\ \overline{4,5}\}$$
$$\pi_4 = \{\overline{1,3,5};\ \overline{2,4,6}\}$$

S.P. partitions for L



S.P. lattice

Let $X = \{\pi_1,\pi_2\}$ and $Y = \{\pi_1,\pi_4\}$

The nonredundant sets X and Y both realize machine L. However, to realize π_4 requires 1 state variable, while π_2 requires 2 state variables.

Lemma 4.3 If $X = \{\pi_i, \pi_j\}$ and $Y = \{\pi_i, \pi_k\}$ are nonredundant sets of S.P. partitions, such that $\pi_j || \pi_k$, $e(\pi_k) < e(\pi_j)$, $\pi_i || \pi_j$, $\pi_i || \pi_k$, and

$$\pi_i \cdot \pi_j = \pi_i \cdot \pi_k = \pi, \quad \text{then} \quad R_x > R_y.$$

Proof

$$R_x = e(\pi_i) \cdot 2^{N(IP)+e(\pi_i)} + e(\pi_j) \cdot 2^{N(IP)+e(\pi_j)}$$

$$R_y = e(\pi_i) \cdot 2^{N(IP)+e(\pi_i)} + e(\pi_k) \cdot 2^{N(IP)+e(\pi_k)}$$

$$R_x - R_y = 2^{N(IP)}\left[e(\pi_j) \cdot 2^{e(\pi_j)} - e(\pi_k) \cdot 2^{e(\pi_k)}\right]$$

Since $e(\pi_j) > e(\pi_k)$,

$$e(\pi_j) \cdot 2^{e(\pi_j)} > e(\pi_k) \cdot 2^{e(\pi_k)} \quad \text{and} \quad R_x > R_y .$$

For machine L

$$R_x = 2 \cdot 2^{1+2} + 2.2^{1+2}$$
$$= 32 \text{ bits}$$

$$R_y = 2.2^{1+2} + 1.2^{1+1}$$
$$= 20 \text{ bits}$$

$$R_x - R_y = 12 \text{ bits.}$$

Using the formula from Lemma 4.3, with $\pi_k = \pi_4$ and $\pi_j = \pi_2$,

$$R_x - R_y = 2^{N(IP)}\left[e(\pi_2) \cdot 2^{e(\pi_2)}\right] - e(\pi_4) \cdot 2^{e(\pi_4)}$$
$$= 2[2.2^2 - 1 \cdot 2]$$
$$= 12 \text{ bits.}$$

Hartmanis and Stearns have only identified the three types of redundancy just described. It is possible to define other types. For example, uneconomic pairs could be classified as a form of subtle redundancy.

In the following example we define a new type of redundancy and demonstrate that ROM evaluation is able to detect it.

Example 4.12

In Example 4.9, one possible decomposition is $X = \{\pi_1,\pi_2,\pi_3\}$, where $\pi_1\cdot\pi_2\cdot\pi_3 = 0$. There is another decomposition, $Y = \{\pi_1\ \pi_3\ \pi_4\}$ for which $\pi_1\cdot\pi_3\cdot\pi_4 = 0$. The partition π_4 is a submachine of π_2. Subsequently, the set Y should provide a more economical decomposition than the set X. Actually, this redundancy is submachine redundancy applied to triples rather than pairs.

$$\begin{aligned} R_x &= R_1 + R_2 + R_3 \\ &= 1\cdot 2^{3+1+1} + 1\cdot 2^{3+1+1} + 1\cdot 2^{3+1} \\ &= 80 \text{ bits.} \end{aligned}$$

$$\begin{aligned} R_y &= R_1 + R_3 + R_4 \\ &= 1\cdot 2^{3+1+1} + 1\cdot 2^{3+1} + 1\cdot 2^{3+1} \\ &= 64 \text{ bits.} \end{aligned}$$

Thus,ROM evaluation is able to indicate the subtle redundancy in decomposition X.

4.6 Discussion

In this chapter two evaluation methods for machine decomposition have been presented. The first method, uneconomic pairs, has not been developed as fully as possible. It is possible to extend the uneconomic pair concept to include subtle redundancy. This is demonstrated for submachine redundancy where $\pi_i\cdot\pi_j = \pi_i\cdot\pi_k$ and $\pi_k > \pi_j$. Clearly, the pair $\{\pi_i,\pi_j\}$ is an uneconomic pair with respect to the pair $\{\pi_i,\pi_k\}$. Similarly, the other types of subtle redundancy define uneconomic pairs, which can then be deleted prior to generating the nonredundant sets. Taking this approach ensures that only the most economical nonredundant sets will be derived by the algorithm of Chapter 3.

However, reducing the number of possible decompositions is not always desirable. For example, a logic designer may have available a component he wishes to use in building a machine. By reducing the number of decompositions, the possibility of using that component is also reduced. Generating all the decompositions gives the designer greater freedom as to how he will build the machine.

ROM evaluation permits all possible decompositions to be generated while providing a measure of the variable dependence of each decomposition. To further demonstrate the effectiveness of ROM evaluation, a programmable logic array (PLA) implementation [75],[76],[77], of machine E in Example 4.5 is considered.

PLAs operate in a manner similar to ROMs with the difference being that the truth table is not specified in its entirity. Rather, the product terms from the sum of products equations are entered into the PLA along with their corresponding output functions. Since the number of product terms for a set of equations is considerably smaller than the number of truth table entries, a PLA requires fewer memory elements than an equivalent ROM.

Example 4.13 The next-state equations for the random state assignment in Example 4.5 require 10 product terms.

$$Y_1 = \bar{y}_1\bar{y}_2y_3\bar{x} + y_2\bar{y}_3\bar{x} + y_2y_3x$$

$$Y_2 = \bar{y}_2x + \bar{y}_1\bar{y}_2\bar{y}_3 + y_1y_3$$

$$Y_3 = y_1x + y_2\bar{y}_3x + y_2y_3\bar{x} + \bar{y}_1\bar{y}_2\bar{x}$$

Thus, a PLA implementation would require a PLA with at least 4 inputs and 10 words (for the product terms). Each word requires 3 output bits. The next-state equations derived from the decomposition assignment has only 4 product terms.

$$Y_1 = y_1x + \bar{y}_1\bar{x}$$

$$Y_2 = y_3\bar{x}$$

$$Y_3 = \bar{y}_2\bar{y}_3$$

A PLA with 4 inputs and 4 words with 3 outputs each would suffice to realize the above set of equations.

Evaluating decompositions in terms of PLA size requires the derivation of the next-state equations. Thus, as for diode counting, a Boolean simplification step is necessary before an evaluation can be performed.

From the above discussion and the examples presented throughout this chapter, it is apparent that ROM evaluation is a practical and useful evaluation technique for machine decomposition.

Chapter 5 Extended Substitution Property Theory

5.1 Introduction

As indicated in Chapter 1, S.P. partition theory has been extended to other areas of sequential machine theory. In this chapter we examine some of these extensions, multiple machine synthesis and asynchronous machine decomposition, and further extend these areas of research. In addition, the decomposition of incompletely specified machines is investigated. This latter topic is of great importance to sequential machine decomposition, as the majority of practical machines are incompletely specified. Also, because of the "don't-care" conditions, it would appear that there is a greater possibility of S.P. partitions existing for incompletely specified machines than for completely specified machines. Thus, in order to facilitate the use of decomposition theory, it is important that techniques of handling incompletely specified machines be developed.

Hartmanis and Stearns [31] have extended the partition pair concept to incompletely specified machines. However, they have not examined the problem of defining S.P. partitions for incompletely specified machines. Some of the theory they develop for weak and extended partition pairs can be adapted to derive a definition of an S.P. partition for an incompletely specified machine. Two types of S.P. partitions are considered for incompletely specified machines, weak substitution property and extended substitution property partitions.

Substitution property partition theory, however, does not readily apply to the resulting definitions. In particular, Hartmanis and Stearns' method for deriving S.P. partitions and S.P. lattices is not applicable. In the next section, theory is developed which enables the derivation of S.P. partitions and lattices for incompletely specified machines. (The results are also applicable to completely specified machines.)

The two basic machine connections, serial and parallel, can be used for constructing networks of incompletely specified machines. However, the derivation of decompositions for incompletely specified machines is not as straightforward as for completely specified machines. The problem arises because the sum of two weak (extended) substitution property partitions is not necessarily a weak (extended) substitution partition. Consequently, for two weak (extended) substitution property partitions whose sum is not I, it is not always possible to factor out a common submachine.

Hartmanis [27] has introduced a decomposition which can be used to reduce the state variable dependence for a serial composition. In Section 5.4, we show that this decomposition can be adapted to economically realize two weak substitution property partitions whose sum is not a weak substitution property partition. The adapted decomposition combines features of both parallel and serial decompositions and in terms of ROM evaluation is midway between the two.

The problem of realizing sequential machines is often not confined to simply the problem of realizing one machine. In many cases, the logic designer will have a collection of machines that are to be realized together. Kohavi and Smith [44], [62] and Kohavi [43] have examined the problem of realizing two or more machines in combination. For some machines, the joint synthesis is more economical than two separate syntheses. The approach developed involves finding common component machines for the two machines, thus reducing duplication.

Kohavi and Smith have presented an intuitive approach to this problem. In Section 5.5, we review the work of Kohavi and Smith and present a theoretical basis for their work. The theory developed also provides the base for an alternate method of multiple machine decomposition, which utilizes the structure of the component machines.

Tan, Menon, and Friedman [65] have investigated the problem of extending decomposition and reduced dependence theory to asynchronous machines. In their work on the decomposition of asynchronous machines with S.P. partitions, they have concentrated on finding single transition time assignments, as defined by Tracey [67]. In determining whether Tracey's conditions can be met by an S.P. partition, Tan et al use the usual inspection method. Properties relating S.P. partitions to Tracey's conditions are presented in Section 5.6. Use of the properties developed simplifies the determination of whether or not Tracey's conditions are satisfied.

5.2 Weak Substitution Property

In this section, substitution property theory is applied to incompletely specified machines using weak substitution property partitions. The definition of weak substitution property partitions presented parallels that of Hartmanis and Stearns for weak partition pairs (Definition 2.3).

Definition 5.1 If $M = (S, IP, OP, \delta, \lambda)$ is a machine with don't-care conditions, a partition π on the set of states S is said to have the weak substitution property, if and only if $s \equiv t(\pi)$ implies that $\delta(s,i) \equiv \delta(t,i)(\pi)$ for all $i \in IP$, whenever $\delta(s,i)$ and $\delta(t,i)$ are both specified.

Example 5.1 Consider the incompletely specified machine A, designated by the table

	i_0	i_1	i_2
1	-	1	3
2	5	2	3
3	1	2	-
4	3	5	4
5	-	1	2

A

$\pi = \{\overline{1,5}; \overline{2,3}; \overline{4}\}$ is a weak substitution property partition for the incompletely specified machine A.

Unfortunately, the weak substitution property partitions for an incompletely specified machine cannot be derived by the same method used for deriving S.P. partitions. This is because the corollary, which Hartmanis and Stearns use as a basis for generating S.P. partitions, does not apply to incompletely specified machines. The corollary (presented as Theorem 2.5) states that for any partition x, there exists an integer K such that the smallest S.P. partition y, $y \geq x$, can be found by calculating $y = y_k = \sum_{i=0}^{k} m^i(x)$, where $k \geq K$.

Applying this corollary to the incompletely specified machine A of Example 5.1, we are unable to derive the weak substitution property partitions. This is illustrated by the following example.

Example 5.2 Let $x = \{\overline{1,3};\ \overline{2};\ \overline{4};\ \overline{5}\} = m^0(x)$.

Then $m^1(x) = \{\overline{1,2};\ \overline{3};\ \overline{4};\ \overline{5}\}$ and $y_1 = m^1(x) + m^0(x) = \{\overline{1,2,3};\ \overline{4};\ \overline{5}\}$.

$$\begin{aligned} m^2(x) &= m(m^1(x)) \\ &= \{\overline{1,2};\ \overline{3};\ \overline{4};\ \overline{5}\} \\ y_2 &= \{\overline{1,2,3};\ \overline{4};\ \overline{5}\} \end{aligned}$$

Since $y_1 = y_2$, $y_1 = \{\overline{1,2,3};\ \overline{4};\ \overline{5}\}$ should have the weak substitution property. However, $m(y_1) = \{\overline{1,2,5};\ \overline{3};\ \overline{4}\}$. Thus $y_1 = \{\overline{1,2,3};\ \overline{4};\ \overline{5}\}$ is not a weak substitution property partition on A.

Actually, $\{\overline{1,2,3,5};\ \overline{4}\}$ is the smallest partition greater than $x = \{\overline{1,3};\ \overline{2};\ \overline{4};\ \overline{5}\}$, which has the weak substitution property.

Thus, Hartmanis and Stearns' method is not capable of deriving the weak substitution property partitions for an incompletely specified machine. A new method, which can derive both S.P. and weak substitution property partitions, is presented. The basis for the method is provided by Theorem 5.1. To prove the theorem we use the following notation.

Notation For a completely or incompletely specified machine M with a set of states S and a partition x on S, we write $y_0 = x$ and $y_j = y_{j-1} + m(y_{j-1})$, for all $j \geq 1$.

Theorem 5.1 For an incompletely specified machine M, with a set of state S there exists an integer K such that for all $k \geq K$ and for all partitions x on S, $\min\{y \mid y \geq x$ and y has the weak substitution property$\} = y_k$.

Proof Since $y_{j+1} = m(y_j) + y_j$, $y_{j+1} \geq y_j$. However, as the set of states S is finite, there exists some k such that $y_{k+1} = y_k$.

Therefore, $m(y_k) \leq y_{k+1} = y_k$ and y_k is a weak substitution property partition. To show that y_k is the minimum, let y' be a weak substitution property partition such that $y' \geq x$ and $y' \geq m(y')$. Thus, we must show $y' \geq y_j$ for any value of j.

The proof is by mathematical induction on j. By definition $y' \geq x = y_0$, which establishes the basis for the induction.

Assume for some positive integral value n that $y' \geq y_n$ and $y' \geq m(y') \geq m(y_n)$.

Therefore, $y' \geq m(y_n) + y_n$ and $y' \geq y_{n+1}$. Thus, $y' \geq y_j$ for any value of j and $y' \geq y_k$. It can be shown that there exists a K such that y_K is the minimum weak substitution property partition greater than or equal to x, for all x.

The following corollary proves that Theorem 5.1 applies to completely specified machines as expected.

Corollary 5.1.1 For a completely specified machine $y_j = \sum_{i=0}^{j} m^i(x)$.

Proof. The proof is by mathematical induction. The basis is established by showing that $y_j = \sum_{i=0}^{j} m^i(x)$, for j=1.

$$\begin{aligned} y_j &= m(y_{j-1}) + y_{j-1} \\ y_1 &= m(y_0) + y_0 \\ &= m(x) + x \\ &= \sum_{i=0}^{1} m^i(x) \end{aligned}$$

Assume true for j = n. That is

$$y_n = \sum_{i=0}^{n} m^i(x).$$

It must be proven that for j = n+1

$$y_{n+1} = \sum_{i=0}^{n+1} m^i(x).$$

$$\begin{aligned} y_{n+1} &= m(y_n) + y_n \\ &= m\left(\sum_{i=0}^{n} m^i(x)\right) + \sum_{i=0}^{n} m^i(x) \\ &= \sum_{i=1}^{n+1} m^i(x) + \sum_{i=0}^{n} m^i(x) \end{aligned}$$

$$\therefore \quad y_{n+1} = \sum_{i=0}^{n+1} m^i(x)$$

$$\text{Thus} \quad y_j = \sum_{i=0}^{j} m^i(x) \ .$$

The following corollary establishes that $\sum_{i=0}^{j} m^i(x)$ cannot be used to derive the weak substitution property partitions for an incompletely specified machine.

<u>Corollary 5.1.2</u> For an incompletely specified machine y_j is not necessarily equal to $\sum_{i=0}^{j} m^i(x)$.

<u>Proof</u> This corollary is proved by observing that the proof by mathematical induction in Corollary 5.1.1, does not apply to incompletely specified machines. In the proof of the conclusion of Corollary 5.1.1, the following rule is used; $m\left(\sum_{i=0}^{n} m^i(x)\right) = \sum_{i=1}^{n+1} m^i(x)$. This is true, since for a completely specified machine $m(x_1+x_2) = m(x_1) + m(x_2)$. However, for an incompletely specified machine, $m(x_1+x_2) \geq m(x_1) + m(x_2)$.
Thus, $y_{n+1} \geq \sum_{i=0}^{n+1} m^i(x)$ and $y_j \geq \sum_{i=0}^{j} m^i(x)$.

<u>Example 5.3</u> Theorem 5.1 will be used to find the weak substitution property partitions of a machine. Consider $x = \{\overline{1,3};\ \overline{2};\ \overline{4};\ \overline{5}\}$ of machine A in Example 5.1.

$$y_j = y_{j-1} + m(y_{j-1})$$

As $y_0 = x = \{\overline{1,3};\ \overline{2};\ \overline{4};\ \overline{5}\}$ and $m(y_0) = \{\overline{1,2};\ \overline{3};\ \overline{4};\ \overline{5}\}$ then
$y_1 = \{\overline{1,2,3};\ \overline{4};\ \overline{5}\}$

$m(y_1) = \{\overline{1,2,5};\ \overline{3};\ \overline{4}\}$ and $y_2 = \{\overline{1,2,3,5};\ \overline{4}\}$
$m(y_2) = \{\overline{1,2,3,5};\ \overline{4}\}$ and $y_3 = \{\overline{1,2,3,5};\ \overline{4}\}$

Since $y_2 = y_3$, $y_2 = \{\overline{1,2,3,5};\ \overline{4}\}$ is the smallest weak substitution property partition greater than x.

Note that in Example 5.1

$$\sum_{i=0}^{1} m^i(x) = \sum_{i=0}^{2} m^i(x) = \{\overline{1,2,3};\ \overline{4};\ \overline{5}\}.$$

Thus, $y_2 \geq \sum_{i=0}^{2} m^i(x)$, as predicted by Corollary 5.1.2.

In order to derive the weak substitution property lattice the least upper bounds and the greatest lower bounds for the partitions must be obtained. Lemma 5.1 follows from the work of Hartmanis and Stearns.

Lemma 5.1 If π_i and π_j are weak substitution property partitions on the set of states of an incompletely specified machine M, then the partition $\pi_i \cdot \pi_j$ is also a weak substitution property partition.

It can be shown by a counter-example, that for π_i and π_j weak substitution property partitions, $\pi_i + \pi_j$ is not necessarily a weak substitution property partition.

Example 5.4

	i_0	i_1	i_2	i_3
1	2	3	–	4
2	3	5	–	–
3	4	6	3	–
4	5	3	–	1
5	–	6	–	–
6	–	–	4	2

B

$\pi_1 = \{\overline{1};\ \overline{2,4};\ \overline{3,5};\ \overline{6}\}$ and $\pi_2 = \{\overline{1};\ \overline{2,6};\ \overline{3};\ \overline{4};\ \overline{5}\}$ are weak substitution property partitions on B.

However, $\pi_1 + \pi_2 = \{\overline{1};\ \overline{2,4,6};\ \overline{3,5}\}$ is not a weak substitution property partition. Thus, all the weak substitution property partitions cannot be found by simple partition addition.

Hartmanis and Stearns show that the weak partition pairs for a machine form a lattice with the bounds given by

$$\text{glb}\ ((\pi,\pi'),(\tau,\tau')) = (\pi\cdot\tau,\pi'\cdot\tau') \text{ and}$$
$$\text{lub}\ ((\pi,\pi'),(\tau,\tau')) = (\pi+\tau,\pi+\tau+m(\pi+\tau)).$$

For pair algebras the S.P. partitions, which are a subset of the partition pairs, form a sublattice of the lattice of partition pairs. Our definition of weak substitution property partitions ensures that they are a subset of the weak partition pairs. Thus, it might be expected that the weak substitution property partitions form a sublattice of the weak partition pairs. Unfortunately, this is not the case. A counter-example is given using the partitions π_1 and π_2 of Example 5.4.

<u>Example 5.5</u> (π_1,π_1) and (π_2,π_2) are weak partition pairs.

$$\begin{aligned}\text{lub}((\pi_1,\pi_1),(\pi_2,\pi_2)) &= (\pi_1+\pi_2,\pi_1+\pi_2+m(\pi_1+\pi_2))\\ &= (\{\overline{1};\ \overline{2,4,6};\ \overline{3,5}\},\{\overline{1,2,4,6};\ \overline{3,5}\})\end{aligned}$$

is a weak partition pair. However, neither $\{\overline{1},\ \overline{2,4,6};\ \overline{3,5}\})$ nor $\{\overline{1,2,4,6};\ \overline{3,5}\}$ is a weak substitution property partition.

Consequently, in order to form a lattice for the weak substitution property partitions a new least upper bound must be derived.

<u>Notation</u> For any partition τ, let

$Y(\tau) = y_k$, such that $y_k = y_{k+1}$, where $y_0 = \tau$ and $y_j = y_{j-1} + m(y_{j-1})$ for $j \geq 0$.

Lemma 5.2 follows directly from Theorem 5.1.

<u>Lemma 5.2</u> If π_i and π_j are weak substitution property partitions on the set of states of an incompletely specified machine M, then the partition $Y(\pi_i+\pi_j)$ is also a weak substitution property partition.

Thus the weak substitution property partitions for an incompletely specified machine form a lattice with the bounds given by

$$\text{glb}(\pi_1,\pi_2) = \pi_1\cdot\pi_2 \text{ and}$$
$$\text{lub}(\pi_1,\pi_2) = Y(\pi_1+\pi_2).$$

The use of Lemma 5.2 to derive weak substitution property partitions is demonstrated below.

Example 5.6

	i_0	i_1	i_2
1	1	-	-
2	-	4	5
3	3	5	6
4	2	5	3
5	1	6	2
6	-	6	-

C

$\pi_1 = \{\overline{1,2}; \overline{3}; \overline{4}; \overline{5}; \overline{6}\}$ and $\pi_2 = \{\overline{1,3}; \overline{2}; \overline{4}; \overline{5}; \overline{6}\}$ are weak substitution property partitions. Deriving the least upper bound of π_1 and π_2, $\text{lub}(\pi_1,\pi_2) = Y(\pi_1+\pi_2)$.

$$\begin{aligned} y_0 = \pi_1+\pi_2 \text{ and } m(y_0) &= m(\pi_1+\pi_2) \\ &= m(\{\overline{1,2,3}; \overline{4}; \overline{5}; \overline{6}\}) \\ &= \{\overline{1,2,3}; \overline{4,5,6}\} \end{aligned}$$

$$\begin{aligned} y_1 &= y_0 + m(y_0) \\ &= \{\overline{1,2,3}; \overline{4,5,6}\} \text{ and } m(y_1) = \{\overline{1,2,3}; \overline{4,5,6}\} \end{aligned}$$

Thus, $y_2 = y_1 + m(y_1) = \{\overline{1,2,3}; \overline{4,5,6}\}$

Since $y_2 = y_1$, $\text{lub}(\pi_1,\pi_2) = Y(\pi_1+\pi_2) = \{\overline{1,2,3}; \overline{4,5,6}\}$

The remaining weak substitution property partitions can be found using Theorem 5.1 and Lemma 5.2.

Once the weak substitution property partitions have been generated, economical decompositions can be derived using the methods of Chapters 3 and 4.

5.3 Extended Substitution Property

Following Hartmanis and Stearn's definition of extended partition pairs for incompletely specified machines, we give a definition for an extended substitution property partition. First an extended sequential machine is defined.

Definition 5.2 For an incompletely specified machine $M = (S, IP, OP, \delta, \lambda)$, an extended machine is the machine $M' = (S \cup C, IP, OP \cup D, \delta, \lambda)$, with each don't care state condition designated by a distinct element from the set C and each don't-care output condition designated by a distinct element from the set D.

Definition 5.3 For an extended machine M', a partition π on $S \cup C$ is an extended substitution property partition if and only if for all $s, t \in S$, $s \equiv t(\pi)$ implies $\delta(s,i) \equiv \delta(t,i)(\pi)$ for all $i \in IP$.

Example 5.6 An extended machine E' is derived for the incompletely specified machine E

	i_0	i_1	i_2	i_3	z
1	1	-	3	2	1
2	3	3	4	-	0
3	2	-	-	2	-
4	-	4	1	3	0

E

Let $E' = (S \cup C, IP, OP \cup D, \delta, \lambda)$, where $C = \{c_1, c_2, c_3, c_4, c_5\}$ and $D = \{d_1\}$

	i_0	i_1	i_2	i_3	z
1	1	c_1	3	2	1
2	3	3	4	c_2	0
3	2	c_3	c_4	2	d_1
4	c_5	4	1	3	0

E'

An extended substitution property partition for E' is

$\pi_1 = \{\overline{1}; \overline{2,3,c_2,c_3}; \overline{4,c_4}; \overline{c_1}; \overline{c_5}\}$

The weak substitution property partitions can be derived from the extended substitution property partitions.

Definition 5.4 A reduced partition, π', is an extended substitution property partition π, from which all elements of C have been deleted.

Lemma 5.3 A reduced partition π' on the extended machine M', is a weak substitution property partition on the incompletely specified machine.

Proof The proof is obvious from the definition of weak substitution property partitions and reduced partitions.

Thus, by finding all the extended substitution property partitions, we can also find all of the weak substitution property partitions.

It can be proven that the greatest lower bound of two extended substitution property partitions is obtained by partition multiplication. The least upper bound, however, cannot be found by partition addition, as demonstrated below.

Example 5.7

	i_0	i_1	i_2	i_3
1	c_1	3	5	2
2	5	c_2	c_3	c_4
3	6	6	c_5	7
4	c_6	c_7	2	c_8
5	c_9	6	1	4
6	3	c_{10}	2	3
7	7	4	c_{11}	7

M'

$$\pi_1 = \{\overline{1,2,c_4};\ \overline{3,c_2};\ \overline{5,c_1,c_3};\ \overline{4};\ \overline{6};\ \overline{7};\ \overline{c_5};\ \overline{c_6};\ \overline{c_7};\ \overline{c_8};\ \overline{c_9};\ \overline{c_{10}};\ \overline{c_{11}}\}$$

and

$$\pi_2 = \{\overline{1,4,c_3,c_4};\ \overline{2,5,c_8,c_9};\ \overline{3,c_7};\ \overline{6,c_2};\ \overline{7};\ \overline{c_1,c_6};\ \overline{c_5};\ \overline{c_{10}};\ \overline{c_{11}}\}$$

are extended substitution property partitions for M'.
However, $\pi_1 + \pi_2 = \{\overline{1,2,4,5,c_1,c_3,c_4,c_6,c_8,c_9};\ \overline{3,6,c_2,c_7};\ \overline{7};\ \overline{c_5};\ \overline{c_{10}};\ \overline{c_{11}}\}$ is not an extended substitution property partition, since

$\{3,6\} \xrightarrow{i_3} \{3,7\}$.

In order to find the least upper bound of two extended substitution property partitions, we would have to derive $Y(\pi_1 + \pi_2)$. Thus the use of extended substitution property partitions does not provide any advantage over weak substitution property partitions. The additional effort required to manipulate the elements of C make it easier to use weak substitution property partitions.

5.4 Partial Serial Decompositions

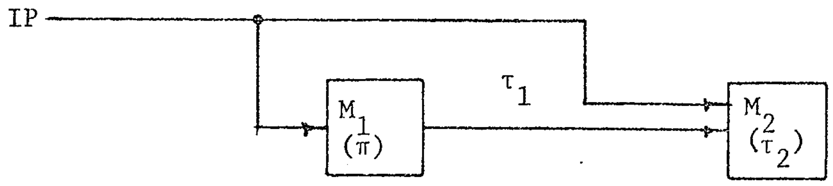
A machine decomposition developed by Hartmanis [27] as a special form of the serial decomposition is examined in this section. Hartmanis used the decomposition to reduce the state variable dependence for certain types of serial decompositions. That is, for some serial decompositions not all the information available in the front machine is needed to realize the tail machine. For cases where this is possible, the tail machine is simpler to construct because of the reduced state variable dependence. Hartmanis' Corollary 1 [27] which establishes necessary and sufficient conditions for this decomposition is stated as Theorem 5.2.

Theorem 5.2 (Hartmanis) If π is a partition with S.P. for the sequential machine M, then we can use π to realize M from two concurrently operating sequential machines connected in series, M_1 and M_2, such that (1) M_1 computes the block of π which contains the state of M, and (2) M_2 has q states and receives at most d different outputs from M_1, if and only if there exist two partitions τ_1 and τ_2 on the set of states of M such that

$$\begin{aligned} \tau_1 &> \pi, \\ \tau_2 \cdot \pi &= 0 \\ e(\tau_1) &\leq d, \\ e(\tau_2) &\leq q, \end{aligned}$$

and $(\tau_1 \cdot \tau_2, \tau_2)$ is a partition pair for M.

In order to facilitate discussion of this decomposition, we refer to it as a partial serial decomposition. A partial serial decomposition for a machine M is illustrated in Figure 5.1.



Partial serial decomposition

Figure 5.1

Our interest in the partial serial decomposition is not in reducing variable dependence for a serial decomposition, but in using the decomposition for realizing incompletely specified machines. It was proven, in the previous two sections, that the sum of weak (extended) substitution property partitions is not necessarily a weak (extended) substitution property partition. Thus, a factored realization of two weak (extended) substitution property partitions is not always possible and the partitions may have to be realized as separate machines. The partial serial decomposition, however, enables a more economical realization of two weak (extended) substitution property partitions. Using the partial serial decomposition, a common partition, τ_1, for the weak (extended) substitution property partitions can be realized in just one of the machines, M_1, and this state information made available to the other machine, M_2. Essentially, a common machine is factored from just one of the machines corresponding to the weak (extended) substitution property partitions, instead of from both.

The following lemma proves that two S.P. partitions, whose sum is not I, can be realized as a partial serial decomposition. The theory and examples presented, which are for completely specified machines, apply equally to incompletely specified machines.

Lemma 5.4 For a machine M there exists a partial serial decomposition of M if there are two S.P. partitions π and π_1 on M such that $\pi + \pi_1 = \pi_2 \neq I$, $\pi \cdot \pi_1 = 0$, $\pi_2 > \pi$, and $\pi_2 > \pi_1$.

Proof The lemma is proved by showing that it is possible to construct partitions τ_1 and τ_2, which with π, satisfy the sufficiency conditions of Theorem 5.2.

Let $\tau_1 = \pi_2$. Therefore, $\tau_1 > \pi$.

Since $\tau_1 = \pi_2 > \pi_1$, there exists a partition τ_2 such that

$\tau_1 \cdot \tau_2 = \pi_1$.

$$\pi \cdot \pi_1 = 0$$

$$\pi \cdot \tau_1 \cdot \tau_2 = 0$$

As $\tau_1 > \pi$, $\pi \cdot \tau_1 \cdot \tau_2 = \pi \cdot \tau_2$

$\therefore\ \pi \cdot \tau_2 = 0$

(π_1,π_1) is a partition pair since π_1 is an S.P. partition.

$\therefore\ (\pi_1,\tau_2)$ is also a partition pair $(\tau_2 > \pi_1)$.

Thus, $(\tau_1 \cdot \tau_2, \tau_2)$ is a partition pair.

The conditions $e(\tau_1) \le d$ and $e(\tau_2) \le q$ can be satisfied by definition.

In the following example, a partial serial decomposition is obtained for machine M.

Example 5.8

	i_0	i_1	i_2	i_3	i_4
1	2	1	3	4	5
2	1	2	6	7	8
3	3	6	5	8	7
4	4	7	5	1	6
5	8	5	6	6	1
6	6	3	8	5	4
7	7	4	8	2	3
8	5	8	3	3	2

M

$\pi = \{\overline{1,2}; \overline{3,6}; \overline{4,7}; \overline{5,8}\}$ and $\pi_1 = \{\overline{1,8}; \overline{2,5}; \overline{3,4}; \overline{6,7}\}$ are S.P. partitions on M. Since $\pi + \pi_1 \ne I$ and $\pi \cdot \pi_1 = 0$, Lemma 5.4 can be used to construct a partial serial decomposition for M.

The head machine, M_1 of Figure 5.1, is constructed using S.P. partition π. Define $\tau_1 = \pi + \pi_1 = \{\overline{1,2,5,8}; \overline{3,4,6,7}\} = \{\overline{a_1}; \overline{a_2}\}$ and $\tau_b = \{\overline{1,2,3,6}; \overline{4,5,7,8}\} = \{\overline{b_1}; \overline{b_2}\}$ such that $\tau_1 \cdot \tau_b = \pi$.

		i_0	i_1	i_2	i_3	i_4
(1,2)	a_1b_1	a_1b_1	a_1b_1	a_2b_1	a_2b_2	a_1b_2
(5,8)	a_1b_2	a_1b_2	a_1b_2	a_2b_1	a_2b_1	a_1b_1
(3,6)	a_2b_1	a_2b_1	a_2b_1	a_1b_2	a_1b_2	a_2b_2
(4,7)	a_2b_2	a_2b_2	a_2b_2	a_1b_2	a_1b_1	a_2b_1

M_1

Machine M_2, of Figure 5.1, is realized using the partition τ_2, where $\tau_1 \cdot \tau_2 = \pi_1$. Let $\tau_2 = \{\overline{1,3,4,8}; \overline{2,5,6,7}\} = \{\overline{c_1}; \overline{c_2}\}$.
As proven in Lemma 5.4, $(\tau_1 \cdot \tau_2, \tau_2)$ is a partition pair.

		a_1i_0	a_1i_1	a_1i_2	a_1i_3	a_1i_4	a_2i_0	a_2i_1	a_2i_2	a_2i_3	a_2i_4
(1,3,4,8)	c_1	c_2	c_1	c_1	c_1	c_2	c_1	c_2	c_2	c_1	c_2
(2,5,6,7)	c_2	c_1	c_2	c_2	c_2	c_1	c_2	c_1	c_1	c_2	c_1

M_2

Connecting M_1 and M_2 as in Figure 5.1 produces the machine M'.

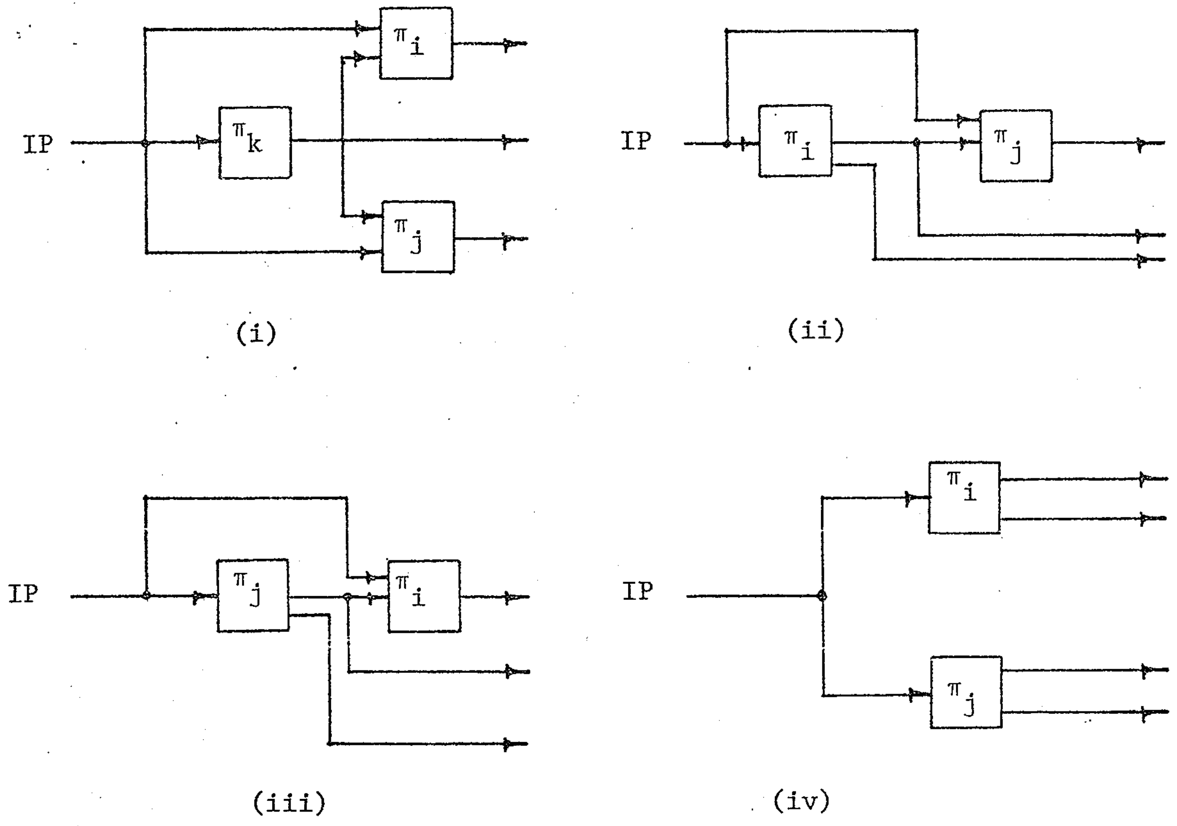
	i_0	i_1	i_2	i_3	i_4
$a_1b_1c_1$	$a_1b_1c_2$	$a_1b_1c_1$	$a_2b_1c_1$	$a_2b_2c_1$	$a_1b_2c_2$
$a_1b_1c_2$	$a_1b_1c_1$	$a_1b_1c_2$	$a_2b_1c_2$	$a_2b_2c_2$	$a_1b_2c_1$
$a_2b_1c_1$	$a_2b_1c_1$	$a_2b_1c_2$	$a_1b_2c_2$	$a_1b_2c_1$	$a_2b_2c_2$
$a_2b_2c_1$	$a_2b_2c_1$	$a_2b_2c_2$	$a_1b_2c_2$	$a_1b_1c_1$	$a_2b_1c_2$
$a_1b_2c_2$	$a_1b_2c_1$	$a_1b_2c_2$	$a_2b_1c_2$	$a_2b_1c_2$	$a_1b_1c_1$
$a_2b_1c_2$	$a_2b_1c_2$	$a_2b_1c_1$	$a_1b_2c_1$	$a_1b_2c_2$	$a_2b_2c_1$
$a_2b_2c_2$	$a_2b_2c_2$	$a_2b_2c_1$	$a_1b_2c_1$	$a_1b_1c_2$	$a_2b_1c_1$
$a_1b_2c_1$	$a_1b_2c_2$	$a_1b_2c_1$	$a_2b_1c_1$	$a_2b_1c_1$	$a_1b_1c_2$

M'

Renaming the states of M' such that $a_1b_1c_1 = 1$, $a_1b_1c_2 = 2$, $a_2b_1c_1 = 3, \ldots\ldots, a_1b_2c_1 = 8$, gives M.

Partial Serial Decomposition Evaluation

For a state machine with three nontrivial S.P. partitions, π_i, π_j, and π_k, such that $\pi_i \cdot \pi_j = 0$ and $\pi_i + \pi_j = \pi_k$, there are four possible ways to decompose the machine. This is demonstrated in Figure 5.2.



Possible decompositions for π_i and π_j

Figure 5.2

For decompositions (ii) and (iii), the common submachine has been factored out of one of the partitions and computed by the other. This reduction in variable dependency for one of the partitions should make decompositions (ii) and (iii) more economical than decomposition (iv). However, since the common submachine has not been factored out of both partitions, decomposition (i) should be more economical than both (ii) and (iii).

Evaluation of the above decompositions using the ROM size formula, presented in Section 4.4, verifies the above assumptions. First, some notation is necessary.

Notation

(1) The ROM size of the parallel decomposition $\{\pi_i,\pi_j\}$ will be denoted by R_1.

(2) R_{1_i} denotes the ROM size of the partial serial decomposition with π_i as the head machine, while R_{1_j} denotes the ROM size with π_j as the head machine.

(3) The ROM size of the decomposition $\{\pi_i,\pi_j,\pi_k\}$ is denoted R_2.

LEMMA 5.5 For the S.P. partitions π_i,π_j, and π_k where $\pi_i+\pi_j = \pi_k$, $R_1 > R_{1_i} > R_2$ and $R_1 > R_{1_j} > R_2$.

Proof From Lemma. 4.1 we know that

$$R_1 = 2^{N(IP)+e(\pi_k)}\left[e(\pi_k)\cdot 2^{e(\pi_k|\pi_i)} + e(\pi_k)\cdot 2^{e(\pi_k|\pi_j)}\right] + 2^{N(IP)+e(\pi_k)}\left[e(\pi_k|\pi_i)\cdot 2^{e(\pi_k|\pi_i)} + e(\pi_k|\pi_j)\cdot 2^{e(\pi_k|\pi_j)}\right]$$

and

$$R_2 = e(\pi_k)\cdot 2^{N(IP)+e(\pi_k)} + 2^{N(IP)+e(\pi_k)}\left[e(\pi_k|\pi_i)\cdot 2^{e(\pi_k|\pi_i)} + e(\pi_k|\pi_j)\cdot 2^{e(\pi_k|\pi_j)}\right].$$

The ROM size of R_{1_i} is calculated as follows

$$R_{1_i} = e(\pi_i)\cdot 2^{N(IP)+e(\pi_i)} + e(\pi_k|\pi_j)\cdot 2^{N(IP)+e(\pi_k)+e(\pi_k|\pi_j)}$$

$$= [e(\pi_k)+e(\pi_k|\pi_i)]\cdot 2^{N(IP)+e(\pi_k)+e(\pi_k|\pi_i)} + e(\pi_k|\pi_j)\cdot 2^{N(IP)+e(\pi_k)+e(\pi_k|\pi_j)}$$

Thus, $R_{1_i} = e(\pi_k)\cdot 2^{N(IP)+e(\pi_k)+e(\pi_k|\pi_i)} + 2^{N(IP)+e(\pi_k)}\left[e(\pi_k|\pi_i)\cdot 2^{e(\pi_k|\pi_i)} + e(\pi_k|\pi_j)\cdot 2^{e(\pi_k|\pi_j)}\right].$

$$R_1 - R_{1_i} = e(\pi_k)\cdot 2^{N(IP)+e(\pi_k)+e(\pi_k|\pi_j)}$$

and

$$R_{1_i} - R_2 = e(\pi_k)\cdot 2^{N(IP)+e(\pi_k)}\left[2^{e(\pi_k|\pi_i)} - 1\right]$$

Therefore, $R_1 > R_{1_i} > R_2$.

Similarly, $R_1 > R_{1_j} > R_2$.

Corollary 5.5.1 For the two partial serial decompositions for the set $\{\pi_i, \pi_j\}$, where $\pi_i + \pi_j = \pi_k \neq I$, then $R_{1_i} \geq R_{1_j}$ if and only if $e(\pi_k | \pi_i) \geq e(\pi_k | \pi_j)$.

Example 5.9 Calculate the ROM costs for the four decompositions of π_1 and π_2 of machine M, in Example 5.8.

Let $\pi_k = \pi_3$, $\pi_i = \pi_1$, and $\pi_j = \pi_2$.

$$R_1 = 2 \cdot 2^{3+2} + 2 \cdot 2^{3+2}$$
$$= 128 \text{ bits}$$
$$R_2 = 1 \cdot 2^{3+1} + 1 \cdot 2^{3+1+1} + 1 \cdot 2^{3+1+1}$$
$$= 80 \text{ bits}$$
$$R_{1_i} = 2 \cdot 2^{3+2} + 1 \cdot 2^{3+1+1}$$
$$= 96 \text{ bits}$$

Thus, $R_1 > R_{1_i} > R_2$

$$R_1 - R_{1_i} = 32 \text{ bits}$$

Calculating $R_1 - R_{1_i}$ using Lemma 5.5 gives

$$R_1 - R_{1_i} = e(\pi_3) \cdot 2^{N(IP) + e(\pi_3) + e(\pi_3 | \pi_2)}$$
$$= 1 \cdot 2^{3+1+1}$$
$$= 32 \text{ bits}$$

Since $e(\pi_k | \pi_i) = e(\pi_k | \pi_j)$, $R_{1_i} = R_{1_j}$ and neither partial serial decomposition is more economical than the other.

Application to Incompletely Specified Machines

Partial serial decompositions are not as economical as type (i) decompositions when applied to completely specified machines or incompletely specified machines, where π_i, π_j, and $\pi_i + \pi_j = \pi_k$ are weak substitution property partitions. However, for incompletely specified machines where $\pi_k = \pi_i + \pi_j$ is not a weak substitution property partition, it is not possible to compute π_k separately. Consequently, the partial serial decomposition is the most economical decomposition that can be used in this case.

The partial serial decomposition for incompletely specified machines is calculated the same way as for completely specified machines. However, the machine realized is greatly influenced by which is chosen to be the head machine. This is demonstrated by the following example.

Example 5.10

	i_0	i_1	i_2	i_3
1	2	3	-	4
2	3	5	4	2
3	4	2	3	-
4	5	3	-	1
5	-	2	-	-

M

For M, $\pi = \{\overline{1}; \overline{2}; \overline{3,5}; \overline{4}\}$ and $\pi_1 = \{\overline{1,4}; \overline{2,5}; \overline{3}\}$ are weak substitution property partitions on M. However, $\tau_1 = \pi+\pi_1 = \{\overline{1,4}; \overline{2,3,5}\}$ is not a weak substitution property partition. Since, $\pi \cdot \pi_1 = 0$, machine M can be realized as a partial serial decomposition using π and π_1. A decomposition is constructed using π as the head machine, M_1.

Let $\tau_1 = \{\overline{1,4}; \overline{2,3,5}\} = \{\overline{a_1}; \overline{a_2}\}$ and $\tau_b = \{\overline{1,2}; \overline{3,4,5}\} = \{\overline{b_1}; \overline{b_2}\}$.

		i_0	i_1	i_2	i_3
(1)	a_1b_1	a_2b_1	a_2b_2	-	a_1b_2
(4)	a_1b_2	a_2b_2	a_2b_2	-	a_1b_1
(2)	a_2b_1	a_2b_2	a_2b_2	a_1b_2	a_2b_1
(3,5)	a_2b_2	a_1b_2	a_2b_1	a_2b_2	-

M_1

The tail machine M_2 is realized using the partition $\tau_2 = \{\overline{1,3,4}; \overline{2,5}\} = \{\overline{c_1}; \overline{c_2}\}$, where $\tau_1 \cdot \tau_2 = \pi_1$

		i_0a_1	i_1a_1	i_2a_1	i_3a_1	i_0a_2	i_1a_2	i_2a_2	i_3a_2
(1,3,4)	c_1	c_2	c_1	-	c_1	c_1	c_2	c_1	-
(2,5)	c_2	-	-	-	-	c_1	c_2	c_1	c_2

M_2

Connecting M_1 and M_2 as a partial serial decomposition produces the incompletely specified machine M'

	i_0	i_1	i_2	i_3
$a_1b_1c_1$	$a_2b_1c_2$	$a_2b_2c_1$	-	$a_1b_2c_1$
$a_2b_1c_2$	$a_2b_2c_1$	$a_2b_2c_2$	$a_1b_2c_1$	$a_2b_1c_2$
$a_2b_2c_1$	$a_1b_2c_1$	$a_2b_1c_2$	$a_2b_2c_1$	-
$a_1b_2c_1$	$a_2b_2c_2$	$a_2b_2c_1$	-	$a_1b_1c_1$
$a_2b_2c_2$	$a_1b_2c_1$	$a_2b_1c_2$	$a_2b_2c_1$	-- c_2

M'

Renaming the states of M' such that $a_1b_1c_1 = 1$, $a_2b_1c_2 = 2$, and so forth, gives the machine M_a.

	i_0	i_1	i_2	i_3
1	2	3	-	4
2	3	5	4	2
3	4	2	3	-
4	5	3	-	1
5	4	2	3	{2,5}

M_a

An obvious difference between M and M_a is that next-states $\delta(5,i_0)$, $\delta(5,i_2)$, and $\delta(5,i_3)$ for M_a are not unspecified. $\delta(5,i_0)$ and $\delta(5,i_2)$ are specified by a single state, while $\delta(5,i_3)$ is specified by a subset of states. Another difference is that $\pi_1 = \{\overline{1,4}; \overline{2,5}; \overline{3}\}$ is not a weak substitution property partition for M_a.

These differences are related as will be demonstrated. In realizing π, implicit assumptions are made about the unspecified next-states of M. For π to be a weak substitution property partition on M, $\delta(5,i_0)$ must be 4 as $3 \equiv 5(\pi)$ and $\delta(3,i_0) = 4$. Similarly, the next-states for $(5,i_2)$ and $(5,i_3)$ are determined by π. Thus, a weak substitution property partition on an incompletely specified machine implicitly specifies the unspecified next states.

However, specifying $\delta(5,i_0) = 4$, prevents π_1 from being a weak substitution property partition on M_a, since $2 \equiv 5(\pi_1)$ and $\delta(2,i_0) = 3 \not\equiv 4 = \delta(5,i_0)(\pi_1)$.

This conflict between the implicit next-state assumptions required by π and π_1 also explains why $\pi+\pi_1$ is not a weak substitution property partition. For π to be a weak substitution property partition, π_1 cannot be a weak substitution property partition. Similarly, the converse holds. Since π and π_1 cannot be weak substitution property partitions at the same time, it follows that their sum cannot be a weak substitution property partition.

We may regard the "don't-care" conditions of M as specifying that M may enter any of the states of S = {1,2,3,4,5}.

	i_0	i_1	i_2	i_3
1	2	3	{1,2,3,4,5}	4
2	3	5	4	2
3	4	2	3	{1,2,3,4,5}
4	5	3	{1,2,3,4,5}	1
5	{1,2,3,4,5}	2	{1,2,3,4,5}	{1,2,3,4,5}

M

Thus machine M_a realizes a subset of the possible realizations for M. The realization of a subset instead of the complete set is not unique to partial serial decompositions. Choosing any weak substitution property partition on M implicitly specifies the "don't-care". This in turn only permits a subset of machines to be realized.

Thus far in this chapter we have been examining the application of S.P. theory to incompletely specified machines. In the following section S.P. theory is extended to asynchronous machines and the economical decomposition of multiple machines.

5.5 Synthesis of Multiple Sequential Machines

Background

Kohavi and Smith [44],[62] and Kohavi [43] introduce the concept of composite machines in order to synthesize multiple machines. Their definition of composite machines is only for two machines, although it is easily extended for n machines.

Definition 5.5 For two machines M_1 and M_2 with initial states s_1 and r_1, respectively, the composite machine, M_1M_2 is that machine having initial state s_1r_1 and all subsequent states which are implied in a chain fashion by s_1r_1 and the inputs.

Notation A state, r_is_j, on the composite machine is called a composite state. The partition π_i, on the composite machine, formed by placing composite states in the same block of π_i, if and only if their substates belonging to M_i are equal, is called the state-consistent partition with respect to M_i. Note that the state-consistent partition with respect to M_i is equivalent to M_i.

Example 5.11

We derive the composite machine for machines M_1 and M_2, with initial states r_1 and s_1, respectively.

	i_0	i_1	OP_1
r_1	r_1	r_2	0
r_2	r_2	r_3	1
r_3	r_3	r_4	1
r_4	r_4	r_1	0

M_1

	i_0	i_1	OP_2
s_1	s_3	s_2	0
s_2	s_4	s_3	0
s_3	s_1	s_4	0
s_4	s_2	s_1	1

M_2

The initial states r_1 and s_1 are combined. Under input i_0, r_1s_1 implies that r_1s_3 should also be combined. Continuing in this manner the composite machine is produced.

	i_0	i_1
r_1s_1	r_1s_3	r_2s_2
r_1s_3	r_1s_1	r_2s_4
r_2s_2	r_2s_4	r_3s_3
r_2s_4	r_2s_2	r_3s_1
r_3s_3	r_3s_1	r_4s_4
r_3s_1	r_3s_3	r_4s_2
r_4s_4	r_4s_2	r_1s_1
r_4s_2	r_4s_4	r_1s_3

M_1M_2

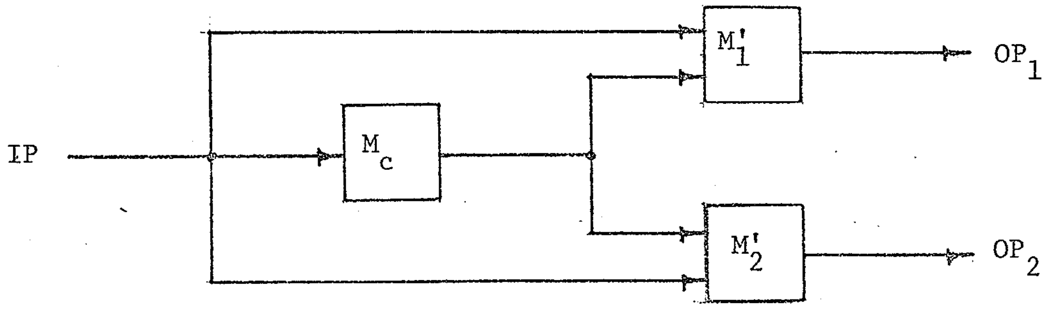
For the composite machine M_1M_2, the state-consistent partitions with respect to M_1 and M_2 are

$$\pi_1 = \{\overline{r_1s_1,r_1s_3};\ \overline{r_2s_2,r_2s_4};\ \overline{r_3s_1,r_3s_3};\ \overline{r_4s_2,r_4s_4}\}$$

and $$\pi_2 = \{\overline{r_1s_1,r_3s_1};\ \overline{r_2s_2,r_4s_2};\ \overline{r_1s_3,r_3s_3};\ \overline{r_2s_4,r_4s_4}\}\ ,$$

respectively.

For a composite machine M_1M_2 if a partition π_c exists such that $\pi_c > \pi_1,\pi_2$, then M_1 and M_2 can be realized by a cascade decomposition of machine M_c and successor machines M_1' and M_2'. Machine M_c is referred to as a common factor machine for M_1 and M_2. In Figure 5.3, M_c realizes partition π_c and M_k' realizes π_k, $k = 1,2$.



Two machines with a common factor machine

Figure 5.3

Definition 5.6 A common factor machine will be said to be a maximum common machine when it, together with the two successor machines, is the largest machine capable of generating the required number of states in M_1M_2, such that the number of internal state variables upon which the outputs, OP_1 and OP_2, depend is not increased.

Kohavi and Smith show that the maximum number of state variables, $k_c$max, in a maximum common machine is given by

$$k_c\text{max} = k_1 + k_2 - k_{M_1M_2},$$

where k_1 and k_2 are the number of states in M_1 and M_2, respectively, and $k_{M_1M_2}$ is the number of states in the composite machine.

For some machines, the maximum common machine can be obtained by summing the state-consistent partitions.

Example 5.12 Summing the state-consistent partitions of Example 5.12 produces

$$\begin{aligned}\pi_c &= \pi_1 + \pi_2\\ &= \{\overline{r_1s_1, r_1s_3, r_3\ s_3, r_3s_1};\ \overline{r_2s_2, r_2s_4, r_4s_2, r_4s_4}\}\end{aligned}$$

For the composite machine M_1M_2, $k_{M_1M_2} = 3$. Thus,

$$\begin{aligned}k_c\text{max} &= k_1 + k_2 - k_{M_1M_2}\\ &= 1 \text{ and } k_c = k_c\text{max}.\end{aligned}$$

We define the successor machines

$\pi_{s_1} = \{\overline{r_1s_1, r_1s_3, r_2s_2, r_2s_4};\ \overline{r_3s_1, r_3s_3, r_4s_2, r_4s_4}\}$ and

$\pi_{s_2} = \{\overline{r_1s_1, r_3s_1, r_2s_2, r_4s_2};\ \overline{r_1s_3, r_3s_3, r_2s_4, r_4s_4}\}$. Since $\pi_c \cdot \pi_{s_1} = \pi_1$ and $\pi_c \cdot \pi_{s_2} = \pi_2$, the outputs OP_1 and OP_2 can be realized from π_c, π_{s_1}, and π_{s_2} without increasing the number of internal variables that they are dependent upon. At the same time, the use of M_c as a common factor machine does not entail more state variables than required for the composite machine M_1M_2.

However, for most machines, the maximum common machine cannot always be obtained by simply summing the state-consistent partitions. It may be necessary to split states of the composite machine in order to obtain the maximum common machine. This is demonstrated in the following example.

Example 5.13

	i_0	i_1
1	2	4
2	1	3
3	1	4
4	2	3

M_1

	i_0	i_1
a	a	c
b	a	b
c	b	c

M_2

The composite machine is

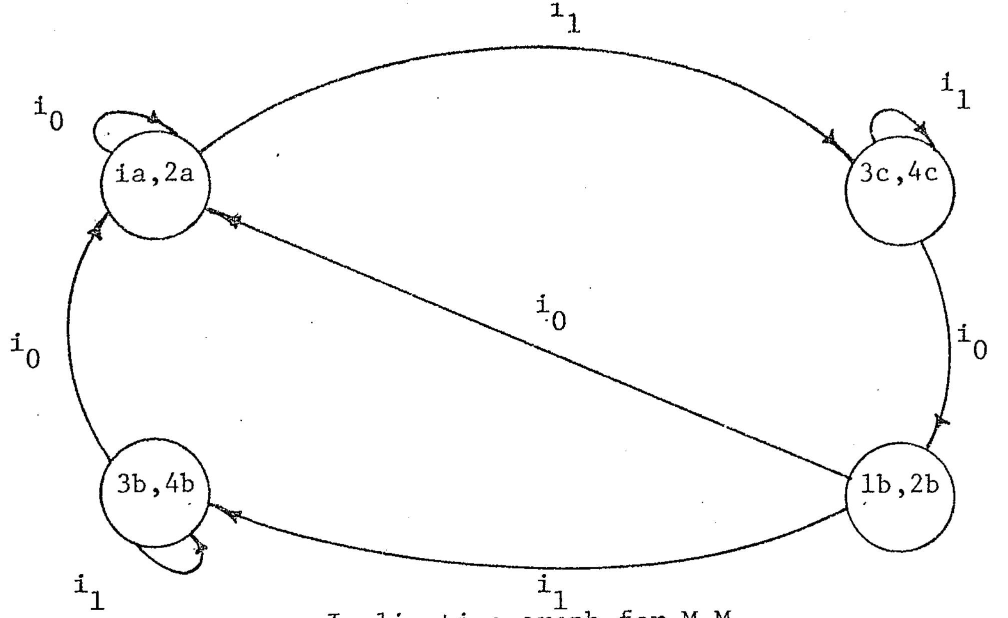
	i_0	i_1
1a	2a	4c
2a	1a	3c
4c	2b	3c
3c	1b	4c
2b	1a	3b
1b	2a	4b
3b	1a	4b
4b	2a	3b

M_1M_2

State-consistent partitions are $\pi_1 = \{\overline{1a,1b}; \overline{2a,2b}; \overline{3b,3c}; \overline{4b,4c}\}$ and $\pi_2 = \{\overline{1a,2a}; \overline{1b,2b,3b,4b}; \overline{3c,4c}\}$

$$\pi_c = \pi_1 + \pi_2 = \{\overline{1a,1b,2a,2b,3b,3c,4b,4c}\}$$
$$= I\ .$$

Obviously M_c is not a maximum common machine for M_1M_2. However, a maximum common machine can be obtained by splitting states of M_1M_2. This is accomplished by constructing an implication graph [41], identifying composite states 1a and 2a.



Implication graph for M_1M_2

Figure 5.4

The new partition $\pi' = \{\overline{1a,2a}; \overline{1b,2b}; \overline{3b,4b}; \overline{3c,4c}\}$ implicitly splits state b on M_2, giving the equivalent machine M_2' .

	i_0	i_1
a	a	c
b	a	b'
b'	a	b'
c	b	c

M_2'

The new common machine factor is obtained by summing the partitions π_1 and π'.

$$\pi_c = \pi_1 + \pi' = \{\overline{1a,1b,2a,2b}; \overline{3c,3b,4b,4c}\}.$$

It can easily be seen that π_c is the maximum common machine for M_1 and M_2'. Thus with state-splitting it is possible to economically synthesize multiple machines.

Submachine Equivalence

We now present a more formal treatment of the synthesis of multiple machines. The theory presented examines the structure of composite machines as determined by the structure of the component machines. The results obtained provide a theoretical justification of the methods of Kohavi and Smith and permit a more general examination of multiple machine synthesis.

The generation of a composite machine, from machines M_1

and M_2, associates states of M_1 with states of M_2, and vice versa.

<u>Definition 5.7</u> For two machines $M_1 = (S_1, IP_1, OP_1, \delta_1, \lambda_1)$ and $M_2 = (S_2, IP_2, OP_2, \delta_2, \lambda_2)$ with start states a_1 and a_2, respectively, the states $x_1 \in S_1$ and $x_2 \in S_2$ are said to be <u>M_1M_2 equivalent, $x_1 \equiv x_2(M_1M_2)$</u>, if and only if

(i) $x_1 = a_1$ and $x_2 = a_2$, or

(ii) $t_1 \equiv t_2\ (M_1M_2)$, $\delta_1(t_1,i) = x_1$, and
$\delta_2(t_2,i) = x_2$, for some $i \in IP$.

(For convenience we will not define M_2M_1 equivalence but instead will use M_1M_2 equivalence to represent both.)

States from S_1 and S_2 may be M_1M_2 equivalent with more than just one state of S_2 and S_1, respectively.

<u>Definition 5.8</u> The equivalence set for state a on M_2, <u>aM_2</u>, $a \in S_1$ is the set

$$aM_2 = \{b | b \in S_2 \text{ and } a \equiv b(M_1M_2)\} .$$

The equivalence sets for two machines, M_1 and M_2, are related. For simplicity the following results will be developed for state machines.

<u>Theorem 5.3</u> For state machines $M_1 = (S_1, IP, \delta_1)$ and $M_2 = (S_2, IP, \delta_2)$, if $b \in S_2$ then

$bM_1 = \{a | a \in S_1 \text{ and } b \in aM_2\}$.

<u>Proof</u> Denote $\{a | a \in S_1 \text{ and } b \in aM_2\}$ by Y.

(i) Let $x \in bM_1$.
Therefore, $x \in S_1$ and $x \equiv b(M_1M_2)$.
Since $b \in S_2$ and $x \equiv b(M_1M_2)$, $b \in xM_2$.
By definition of Y, $x \in Y$ and $bM_1 \subseteq Y$.

(ii) Let $x \in Y$.
Therefore, $x \in S_1$ and $b \in xM_2$.
Since $b \in xM_2$, then $x \equiv b(M_1M_2)$ and $x \in bM_1$.
Thus, $y \subseteq bM_1$ and $bM_1 = Y = \{a | a \in S_1 \text{ and } b \in aM_2\}$.

In the following example we demonstrate the calculation of equivalence sets for two machines, M_1 and M_2.

Example 5.14

	i_0	i_1
r_1	r_1	r_2
r_2	r_2	r_3
r_3	r_3	r_4
r_4	r_4	r_1

M_1

	i_0	i_1
s_1	s_3	s_2
s_2	s_4	s_3
s_3	s_1	s_4
s_4	s_2	s_1

M_2

Since r_1 and s_1 are the start states of M_1 and M_2, respectively, $r_1 \equiv s_1(M_1M_2)$. Also since $r_1 \equiv s_1(M_1M_2)$, $\delta(r_1,i_0) = r_1$ and $\delta_2(s_1,i_0) = s_3$, then $r_1 \equiv s_3(M_1M_2)$. The M_1M_2 equivalent states for M_1 and M_2 are (r_1,s_1), (r_1,s_3), (r_2,s_2), (r_2,s_4), (r_3,s_3), (r_3,s_1), (r_4,s_4), and (r_4,s_2).

The M_1M_2 equivalent states can also be determined from the composite machine.

	i_0	i_1
r_1s_1	r_1s_3	r_2s_2
r_1s_3	r_1s_1	r_2s_4
r_2s_2	r_2s_4	r_3s_3
r_2s_4	r_2s_2	r_3s_1
r_3s_3	r_3s_1	r_4s_4
r_3s_1	r_3s_3	r_4s_2
r_4s_4	r_4s_2	r_1s_1
r_4s_2	r_4s_4	r_1s_3

M_1M_2

Since states r_2 and r_4 are associated with each other during the formation of M_1M_2, $r_2 \equiv s_4$ (M_1M_2). The other equivalent states can be found in a similar manner.

For state r_1,$r_1 \equiv s_1(M_1M_2)$ and $r_1 \equiv s_3(M_1M_2)$. Thus,

$r_1M_2 = \{s_1,s_3\}$. Similarly, $r_2M_2 = \{s_2,s_4\}$, $r_3M_2 = \{s_1,s_3\}$, and $r_4M_2 = \{s_2,s_4\}$.

Using Theorem 5.3, the equivalence sets on M_1 can be determined from the equivalence sets on M_2. That is, since $s_2 \in r_2M_2$ and $s_2 \in r_4M_2$, $s_2M_1 = \{r_2,r_4\}$.
Similarly, $s_1M_1 = \{r_1,r_3\}$, $s_3M_1 = \{r_1,r_3\}$, and $s_4M_1 = \{r_2,r_4\}$.

Blocks of states on M_1 are associated with blocks of states on M_2 by the equivalence sets for the machines.

Definition 5.9 For state machines $M_1 = (S_1,IP,\delta_1)$ and $M_2 = (S_2,IP,\delta_2)$, a block of states, B, on S_1 induces an equivalence block, C, on S_2 where $C = \bigcup_{a\in B} a\,M_2$.

A partition on S_1 groups the states of S_2 into equivalence blocks. The set of equivalence blocks produced are not necessarily disjoint. We define a way of grouping the equivalence blocks, which is a generalization of a partition.

Definition 5.10 A set of subsets of S, B_i is called a collection on S, if and only if $\cup B_i = S$.

Definition 5.11. The collection formed on S_2 by partition τ on S_1, consisting of the equivalence blocks induced by τ, is called the equivalence collection, τ_{ec}.

We must now establish a basis for determining the common component machines of a composite machine.

Theorem 5.4 For the state machines $M_1 = (S_1,IP,\delta_1)$ and $M_2 = (S_2,IP,\delta_2)$, if π is an S.P. partition on S_1, then the equivalence collection π_{ec} on S_2 also has S.P..

Proof. If for $x_1,x_2 \in S_2$, $x_1 \equiv x_2$ (π_{ec}), then x_1 and x_2 are in the same block of π_{ec} as the result of one of the conditions below:

(i) $x_1, x_2 \in rM_2$, where $r \in S_1$(i.e., $r \equiv x_1\ (M_1M_2)$ and $r \equiv x_2(M_1M_2)$.

(ii) $x_1 \in r_jM_2$ and $x_2 \in r_kM_2$ and $r_j \equiv r_k(\pi)$.

Assume condition (i).

Therefore, $\delta_1(r,i) \equiv \delta_2(x_1,i)(M_1M_2)$ and

$\delta_1(r,i) \equiv \delta_2(x_2,i)(M_1M_2)$, $i \in IP$.

Thus, $\delta_2(x_1,i)$ and $\delta_2(x_2,i) \in \delta_1(r,i)M_2$ and $\delta_2(x_1,i) \equiv \delta_2(x_2,i)(\pi_{ec})$.

Assume condition (ii)

$\delta_1(r_j,i) \equiv \delta_2(x_1,i)(M_1M_2)$ and

$\delta_1(r_k,i) \equiv \delta_2(x_2,i)(M_1M_2)$, $i \in IP$.

Therefore, $\delta_2(x_1,i) \in \delta_1(r_j,i)$ and $\delta_2(x_2,i) \in \delta_1(r_k,i)$.

Since $\delta_1(r_j,i) \equiv \delta_1(r_k,i)(\pi)$, $\delta_1(r_j,i)M_2$ and $\delta_1(r_k,i)\ M_2 \subseteq C_\ell$, where C_ℓ is a block of π_{ec}.

Thus, $\delta_2(x_1,i) \equiv \delta_2(x_2,i)(\pi_{ec})$ and π_{ec} has S.P.

Theorem 5.4 is used to construct the common component machines for the composite machine of

Example 5.15

	i_0	i_1
r_1	r_1	r_2
r_2	r_2	r_3
r_3	r_3	r_4
r_4	r_4	r_1

M_1

	i_0	i_1
s_1	s_3	s_2
s_2	s_4	s_3
s_3	s_1	s_4
s_4	s_2	s_1

M_2

$\pi = \{\overline{r_1,r_3};\ \overline{r_2,r_4}\}$ is an S.P. partition on M_1. The equivalence collection, π_{ec}, induced by π on S_2 is $\pi_{ec} = \{\overline{s_1,s_3};\ \overline{s_2,s_4}\}$, since $r_1M_2 = \{s_1,s_3\}$, $r_3M_2 = \{s_1,s_3\}$, $r_2M_2 = \{s_2,s_4\}$, and $r_4M_2 = \{s_2,s_4\}$.

By inspection of M_2 it can easily be seen that π_{ec} has S.P.

It can also be shown that there exists a state isomorphism from an S.P. partition on M_1 to the equivalence collection on M_2. The isomorphism is defined by the equivalence sets.

Theorem 5.5 For state machines $M_1 = (S_1, IP, \delta_1)$ and $M_2 = (S_2, IP, \delta_2)$, the S.P. partition π on M_1 and the equivalence partition π_{ec} on M_2 are equivalent. That is, there exists a state isomorphism from state machine π to state machine π_{ec}.

Proof Define function $f:B \rightarrow C$, where B is a block of π and C is a block of π_{ec} to be the mapping formed by the equivalence blocks of π on S_2. (f is a 1-1 mapping by definition.)

It must be shown that

$$\delta_2(f(B_i),x) = f(\delta_1(B_i,x)),\ x \in IP.$$

$$\begin{aligned}
\delta_2(f(B_i),x) &= \delta_2(C_i,x) \\
&= \delta_2(\{c \mid c \in S_2, b \in B_i \text{ and } b \equiv c(M_1M_2)\},x) \\
&= \{d \mid d \in S_2, b \in B_i \text{ and } \delta_1(b,x) \equiv d(M_1M_2)\}. \\
f(\delta_1\ (B_i,x)) &= f(\{\delta_1(b,x) \mid b \in B_i\}) \\
&= \{d \mid d \in S_2,\ b \in B_i \text{ and } \delta_1(b,x) \equiv d(M_1M_2)\}
\end{aligned}$$

Thus the equivalence blocks for an S.P. partition on M_1 induce an equivalent machine on M_2.

Example 5.16

	i_0	i_1
1	1	2
2	3	4
3	1	2
4	3	4

M_1

	i_0	i_1
a	a	b
b	c	d
c	a	b
d	e	d
e	a	d

M_2

The composite machine for M_1 and M_2 is

	i_0	i_1
1a	1a	2b
2b	3c	4d
3c	1a	2b
4d	3e	4d
3e	1a	2d
2d	3e	4d

M_1M_2

Equivalence sets for the states of M_1 on M_2 are $1M_2 = \{a\}$, 2M2 = $\{b,d\}$, $3M_2 = \{c,e\}$, and $4M_2 = \{d\}$.

$\pi_1 = \{\overline{1,3}; \overline{2,4}\}$ is an S.P. partition on M_1.
The equivalence sets define function f to be

$$f:\{1,3\} \rightarrow \{a,c,e\}$$
$$f:\{2,4\} \rightarrow \{b,d\}.$$

Thus, $\pi_{1_{ec}} = \{\overline{a,c,e}; \overline{b,d}\}$. The state machines for π_1 and $\pi_{1_{ec}}$ can be seen to be equivalent.

	i_0	i_1
{1,3}	{1,3}	{2,4}
{2,4}	{1,3}	{2,4}

π_1

	i_0	i_1
{a,c,e}	{a,c,e}	{b,d}
{b,d}	{a,c,e}	{b,d}

$\pi_{1_{ec}}$

The equivalent partition on the composite machine is $\pi_c = \{\overline{1a,3c,3e}; \overline{2b,2d,4d}\}$.

Since $k_c\max = k_1+k_2-k_{M_1M_2}$
$= 2 + 3 - 3$
$= 2$ and $k_c = 1$, π_c is not the maximum common machine. However, the state to split in order to find the maximum common machine can be determined by examining equivalence collections.

The equivalence collection on M_2 for the identity partition $I_1 = \{\overline{1}; \overline{2}; \overline{3}; \overline{4}\}$ is $I_{1_{ec}} = \{\overline{a}; \overline{b,d}; \overline{c,e}; \overline{d}\}$.

State d occurs twice, in separate blocks, in $I_{1_{ec}}$. Thus by splitting state d to produce M_2', there will exist a partition on M_2' equivalent to the identity partition of M_1.

	i_0	i_1
a	a	b
b	c	d'
c	a	b
d	e	d'
d'	e	d'
e	a	d

M_2'

The new composite machine is

	i_0	i_1
1a	1a	2b
2b	3c	4d'
3c	1a	2b
4d'	3e	4d'
3e	1a	2d
2d	3e	4d'

M_1M_2'

The equivalence collection on M_2' for the identity partition $I_1 = \{\overline{1}; \overline{2}; \overline{3}; \overline{4}\}$ is $I_{1_{ec}} = \{\overline{a}; \overline{b,d}; \overline{c,e}; \overline{d'}\}$ and the equivalent partition on M_1M_2' is

$\pi_c' = \{\overline{1a}; \overline{2b,2d}; \overline{3c,3e}; \overline{4d'}\}$.

$k_c\text{max} = k_1 + k_2' - M_1M_2'$

$= 2 + 3 - 3$

$= 2$ and $k_c' = 2$

Thus, π_c' is a maximum common machine for M_1M_2'.

In this case we did not have to construct an implication graph for the composite machine M_1M_2 in order to determine which states to split. Instead an examination of the structure of the composite machine, as determined by the component machines, indicated which state to split.

5.6 S.P. Partition Properties on Asynchronous Machines

Tracey [67] has established conditions, which, if met, ensure a critical-race free assignment for asynchronous machines. In this section we present theory relating S.P. partition properties to Tracey's conditions. First, however, asynchronous sequential machine theory is introduced and Tracey's conditions summarized.

Background

The state transitions which occur in an asynchronous machine are not controlled by a clock pulse, as in a synchronous machine. Instead, transitions occur whenever one of the input variables changes. For an asynchronous machine to detect a change, the input variables must be continually present. Consequently, an asynchronous machine will continue to change states until it reaches a stable state.

Definition 5.12 A state s_a is a stable state, under input i_k, if $\delta(s_a, i_k) = s_a$.

In the tabular representation of an asynchronous machine, stable states will be circled.

Example 5.17

	i_1	i_2	i_3	i_4
1	①	2	3	①
2	1	②	3	②
3	③	5	③	4
4	1	2	④	④
5	3	⑤	⑤	4

M

An asynchronous machine for which no input change leads to a sequence of state transitions is known as a normal mode machine. Thus, for this example, M is a normal mode machine.

Definition 5.13 If during the transition from state s_a to s_c, the state variables may assume the binary values associated with state s_b, s_b is said to be an intermediate state between states s_a and s_c.

If two-transitions share a common intermediate state, erroneous results may occur during either transition.

Definition 5.14 · For transitions $\delta(s_b,i_k) = s_d$ and $\delta(s_b,i_k) = s_e$, $s_d \neq s_e$, a critical race exists if the two transitions share a common state.

State assignments for which all the state variables that are required to change can do so simultaneously, without critical races, are called single transition time (S.T.T.) assignments. Single transition time assignments which have a unique coding for each state are called unicode single transition time (U.S.T.T.) assignments.

The following definition establishes terminology necessary to state the conditions for a unicode single transition time assignment, on a normal mode table.

Definition 5.15 Let X and Y be disjoint subsets of the states of a machine. The pair (X,Y) is called a partial state dichotomy (or partial dichotomy) of the machine.

A state variable y_i covers a partial dichotomy (U,V), if $y_i = 0\ \forall s_k \in U$ and $y_i = 1\ \forall s_j \in V$. For the transitions $\delta(s_a,i_j) = s_b$ and $\delta(s_c,i_j) = s_d$, if $(s_a s_b, s_c s_d)$ is a partial dichotomy, then the partial dichotomy $(s_a s_b, s_c s_d)$ is said to be associated with the transitions. (When the sets U and V contain only one or two states the notation $(u_i u_j, v_i v_j)$, without the set parenthesis, will be used to indicate a partial dichotomy.)

The necessary and sufficient conditions for a unicode single transition time assignment for a normal mode table, established by Tracey, are:

(1) The partial dichotomy associated with every pair of transitions occurring in any column of the table should be covered by some state variable.

(2) There must be a unique coding for every state.

Properties relating S.P. partitions to partial dichotomies are presented in the following section.

Properties

The properties of S.P. partitions established provide means of determining the partial dichotomies that can or cannot be covered by an S.P. partition. Theorem 5.6 can be used to determine some of the partial dichotomies that cannot be covered by a particular S.P. partition.

Theorem 5.6 For an S.P. partition $\pi = \{\overline{B_1};\ldots;\overline{B_p}\}$, if $s_i \equiv s_k(\pi)$ and $\delta(s_i,i_e) = s_j$ and $\delta(s_k,i_e) = s_m$, then there is no two-block partition ρ, $\rho \geq \pi$, which will cover the partial dichotomies associated with the transitions $\delta(s_i,i_e) = s_j$ and $\delta(s_k,i_e) = s_m$.

Proof. Since $s_i \equiv s_k(\pi)$ and π is an S.P. partition, $s_j \equiv s_m(\pi)$. For a two-block partition ρ, $\rho \geq \pi$, to cover the partial dichotomy (s_is_j, s_ks_m), we must have $s_i \equiv s_j \not\equiv s_k \equiv s_m(\rho)$

However, since $s_i \equiv s_k(\pi)$ and $\rho \geq \pi$, $s_i \equiv s_k(\rho)$.

Therefore, there is no two-block partition ρ, $\rho \geq \pi$, which covers the partial dichotomies associated with the transitions $\delta(s_i,i_e) = s_j$ and $\delta(s_k,i_e) = s_m$, when $s_i \equiv s_k(\pi)$.

In the following example Theorem 5.6 is used to determine the partial dichotomies which cannot be covered by a particular S.P. partition.

Example 5.18

	i_1	i_2	i_3	i_4	i_5
1	①	4	①	2	①
2	②	5	3	②	1
3	③	6	③	2	1
4	1	④	④	5	④
5	2	⑤	6	⑤	4
6	3	⑥	⑥	5	4

M

The S.P. partitions for M are

$\pi_1 = \{\overline{1,2,3}; \overline{4,5,6}\}$

$\pi_2 = \{\overline{1,3}; \overline{2}; \overline{4,6}; \overline{5}\}$

$\pi_3 = \{\overline{1,4}; \overline{2,5}; \overline{3,6}\}$

$\pi_4 = \{\overline{1,3,4,6}; \overline{2,5}\}$

$\pi_5 = \{\overline{1}; \overline{2,3}; \overline{4}; \overline{5,6}\}$

$\pi_6 = \{\overline{1,4}; \overline{2,3,5,6}\}$

The partial dichotomy (14,36) is associated with the transitions $\delta(1,i_2) = 4$ and $\delta(3,i_2) = 6$. We now try to find a two-block partition ρ, $\rho \geq \pi$, which covers (14,36). In order for ρ to cover (14,36) blocks (1,3) and (4,6) of π_2 must be joined. However, this also unites states 3 and 6, which must be kept separate if ρ is to cover (14,36). Thus, there is no two-block partition ρ, $\rho \geq \pi_2$, which covers (14,36).

Since $1 \equiv 3$ (π_2), Theorem 5.6 indicates immediately that there is no two-block partition, greater than π_2, which covers the partial dichotomy (14,36).

When realizing an S.P. partition on an asynchronous machine, not all of the partial dichotomies need be examined to determine whether they could be covered by the S.P. partition. Instead, it is only necessary that representative states from the blocks of the partition be examined.

<u>Theorem 5.7</u> For an S.P. partition π, $\pi = \{\overline{B_1};\ldots;\overline{B_n}\}$, let $s_{r_1}, s_{r_2} \in B_r$ and $s_{p_1}, s_{p_2} \in B_p$, where $B_r \not\equiv B_p$. The partial dichotomy $(s_{r_1} s_{j_1}, s_{p_1} s_{m_1})$ associated with the transitions $\delta(s_{r_1}, i_e) = s_{j_1}$ and $\delta(s_{p_1}, i_e) = s_{m_1}$ is covered by a two-block partition $\rho, \rho \geq \pi$, if and only if the partial dichotomy $(s_{r_2} s_{j_2}, s_{p_2} s_{m_2})$, associated with the transitions $\delta(s_{r_2}, i_e) = s_{j_2}$ and $\delta(s_{p_2}, i_e) = s_{m_2}$, is covered by ρ.

<u>Proof</u> Assume the partial dichotomy $(s_{r_1} s_{j_1}, s_{p_1} s_{m_1})$ is covered by ρ, where $\rho \geq \pi$.

i.e., $s_{r_1} \equiv s_{j_1} \equiv s_{p_1} \equiv s_{m_1}$ (ρ).

Since π is an S.P. partition and $s_{r_1} \equiv s_{r_2}$ (π), then $s_{j_1} \equiv s_{j_2}$ (π).

Similarly, $s_{p_1} \equiv s_{p_2}$ (π) and $s_{m_1} \equiv s_{m_2}$ (π). Since $\rho \geq \pi$,

$$s_{r_2} \equiv s_{r_1} \equiv s_{j_1} \equiv s_{j_2} \ (\rho) \text{ and } s_{p_2} \equiv s_{p_1} \equiv s_{m_1} \equiv s_{m_2} \ (\rho).$$

Thus, $s_{r_2} \equiv s_{j_2} \not\equiv s_{p_2} \equiv s_{m_2}$ (ρ) and the partial dichotomy $(s_{r_2} s_{j_2}, s_{p_2} s_{m_2})$ is covered by ρ.

Similarly, if $(s_{r_2} s_{j_2}, s_{p_2} s_{m_2})$ is covered by ρ, it can be shown that $(s_{r_1} s_{j_1}, s_{p_1} s_{m_1})$ is also covered by ρ.

Theorem 5.7 reduces the amount of work that must be done to determine whether a partial dichotomy can be covered by an S.P. partition.

Example 5.19 For partition π_2 in Example 5.19, the partial dichotomy (12,45) associated with the transitions, $\delta(1,i_4) = 2$ and $\delta(4,i_4) = 5$, can be covered by a two-block partition ρ, $\rho \geq \pi_2$, where $\rho = \{\overline{1,2,3};\ \overline{4,5,6}\}$.

Another partial dichotomy (32,65) associated with the transitions $\delta(3,i_4) = 2$ and $\delta(6,i_4) = 5$, occurs under input i_4.

Since $1 \equiv 3(\pi_2)$ and $4 \equiv 6(\pi_2)$, we have by Theorem 5.7 that the partial dichotomy (32,65) is also covered by ρ.

The next theorem makes use of the π-image concept for a machine introduced in Section 2.2. The π-image of a machine is the submachine defined by an S.P. partition π on the machine. Examination of the partial dichotomies formed by the π-image of a machine permits the determination of the partial dichotomies that could be covered by the S.P. partition π.

Theorem 5.8 All **partial dichotomies** $(s_r s_p, s_k s_m)$ under input i_e, where $s_r \in B_r$, $s_p \in B_p$, $s_k \in B_k$, and $s_m \in B_m$, are covered by a two-block partition $\rho, \rho \geq \pi$, if and only if there is a partial dichotomy on the blocks of π, $(B_r B_p, B_k B_m)$, associated with the transitions $\delta_\pi(B_r, i_e) = B_p$ and $\delta_\pi(B_k, i_e) = B_m$.

Proof (i) Assume that there is a partial dichotomy $(B_r B_p,\ B_k B_m)$ associated with the transitions $\delta_\pi(B_r, i_e) = B_p$ and $\delta_\pi(B_k, i_e) = B_m$. Form a partition π' of the blocks of π putting B_r and B_p in the same block B' and B_k and B_m in an opposing block B". Next form a two-block partition ρ of π', keeping B' and B" in different blocks. $\rho \geq \pi' \geq \pi$.

For a partial dichotomy $(s_r s_p, s_k s_m)$ under input i_e, where $s_r \in B_r$, $s_p \in B_p$, $s_k \in B_k$, and $s_m \in B_m$, we have
$s_r, s_p \in B'$ and $s_k, s_m \in B''$.
$s_r \equiv s_p \not\equiv s_k \equiv s_m(\rho)$.

Therefore, $(s_r s_p, s_k s_m)$ is covered by ρ, where $\rho \geq \pi$.

(ii) Assume $(s_r s_p, s_k s_m)$ is covered by a two block partition ρ, where $\rho \geq \pi$.

Assume also that $(B_r B_p, B_k B_m)$ is not a partial dichotomy.

That is (i) $B_r = B_k$, or

(ii) $B_r = B_m$, or

(iii) $B_p = B_k$, or

(iv) $B_p = B_m$.

For (i) $B_r = B_k$, we have $s_r \equiv s_k(\pi)$.

Therefore, there does not exist a two-block partition ρ, $\rho \geq \pi$, which covers the partial dichotomy $(s_r s_p, s_k s_m)$. Similar contradictions occur for (ii), (iii), and (iv).

Therefore, the partial dichotomy $(s_r s_p, s_k s_m)$, associated with the input i_e, is covered by ρ, $\rho \geq \pi$, if and only if $(B_p B_r, B_k B_m)$ is a partial dichotomy associated with the input i_e for machine M_π.

<u>Example 5.20</u> Consider the π_5-image for machine M in Example 5.18. $\pi_5 = \{\overline{1}; \overline{2,3}; \overline{4}; \overline{5,6}\}$.

		i_1	i_2	i_3	i_4	i_5
{1}	a	ⓐ	c	ⓐ	b	ⓐ
{2,3}	b	ⓑ	d	ⓑ	ⓑ	a
{4}	c	a	ⓒ	ⓒ	d	ⓒ
{5,6}	d	b	ⓓ	ⓓ	ⓓ	c

M_{π_5}

The partial dichotomy (ac,bd) is associated with the transitions $\delta_{\pi_5}(a,i_2) = c$ and $\delta_{\pi_5}(b,i_2) = d$. By Theorem 5.8, the partial dichotomies (14,25) and (14,36), associated with input i_2, are covered by a two-block partition ρ, $\rho \geq \pi_5$.

Choosing $\rho = \{\overline{a,c}; \overline{b,d}\} = \{\overline{1,4}; \overline{2,3,5,6}\}$, we obtain a two-block partition which covers the partial dichotomies (14,25) and (14,36).

For machine M, the partial dichotomy (12,45) associated with the transitions $\delta(1,i_4) = 2$ and $\delta(4,i_4) = 5$ is covered by a two=block partition ρ, $\rho \geq \pi_5$, where $\rho = \{\overline{1,2,3}; \overline{4,5,6}\} = \{\overline{a,b}; \overline{c,d}\}$.

By Theorem 5.8, since $1 \epsilon a$, $2 \epsilon b$, $4 \epsilon c$, and $5 \epsilon d$, (ab,cd) is a partial dichotomy, associated with the transitions $\delta_{\pi_5}(a,i_4) = b$ and $\delta_{\pi_5}(c,i_4) = d$, for machine M_{π_5}. Examination of M_{π_5} confirms this.

Any parital dichotomies covered by a partition π_1 will also be covered by a partition π_2, where $\pi_2 < \pi_1$.

Lemma 5.6 For two S.P. partitions π_1 and π_2, where $\pi_1 > \pi_2$, the set of partial dichotomies satisfied by π_1 is a proper subset of the partial dichotomies covered by π_2.

Proof If (x_1x_2, y_1y_2) is a partial dichotomy covered by π_1, then there is a two-block partition ρ, $\rho \geq \pi_1$, such that
$x_1 \equiv x_2 \not\equiv y_1 \equiv y_2(\rho)$.

Since $p > \pi_1 > \pi_2$, π_2 also covers the partial dichotomy (x_1x_2, y_1y_2).

The converse, that every partial dichotomy covered by π_2 is covered by π_1, is obviously not true. Thus,the set of partial dichotomies covered by π_1 is a proper subset of the partial dichotomies covered by π_2.

5.7 Discussion

Two possible methods of extending S.P. partition theory to incompletely specified machines, weak and extended substitution property partitions, have been presented in this chapter. (We shall refer to weak and extended substitution property partitions collectively as S.P. partitions for incompletely specified machines.) In both cases, the definition of S.P. partitions parallels Hartmanis and Stearns' [31] definitions of partition pairs for incompletely specified machines. Unfortunately, S.P. theory does not adapt as easily to incompletely specified machines as partition pair theory. The problems encountered have been illustrated by a detailed examination of weak substitution property partitions. Because of these difficulties, it has been necessary to define a new operator for deriving S.P. partitions. Fortunately, the new operator can also be used to find the greatest lower bound of two S.P. partitions.

A major problem associated with weak partition pairs is that their sum is not necessarily a weak partition pair. This problem was solved by the extended partition pair. However, the extended substitution property partition is not able to overcome this problem for S.P. partitions on incompletely specified machines. Thus, because of the extra set of labels required for extended substitution property partitions, weak substitution property partitions are simpler to use.

The problem associated with obtaining the least upper bound of S.P. partitions for incompletely specified machines, also makes the economic decomposition of the machine difficult. The difficulty arises because it is not always possible to factor out a common submachine for two S.P. partitions whose sum is not I. To bridge the gap between a strictly parallel and a "cascaded series-parallel" decomposition, a decomposition used by Hartmanis has been adapted to this special case. This decomposition, combining features of parallel and serial decompositions, has been shown to be a useful tool for the economical decomposition of incompletely specified machines.

The synthesis of multiple sequential machines has also been investigated. As might be expected, the structure of a composite machine has been shown to be dependent upon the structure of the component machines. Thus, the determination of common machine factors can be accomplished by an examination of the structure of the component machines, without the necessity of constructing an implication graph for the composite machine.

A brief examination of the decomposition of asynchronous machines has been presented. The result of which is a set of properties that simplify the determination of single transition time assignments. These properties have immediate application to the work of Tan, Menon, and Friedman [65] on the decomposition of asynchronous sequential machines.

Chapter 6 Sequential Machine Synthesis with Cellular Arrays

6.1 Introduction

The previous chapters have been concerned with various aspects of the decomposition of sequential machines. It was observed that decomposition is a tool that can be used to simplify sequential machine realization. However, for decomposed and undecomposed machines, the problem of finding a hardware implementation still remains.

Present integrated circuit technology permits the fabrication of large complex circuits on single chips. Obviously, for the manufacturing process, it is more economical to produce one standard chip, which can later be modified, than to produce an assortment of chips, each for different applications. One possible standard chip would consist of simple cells connected in an array configuration.

This approach has been investigated extensively for combinational logic. (Minnick [55] provides a comprehensive review of this research.) Only two papers, Ferrari and Grasselli [15] and Hu [33] have investigated the use of cellular arrays for sequential machine realization.

In this chapter the cellular array and synthesis technique developed by Hu is examined. A new synthesis technique for Hu's array is then presented. In addition, a new cellular array which overcomes some of the limitations of Hu's array is introduced.

The main characteristics of a cellular array are:

(i) cells must be identical;

(ii) cells must be simple; that is, each cell must contain only a few bits of memory and a small amount of combinational logic;

(iii) intercell connections must be regular except for final synthesis connections.

These characteristics require that a simple, regular model of sequential machine operation be used for cellular array synthesis. The state diagram and state table models are too complex to be implemented directly by cellular arrays.

The model Hu bases his cellular array on, is the transition matrix. A transition matrix, T, for a machine M is an $n \times n$ matrix which represents the transitive relationship between each pair of states.

Example 6.1

	i_0	i_1	z_0
1	5	2	0
2	6	6	0
3	1	5	1
4	2	4	0
5	3	1	1
6	4	3	0

M

For the transition matrix, T, the entry at the intersection of row r and column c is i_k if $\delta(s_r, i_k) = s_c$; otherwise, the entry is 0.

	1	2	3	4	5	6
1	0	i_1	0	0	i_0	0
2	0	0	0	0	0	i_0+i_1
3	i_0	0	0	0	i_1	0
4	0	i_0	0	i_1	0	0
5	i_0	0	i_1	0	0	0
6	0	0	i_1	i_0	0	0

T

In some cases, a transition matrix is still too complex to be realized directly by an array of simple cells. To overcome this problem, Hu uses i_h-transition matrices, which show the transitive relationships between states, for input i_h. In the following section we will examine i_h-transition matrices and Hu's synthesis technique.

6.2 Hu's Cellular Synthesis Method

The cellular array developed by Hu for sequential machine realization is based on the transition matrix concept. In order to keep the cells simple, i_h-transition matrices are used. An i_h-transition matrix, T_{i_h}, for a sequential machine, specifies the next-state relations for the input i_h.

Example 6.2

	i_1	i_2	i_3	i_4	z_1	z_2
1	3	1	4	2	0	1
2	1	5	4	2	0	0
3	3	2	2	5	0	1
4	4	1	4	2	1	1
5	5	4	3	5	1	0

M

The i_h-transition matrix, $h = 1,\ldots,4$ for M is constructed by inserting i_h at $T_{i_h}(j,k)$ if $\delta(s_j,i_h) = s_k$.

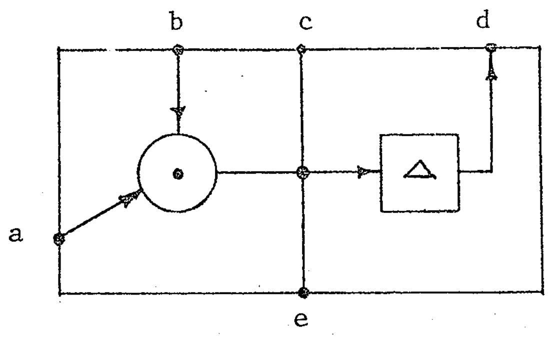
$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 0 & i_1 & 0 & 0 \\ 2 & i_1 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & i_1 & 0 & 0 \\ 4 & 0 & 0 & 0 & i_1 & 0 \\ 5 & 0 & 0 & 0 & 0 & i_1 \end{array} \qquad \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & i_2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & i_2 \\ 3 & 0 & i_2 & 0 & 0 & 0 \\ 4 & i_2 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & i_2 & 0 & 0 \end{array}$$

T_{i_1} $\qquad$ T_{i_2}

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 0 & 0 & i_3 & 0 \\ 2 & 0 & 0 & 0 & i_3 & 0 \\ 3 & 0 & i_3 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & i_3 & 0 \\ 5 & 0 & 0 & i_3 & 0 & 0 \end{array} \qquad \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & i_4 & 0 & 0 & 0 \\ 2 & 0 & i_4 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & i_4 \\ 4 & 0 & i_4 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & i_4 \end{array}$$

T_{i_3} $\qquad$ T_{i_4}

The cell, Figure 6.1, developed by Hu is capable of realizing the state transitions represented at the row and column intersections of the i_h-transition matrices. Thus, with the proper inter-cellular connections, an n × n array of cells can realize the behaviour of an i_h-transition matrix.



Hu's cell

Figure 6.1

Hu's synthesis procedure consists of determining the connections that must be made between cells in order that an array realize an i_h-transition matrix. Rather than present the synthesis procedure, we will describe the operation of an array which realizes a simple machine. From this description, and accompanying diagram, the necessary steps in the synthesis procedure will be clear.

<u>Example 6.3</u>

	i_0	i_1	z_1
1	2	1	0
2	1	2	1

M

The i_h-transition matrices are

$$\begin{array}{c|cc} & 1 & 2 \\ \hline 1 & 0 & i_0 \\ 2 & i_0 & 0 \end{array} \qquad \begin{array}{c|cc} & 1 & 2 \\ \hline 1 & i_1 & 0 \\ 2 & 0 & i_1 \end{array}$$

T_{i_0} $\qquad$ T_{i_1}

The cellular array in Figure 6.2 realizes M.

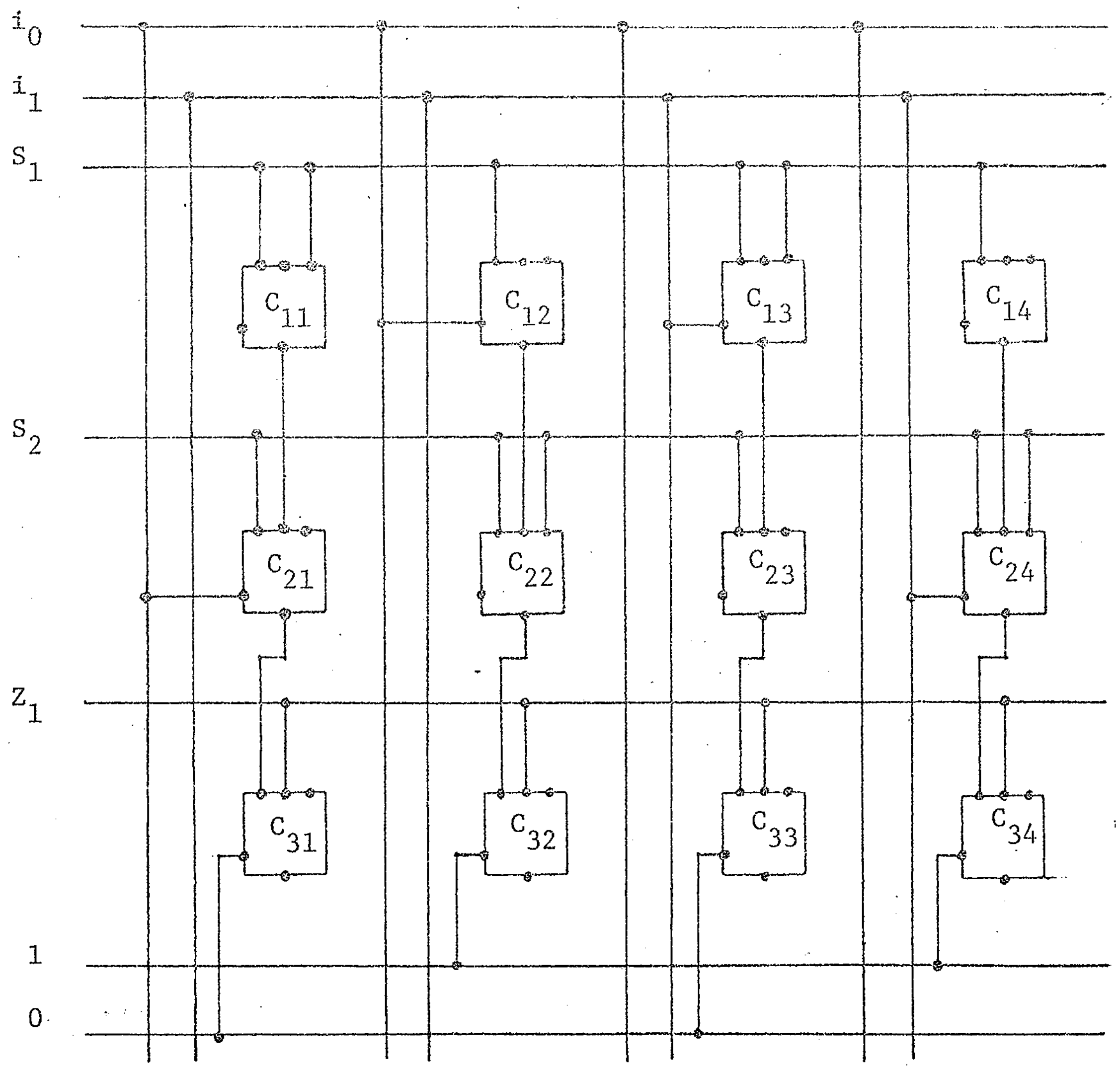


Figure 6.2

In addition to realizing next-state behaviour, the cellular array also realizes the output functions.

The S_1 and S_2 lines carry present-state information to the cells; lines 0 and 1 are used in determining the output function, which is transmitted by line Z_1. Only one cell will have its delay "on" at any time. The "on" delay will enable one of the S_j lines, providing the present-state information.

The array operation is demonstrated by sample state transitions.

(i) Assume the delay in cell C_{22} is "on" (state 2). If input i_0 is applied, the AND gate in C_{21} is enabled, which will set the delay in C_{11} to "on" (state 1). The AND gate in C_{31} is not enabled, so the output Z_1 is 0. (If the a terminal of C_{31} has been connected to the 1 line, Z_1 would have had a 1 output.)

(ii) Assume the delay in C_{11} is "on" (state 1). Applying input i_1 will enable the AND gate in C_{13} and set the delay in C_{13} to "on" (state 1). Output Z_1 remains at 0.

The remaining two state transitions for M can be inferred in a similar fashion.

In the above example, one cellular array was used for each i_h-transition matrix. This one-to-one relationship does not hold true for all sequential machines. A reduction in the number of cellular arrays needed can be effected by making use of compatible inputs.

<u>Definition 6.1</u> For a machine M, two inputs i_j and i_k are said to be <u>compatible</u> ($i_j \sim i_k$) if and only if

$$\delta(s_h,i_j) \neq \delta(s_h,i_k) \quad \text{for} \quad h=1,\dots,n \quad \text{and} \quad j \neq k.$$

<u>Example 6.4</u>

	i_0	i_1	z_1
1	1	4	0
2	3	4	1
3	5	2	1
4	3	4	0
5	1	2	0

M

$$i_0 \sim i_1 \quad \text{since} \quad \delta(x,i_0) \neq \delta(x,i_1), \text{ for } x=1,\dots,5 .$$

Combining the i_0 and i_1-transition matrices for M, it can be seen that there is at most one input, i_0 or i_1, at any row and column intersection.

	1	2	3	4	5
1	i_0	0	0	i_1	0
2	0	0	i_0	i_1	0
3	0	i_1	0	0	i_0
4	0	0	i_0	i_1	0
5	i_0	i_1	0	0	0

$T_{i_0} + T_{i_1}$

Thus the next state behaviour of M for the compatible inputs, i_0 and i_1, could be realized with just one cellular array.

<u>Definition 6.2</u> A set of inputs, $X = \{x_i | i = 1, ..., p\}$ is an <u>input compatible set</u> if $x_k \sim x_j$, for all $x_j, x_k \in X$.

<u>Definition 6.3</u> An input compatible set, X, is a <u>maximal input compatible set</u> if there is no other input compatible set, Y, such that $X \subset Y$.

A sequential machine can be realized from a collection of maximal input compatible sets which cover all the m inputs of M. Hu's algorithm for determining a minimal cover of input compatible sets follows.

Step 1: Derive the input compatible pairs using an input compatibility table.

Step 2: Obtain the maximal input compatible sets from the input compatible pairs.

Step 3: Derive a minimal cover from the maximal input compatible sets.

The algorithm is illustrated using machine M of Example 6.2.

<u>Example 6.5</u> The input compatibility table has an entry for each pair of inputs; an X entry indicates incompatibility, while a √ entry indicates compatibility.

i_2	√		
i_3	X	X	
i_4	X	√	√
	i_1	i_2	i_3

The input compatible pairs are (i_1,i_2), (i_2,i_4), and (i_3,i_4). No further input compatible sets can be derived. Thus, a minimal cover must be determined from the pairs.

The only minimal cover possible is $\{(i_1,i_2), (i_3,i_4)\}$. Using the cover derived, M can be realized using 2 cellular arrays instead of 4.

In the following section, a new synthesis procedure for Hu's array is presented. As will be shown, the number of arrays required using the new procedure is less than or equal to the number required using Hu's synthesis procedure.

6.3 Cellular Synthesis

The synthesis method for Hu's cellular array developed in this section is based on incompatible inputs, rather than compatible inputs.

Definition 6.4 For a machine $M = (S,IP,OP,\delta,\lambda)$, two inputs i_j and i_k in IP are said to be incompatible with respect to state s $i_j \not\sim_s i_k$, if

$$\delta(s,i_j) = \delta(s,i_k).$$

Lemma 6.1 The relation $\not\sim_s$ is an equivalence relation on the set of inputs of a machine M.

Proof (i) $\not\sim_s$ is reflexive, $i_j \not\sim_s i_j$, since $\delta(s,i_j) = \delta(s,i_j)$.
(ii) If $i_j \not\sim_s i_k$ then $\delta(s,i_j) = \delta(s,i_k)$, which implies $i_k \not\sim_s i_j$. Thus, $\not\sim_s$ is symmetric.
(iii) If $i_j \not\sim_s i_k$ and $i_k \not\sim_s i_\ell$, then $\delta(s,i_j) = \delta(s,i_k) = \delta(s,i_\ell)$. Therefore, $i_j \not\sim_s i_\ell$ and $\not\sim_s$ is transitive.

Incompatible inputs with respect to a state s can be grouped to form sets of incompatible inputs.

Definition 6.5 A set of inputs $X_s = \{x_1,\ldots,x_\ell\}$, $X_s \subseteq IP$, is said to be an incompatible set with respect to s if

$$x_k \not\sim_s x_{k+1}, \quad k = 1, \ldots, \ell-1.$$

Since $\not\sim_s$ is an equivalence relation the above definition ensures that $x_j \not\sim_s x_i$ for any $x_j,x_i \in X_s$. Subsequently, all pairs do not have to be examined for incompatibility when determining incompatible sets.

<u>Definition 6.6</u> A <u>maximal incompatible set of inputs with respect to s is</u> an incompatible set with respect to s, which is not a proper subset of any other incompatible set with respect to s.

For any state there may be more than one maximal incompatible set.

<u>Example 6.6</u>

	i_1	i_2	i_3	i_4	i_5	i_6	i_7
1	1	5	8	7	4	6	4
2	1	4	4	4	3	5	5
3	2	6	6	2	5	4	6
4	6	5	8	2	1	3	7
5	3	7	5	4	2	2	8
6	5	2	3	1	6	1	1
7	4	8	3	5	7	8	2
8	3	2	1	1	8	7	3

M

Both $\{i_2,i_3,i_4\}$ and $\{i_6,i_7\}$ are maximal incompatible sets with respect to state 2.

<u>Notation</u>: Let G_{j_s}, $j=1,\ldots,r$, represent all the maximal incompatible sets of inputs with respect to s.

<u>Lemma 6.2</u> $G_{k_s} \cap G_{\ell_s} = \phi$ for $k \neq \ell$

<u>Proof</u> Assume $G_{k_s} \cap G_{\ell_s} \neq \phi$.

Since $\not\sim_s$ is an equivalence relation, $G_{k_s} \cup G_{\ell_s}$ is a maximal incompatible set of inputs with respect to s. This contradicts our assumption that the G_{j_s}, $j=1,\ldots,r$ are maximal incompatible sets. Thus, $G_{k_s} \cap G_{\ell_s} = \phi$, for $k \neq \ell$.

We will denote the largest maximal incompatible set of inputs for state s by G_s and the number of inputs in G_s by μ_s. Thus, for machine M of Example 6.6, $G_2 = \{i_2,i_3,i_4\}$ and $\mu_2 = 3$.

A new type of transition matrix, which can be implemented using Hu's cellular array, is now presented. A <u>single transition matrix</u> is a transition matrix for which there is, at most, one input entry at the intersection of each row and column. The input entries in a single transition matrix do not all have to be the same, nor represent all the state transitions for a particular input.

For machine M of Example 6.6, a possible single transition matrix follows.

$$\begin{array}{c|c|c|c|c|c|c|c|c|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline
1 & i_1 & 0 & 0 & i_5 & 0 & 0 & i_4 & 0 \\ \hline
2 & i_1 & 0 & i_5 & i_4 & i_7 & 0 & 0 & 0 \\ \hline
3 & 0 & i_1 & 0 & 0 & i_5 & 0 & 0 & 0 \\ \hline
4 & i_5 & i_4 & 0 & 0 & 0 & i_1 & i_7 & 0 \\ \hline
5 & 0 & 0 & 0 & 0 & i_3 & 0 & i_2 & i_7 \\ \hline
6 & i_6 & i_2 & i_3 & 0 & 0 & 0 & 0 & i_6 \\ \hline
7 & 0 & i_7 & i_3 & 0 & 0 & 0 & i_5 & i_2 \\ \hline
8 & i_3 & i_2 & i_7 & 0 & 0 & 0 & i_6 & 0 \\ \hline
\end{array}$$

<u>Lemma 6.3</u> A machine $M = (S, IP, OP, \delta, \lambda)$ is realized by a set of single transition matrices if and only if for every next-state transition $\delta(s,i)=t$, there is an i entry at (s,t) of one of the single transition matrices.

<u>Proof</u> The proof is obvious.

For the minimal synthesis of a machine M, using single transition matrices, we must determine the minimal number required to realize M.

<u>Notation</u>: Let $\mu = \max \{\mu_s | s \in S\}$.

<u>Theorem 6.1</u> The minimal number of single transition matrices required to realize a machine M is given by μ.

<u>Proof</u> (Proof by construction)

Assume that $\mu_s = \mu, s \in S$.

For the set $G_s = \{g_1,\ldots,g_{\mu_s}\}$, if $\delta(s,g_k) = r$, then insert input g_k at row and column intersection (s,r) of single transition matrix k, for $k=1,\ldots,\mu_s$.

For the remaining state table entries, insert i_ℓ at (x,y) of one of the single transition matrices if $\delta(x,i_\ell)=y$. If state transition $\delta(x,i_p) = y$ has not been entered into one of the state transition matrices and (x,y) of all the state transition matrices are non-zero, then

$$\delta(x,x_1) = \delta(x,x_2) = \ldots\ldots = \delta(x,x_{\mu_s}) = \delta(x,i_p) = y.$$

That is, $G_x = \{x_i,\ldots\ldots,x_{\mu_s},i_p\}$ is an incompatible set of inputs with respect to x and $\mu_x = \mu_{s+1}$. This contradicts the assumption that $\mu = \mu_s$.

Thus, all the remaining state table entries can be inserted into at least one of the single transition matrices and M can be realized by μ single transition matrices.

Cellular realizations of M in Example 6.6 are now given using Hu's minimal synthesis method and synthesis using incompatible inputs.

Example 6.7 First, using Hu's method, an input compatibility table is constructed.

i_2	✓					
i_3	✓	X				
i_4	X	X	X			
i_5	✓	✓	✓	✓		
i_6	✓	X	✓	X	X	
i_7	X	X	X	X	X	X
	i_1	i_2	i_3	i_4	i_5	i_6

The maximal input compatible classes are $\{i_7\},\{i_4,i_5\}$, $\{i_1,i_2,i_5\},\{i_1,i_3,i_5\}$, and $\{i_1,i_3,i_6\}$.

A minimal cover can be formed using the input compatible sets $\{i_1,i_2,i_5\},\{i_1,i_3,i_6\},\{i_4,i_5\}$ and $\{i_7\}$. Thus, four 8 × 8 cellular arrays will be required to realize M. A simpler cover $\{i_1,i_3,i_6\},\{i_2\}$, $\{i_4,i_5\}$, and $\{7\}$, not using the maximal input compatible classes, still requires 4 arrays.

Now determine the maximal input incompatible sets.

state	maximal incompatible sets
1	$\{i_5,i_7\}$
2	$\{i_2,i_3,i_4\},\{i_6,i_7\}$
3	$\{i_1,i_4\},\{i_2,i_3,i_7\}$
4	
5	$\{i_5,i_6\}$
6	$\{i_4,i_6,i_7\}$
7	$\{i_2,i_6\}$
8	$\{i_1,i_7\},\{i_3,i_4\}$

Thus $\mu = \mu_2 = \mu_3 = 3$ and M can be realized using three 8 × 8 cellular arrays. A possible STM realization is illustrated below.

	1	2	3	4	5	6	7	8
1	i_1				i_2		i_4	
2	i_1			i_2				
3		i_1				i_2		
4		i_4			i_2	i_1		
5				i_4			i_2	
6	i_4	i_2				i_5		
7				i_1	i_4			i_2
8		i_2	i_1					

	1	2	3	4	5	6	7	8
1				i_5		i_6		
2			i_5	i_3	i_6			
3					i_5	i_3		
4	i_5							i_3
5		i_5	i_1		i_3			
6	i_6		i_3					
7			i_3				i_5	i_6
8	i_3						i_6	i_5

	1	2	3	4	5	6	7	8
1				i_7				i_3
2				i_4	i_7			
3		i_4		i_6		i_7		
4			i_6				i_7	
5		i_6						i_7
6	i_7				i_1			
7		i_7						
8	i_4		i_7					

Single transition matrix realization of M

The cell connections necessary to realize a single transition matrix using Hu's cellular array are similar to those for an input-transition matrix, and as such will not be illustrated.

The final transition matrices used by Hu's synthesis technique are referred to as C matrices. In the previous example, it was possible to realize a machine using fewer single transition matrices than C matrices. It can be shown that the number of C matrices for a realization is an upper bound on the number of single transition matrices necessary.

Notation: Let c be the number of input compatible classes in the minimal cover for machine M.

Theorem 6.2 For a sequential machine M, $\mu \leq c$.

Proof Assume $\mu > c$. That is, the number of inputs in the largest maximal input compatible set is greater than c.

Let $B_j, j=1,\ldots,c$ represent the maximal input compatible classes in the minimal cover.

Since $c < \mu$, there must be at least two inputs $i_k, i_\ell \in B_m$ such that $i_k \not\sim_s i_\ell$ for some $s \in S$. Thus, $\delta(s,i_k) = \delta(s,i_\ell)$, which contradicts the assumption that B_m is a maximal compatible set.

Therefore, $\mu \leq c$.

Thus, input incompatibility synthesis will never require more cellular arrays than input compatibility synthesis. Also, since input incompatibility synthesis does not require the derivation of a minimal cover, it is simpler than input compatibility synthesis.

Input incompatibility synthesis can easily be applied to cellular arrays having more than one input terminal per cell. We will use the term incompatibility level, IL, to refer to the number of incompatible inputs that can be applied to a cell.

In general, the number of arrays required for input incompatibility synthesis is given by $[\mu/IL]$.

For machine M of Example 6.7, if a cell with IL = 2 is used, the number of cellular arrays required is

$$\begin{aligned}[\mu/IL] &= [3/2] \\ &= 2.\end{aligned}$$

6.4 State Splitting

Another technique for reducing the number of cellular arrays necessary to realize a machine, involves state splitting. In Chapter 5, state splitting was seen to be a valuable tool in the economical decomposition of machines. State splitting can also be used to reduce the number of incompatible inputs for a machine. This in turn can reduce the number of single transition matrices required in a realization.

Example 6.8

	i_1	i_2	i_3	i_4
1	3	7	2	5
2	1	8	1	6
3	1	6	5	1
4	2	5	5	5
5	2	4	8	4
6	1	3	7	6
7	4	4	7	2
8	3	3	8	1

M

Constructing the maximal input compatible sets with respect to s gives:

state	maximal incompatible set	μ_s
1	-	1
2	$\{i_1,i_3\}$	2
3	$\{i_1,i_4\}$	2
4	$\{i_2,i_3,i_4\}$	3
5	$\{i_2,i_4\}$	2
6	-	1
7	$\{i_1,i_2\}$	2
8	$\{i_1,i_2\}$	2

$\mu = \mu_4 = 3$

Consequently, 3 single transition matrices are required to realize M. Examining the set $\{i_2,i_3,i_4\}$ it can be seen that the input incompatibility is caused by state 5. By splitting state 5 to produce 5 and 5', the value of μ_4 can be lowered to 2.

	i_1	i_2	i_3	i_4
1	3	7	2	5
2	1	8	1	6
3	1	6	5	1
4	2	5	5	5'
5	2	4	8	4
5'	2	4	8	4
6	1	3	7	6
7	4	4	7	2
8	3	3	8	1

M'

The maximal input incompatible set with respect to 4 is now $\{i_2,i_3\}$. No new input incompatibilities have been introduced, so M' can be realized using 2 single transition matrices.

Splitting states necessitates using larger arrays to accommodate the extra states. For the above example, two 9 × 9 cellular arrays would be required. The number of cells is 2 × 9 × 9 = 162. Without state splitting, three 8 × 8 arrays are needed for a total of 3 × 8 × 8 = 192 cells.

Evaluating a realization by counting the number of cells used does not give an effective measure of the realization. For example one realization may use fewer cells than another realization, but require larger arrays than the logic designer has available.

Consequently, state splitting should be used to accommodate the realization to the cellular arrays available, rather than to minimize the number of cells.

Note that state splitting can also be applied to Hu's minimal synthesis method.

Example 6.9

	i_1	i_2
1	3	2
2	5	1
3	6	6
4	2	5
5	1	4
6	3	2

M

The maximal input compatible classes are $\{i_1\}$ and $\{i_2\}$, requiring two arrays to realize M. However, by splitting state 6 to give 6 and 6', i_1 and i_2 can be made compatible. Thus, the number of arrays needed is reduced to 1.

6.5 Problems with Hu's Cellular Array

Hu has proposed his cellular array as a general device for the synthesis of sequential machines. However, in order to use the arrays economically, they must be tailored to the sequential machine. That is, the number of state and output rows in a cellular array should not greatly exceed the number of states and outputs, respectively, in the machine. This requirement entails maintaining an inventory of various sized arrays.

The possibility of interconnecting arrays to form a larger array is ruled out, because the connections between all rows are not standard; the connection between two state rows differs from the connection between a state and output row. Thus, the largest machine that can be realized is determined by the size of the largest cellular array that is available.

This problem can be solved by removing all output rows from the array. Because all inter-row connections are now regular, any large machine can be synthesized from an inventory of small arrays. However, the output functions will now have to be realized external to the array.

Examining Hu's cellular array it is apparent that each output cell is utilized essentially as a two-input AND gate. Thus, each output function can be realized by connecting a row of two-input AND gates to the array.

There is another reason for not using the basic cell to realize the output functions; the inter-row connections permit only one output function to be realized for a sequential machine.

Example 6.10

	i_0	i_1	z_0	z_1
1	1	2	1	1
2	2	1	0	1

M

A realization using Hu's cellular array is given.

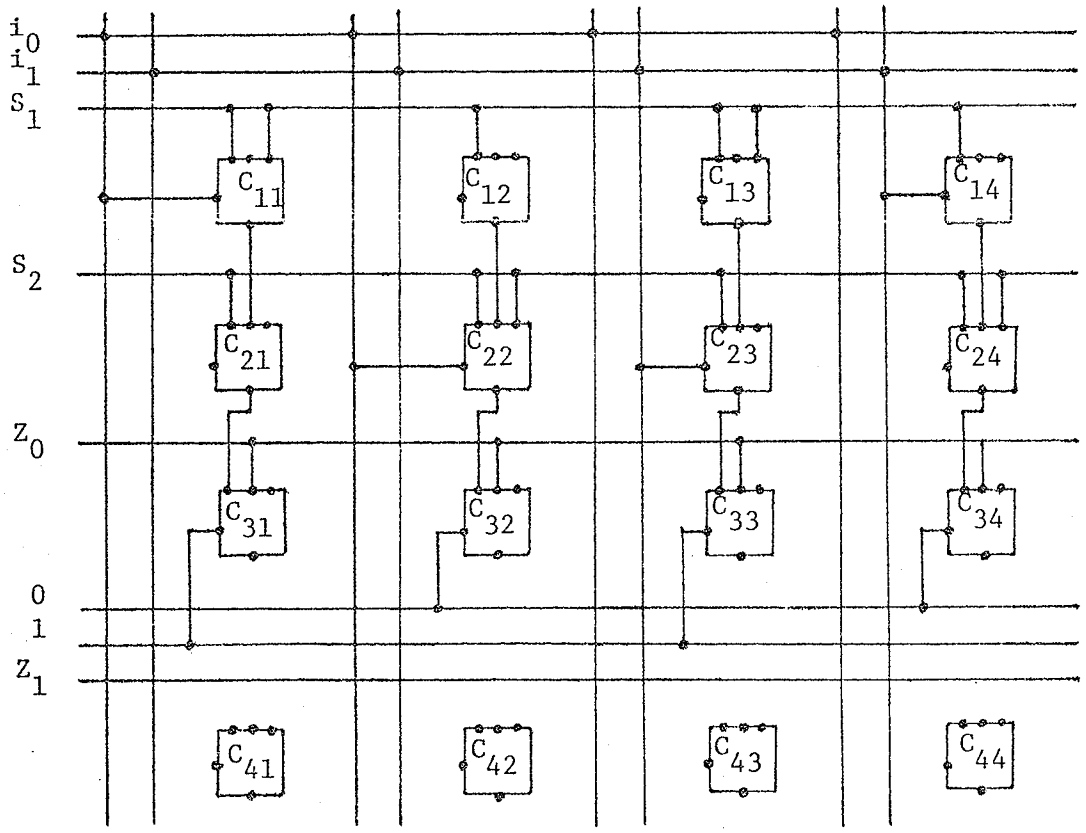


Figure 6.3

From Fig. 6.3 it can be seen that state information is not transmitted beyond the first output row (row 3). Thus row 4 does not receive any state information and cannot be used to realize output z_1.

A reduced array, without any output cells, can be used more effectively in realizing large sequential machines. Hu's minimal synthesis procedure and the procedures developed in Sections 6.3 and 6.4 are applicable to the reduced array. However, the realizations produced by all procedures cannot effectively use a standard cellular array.

In Example 6.7, Hu's procedure produces the cover $\{i_1,i_2,i_5\}$ $\{i_1,i_3,i_6\}$, $\{i_4,i_5\}$, and $\{i_7\}$. If a standard 8 × 8 cellular array is to be used, the array will require 3 input lines, even though only two arrays will fully utilize all the input lines. Similarly, the method of Section 6.3 will not necessarily produce single transition matrices which use an equal number of inputs. It is difficult to predict in advance the maximum number of input lines required for an array. Consequently, any inventory would have to contain "worst-case" cellular arrays.

6.6 A New Cellular Array

A new type of cellular array, which has been standardized to realize any sequential machine, is introduced. The evolution of the array from an early version is presented to illustrate the design and operation of the array.

The basis for the array is the $x\bar{x}$ transition matrix. For a binary input variable x, the $x\bar{x}$ transition matrix is a transition matrix which represents the state transitions that occur for input values x and $\bar{x}$.

Example 6.11

x_1x_2	00	01	11	10
1	3	1	4	2
2	1	5	4	2
3	3	2	1	5
4	5	1	4	2
5	5	4	3	5

M

For M, the $x\bar{x}$ transition matrices are

	1	2	3	4	5
1	$\bar{x}_1$	x_1	$\bar{x}_1$	x_1	
2	$\bar{x}_1$	x_1		x_1	$\bar{x}_1$
3	x_1	$\bar{x}_1$	$\bar{x}_1$		x_1
4	$\bar{x}_1$	x_1		x_1	$\bar{x}_1$
5			x_1	$\bar{x}_1$	$x_1+\bar{x}_1$

$x_1\bar{x}_1$

	1	2	3	4	5
1	x_2	$\bar{x}_2$	$\bar{x}_2$	x_2	
2	$\bar{x}_2$	$\bar{x}_2$		x_2	x_2
3	x_2	x_2	$\bar{x}_2$		$\bar{x}_2$
4	x_2	$\bar{x}_2$		x_2	$\bar{x}_2$
5			x_2	x_2	$\bar{x}_2$

$x_2\bar{x}_2$

A cell which can be used to implement the above $x\bar{x}$ transition matrix is illustrated in Fig. 6.4.

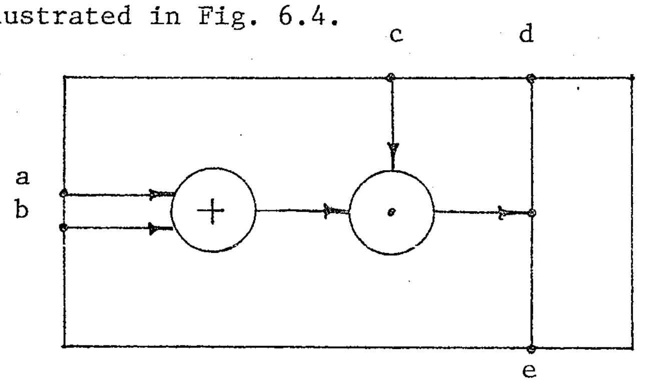


Figure 6.4

The method to realize an $x\bar{x}$ transition matrix using an array of cells of the type depicted in Fig. 6.4 is similar to Hu's synthesis method for a transition matrix. The main difference being that the method has been adapted to handle cells with two inputs. The cellular array in Figure 6.5 realizes the $x_1\bar{x}_1$ transition matrix of Example 6.11. The cellular interconnections are obvious from the transition matrix diagram.

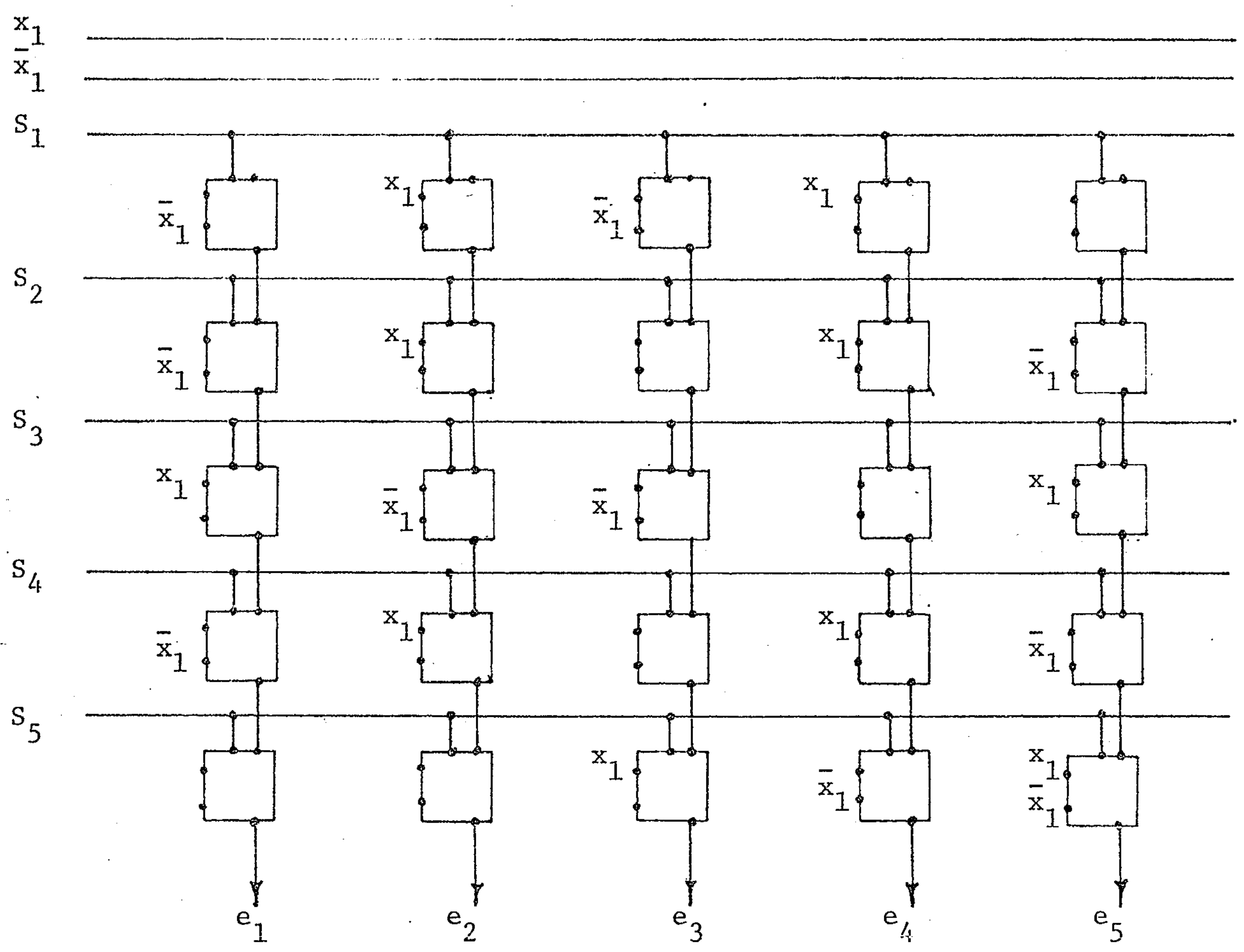


Figure 6.5

During the operation of this cellular array, more than one of the e_i,i=1,...,5, lines will be enabled for either input x_1 or $\bar{x}_1$. Essentially, the array indicates, for an input $x_1(\bar{x}_1)$ the possible next-states.

That is, if in state 3 with input x_1, the next-state is either 1 or 5. Similarly, the cellular array for the $x_2\bar{x}_2$ transition matrix indicates possible next-states for inputs x_2 and $\bar{x}_2$. For state 3 and input $\bar{x}_2$, the possible next-states are 3 and 5. In conjunction with the $x_1\bar{x}_1$ cellular array, the next-state for M in state 3 with input $\bar{x}_1x_2$ is 5. Thus, the next state can be found by ANDing the possible next-states. The next-state is then passed through a delay to give the present state, as illustrated in Figure 6.6.

in Figure 6.6

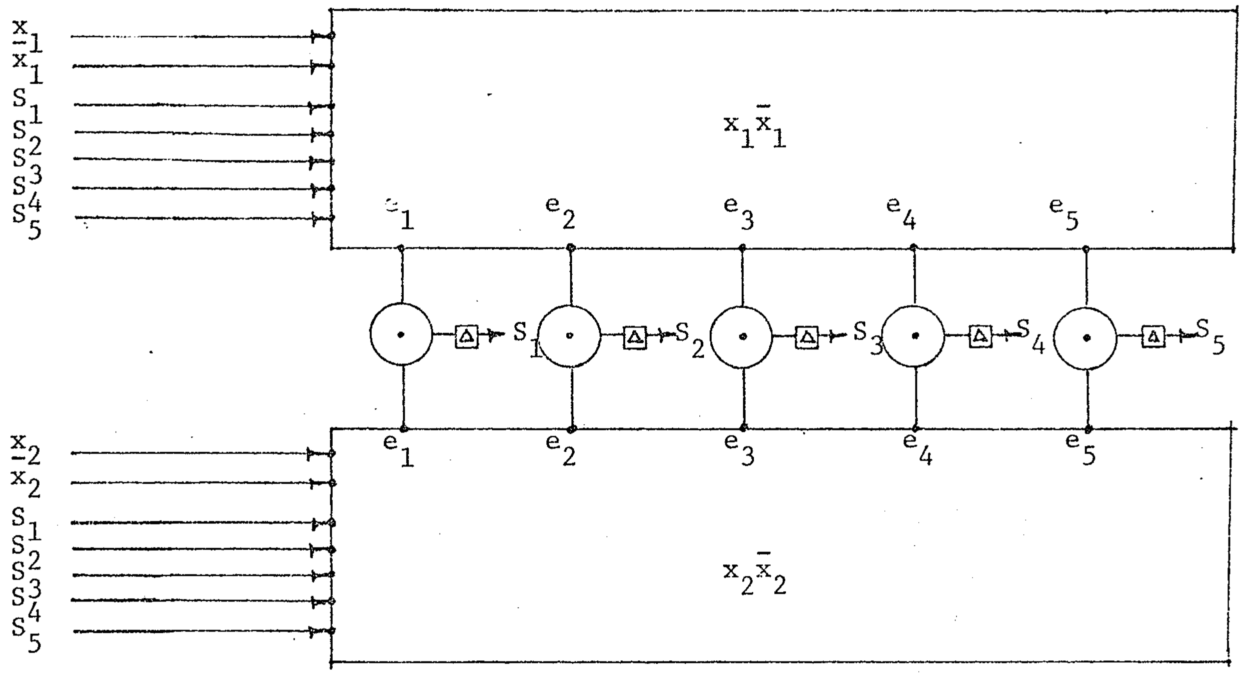


Figure 6.6

The cell of Figure 6.4 is not capable of realizing all sequential machines. A problem occurs when there are incompatible inputs that are the complement of each other. When this happens a cellular realization will select multiple next-states.

<u>Example 6.12</u>

$\underline{x_1x_2}$

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

M

The $x\bar{x}$ transition matrices are:

$$
\begin{array}{c|c|c|c|c|c|}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & \bar{x}_1 & x_1 & \bar{x}_1 & x_1 & \\
\hline
2 & \bar{x}_1 & x_1 & & x_1 & \bar{x}_1 \\
\hline
3 & & & x_1 & \bar{x}_1 & x_1,\bar{x}_1 \\
\hline
4 & \bar{x}_1 & x_1 & & x_1 & \bar{x}_1 \\
\hline
5 & & & x_1 & \bar{x}_1 & x_1,\bar{x}_2 \\
\hline
\end{array}
\qquad
\begin{array}{c|c|c|c|c|c|}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & x_2 & \bar{x}_2 & \bar{x}_2 & x_2 & \\
\hline
2 & \bar{x}_2 & \bar{x}_2 & & x_2 & x_2 \\
\hline
3 & & & x_2 & \bar{x}_2 & x_2,\bar{x}_2 \\
\hline
4 & x_2 & \bar{x}_2 & & x_2 & \bar{x}_2 \\
\hline
5 & & & x_2 & x_2 & \bar{x}_2 \\
\hline
\end{array}
$$

$$x_1\bar{x}_1 \qquad\qquad x_2\bar{x}_2$$

Examining the $x\bar{x}$ transition matrices it can be seen that if M is in state 3 and received input 00, columns 4 and 5 of both transition matrices are enabled. Using the cellular arrays of Figure 6.6, machine M would enter two next-states instead of just one. Similarly, when the input is 11 for state 3, columns 3 and 5 of both transition matrices are enabled.

The above problems occur as a result of the binary complemented inputs, 01 and 10, incompatibility with respect to state 3. That is, $\delta(3,01) = \delta(3,10) = 5$. These state transitions require that state 5 be the next-state in the $x\bar{x}$ transition matrices for all possible values of the inputs x_1 and x_2.

Similar problems would have occurred if the binary complemented inputs 00 and 11 had been incompatible with respect to any state s.

The problem of multiple next-states for binary complemented inputs is overcome by state-splitting.

Example 6.13. The binary complemented incompatible inputs of machine M in the previous example can be made compatible by splitting state 5.

x_1x_2	00	01	11	10
1	3	1	4	2
2	1	5	4	2
3	4	5	3	5'
4	5	1	4	2
5	5	4	3	5
5'	5	4	3	5

M'

It is only necessary to split a state once in order to remove all binary complemented input incompatibilites caused by that state. The next-state behaviour of the original state and the split-state can be made identical. This possibility will be used later when realizing M'. The new $x\bar{x}$ transition matrices are:

	1	2	3	4	5	5'
1	$\bar{x}_1$	x_1	$\bar{x}_1$	x_1		
2	$\bar{x}_1$	x_1		x_1	$\bar{x}_1$	
3			x_1	$\bar{x}_1$	$\bar{x}_1$	x_1
4	$\bar{x}_1$	x_1		x_1	$\bar{x}_1$	
5			x_1	$\bar{x}_1$	$x_1,\bar{x}_1$	
5'			x_1	$\bar{x}_1$	$x_1,\bar{x}_1$	

	1	2	3	4	5	5'
1	x_2	$\bar{x}_2$	$\bar{x}_2$	x_2		
2	$\bar{x}_2$	$\bar{x}_2$		x_2	x_2	
3			x_2	$\bar{x}_2$	x_2	$\bar{x}_2$
4	x_2	$\bar{x}_2$		x_2	$\bar{x}_2$	
5			x_2	x_2	$\bar{x}_2$	
5'			x_2	x_2	$\bar{x}_2$	

The $x_1\bar{x}_1$ and $x_2\bar{x}_2$ transition matrices realize M' without producing multiple next-states.

A modified cell which will allow any n state machine to be realized using n × n cellular arrays is presented.

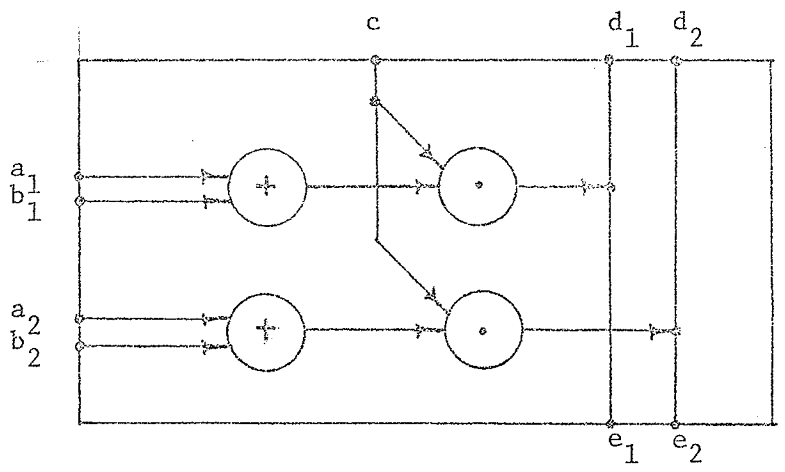


Figure 6.7

Essentially, the cell contains two circuits of the type found in the original cell. The top circuit can be used to produce the next-state s_j; while the bottom circuit can be used to produce the split next-state s'_j.

In a cellular realization of a machine M, the next-state is found by ANDing all the $e_1(e_2)$ outputs for state $s_j(s_j')$. Since, the next-state behaviour for states s_j and s_j' are identical, the present state S_j is determined by $S_j = s_j + s_j'$. Thus, it is not necessary to introduce another state when realizing machines with split states, as an n state machine can be realized from an $n \times n$ cellular array. The synthesis method presented demonstrates how this can be done.

<u>Synthesis Method for Modified Cell</u>

(1) For machine M, split all binary complemented incompatible inputs.

(2) $i \leftarrow 1$.

(3) Find the $x_i\bar{x}_i$ transition matrix.

(4) For the cellular array C_i:

- (a) $j \leftarrow 1$;
- (b) terminal c of each cell of row j is connected to the state line S_j;
- (c) terminal $d_1(d_2)$ of each cell in row j is connected to the $e_1(e_2)$ terminal of the cell above;
- (d) connect $x_i(\bar{x}_i)$ to terminal $a_i(b_i)$ in cell $C_i(j,k)$ if entry (j,k) in the $x_i\bar{x}_i$ transition matrix is $x_i(\bar{x}_i)$;

(e) connect $x_i(\bar{x}_i)$ to terminal $a_2(b_2)$ in cell $C_i(j,k)$ if entry (j,k') in the $x_i\bar{x}_i$ transition matrix is $x_i(\bar{x}_i)$;

(f) $j \leftarrow j+1$

if $j \leq n$, where n is the number of states, go to 4(c); otherwise go to 5;

(5) $i \leftarrow i+1$

if $i \leq m$, where m is the number of input variables, go to 3; otherwise go to 6;

(6) The corresponding $e_r, r=1,2$ terminals of all arrays are ANDed to produce the next-state values s_j and $s_j', j=1,\ldots,n$;

(7) The s_j and s_j' are then ORed and passed through a delay to give S_j.

Example 6.14 The $x_1\bar{x}_1$ transition matrix for Example 6.13 is realized using the new cellular array.

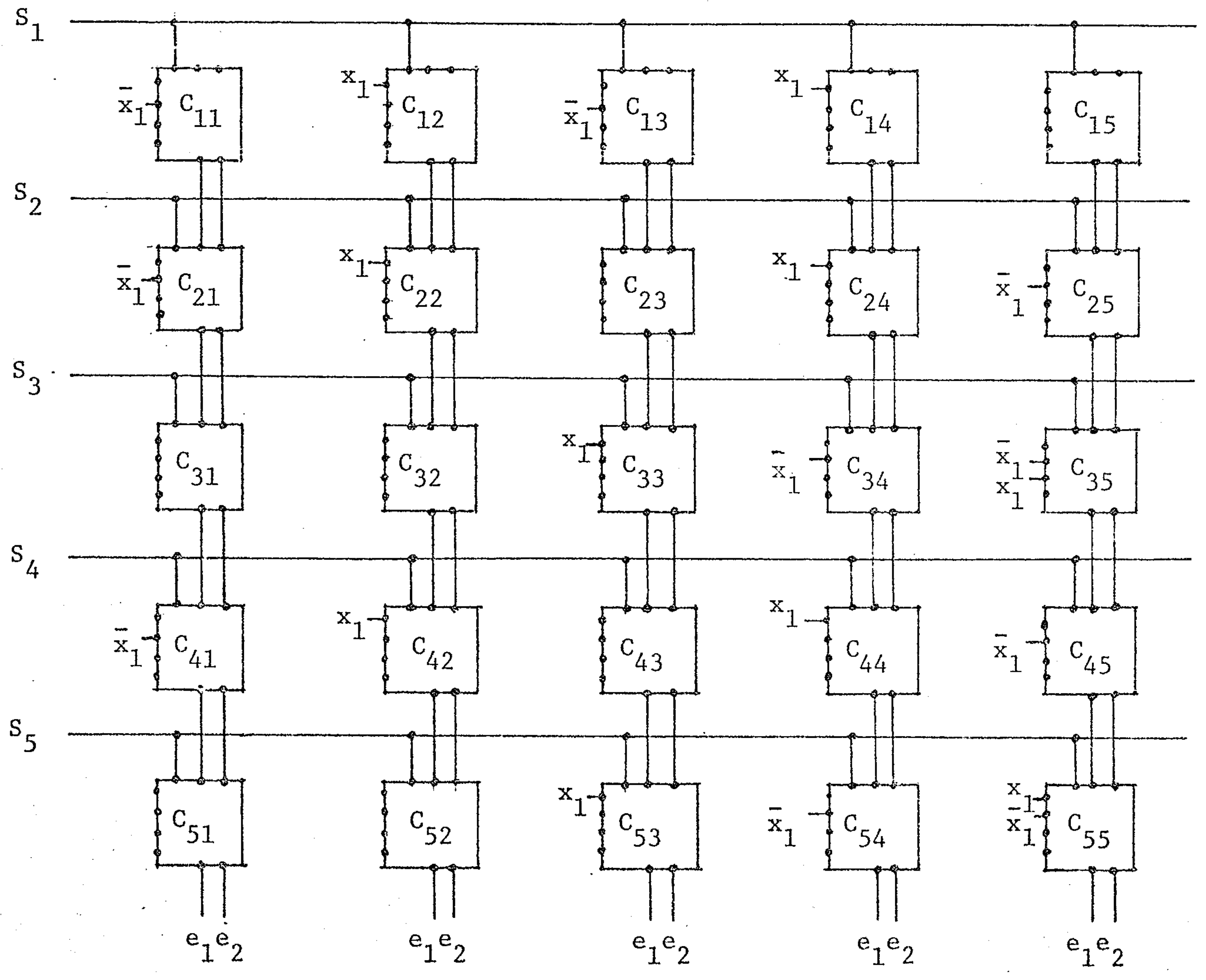


Figure 6.8

The $x_2\bar{x}_2$ cellular array can be implemented in a similar manner. Connecting the $x_1\bar{x}_1$ and $x_2\bar{x}_2$ cellular arrays togehter, following rules (6) and (7), a cellular realization for M is obtained.

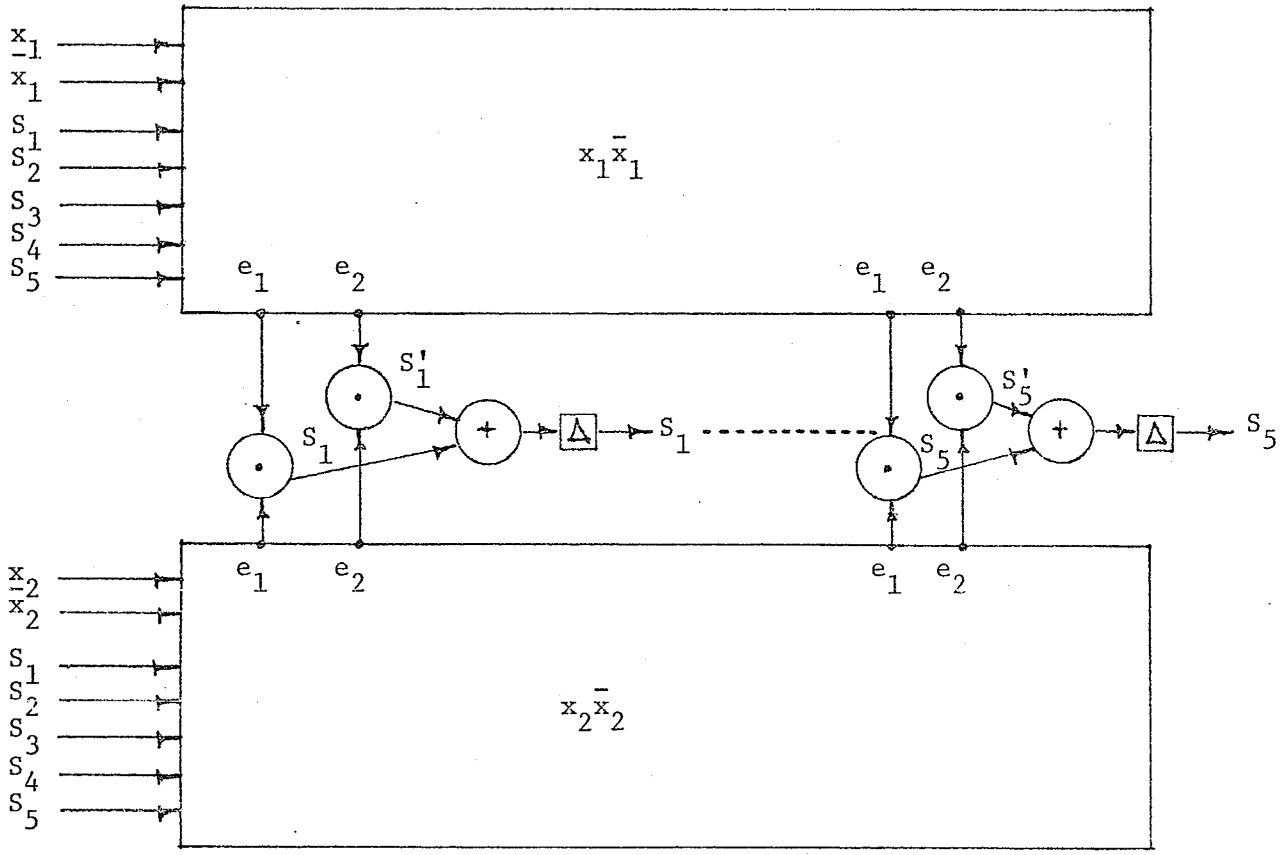


Figure 6.9

An important feature of the cellular array presented is that any machine can be realized from arrays having only two input lines. This permits the uniform interconnection of arrays to realize any size sequential machine.

The number of arrays to realize any machine is independent of input compatibility and can be determined prior to the realization. The number, N, required is given by the formula

$N = [\log_2 k]$, where k is number of inputs.

For a machine with 15 inputs, the number of cellular arrays would only be $[\log_2 15] = 4$. Thus, the number of arrays required grows at a relatively slow rate as the machine size increases.

6.7 Conclusion

The use of cellular arrays for sequential machine realization has been investigated in this chapter. The various arrays discussed all incorporate the transition matrix concept in their design. Differences in the arrays are due to the variant forms of transition matrices used; that is, input or input variable transition matrices.

The cellular array presented by Hu is designed to implement the state transitions from a single input. Thus an m input sequential machine would require m cellular arrays. However, minimal covers of compatible inputs are able, in most cases, to reduce the number of arrays required.

An alternative method of using Hu's array was also developed. This approach, instead of requiring all state transitions for a particular input to be represented in a single array, allowed the transitions for an input to be distributed among several arrays. It was proved that this method used fewer, or not more, arrays than Hu's method. In addition, this synthesis method ensured that the minimal cellular realization of a machine could be found in a straightforward manner, whereas Hu's method involved obtaining a minimal cover of compatible inputs. For a machine with a large number of inputs, the derivation of a minimal cover is an involved process.

There are several problems attendant upon the use of Hu's array for machine realization. First, the array cannot realize more than one output function; to realize even one output function requires irregular inter-cell connections. This problem can only be overcome by implementing the output functions external to the array and using the array only to realize state transition behaviour.

The second, more serious, problem concerns the standardization of cellular arrays. As the number of inputs for a machine increases, the number of input lines required for the cellular arrays also increases. Consequently, it is not possible to manufacture a standard array with a fixed number of lines. The ability to connect arrays together to form a larger array is reduced, as the resulting array will probably not have enough input lines to realize a large machine.

A solution to this problem uses the $x\bar{x}$ transition matrices developed in this chapter. The array presented to realize an $x\bar{x}$ transition matrix requires only two input lines, for any size transition matrix. Thus, a standard size array can be produced, which can easily be interconnected to realize any sequential machine.

Chapter 7 Machine Synthesis using Cascaded Combinational Logic Boxes

7.1 Introduction

A novel approach to the problem of sequential machine realization is presented in the work of Haring [23] and of Menger [53]. The underlying premise in both works is that a column in the state table representation of a sequential machine can be realized by a cascaded connection of logic boxes called generators. The essential difference between Haring and Menger lies in the set of generators used by each.

Haring uses the three generator set {a,b,c}, where the function performed by each generator is represented in Fig. 7.1.

	a	b	c
1	2	2	1
2	1	3	1
3	3	4	3
.	.	.	.
i	i	i+1	i
.	.	.	.
n-1	n-1	n	n-1
n	n	1	n

Haring's Generators
Figure 7.1

Notation: For an n state machine, the n^n possible state columns are referred to as maps. A map for which all possible n states occur is a permutation map, or simply a permutation; while a map for which some of the n states are missing is a non-permutation map.

An algorithm to obtain a generator realization for any map is developed by Haring using the operator set $\{\alpha,\beta,\gamma\}$. The operators α,β and γ corresponding to a,b, and c, respectively, manipulate states in specified positions. For example, α will interchange the states in positions 1 and 2 of the column, regardless of which states are in these positions. A graphical representation of α,β, and γ follows.

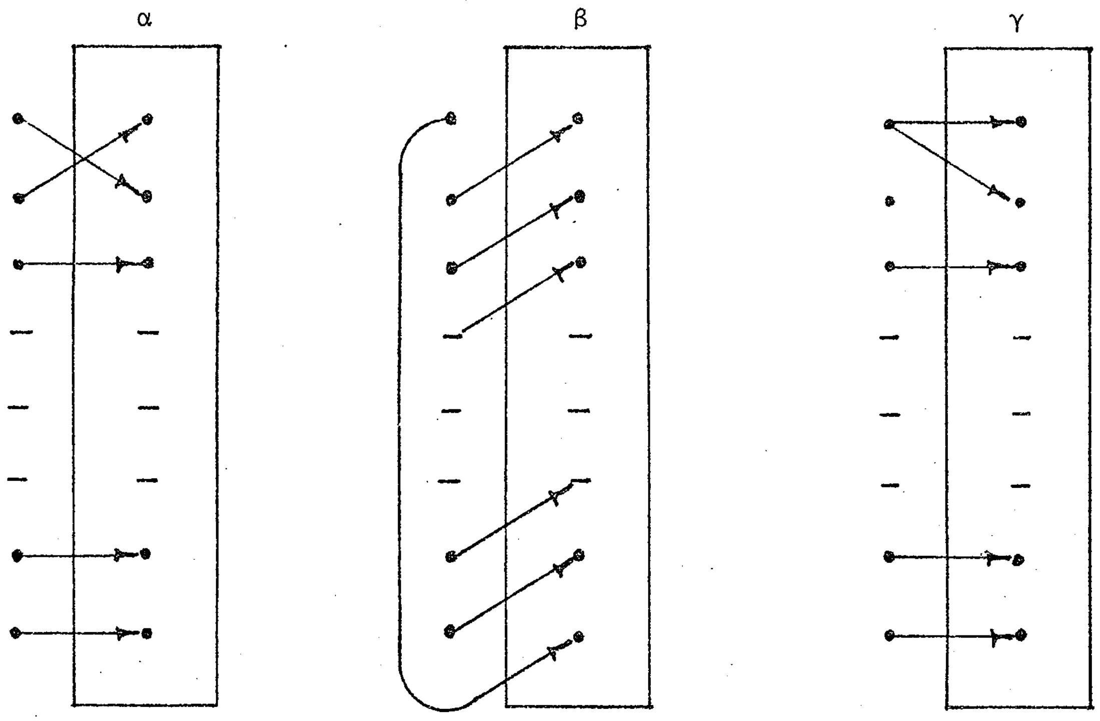


Figure 7.2

Haring proves that an operator realization for a map defines a generator realization for that same map. The generator sequence can be obtained by simply reversing the operator sequence and substituting a for α, b for β, and c for γ, in the reversed sequence.

Since operators are easier to deal with than generators, a generator realization is derived by first finding an operator realization, and then reversing the operator sequence with appropriate substitutions.

The main obstacle to using the set {a,b,c} to realize a map is that the upper bound on the length of the generator sequence increases rapidly as the number of states increases. This has an obvious detrimental effect on the cost and delay time of a realization.

To circumvent this problem, Menger [53] has proposed an enlarged set of generators. The number of generators, in this case, is not fixed, but increases as the number of states increases. This has the effect of allowing the length of generator realizations to grow linearly with any increases in the number of states. Since Menger's work is relevent to the new generator set we propose in Section 7.4, a detailed description is provided in the following section. (In addition, a minor correction to Menger [53] is given in Section 7.3.)

The new generator set proposed is similar in concept to that of

Menger and Haring; with the difference again being the size of the generator set. Increasing the generator set creates problems with obtaining a minimal realization of a map, in terms of the new generators. A heuristic algorithm for deriving realizations is presented.

As Menger's generator set is a subset of our proposed set, any realization using our generators should be no worse than a realization using Menger's generators. In fact for all the sample maps realized, realizations using the expanded generator set have been shorter than Menger's realizations.

One method of implementing the generators is by using cellular arrays, as demonstrated by Huang [35] and Krishnan and Smith [45]. Huang implements the three universal generators of Haring using identical cellular arrays. (Boundary conditions are used to set the function of the arrays.) A cellular realization of a generator sequence has the benefit of uniform connections between the array. As already noted, the length of a generator sequence grows rapidly as the number of states increases.

Krishhan and Smith attempt to correct the above problem by deriving cellular implementations for Menger's expanded generator set. However, the cellular implementation they use has certain disadvantages:

(1) the cells in the array are not identical;

(2) the inter-array connections are not uniform.

Problem (1) prevents rows of array from being joined together to form a larger row and thus prevents the array being used as a modular building block. Problem (2) prevents the fabrication of rows of arrays on a single chip. This is because the connections between arrays cannot be specified until after the generator sequence for a column has been determined. A cellular array which eliminates the above problems is presented in Section 7.5.

Some common notation, used throughout the following sections, is presented next.

A cycle is a closed loop of states. Cycles will usually be represented as a list of states enclosed in parenthesis.

For example, (3,7,5,2) where the state in position i-1 maps into the state in position i. That is, state 3 maps into state 7, 7 into 5, etc. The state in the last position wraps around and maps into the state in the first position. Clearly, a permutation is a collection of disjoint cycles.

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

P

The permutation P can be represented by the cycles

P = (1,6), (2,3,7,5), (4), (8).

Often unit cycles are deleted when describing a permutation. e.g., P = (1,6), (2,3,7,5). The permutation column P can also be represented as P = <6,3,7,4,2,1,5,8>, which contains the information, state 1 maps into state 6, 2 into 3, 3 into 7, and so forth.

The identity permutation, E, is the permutation consisting of only unit cycles.

E = (1),(2),...,(n) or E = <1,2,...,n>.

A new permutation Q obtained from permutation P by interchanging states r and s is denoted by, Q = P•(r,s).

For example, let P = <5,2,6,3,4,1>.

Q = P•(5,3)

= <3,2,6,5,4,1>.

7.2 Menger's Algorithm

Generators and Operators

Menger [53] uses a generator set consisting of 2(n-1) generators, where n is the number of states in a machine. The generators are illustrated in Figure 7.3.

	b_2	b_3	....	b_n	c_2	c_3	....	c_n
1	2	3		n	1	1		1
2	1	2		2	1	2		2
3	3	1		3	3	1		3
.	.	.		.	.	.		.
i	i	i		i	i	i		i
.	.	.		.	.	.		.
.	.	.		.	.	.		.
n	n	n		1	n	n		1

Menger's Generator Set

Figure 7.3

A <u>map realization</u> of map M is any sequence $d_1 \ldots d_m$ of generators, $d_i \in \{b_2, \ldots, b_n, c_2, \ldots, c_n\}$ such that $d_1 \ldots d_m = M$; that is

$$d_1 \ldots d_m(s) = d_m(\ldots . d_1(s)) = M(s) \text{ for each state } s = 1, \ldots ., n.$$

Menger defines a set of operators which correspond to the generators. Rather than exchanging prescribed states like the generators, the operators exchange states in prescribed positions. The operators, which are easier to manipulate than generators, are illustrated in Fig. 7.4.

	β_2	β_3	....	β_n	γ_2	γ_3	....	γ_n
s_1	s_2	s_3		s_n	s_1	s_1		s_1
s_2	s_1	s_2		s_2	s_1	s_2		s_2
s_3	s_3	s_1		s_3	s_1	s_1		s_3
.	.	.		.	.	.		.
s_i	s_i	s_i		s_i	s_i	s_i		s_i
.	.	.		.	.	.		.
.	.	.		.	.	.		.
s_n	s_n	s_n		s_1	s_n	s_n		s_1

Operators (s_i represents the state in position i)

Figure 7.4

An operator realization of map M is any sequence $\delta_1 \dots \delta_m$ of operators, $\delta_i \in \{\beta_2, \dots, \beta_n, \gamma_2, \dots, \gamma_n\}$, which operating upon the identity map E yields M. That is,

$$\delta_1 \dots \delta_m(E) = \delta_m(\dots \delta_1(E)) = M .$$

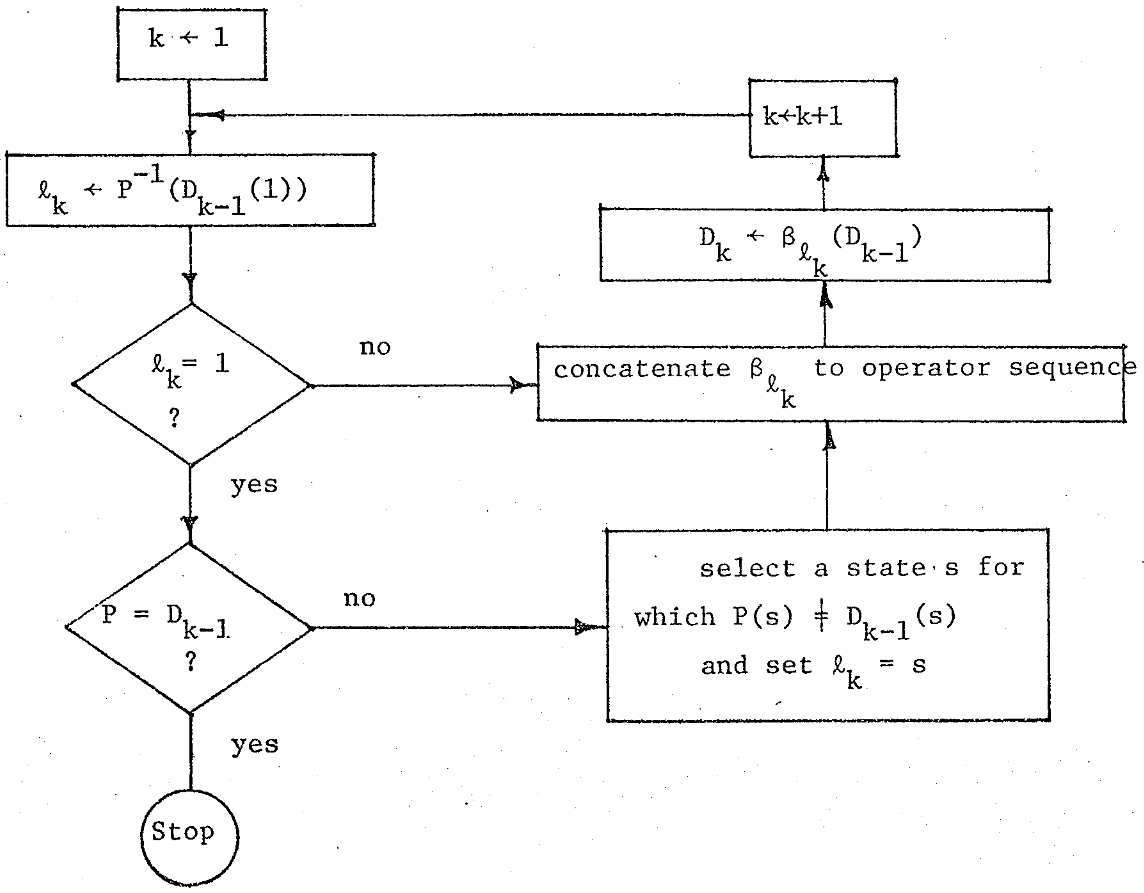
Operators and generator realizations are related by Lemma 7.1. For the operator and generator sequences $\delta_1 \dots \delta_m$ and $d_m \dots d_1$, δ_i is the image of d_i under the "natural" correspondence $\beta_k \leftrightarrow b_k$, $\gamma_k \leftrightarrow c_k$, for each $k=2, \dots, n$.

Lemma 7.1 (Menger [53]). If $\delta_1 \dots \delta_m(E) = M$, then $d_m \dots d_1 = M$, and vice versa.

Realization of Permutation Maps

Before considering non-permutation maps, Menger first develops an algorithm for obtaining an operator realization of a permutation map P. The algorithm developed is easy to apply and guarantees the minimal operator realization of P.

Notation: Let $D_0 = E$ and D_k be the n-tuple resulting from the kth iteration of the algorithm


$k \leftarrow 1$
$\ell_k \leftarrow P^{-1}(D_{k-1}(1))$
$\ell_k = 1$?
no
yes
$P = D_{k-1}$?
no
yes
Stop
select a state s for which $P(s) \neq D_{k-1}(s)$ and set $\ell_k = s$
concatenate β_{ℓ_k} to operator sequence
$D_k \leftarrow \beta_{\ell_k}(D_{k-1})$
$k \leftarrow k+1$

Example 7.1 An operator sequence for $P = <2,1,4,3>$ is obtained using Menger's algorithm

$$D_0 = <1,2,3,4>$$

$$k=1 \qquad \ell_1 = P^{-1}(D_0(1)) = 2$$

Since $\ell_1 \neq 1$, β_2 is added to the operator sequence

$D_1 = \beta_2(D_0) = <2,1,3,4>$ operator sequence: β_2

$k = 2$

$\ell_2 = P^{-1}(D_1(1)) = 1$ and $D_1 \neq P$

Since $D_1(3) = 3 \neq 4 = P(s)$, set $s=3$ and $\ell_2=3$

$D_2 = \beta_3(D_1) = <3,1,2,4>$ operator sequence: $\beta_2\beta_3$

$k = 3$

$\ell_3 = P^{-1}(D_2(1)) = 4$

$D_3 = \beta_4(D_2) = <4,1,2,3>$ operator sequence: $\beta_2\beta_3\beta_4$

$k = 4$

$\ell_4 = P^{-1}(D_3(1)) = 3$

$D_4 = \beta_3(D_3) = <2,1,4,3>$ operator sequence: $\beta_2\beta_3\beta_4\beta_3$

$k = 5$

$\ell_5 = P^{-1}(D_4(1)) = 1$

Since $D_4 = P$, the algorithm stops and a minimal operator realization of P is $\beta_2\beta_3\beta_4\beta_3$.

Notation: A permutation P with ℓ cycles will have its cycle structure denoted by $w_{j_1} w_{j_2} \ldots . w_{j_\ell}$; w_{j_1} is the cycle containing state 1 and j_i denotes the number of states in cycle w_{j_i} .

The term $\$(P)$ will be used to refer to the number of operators required to realize permutation P.

Theorem 7.1 (Theorem 3.2 [53]) For all n and any n-state permutation P, the algorithm just described yields a realization for which $\$(P) = (j_1-1) + \sum_{*}(j_i+1)$, where the * indicates that the summation is taken over just those j_i where $2 \leq i \leq \ell$ and $j_i \geq 2$.

Realization of Non-Permutation Maps

For a given map M and any state s, let $M^{-1}(s)$ denote the set of states t_i (possibly empty) satisfying $M(t_i)=s$. The term $\phi(M)$ denotes the set of states s_i for which $M^{-1}(s_i)$ is the null set.

Definition 7.1.i [53] A permutation P is compatible with map M if and only if for each state $s \notin \phi(M)$, $P^{-1}(s) \in M^{-1}(s)$ holds.

Definition 7.1.ii [53] A permutation P is compatible with map M if and only if $\{s | M(P^{-1}(s)) \neq s \} = \phi(M)$.

Let $\sigma = \delta_1 \ldots \delta_m$ be an arbitrary sequence of operators in $\{\beta_2, \ldots, \beta_n, \gamma_2, \ldots, \gamma_n\}$, with the elements in σ not necessarily distinct. Denote the length of σ by $\$(\sigma) = m$.

The number of operators in σ belonging to $\{\gamma_2, \ldots, \gamma_n\}$ is denoted by $\$\gamma(\sigma)$; let π_σ denote the sequence of β operators (possibly empty) which remain after all γ operators in σ have been deleted.

The following lemmas and theorem provide a cost function for a minimal realization of map M.

Lemma 7.2 (Lemma 3.1 [53]) If sequence σ is a realization of map M and $\$\gamma(\sigma) = |\phi(M)|$, then π_σ is the realization of a permutation that is compatible with map M.

Lemma 7.3 (Lemma 3.2 [53]) For every map M there exists at least one minimum realization σ for which $\$\gamma(\sigma) = |\phi(M)|$.

Definition 7.2 [53] The state set A(M,P) is the set containing those states s, $s \neq 1$, for which $|M^{-1}(s)| \geq 2$ and $P(s) = s$.

Lemma 7.4 (Lemma 3.4 [53]) Given map M, for every permutation P that is compatible with map M, one can construct a realization σ of M satisfying both $\pi_\sigma(E) = P$ and $\$(\sigma) = |\phi(M)| + \$(P) + 2\cdot|A(M,P)|$.

<u>Theorem 7.2</u> (Theorem 3.3 [53] For any map M there exists at least one permutation P that is compatible with map M, for which the sequence constructed as in the proof of Lemma 7.4 is a minimum realization for M of length $\$(\sigma) = |\phi(M)| + \$(P) + 2\cdot|A(M,P)|$.

<u>Example 7.2</u> For map M = <4,2,2,4> permutation P = <4,2,3,1> is compatible with map M.

$$\begin{aligned} \phi(M) &= \{1,3\} \\ A(M,P) &= \{2\} \\ \$(P) &= (j_1-1) + \textstyle\sum_*(j_i+1) \\ &= (2-1) + 0 \\ &= 1 \end{aligned}$$

Thus, a minimum realization for M would require

$$\begin{aligned} \$(\sigma) &= \$(P) + |\phi(M)| + 2\cdot|A(M,P)| \\ &= 1 + 2 + 2\cdot 1 \\ &= 5 \text{ operators} \end{aligned}$$

	β_4	γ_4	β_2	γ_3	β_2
1	4	4	2	2	4
2	2	2	4	4	2
3	3	3	3	2	2
4	1	4	4	4	4

$\sigma = \beta_4,\gamma_4,\beta_2,\gamma_3,\beta_2$ is a 5 operator realization of M using the permutation P; $\pi_\sigma = \beta_4,\beta_2,\beta_2$ is a realization of P (Lemma 7.2).

As a result of Theorem 7.2 it is possible to obtain a minimum realization of a map M by evaluating $\$(P) + 2\cdot|A(M,P)|$ for every permutation P compatible with map M, and choosing the permutation which minimizes this function. However, because of the great many permutations compatible with map M, for any map M, this is not a practical approach.

To overcome this problem, Menger defines a new pseudo cost function \$'(M,P), based on the redefined cost function \$'(P) and the new set A'(M,P).

$$\$'(P) = \sum_{i=1}^{\ell}(j_i+1), \quad \text{where} \quad j_i \geq 2.$$

$$A'(M,P) = \{s \mid P(s) = s \text{ and } |M^{-1}(s)| \geq 2\} .$$

Let $\$'(M,P) = |\phi(M)| + \$'(P) + 2\cdot|A'(M,P)|$.

Theorem 7.3 (Theorem 3.4 [53] For a given map M and any permutation P, compatible with map M, $\$'(M,P) - \$(\sigma) = 0$ or 2.

Thus, obtaining a permutation P compatible with map M, which minimizes \$'(M,P) will lead to a realization of M, at most two operators costlier than the true minimum cost realization of M.

Definition 7.3 [53] A permutation Q compatible with map M, which minimizes \$'(M,P) is said to be minimally compatible with map M.

The following algebraic rules are used by Menger to enable \$'(M,P) to be minimized.

AR1 If P is compatible with map M, then $P\cdot(u,v)$ is also compatible with map M for any two states $u,v \in \phi(M)$.

AR2 If P is compatible with map M, then $P\cdot(u,p)$ is also compatible with map M for any state $u \in \phi(M)$, where $p = M(P^{-1}(u))$.

For a permutation P, a cycle in P is special if and only if it contains two or more states, at least one of which is in $\phi(M)$.

The following necessary conditions must be satisfied by any permutation P that is minimally compatible with map M. (These conditions are proved using the algebraic rules, AR1 and AR2.)

NC1 P can have at most one special cycle.

NC2 If $u \in \phi(M)$ and $P(u) \neq u$, then

(a) $P(u) \notin \phi(M)$

(b) $P(u) \neq M(p^{-1}(u))$

(c) $M(P^{-1}(u))$ appears in the special cycle.

The conditions NC1 and NC2 are used, in turn to prove lemmas which determine a permutation P, such that P is minimally compatible with map M. The general form of each lemma is to state specific conditions, which must be met, for a state p to be assigned to a position in P. Rather than state the lemmas explicitly, they are presented implicitly when Menger's algorithm for non-permutation map realizations is given. A brief description of the use of the lemmas in Phase I of the algorithm follows. (Readers interested in the statement and proof of the lemmas are referred to Lemmas 3.5-3.9 of Menger [53].)

For $\$'(M,P) = |\phi(M)| + \$'(P)+2\cdot|A'(M,P)|$ it can be seen that only the second and third terms in the function are dependent on the permutation P. Thus, in order to minimize $\$'(M,P)$, $\$'(P)$ and $|A'(M,P)|$ must be minimized. One of the lemmas permits the set A'(M,P) to be determined directly from the map M. Now only the term $\$'(P)$ remains undetermined in $\$'(M,P)$. The remaining lemmas indicate how the states, $s \notin A'(M,P)$, can be assigned to positions in P (consistent with compatibility) so as to minimize $\$'(P)$.

Menger's Algorithm

Phase I A. Identify the states which do not appear as images under M; i.e. $\phi(M)$.

B. Each remaining state p appears one or more times as an image under M.

(i) If $M^{-1}(p)$ contains one or more states that are neither in $\phi(M)$ nor p itself, take $P^{-1}(p)$ to be one of these states.

(ii) If $M^{-1}(p)$ contains states p and possibly other states which are exclusively in $\phi(M)$, take $P^{-1}(p) = p$. (Note that this step determines A'(M,P).)

(iii) If $M^{-1}(p)$ contains only states that are in $\phi(M)$, and this must apply to all image states that did not qualify for (i) and (ii), then $P^{-1}(p)$ can be taken to be any state in $M^{-1}(p)$.

C. For each state $s \in \phi(M)$ not assigned an image in B (iii), take $P^{-1}(s) = s$.

D. Any states which remain unassigned as images at this point must be members of $\phi(M)$. These may be assigned to the remaining vacancies in an arbitrary manner.

PHASE II A. Denote the current permutation by R. If R has two distinct non-unit cycles each of which contains at least one state in $\phi(M)$, then a new permutation R' is formed by exchanging these two images (AR1). Repeat Step A as long as it applies.

B. If the current permutation, R, has a non-unit cycle which contains a state $s \in \phi(M)$, such that $M(R^{-1}(s))$ is not also in this cycle, then these two images must be exchanged to obtain a new permutation. Step B is repeated as long as applicable.

<u>PHASE III</u> An operator realization for the permutation Q, resulting from Phase II, is derived.

<u>PHASE IV</u> For each state t, where $|M^{-1}(t)| \geq 2$, there either is or is not an index i such that $t = D_i(1)$.

A. If such an i exists, then certain γ-operators must be inserted in the operator realization of Q at this point. Specifically, for each state $s \neq t$ satisfying $M(Q^{-1}(s)) = t$, an operator γ_r must be inserted after the i^{th} β operator, where $r = D_i^{-1}(s)$.

B. For each state t where there is no i such that $D_i(1) = t$, the operator sequence obtained in A is augmented with β_t. Immediately following β_t further γ operators are attached. Specifically for each state $s \neq t$, such that $M(Q^{-1}(s)) = t$, attach the operator γ_r, where $r = Q^{-1}(s)$. Finally a second β_t is attached.

C. Reverse the operator realization and replace each β_k by b_k and each γ_k by c_k, $k=2,\ldots,n$, to obtain the desired minimum map-generator realization of M.

The following example demonstrates the application of Menger's algorithm to a map M.

Example 7.3 $M = \langle 1,1,2,2,5,5,8,7,7,9 \rangle$

PHASE I A. $\phi(M) = \{3,4,6,10\}$

B. (i) $M^{-1}(1) = \{1,2\}$, $M^{-1}(8) = \{7\}$, $M^{-1}(9) = \{10\}$, and $M^{-1}(7) = \{8,9\}$

$P^{-1}(1) = 2$, $P^{-1}(8) = 7$, and $P^{-1}(9) = 10$ are forced; choose $P^{-1}(7) = 9$.

(ii) $M^{-1}(5) = \{5,6\}$ and $6 \in \phi(M)$. Therefore, $P^{-1}(5) = 5$ and $A'(M,P) = \{5\}$.

(iii) $M^{-1}(2) = \{3,4\} \subseteq \phi(M)$; choose $P^{-1}(2) = 3$.

C. States $4,6 \in \phi(M)$ have not been assigned an image; set $P^{-1}(4) = 4$ and $P^{-1}(6) = 6$.

D. The remaining unassigned states are 1 and 8. Assigning the remaining images, 3 and 10, arbitrarily, set $P^{-1}(3) = 1$ and $P^{-1}(10) = 8$.

The resulting permutation R is

$$R = \langle 3,1,2,4,5,6,8,10,7,9 \rangle$$

PHASE II The cyclic representation of R is

$$R = (1,3,2)(4)(5)(6)(7,8,10,9)$$

A. Since 3 is in cycle (1,3,2) and 10 is in cycle (7,8,10,9), apply (3,10)

$$R' = R\cdot(3,10)$$
$$= (1,10,9,7,8,3,2)(4)(5)(6)$$

Step A no longer applies.

B. Step B does not apply.

PHASE III $Q = \langle 10,1,2,4,5,6,8,3,7,9 \rangle$

A minimal operator realization of Q is

$$\beta_2, \beta_3, \beta_8, \beta_7, \beta_9, \beta_{10}$$

PHASE IV An operator realization of M is presented, from which the application of Steps A and B can be deduced.

	γ_{10}	β_2	γ_4	β_3	β_8	β_7	γ_8	β_9	β_{10}	β_5	γ_6	β_5
1	1	2	2	3	8	7	7	9	1	5	5	1
2	2	1	1	1	1	1	1	1	1	1	1	1
3	3	3	3	2	2	2	2	2	2	2	2	2
4	4	4	2	2	2	2	2	2	2	2	2	2
5	5	5	5	5	5	5	5	5	5	1	1	5
6	6	6	6	6	6	6	6	6	6	6	5	5
7	7	7	7	7	7	8	8	8	8	8	8	8
8	8	8	8	8	3	3	7	7	7	7	7	7
9	9	9	9	9	9	9	9	7	7	7	7	7
10	1	1	1	1	1	1	1	1	9	9	9	9

$$\begin{aligned} \$'(M,P) &= |\phi(M)| + \$'(P) + 2\cdot|A'(M,P)| \\ &= 4 + 6 + 2\cdot 1 \\ &= 12 \end{aligned}$$

In proving a theorem, essential to his algorithm, Menger assumes an irredundancy condition to hold. However, this irredundancy condition is inadequate. In the following section we show that an extra irredundancy condition is necessary to prove the theorem.

7.3 Irredundancy Conditions

To establish the pseudo-cost function, $\$'(M,P) = |\phi(M)| + \$'(P) + 2\cdot|A'(M,P)|$, for a nonpermutation map M, Menger utilizes Lemma 7.3. The lemma is a consequence of the much stronger theorem.

Theorem 7.4 ([53] p.3-35) If sequence $\sigma = \delta_1 \cdots \delta_m$ is a realization of map M for which $\$\gamma(\sigma) > |\phi(M)|$, one can construct a second realization σ' of M satisfying both $\$\gamma(\sigma') < \$\gamma(\sigma)$ and $\$(\sigma') \le \$(\sigma) = m$.

In order to prove the theorem, Menger assumes a redundancy condition which we state formally below.

Definition 7.4 [53] If $\sigma = \delta_1 \cdots \delta_m$ is a realization of map M and there exists an index k and state t such that $M_k(1) = M_k(t)$ and $\delta_{k+1} = \gamma_t$, then realization σ is redundant.

Obviously, if a realization σ is redundant, then the operator δ_{k+1} which causes the redundancy condition can be deleted to give $\sigma' = \delta_1 \ldots, \delta_k, \delta_{k+2}, \ldots, \delta_m$, which is still a realization of M.

The theorem is proved assuming that the operator sequence is irredundant in the sense of Definition 7.4. That is, if an operator sequence is not redundant and $\$\gamma(\sigma) > |\phi(M)|$, then "...there must exist an index k and state p such that both $M_k(p) \notin \phi(M)$ and $\delta_{k+1} = \gamma_p$, for otherwise it would follow with the irredundancy of σ that $\$\gamma(\sigma) = |\phi(M)|$." (p.3-35 [53])

This is not strictly true and a counter example is given below. Later, we state another redundancy condition which makes Menger's statement applicable.

<u>Example 7.4</u> $M = <1,2,2,1>$ $\phi(M) = \{3,4\}$

$\sigma = \delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_6, \delta_7 = \beta_4, \gamma_3, \beta_2, \gamma_2, \beta_3, \beta_4, \gamma_4$

	β_4	γ_3	β_2	γ_2	β_3	β_4	γ_4
1	4	4	2	2	4	1	1
2	2	2	4	2	2	2	2
3	3	4	4	4	2	2	2
4	1	1	1	1	1	4	1
	M_1	M_2	M_3	M_4	M_5	M_6	M_7

σ is irredundant since $\delta_{i+1} = \gamma_s$ implies $M_i(1) \neq M_i(s)$ for all indices $1 \le i \le m-1$. Examining σ for cases where $\delta_{k+1} = \gamma_p$ and $M_k(p) \notin \phi(M)$:

<u>k=6 and p=4</u>: $\delta_{k+1} = \delta_7 = \gamma_4$ and $M_6(4) = 4 \in \phi(M)$
<u>k=3 and p=2</u>: $\delta_{k+1} = \delta_4 = \gamma_2$ and $M_3(2) = 4 \in \phi(M)$
<u>k=1 and p=3</u>: $\delta_{k+1} = \delta_2 = \gamma_3$ and $M_1(3) = 3 \in \phi(M)$.

Thus, there does not exist an index k and a state p such that both $M_k(p) \notin \phi(M)$ and $\delta_{k+1} = \gamma_p$. But $\$\gamma(\sigma) = 3 > |\phi(M)| = 2$. This contradicts the assumption made by Menger in proving the theorem. Consequently the sequence σ is irredundant and cannot be reduced using Menger's theorem.

However, $\delta_2 = \gamma_3$ can be removed from σ to give σ' such that $\sigma'(E) = M$.

	β_4	β_2	γ_2	β_3	β_4	γ_4
1	4	2	2	3	1	1
2	2	4	2	2	2	2
3	3	3	3	2	2	2
4	1	1	1	1	3	1

The following definition states another essential redundancy condition for operator realizations.

Definition 7.5 If $\sigma = \delta_1 \ldots \delta_m$ is a realization of map M and there exists an index k such that $M_k(1) \in \phi(M)$, $M_k(t) \in \phi(M)$, and $\delta_{k+1} = \gamma_t$, then the realization σ is redundant.

If a realization σ is redundant by Definition 7.5, then δ_{k+1} can be deleted from σ to give σ', which is still a realization of M.

Definition 7.6 The operator realization $\sigma = \delta_1 \ldots \delta_m$ is irredundant if

(i) $\delta_{i+1} = \gamma_s$, implies $M_i(1) \neq M_i(s)$; and

(ii) $\delta_{i+1} = \gamma_s$, implies $M_i(1) \notin \phi(M)$ and $M_i(s) \notin \phi(M)$.

7.4 A New Generator Set

In this section we propose a new set of generators for realizing maps. The generators are:

$$G = \{b_{1,2}, \ldots, b_{1,n}, b_{2,3}, \ldots, b_{2,n}, \ldots, b_{n-1,n}, \\ c_{1,2}, \ldots, c_{1,n}, c_{2,3}, \ldots, c_{2,n}, \ldots, c_{n-1,n}\}.$$

Generator $b_{i,j}$ substitutes state i for every occurrence of state j, and state j for every occurrence of state i.

$$b_{i,j} = \langle 1, \ldots, i-1, j, i+1, \ldots, j-1, i, j+1, \ldots, n \rangle.$$

Generator $c_{i,j}$ substitutes state i for every occurrence of state j.

$$c_{i,j} = \langle 1, \ldots, i-1, i, i+1, \ldots, j-1, i, j+1, \ldots, n \rangle.$$

An algorithm for deriving realizations for permutation maps using the $b_{i,j}$ generators is developed. The algorithm is straightforward and obtains the minimal generator realization.

The derivation of a minimal realization for a non-permutation map is not as direct. At present we can only establish a maximum bound on the generator length for a non-permutation realization. The reason for this is that there are many possible ways to derive a non-permutation realization. An exhaustive search of all possible realizations, in most cases, is not feasible.

A heuristic algorithm, which limits the number of realizations examined is presented. The emphasis has been placed on developing a simple algorithm which obtains a near minimal realization quickly.

The theoretical basis for the algorithm presented can, in some cases, be obtained by simply extending Menger's definitions and theorems to apply to the new generator set. For these instances the definition and theorems are restated with acknowledgements to Menger.

Definition 7.7 (Menger [53]) A map realization of a map M is any sequence $d_1 \ldots . d_m$ of generators $d_i \in G$, such that $d_1 \ldots d_m = M$. That is,

$$d_1 \ldots . d_m(s) = d_m(\ldots . d_1)(s)) = M(s), \quad \text{for}$$

each state $s = 1, \ldots ., n$.

Like Menger, we develop an algorithm which uses operators to derive the generator sequence. The n-state operators are

$$G' = \{\beta_{1,2}, \ldots, \beta_{1,n}, \beta_{2,3}, \ldots, \beta_{2,n}, \ldots, \beta_{n-1,n},$$
$$\gamma_{1,2}, \ldots, \gamma_{1,n}, \gamma_{2,3}, \ldots, \gamma_{2,n}, \ldots, \gamma_{n-1,n}\}$$

Operator $\beta_{i,j}$ interchanges the states in position i and position j, while $\gamma_{i,j}$ replaces the state in position j with the state in position i, as illustrated in Figure 7.5.

	$\beta_{1,2}$	$\cdots$	$\beta_{1,n}$	$\beta_{2,3}$	$\cdots$	$\beta_{2,n}$	$\cdots$	$\beta_{n-1,n}$	$\gamma_{1,2}$	$\cdots$	$\gamma_{1,n}$	$\gamma_{2,3}$	$\cdots$	$\gamma_{2,n}$	$\cdots$	$\gamma_{n-1,n}$
s_1	s_2		s_n	s_1		s_1		s_1	s_1		s_1	s_1		s_1		s_1
s_2	s_1		s_2	s_3		s_n		s_2	s_1		s_2	s_2		s_2		s_2
s_3	s_3		s_3	s_2		s_3		s_3	s_2		s_3	s_2		s_3		s_3
.	.		.	.		.		.	.		.	.		.		.
.	.		.	.		.		.	.		.	.		.		.
.	.		.	.		.		.	.		.	.		.		.
s_{n-1}	s_{n-1}		s_{n-1}	s_{n-1}		s_{n-1}		s_n	s_{n-1}		s_{n-1}	s_{n-1}		s_{n-1}		s_{n-1}
s_n	s_n		s_1	s_n		s_1		s_{n-1}	s_n		s_1	s_n		s_2		s_{n-1}

new operator set G'

Figure 7.5

Definition 7.8 (Menger [53]) An operator realization of map M is any sequence $\delta_1 \ldots . \delta_m$ of operators $\delta_i \in G'$, which operating upon the identity map E yields map M. That is,

$$\delta_1 \ldots . \delta_m(E) = \delta_m(\ldots . \delta_1(E)) = M.$$

The basic result which allows a generator realization to be obtained from an operator realization is now stated.

Lemma 7.5 (Menger [53]) If $\delta_1 \ldots . \delta_m(E) = M$, then $d_m \ldots . d_1 = M$ and vice versa. For each $\delta_i = \beta_{x,y}$, substitute $d_i = b_{x,y}$ and for each $\delta_i = \gamma_{x,y}$, substitute $d_i = c_{x,y}$.

Example 7.5 For map M = <8,8,2,6,2,4,4,4> we have the operator sequence $\delta_1 \ldots . \delta_7 = \beta_{1,8}, \gamma_{2,3}, \beta_{2,5}, \gamma_{1,2}, \beta_{4,6}, \gamma_{6,7}, \gamma_{6,8}$.

	$\beta_{1,8}$	$\gamma_{2,3}$	$\beta_{2,5}$	$\gamma_{1,2}$	$\beta_{4,6}$	$\gamma_{6,7}$	$\gamma_{6,8}$
1	8	8	8	8	8	8	8
2	2	2	5	8	8	8	8
3	3	2	2	2	2	2	2
4	4	4	4	4	6	6	6
5	5	5	2	2	2	2	2
6	6	6	6	6	4	4	4
7	7	7	7	7	7	4	4
8	1	1	1	1	1	1	4

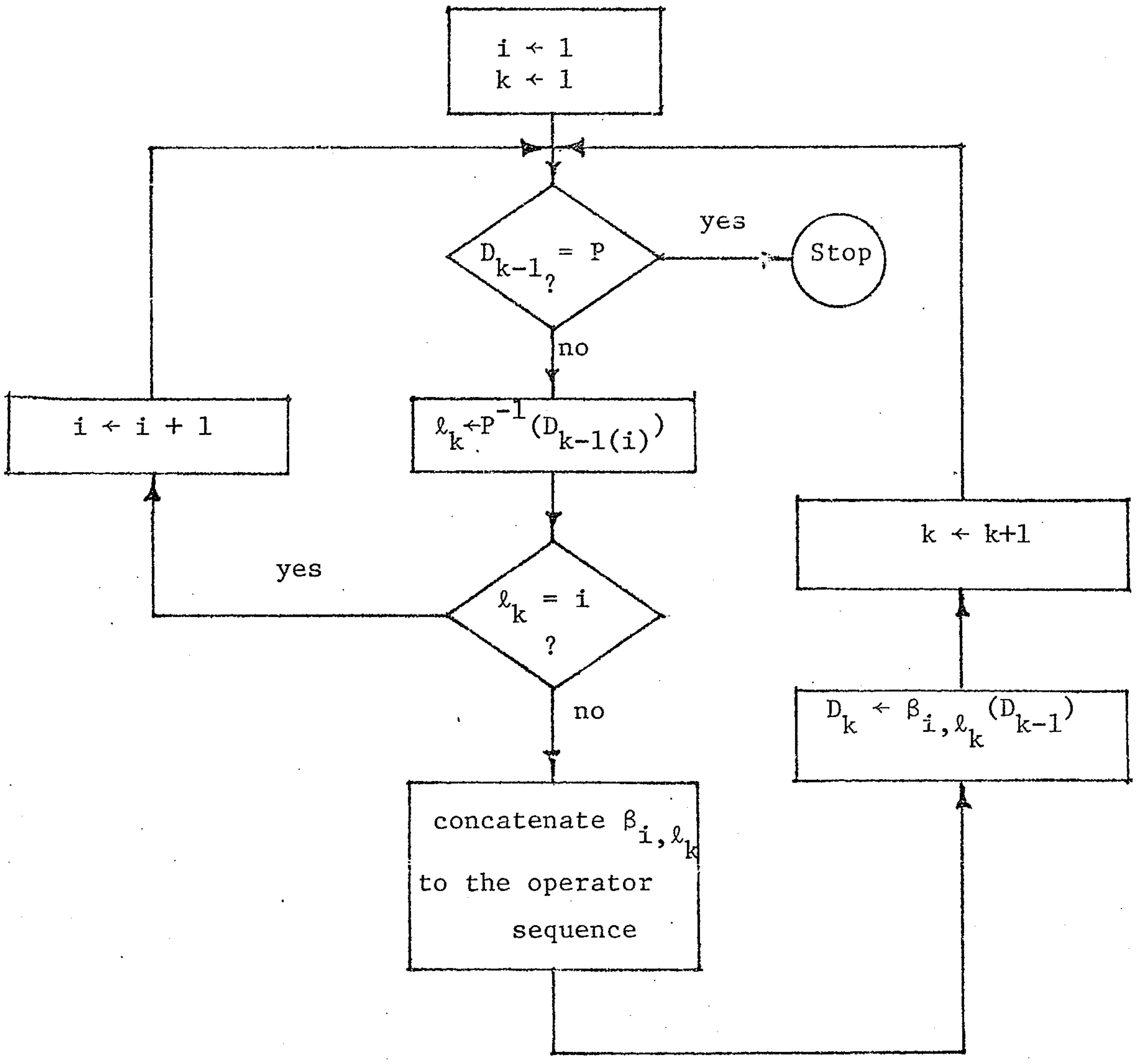
The corresponding generator sequence is

$$d_1 \ldots d_7 = c_{6,8}, c_{6,7}, b_{4,6}, c_{1,2}, b_{2,5}, c_{2,3}, b_{1,8}$$

	$c_{6,8}$	$c_{6,7}$	$b_{4,6}$	$c_{1,2}$	$b_{2,5}$	$c_{2,3}$	$b_{1,8}$
1	1	1	1	1	1	1	8
2	2	2	2	1	1	1	8
3	3	3	3	3	3	2	2
4	4	4	6	6	6	6	6
5	5	5	5	5	2	2	2
6	6	6	4	4	4	4	4
7	7	6	4	4	4	4	4
8	6	6	4	4	4	4	4

Parallel to Menger, we develop an algorithm which gives an operator realization for any permutation P. Basically, a permutation is a collection of cycles. For a cycle c, if i is the smallest state in the cycle, the cycle c can be derived from the identity vector E using operators of the form $\beta_{i,j}$, where state j is in cycle c.

The algorithm follows in the form of a flowchart.



$i \leftarrow 1$
$k \leftarrow 1$
$D_{k-1} = P$?
yes
Stop
no
$i \leftarrow i + 1$
$\ell_k \leftarrow P^{-1}(D_{k-1}(i))$
$k \leftarrow k+1$
yes
$\ell_k = i$?
no
$D_k \leftarrow \beta_{i,\ell_k}(D_{k-1})$
concatenate β_{i,ℓ_k} to the operator sequence

<u>Example 7.6</u> $P = <2,1,4,3>$

Set $D_0 = E = <1,2,3,4>$ $i=1$ and $k=1$

$\ell_k = P^{-1}(D_{k-1}(i))$

$\ell_1 = P^{-1}(D_0(1))$

$= 2$ and $\ell_k \neq i$

operator sequence: $\beta_{i,\ell_k} = \beta_{1,2}$

$D_1 = \beta_{1,2}(<1,2,3,4>)$

$D_1 = <2,1,3,4>$

$k = k + 1 = 2$

$\ell_2 = P^{-1}(D_1(1))$

$= 1$ and $\ell_k = i$

$i = i + 1 = 2$

$\ell_2 = P^{-1}(D_1(2))$

$= 2$ and $\ell_k = i$

$i = i + 1 = 3$

$\ell_2 = P^{-1}(D_1(3))$

$= 4$ and $\ell_k \neq i$

operator sequence: $\beta_{1,2}$, $\beta_{3,4}$

$D_2 = \beta_{3,4}(<2,1,3,4>)$

$= <2,1,4,3>$.

Since $D_2 = P$, the algorithm stops and $\beta_{1,2}$, $\beta_{3,4}$ is an operator realization of P.

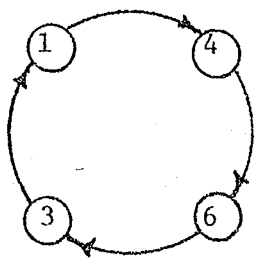
	$\beta_{1,2}$	$\beta_{3,4}$
1	2	2
2	1	1
3	3	4
4	4	3

In addition to using the notation (a,b,...,x,y) to indicate cycles, cycle diagrams are also used.

Example 7.7

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

The cycle (1,4,6,3) of permutation P can be represented by the diagram



The cycle diagram also provides a method of determining an operator realization. Starting with the smallest state in the cycle, trace backwards through the cycle to find a realization for the cycle. That is, for the above cycle an operator realization is $\beta_{1,3}, \beta_{1,6}, \beta_{1,4}$.

An important point to note is that, as long as the operators for a cycle are applied in the order that they were determined, they can be intermixed with operators of another cycle.

The operator sequence to realize P found using the algorithm is $\beta_{1,3}, \beta_{1,6}, \beta_{1,4}, \beta_{2,5}, \beta_{2,7}$. However, the sequence $\beta_{1,3}, \beta_{2,5}, \beta_{1,6}, \beta_{2,7}, \beta_{1,4}$ is still a realization of P since the ordering within the cycles has not changed.

The number of $\beta_{i,j}$ operators required to realize a permutation map can be determined using Theorem 7.5.

As the proofs for Lemma 7.6 and Theorem 7.5 are obvious, they will merely be stated.

<u>Lemma 7.6</u> For a cycle c, where $|c| = n$ is the number of states in c, the length of the operator sequence to realize c is $n-1$.

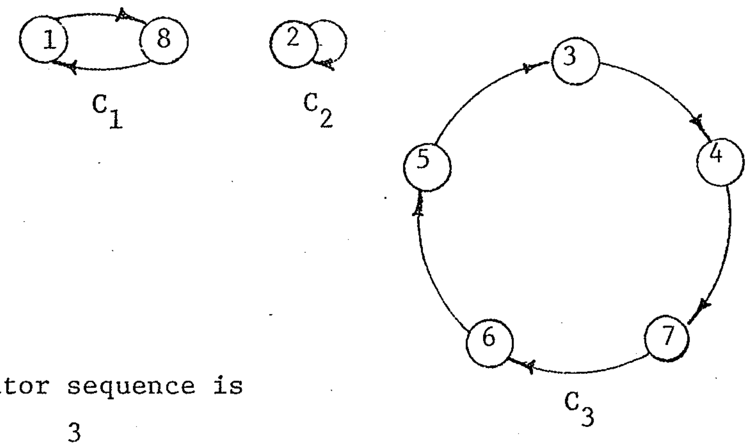
<u>Theorem 7.5</u> For a permutation state column P, with m cycles, $c_i, i=1,\ldots,m$, the length of a minimal operator realization of P is

$$L = \sum_{i=1}^{m}(|c_i|-1) .$$

Consequently, it is possible to find the number of operators required to realize a permutation P by examining the cycles of P.

Example 7.8

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



The length of the operator sequence is

$$L = \sum_{i=1}^{3} (|c_i| - 1)$$

$$= (2-1) + (1-1) + (5-1)$$

$$= 5$$

An operator realization for P of length 5 follows

	$\beta_{1,8}$	$\beta_{3,5}$	$\beta_{3,6}$	$\beta_{3,7}$	$\beta_{3,4}$
1	8	8	8	8	8
2	2	2	2	2	2
3	3	5	6	7	4
4	4	4	4	4	7
5	5	3	3	3	3
6	6	6	5	5	5
7	7	7	7	6	6
8	1	1	1	1	1

The worst-case conditions for a permutation realization can also be determined from Theorem 7.5. Simplifying the cost function gives

$$L = \sum_{i=1}^{m} (|c_i|-1)$$

$$= \sum_{i=1}^{m} |c_i| - \sum_{i=1}^{m} 1$$

$\therefore$ L = n − m, where n is the number of states in the permutation and

m is the number of cycles.
(Note that cycles of length 1 are included in the calculation.)

Obviously, the worst-case condition occurs when there is only one cycle involving all the states. When this occurs the length of the operator realization is n-1. (The worst-case for permutations realized with Menger's operators required $\frac{3n}{2}$ operators. Thus, the worst-case conditions for both operator sets are a linear function of the number of states. For our operator set there should be a distinct saving in operators required when dealing with large state sets.)

As the number of cycles in a permutation increases, the number of operators in the realization decreases. This fact is utilized when determining operator realizations for non-permutation maps.

Non-Permutation Realizations

To develop the algorithm for non-permutation realizations, it must first be established that the number of $\gamma_{i,j}$ operators in a minimal realization is given by $|\phi(M)|$.

Definition 7.9 (Menger [53]) A permutation P is compatible with map M if and only if $\{s | M(P^{-1}(s) \neq s\} = \phi(M)$.

Notation (Menger [53]): Let $\sigma = \delta_1 \ldots \delta_m$ be an arbitrary sequence of operators in

$$G = \{\beta_{1,2},\ldots,\beta_{1,n}, \beta_{2,3}\ldots,\beta_{2,n},\ldots,\beta_{n-1,n}, \gamma_{1,2},\ldots,\gamma_{1,n},\gamma_{2,3},\ldots,\gamma_{2,n},\ldots,\gamma_{n-1,n}\}$$

(i) the length of σ is denoted by $\$(\sigma) = m$;
(ii) the number of $\gamma_{i,j}$ operators in σ is denoted by $\$\gamma(\sigma)$;
(iii) the sequence of $\beta_{i,j}$ operators which remains after all $\gamma_{i,j}$ operators have been deleted from σ is denoted $\pi\sigma$.

Lemma 7.7 (Menger [53]) If sequence σ is a realization of map M and $\$\gamma(\sigma) = |\phi(M)|$, then $\pi\sigma$ is the realization of a permutation that is compatible with map M.

<u>Lemma 7.8</u> (Menger [53]) For every map M there exists at least one minimum realization σ for which $\$\gamma(\sigma) = |\phi(M)|$.

Lemma 7.8 is a corollary of a theorem by Menger ([53] p.3-35) which states that for a realization σ, of map M, for which $\$\gamma(\sigma) > |\phi(M)|$ a second realization σ' can be constructed for which $\$\gamma(\sigma') < \$\gamma(\sigma)$ and $\$(\sigma') \leq \$(\sigma)$. This theorem, however, does not apply directly to the expanded generator set. The reason being that the additional operators require more conditions to be examined to prove the theorem. Before the theorem can be proved, the irredundancy condition of section 7.3 must be established for the expanded generator set.

<u>Definition 7.10</u> An operator realization σ is <u>γ-redundant</u> if:

(i) there exists an index k and states r and t such that $r < t$, $M_k(r) = M_k(t)$ and $\delta_{k+1} = \gamma_{r,t}$; or

(ii) there exists an index k and states r and t such that $r < t$, $M_k(r) \in \phi(M)$, $M_k(t) \in \phi(M)$, and $\delta_{k+1} = \gamma_{r,t}$.

A realization which is not γ-redundant is said to be <u>γ-irredundant</u>.

Clearly, for cases where σ is γ-redundant, δ_{k+1} can be removed from the operator sequence σ yielding $\sigma' = \delta_1 \ldots \delta_k, \delta_{k+2}, \ldots, \delta_m$, which is still a realization of M.

<u>Theorem 7.6</u> If the sequence $\sigma = \delta_1 \ldots \delta_m$ is a γ-irredundant realization of map M for which $\$\gamma(\sigma) > |\phi(M)|$, then a second realization σ' for M can be constructed satisfying both $\$\gamma(\sigma') < \$\gamma(\sigma)$ and $\$(\sigma') \leq \$(\sigma) = m$.

<u>Proof</u> Since σ is γ-irredundant, there must exist indices k, r, and t (where $r < t$) such that both $M_k(t) \notin \phi(M)$ and $\delta_{k+1} = \gamma_{r,t}$ hold; otherwise, $\$\gamma(\sigma) = |\phi(M)|$. For k and t as given above, define $q = M_k(t)$ and let j be the largest index less than or equal to k such that $\delta_j = \gamma_{s,t}, \beta_{s,t}, \gamma_{t,p}$, or $\beta_{t,p}$.

Assume that such a j does not exist. Then the operator subsequence $\delta_1 \ldots \delta_k$ does not affect the t^{th} position. That is, $t = M_1(t) = \ldots = M_k(t) = q$ and $|M_1^{-1}(q)| = \ldots = |M_k^{-1}(q)| = 1$. Since σ is γ-irredundant, $M_k^{-1}(r) \neq q$. As $M_{k+1} = \gamma_{r,t}(M_k)$, it follows that $|M_{k+1}^{-1}(q)| = 0$.

Thus, $|M^{-1}(q)| = 0$, because a state eliminated in the course of an operator realization cannot reappear later. However, this contradicts $M_k(t) = q \notin \phi(M)$. Therefore, the index j must exist. The remainder of the proof is divided into 4 sections. Each section proves for one of the possible $\delta_j (\gamma_{s,t}, \gamma_{t,p}, \beta_{s,t}$, or $\beta_{t,p})$, that a γ operator in σ can either be deleted from σ or replaced by a β operator, such that the new sequence, σ', is also a realization of M.

(i) Assume $\delta_j = \gamma_{s,t}$; then δ_j can be deleted from σ and the remaining sequence $\sigma' = \delta_1, \ldots, \delta_{j-1}, \delta_{j+1}, \ldots, \delta_m$, is still a realization of M. $\sigma'(E) = M$ can be shown to hold by letting $M_1', \ldots, M_{j-1}', M_{j+1}', \ldots, M_m'$ denote the intermediate maps associated with σ'. Obviously, $M_1' = M_1, \ldots, M_{j-1}' = M_{j-1}$. Let $c = M_{j-1}'(t) = M_{j-1}(t)$. By definition of j, the operator sequence $\delta_{j+1} \ldots \delta_k$ does not affect the t^{th} position. Consequently, $c = M_{j-1}'(t) = M_{j+1}'(t) = \ldots = M_k'(t)$.
For all $d \neq t$, $M_{j+1}'(d) = M_{j+1}(d)$. Therefore,
$M_{j+1}'(d) = M_{j+1}(d), \ldots, M_k'(d) = M_k(d)$, $M_{k+1}'(d) = M_{k+1}(d)$. As $M_{k+1}' = \gamma_{r,t}(M_k')$, $M'_{k+1}(t) = M_k'(r) = M_k(r) = M_{k+1}(t)$, and $M_{k+1}' = M_{k+1}$.
Thus, $\sigma'(E) = M$, $\$\gamma(\sigma') = \$\gamma(\sigma) - 1$ and $\$(\sigma') = m-1$.
(Assumption (i) is represented graphically in Figure 7.6.i.)

(ii) Assume $\delta_j = \gamma_{t,p}$; then δ_j can be replaced by $\beta_{t,p}$ and the resulting sequence $\sigma' = \delta_1 \ldots \delta_{j-1}, \beta_{t,p} \delta_{j+1}, \ldots \delta_m$ is also a realization of M.

$$M_1' = M_1, \ldots, M_{j-1}' = M_{j-1}$$

$$\begin{aligned} M_j' &= \beta_{t,p}(M_{j-1}') \\ &= \beta_{t,p}(M_{j-1}) \\ \therefore M_j'(p) &= M_{j-1}'(t) \\ &= M_{j-1}(t) \end{aligned}$$

$$M_j = \gamma_{t,p}(M_{j-1})$$
$$M_j(p) = M_{j-1}(t) \text{ and } M_j'(p) = M_j(p).$$

$\therefore$ for all c such that $c \neq t$, $M_j'(c) = M_j(c)$ and

$$M_{j+1}'(c) = M_{j+1}(c), \ldots, M'_k(c) = M_k(c),\ M_{k+1}'(c) = M_{k+1}(c)$$
$$M_{k+1} = \gamma_{r,t}(M_k).$$

Thus, $M_{k+1}(t) = M_k(r)$

$M'_{k+1} = \gamma_{r,t}(M'_k)$

$M'_{k+1}(t) = M'_k(r) = M_k(r) = M_{k+1}(t)$

Therefore, $M_{k+1} = M'_{k+1}$ and

$\sigma'(E) = M$, $\$\gamma(\sigma') = \$\gamma(\sigma)-1$, and $\$(\sigma') = \$(\sigma) = m$.
(Refer to Figure 7.6(ii).)

(iii) Assume $\delta_j = \beta_{s,t}$. Then there must exist an ℓ such that $\delta_\ell = \gamma_{u,s}$, $\gamma_{s,v}$, $\beta_{u,s}$, or $\beta_{s,v}$; that such an index ℓ exists is proved by the same argument for the existence of index j.

(a) Assume $\delta_\ell = \gamma_{u,s}$.
By an argument similar to (i), it can be shown that δ_ℓ can be deleted from σ to give $\sigma' = \delta_1 \ldots \delta_{\ell-1}, \delta_{\ell+1} \ldots \delta_m$, which is still a realization of M. Thus,
$\$\gamma(\sigma') = \$\gamma(\sigma)-1$ and $\$(\sigma')=m-1$. (Figure 7.6.iii(a).)

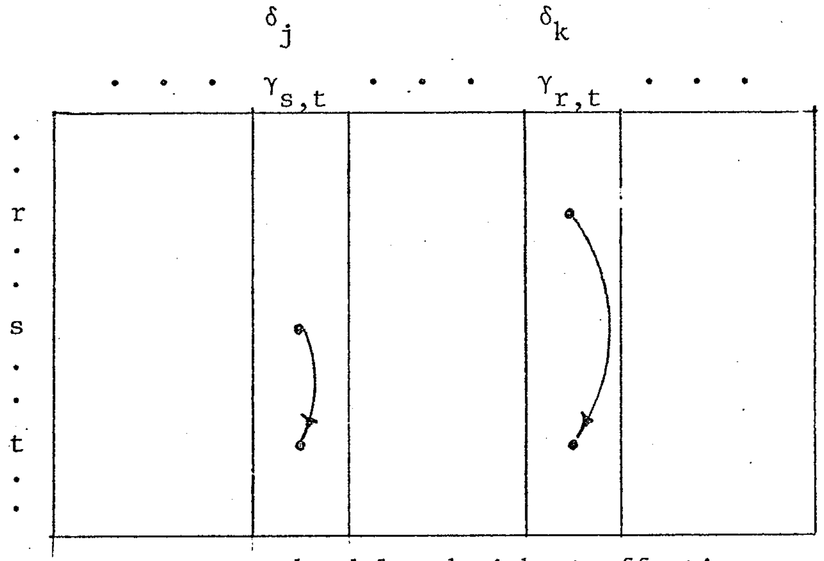
(b) Assume $\delta_\ell = \gamma_{s,v}$
By an argument similar to (ii) it can be shown that δ_ℓ can be replaced by $\beta_{s,v}$ and that $\sigma'(E) = M$,
$\$\gamma(\sigma') = \$\gamma(\sigma)-1$ and $\$(\sigma') = \$(\sigma)$.
(Fig.7.6.iii(b).)

(c) Assume $\delta_j = \beta_{u,s}$, then there must exist an index ℓ', such that $\delta_{\ell'} = \gamma_{p,u}$, $\gamma_{u,w}$, $\beta_{p,u}$, or $\beta_{u,w}$.
Eventually, case (i) or (ii) must hold; otherwise,
$|M_1^{-1}(q)| = \ldots. = |M_k^{-1}(q)| = 1$ so that
$|M_{k+1}^{-1}(q)| = \ldots.. = |M_m^{-1}(q)| = 0$, implying $q \in \phi(M)$, which is a contradiction.

(d) Assume $\delta_\ell = \beta_{s,v}$. Proved as iii(c).

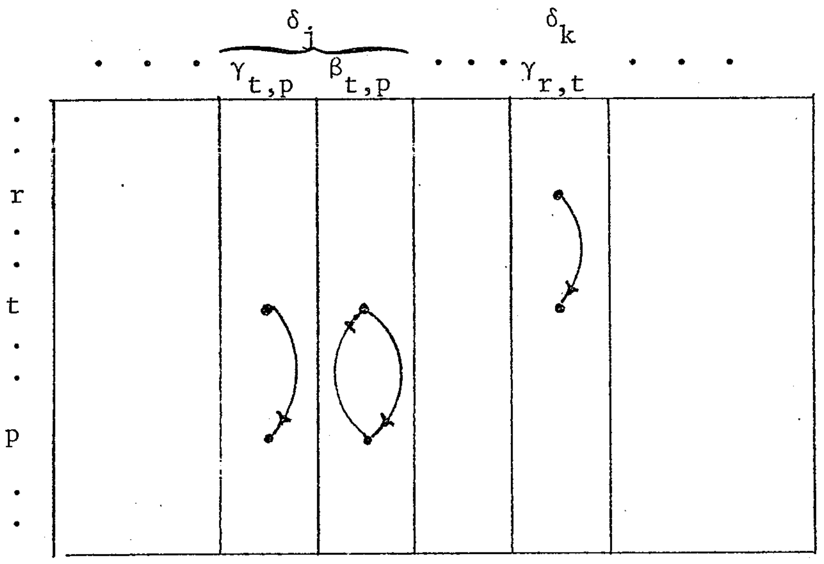
(iv) $\delta_j = \beta_{t,p}$. Proved as iii.

Q. E. D.



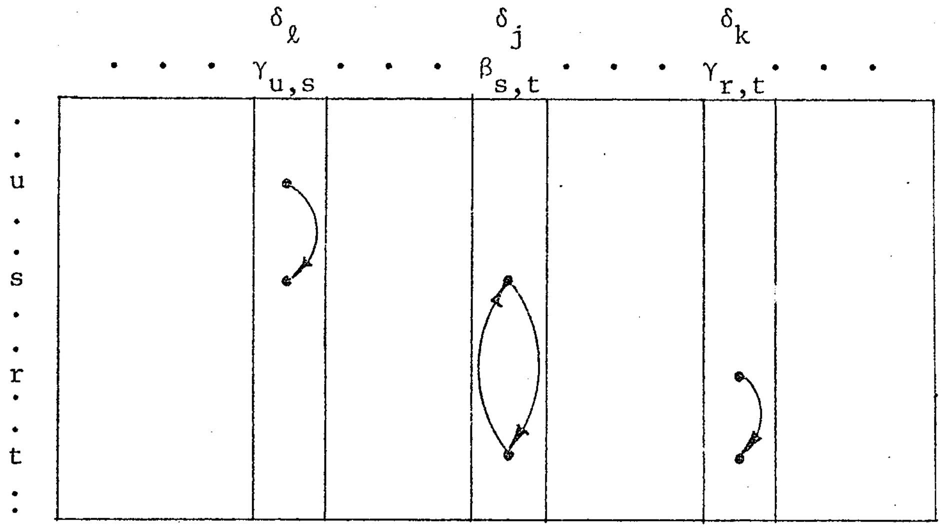
$\gamma_{s,t}$ can be deleted without affecting σ

Figure 7.6.(i)


δ_j
δ_k
$\ldots \gamma_{t,p}\ \beta_{t,p} \ldots \gamma_{r,t} \ldots$
r
t
p

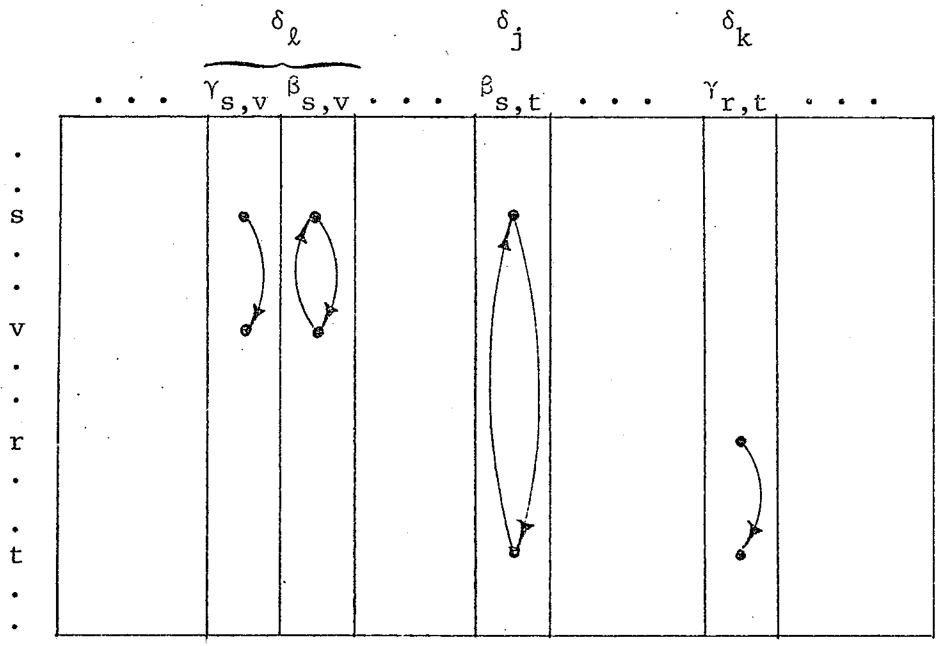
$\gamma_{t,p}$ can be replaced by $\beta_{t,p}$ without affecting σ

Figure 7.6(ii)



$\gamma_{u,s}$ can be deleted

Figure 7.6.(iiia)



$\gamma_{s,v}$ can be replaced by $\beta_{s,v}$

Figure 7.6 (iiib)

Menger uses the set $A'(M,P)$ to determine states for which β operators, in excess of the number required for permutation P, will be required to realize M. The set $A'(M,P)$ can easily be determined by examining map M.

For the expanded set of operators, the set of states for which extra β operators will be required cannot be found as easily. Following, some of the problems associated with this process are illustrated.

Notation: Let $\tau = \delta_1 \ldots \delta_m$ be a sequence of $\beta_{i,j}$ operators which realizes P. That is, $\delta_1 \ldots \delta_m(E) = P$. Denote $E = P_0$, $P_i = \delta_i(P_{i-1})$ and $P_n = P$.

Definition 7.11 For a map M and a permutation P compatible with map M, a state s is said to reset state t in P if and only if there exists a sequence τ, of length $\$(P)$, such that for one P_i, $P_i^{-1}(s) < P_i^{-1}(t)$.

Definition 7.12 For a map M and permutation P compatible with map M, if $M(s) \nmid P(s)$, then M(s) covers P(s).

It should be observed that the set of all covered states is the set $\phi(M)$.

Definition 7.13 If M(s) covers P(s) and M(s) resets P(s) in P, then M(s) replaces P(s) in P.

Example 7.9 $M = <4,4,5,1,1>$ and $P = <4,2,5,1,3>$

$M(2) = 4$ covers $P(2) = 2$

$M(5) = 1$ covers $P(5) = 3$

The minimal operator realization $\tau = \beta_{1,4}, \beta_{3,5}$ realizes permutation P.

For $P_0 = <1,2,3,4,5>$, $P_0^{-1}(1) < P_0^{-1}(3)$ and 1 replaces 3 in τ.

$$P_1 = \beta_{1,4}(<1,2,3,4,5>$$
$$= <4,2,3,1,5>$$

Since $P_1^{-1}(4) < P_1^{-1}(2)$, 4 replaces 2.

Obviously, all replaceable states in P will not require extra β operators, since they can be set equal to their value in M during the realization of M. The set N(M,P) is used to denote all the non-replaceable states in P. For each state s, $s \in N(M,P)$, an extra β operator will be required to realize M.

The function, $¢(\sigma) = |\phi(M)| + \$(P) + |N(M,P)|$, reasonably approximates the number of operators required to realize M, using a permutation P compatible with map M. No formal proof will be given for this function. Instead, it is demonstrated that, in most cases, the evaluation of this function is too difficult to allow the function to be used as a tool for deriving minimal realizations.

<u>Example 7.10</u>

M = <4,3,2,4> and
P = <1,3,2,4>, compatible with map M.

The sequence $\tau = \beta_{2,3}$ realizes P.
Thus, $\$(P) = 1$ and $\phi(M) = \{1\}$
Since the only minimal sequence for P is $\beta_{2,3}$, state 4 does not reset 1 in P.

$$N(M,P) = \{1\}$$

calculating $¢(\sigma)$,

$$¢(\sigma) = |\phi(M)| + \$(P) + |N(M,P|$$
$$= 1 + 1 + 1$$
$$= 3$$

An operator sequence realizing M and requiring 3 operators is

	$\beta_{2,3}$	$\beta_{1,4}$	$\gamma_{1,4}$
1	1	4	4
2	3	3	3
3	2	2	2
4	4	1	4

Notation: Let the term c_s denote the cycle of P which contains state s. The smallest state in the cycle c_s will be denoted by min (c_s).

Lemma 7.9 For a permutation P, state s resets state t if one of the following holds:

(i) $s < t$

(ii) $s < P^{-1}(t)$

(iii) $P^{-1}(s) < t$

(iv) $P^{-1}(s) < P^{-1}(t)$

(v) $\min(c_s) < t$

(vi) $\min(c_s) < P^{-1}(t)$.

Proof It can easily be seen that a minimal realization, $\tau = \delta_1, \ldots \delta_m$, of P can be constructed such that

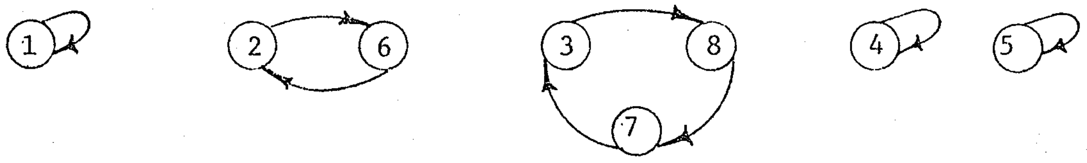
$$P_k^{-1}(s) < P_k^{-1}(t) \quad \text{for} \quad P_k = \delta_k(\ldots.\delta_1(E)),\ k=1,\ldots,m\ .$$

Lemma 7.10 Conditions (i) and(iii) of Lemma 7.9 can be replaced by condition (v); conditions (ii) and (iv) can be replaced by (vi).

Proof Obvious.

For a permutation P compatible with map M we can determine, using Lemma 7.10, whether there exists a sequence of $\beta_{i,j}$ operators such that state s resets state t. However, there may not exist a unique minimal sequence τ, such that every replaceable state can be reset in the sequence τ.

Example 7.11 Let M = <1,8,8,4,5,3,3,7> and P = <1,6,8,4,5,2,3,7>, compatible with map M. 8 covers 6 and 3 covers 2.



From the cycle diagrams

$\min(c_8) = 3 < 6$ and 8 covers 6

$\min(c_3) = 3 < P^{-1}(2) = 6$ and 3 covers 2.

The length of a minimal operator realization of P is

$$\$(P) = \sum_{i=1}^{p} (|c_i|-1)$$

$$= (1-1) + (2-1) + (3-1) + (1-1) + (1-1)$$

$$= 3$$

Even though there are no nonreplaceable states in P, it is not possible to find a realization of P such that all covered states are replaced. The possible minimal realizations of P are

(i) $\beta_{2,6}\ \beta_{3,7}\ \beta_{3,8}$ (ii) $\beta_{3,7}\ \beta_{3,8}\ \beta_{2,6}$ (iii) $\beta_{3,7}\ \beta_{2,6}\ \beta_{3,8}$

	$\beta_{2,6}$	$\beta_{3,7}$	$\beta_{3,8}$
1	1	1	1
2	6	6	6
3	3	7	8
4	4	4	4
5	5	5	5
6	2	2	2
7	7	3	3
8	8	8	7

$\beta_{3,7}$	$\beta_{3,8}$	$\beta_{2,6}$
1	1	1
2	2	6
7	8	8
4	4	4
5	5	5
6	6	2
3	3	3
8	7	7

$\beta_{3,7}$	$\beta_{2,6}$	$\beta_{3,8}$
1	1	1
2	6	6
7	7	8
4	4	4
5	5	5
6	2	2
3	3	3
8	8	7

Figure 7.7

For Figure 7.7.(i) 3 resets 2, but 8 does not reset 6; for Figure 7.7 (ii) 8 resets 6, but 3 does not reset 2. Neither 8 resets 6 nor 3 resets 2 in Figure 7.7(iii).

Definition 7.14 Assume for permutation P compatible with map M, there exists minimal realizations τ_1 and τ_2 such that state s replaces state t and state x replaces state y, but there does not exist a minimal realization τ such that both s replaces t and x replaces y. The states [s,t] and [x,y] are said to be mutually nonreplaceable in P.

<u>Theorem 7.7</u> If [s,t] and [x,y] are mutually nonreplaceable in P, then s and x are in the same cycle c_1 and t and y are in the same cycle c_2, $c_1 \neq c_2$.

<u>Proof</u> All possible cyclic combinations of s,t,x, and y are examined. It is shown that the only possible combination for which a minimal realization of P can be constructed such that [s,t] and [x,y] are mutually nonreplaceable is (..,s,..,x,..),(..,t,..,y,..). For all other combinations it is demonstrated that a realization such that s resets t and x resets y can be constructed.

(1) (..,s,..), (..,t,..), (..,x..), (..,y,..); put s and x into positions $\min(c_s)$ and $\min(c_x)$, respectively. It can be seen that s resets t and x resets y, contradicting the assumption of mutual nonreplaceability. (We are making use of the fact that the cycle operators can be applied in any order, as long as the operator order within a cycle is observed.)

(2) (..,s,..,t,..), (..,x,..), (..,y,..). Clearly, an operator sequence such that s resets t and x resets y can be derived.

(3) (..,s,..,x,..), (..,t,..), (..,y,..).

$\min(c_s) = \min(c_x) = u.$

$\therefore\ u < t$ or $u < P^{-1}(t)$

$u < y$ or $u < P^{-1}(y)$

Apply β operators for (..,t,..) and (..,y,..) if $u < P^{-1}(t)$ and $u < P^{-1}(y)$, respectively. Put s and x into position u, alternately. Thus, for this realization s resets t and x resets y.

(4) (..,s,..,y,..), (..,t,..), (..,x,..)

Apply β operators to place x into position $\min(c_x)$.

If $\min(c_x) < y$, then x can reset y;

if $\min(c_s) < t$, then apply β operators for (..,s,..y,..) so that s can reset t;

if $\min(c_s) < P^{-1}(t)$, s can reset t if the β operators for t are applied.

(5) (..,s,..), (..,t,..), (..,x,..,y,..). This case is similar to (2).

(6) (..,s,..), (..x,..,t,..), (..,y,..). An operator sequence can be derived in a manner similar to (4).

(7) (..,s,..), (..t,..,y,..), (..,x,..). This case is similar to (1) and an operator realization is constructed by applying operators to place s and x into position $\min(c_s)$ and $\min(c_x)$.

(8) (..,s,..,t,..,x,..), (..,y,..). If $\min(c_x) < P^{-1}(y)$, apply β operators for (..,y,..) such that x resets y. Since s and t are in the same cycle, s resets t. If $\min(c_x) < y$, the operators for (..,s,..t,..,x,..) must be the first operators in the sequence.

(9) (..,s,..,t,..,y,..), (..,x,..). Any operator sequence which first places x into position $\min(c_x)$ can be used.

(10) (..,s,..,x,..,y,..), (..t,..). This case is similar to (8).

(11) (..,t,..,x,..,y,..), (..,s,..). This case is similar to (9).

(12) (..,s,..,t,..), (..,x,..,y,..). Clearly, for any operator realization, s resets t and x resets y.

(13) (..,s,..,y,..), (..,x,..,t,..).
Assume $\min(c_s) < \min(c_x)$. First apply β operators to put x into position $\min(c_x)$. If $\min(c_x) < y$, then x resets y. If $\min(c_x) < P^{-1}(y)$ apply β operators for (..,s,..,y,..) so that x can reset y. Since $\min(c_s) < \min(c_x)$, then $\min(c_s) < t$ and $\min(c_s) < P^{-1}(t)$ and t can be reset by s while β operators for (..,s,..,y,..) are being applied.
A similar operator sequence can be constructed if $\min(c_x) < \min(c_s)$.

(14) (..,s,..,t,..,x,..,y,..). Obviously, s resets t and x resets y in any operator realization.

(15) (..,s,..,x,..), (..,t,..,y,..).
For this case, a minimal operator realization such that s resets t and x resets y cannot be constructed. Thus, this is the possible combination of cycles for which [s,t] and [x,y] are mutually nonreplaceable.

Theorem 7.8 If [s,t] and [x,y] are mutually nonreplaceable in permutation P, then there exists a permutation Q, $Q = P\cdot(s,t)\cdot(x,y)$ such that [s,t] and [x,y] are not mutually nonreplaceable in Q and $\$(Q) = \$(P)$.

Proof For [s,t] and [x,y] to be mutually nonreplaceable in P,

$$P^{-1}(s) > P^{-1}(t) \text{ and } P^{-1}(x) > P^{-1}(y).$$

For $Q = P\cdot(s,t)\cdot(x,y)$, $Q^{-1}(s) < Q^{-1}(t)$ and $Q^{-1}(x) < Q^{-1}(y)$.

Thus [s,t] and [x,y] are not mutually nonreplaceable in Q.

Q is compatible with map M by application of AR2($t,y \in \phi(M)$).

Let $Q_1 = P\cdot(s,t)$.

Since, s and t are not in the same cycle in P, they are in the same cycle in Q_1,

$\therefore$ $\$(Q_1) = \$(P)+1$. ($Q_1$ has one less cycle than P and thus one more β operator than P.)

In addition x and y are in the same cycle in Q_1.

Let $Q = Q_1\cdot(x,y)$. x and y are not in the same cycle in Q.

$\therefore$ $\$(Q) = \$(Q_1)-1 = \$(P)$.

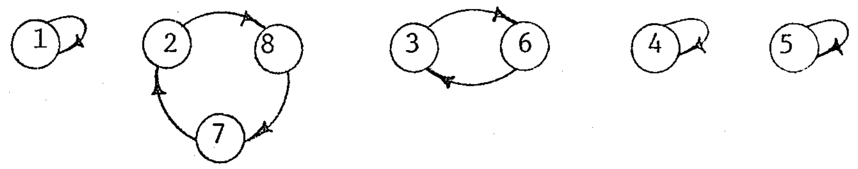
Example 7.12 M = <1,8,8,4,5,3,3,7> and P = <1,6,8,4,5,2,3,7> .
In Example 7.11 it was shown that [8,6] and [3,2] are mutually non-replaceable in P.

Applying Theorem 7.8

$$Q = P\cdot(8,6)\cdot(3,2)$$
$$= \langle 1,8,6,4,5,3,2,7\rangle$$

Since $Q^{-1}(8) = 2 < 3 = Q^{-1}(6)$ and $Q^{-1}(3) = 6 < 7 = Q^{-1}(2)$, [8,6] and [3,2] are not mutually nonreplaceable in Q.

Q :



$\therefore$ $\$(Q) = 3 = \$(P)$ and Q compatible with map M.

As can be seen from Examples 7.11 and 7.12 detection of mutually nonreplaceable states is an ad hoc procedure, which requires examination of the minimal realization of a permutation. Another problem in the detection of all the replaceable states is illustrated below.

<u>Example 7.13</u> $M = <4,5,3,5,5,2>$ and $P = <4,6,3,1,5,2>$

P:
$$\begin{tikzcd}
1 \arrow[r, bend left] & 4 \arrow[l, bend left] & 2 \arrow[r, bend left] & 6 \arrow[l, bend left] & 3 \arrow[loop right] & 5 \arrow[loop right]
\end{tikzcd}$$

5 covers 6; since $\min(c_5) < 6$, 5 replaces 6.

5 covers 1; but since $\min(c_5) \nmid 1$ and $\min(c_5) \nmid P^{-1}(1)$, 5 does not replace 1. Consequently, an extra β operator will be required to replace 1.

However, by first replacing 6 by 5, 1 can be replaced by 5 without intoducing an extra operator. The replacement of a state by another replaced state is referred to as <u>indirect replacement</u>. The sequence $\beta_{1,4}$, $\gamma_{5,6}$, $\beta_{2,6}$, $\gamma_{2,4}$ is a realization of M.

	$\beta_{1,4}$	$\gamma_{5,6}$	$\beta_{2,6}$	$\gamma_{2,4}$
1	4	4	4	4
2	2	2	5	5
3	3	3	3	3
4	1	1	1	5
5	5	5	5	5
6	6	5	2	2

Obviously, the concept of indirect replacement can be extended to n^{th} level indirect replacement, where there exists a chain of n replaced states.

Menger was able to determine the set A'(M,P) directly from map M. Thus, to minimize \$'(M,P), a permutation P compatible with map M such that \$(P) was minimal, had to be found. However, as demonstrated, to determine N(M,P) requires each permutation P compatible with map M to be examined separately. As there are a considerable number of compatible permutations for any map M, examining each permutation is not feasible. The algorithm, which is presented later, will not consider N(M,P) when determining the permutation to use in realizing M. The justification for dropping N(M,P) from the cost formula is as follows.

For each state n in $N(M,P)$ an extra β operator will be required. This β operator exchanges state n and a state u, where $u \in \phi(M)$ and $n = M(P^{-1}(u))$. (If there exists more than one possible state u, the state in the lowest position of P will be chosen to ensure replaceability.)

The new permutation, $Q = P(u,n)$ is compatible with map M, by AR2 of Menger. Similarly, the permutation R resulting after the application of all β operators required by $N(M,P)$ is compatible with map M. The permutation R is the minimal permutation compatible with map M for which no nonreplaceable states exist. Consequently, our algorithm is designed to determine R for any map M.

The steps in Phase I of the algorithm attempt to set $N(M,P)$ to the null set and at the same time reduce $\$(P)$. Whenever a conflict between the aims occurs, the algorithm opts for obtaining a minimal realization.

The reason for this is that by reducing $\$(P)$, $N(M,P)$ may or may not be increased. Thus, there is the possibility of reducing $¢(\sigma)$. However, by minimizing $N(M,P)$, $|N(M,P)|$ decreases by 1, but $\$(P)$ increases by 1 and $¢(\sigma)$ remains constant.

During Phase II the necessary $|\phi(M)|$ γ operators are found by comparing M and P. Next the order in which the γ operators and the $\$(P)$ β operators must be applied to realize M is determined. Unfortunately, this part of the algorithm has not been formalized and considerable searching may be required to determine the correct sequence.

At this time, nonreplaceable states, if any exist, must also be found. As demonstrated a state which appears nonreplaceable may be indirectly replaceable. Consequently, the determination of nonreplaceable states requires an exhaustive search.

Method B of Phase II presents a straightforward method of deriving an operator realization. This method, however, does not guarantee the shortest realization. Once the reader is familiar with the operators, improvements to the operator sequence found by Method B can easily be found.

Algorithm

Phase I

(1) For all states p such that $|M^{-1}(p)| = 1$ and $M(s) = p$, set $R(s) = p$.

(2) For all states p such that $|M^{-1}(p)| \geq 2$ and $p \in M^{-1}(p)$, set $R(p) = p$.

(3) For all states p such that $|M^{-1}(p)| \geq 2$ and $p \notin M^{-1}(p)$, set $R(s) = p$ where $s = \min(M^{-1}(p))$.

(4) The remaining states $p \in \phi(M)$, are assigned positions in P arbitrarily.

(5) Menger's rules AR1 and AR2 are then applied, giving permutation P, in order to increase the number of cycles. That is, reduce the number of β operators required to realize P. (This step may increase the number of nonreplaceable states.)

Phase II

Method A

(1) Determine the γ operators.

(2) Determine the nonreplaceable states N(M,P).

(3) Derive a sequence realizing M using the $|\phi(M)|$ γ operators, the $\$(P)$ β operators and the $|N(M,P)|$ β operators.

Method B

(1) Determine the γ operators and apply as many as possible to the identity map E.

(2) Apply the first $\$(P)$ β operator. (Note the β operators are ordered in the sequence they were determined by the permutation algorithm.)

(3) Apply as many γ operators as possible; if there are no more β operators belonging to P, go to (5).

(4) Apply the next β operator; go to (3).

(5) If $M = P$, algorithm stops.
If $M \neq P$, then N(M,P) is not empty. Apply the necessary β and γ operators in order to obtain M. (Ideally $¢(\sigma) = |\phi(M)| + \$(P)$.)

Example 7.14 The algorithm will be applied to the map used in Menger [53].

M = <1,1,1,3,3,20,7,7,10,9,9,12,14,13,13,15,1,17,17,12>

ϕ(M) = {2,4,5,6,8,11,16,18,19}

Phase I

Step 1: R = <-,-,-,-,-,20,-,-,10,-,-,-,14,-,-,15,-,-,-,->

Step 2: R = <1,-,-,-,-,20,7,-,10,-,-,12,14,-,-,15,-,-,-,->

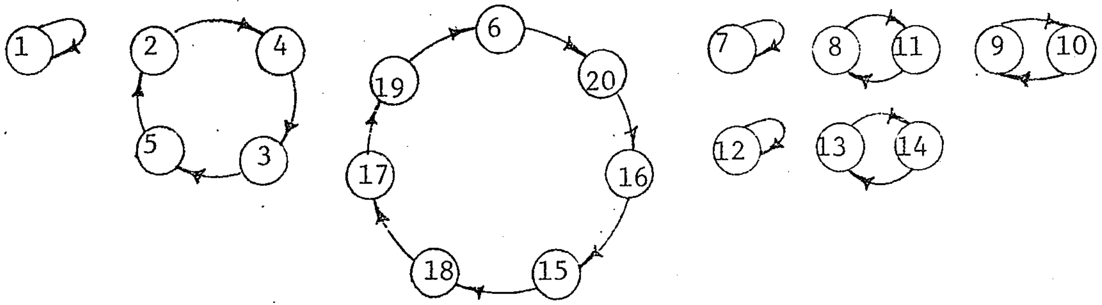
Step 3: R = <1,-,-,3,-,20,7,-,10,9,-,12,14,13,-,15,-,17,-,->

Step 4: Clearly, at this point for u $\in$ ϕ(M) and R(u) not assigned a state, we should set R(u) = u in order to reduce the number of permutation operators. However, to show that Menger's rules, AR1 and AR2, apply no matter how the states in ϕ(M) are assigned, this will not be done.

R = <1,4,5,3,2,20,7,11,10,9,8,12,14,13,18,15,19,17,6,16>

Step 5:

R =



For cycle (2,4,3,5) states 2,4 $\in$ ϕ(M);

(6,20,16,15,18,17,19) 6,16 $\in$ ϕ(M);

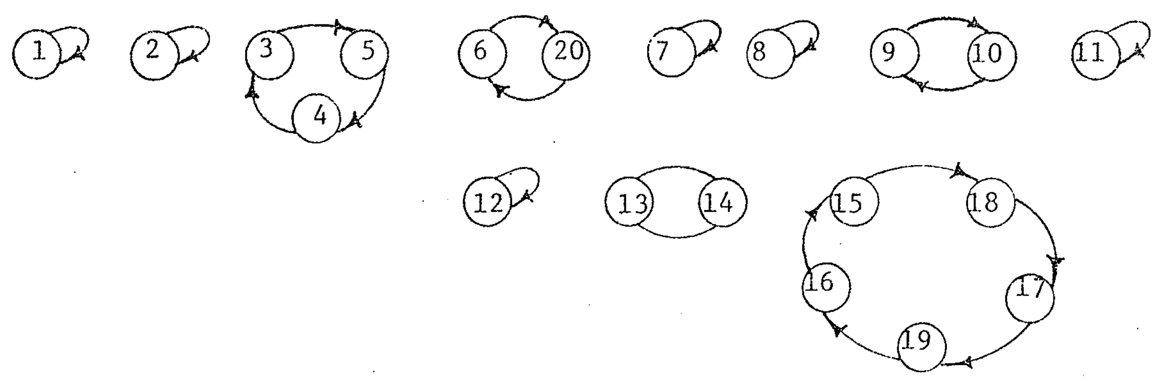
(8,11) 8,11 $\in$ ϕ(M)

Applying the indicated permutations to R

R_1 = R·(2,4)·(6,16)·(8,11)

= <1,2,5,3,4,20,7,8,10,9,11,12,14,13,18,15,19,17,16,6>

R_1:



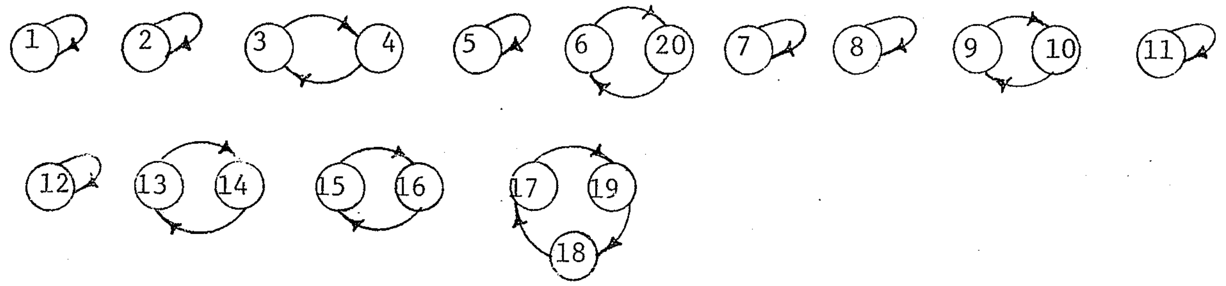
For (3,5,4), $4,5 \in \phi(M)$

(15,18,17,19,16), $16,18 \in \phi(M)$

$R_2 = R_1 \cdot (4,5) \cdot (16,18)$

$= <1,2,4,3,5,20,7,8,10,9,11,12,14,13,16,15,19,17,18,6>$

R_2:

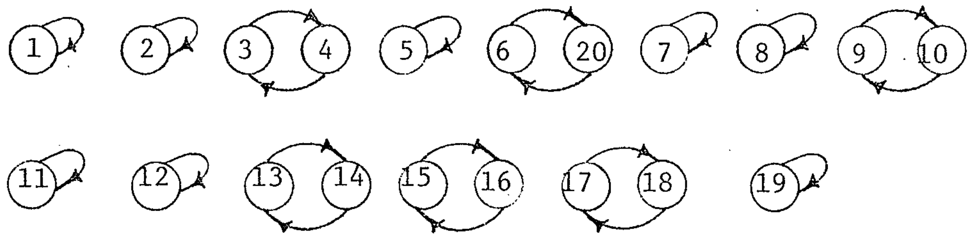


For (17,19,18), $18,19 \in \phi(M)$

$R_3 = R_2 \cdot (18,19)$

$= <1,2,4,3,5,20,7,8,10,9,11,12,14,13,16,15,18,17,19,6>$

R_3:



Since no further cycles can be induced, let $P = R_3$. The minimal operator realization for P requires 6 operators, as determined by the permutation algorithm:

$$\beta_{3,4}, \beta_{6,20}, \beta_{9,10}, \beta_{13,14}, \beta_{15,16}, \beta_{17,18}$$

No nonreplaceable states exist. Thus, a minimal operator realization for M requires

$$\begin{aligned} ¢(\sigma) &= |\phi(M)| + \$(P) + |N(M,P)| \\ &= 9 + 6 + 0 \\ &= 15 \text{ operators.} \end{aligned}$$

Method B will be used for Phase II. The steps in the method will not be stated explicitly, but are obvious on examination of Figure 7.8.

	$\gamma_{1,2}$	$\gamma_{1,4}$	$\gamma_{1,18}$	$\gamma_{3,5}$	$\gamma_{7,8}$	$\gamma_{9,11}$	$\gamma_{13,16}$	$\gamma_{17,19}$	$\beta_{3,4}$	$\beta_{6,20}$	$\gamma_{12,20}$	$\beta_{9,10}$	$\beta_{13,14}$	$\beta_{15,16}$	$\beta_{17,18}$
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
4	4	1	1	1	1	1	1	1	3	3	3	3	3	3	3
5	5	5	5	3	3	3	3	3	3	3	3	3	3	3	3
6	6	6	6	6	6	6	6	6	6	20	20	20	20	20	20
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	7	7	7	7	7	7	7	7	7	7	7
9	9	9	9	9	9	9	9	9	9	9	9	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	9	9	9	9
11	11	11	11	11	11	9	9	9	9	9	9	9	9	9	9
12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
13	13	13	13	13	13	13	13	13	13	13	13	13	14	14	14
14	14	14	14	14	14	14	14	14	14	14	14	14	13	13	13
15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
16	16	16	16	16	16	16	13	13	13	13	13	13	13	15	15
17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	1
18	18	18	1	1	1	1	1	1	1	1	1	1	1	1	17
19	19	19	19	19	19	19	19	17	17	17	17	17	17	17	17
20	20	20	20	20	20	20	20	20	20	6	12	12	12	12	12

Figure 7.8

For map M, method B derived a minimal realization. (Menger obtained a realization of M which required 25 operators. Thus, for this example there is a considerable saving resulting from the use of the expanded operator set.)

Notation: Let $\lambda(M) = \{s \mid |M^{-1}(s)| \geq 2\}$.

Using the set $\lambda(M)$, an upper bound on the number of operators required using method B can be established. First, a bound on the number of extra β operators must be found.

For Step (5) of Method B, if $M \nmid P$ then extra β operators must be applied to remove nonreplaceable states. Assume the worst case, that is $\phi(M) = N(M,P)$. To remove all nonreplaceable states will require $|\lambda(M)|$ β operators. These β operators will place each state $s \in \lambda(M)$ at $P(\min(M^{-1}(s)))$. From this position, s resets every state $P(t)$, where $M(t) = s$.

Thus, an upper bound on the operators required using Method B is

$$¢(\sigma) \leq |\phi(M)| + \$(P) + |\lambda(M)|$$

Example 7.15

$$M = <5,5,3,1,5,2,10,10,8,9> \ , \ \phi(M) = \{4,6,7\}$$
$$\lambda(M) = \{5,10\}$$

Phase I

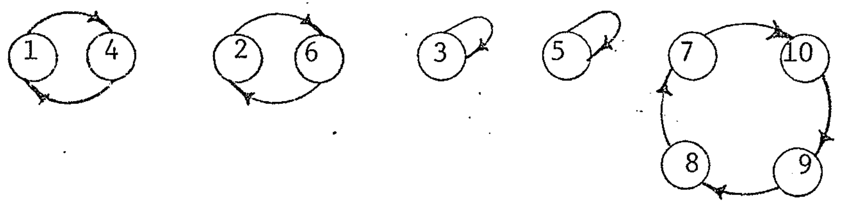
Step 1: R = <–,–,3,1,–,2–,–,–,8,9>

Step 2: R = <–,–,3,1,5,2,–,–,8,9>

Step 3: R = <–,–,3,1,5,2,10,–,8,9>

Step 4: R = <4,6,3,1,5,2,10,7,8,9>

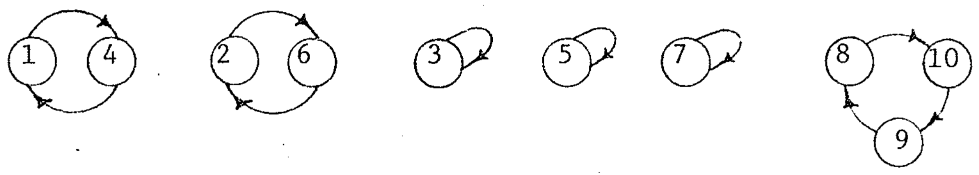
Step 5:



For (7,10,9,8), $7 \in \phi(M)$, $M(R^{-1}(7)) = M(8) = 10$ (AR2)

Applying (7,10)

$R_1 = <4,6,3,1,5,2,7,10,8,9>$

$R_1 =$ 

<u>Phase II</u> (Method B)

$$\begin{aligned} ¢(\sigma) &\le |\phi(M)| + \$(P) + |\lambda(M)| \\ &\le 3 + 4 + 2 \\ &\le 9 \end{aligned}$$

Thus, a maximum of 9 operators will be required using Method B.

	$\gamma_{5,6}$	$\beta_{1,4}$	$\beta_{2,6}$	$\beta_{8,9}$	$\beta_{8,10}$
1	1	4	4	4	4
2	2	2	5	5	5
3	3	3	3	3	3
4	4	1	1	1	1
5	5	5	5	5	5
6	5	5	2	2	2
7	7	7	7	7	7
8	8	8	8	9	10
9	9	9	9	8	8
10	10	10	10	10	9

Step (5) is entered after the last of the $(P) β operators, $\beta_{8,10}$, is applied. At this point M has not been realized. However, applying β operators $\beta_{1,2}$ and $\beta_{7,8}$ and γ operators $\gamma_{1,2}$ and $\gamma_{7,8}$ produces M.

	$\beta_{1,2}$	$\beta_{7,8}$	$\gamma_{1,2}$	$\gamma_{7,8}$
4	5	5	5	5
5	4	4	5	5
3	3	3	3	3
1	1	1	1	1
5	5	5	5	5
2	2	2	2	2
7	7	10	10	10
10	10	7	7	10
8	8	8	8	8
9	9	9	9	9

In this example an operator realization of M uses the maximum number of operators, 9. This number can be reduced by rearranging the operators as allowed for in Method A.

	$\gamma_{5,6}$	$\beta_{2,6}$	$\gamma_{2,4}$	$\beta_{1,4}$	$\beta_{8,9}$	$\beta_{8,10}$	$\beta_{7,8}$	$\gamma_{7,8}$
1	1	1	1	5	5	5	5	5
2	2	5	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3
4	4	4	5	1	1	1	1	1
5	5	5	5	5	5	5	5	5
6	5	2	2	2	2	2	2	2
7	7	7	7	7	7	7	10	10
8	8	8	8	8	9	10	7	10
9	9	9	9	9	8	8	8	8
10	10	10	10	10	10	9	9	9

By applying $\beta_{2,6}$ before $\beta_{1,4}$, state 4 can be indirectly replaced by state 5 (via state 6). However, there is no way to indirectly replace state 7 with state 10 and an extra β operator is required. (A realization using Menger's operators requires 11 operators.)

7.5 Cellular Realizations of Generators

A cellular array, which solves the problems associated with the implementation of Menger's generators using Krishnan and Smith's array [45], is presented in this section. The notation used to describe the transposition and partially reset machines is modified from that used by Krishnan and Smith, in order to facilitate the de·cription of the cellular array properties. The transposition and partially reset machines for an n+1 state machine are given below.

	T_{01}	T_{02}	....	T_{0n}	S_{01}	S_{02}	....	S_{0n}
0	1	2		n	0	0		0
1	0	1		1	0	1		1
2	2	0		2	2	0		2
•								
•								
•								
•								
n	n	n		0	n	n		0

Figure 7.9

(The transposition and partially reset machines correspond to the b and c generators, respectively.)

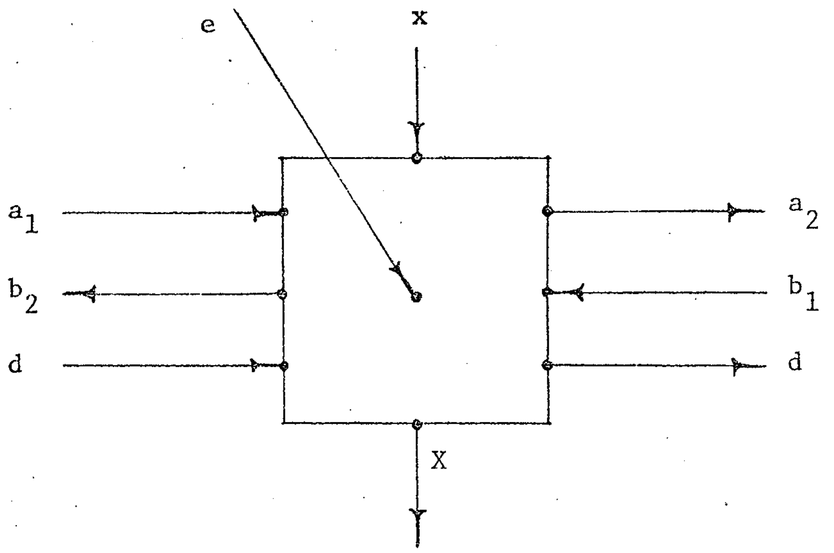
We retain Krishnan and Smith's coding algorithm. However, modifications are necessary to reflect the new notation.

Algorithm (1) Code state 0 with binary vector (0,...,0)

(2) Code the remaining states, i=2,...,n with the binary equivalent to 2^{i-1}.

The present state of the machine will be indicated by the vector $\bar{x} = (x_1,\ldots,x_n)$ and the next-state by vector $\bar{X} = (X_1,\ldots,X_n)$.

Control values are used to determine the operation of each cell in the array. The basic cell is illustrated in Figure 7.10.



where, $a_2 = a_1 + x,$

$b_2 = b_1 + x,$ and

$X = x\bar{e} + \bar{a}_1\bar{b}_1 + \bar{x}\,e\,d$

Figure 7.10

A one-dimensional array of length n, to realize a transposition machine, T_{0j}, or a partially reset machine, S_{0j}, is obtained by setting the e and d parameters of the array. A 0 input is provided at the leftmost a_1 terminal and the rightmost b_1 terminal of the array. Figure 7.11 shows the cellular interconnections necessary for the array.

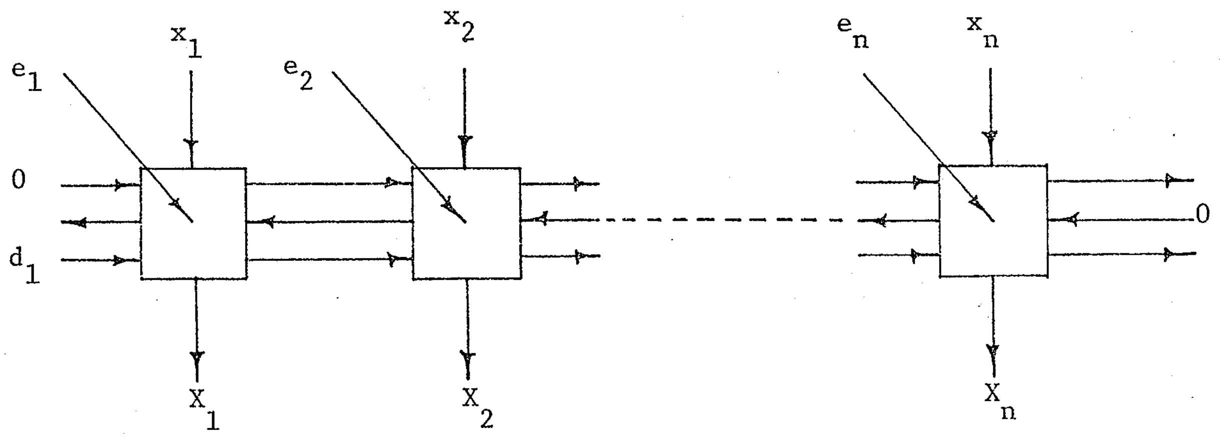


Figure 7.11

The following lemmas justify the use of the cellular arrays in realizing Menger's generators.

<u>Lemma 7.11</u> The transposition machine T_{0j} can be realized if $e_j = 1$, $e_i = 0 \ \forall_i \neq j$, and $d_1 = 1$.

<u>Proof</u> For $i \neq j$, $X_i = x_i$ and $X_j = \bar{a}_1\bar{b}_1\bar{x}_j$

(i) For state s_k, where $s_k \neq s_0$ and $s_k \neq s_j$, $x_k = 1$
Thus $a_{1_j} = 1$ or $b_{1_j} = 1$ and $X_j = 0$.
The remaining x_i values are not changed so

$$(X_1,\ldots,X_n) = (x_1,\ldots,x_n).$$

(ii) For state s_0, $a_{1_j} = 0$ and $b_{1_j} = 0$

$$\begin{aligned} X_j &= \bar{x}_j \\ &= 1 \end{aligned}$$

$$\therefore (0,\ldots,\underset{j}{0},\ldots,0) \Rightarrow (0,\ldots,\underset{j}{1},\ldots,0)$$

(iii) For state s_j, $a_{1_j} = 0$ and $b_{1_j} = 0$

$$\begin{aligned} X_j &= \bar{x}_j \\ &= 0 \end{aligned}$$

$$\therefore (0,\ldots,\underset{j}{1},\ldots,0) = (0,\ldots,\underset{j}{0},\ldots,0)$$

Thus, all required permutations are realized by the array.

<u>Lemma 7.12</u> The partially reset machine S_{0j} can be realized if $e_j = 1$, $e_i = 0 \ \forall_i \neq j$, and $d_1 = 0$.

<u>Proof</u> $X_i = x_i$, for $i \neq j$ and $X_j = 0$

$$\therefore (X_1,\ldots,X_j,\ldots,X_n) = (x_1,\ldots,\underset{j}{0},\ldots,x_n)$$

That is, for state s_k, $s_k \neq s_j$, the state vector X is unchanged. For state s_j, $X_i = 0$, for $i = 1,\ldots,n$. Thus the next state is s_0.

The application of the cellular array to the realization of a single column nonpermutation machine is shown in the next example.

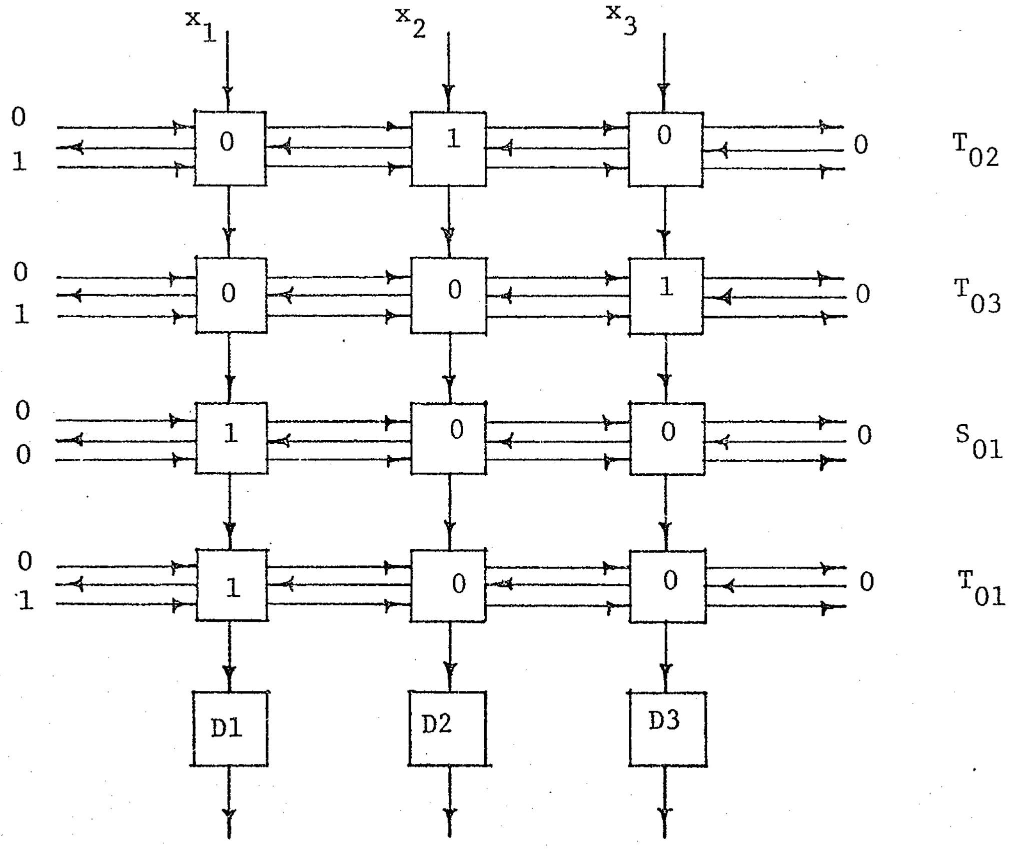
Example 7.16

	M
0	2
1	1
2	3
3	1

The generator sequence derived using Menger's algorithm is

	T_{02}	T_{03}	S_{01}	T_{01}
0	2	2	2	2
1	1	1	0	1
2	0	3	3	3
3	3	0	0	1

A cellular array realization of the generator sequence is



As can be seen the array uses identical cells with regular intercellular connections. Consequently, a two-dimensional array of cells can easily be fabricated with permanent interconnections. Then by setting the boundary and e values, any series of generators may be produced.

An obvious realization for the proposed cell is

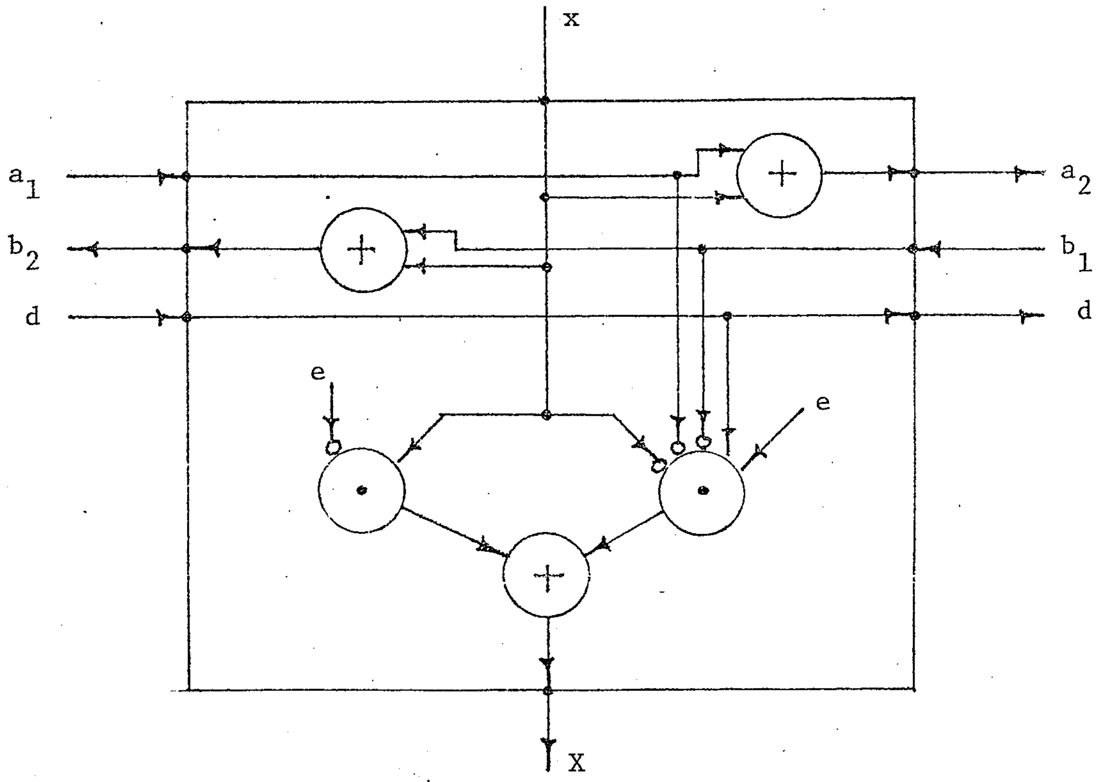


Figure 7.12

Using a separate line to provide the e values to each cell would require an excessive number of boundary connections for integrated circuit technology. An alternative would be to use a programmable array which would contain the e values in a register in each cell. Either a linear or a coincident select technique could be used.

7.6 Conclusion

Menger's synthesis method for sequential machines has been examined in detail in this chapter. The reason for this detailed study, is that an understanding of Menger's techniques is essential to any research into the use of a larger generator set for machine synthesis. In fact, any such studies would appear to inevitably use Menger's theory as a basis, as indicated by our own research.

The benefit of using a larger generator set has been demonstrated by Menger; a sequential machine synthesis will require fewer generators. The length of a generator sequence is now a linear function $(\frac{3n}{2})$ of the number of states, rather than an exponential function, as with Haring's generators. The benefit of a shorter sequence becomes apparent when we consider the reduced cost and also the shorter delay time of a realization.

The logical question arising from Menger's work is, "If 2(n-1) generators can reduce the generator sequence length, could not a larger generator set reduce the length even more?". Section 7.4 is an examination of this question. Although not answering the question conclusively for both permutation and non-permutation maps, it has provided further insight into generator realizations.

Increasing the number of permutation generators has a measureable effect on permutation realizations; the length of the permutation realization can be reduced. The maximum length, using the expanded generators, was shown to be n-1, where n is the number of states. For small machines there is not a great difference between the length of Menger's realizations and ours. However, as the number of states increases there is the possibility of substantial savings using the expanded generator set.

The realization of a permutation, with the expanded generator set, is straightforward and guarantees the minimal realization each time. This is to be expected for any generator set, since a permutation algorithm simply has to find separate realizations for each cycle of the permutation. As the cycles of a permutation are well defined, a generator realization should also be well defined.

The problem in using the expanded generator set occurs when realizing non-permutation maps. The reason for the problem is the multiplicity of opportunities for replacing a state t, $t \in \phi(M)$.

For Menger's algorithm, all states q which covered a state $t \in \phi(M)$ but which could not replace t, were easily determined. This was because a state could not reset another state unless it was contained in a non-unit cycle. However, with the expanded generator set a state q can reset other states and still be in a unit cycle. Thus, the problem of whether or not a state q in a permutation P compatible with map M could replace another state t, is not resolvable by a simple examination of map M. The solution to the problem necessitates an exhaustive search of all permutations P compatible with map M, in order to find a minimal realization. Clearly, this problem can be expected to occur in any expanded generator set, where there is more than one position from which a state can replace another state.

Menger demonstrated that the upper bound for $\$'(M,P)$ for any map was $\frac{3n}{2}$. At present we have not been able to establish an absolute bound for any map realization using the expanded generator set. However, from examining some possible worst cases, it appears likely that an upper limit on a map realization is n generators. The reasoning behind this assumption follows:

The cost function for a realization σ of a map M is

$$¢(\sigma) \leq |\phi(M)| + \$(P) + |\lambda(M,P)|.$$

We will consider $¢(\sigma)$ when each of its components approaches a maximum. It will be shown that no two components approach a maximum together.

(i) $|\phi(M)| = n-1$

(a) e.g., $M = <10,10,10,10,10,10,10,10,10,10>$

An obvious compatible permutation is

$P = <1,2,3,4,5,6,7,8,9,10>$

$\phi(M) = \{1,2,3,4,5,6,7,8,9\}$, $\$(P) = 0$, $\lambda(M) = \{10\}$

$$\therefore \quad ¢(\sigma) \leq |\phi(M)| + \$(P) + |\lambda(M,P)|$$
$$\leq 10$$

(b) Increasing $\$(P)$ and/or $\lambda(M,P)$ decreases $\phi(M)$, and therefore does not increase $¢(\sigma)$.

e.g., $M = <3,3,10,10,10,10,10,10,10,10>$

Let $P = <3,2,1,4,5,6,7,8,9,10>$

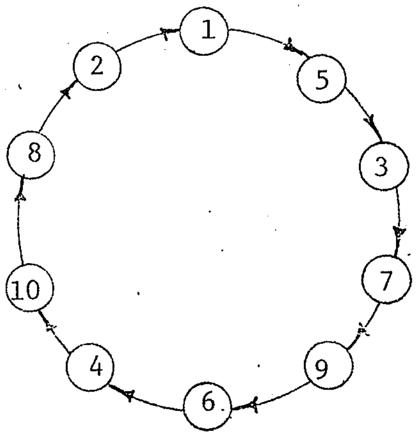
$\phi(M) = \{1,2,4,5,6,7,8,9\}$ $\$(P) = 1$, $\lambda(M,P) = \{10\}$

$$\therefore\ ¢(\sigma) \le |\phi(M)| + \$(P) + |\lambda(M,P)|$$
$$\le 8+1+1$$
$$\le 10$$

(ii) $\$(P) = n-1$

(a) Let $M = <5,1,7,10,8,4,9,2,6,8>$

$P = <5,1,7,10,3,4,9,2,6,8>$



$\$(P) = n-1=9$

$\phi(M) = \{3\}$

Since P contains just one cycle, $\lambda(M,P)$ is empty

$$¢(\sigma) \le |\phi(M)| + \$(P) + |\lambda(M,P)|$$
$$\le 1 + 9 + 0$$
$$\le 10$$

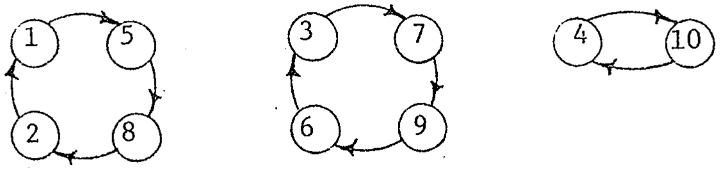
(b) Increase $\phi(M)$ and try to maintain $\$(P) = n-1$

Let $M = <5,1,7,10,8,10,9,2,6,8>$ and

$P = <5,1,7,10,3,4,9,2,6,8>$

Again $\$(P) = 9$ and now $\phi(M) = \{3,4\}$. However, P is not a minimal permutation compatible with map M since 3 and 4 $\in \phi(M)$ are in the same cycle.

A minimal permutation compatible with map M derivable from P, using AR1 and AR2, is $Q = <5,1,7,10,8,3,9,2,6,4>$



$\$(Q) = 7$, $\phi(M) = \{3,4\}$, and $\lambda(M,P)$ is empty.

$$\therefore\ ¢(\sigma) \le |\phi(M)| + \$(P) + |\lambda(M,P)|$$
$$\le 9$$

Instead of $¢(\sigma)$ increasing, it has decreased.

(iii) $|\lambda(M,P)| = n/2$

e.g., $M = <6,7,8,9,10,6,7,8,9,10>$

For $|\lambda(M,P)|$ to be maximum, $P = <1,2,3,4,5,6,7,8,9,10>$

Thus, $\$(P) = 0$, $\phi(M) = \{1,2,3,4,5\}$, $\lambda(M,P) = \{6,7,8,9,10\}$

$$\begin{aligned} ¢(\sigma) &\leq |\phi(M)| + \$(P) + |\lambda(M,P)| \\ &\leq 10 \end{aligned}$$

Again, the maximum length for a map realization is n.

Examining $¢(\sigma) \leq |\phi(M)| + \$(P) + |\lambda(M,P)|$ when one of the functions components is worst-case, it has been shown that $¢(\sigma) \leq n$. Trying to increase the other components and hold the worst-case component constant has not increased $¢(\sigma)$. Actually, $¢(\sigma)$ often decreases when this is attempted. From other examples we have examined, when all the components $¢(\sigma)$ contribute to $¢(\sigma)$, the cost $¢(\sigma)$ is usually less than n. Thus, it would appear that the worst-case for $¢(\sigma)$ occurs when one of the components of $¢(\sigma)$ is worst-case.

The arguments presented above, although not conclusive, do indicate that the maximum generator realization length for a map is n. If this is the case, then savings will again result when using the expanded generator set for large machines.

Chapter 8 Discussion and Conclusion

The results of this thesis are briefly summarized in this chapter. However, before this summary is presented, a couple of points about the usefulness of decomposition theory are made.

Decomposition theory is not universally applicable to all machines. For many machines, S.P. partitions do not exist and alternate synthesis techniques would have to be used. To alleviate this problem, Kohavi [41] developed the implication graph as a tool to induce structure on nonstructured machines, and to improve the structure of machines with S.P. partitions. The implication graph can thus be used to extend the application of structure theory. However, Kohavi used the implication graph with the implicit restriction that the number of state variables necessary to realize a machine was not to be increased. The usefulness of the implication graph is enhanced if this restriction is not rigidly observed.

The implication graph has only been used for completely specified machines. Applying the implication graph to incompletely specified machines is not as straightforward as would initially appear. In Chapter 5 we demonstrated that an S.P. partition for an incompletely specified machine made implicit assumptions about the unspecified next-states. Using an S.P. partition to realize a machine could destroy some of the structure of the machine. It would appear that using an implication graph for an incompletely specified machine could induce structure on the machine, but could at the same time nullify the existing structure. An examination of this inter-relationship would be necessary before implication graphs could be used for incompletely specified machines.

The concept of decomposition is relevant to the art of computer programming, where the ability to split a large program into subprograms simplifies the programming task. It would be of interest to examine whether algebraic structure theory was applicable to the decomposition of a program.

8.1 Decomposition Algorithm

Decomposition theory is available to the logic designer as one of many methods to synthesize sequential machines. As such, it should be possible for the designer to quickly determine whether any decompositions exist and if so, the nature of the decompositions. The "inspection" method of Hartmanis and Stearns for deriving decompositions is inadequate for large machines. Curtis' [8] method was examined in Chapter 2 and found to generate redundant decompositions and also to omit some of the possible nonredundant decompositions. These and other problems mentioned render the method inadequate.

A simple, efficient algorithm for obtaining the decompositions of a sequential machine was formulated in Chapter 3. The algorithm is based on extensions made to Hartmanis and Stearns' definition of a non-redundant set of partitions. A theoretical basis for the algorithm was developed which guaranteed that all the nonredundant decompositions, and only the nonredundant decompositions, were generated. In addition, properties of nonredundant sets of partitions were developed which simplify the application of the algorithm.

As the number of decompositions for a machine may be large, it is necessary that any method of generating decompositions be easily programmed. The numerous examples presented using the algorithm aptly demonstrate that the algorithm is well suited to implementation as a program. Chapter 4 presented a new representation of the S.P. partition lattice which could be used in a computer implementation of the algorithm.

8.2 Decomposition Evaluation

Two methods of evaluating decompositions were presented in Chapter 4. The first method eliminated pairs of partitions for which a more economical pair existed. However, for most machines there were still a large number of decompositions after the uneconomic pairs were deleted. It was demonstrated that the uneconomic pair concept could be extended to include the "subtle" redundancy indicated by Hartmanis and Stearns. Thus, it is possible to eliminate all but the most economical decompositions for a sequential machine, simplifying the choice a designer must make. This approach, however, was decided to be inappropriate. The reason being that this technique would severely restrict the possible realizations for a machine; realizations which are desirable to the logic designer may be eliminated as uneconomical.

The second evaluation techinque, which is based on the cost of a ROM realization remedies this problem. None of the decompositions for a machine are deleted. Instead, the ROM cost for each decomposition is calculated as the decomposition is derived. Thus the logic designer is able to examine all possible decompositions and at the same time obtain a measure of the relative merit of each.

The above evaluation technique, although based on ROMs, is device independent. This is because the ROM formula evaluates state and input variable dependence, which is constant regardless of the implementation hardware. We have also shown that ROM evaluation is able to indicate uneconomic pairs and subtle redundancy.

8.3 Incompletely Specified Machines and other Applications

In Chapter 5, decomposition theory was extended to incompletely specified machines using weak and extended substitution property partitions. The properties of both were identical, but weak substitution property partitions were found to be much easier to work with.

Problems that arose with determining the weak substitution property partitions were solved by the introduction of a new operator. The new operator was also necessary in order to determine the least upper bound of two weak substitution property partitions. As might be expected, the operator generalized to the completely specified case and could be used to find the S.P. partitions for a completely specified case.

For two S.P. partitions, whose sum π is not equal to I, a common submachine can be factored from both partitions and computed separately. Factoring a common submachine and realizing it separately is more economical than realizing it in each machine. For incompletely specified machines, where the sum of two weak substitution property partitions is not necessarily a weak substitution property partition, this is not always possible. As a compromise, a decomposition developed by Hartmanis [27], was adapted to handle this possibility.

This decomposition, which we have called a partial serial decomposition, is used by Hartmanis to obtain an economical serial decomposition of a machine using an S.P. partition π. However, we have shown that the same decomposition can be obtained using two S.P. partitions π_1 and π_2, where $\pi_1+\pi_2 = \pi \neq I$. The partial serial decomposition, when realized with π_1 and π_2, factors π from one of the partitions and realizes it in the other. Thus the partial serial decomposition is midway between a parallel decomposition, in which a common submachine is not factored from either machine, and the series-parallel decomposition, in which the common submachine is factored from both machines. A ROM evaluation of the three decompositions, also shows that the partial serial decompositions is midway, economically, between the other two partitions. The partial serial decomposition, although applicable to both completely and incompletely specified machines, is more relevent to the incompletely specified case.

Two other aspects of machine decomposition were briefly examined in Chapter 5. A theoretical basis for the work of Kohavi and Smith [44] and Smith and Kohavi [62] on the synthesis of multiple machines was developed. The theory presented proved that the structure of the composite machine was dependent of the structure of the component machines.

In order for a state assignment for an asynchronous sequential machine to be critical-race free, certain conditions, Tracey [67], must be satisfied. Properties relating S.P. partitions to the partial dichotomies of an asynchronous machine were developed. For asynchronous machines, these properties simplify the determination of single transition time assignments using S.P. partitions.

8.4 Machine Synthesis with Complex, Functional Components

The availability of MSI and LSI integrated circuits has dramatically affected sequential circuit design. No longer is it necessary to derive a minimal realization in terms of logic gates and memory elements. Instead inexpensive chips, which realize complex functions, can be used to implement sequential machines directly. However, as we have indicated, algebraic structure theory is still relevant. This is because it enables the designer to split a large synthesis problem into several smaller ones, thus simplifying the implementation problem.

In this thesis we have examined two possible ways of utilizing the new integrated circuit technologies for machine synthesis, sequential cellular arrays and generator sequences.

Hu [33] has used the transition matrix concept to develop a cellular array which can be used for machine realization. In Chapter 6, minor improvements to Hu's cellular synthesis method have been suggested. In addition, a new synthesis method for Hu's array has been presented. The new method derives a realization which requires fewer, or at worst the same number of, arrays than a minimal realization produced by Hu's method.

Problems with using Hu's array as a component for the realization of large machines have been presented and modifications suggested. However, as the number of inputs for a machine increases the number of input lines per array also increases. This further restricts the usefulness of Hu's array as a component in realizing large machines. Consequently, a new array, based on the $x\bar{x}$ transition matrix has been proposed. For this array the number of input lines per array does not increase as the number of inputs for a machine increases. Thus, any machine can be realized by a standard array, or a collection of standard arrays. A synthesis method was presented for the new array.

An alternative method for using complex, functional components for machine synthesis was developed by Haring [23] and Menger [53]. Haring has shown that a set of 3 basic generators can be used to realize any column of a sequential machine, and thus any machine. The problem with using only 3 generators is that the number of generators required to realize a column increases rapidly as the number of states increases.

Menger introduced a set of 2(n-1) generators, for an n state machine, in order to reduce the number of generators required to realize a column. Menger's synthesis methods generate realizations such that the number of generators used varies linearly with the number of states. In Chapter 7, Menger's algorithm was examined extensively and an additional irredundancy condition was introduced.

Following Menger's lead we have presented a new generator set, which is a logical extension of Menger's generators. Problems associated with the synthesis of a machine using the new generator set have been examined. A simple algorithm for determining a realization for a permutation column was presented. The benefits of using the new generators for permutation columns was easy to calculate. With Menger's generator set a realization for a permutation map requires $\frac{3n}{2}$ generators, while a realization using our generator set requires n-1 generators. This saving is significant when a large machine is being realized.

Deriving a realization algorithm for non-permutation columns is not as straightforward. The difficulty arises because of the introduction of non-permutation operators which reset states from more than one position in the map. These operators, while providing greater freedom in realizing operator sequences, also allow greater variability in the possible sequences. Subsequently, it has only been possible to develop a heuristic method for generating operator sequences for non-permutation maps.

The problems associated with finding operator sequences with the proposed operator set are helpful in indicating the types of problems that could occur with any expanded operator set.

References

(1) T. Ae and N. Yoshida, "A Method of Synthesis and Decomposition of Autonomous Linear Sequential Circuits", Information and Control, vol.23, no.4, November 1973, pp. 382-392.

(2) A. E. Almaine and M. E. Woodward,"Computer Program for S.P. Partitions of Sequential Machines", Electronic Letters, vol.10, no.21, October 17, 1974, pp. 445-446.

(3) D. B. Armstrong, "A Programmed Algorithm for Assigning Internal Codes to Sequential Machines", IRE Trans. Elec. Comp., vol.EC-11, no.4, August 1962, pp. 466-472.

(4) D. B. Armstrong, "On the Efficient Assignment of Internal Codes to Sequential Machines", IRE Trans. Elec. Comp. vol.EC-11, no.5, October 1962, pp. 611-622.

(5) Thomas F. Arnold, C.-J. Tan, and M. Newborn, "Iteratively Realized Sequential Circuits", IEEE Trans. Elec. Comp., vol.C-19, no.1, January 1970, pp. 54-66.

(6) G. C. Bacon, "The Decomposition of Stochastic Automata", Information and Control, vol.7, September 1964, pp. 320-339.

(7) T. L. Booth, "Sequential Machines and Automata Theory", John Wiley and Sons, Inc., New York, 1964.

(8) H. A. Curtis, "Multiple Reduction of Variable Dependency of Sequential Machines", Jour. of the Assoc. for Comput. Mach., vol.9, no.3, July 1962, pp. 324-344.

(9) H. A. Curtis, "Use of Decomposition Theory in the Solution of the State Assignment Problem of Sequential Machines", Jour.of the Assoc. for Comput. Mach., vol.10, no.3, July 1963, pp. 386-412.

(10) H. A. Curtis, "Tan-like State Assignments for Synchronous Sequential Machines", IEEE Trans. Elec. Comp., vol. C-22,no.2, February 1973, pp. 181-187.

(11) P. Das, "Fault-Detection Experiments for Parallel-Decomposable Sequential Machines", IEEE Trans. Elec. Comp., vol.C-24, no.8, November 1975, pp. 1104-1108.

(12) S. C. De Sarkar, S. Bandyopadhyay, and A. K. Choudhury, "Unate Cascade Realizations of Synchronous Sequential Machines", IEEE Trans. Elec. Comp. vol.C-23, no.10, October 1974, pp. 1008-1019.

(13) T. A. Dolotta and E. J. McCluskey, "The Coding of Internal States of Sequential Machines", IEEE Trans. Elec. Comp. vol.EC-13, no.5, October 1964, pp. 549-562.

(14) E. H. Farr, "Lattice Properties of Sequential Machines", Jour.of the Assoc. for Comp. Mach., vol.10, no.3, July 1963, pp. 365-385.

(15) D. Ferrarri and A. Grasselli, "A Cellular Structure for Sequential Networks", IEEE Trans. Elec. Comp., vol.C-18, no.10, October 1969, pp. 947-953.

(16) A. D. Friedman, "Feedback in Synchronous Sequential Switching Networks", IEEE Trans. Elec. Comp., vol.EC-15, no.3, June 1966, pp. 354-367.

(17) G. B. Gerace and G. Gestri, "Decomposition of Synchronous of Synchronous Sequential Machines into Synchronous and Asynchronous Submachines", Information and Control, vol.11, 1967, pp. 568-591.

(18) G. B. Gerace and G. Gestri, "Decomposition of a Synchronous Machine into an Asynchronous Submachine Driving a Synchronous One", Information and Control, vol.12, 1968, pp.538-548.

(19) G. Gestri, "Synthesis of Multiple Sequential Machines Having Different Inputs", IEEE Trans. Elec. Comp., vol.C-19, no.11, November 1970, pp. 1105-1108.

(20) G. Gestri, "Serial Realization of the Next State and Output Functions of a Sequential Machine", Information and Control, vol.21, no.5, December 1972, pp. 466-475.

(21) A. Gill and J. R. Flexer, "Periodic Decomposition of Sequential Machines", Jour. Assoc. Comp. Mach., vol.14, no.4, October 1967, pp. 666-676.

(22) J. W. Grzymala-Busse, "On the Decomposition of Periodic Representations of Sequential Machines", IEEE Trans. Elec. Comp., Vol.C-20, no.8, August 1971, pp. 929-933.

(23) D. R. Haring, "Sequential-Circuit Synthesis: State Assignment Aspects", M.I.T. Press, Cambridge, Mass., 1966.

(24) J. Hartmanis, "Symbolic Analysis of a Decomposition of Information Processing Machines", Information and Control, vol.3, no.2, June 1960, pp. 154-178.

(25) J. Hartmanis, "On the State Assignment Problem for Sequential Machines I", IRE Trans. Elec. Comp., vol.EC-10, no.2, June 1961, pp. 157-165.

(26) J. Hartmanis, "Maximal Autonomous Clocks of Sequential Machines", IRE Trans. Elec. Comp., vol.EC-11, no.1, February 1962, pp.83-86.

(27) J. Hartmanis, "Loop-Free Structure of Sequential Machines", Information and Control, vol.5, no.1, March 1962, pp. 25-43.

(28) J. Hartmanis, "Further Results on the Structure of Sequential Machines", Jour. Assoc. Comp. Mach., vol.10, no.1, January 1963, pp. 78-88.

(29) J. Hartmanis and R. E. Stearns, "Some Dangers in State Reduction of Sequential Machines", Information and Control, vol.5, no.3, September 1962, pp. 252-260.

(30) J. Hartmanis and R. E. Stearns, "Pair Algebras and their Application to Automata Theory", Information and Control, vol.7, no.4, December 1964, pp. 485-507.

(31) J. Hartmanis and R. E. Stearns, "Algebraic Structure Theory of Sequential Machines", Prentice-Hall, Inc., Englewood Cliffs, N.J., 1966.

(32) B. V. Howard, "Partition Methods for r.o.m. Sequential Machines", Electronic Letters, 8, 1972, pp. 334-336.

(33) S. C. Hu, "Cellular Synthesis of Synchronous Sequential Machines", IEEE Trans. Elec. Comp., vol, C-21, no.12, December 1972, pp. 1399-1405.

(34) J.-C. Huang, "Cellular-Array Realization of Finite-State Sequential Machines", Moore School Rept. 69-23 to U. S. Army Research Office (Durham), D.D.C., AD 688 623, 1969.

(35) J.-C. Huang, "A Universal Cellular Array", IEEE Trans. Elec. Comp. vol.C-20, no.3, March 1971, pp. 317-320.

(36) D. A. Huffman, "The Synthesis of Sequential Switching Networks", Jour. of the Franklin Institute, vol. 257, no.3, March 1954, pp. 161-190, and no.4, April 1954, pp. 275-303.

(37) K. Hwang, "Periodic Realization of Synchronous Sequential Machines", IEEE Trans. Elec. Comp. vol.C-22. no.10, October 1973, pp.923-927.

(38) R. M. Karp, "Some Techniques of State Assignment for Synchronous Sequential Machines", IEEE Trans. Elec. Comp. vol.EC-13, no.5, October 1964, pp. 507-518.

(39) L. I. Kinney, "Decomposition of Asynchronous Sequential Switching Circuits", IEEE Trans. Elec. Comp., vol.C-19, no.6, June 1970, pp. 515-529.

(40) R. Knispel, "Read only Memory Evaluation of Sequential Machine Decomposition", Proc. Third Manitoba Conf. of Num. Math., October 1973, pp. 253-270.

(41) Z. Kohavi, "Secondary State Assignment for Sequential Machines", IEEE Trans. Elec. Comp., vol.EC-13, no.3, June 1964, pp.193-203.

(42) Z. Kohavi, "Reduction of Output Dependency in Sequential Machines", IEEE Trans. Elec. Comp. vol. EC-12, December 1965, pp. 932-934.

(43) Z. Kohavi, "Switching and Finite Automata Theory", McGraw Hill, 1970.

(44) Z. Kohavi and E. J. Smith, "Decomposition of Sequential Machines", Proc. Sixth Ann. Symp. on Switching Theory and Logic Design", Ann Arbor, Michigan, October 1965, pp. 52-61.

(45) R. Krishnan and E. J. Smith, "A Cellular Realization for Sequential Machines", an unpublished paper.

(46) S. N. Kukreja and I-Ngo Chan, "Combinational and Sequential Cellular Structures", IEEE Trans. Elec. Comp., vol.C-22, no.9, September 1973, pp. 813-823.

(47) O. P. Kuznetsov, "The Parallel Decomposition of Automata with Separation of Inputs", Automat. i Telemekh. (U.S.S.R.), no.3, 1969, pp. 104-109, English translation Automat. Remote Control.

(48) F. Kvamme, "Standard Read-Only-Memories Simplify Complex Logic Design, Electronics, Jan. 15, 1970, pp. 88-95.

(49) D. W. Lewin, "Logical Design of Switching Circuits", Nelson, London, 1968.

(50) D. W. Lewin, "Outstanding Problems in Logic Design", The Radio and Elec. Eng., vol.44, no.1, January 1974, pp. 9-17.

(51) P. J. Marino, "A Linear Decomposition for Sequential Machines", IEEE Trans. Elec. Comp., vol.C-19, No.10, October 1970, pp.956-963.

(52) G. H. Mealy, "A Method for Synthesizing Sequential Circuits", Bell Systems Tech. Jour., vol.34, no.5, September 1955, pp. 1045-1079.

(53) K. S. Menger, "Synthesis of Sequential Machines by Iterated Combinational Logic Blocks" in Synthesis of Sequential Switching Networks, Electronic Systems Lab., M.I.T., AD-608881, October 1964, Chapter 3.

(54) P. R. Menon, "On Sequential Machine Decompositions for Reducing the Number of Delay Elements", Information and Control, vol.15, 1969, pp. 274-287.

(55) R. C. Minnick, "A Survey of Microcellular Research", Jour. Assoc. Comp. Mach., vol.14, no.2, April 1967, pp. 203-241.

(56) E. F. Moore, "Gedanken-Experiments on Sequential Machines", Automata Studies, 34, Princeton, N.J., Princeton University Press, 1956.

(57) E. F. Moore, ed., "Sequential Machines: Selected Papers", Reading, Mass., Addison-Wesley, 1964.

(58) M. Newborn, "A Synthesis Technique for binary input-binary output Synchronous Sequential Moore Machines", IEEE Trans. Elec. Comp., vol.C-17, July 1968, pp. 697-699.

(59) M. M. Newborn and T. F. Arnold, "Universal Modules for Bounded Signal Fan-Out Synchronous Sequential Circuits", IEEE Trans. Elec. Comp., vol.C-21, no.1, January 1972, pp. 63-73.

(60) W. J. Sacco, A Computer Technique Useful for Some Problems in the Partitioning Theory of Sequential Machines", Ballistic Research Lab. Memorandum Report No. 1733, March 1966.

(61) Gabriėle Saucier, "Next-State Equations of Asynchronous Sequential Machines", IEEE Trans. Elec. Comp., vol.C-21, no.4, April 1972, pp. 397-399.

(62) E. J. Smith and Z. Kohavi, "Synthesis of Multiple Sequential Machines", Proc. Seventh Ann. Symp. on Switching Theory and Logic Design, Berkley, California, October 1966, pp. 160-171.

(63) R. E. Stearns and J. Hartmanis, "On the State Assignment Problem for Sequential Machines II", IRE Trans. Elec. Comp., vol.EC-10, no.4, December 1961, pp. 593-603.

(64) J. R. Storey, H. J. Harrison, and E. A. Reinhard, "Optimum State Assignment for Synchronous Sequential Circuits", IEEE Trans. Elec. Comp., vol.C-21, no.12, December 1972, pp. 1365-1373.

(65) C.-J. Tan, P. K. Menon, and A. D. Friedman, "Structural Simplification and Decomposition of Asynchronous Sequential Circuits", IEEE Trans. Elec. Comp. vol. C-18, no.9, September 1969, pp. 830-838.

(66) H. C. Torng, "An Algorithm for Finding Secondary Assignments of Synchronous Sequential Machines", IEEE Trans. Elec. Comp., vol.C-17, May 1968, pp. 461-469.

(67) J. H. Tracey, "Internal State Assignment for Asynchronous Sequential Machines", IEEE Trans. Elec. Comp., vol.EC-15,no.4, August 1966, pp. 551-560.

(68) G. L. Tumbush and J. E. Brandeberry, "A State Assignment Technique for Sequential Machines using J-K Flip Flops", IEEE Trans. Elec. Comp., vol.C-23, no.1, January 1974, pp. 85-86.

(69) J. D. Ullman and P. Weiner, "Uniform Synthesis of Sequential Circuits", Bell Systems Tech. Jour., May-June 1969, pp. 1115-1127.

(70) P. Weiner and E. J. Smith, "Optimization of Reduced Dependencies for Synchronous Sequential Machines", IEEE Trans.Elec. Comp., vol.EC-16, no.6, December 1967, pp. 835-847.

(71) C.-C. Yang, "Generation of all Closed Partitions of the State Set of a Sequential Machine", IEEE Trans. Elec. Comp., vol.C-23, no.11, May 1974, pp. 530-533.

(72) M. Yoeli, "The Cascade Decomposition of Sequential Machines", IRE Trans. Elec. Comp., vol.EC-10, No.4, December 1961, pp. 587-592.

(73) M. Yoeli, "Cascade-Parallel Decomposition of Sequential Machines", IRE Trans. Elec. Comp., vol.EC-12, no.3, June 1963, pp. 322-324.

(74) T. A. Zahle, "On Coding the States of Sequential Machines with the Use of Partition Pair", IEEE Trans. Elec. Comp., vol.EC-15, no.2, April 1966, pp. 249-253.

(75) T. R. Blakeslee, "Digital Design with Standard MSI and LSI", John Wiley and Sons, Toronto, 1975.

(76) U. Piel and P. Holland, "Application of a High Speed Programmable Logic Array", Computer Design, December 1973, pp. 94-96.

(77) H. Fleisher and L. I. Maissel, "An Introduction to Array Logic", IBM Journal of Research and Development, Vol. 19, No. 2, March 1975, pp. 98-109.