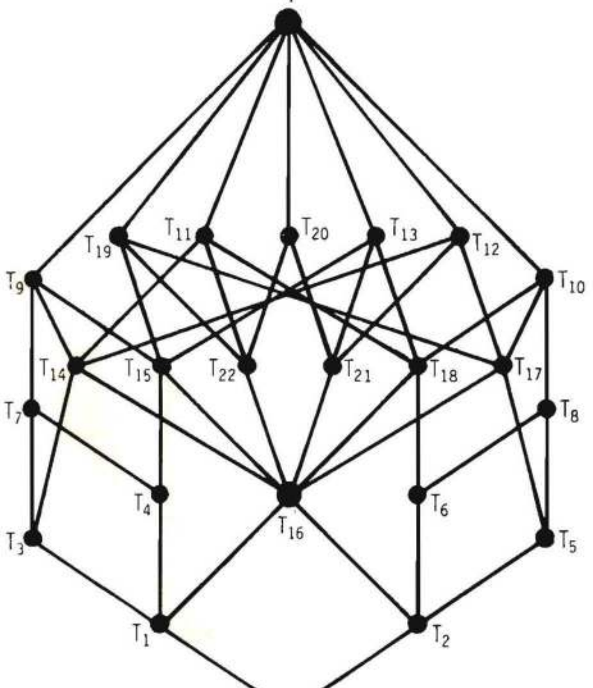
Trinity algebra and full-decompositions of sequential machines

TRINITY ALGEBRA AND FULL-DECOMPOSITIONS OF SEQUENTIAL MACHINES



HOU Yibin

TRINITY ALGEBRA AND FULL-DECOMPOSITIONS OF SEQUENTIAL MACHINES

PROEFSCHRIFT

TER VERKRIJGING VAN DE GRAAD VAN DOCTOR IN DE
TECHNISCHE WETENSCHAPPEN AAN DE TECHNISCHE
HOGESCHOOL EINDHOVEN, OP GEZAG VAN DE RECTOR
MAGNIFICUS, PROF.DR. F.N. HOOGE,VOOR
EEN COMMISSIE AANGEWEZEN DOOR HET COLLEGE
VAN DEKANEN IN HET OPENBAAR TE VERDEDIGEN OP
VRIJDAG 30 MEI 1986 TE 14.00 UUR

DOOR

HOU Yibin

GEBOREN TE SHAANXI,CHINA

DIT PROEFSCHRIFT IS GOEDGEKEURD
DOOR DE PROMOTOREN

prof.ir. A. Heetman
en
prof.dr. J.H. van Lint

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Hou Yibin

Trinity algebra and full-decompositions of sequential
machines / Hou Yibin. - [S.l. : s.n.] . - Fig., tab.-
Proefschrift Eindhoven. - Met. lit. opg., reg.
ISBN 90-9001285-0
SISO 664.2 UDC 681.325.65:519.6 UGI 650
Trefw.: automatentheorie.

献 给 祖 国

Voor mijn vaderland

To my motherland

Acknowledgements

The investigations described in this thesis were performed at the Eindhoven University of Technology, in The Netherlands.

I am greatly indebted to the University and the Faculty of Electrical Engineering for offering me the opportunity to study and to publish the results of these investigations in their present form. This reflects a friendship and close cooperation between the University of Eindhoven and the Xian Jiaotong University, in China. I appreciate the contribution of the chairman of Friendship Association of Holland and China, Prof. dr.ir. P. Eykhoff, for the cooperation between the two Universities and for his kindnees to me and the other Chinese scholars and students in Eindhoven.

I wish to express my gratitude to all those who have contributed to this work in any way.

I indebted to my promotor Prof. ir. A. Heetman and co-promotor Prof. dr.ir. S.H. v. Lint for their interest in my work and for making it possible for me to present this work as a thesis.

I am grateful to Mr. C.P.J. Schnabel, Mr. M.J.M. van Weert and Mr. P.M.C.M. van den Eijnden for their discussions and suggestions in the early stages of my work. I would like to thank Mr. A.G.M. Geurts for his advice with respect to my program writing and for his help in the other aspects.

I would like to thank Dr. P.R. Attwood for reading the manuscript of this thesis and making corrections to the English.

Mr. C.P.G. v. d. Watering made an excellent job of typing the manuscript and my reports and I would like record my gratitude here.

I wish to thank Mr. I.V. Bruza who enthusiastically helped me to find books, papers and to verify the reference list.

Gratitude is also expressed to all members of the EB group who gave their support to this work in any way.

I repeat my appreciation of the help from Prof.ir. A. Heetman and the International Neighbour Group in Eindhoven for making the stay in Holland of my wife and myself very pleasant.

Finally, I wish to thank my wife, Xiuzhen, for her consistent encougragement and assistance with this thesis.

Eindhoven, May 1986

CONTENTS:

CHAPTER 1

INTRODUCTION

In the past decade, digital (circuit and system) design has undergone dramatic changes. Today, digital designers rarely build any components or devices that are available in integrated circuit forms. This is because digital integrated circuits are not only convenient and easy to use but also cost less. One type of integrated circuits, which has become very popular in digital design in recent years, is the array logic. Array logic is defined as the use of memory-like structures for performing combinational logic and sequential logic. Corresponding to the combinational logic the integrated circuit is called a programmable logic array (PLA), when corresponding to the sequential logic, it is called a programmable logic sequencer (PLS). A PLA comprises both an AND array and an OR array, normally. If we put some clocked output flip-flops and appropriate feedback in a PLA then a PLS is built. The PLS is a fully implemented Mealy machine on a chip [17]. Theoretically speaking, any logic design can be implemented by a logic array if we neglect the practical size of the integrated circuit. However, unfortunately, as we know, an integrated circuit chip is limited not only with the size of the circuit but also especially with the pins of integrated circuits, while the number of pins is related to the numbers of inputs and outputs of the logical system to be implemented. To implement a practical logical system by the integrated circuits available, such as PLA and PLS, leads to a practical problem - how to decompose a large logic system into several smaller logic systems - each can be implemented by today's array logic integrated circuit.

Owing to the fact that there exist two abstract mathematical models for logic circuits (one is switching algebra for combinational circuits, and the other is a sequential machine for sequential circuits) the research on this problem centers on a theoretical problem - how to decompose a larger Boolean function into smaller Boolean functions - each can be implemented by a PLA, or how to decompose a larger sequential machine into the interconnection of some smaller sequential machines - each can be implemented by a PLS. This theory is referred to the decomposition theory.

The decomposition theory for Boolean functions has been well-developed in much literature, such as [1,2,18,25,28]. The theory and methods have been applied to the PLA implementation of Boolean functions [26,27]. Hence, the theoretical problem for PLA implementation has been largely solved due to the simplicity of Boolean functions.

Historically, a decomposition theory for sequential machines means an organized body of techniques and results dealing with the problems of how sequential machines can be realized from sets of smaller component machines, how these component machines have to be interconnected, and how "information" flows in and between these machines when they are in operation. The research on the theory was started in the early 1960's. For the technologies during that period, the relevant problems were primarily concerned with component reduction. In sequential circuits, a component reduction is mainly associated with reducing the set of states of the sequential machines in question. Therefore, a "smaller", or "simpler", component machine was defined as a component machine with fewer states than the original machine [12,15]. The definition has been applied and has served as a standard for a decomposition whether it is trivial or not by most of the literature and books about the decomposition of sequential machines [9,16]. With the development of integrated circuit technology and the advent of large scale integration (LSI) and very large scale integration (VLSI) in digital systems design, the problems concerned with fewer components have become less relevant [8]. Consequently, in the view of PLS implementation of sequential machines, the definition does not meet the requirements for sequential circuit design using today's PLS packages. A "smaller" component machine must require fewer pins of PLS package than the original machine in order to implement it. In other words, this means that a smaller component machine must have fewer states, inputs and outputs than the machine to be decomposed. It will be apparent that, when we consider this kind of decomposition, we have to deal not only with the number of states but also with the number of inputs and outputs too. We refer to the decomposition as a full-decomposition. We should develop the decomposition theory or look for some new way for this purpose. This thesis arose from this need. The work discussed in this thesis is one approach to the subject. In it we shall propose a method for decomposing a sequential machine into interconnection of component machines, if they exist, each of them has less states, less inputs and less outputs. The method is primarily based on the concepts of partition trinity and forced-trinity which will be discussed later.

The problem of PLS implementation of a sequential machine serves as a wedge to the full-decomposition theory. In this thesis we are mainly concerned with the problem only at the abstract algebra level. The study and results are significant, not only in the sense of developing decomposition theory, but also in any other area of applying machine theory with similar requirements.

This thesis contains nine chapters. A brief description of each chapter follows:-

Chapter 1 describes and expands the full-decomposition problem.

Some general concepts on machines are described in Chapter 2. We discuss the different types of decompositions and make a classification of them by introducing a universal connection model.

Chapter 3 describes the partition trinity, trinity algebra and its properties. It provides the mathematical foundation of full-decomposition theory.

In Chapters 4 and 5 we apply the concepts of partition trinity and forced trinity to parallel full-decomposition and serial full-decomposition of sequential machines. A H-decomposition is defined and presented in Chapter 6. It resembles a parallel full-decomposition and is a supplement to the full-decomposition theory. A wreath decomposition is also discussed in this chapter by partition trinities.

Chapter 7 extends the theory from completely specified machines to incompletely specified machines. It is shown that most of the results can be used for incompletely specified machines.

In Chapter 8 we discuss how to use computers for machine decompositions. Many algorithms for them are presented.

The final chapter is devoted to a discussion of further topics which are worthwhile studying for the development of the full-decomposition theory of machines.

/ / /

CHAPTER 2

MACHINES AND THEIR DECOMPOSITIONS

In this chapter, we are going to discuss the general concepts on basic models for sequential machines and on types of decompositions of them. Three basic models of machines are defined in section 2.1. Section 2.2 gives some notations and machine functions which makes it easier to discuss and deal with the topics in this thesis. In section 2.3, a brief introduction to the decomposition theory of sequential machines is given. In the last section a universal connection of two machines is presented and many decompositions derived from it are defined and analysed with the main techniques which are available or are developed in this thesis.

2.1 Machines

In practice, many complex processes, not only in the area of computer systems and their associated languages and software, but also in the areas of biology, psychology, biochemistry etc., can be regarded as behaving rather like machines. Any given system or design problem can be described by a sequential machine as defined below. The terms sequential machine, finite-state machine, finite automaton, and simply machine are synonyms. In essence, sequential machines are mathematical models which describe sequential systems, such as sequential circuits. Since a sequential machine is merely an abstract model, it may be used to decribe the operational behaviour of systems other than sequential circuits. Indeed, the term "machine" used here does not imply that a sequential machine has to be real physical machine or machine-like object. On the contrary, it does not even have to be tangible; any physical or abstract phenomenon may be called a sequential machine as long as it satisfies the axioms of this model.

2.1.1 Basic Models of Machines

The theory of machines is concerned with mathematical models for discrete, deterministic information-processing devices and systems, such as digital computers, digital control units, electronic circuits with synchronized delay elements, and so on. All these devices and systems have the following common properties, which are abstracted in the definition of a sequential machine.

DEFINITION 2.1

A sequential machine or *Mealy machine* is a system which can be characterized by a quintuple,

$$M = (I, S, O, \delta, \lambda)$$

where I is a finite nonempty set of input symbols,
S is a finite nonempty set of internal states,
O is a finite set of output symbols,
δ is a *next-state function*, which maps $S \times I$ to S,
λ is an *output function*, which maps $S \times I$ to O.
(End of Definition 2.1)

We refer to the next-state function and output function as *machine functions* throughout this thesis.

A machine may be presented in the form of a table or a diagram. The table and the diagram in question are called the *transition table* and the *transition diagram* of the machine, respectively. The table, or the diagram, is defined by the next-state function and output function. In this thesis, mainly, the form of the table will be used.

From the definition of machines, if for any pair of inputs, x_i and x_j, in I, the output function satisfies, for all s in S, there will exist an output value, say $y \in O$, such that

$$\lambda(s,x_i) = \lambda(s,x_j) = y$$

then, the mapping λ becomes independent of inputs, i.e.,

$$\lambda : S \to O.$$

In this case, the machine is called a *Moore machine* and is defined by:

DEFINITION 2.2

A sequential machine is said to be of the Moore type (*Moore machine*) if its output function is function of its states only:

$\lambda : S \to O.$

(End of Definition 2.2)

Therefore, a Moore machine is a special case of Mealy machines. It can be converted into Mealy machine and vice versa. A *state-dependent machine* is an alternate name for Moore machine, in some books. In this thesis, we are mainly concerned with Mealy machines.

In some situations we are only interested in the internal states and not in the outputs of a system. This leads to a machine without outputs, which is a special case of the Mealy machines when the output function is a null relation or the output set is an empty set. These machines are called *state machines* and a precise definition is given as follows.

DEFINITION 2.3

A state machine is a triple :-

$M = (I, S, \delta)$

where: I and S are input set and state set, respectively and δ is a transition function.

(End of Definition 2.3)

In some books, a state machine is also referred to as a *semi automaton*.

In the definitions given above, the next-state function was a mapping from $S \times I \to S$, which means, for any $s \in S$ and $x \in I$, $\delta(s,x) \in S$. This kind of machine is called *deterministic* machines. In contrast to this, there is another function which maps $S \times I$ to some subset of S, that is, $\delta(s,x) \subseteq S$. This kind of machine is said to be *nondeterministic*. In this thesis, we are concerned only with deterministic machines.

Broadly speaking, the relation δ: $S \times I \to S$ or λ: $S \times I \to O$ may be a partial function, which implies that, for some $s \in S$ and $x \in I$, $\delta(s,x)$ is probably not specified. The machines with undefined next-states or outputs are referred to as *incompletely specified* machines, while the machines without undefined next-states and outputs are referred to as *completely specified* machines. In most of the chapters of this thesis, the discussions relate to completely specified machines.

Machine theory is the study of abstract computing devices, their organization, their structure and computational power. In the thesis we are mainly concerned with the structural aspect of it, which is referred to as *algebraic structure theory of machines*. In particular, by the theory, we learn how a quite large machine can be partitioned into a set of smaller component machines, each of which can be realized by the currently available LSI and VLSI circuits, also how these component machines have to be interconnected.

In this thesis, a rather informal notation for logical deductions in the proofs of propositions and theorems is used, as explained here. Let P, Q be two statements. Then the notation :-

$$\begin{array}{ll} & P \\ \Rightarrow & Q \quad \{R\} \end{array}$$

means that P implies Q under the reason R.
Similarly we have :-

$$\begin{array}{ll} & P \\ \Leftrightarrow & Q \quad \{R\}. \end{array}$$

A statement may be of the form :-

$$D : E$$

where D is a domain and E is a predicate or a logical statement expression, stating that E holds in D. When more than one variable exists in D, each domain is separated by a space. In some cases, domain D may be omitted if D is clear from the context.

An expression may include not only the logical conjunctions $\wedge$ or $\vee$, but also those on sets such as $\subseteq$, $\in$. For example, "$B \subseteq B' \in A \wedge C \subseteq C' \in A$" means that "both that B is a subset of some B′ in A and that C is a subset of C′ in A" are true.

The hint {R} sometimes may be in a form {calculus} which indicates that an appeal to everyday mathematics, like arithmetic or predicate calculus, is meant.

2.2 Machine functions

By the definition of machines, generally speaking, we shall present the machine $M = (I, S, O, \delta, \lambda)$ with an input symbol $x \in I$ while it is in some state, say $s \in S$. The machine then outputs $\lambda(s,x)$ while it moves to state $\delta(s,x)$. This notion is somewhat cumbersome and we shall introduce the idea of mappings (or functions) induced by the input.

From the viewpoint of inputs, the machine functions, δ and λ, can be considered as sets of functions induced by all inputs :-

$$\delta = \{\delta_x \mid \delta_x : S \to S \text{ and } x \in I\}$$

and $$\lambda = \{\lambda_x \mid \lambda_x : S \to S \text{ and } x \in I\}$$

where $\delta_x : S \to S$ is defined by

$$\forall s \in S \quad \forall x \in I \; : \; \delta_x(s) = \delta(s,x)$$
$$\lambda_x(s) = \lambda(s,x).$$

The δ_x and λ_x are called the *next-state function* and *output function*, respectively, *with respect to input* x. For the sake of convenience of operations on the machine functions with respect to different inputs, we write :-

$\delta_x(s)$ as $s\delta_x$ and $\lambda_x(s)$ as $s\lambda_x$.

Finally, we make

<u>Notation 2.1</u>

$$s\delta_x = \delta_x(s) = \delta(s,x)$$
$$s\lambda_x = \lambda_x(s) = \lambda(s,x)$$

for all $s \in S$ and $x \in I$.

(End of Notation 2.1)

From the notation introduced above, we have the following convenient rules for the operations on different input sequences.

<u>Property 2.1</u>

Let $x, y \in I$. Then, for any $s \in S$

$$s\delta_{xy} = (s\delta_x)\delta_y = s\delta_x\delta_y;$$
$$s\lambda_{xy} = (s\delta_x)\lambda_y = s\delta_x\lambda_y;$$

Proof.

$$\begin{aligned} s\delta_{xy} &= \delta(s,xy) & s\lambda_{xy} &= \lambda(s,xy) \\ &= \delta(\delta(s,x),y) & &= \lambda(\delta(s,x),y) \\ &= (s\delta_x)\delta_y & &= (s\delta_x)\lambda_y \\ &= s\delta_x\delta_y & &= s\delta_x\lambda_y \end{aligned}$$

(End of Property 2.1)

It shows the convenience that the notation gives namely natural operational order from left to right.

Property 2.2

Let I^* denote the set of all finite-length sequences of elements of I.

Then, for $x = x_1x_2...x_k$ in I^*, $x_i \varepsilon I$, $1 \le i \le k$,

$$s\delta_x = s\delta_{x_1x_2...x_k} = s\delta_{x_1}\delta_{x_2}...\delta_{x_k}$$

$$s\lambda_x = s\lambda_{x_1x_2...x_k} = (s\delta_{x_1}...\delta_{x_{k-1}})\lambda_{x_k}$$

Proof. Repeatedly apply Property 2.1.

(End of Property 2.2)

So, δ_x and λ_x are functions with respect to an input word x in I^* :

$$\delta_x : S \to S,$$
$$\lambda_x : S \to O.$$

Property 2.3

If $x = \varepsilon \varepsilon I^*$, then for all $s \varepsilon S$,

$$s\delta_x = s \quad \text{and} \quad s\lambda_x = \varepsilon \varepsilon O^*$$

where ε is a null word.

Proof. $\delta(s,\varepsilon) = s$ and $\lambda(s,\varepsilon) = \varepsilon$.

(End of Property 2.3)

Let A be a set. The power set of A is defined as set $\{a \mid a \subseteq A\}$ and is denoted by 2^A because it has an interesting property: $|2^A| = 2^{|A|}$. Therefore, in other words, 2^A is the set of all subsets of A. Let S and O be sets of states and outputs of a machine. For power sets 2^S and 2^O, we have the following functions defined in Notation 2.2.

<u>Notation 2.2</u>

Two partial functions,

$$\overline{\delta}_X : 2^S \to 2^S \quad \text{and} \quad \overline{\lambda}_X : 2^S \to 2^O$$

are defined by $Q\overline{\delta}_X = \{q\delta_X \mid q \in Q \subseteq S\}$

$Q\overline{\lambda}_X = \{q\lambda_X \mid q \in Q \subseteq S\}$

where $x \in I$.

If $x \subseteq I$, then $\overline{\delta}_X$ and $\overline{\lambda}_X$ are defined by

$$Q\overline{\delta}_X = \{q\overline{\delta}_{X_i} \mid q \in Q \wedge x_i \in x\}$$
$$Q\overline{\delta}_X = \{q\overline{\delta}_{X_i} \mid q \in Q \wedge x_i \in x\}$$

(End of Notation 2.2)

By the definition the following results are apparent.

<u>Property 2.4</u>

Let $Q_1, Q_2 \subseteq S$ and $x_1, x_2 \subseteq I$.

i) $Q_1 \subseteq Q_2 \Rightarrow Q_1\overline{\delta}_{X_1} \subseteq Q_2\overline{\delta}_{X_1} \wedge Q_1\overline{\lambda}_{X_1} \subseteq Q_2\overline{\lambda}_{X_1}$

ii) $x_1 \subseteq x_2 \Rightarrow Q_1\overline{\delta}_{X_1} \subseteq Q_1\overline{\delta}_{X_2} \wedge Q_1\overline{\lambda}_{X_1} \subseteq Q_1\overline{\lambda}_{X_2}$

iii) $x_1 \subseteq x_2 \wedge Q_1 \subseteq Q_2$
$\Rightarrow Q_1\overline{\delta}_{X_1} \subseteq Q_2\overline{\delta}_{X_2} \wedge Q_1\overline{\lambda}_{X_1} \subseteq Q_2\overline{\lambda}_{X_2}$

Proof. The properties (i) and (ii) follow directly from the definition of $\overline{\delta}_X$ and $\overline{\lambda}_X$. The property (iii) is evident because

$$Q_1 \subseteq Q_2 \Rightarrow \forall x \subseteq I : Q_1\overline{\delta}_X \subseteq Q_2\overline{\delta}_X \qquad \{(i)\} \qquad (1)$$
$$x_1 \subseteq x_2 \Rightarrow \forall Q \subseteq S : Q\overline{\delta}_{X_1} \subseteq Q\overline{\delta}_{X_2} \qquad \{(ii)\} \qquad (2)$$

Substituting x by x_1 in (1) and Q by Q_1 in (2) we have

$$Q_1\overline{\delta}_{X_1} \subseteq Q_2\overline{\delta}_{X_1} \quad \text{and} \quad Q_2\overline{\delta}_{X_1} \subseteq Q_2\overline{\delta}_{X_2}$$

By the transitivity of set inclusion we know

$$Q_1\overline{\delta}_{X_1} \subseteq Q_2\overline{\delta}_{X_2}.$$

For $Q_1\overline{\lambda}_{X_1} \subseteq Q_2\overline{\lambda}_{X_2}$,

the procedure of proof is exactly the same as above.

(End of Property 2.4)

<u>Property 2.5</u>

If $Q_1, Q_2 \subseteq S$, $x \in I$, then

$$Q_1\bar{\delta}_x \cup Q_2\bar{\delta}_x = (Q_1 \cup Q_2)\bar{\delta}_x$$
$$Q_1\bar{\lambda}_x \cup Q_2\bar{\lambda}_x = (Q_1 \cup Q_2)\bar{\lambda}_x$$

Proof. Let $Q_1 = \{p_1, p_2, \ldots, p_m\}$ and
$Q_2 = \{q_1, q_2, \ldots, q_n\}$, $m \le |S|$, $n \le |S|$.

Then,

$$Q_1\bar{\delta}_x \cup Q_2\bar{\delta}_x$$
$$= \{p_1\delta_x, p_2\delta_x, \ldots, p_m\delta_x\} \cup \{q_1\delta_x, q_2\delta_x, \ldots, q_n\delta_x\}$$
$$= \{p_1\delta_x, \ldots, p_m\delta_x, q_1\delta_x, \ldots, q_n\delta_x\}$$
$$= \{p_1, \ldots, p_m, q_1, \ldots, q_n\}\bar{\delta}_x$$
$$= (Q_1 \cup Q_2)\bar{\delta}_x$$

(End of Property 2.5)

<u>Property 2.6</u>

If $Q \subseteq S$, $x_1, x_2 \subseteq I$, then

$$Q\bar{\delta}_{x_1} \cup Q\bar{\delta}_{x_2} = Q\bar{\delta}_{(x_1 \cup x_2)},$$
$$Q\bar{\lambda}_{x_1} \cup Q\bar{\delta}_{x_2} = Q\bar{\lambda}_{(x_1 \cup x_2)}.$$

Proof. Suppose $Q = \{q_1, q_2, \ldots, q_n\}$,
$x_1 = \{i_1, i_2, \ldots, i_k\}$ and
$x_2 = \{j_1, j_2, \ldots, j_l\}$, $k \le |I|$, $l \le |I|$.

$$Q\bar{\delta}_{x_1} \cup Q\bar{\delta}_{x_2}$$
$$= \{q_1\delta_{i_1}, \ldots, q_1\delta_{i_k}\} \cup, \ldots, \cup \{q_n\delta_{i_1}, \ldots, q_n\delta_{i_k}\}$$
$$\cup \{q_1\delta_{j_1}, \ldots, q_1\delta_{j_l}\} \cup, \ldots, \cup \{q_n\delta_{j_1}, \ldots, q_n\delta_{j_l}\}$$
$$= \{q_1\delta_{i_1}, \ldots, q_1\delta_{i_k}, q_n\delta_{j_1}, \ldots, q_n\delta_{j_l}\}$$
$$\cup, \ldots, \cup \{q_n\delta_{i_1}, \ldots, q_n\delta_{i_k}, q_n\delta_{j_1}, \ldots, q_n\delta_{j_l}\}$$
$$= \{q_1, \ldots, q_n\}\bar{\delta}_{(i_1, \ldots, i_k, j_1, \ldots, j_l)}$$
$$= Q\bar{\delta}_{(x_1 \cup x_2)}.$$

With similar argument we can prove that

$$Q\bar{\lambda}_{x_1} \cup Q\bar{\lambda}_{x_2} = Q\bar{\lambda}_{(x_1 \cup x_2)}.$$

(End of Property 2.6)

Property 2.7

Let $Q_1, Q_2 \subseteq S$, $x \in I$, $x_1, x_2 \subseteq I$. Then

$$(Q_1 \cap Q_2)\overline{\delta}_x \subseteq Q_1\overline{\delta}_x \cap Q_2\overline{\delta}_x,$$

$$Q_1\overline{\delta}_{(x_1 \cap x_2)} \subseteq Q_1\overline{\delta}_{x_1} \cap Q_1\overline{\delta}_{x_2};$$

$$(Q_1 \cap Q_2)\overline{\lambda}_x \subseteq Q_1\overline{\lambda}_x \cap Q_2\overline{\lambda}_x,$$

$$Q_1\overline{\lambda}_{(x_1 \cap x_2)} \subseteq Q_1\overline{\lambda}_{x_1} \cap Q_1\overline{\lambda}_{x_2}.$$

Proof.

i) For all q in $Q_1 \cap Q_2$, $q \in Q_1$ and $q \in Q_2$

imply $q\overline{\delta}_x \subset Q_1\overline{\delta}_x$ and $q\overline{\delta}_x \subset Q_2\overline{\delta}_x$.

That is, $q\overline{\delta}_x \subset Q_1\overline{\delta}_x \cap Q_2\overline{\delta}_x$.

Therefore, $(Q_1 \cap Q_2)\overline{\delta}_x \subset Q_1\overline{\delta}_x \cap Q_2\overline{\delta}_x$.

ii) If $Q_1 = Q_2$, $Q_1 \cap Q_2 = Q_1 = Q_2$, then

$$(Q_1 \cap Q_2)\overline{\delta}_x = Q_1\overline{\delta}_x \cap Q_2\overline{\delta}_x.$$

Hence, $(Q_1 \cap Q_2)\overline{\delta}_x \subseteq Q_1\overline{\delta}_x \cap Q_2\overline{\delta}_x$.

In the same way, we have other three relations.

(End of Property 2.7)

Property 2.8

Let $Q_1, Q_2 \subseteq S$, $x_1, x_2 \subseteq I$. Then

$$(Q_1 \cup Q_2)\overline{\delta}_{(x_1 \cup x_2)} = \bigcup_{i,j=1,2} Q_i\overline{\delta}_{x_j},$$

$$(Q_1 \cup Q_2)\overline{\lambda}_{(x_1 \cup x_2)} = \bigcup_{i,j=1,2} Q_i\overline{\lambda}_{x_j}.$$

Proof.

$$\begin{aligned}
&(Q_1 \cup Q_2)\overline{\delta}_{(x_1 \cup x_2)} \\
&= Q_1\overline{\delta}_{(x_1 \cup x_2)} \cup Q_2\overline{\delta}_{(x_1 \cup x_2)} && \{\text{Prop. } 2.5\} \\
&= Q_1\overline{\delta}_{x_1} \cup Q_1\overline{\delta}_{x_2} \cup Q_2\overline{\delta}_{x_1} \cup Q_2\overline{\delta}_{x_2} && \{\text{Prop. } 2.6\} \\
&= \bigcup_{i,j=1,2} Q_i\overline{\delta}_{x_j}. && \{\text{calculus}\}
\end{aligned}$$

Similarly we have $(Q_1 \cup Q_2)\overline{\lambda}_{(x_1 \cup x_2)} = \bigcup_{i,j=1,2} Q_i\overline{\lambda}_{x_j}$.

(End of Property 2.8)

From Property 2.8 it is easy to see that

$$Q_1\overline{\delta}_{x_1} \cup Q_2\overline{\delta}_{x_2} \neq (Q_1 \cup Q_2)\overline{\delta}_{(x_1 \cup x_2)},$$

$$Q_1\overline{\lambda}_{x_1} \cup Q_2\overline{\lambda}_{x_2} \neq (Q_1 \cup Q_2)\overline{\lambda}_{(x_1 \cup x_2)}.$$

For the sake of convenience we make

<u>Notation 2.3</u>

$$Q_1\overline{\delta}_{x_1} \times Q_2\overline{\delta}_{x_2} = (Q_1 \cup Q_2)\overline{\delta}_{(x_1 \cup x_2)};$$

$$Q_1\overline{\lambda}_{x_1} \times Q_2\overline{\lambda}_{x_2} = (Q_1 \cup Q_2)\overline{\lambda}_{(x_1 \cup x_2)}.$$

(End of Notation 2.3)

<u>Notation 2.4</u>

Let $x = x_1x_2...x_k \in I^*$, $s \in S$. Then, functions

$$\tilde{\delta}_x : S \to S^*$$

and $$\tilde{\lambda}_x : S \to O^*$$

are defined by

$$s\tilde{\delta}_x = s\tilde{\delta}_{x_1...x_k}$$
$$= (s\delta_{x_1})(s\delta_{x_1x_2})...(s\delta_x)$$

and

$$s\tilde{\lambda}_x = s\tilde{\lambda}_{x_1...x_k}$$
$$= (s\lambda_{x_1})(s\lambda_{x_1x_2})...(s\lambda_x).$$

(End of Notation 2.4)

Obviously, $s\tilde{\delta}_x$ and $s\tilde{\lambda}_x$ record the tracks of a machine under input sequence x.

<u>Property 2.9</u>

Let $x_1, x_2 \in I$. Then, for $s \in S$

$$s\tilde{\delta}_{x_1x_2} = (s\delta_{x_1})(s\delta_{x_1x_2})$$

$$s\tilde{\lambda}_{x_1x_2} = (s\lambda_{x_1})(s\lambda_{x_1x_2})$$

Proof. Take k=2 in Notation 2.4.

(End of Property 2.9)

Property 2.10

Let $x_1, x_2 \in I^*$. Then

$$s\tilde{\delta}_{x_1x_2} = (s\tilde{\delta}_{x_1})(s\delta_{x_1}\tilde{\delta}_{x_2})$$

$$s\tilde{\lambda}_{x_1x_2} = (s\tilde{\lambda}_{x_1})(s\delta_{x_1}\tilde{\lambda}_{x_2})$$

Proof. Take $x = x_1x_2$ in Notation 2.4.

(End of Property 2.10)

Notation 2.5

Let A be a collection of n-arrangements of the state set, and let B be a collection of n-arrangements of the output set, and $x \in I$.

Then vector functions,

$$\vec{\delta}_x : A \to A \qquad \vec{\lambda}_x : A \to B$$

are defined for any arrangement in A

$$a = (a_1a_2 \dots a_n)$$

$$a\vec{\delta}_x = (a_1\delta_x)(a_2\delta_x)\dots(a_n\delta_x)$$

$$a\vec{\lambda}_x = (a_1\lambda_x)(a_2\lambda_x)\dots(a_n\lambda_x)$$

(End of Notation 2.5)

It is obvious that $\vec{\delta}$ keeps n endpoints of n tracks of a machine under input x. From the definition in Notation 2.5, it is easy to induce the following properties.

Property 2.11

If $x, y \in I$ and $a \in A$, then

$$a\vec{\delta}_{xy} = (a\vec{\delta}_x)\vec{\delta}_y,$$

$$a\vec{\lambda}_{xy} = (a\vec{\delta}_x)\vec{\lambda}_y.$$

(End of Property 2.11)

Property 2.12

If $x = x_1 \dots x_n \in I^*$ and $a \in A$, then

$$a\vec{\delta}_x = (\dots((a\vec{\delta}_{x_1})\vec{\delta}_{x_2})\dots\vec{\delta}_{x_n})$$

$$= a\vec{\delta}_{x_1}\vec{\delta}_{x_2}\dots\vec{\delta}_{x_n}$$

$$a\vec{\lambda}_x = a\vec{\delta}_{x_1\dots x_{n-1}}\ \vec{\lambda}_{x_n}.$$

If $x = \varepsilon \in I^*$, then $a\vec{\delta}_\varepsilon = a \quad a\vec{\lambda}_\varepsilon = \varepsilon$.

(End of Property 2.12)

<u>Summary</u>

1. $\delta_x: S \to S$; $\lambda_x: S \to O$;
   $s \in S, x \in I^*$: $s\delta_x \in S$; $s\lambda_x \in O$.
2. $\bar{\delta}_x: 2^S \to 2^S$;
   $Q \subseteq S, x \subseteq I$: $Q\bar{\delta}_x \subseteq S$; $Q\bar{\delta}_x \in 2^S$; $Q\bar{\lambda}_x \subseteq O$; $Q\bar{\lambda}_x \in 2^O$.
3. $\tilde{\delta}_x : S \to S^*$; $\tilde{\lambda}_x: S \to O^*$;

   a) $s \in S, x \in I^*$: $s\tilde{\delta}_x \in S^*$; $s\tilde{\lambda}_x \in O^*$;

   b) $x \in I$: $S\tilde{\delta}_x \in S^*$; $s\tilde{\lambda}_x \in O^*$.
4. $\vec{\delta}_x: A \to A$; $\vec{\lambda}_x: A \to B$;
   $x \in I^*$, $a \in A$:
   $a\vec{\delta}_x \in A$, $a\vec{\lambda}_x \in B$;
   $a\vec{\delta}_\varepsilon = a$, $a\vec{\lambda}_\varepsilon = \varepsilon$.

2.3 Decomposition of machines

The decomposition theory of machines states that, for a given finite state machine M, the theory finds some "simpler" machines $M_1,M_2,\ldots,M_n$, in some sense and constructs them so that the connections of $M_1,M_2,\ldots,M_n$ can realize the machine M. That is, we expect statements of the form :-

$$M = M_1\omega_1M_2\omega_2\ldots\omega_{n-1}M_n$$

where $M,M_1,\ldots,M_n$ are the machines and $\omega_1,\omega_2,\ldots,\omega_{n-1}$ are the connections defined in suitable ways.

When we say "simpler" machines, there are different meanings for the word "simpler". During the 1960's, it meant that the number of states in the component machines was less than in the original machine, because it was associated with the number of memory components for the physical implementation of machines.

To cut down the cost of implementation, we must reduce the number of states in the mathematical models. With the development of LSI and VLSI techniques, the problem of reducing the components becomes less important. But the number of pins of an IC still is a serious limitation. Presently, the "simpler" means less pins, which appears mathematically as fewer inputs and outputs, as well as states of the machines. In this thesis, we shall consider decompositions based on the latter meaning of "simpler".

Decompositions can be classified in different ways. According to the number of component machines, there are two types of decompositions: the *simple decomposition* and the *complex decomposition*. A simple decomposition is necessarily of the form :-

$$M = M_1 \ \omega \ M_2,$$

that is, it contains only two component machines M_1 and M_2. If it contains more than two component machines, the decomposition is said to be complex. A *state decomposition* is characterized by the mapping on sets of states; for instance, for simple decomposition,

$$\Phi : S \to S_1 \times S_2$$

which means that the component machines have common inputs.
A full-decomposition is characterized not only by the state mapping Φ, but also, by mappings on input sets and output sets :-

$$\Psi : I \to I_1 \times I_2 \qquad\qquad \theta : O \to O_1 \times O_2$$

with some restrictions: $|S_i| < |S|$, $|I_i| < |I|$, and $|O_i| < |O|$, i=1,2. It is apparent that state decomposition is just a special case of full-decomposition.

Also, the decompositions can be classified according to the relationships existing between the component machines. If one component machine takes some messages, such as states or outputs, from another component machine, the decomposition is said to be a *serial decomposition*. Otherwise, the decomposition is a *parallel decomposition*. For complex decompositions, there also exist *series paralled decompositions*, in which some machines are connected in parallel and some in series.

Due to the different approaches to decompositions there are different theories which are used in the books and literature about decompositions. One of them is algebraic theory. It involves semigroups [5,6,9,16] and partition [11-15] theories. But most of them are concerned with the partition concept [5,6,9,11-16]. In this thesis, we are going to study the simple full-decompositions of Mealy machines using the trinity theory based on the partition concept.

2.4 A Universal Connection Model and Decompositions

In this section a universal connection model is introduced. A number types of decompositions are derived from the model and discussed.

2.4.1 A Universal Connection Model

Consider how to connect two machines, M_1 and M_2,

$$M_i = (I_i, S_i, O_i, \delta^i, \lambda^i), \ i=1,2.$$

We take Ω as a variable to denote a set of S_1, O_1 or an empty set $\emptyset$ and I_2' as a middle variable to hold a projection from an input set I to M_2. If we make three relations η_1, η_2 and η_3 by

η_1: from I to I_1 and I_2';

η_2: from Ω and I_2' to I_2;

η_3: from O_1 and O_2 to O,

then M_1 and M_2 have been connected by η_1, η_2 and η_3 and a machine with input and output sets I and O has been realized by the connection. Since Ω and I_2' are variable, the connection includes many different connections by assigning Ω and I_2'. Thus, the connection is called an *universal connection* precisely defined by Definition 2.4.

<u>DEFINITION 2.4</u>

A universal connection of two machines M_1 and M_2 is the machine $M_1 \text{ c } M_2$ described by

$$M_1 \text{ c } M_2 = (I, S_1 \times S_2, O, \delta^c, \lambda^c)$$

where I and O are defined by η_1^{-1} and η_3;

δ^c and λ^c are defined by

$$(s_1, s_2)\delta_x^c = (s_1 \delta^1_{\eta_1(\cdot x)}, s_2 \delta^2_{\eta_2(\omega, \eta_1(x \cdot))}),$$

$$(s_1, s_2)\lambda_x^c = \eta_3(s_1 \lambda^1_{\eta_1(\cdot x)}, s_2 \lambda^2_{\eta_2(\omega, \eta_1(x \cdot))}),$$

for all $(s_1, s_2) \in S_1 \times S_2$, $x \in I$ and $\omega \in \Omega$.

(End of Definition 2.4)

A universal connection model is illustrated by Fig. 2.1.

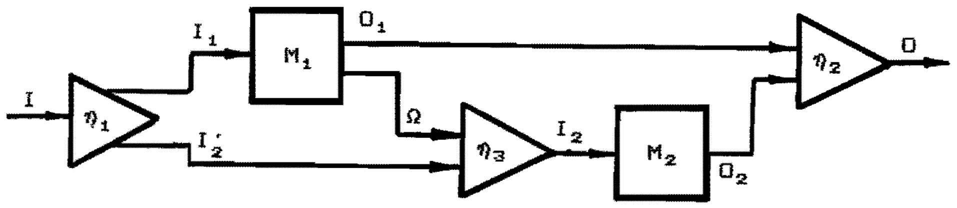


Fig. 2.1 Universal Connection

Note that $\eta_1(\cdot i)$ denotes the first component of $\eta_1(i)$ and $\eta_1(i\cdot)$ the second component of $\eta_1(i)$. In the figure, a trilateral sign represents a relation and the direction of a sign indicates the direction of a mapping. We will apply these notations throughout the thesis.

A universal connection model presents just a general connection of two machines. When the relations and variables η_1, η_2 and η_3 are specified, it will give a practical connection. In other words, a universal model includes all the simple connections. Since a great number of simple connections can be derived in this way, we are going to derive some of the decompositions which are available or have been developed in this thesis.

2.4.2 Machine decompositions

In this section, some serial and parallel decomposition types are introduced that are based on different assignments of the quadruple $(\Omega,\eta_1,\eta_2,\eta_3)$. An assignment represents a set of concrete definitions of Ω and the relations.

From the model, we know that a parallel connection can be obtained if we make $\Omega = \emptyset$. Otherwise, the model is connected in series. Furthermore, if η_3 is a null relation and $O_1 = O_2 = \emptyset$, the model serves for connecting states machines.

Let $\Omega \neq \emptyset$. Then, many serial decompositions are obtained as follows, by making particular definitions for the relations.

Serial Decompositions

1. Serial decompositions with common inputs.

ASSIGNMENT 1.

$\Omega = S_1/O_1$;

η_1: $I \to I_1 \times I_2'$ $\{I=I_1=I_2'$; $\eta_1(x)=(x,x)$, $x \in I\}$;

η_2: $\Omega \times I \to I_2$ $\{I_2=\Omega \times I$; identity$\}$;

η_3: $O_1 \times O_2 \to O$.

Substituting them in the model, we get a serial decomposition. The structure is shown in Fig. 2.2.

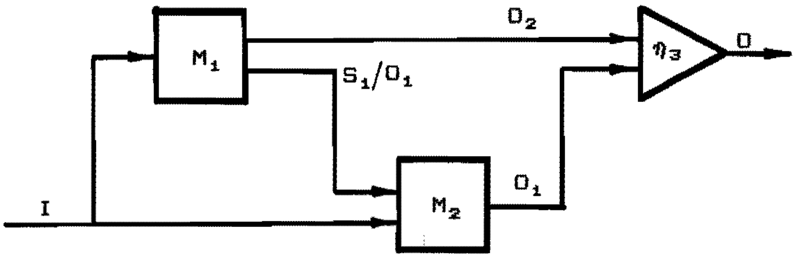
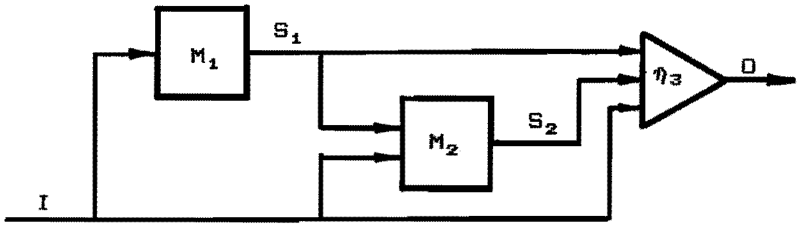


Fig. 2.2 A serial decomposition

Since O_1 and O_2 are functions of S_1, S_2 and I, the relation η_3 also can be written as follows

η_3 : $S_1 \times S_2 \times I \to O$

Fig. 2.3 gives the connection under the definitions above.



$\delta((s_1,s_2),x) = (\delta^1(s_1,x),\delta^2(s_2,x))$

$\lambda((s_1,s_2),x) = \eta_3(s_1,s_2,x)$

Serial decomposition with output functions.

Fig. 2.3 $M_1 \to M_2$

The pattern of decompositions based upon this type of connection are described in most of the literature about machine decompositions. Hartmanis gave a detailed discussion on the way how to get a serial decomposition in [13-15]. The decompositions were called *serial decompositions with common input and output functions*. The key for finding such a serial decomposition is to look for an SP partition in a given machine. If the partition exists, then the machine can be decomposed into a network consisting of two component machines M_1 and M_2.

Because, for any $s \in S$ there certainly is a corresponding s_1 and s_2 such that $\delta(s,x)$ can be mapped to $(\delta^1(s_1,x),\delta^2(s_2,x))$, the $\lambda(s,x)$ then can be represented by the combination of s_1, s_2 and x. Hence, η_3 is defined by

$$\eta_3(s_1,s_2,x) = \lambda(s,x)$$

if $s = (s_1,s_2)$.

For this type of decomposition, we should note that it only realizes a state decomposition which means that, for each of the component machines the number of inputs is larger than or equal to that of the original machines. Moreover, the outputs of machine M are given by η_3 which is a complicated mapping rather than $\eta_3' : O \to O_1 \times O_2$.

A proper input and output decomposition should be of proper mappings

$$I \to I_1 \times I_2, \quad O \to O_1 \times O_2 \ .$$

2. Complete Serial Decompositions

ASSIGNMENT 2.

$\Omega = O_1$;

η_1: $I \to I_1$ $\{I_2' = \emptyset;\ I_1 = I\}$;

η_2: $\Omega \to I_2$ $\{I_2' = \emptyset;\ I_2 = \Omega$ (identity) or $I_2 \neq \Omega\}$;

η_3: $O_2 \to O$ $\{O = O_2\}$

Assignment 2 states that if we make some restrictions such as $O_1 \neq \emptyset$, omitting output function λ and I_2', then, Fig. 2.3 becomes either Fig. 2.4(a) or 2.4(b).

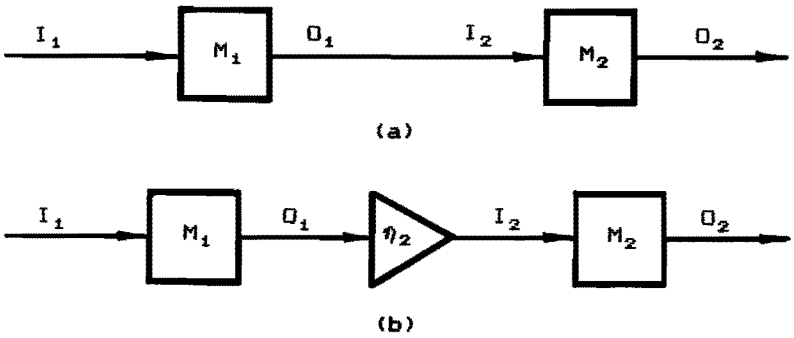


Fig. 2.4 Completely serial decompositions.

The decomposition based on a completely serial connection is called a *completely serial decomposition*. The connection shown in Fig. 2.4(a) appeared in [15,29] and the one shown in Fig. 2.4(b) was defined in [16].

3. General Serial Decompositions.

ASSIGNMENT 3.

$\Omega = S_1$;

η_1: $I \to I_1 \times I'_2$ $\{I=I_1=I'_2$; identity$\}$;

η_2: $S_1 \times I \to I_2$ $\{\eta_2 = \{f_x: S_1 \to I_2\}, x \in I\}$;

η_3: $O_1 \times O_2 \to O$.

Let $I = I_1 = I'_2$ and let η_1 be an identity relation between I and (I_1, I'_2), $\eta_2 = \{f_x \mid f_x: S_1 \to I_2$ and $x \in I\}$, $\Omega = S_1$.
A general serial connection is formed and shown in Fig. 2.5.

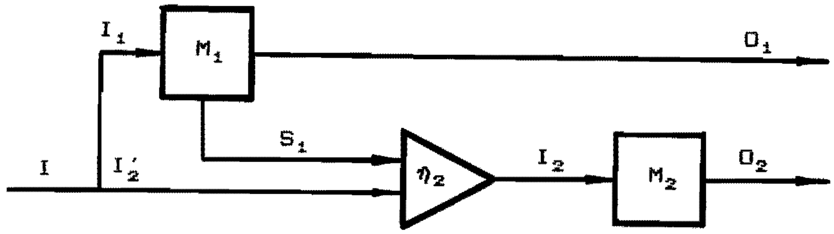


Fig. 2.5 General Serial Decomposition

If a machine can be realized by two component machines that are connected in the way indicated in Fig. 2.5, then, the connection is a general serial decomposition of a machine. It implies a special case as Fig. 2.5 where $|I_2| = |I| \times |S_1|$.

In [16] it was pointed out that when there are two machines M_1 and M_2 of which the semigroups cover the semigroup of M, then, the general serial connection of M_1 and M_2 covers M.

4. Wreath Decomposition.

ASSIGNMENT 4.

$\Omega = S_1$;
η_1: $I \rightarrow I_1 \times I'_2$ $\{I = I_1 \times I'_2\}$;
η_2: $S_1 \times I'_2 \rightarrow I_2$ $\{\eta_2 = I_2^{S_1} = \{f: S_1 \rightarrow I_2\}, f \in I'_2\}$;
η_3: $O_1 \times O_2 \rightarrow O$.

From a general serial connection, if we give a definition for η_1 as

$\eta_1 : I \rightarrow I_1 \times I'_2$

and take an extreme case of η_2 as

$\eta_2 = I_2^{S_1} = \{f: S_1 \rightarrow I_2\}$

then, a wreath connection of M_1 and M_2 is defined and it is illustrated in Fig. 2.6.

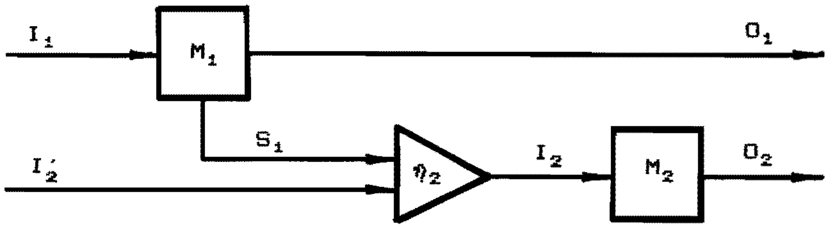


Fig. 2.6 Wreath Connection

A wreath decomposition is discussed with the semigroup theory in [16]. In Chapter 6 of this thesis, we shall discuss it with partition trinity theory.

5. Serial Full-decompositions

ASSIGNMENT 5.

$\Omega = S_1/O_1$;

η_1: $I \to I_1 \times I'_2$ $\{I = I_1 \times I'_2\}$;

η_2: $\Omega \times I'_2 \to I_2$ $\{I_2 = \Omega \times I'_2\}$;

η_3: $O_1 \times O_2 \to O$ $\{O = O_1 \times O_2\}$

Another important special case of general serial connections is to make the retraction η_2 an identity mapping from (ω, I'_2) to I_2, $\Omega = S_1$ or O_1, and η_1 be an identity mapping from I to $I_1 \times I'_2$.

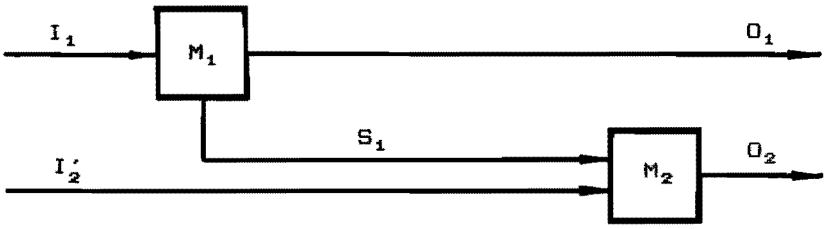


Fig. 2.7 Serial Full-decomposition

Serial full-decomposition will be defined by this connection in Chapter 5 and the methods for these decompositions will be described too. Since the difference required for the connected information is S_1 or O_1, the methods appear to be quite different. The decomposition refers to *state serial full-decomposition (type II)* for $\Omega = S_1$, as well as, *output serial full-decomposition (type I)* for $\Omega = O_1$.

Now, we consider the case of $\Omega = \emptyset$ which offers some parallel decompositions using the different definitions of the relations.

6. Partial Parallel Decompositions

ASSIGNMENT 6.

$\Omega = \emptyset$;

$\eta_1 : I \to I_1 \times I_2'$ $\{I = I_1 = I_2'\}$;

$\eta_2 : I_2' \to I_2$ $\{I_2 = I_2'\}$;

$\eta_3 : O_1 \times O_2 \to O$.

We take $I = I_1 = I_2'$ and η_1 as an identity mapping from I to $I_1 \times I_2'$. Moreover, η_2 is defined as an identity mapping from I_2' to I_2 and η_3 as $O_1 \times O_2 \to O$. A parallel connection with common inputs is obtained. The connection is called a *partial parallel connection*.

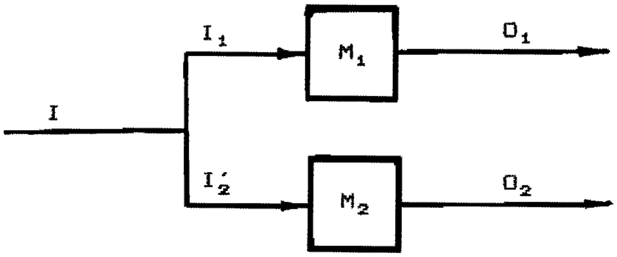


Fig. 2.8 Partial Parallel Decomposition.

A machine can be decomposed into a partial parallel connection of two component machines, if it exists. Such a decomposition is discussed in most of the books on the subject of machine decomposition theory. The key for decomposing a given machine is to find two orthogonal SP partitions. If there are no such partitions for the machine, it means that the machine cannot be decomposed in parallel [8,12,15].

If the SP partitions are output consistent, then, the outputs can be mapped into a proper product of O_1 and O_2. Otherwise, we have to use a mapping $\eta_3 : S_1 \times S_2 \times I \to O$ in order to produce the outputs of the original machine. When M_1 and M_2 are state machines, the decomposition is discussed in [23,24].

7. Parallel Full-decomposition.

ASSIGNMENT 7.

$\Omega = \emptyset$;

η_1: $I \rightarrow I_1 \times I'_2$ $\{I = I_1 \times I'_2\}$;

η_2: $I'_2 \rightarrow I_2$ $\{I_2 = I'_2\}$;

η_3: $O_1 \times O_2 \rightarrow O$ $\{O = O_1 \times O_2\}$.

Now, we will consider a special case of partial parallel decomposition. If we make the relation η_1 a proper direct product of I_1 and I_2, i.e.

$$\eta_1 : I \rightarrow I_1 \times I_2 ,$$

then, a model of a parallel full-decomposition is obtained. We are especially interested in this decomposition, because it gives the exact decomposition of states, inputs and outputs which leads to a reduction of the number of pins on devices implementing the decomposition.

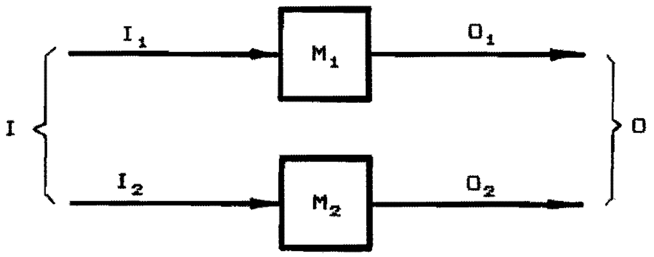


Fig. 2.9 Parallel Full-decomposition

In Chapter 4 of this thesis, we shall discuss methods to find such a full-decomposition, if it exists, for a given machine using the theory of a partition trinity.

8. H-decomposition

ASSIGNMENT 8.

$\Omega = \emptyset$;

η_1: $I \to I_1 \cup I_2'$;

η_2: $I_2' \to I_2 \quad \{I_2 = I_2'\}$;

η_3: $O_1 \cup O_2$ / $O_1 \times O_2 \to O$.

Based on the definition for a parallel full-decomposition, we introduce another decomposition which looks like a full-decomposition by making the mappings into the union of inputs or outputs of component machines. Particularly,

$$\eta_1 : I \to I_1 \cup I_2$$
$$\eta_3 : O_1 \cup O_2 \to O \text{ or } O_1 \times O_2 \to O.$$

With these definitions the component machine works like:

$$\delta((s_1,s_2),i) = \begin{cases} (\delta^1(s_1,i),s_2) & \text{if } i \in I_1 \\ (s_1,\delta^2(s_2,i)) & \text{if } i \in I_2 \end{cases}$$

Which means, for some input,s one component machine acts and the other keeps stationary. Therefore, we call it an *H-decomposition*.

An H-decomposition has the same structure as a parallel full-decomposition, except for the definition of η_1. It is supplementary to the full-decomposition theory. A detailed discussion will be given in Chapter 6 later. A similar decomposition only on states is described in [2,3].

9. The Holonomy Decomposition

In the algebraic decomposition theory of sequential machines, the first major well-known result was the holonomy decomposition [6,16]. It is also called the *Krohn-Rhodes decomposition* due to Krohn and Rhodes who gave an algorithmic procedure for such a decomposition [19]. The Krohn-Rhodes decomposition theorem says that every semiautomaton can be covered by direct and cascade products of semiautomata of two kinds: (a) simple grouplike semiautomata, (b) two-state reset semiautomata [9]. In other words, every finite state machine can be realized by a series-parallel connection of permutation machines and two-state reset-identity machines. The series-parallel connection is depicted in Fig. 2.10, which is copied from [8]. The n is the number of states of the machine to be decomposed; P denotes a permutation machine and R represents a two-state reset-identity machine.

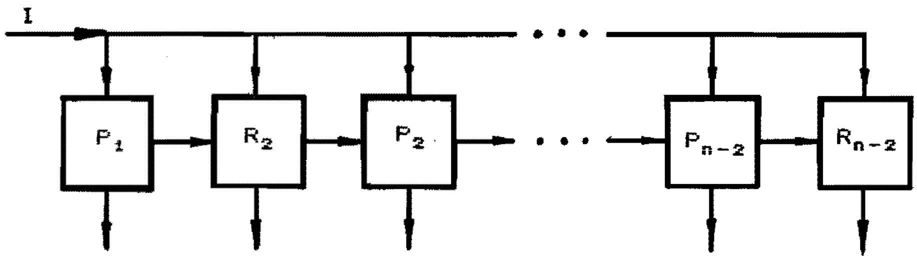


Fig. 2.10 Canonical Decomposition of Finite State Machine

The theorem is excellent because it can be adapted to every state machine unconditionally. Thus, an alternate name for it is the universal canonical decomposition theorem. However, the reasons for hesitating to apply it to the full-decomposition are twofold. One is: that all component machines, in general, take the same inputs from a common set I. Another is because: the decomposition is a complex decomposition and not considered in this thesis.

CHAPTER 3

PARTITION TRINITY AND TRINITY ALGEBRA

In this chapter we will begin by developing some mathematical tools and theorems which are fundamental to the theory of full-decomposition of sequential machines.

3.0 Introduction

As we know, the elementary structure theory of serial or parallel realizations of state behaviours is derived through state partitions which represent self-dependent information. The concepts of information and information dependence are very basic and underlie all the structure results. In this chapter, we wish to consider more useful mathematical tools for describing the concepts of information and information dependence in all the aspects of a sequential machine.

From the available theory, we know that, if a partition π on the set of states of a sequential machine has the substitution property, then as long as we know the block of π which contains a given state of the machine, we can compute the block of π to which that state will be transformed by any given input sequence.

Furthermore, if partitions π and τ form an S-S pair (π,τ) on the machine, then, as long as we know the block of π which contains the state of the machine, we can compute the block of τ to which this state will be transferred by the machine, for every input. Similarly, if (ξ,τ) is an I-S pair, then as long as we only know the block ξ which contains the input of the machine, we can compute for every present state the block of τ to which this input makes the state transferred by the machine, and so on. It may be said that a pair gives the information dependence in the part aspect, such as, present state to next state, input to next state, and etc. The concept of partition trinity is more general and is introduced to study how all the information flows through a sequential machine when it is in operation.

From the discussion that follows, we will know that, from the viewpoint of mathematics, the partition trinity is the hard-core of all concepts of mathematics for a sequential machine, because some partitions have the PP property, some PP's have a SP and some PP's with SP have partition trinity property. Fig. 3.1 shows the inclusion relations among the concepts of partitions, partition pairs, SP partitions, and partition trinities on a machine.

P : Partitions

PP: Partition Pairs

SP: SP partitions

PT: Partition Trinities

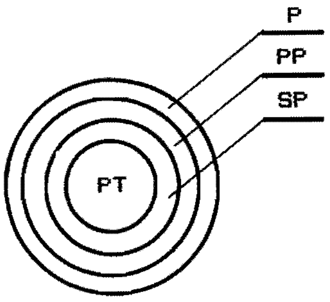


Fig. 3.1 Inclusion relation among P,PP,SP and PT concepts

3.1 Partition Trinity

3.1.1 Partition Pair

The concept of a partition pair (PP) was first introduced for the study of sequential machines by Hartmanis [10,14]. Here, we will recall some of its main points and derive some properties of them in order to develop it to a higher level, as a mathematical tool for the further study of sequential machines.

DEFINITION 3.1

For a machine $M = (I,S,O,\delta,\lambda)$, let π, τ, ξ and ω be the partitions on M and, in particular

π, τ on S; ξ on I; ω on O.

Then, we define

i) (π,τ) is an S-S pair *if and only if*
$\forall B\in\pi,\ \forall x\in I :\ B\bar{\delta}_x \subseteq B' \in \tau$

ii) (ξ,τ) is an I-S pair *if and only if*
$\forall C\in\xi,\ \forall s\in S :\ s\bar{\delta}_C \subseteq B' \in \tau$

iii) (π,ω) is an S-O pair *if and only if*
$\forall B\in\pi,\ \forall x\in I :\ B\bar{\lambda}_x \subseteq Q \in \omega$

iv) (ξ,ω) is an I-O pair *if and only if*
$\forall C\in\xi,\ \forall s\in S :\ s\bar{\lambda}_C \subseteq Q \in \omega$

(End of Definition 3.1)

LEMMA 3.1

If (π_1,τ_1) and (π_2,τ_2) are PP's on a machine M, then

i) $(\pi_1\cdot\pi_2,\ \tau_1\cdot\tau_2)$ is an PP on M, and

ii) $(\pi_1+\pi_2,\ \tau_1+\tau_2)$ is an PP on M.

Proof. Suppose (π_1,τ_1) and (π_2,τ_2) are S-S pairs.

i) $B\in(\pi_1\cdot\pi_2)$

$\Rightarrow B \subseteq B'\in\pi_1 \wedge B \subseteq B''\in\pi_2$ {def. of partition product [15]}

$\Rightarrow B\bar{\delta}_x \subseteq \mathcal{B}'\in\tau_1 \wedge B\bar{\delta}_x \subseteq \mathcal{B}''\in\tau_2$ {(π_1,τ_1), (π_2,τ_2)}

$\Rightarrow B\bar{\delta}_x \subseteq \mathcal{B}'\cap\mathcal{B}''$ {calculus}

$\Rightarrow B\bar{\delta}_x \subseteq \mathcal{B}\in(\tau_1\cdot\tau_2)$ {def. of partition product}

which shows that $(\pi_1\cdot\pi_2,\tau_1\cdot\tau_2)$ is an PP.

ii) $B\in(\pi_1+\pi_2)$

$\Rightarrow \exists B_1,B_2,\ldots,B_k,\ B_i\in\pi_1 \vee B_i\in\pi_2$: {def. of partition sum [15]}

$B_j\cap B_{j+1}\neq\emptyset \ \wedge\ \bigcup_{i=1}^{k} B_i = B \quad j=1..k-1$

$\Rightarrow B\bar{\delta}_x$ {statement}

$= (\bigcup_{i=1}^{k} B_i)\bar{\delta}_x$ {substitution}

$= \bigcup_{i=1}^{k} (B_i\bar{\delta}_x)$ {Prop. 2.5}

$\Rightarrow B_i\bar{\delta}_X \cap B_{i+1}\bar{\delta}_X \neq \emptyset \quad i=1..k-1 \qquad \{B_i \cap B_{i+1} \neq \emptyset\}$

$\wedge (B_j\bar{\delta}_X \subseteq B' \in \tau_1 \quad \text{if } B_j \in \pi_1 \qquad \{(\pi_1,\tau_1)\}$

$\vee B_j\bar{\delta}_X \subseteq B'' \in \tau_2 \quad \text{if } B_j \in \pi_2) \qquad \{(\pi_2,\tau_2)\}$

$\Rightarrow \bigcup_{i=1}^{k} (B_i\bar{\delta}_X) \subseteq \mathcal{B} \in (\tau_1+\tau_2) \qquad$ {def. of partition sum}

$= B\bar{\delta}_X \subseteq \mathcal{B} \in (\tau_1+\tau_2). \qquad$ {substitution}

Therefore, we have that

$(\pi_1+\pi_2,\ \tau_1+\tau_2)$ is an PP.

In the other cases of I-S, S-O, and I-O pairs, the proofs are the same as shown above, and may be omitted.

(End of Lemma 3.1)

It should be noted that in Lemma 3.1, (π_1,τ_1) and (π_2,τ_2) are always of the same type of pairs; otherwise, the lemma does not hold.

<u>LEMMA 3.2</u>

If (π,τ) is an PP, then

i) $\pi' < \pi$ implies that (π',τ) is an PP ;

ii) $\tau' > \tau$ implies that (π,τ') is an PP ;

iii) $\pi' < \pi$ and $\tau' > \tau$ imply that (π',τ') is an PP.

Proof. We consider the case where (π,τ) is as an I-O pair to prove.

i) $\pi' < \pi \wedge (\pi,\tau) \qquad$ {assume (π,τ) is an PP}

$\Rightarrow \forall B' \in \pi'\ \exists B \in \pi:\ B' \subset B$

$\wedge \forall B \in \pi\ \forall s \in S:\ s\bar{\lambda}_B \subseteq \mathcal{B} \in \tau \qquad$ {definition}

$\Rightarrow s\bar{\lambda}_{B'} \subseteq s\bar{\lambda}_B \subseteq \mathcal{B} \in \tau \qquad$ {$B' \subset B$, Prop. 2.4}

$\Rightarrow s\bar{\lambda}_{B'} \subseteq \mathcal{B} \in \tau. \qquad$ {calculus}

Hence (π',τ) is an I-O pair.

ii) By a similar argument.

iii) For (π',τ) using Lemma 3.2 (ii) again.

In the same way, we can prove for other cases that (π,τ) is an S-S, I-S, or S-O pair.

(End of Lemma 3.2)

Now, we wish to develop a theorem on partitions as follows.

<u>THEOREM 3.1</u>

Let π_1, π_2 and π_3 be partitions on the same set of a machine. If $\pi_1 \le \pi_2$ and $\pi_2 \le \pi_3$, then $\pi_1 \le \pi_3$.

Proof.

$\pi_1 \le \pi_2$ and $\pi_2 \le \pi_3$ imply

$$\pi_1 \cdot \pi_2 = \pi_1,\ \pi_2 \cdot \pi_3 = \pi_2, \qquad (1)$$

$$\pi_1 + \pi_2 = \pi_2,\ \pi_2 + \pi_3 = \pi_3. \qquad (2)$$

Then,
$$\begin{aligned} \pi_1 \cdot \pi_3 &= \pi_1 \cdot \pi_2 \cdot \pi_3 && \{(1)\} \\ &= \pi_1 \cdot \pi_2 && \{(1)\} \\ &= \pi_1; && \{(1)\} \\ \pi_1 + \pi_3 &= \pi_1 + \pi_2 + \pi_3 && \{(2)\} \\ &= \pi_2 + \pi_3 && \{(2)\} \\ &= \pi_3. && \{(2)\} \end{aligned}$$

Hence, $\pi_1 \le \pi_3$.

(Enf of Theorem 3.1)

3.1.2 Partition Trinity

<u>DEFINITION 3.2</u>

A *partition trinity* (π_I, π_S, π_O) on the machine

$$M = (I, S, O, \delta, \lambda)$$

is an ordered triple of partitions on the sets I, S and O, respectively, such that

$$\forall B \in \pi_S\ \forall C \in \pi_I :\quad B\overline{\delta}_C \subseteq B' \in \pi_S \quad \text{and} \quad B\overline{\lambda}_C \subseteq Q \in \pi_O$$

(End of Definition 3.2)

Thus, (π_I, π_S, π_O) is a partition trinity on M *If and only if* the blocks of π_S and π_I are mapped into the blocks of π_S and π_O by M. That is, for every block C in π_I and a block B in π_S, there exist a B′ in π_S and a Q in π_O, such that $B\overline{\delta}_C$ is in and only in B′ and $B\overline{\lambda}_C$ is in and only in Q.

This definition is suitable, in concept, for all kinds of machines, completely specified or incompletely specified. In this case that M is an incompletely specified machine, both $B\overline{\delta}_C$ and $B\overline{\lambda}_C$ probably contain "don't care" conditions. A detailed discussion will be presented in another chapter.

For completely specified machines, we have the following theorem.

THEOREM 3.2

Let $M = (I,S,O,\delta,\lambda)$ be a completely specified machine and π_S,π_I and π_O be three partitions on S, I and O, respectively. Then, (π_I,π_S,π_O) is a partition trinity *if and only if*

i) (π_S,π_S) is an S-S pair, and

ii) (π_I,π_S) is an I-S pair, and

iii) (π_S,π_O) is an S-O pair, and

iv) (π_I,π_O) is an I-O pair.

Proof.

Assume that (π_S,π_S), (π_I,π_S), (π_S,π_O) and (π_I,π_O) are pairs.

$(\pi_S,\pi_S) \wedge (\pi_I,\pi_S) \wedge (\pi_S,\pi_O) \wedge (\pi_I,\pi_O)$

$\Rightarrow \forall B\in\pi_S\ \forall C\in\pi_I\ \forall s\in S\ \forall x\in I$:

$B\overline{\delta}_x \subseteq B'\in\pi_S \wedge s\overline{\delta}_C \subseteq B''\in\pi_S$ $\{(\pi_S,\pi_S),(\pi_I,\pi_S)\}$

$\wedge B\overline{\lambda}_x \subseteq Q'\in\pi_O \wedge s\overline{\lambda}_C \subseteq Q''\in\pi_O$ $\{(\pi_S,\pi_O),(\pi_I,\pi_O)\}$

$\Rightarrow \forall s\in B\ \forall x\in C$: $B'=B'' \wedge Q'=Q''$ $\{(\pi_S,\pi_S),(\pi_S,\pi_O)\}$

$\Rightarrow \forall B\in\pi_S\ \forall C\in\pi_I$: $B\overline{\delta}_C \subseteq B'\in\pi_S \wedge B\overline{\lambda}_C \subseteq Q'\in\pi_O$ {calculus}

$\Rightarrow (\pi_I,\pi_S\ \pi_O)$ is an PT {def. of PT}

Conversely, we assume (π_I,π_S,π_O) is an PT.

(π_I,π_S,π_O)

$\Rightarrow \forall B\in\pi_S\ \forall C\in\pi_I$:

$B\overline{\delta}_C \subseteq B\in\pi_S \wedge B\overline{\lambda}_C \subseteq Q'\in\pi_O$ {def. of PT}

$\Rightarrow$ $B\overline{\delta}_{\{x_1,x_2,\ldots,x_k\}} \subseteq B'\in\pi_S$ {calculus by

$\wedge \{s_1,s_2,\ldots,s_j\}\overline{\delta}_C \subseteq B'\in\pi_S$ $B=\{s_1,s_2,\ldots,s_j\}\in\pi_S$

$\wedge B\overline{\lambda}_{\{x_1,x_2,\ldots,x_k\}} \subseteq Q'\in\pi_O$ $C=\{x_1,x_2,\ldots,x_k\}\in\pi_I\}$

$\wedge \{s_1,s_2,\ldots,s_j\}\overline{\lambda}_C \subseteq Q'\in\pi_O$

$\Rightarrow$ $\bigcup_{i=1}^{k}(B\overline{\delta}_{x_i}) \subseteq B'\in\pi_S$ {Prop. 2.6}

$\wedge \bigcup_{i=1}^{j}(B_i\overline{\delta}_C) \subseteq B'\in\pi_S$ {Prop. 2.5}

$\wedge \bigcup_{i=1}^{k}(B\overline{\lambda}_{x_i}) \subseteq Q'\in\pi_O$ {Prop. 2.6}

$\wedge \bigcup_{i=1}^{j}(s_i\overline{\lambda}_C) \subseteq Q'\in\pi_O$ {Prop. 2.5}

$\Longrightarrow B\overline{\delta}_{x_i} \subseteq B' \in \pi_S$ $\quad i=1..k$

$\wedge \; s_i\overline{\delta}_C \subseteq B' \in \pi_S$ $\quad i=1..j$

$\wedge \; B\overline{\lambda}_{x_i} \subseteq Q' \in \pi_O$ $\quad i=1..k$ $\qquad$ {calculus}

$\wedge \; s_i\overline{\lambda}_C \subseteq Q' \in \pi_O$ $\quad i=1..j$

$\Longrightarrow \forall B\in\pi_S \; \forall C\in\pi_I \; \forall s\in S \; \forall x\in I$:

$\wedge \; B\overline{\delta}_x \subseteq B' \in \pi_S$

$\wedge \; s\overline{\delta}_C \subseteq B' \in \pi_S$

$\wedge \; B\overline{\lambda}_x \subseteq Q' \in \pi_O$ $\qquad$ {calculus}

$\wedge \; s\overline{\lambda}_C \subseteq Q' \in \pi_O$

$\Longrightarrow (\pi_S,\pi_S)$ is an S-S pair

$\wedge \; (\pi_I,\pi_S)$ is an I-S pair

$\wedge \; (\pi_S,\pi_O)$ is an I-S pair $\qquad$ {Def's of pairs}

$\wedge \; (\pi_I,\pi_O)$ is an I-O pair

Hence the theorem.

(End of Theorem 3.2)

It should be mentioned again that Theorem 3.2 holds only for completely specified machines. For incompletely specified machines, it does not hold because (π_S,π_S) and (π_I,π_S) do not imply $B\overline{\delta}_C \subseteq B' \in \pi_S$, if there is a "don't care" condition in $B\overline{\delta}_C$. The concept of trinity for incompletely specified machines will be discussed in a later chapter.

In other words, from a partition trinity (π_I,π_S,π_O), if we only know the block of π_S which contains the state of M, then, we can compute, for every input block the blocks of π_S and π_O to which this state is transferred and the output is formed by M.

Since, from a PT, we know how "ignorance of all information of state, input and output spread" or "all information flows" through a sequential machine when it operates, it is obvious that a PT gives dependences of all the information of a sequential machine and it describes an integral characteristic of the machine. Therefore, it is a more useful tool for studying sequential machines than partition pairs.

Now, we should study the general properties and definitions of partition trinities on a sequential machine.

DEFINITION 3.3

A *cardinal trinity* (N_I, N_S, N_O) of PT (π_I, π_S, π_O) is an ordered triple of positive integers and it expresses the cardinal properties of the partition sets of π_I, π_S and π_O, respectively. Symbolically,

$$(N_I, N_S, N_O) = (|\pi_I| , |\pi_S| , |\pi_O|).$$

where $|x|$ is the cardinality of set x.
(End of Definition 3.3)

DEFINITION 3.4

Partition trinities (π_I, π_S, π_O) and (τ_I, τ_S, τ_O) are said to be *equal if and only if* the corresponding components are identical, that is,

i) $\pi_S = \tau_S$ on S, and
ii) $\pi_I = \tau_I$ on I, and
iii) $\pi_O = \tau_O$ on S.

(End of Definition 3.4)

DEFINITION 3.5

For PT's (π_I, π_S, π_O) and (τ_I, τ_S, τ_O) on a machine M,

$$(\pi_I, \pi_S, \pi_O) \geq (\tau_I, \tau_S, \tau_O)$$

if and only if

i) $\pi_S \geq \tau_S$ on S, and
ii) $\pi_I \geq \tau_I$ on I, and
iii) $\pi_O \geq \tau_O$ on O.

(End of Definition 3.5)

In the same manner, we can define the relations $>$ and $<$.

DEFINITION 3.6

An *identity trinity* T_I of a machine M is defined as

$$T_I = (\pi_I(I), \pi_S(I), \pi_O(I))\}$$

where $\pi_I(I)$, $\pi_S(I)$ and $\pi_O(I)$ are the identity partitions on I, S, and O, respectively.

A *zero trinity* T_O of a machine M is defined as

$$T_O = (\pi_I(O), \pi_S(O), \pi_O(O))\}$$

where $\pi_S(O)$, $\pi_I(O)$ and $\pi_O(O)$ are the zero partitions on S, I and O, respectively.
(End of Definition 3.6)

DEFINITION 3.7

A partition trinity (π_I,π_S,π_O) on a machine M is said to be *nontrivial if and only if*

i) $\pi_S \neq \pi_S(I)$ and $\pi_S \neq \pi_S(O)$, and

ii) $\pi_I \neq \pi_I(I)$ and $\pi_I \neq \pi_I(O)$, and

iii) $\pi_O \neq \pi_O(I)$ and $\pi_O \neq \pi_O(O)$.

(End of Definition 3.7)

DEFINITION 3.8

A partition trinity (π_I,π_S,π_O) is called a *basic partition trinity if and only if*

i) $\pi_I = \sum \{\pi'_I \mid (\pi'_I, \pi_S, \pi_O)$ is an PT on M$\}$, and

ii) $\pi_O = \prod \{\pi'_O \mid (\pi_I, \pi_S, \pi'_O)$ is an PT on M$\}$.

where $\sum$ and $\prod$ denote repeated addition and multiplication on partitions.

(End of Definition 3.8)

3.1.3 Trinity Algebra and Its Basic Properties

In this section, we look at the general properties of partition trinities on a sequential machine and work out some algebraic relationships that the partition trinities satisfy, such as trinity poset, trinity lattice and trinity algebra.

Let T be a set of all the partition trinities on a machine M. Considering the relation $\leq$ defined in Definition 3.5 for T, then, we have the next theorem.

THEOREM 3.3

The trinity set T on a machine M is a poset under relation $\leq$.

Proof. i) For any $x\in T$, $x=x$ implies $x\leq x$.

This states that $\leq$ is reflexive.

ii) Let $x,y\in T$ and $x=(X_I,X_S,X_O)$ and $y=(Y_I,Y_S,Y_O)$

$x\leq y$ implies that

$X_I \leq Y_I$, (1)

and $X_S \leq Y_S$, (2)

and $X_O \leq Y_O$. (3)

$y\leq x$ implies that

$Y_I \leq X_I$, (1')

and $Y_S \leq X_S$, (2')

and $Y_O \leq X_O$. (3')

Combining (1) and (1'), (2) and (2'), and (3) and (3'), we have

$X_I = Y_I$, $X_S = Y_S$, $X_O = Y_O$.

By Definition 3.4 it is true that x=y.

This shows that ≤ is antisymmetric.

iii) For any $x,y,z \in T$, $x \leq y$ and $y \leq z$ provide that

$$X_I \leq Y_I \quad \text{and} \quad Y_I \leq Z_I \tag{4}$$
$$X_S \leq Y_S \quad \text{and} \quad Y_S \leq Z_S \tag{5}$$
$$X_O \leq Y_O \quad \text{and} \quad Y_O \leq Z_O \tag{6}$$

Using Theorem 3.1 and Definition 3.5 for (4) through (6) we obtain

$$x \leq z .$$

This states that ≤ is transitive.

Hence the theorem.

(End of Theorem 3.3)

We introduce two binary operations ⊙ and ⊕ on the poset T, which are defined by the following definition.

DEFINITION 3.9

Let $x,y \in T$ and $x=(X_I,X_S,X_O)$ and $y=(Y_I,Y_S,Y_O)$.
The *trinity multiplication* and *trinity addition* are defined as follows.

$$x \odot y = (X_I \cdot Y_I,\ X_S \cdot Y_S,\ X_O \cdot Y_O)$$
$$x \oplus y = (X_I + Y_I,\ X_S + Y_S,\ X_O + Y_O)$$

where + and · are partition addition and multiplication.

$x \odot y$ is called a *trinity product*,

$x \oplus y$ is called a *trinity sum*.

(End of Definition 3.9)

Having obtained the operations on poset T, a problem naturally arises, that is, whether the trinity product (or sum) of any two PT's is a PT. The following theorem gives the answer and shows the proof in detail.

THEOREM 3.4

For any $x,y \in T$,

i) $x \oplus y \in T$;

ii) $x \odot y \in T$;

iii) $x \oplus T_I = T_I$, $x \odot T_I = x$,

iv) $x \odot T_O = T_O$, $x \oplus T_O = x$.

Proof. Let $x = (X_I, X_S, X_O)$ and $y = (Y_I, Y_S, Y_O)$.

i) $x, y \in T$ implies that

(X_S, X_S) , (Y_S, Y_S) (1)

(X_I, X_S) , (Y_I, Y_S) (2)

(X_S, X_O) , (Y_S, Y_O) (3)

and (X_I, X_O) , (Y_I, Y_O) (4)

are PP's. By Lemma 3.1 and (1)

$(X_S+Y_S,\ X_S+Y_S)$ is an S-S pair.

Similarly,

$(X_I+Y_I,\ X_S+Y_S)$ is an I-S pair.

$(X_S+Y_S,\ X_O+Y_O)$ is an S-O pair.

$(X_I+X_I,\ X_O+Y_O)$ is an I-O pair.

From Theorem 3.2, we know

$x \oplus y = (X_I+Y_I, X_S+Y_S, X_O+Y_O)$ is an PT.

Therefore, $x \oplus y \in T$.

ii) By the same argument as (i).

iii) $x \oplus T_I = (X_I, X_S, X_O) \oplus (\pi_I(I), \pi_S(I), \pi_O(I))$

$= (X_I+\pi_I(I), X_S+\pi_S(I), X_O+\pi_O(I))$

$= (\pi_I(I), \pi_S(I), \pi_O(I))$

$= T_I$;

$x \odot T_I = (X_I, X_S, X_O) \odot (\pi_I(I), \pi_S(I), \pi_O(I))$

$= (X_I \cdot \pi_I(I), X_S \cdot \pi_S(I), X_O \cdot \pi_O(I))$

$= (X_I, X_S, X_O)$

$= x.$

iv) It is similar to (iii).

(End of Theorem 3.4)

The definition and the theorem has shown that, for every pair of x and y in T, $x \odot y$ and $x \oplus y$ certainly exist. This gives a reminder that, under the operations of $\odot$ and $\oplus$, the poset T forms a lattice like the definition given below.

<u>DEFINITION 3.10</u>

A *trinity lattice* L_T is a triplet

$L_T = (T, \odot, \oplus)$

in which, for any $x, y \in T$

$GLB(x,y) = x \odot y$ $LUB(x,y) = x \oplus y$

where T is a nonempty set of all the partition trinities on a sequential machine, and $\odot$ and $\oplus$ are trinity multiplication and addition.

(End of Definition 3.10)

<u>THEOREM 3.5</u>

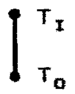
Any machine M has a finite trinity lattice with the identity element T_I and zero element T_O.

Proof. i) For any $x \in T$, by the Theorem 3.4

$$T_I \odot x = x, \quad T_I \oplus x = T_I,$$
$$T_O \odot x = T_O, \quad T_O \oplus x = x.$$

Hence, T_I is the identity element, and T_O is the zero element of L_T of a machine.

ii) Any machine has at least two trinities T_I and T_O which can form the simplest lattice:



iii) A finite machine implies that the partition sets of I, S and O, are finite. Any machine has a finite tri-partition set $L = \{P_I \times P_S \times P_O\}$, where P_S, P_I and P_O are sets of all the partitions on I,S and O, respectively. T is a subset of L; therefore, L_T is finite.

(End of Theorem 3.5)

<u>EXAMPLE</u>

Now, we take the machine A shown in Fig. 3.2 as an example to illustrate the concept of trinity lattice.

		input		
present state	1	2	3	4
1	3/1	1/1	2/2	4/2
2	4/4	2/1	1/4	3/1
3	1/4	3/1	4/3	2/2
4	2/1	4/1	3/1	1/1

next state / output

Fig. 3.2 Machine A

By the computation on a computer Machine A has totally 24 PT's as follows:

π_I π_S π_O

$T_0 = (\{\overline{1},\overline{2},\overline{3},\overline{4}\},\{\overline{1},\overline{2},\overline{3},\overline{4}\},\{\overline{1},\overline{2},\overline{3},\overline{4}\})$

$T_1 = (\{\overline{1},\overline{2},\overline{3},\overline{4}\},\{\overline{1,3},\overline{2,4}\},\{\overline{1,4},\overline{2,3}\})$

$T_2 = (\{\overline{1},\overline{2},\overline{3},\overline{4}\},\{\overline{1,4},\overline{2,3}\},\{\overline{1,2},\overline{3,4}\})$

$T_3 = (\{\overline{1},\overline{2},\overline{3,4}\},\{\overline{1,3},\overline{2,4}\},\{\overline{1,4},\overline{2,3}\})$

$T_4 = (\{\overline{1,2},\overline{3},\overline{4}\},\{\overline{1,3},\overline{2,4}\},\{\overline{1,4},\overline{2,3}\})$

$T_5 = (\{\overline{1,3},\overline{2},\overline{4}\},\{\overline{1,4},\overline{2,3}\},\{\overline{1,2},\overline{3,4}\})$

$T_6 = (\{\overline{1},\overline{2,4},\overline{3}\},\{\overline{1,4},\overline{2,3}\},\{\overline{1,2},\overline{3,4}\})$

$T_7 = (\{\overline{1,2},\overline{3,4}\},\{\overline{1,3},\overline{2,4}\},\{\overline{1,4},\overline{2,3}\})$

$T_8 = (\{\overline{1,3},\overline{2,4}\},\{\overline{1,4},\overline{2,3}\},\{\overline{1,2},\overline{3,4}\})$

$T_9 = (\{\overline{1,2},\overline{3,4}\},\{\overline{1,2,3,4}\},\{\overline{1,2,3,4}\})$

$T_{10}= (\{\overline{1,3},\overline{2,4}\},\{\overline{1,2,3,4}\},\{\overline{1,2,3,4}\})$

$T_{11}= (\{\overline{1},\overline{2,3,4}\},\{\overline{1,2,3,4}\},\{\overline{1,2,3,4}\})$

$T_{12}= (\{\overline{2},\overline{1,3,4}\},\{\overline{1,2,3,4}\},\{\overline{1,2,3,4}\})$

$T_{13}= (\{\overline{3},\overline{1,2,4}\},\{\overline{1,2,3,4}\},\{\overline{1,2,3,4}\})$

$T_{14}= (\{\overline{1},\overline{2},\overline{3,4}\},\{\overline{1,2,3,4}\},\{\overline{1,2,3,4}\})$

$T_{15}= (\{\overline{1,2},\overline{3},\overline{4}\},\{\overline{1,2,3,4}\},\{\overline{1,2,3,4}\})$

$T_{16}= (\{\overline{1},\overline{2},\overline{3},\overline{4}\},\{\overline{1,2,3,4}\},\{\overline{1,2,3,4}\})$

$T_{17}= (\{\overline{1,3},\overline{2},\overline{4}\},\{\overline{1,2,3,4}\},\{\overline{1,2,3,4}\})$

$T_{18}= (\{\overline{1},\overline{3},\overline{2,4}\},\{\overline{1,2,3,4}\},\{\overline{1,2,3,4}\})$

$T_{19}= (\{\overline{4},\overline{1,2,3}\},\{\overline{1,2,3,4}\},\{\overline{1,2,3,4}\})$

$T_{20}= (\{\overline{1,4},\overline{2,3}\},\{\overline{1,2,3,4}\},\{\overline{1,2,3,4}\})$

$T_{21}= (\{\overline{1,4},\overline{2},\overline{3}\},\{\overline{1,2,3,4}\},\{\overline{1,2,3,4}\})$

$T_{22}= (\{\overline{1},\overline{4},\overline{2,3}\},\{\overline{1,2,3,4}\},\{\overline{1,2,3,4}\})$

$T_I = (\{\overline{1,2,3,4}\},\{\overline{1,2,3,4}\},\{\overline{1,2,3,4}\})$

The trinity lattice of machine A is depicted in Fig. 3.3

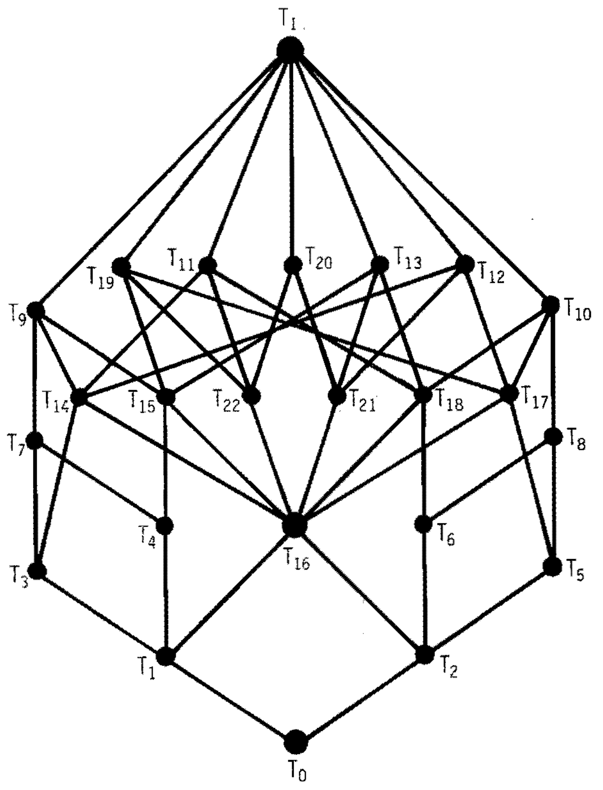


Fig. 3.3 Trinity lattice of machine A

From the lattice, we know that T_3 through T_8 are nontrivial trinities, T_7 and T_8 are basic nontrivial partition trinities, and the rest are trivial trinities. It is easily checked that

$T_3 \odot T_4 = T_1$, $T_3 \oplus T_4 = T_7$; $T_5 \odot T_6 = T_2$, $T_5 \oplus T_6 = T_8$;

and so on.

(End of Example)

Theorem 3.4 states that the operations $\odot$ and $\oplus$ on the set T are closed, which induces us to consider an important property on T given by the following definition.

DEFINITION 3.11

A *trinity algebra* is an algebraic system

$< T; \oplus, \odot; T_I, T_0>$

where T is the set of all partition trinities;

$\oplus$ and $\odot$ are trinity addition and multiplication;

T_I and T_0 are the identity trinity and zero trinity.

(End of Definition 3.11)

Thus, a trinity algebra is a binary relation on T which is closed under trinity operations of $\odot$ and $\oplus$ and contains all the elements such as $(\pi_I(I),\pi_S(I),\pi_O(I))$, and so on.

If we say that partition pairs characterize some transformation of the information that transpires in the operation of a machine, then, we can say that partition trinities characterize all the transformation of information that transpires in the operation of the machine. The property that $x\odot y$ is in T can be interpreted as "the combination of the information in X_S and Y_S or in X_I and Y_I is sufficient to compute the combined information X_S and Y_S or X_O and Y_O". Similarly, $x\oplus y$ states that "the combined ignorance in X_S and Y_S, or in X_I and Y_I, is sufficient to calculate the combined ignorance in X_S and Y_S, or in X_O and Y_O".

In view of the application to the full-decomposition of sequential machines and in view of other possible applications yet undiscovered, we will extract the common properties of trinity algebraic systems in order to derive the algebraic relationships in terms of these properties, in the rest of this section.

<u>THEOREM 3.6</u>

If (π_I,π_S,π_O) is in T, then

i) $\pi'_I \le \pi_I$ implies that (π'_I,π_S,π_O) is in T ;

ii) $\pi'_O \ge \pi_O$ implies that (π_I,π_S,π'_O) is in T ;

iii) $\pi'_I \le \pi_I$ and $\pi'_O \ge \pi_O$ imply that (π'_I,π_S,π'_O) is in T ;

Proof.

i) From Theorem 3.2 we know that
(π_I,π_S,π_O) is an PT implies that
(π_S,π_S) , (π_I,π_S), (π_S,π_O) and (π_I,π_O) are PP's.
By Lemma 3.2 $\pi'_I \le \pi_I$ implies that
(π'_I,π_S) and (π'_I,π_O) are PP's.
Combining them with (π_S,π_S) and (π_S,π_O) gives
that (π'_I,π_S,π_O) is an PT.
Hence, (π'_I,π_S,π_O) is in T.

ii) With the same argument as (i).

iii) For (π'_I,π_S,π'_O) using Theorem 3.6(i) and (ii) again.

(End of Theorem 3.6)

This theorem is useful for computing partition trinities too, since it presents another way of doing the computation.

<u>NOTATION</u> Let S be a set and π be a partition on S. For s and t in S, we write $[s]\pi = [t]\pi$ to denote that s and t are in the same block of π in the following discussions and chapters.
(end of Notation)

The theorem below shows the connection between relationships $\leq$ and operations $\odot$ and $\oplus$.

<u>THEOREM 3.7</u>

In algebraic system T, the multiplication and addition of two elements of T have the following property:

$x \leq y$ *if and only if* $x \odot y = x$ and $x \oplus y = y$.

This property is referred to as the *consistency property*.

Proof. Suppose that $x \leq y$ and

$x = (X_I, X_S, X_O)$ and $y = (Y_I, Y_S, Y_O)$.

$x \leq y$ implies that

$X_S \leq Y_S$, $X_I \leq Y_I$, and $X_O \leq Y_O$.

For $X_S \leq Y_S$, for any two states s and t in S,

$[s]x_S = [t]x_S$ implies $[s]Y_S = [t]Y_S$.

From the definition of partition multiplication and addition, the following relationships certainly exist:

$X_S \cdot Y_S = X_S$ and $X_S + Y_S = Y_S$.

Similarly, we have

$X_I \cdot Y_I = X_I$ and $X_I + Y_I = Y_I$,

$X_O \cdot Y_O = X_O$ and $X_O + Y_O = Y_O$.

That is,

$$\begin{aligned} x \odot y &= (X_I \cdot Y_I,\ X_S \cdot Y_S,\ X_O \cdot Y_O) \\ &= (X_I, X_S, X_O) \\ &= x, \\ x \oplus y &= (X_I + Y_I,\ X_S + Y_S,\ X_O + Y_O) \\ &= (Y_I, Y_S, Y_O) \\ &= y. \end{aligned}$$

Conversely, if $x \odot y = x$ and $x \oplus y = y$,
they mean, for any block B_X in X_S of x, there must exist a B_Y in Y_S of y, such that

$B_X \subseteq B_Y$.

It indicates that

$X_S \leq Y_S$.

With the same argument, we get

$$X_I \le Y_I \quad \text{and} \quad X_0 \le Y_0.$$

By Definition 3.5 we have

$$x \le y.$$

(End of Theorem 3.7)

Finally, some properties on the relationship ≤ and operations ⊙ and ⊕ are derived and given by the following theorem.

<u>THEOREM 3.8</u>

In the algebraic system T, the operations ⊙ and ⊕ for any two elements of T satisfy the idempotent, commutative, associative, and absorptive properties; that is, for any x,y and z in T,

i) *Idempotent* : $x \odot x = x$; $x \oplus x = x$

ii) *Commutative*: $x \odot y = y \odot x$; $x \oplus y = y \oplus x$

iii) *Associative*: $x \odot (y \odot z) = (x \odot y) \odot z$;
$x \oplus (y \oplus z) = (x \oplus y) \oplus z$

iv) *Absorptive* : $x \odot (x \oplus y) = x$; $x \oplus (x \odot y) = x$.

Proof. The properties (i) and (ii) follow directly from the definition of ⊙ and ⊕. The property (iii) is evident since $x \odot (y \odot z)$ and $(x \odot y) \odot z$ are both equal to the greatest lower bound of x,y and z, while $x \oplus (y \oplus z)$ and $(x \oplus y) \oplus z$ are both equal to the least upper bound of x,y an z.

To prove (iv), consider the following three cases:

(1) If $x \le y$, then, by Theorem 3.7, we have

$$x \odot (x \oplus y) = x \odot y$$
$$= x,$$

and

$$x \oplus (x \odot y) = x \oplus x$$
$$= x.$$

(2) If $x \ge y$, then, based on Theorem 3.7 again, we have

$$x \odot (x \oplus y) = x \odot x$$
$$= x;$$

and

$$x \oplus (x \odot y) = x \oplus y$$
$$= x.$$

(3) If $x \ngeq y$ and $y \ngeq x$,

for any $x,y \in T$, it is obvious that

$$x \oplus y \ge x . \qquad (1)$$

By Theorem 3.7, (1) implies

$$x \odot (x \oplus y) = x.$$

Similarly, we have

$$x \odot y \leq x . \qquad (2)$$

Theorem 3.7 shows that

$$x \oplus (x \odot y) = x.$$

(End of Theorem 3.8)

<u>THEOREM 3.9</u>

In the algebraic system T,

i) All elements satisfy the *isotone property*; that is,
if $x \leq y$, then $x \odot z \leq y \odot z$ and $x \oplus z \leq y \oplus z$.

ii) All elements satisfy the *modular inequality*, which is,
if $x \leq z$, then $x \oplus (y \odot z) \leq (x \oplus y) \odot z$.

iii) The *distributive inequalities* are satisfied:

$$x \odot (y \oplus z) \geq (x \odot y) \oplus (x \odot z),$$
$$x \oplus (y \odot z) \geq (x \oplus y) \odot (x \oplus z).$$

Proof. i) If $x \leq y$, then by Theorems 3.7 and 3.8

$$x \odot z = (x \odot y) \odot (z \odot z)$$
$$= (x \odot z) \odot (y \odot z).$$

Based on Theorem 3.7, it implies that

$$x \odot z \leq y \odot z.$$

The second inequality may be proved in a similar way.

ii) Since $x \leq z$ and $x \leq x \oplus y$,

$$x \leq (x \oplus y) \odot z$$

and since $y \odot z \leq z$ and $y \odot z \leq y \leq x \oplus y$

$$y \odot z \leq (x \oplus y) \odot z.$$

Combining these results and in view of the definition of $\oplus$, we obtain

$$x \oplus (y \odot z) \leq (x \oplus y) \odot z.$$

iii) Since $x \odot y \leq x$ and $x \odot y \leq y \leq y \oplus z$,

$$x \odot y \leq x \odot (y \oplus z).$$

From the relations $x \odot z \leq x$ and $x \odot z \leq z \leq y \oplus z$,

$$x \odot z \leq x \odot (y \oplus z).$$

Hence, $x \odot (y \oplus z) \geq (x \odot y) \oplus (x \odot z)$.

Again, the second inequality may be proved in a similar way.

(End of Theorem 3.9)

3.2 Homomorphism and Quotients

In this section, we study the relationships between two machines and those on a machine with respect to different partition trinities, which is the basic idea behind the full-decompositions which will be introduced later.

DEFINITION 3.12

Let $M = (I,S,O,\delta,\lambda)$ and $M' = (I',S',O',\delta',\lambda')$ be machines. If there exist three onto mappings

$\Phi: S \to S'$, $\Psi: I \to I'$ and $\Theta: O \to O'$

such that for any $s \in S$ and $i \in I$,

$$\Phi(s\delta_i) = \Phi(s)\delta'_{\Psi(i)}$$

and $$\Theta(s\lambda_i) = \Phi(s)\lambda'_{\Psi(i)}$$

then the triple (Φ,Ψ,Θ) is called a homomorphism from M to M′ and we write

$$(\Phi,\Psi,\Theta): M \to M'.$$

(End of Definition 3.12)

If (Φ,Ψ,Θ) is one-to-one, then, we call it a monomorphism, and if (Φ,Ψ,Θ) is onto, then, it is called an epimorphism. An isomorphism of machines is both a monomorphism and an epimorphism.

Under the mapping $\Phi: S \to S'$ there exists a partition on S, say π_S, defined by

$$[s]\pi_S = [t]\pi_S \Leftrightarrow \Phi(s) = \Phi(t).$$

For the same reason, we have two partitions, π_I and π_O, under mappings Ψ and Θ. Consequently, we obtain a tri-partition (π_I,π_S,π_O) on M. The tri-partition is called a tri-partition defined by the homomorphism (Φ,Ψ,Θ).

The idea of a partition trinity discussed in the last section leads to a procedure for constructing quotient systems in the following way.

DFINITION 3.13

Let $M = (I,S,O,\delta,\lambda)$ be a machine and $t = (\pi_I,\pi_S,\pi_O)$ be a partition trinity on M. The quotient machine

$$M/t = (X,Q,Y,\delta',\lambda')$$

of M with respect to t is defined by putting

$$Q = \pi_S \ , \ X = \pi_I \quad \text{and} \quad Y = \pi_O$$

and $\delta'(q,x) = q' \Leftrightarrow q\overline{\delta}_x \subseteq q' \in \pi_S$

$\lambda'(q,x) = y' \Leftrightarrow q\overline{\lambda}_x \subseteq y' \in \pi_O$.

for all $q \in Q$ and $x \in X$.

(End of Definition 3.13)

These definitions of δ' and λ' are well-defined since t is a partition trinity which preserves the functions of δ and λ. From Definitions 3.12 and 3.13 we easily get the following theorem which indicates the relationship between M and M/t.

THEOREM 3.10

Let t be a partition trinity on a machine $M = (I,S,O,\delta,\lambda)$. Then there exists a homomorphism

$$(\Phi,\Psi,\Theta): M \to M/t \ .$$

Proof.

Suppose that Φ is defined by that $\Phi(s)$ is the block which contains s and so is $\Psi(i)$. Since t is a trinity, for all $s \in S$ and $i \in I$,

$$s\delta_i \in \Phi(s)\delta'_{\Psi(i)} = \{s'\delta_{i'} \mid s' \in \Phi(s) \wedge i' \in \Psi(i)\}$$

Hence, $\Phi(s\delta_i) = \Phi(s)\delta'_{\Psi(i)}$. With the same argument

we can prove that $\Theta(s\lambda_i) = \Phi(s)\lambda'_{\Psi(i)}$.

(End of Theorem 3.10)

The homomorphism (Φ,Ψ,Θ) is also called the natural epimorphism defined by t, because, for any $q \in Q$, $x \in X$ and $y \in Y$, there at least exists a triple of $s \in S$, $i \in I$ and $z \in O$ such that $\Phi(s) = q$, $\Psi(i) = x$ and $\Theta(z) = y$.

Some remarks concerning the relationships between two quotient machines over the same machine M are worth making.

Suppose that t and t′ are two partition trinities on machine $M = (I,S,O,\delta,\lambda)$. If $t \leq t'$, we can construct an epimorphism from M/t to M/t′. This leads us to a homomorphism theorem for the machines.

THEOREM 3.11

Let M and M′ be machines and

$$(\Phi,\Psi,\Theta):\ M \to M'$$

be an epimorphism. If t′ defined by (Φ,Ψ,Θ) is an PT on M and t is an PT on M satisfying the condition $t \leq t'$, then, there exists an epimorphism

$$(\Phi'',\Psi'',\Theta''):\ M/t \to M'$$

such that

$$(\Phi,\Psi,\Theta) = (\Phi',\Psi',\Theta')\circ(\Phi'',\Psi'',\Theta'')$$
$$= (\Phi'\circ\Phi'',\Psi'\circ\Psi'',\Theta'\circ\Theta'')$$

where $(\Phi',\Psi',\Theta'):\ M \to M/t$ and $\circ$ denotes function composition. Furthermore if $t = t'$ then (Φ'',Ψ'',Θ'') is an isomorphism.

Proof.

Let $t = (\pi_I,\pi_S,\pi_O)$ and $t' = (\pi'_I,\pi'_S,\pi'_O)$.

We define

$$\begin{aligned}
&\Phi'':\ \pi_S \to S'' \text{ by } \Phi''(B) = \Phi(s) \text{ where } s\in B\in\pi_S,\\
&\Psi'':\ \pi_I \to I'' \text{ by } \Psi''(C) = \Psi(i) \text{ where } i\in C\in\pi_I, \qquad (1)\\
&\Theta'':\ \pi_O \to O'' \text{ by } \Theta''(D) = \Theta(y) \text{ where } y\in D\in\pi_O.
\end{aligned}$$

These are well-defined for if $s'\in B$ then there exists a B' in π'_S such that

$$s,s'\in B \subseteq B' \quad\text{and}\quad \Phi(s) = \Phi(s') \qquad \{t \leq t'\}$$

and so are C and D.

For any $B\in\pi_S$ and $C\in\pi_I$,

$$\begin{aligned}
&\Phi''(B\delta'_C)\\
&= \Phi(s'\in B\delta'_C) && \{(1)\}\\
&= \Phi(s\delta_i) && \{s\in B \wedge i\in C)\}\\
&= \Phi(s)\delta''_{\Psi(i)} && \{(\Phi,\Psi,\Theta)\}\\
&= \Phi''(B)\delta''_{\Psi''(C)} && \{(1)\}
\end{aligned}$$

By the similar way we have

$$\Theta''(B\lambda'_c) = \Phi''(B)\lambda''_{\Psi(c)}$$

It is implied that $(\Phi'', \Psi'', \Theta'')$ is an epimorphism.
Secondly, to show communitive homomorphisms,

$\Theta(s\lambda_i)$

$= \Phi(s)\lambda''_{\Psi(i)}$ $\{(\Phi, \Psi, \Theta)\}$

$= \Phi''(B)\lambda''_{\Psi''(c)}$ $\{(1)\}$

$= \Phi''(\Phi'(s))\lambda''_{\Psi''(\Psi'(i))}$ $\{(\Phi', \Psi', \Theta')\}$

$= (\Phi' \circ \Phi'')(s)\lambda''_{(\Psi' \circ \Psi'')(i)}$ {functional composition}

With the same procedure we have

$$\Phi(s\delta_i) = (\Phi' \circ \Phi'')(s)\delta''_{(\Psi' \circ \Psi'')(i)}$$

Hence, the theorem.
Furthermore, if $t = t'$, $(\Phi'', \Psi'', \Theta'')$ becomes one-to-one.
Therefore, it is isomorphic.
(End of Theorem 3.11)

The theorem is also illustrated by the following diagram which shows the communitive property of the homomorphisms.

$$\begin{tikzcd}
 & M \arrow[dl, "{(\Phi', \Psi', \Theta')}"'] \arrow[dr, "{(\Phi, \Psi, \Theta)}"] & \\
M/t \arrow[rr, "{(\Phi'', \Psi'', \Theta'')}"'] & & M'
\end{tikzcd}$$

In the theorem, if $t \geq t'$, it is easy to show that $(\Phi'', \Psi'', \Theta'')$ is in the opposite direction, that is,

$$(\Phi'', \Psi'', \Theta''): M' \rightarrow M/t$$

with the same statements. This is included in the theorem if we consider M' as M/t, and therefore, it is omitted.

3.3 Computation of Partition Trinity Lattice

For applications of partition trinity theory, the first thing is to compute a PT or a PT lattice for a given machine. In this section, we discuss the ways of computing an PT and an PT lattice using the properties given in the last section.

3.3.1 Compute Nontrivial PT's

From the definition we know that the direct method for computing partition trinity is to calculate all the partition pairs of S-S, I-O, S-O and I-O for a machine. Then, compare them and find some partition trinity. But, experiments show that it takes a very long computation, because of the very large number of pairs. From the experiments and examples, we found that the difference between the numbers of partition pairs of different types of pairs was very great. Usually, the number of partition pairs of S-S and S-O were great, while the ones of I-S and I-O were small, because of the structural characteristics of sequential machines. The procedure below gives one of the ways to compute an PT based on the above consideration.

<u>PROCEDURE 3.1</u>

```
    1. Find a nontrivial I-S pair $(\pi_I,\pi_S)$;
    2. If $(\pi_S,\pi_S)$ is not an S-S pair, then go to step 1;
    3. Find an output nontrivial partition $\pi_O$ from $\pi_S$;
    4. If $(\pi_S,\pi_O)$ is not an S-O pair, go to step 1;
    5. If $(\pi_I,\pi_O)$ is not an I-O pair, go to step 1;
    6. $(\pi_I,\pi_S,\pi_O)$ is a nontrivial PT;
    7. Exit.
```

(End of Procedure 3.1)

In Procedure 3.1, because of trial and error, the computation of one pair may take longer in step 1. An alternative way is given by Procedure 3.2 below.

PROCEDURE 3.2

```
1. Compute the set of second components of all the smallest
   S-O pairs ;
2. For any two elements in the set carry out partition addition on
   them; the result is a new output partition that can be used to
   construct an S-O pair with some state partition; after this
   step, a set of all output partitions which are the second
   components of the same S-O pairs;
3. If $\pi_O$ is in the set, compute $M_{S-O}(\pi_O) = \pi_S$;
4. If $(\pi_S, \pi_S)$ is not an S-S pair, go to step 3;
5. For $\pi_S$, compute $M_{I-S}(\pi_S) = \pi_I$;
6. If $(\pi_I, \pi_S)$ is not an I-S pair, go to step 3;
7. If $(\pi_I, \pi_O)$ is not an I-O pair, go to step 3;
8. $(\pi_I, \pi_S, \pi_O)$ is an PT;
9. For all $\pi_O$ in the set, repeat steps 3-8;
```

where $M_{I-S}(\pi_S)$ and $M_{S-O}(\pi_O)$ are two pair operations and are defined by

$$M_{I-S}(\pi_S) = \sum \{\pi'_I \,|\, (\pi'_I, \pi_S) \text{ is an I-S pair}\}$$

$$M_{S-O}(\pi_O) = \sum \{\pi'_S \,|\, (\pi'_S, \pi_O) \text{ is an I-S pair}\}$$

(End of Procedure 3.2)

Another way is suggested by Procedure 3.3. In this procedure, we first compute the SP partitions. This is because we know that SP is the nearest to PT from the inclusion relation diagram in Fig. 3.1, and it will take less time to compute. The procedure also gained by the fact that the number of SP partitions is far smaller than that of all S-S partitions on a machine. Hence, we do not need to compute the pairs of $\{(\pi, \tau)\}$ in which $\pi \neq \tau$.

PROCEDURE 3.3

```
1. Compute all the SP partitions, that is,
       $\{\tau_S \,|\, \tau_S$ is an SP partition$\}$;
2. If $\pi_S \in \{\tau_S\}$, then calculate $\pi_O = m_{O-S}(\pi_S)$;
       if $m_{S-O}(\pi_S) = \pi_O(0)$ or $m_{S-O}(\pi_S) = \pi_O(I)$,
       then go to step 2;
3. Calculate $\pi_I = M_{I-S}(\pi_S)$,
       if $M_{I-S}(\pi_S) = \pi_I(0)$ or $M_{I-S}(\pi_S) = \pi_I(I)$ ,
       then go to step 1;
```

4. If (π_I,π_O) is an I-O pair, then
   (π_I,π_S,π_O) is a basic nontrivial PT,
   otherwise, go to step 2;
5. For all π_S in $\{\tau_S\}$, repeat steps 2-4,

where $m_{S-O}(\pi_S)$ is a pair operation and is defined by

$$m_{S-O}(\pi_S) = \prod \{\pi_O' \mid (\pi_S,\pi_O') \text{ is an S-O pair.}\}$$

(End of Procedure 3.3)

It should be stated that pair operations $M(\pi)$ and $m(\pi)$ are done by a direct method from the transition table on a computer instead of by the definitions of them.

3.3.2 Compute PT Lattice

In this section, we present the general procedure for constructing an PT lattice of a given sequential machine.

<u>PROCEDURE 3.4</u>

1. Compute the set $\{T_b\}$ of all basic nontrivial PT's;
2. For any $x,y\in\{T_b\}$, perform operations $\odot$ and $\oplus$ on them;
   if $x\odot y$ or $x\oplus y$ is a nontrivial PT, put it in $\{T_b\}$;
3. For $z\in\{T_b\}$, $z = (Z_I,Z_S,Z_O)$,
   using Theorem 3.6 for Z_I and Z_O, we get two sets.

   $\{Z_I' \mid Z_I'\le Z_I\}$ and $\{Z_O' \mid Z_O'\le Z_O\}$;
4. $\{(\{Z_I'\}\times Z_S\times\{Z_O'\})\}$ gives a set of PT's which are derived

   from basic PT z;
5. For all $z\in\{T_b\}$, repeat steps 3 and 4;
6. Set up a table in which the rows and columns are PT's;
   for a row x and a column y, if $x\le y$ (or $x\ge y$), then
   put the sign of $\le$ (or $\ge$) on the cross entry of x and y;
   the table is referred to as an "R table";
7. Using the R table, join all PT's together in order to draw
   a lattice diagram.

(End of Procedure 3.4)

CHAPTER 4

PARALLEL FULL-DECOMPOSITIONS

In the preceding chapter the concept of a partition trinity was presented and trinity algebra was discussed systematically. The results developed there will be used in this chapter and following chapters in order to study the full-decompositions of sequential machines. Before we deal with the parallel full-decomposition, we have to make a rule for the relationship between the original machine and a simple network of component machines, which is described by the concept of realization.

4.1 Relationships between Machines

In this section, we consider the relationship between two machines, which will serve as a basis for the decompositions throughout this thesis.

Let $M = (I,S,O,\delta,\lambda)$
and $M' = (I',S',O',\delta',\lambda')$
be two machines with the same type.

<u>DEFINITION 4.1</u>

Machines M and M′ are isomorphic if and only if there exist three one-to-one onto mappings

$\alpha: S \to S'$

$\beta: I \to I'$

$\gamma: O \to O'$

such that

$$\alpha(s\delta_x) = \alpha(s)\delta'_{\beta(x)}$$

and $$\gamma(s\lambda_x) = \alpha(s)\lambda'_{\beta(x)}.$$

(End of Definition 4.1)

We refer to the triple (α,β,γ) of mappings as an isomorphism between M and M'.

The definition states that two sequential machines are isomorphic if and only if they are identical except for a renaming of the states, inputs, and outputs. Machine isomorphism is the most elementary case of two machines imitating each other through the use of combinational circuits, in order to perform the three mappings. If we have a machine M' which is isomorphic to M, then by just placing a combinational circuit in front of the machine M' mapping inputs, and one at the rear of the machine for mapping outputs, and/or one to one side of the machine for mapping states in the case of observing states or of state machines, we can convert it into a machine which behaves like M. The schematic representation of this conversion of M' into M, using three combinational circuits, is shown in Fig. 4.1, where, triangles are combinational circuits and indicate the directions of mappings.

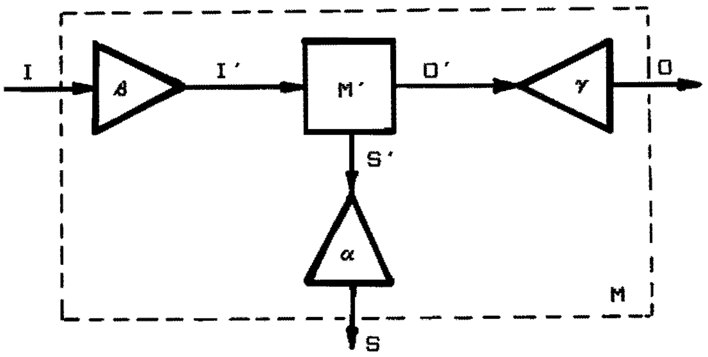


Fig. 4.1
Machine M is simulated by its isomorphic machine M' with combinational circuits.

In the above definition, we defined three one-to-one onto mappings. If we omit the condition of one-to-one, a more general concept is obtained, which has been briefly mentioned in Chapter 3.

Let μ: $S \to S'$, ν: $I \to I'$ and ω: $O \to O'$ be three onto mappings from M to M'. If they satisfy that,
for all s in S and x in I,

$$\mu(s\delta_x) = \mu(s)\delta'_{\nu(x)}$$

and $$\omega(s\lambda_x) = \mu(s)\lambda'_{\nu(x)}$$

then, machines M and M' are said to be homomorphic and M' is said to be a *homomorphic image* of machine M. By the definition in Chapter 3, it means

$$(\mu,\nu,\omega) : M \to M'.$$

Again, we can simulate a machine, M', by another machine, M, with some combinational circuits, if M' is a homomorphic image of M. The schematic representation of this simulation is shown in Fig. 4.2. If ν does not have a unique inverse, then $\nu^{-1}(x)$ is interpreted as any input symbol which is mapped onto x' by ν. Intuitively speaking, the machine M does more than M' can, but it can be modified by attaching combinational circuits in order to imitate its homomorphic image M'.

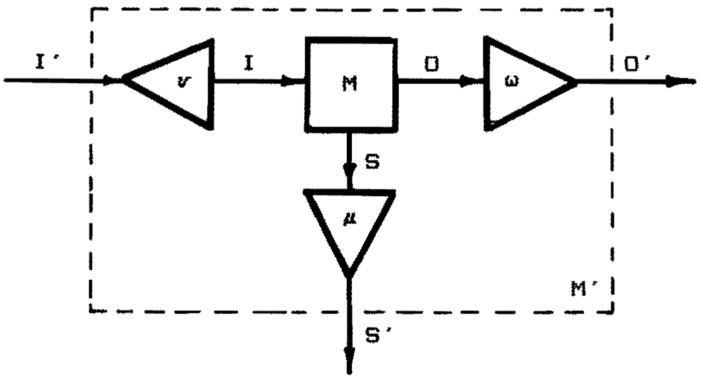


Fig. 4.2
Simulation of the homomorphic image M' of M.

In addition to the isomorphic and homomorphic relations, in practice, we prefer the case of how a machine M' can be used to imitate the behaviour or functions of M. For this, in [22], this was referred to as realization, and in [15,22] it was defined by the concept of covering.

The former is emphasized by the mappings that make M′ behave like M, but the latter concerned M′ producing the same output sequence as M did.

In many applications, we are concerned with not only the outputs of a machine but also with the state changes, of the machine; therefore, we think that realization is suitable in our situations.

A realization is defined as follows. M′ is a realization of M if there exist three mappings: Φ is a mapping of S into nonvoid subsets of S′; Ψ is a mapping of I into I′; and Θ is a mapping of O′ into O, such that (Φ,Ψ,Θ) preserve the properties and binary operations. This definition is not too convenient in practice. The reasons for it are twofold. One is that, in a physical implementation we cannot directly get the combinational circuit designs for some mappings, such as ω. We must calculate $\Phi(-1)$ first. Another reason is that we cannot make the definition coincidential with that for state machines. In the following definition, some improvements will be made.

<u>DEFINITION 4.2</u>

A machine M′ is said to be a realization of machine M if and only if there exist three relations

$\Phi: S' \to S$ is a surjective partial function

$\Psi: I \to I$ is a function

$\Theta: O' \to O$ is a surjective partial function

such that

$$\Phi(s')\delta_x = \Phi(s'\delta'_{\Psi(x)})$$

and $$\Phi(s')\lambda_x = \Theta(s'\lambda'_{\Psi(x)}).$$

(End of Definition 4.2)

We denote the realization by: $M \lhd M'$ and illustrate it diagramatically in Fig. 4.3.

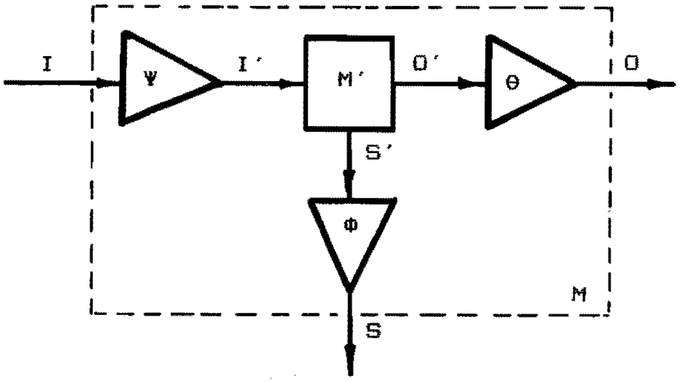


Fig. 4.3 $M \lhd M'$

Fig. 4.3 states that if M′ is a realization of M, then M′ started in a state s′ behaves like M under the interpretation of Q and h when started in Φ(s′), if we consider Φ, Ψ and Θ as three interpretors. In other words, that M′ realizes M means that we can put three combinational circuits of Ψ, Φ and Θ by which M′ works exactly like M under the translations on the inputs, states and outputs of M′.

It should be mentioned here that if M′ realizes M, then the two machines do not necessarily have to be isomorphic or related by homomorphism. There is though, a homomorphism which relates M′ to the reduced machine equivalent to M in the case when Ψ is a one-to-one mapping, as shown in [15].

4.2 Parallel Full-decompositions

In Chapter 2 we have described some meanings of parallel full-decompositions for sequential machines. In this section, we are going to discuss them in detail. A parallel full-decomposition is such a decomposition that the original machine M is decomposed into two component machines M′ and M″ each of them working independently and having fewer states, inputs and outputs. Before studying this decomposition, we make a precise definition of the parallel connection of machines.

<u>DEFINITION 4.3</u>

A parallel connection of two machines

$M' = (I', S', O', \delta', \lambda')$

$M'' = (I'', S'', O'', \delta'', \lambda'')$

is the machine

$M = M' \parallel M'' = (I' \times I'', S' \times S'', O' \times O'', \delta^*, \lambda^*)$

its transition function δ^* and output function λ^* defined by

$$(s', s'')\delta^*_{(x', x'')} = (s'\delta'_{x'},\ s''\delta_{x''})$$

$$(s', s'')\lambda^*_{(x', x'')} = (s'\lambda'_{x'},\ s''\lambda''_{x''})$$

where $s' \in S'$, $x' \in I'$, $s'' \in S''$ and $x'' \in I''$.

(End of Definition 4.3)

<u>DEFINITION 4.4</u>

Machines M′ and M″ are said to be a parallel full-decomposition of $M = (I, S, O, \delta, \lambda)$ *if and only if*

$M \lhd M' \parallel M''$.

(End of Definition 4.4)

THEOREM 4.1

Let $M = (I,S,O,\delta,\lambda)$ and suppose that t' and t'' are two partition trinities on M. If both t' and t'' are non-trivial and orthogonal, namely, $t' \odot t'' = T_0$, then,

$$M \lhd M/t' \parallel M/t''.$$

Proof.

Let $M/t' = M'$ and $M/t'' = M''$

with $t' = (\pi_I, \pi_S, \pi_O)$ and $t'' = (\tau_I, \tau_S, \tau_O)$.

Thus,

$$M' = (\pi_I, \pi_S, \pi_O, \delta', \lambda')$$
$$M'' = (\tau_I, \tau_S, \tau_O, \delta'', \lambda''),$$

where $B'\delta'_{\mathcal{B}'} = B'\overline{\delta}_{\mathcal{B}'}$ and $B'\lambda'_{\mathcal{B}'} = B'\overline{\lambda}_{\mathcal{B}'}$,

and $B''\delta''_{\mathcal{B}''} = B''\overline{\delta}_{\mathcal{B}''}$ and $B''\lambda''_{\mathcal{B}''} = B''\overline{\lambda}_{\mathcal{B}''}$,

for all $B' \in \pi_S$, $\mathcal{B}' \in \pi_I$, $B'' \in \tau_S$, $\mathcal{B}'' \in \tau_I$.

From Definition 4.3,

$$M' \parallel M'' = (\pi_I \times \tau_I, \pi_S \times \tau_S, \pi_O \times \tau_O, \delta^*, \lambda^*)$$

where $(B', B'')\delta^*_{(\mathcal{B}', \mathcal{B}'')} = (B'\delta'_{\mathcal{B}'}, B''\delta''_{\mathcal{B}''})$,

and $(B', B'')\lambda^*_{(\mathcal{B}', \mathcal{B}'')} = (B'\lambda'_{\mathcal{B}'}, B''\lambda''_{\mathcal{B}''})$,

for $B' \in \pi_S$, $B'' \in \tau_S$; $\mathcal{B}' \in \pi_I$, $\mathcal{B}'' \in \tau_I$.

Let $\Psi: I \rightarrow \pi_I \times \tau_I$ be defined by

$\Psi(x) = (\mathcal{B}', \mathcal{B}'')$ such that

$\mathcal{B}' \in \pi_I$, $\mathcal{B}'' \in \tau_I$, $\mathcal{B}' \cap \mathcal{B}'' = x$;

$\Phi: \pi_S \times \tau_S \rightarrow S$ be defined by

$\Phi(B', B'') = s$ such that

$B' \in \pi_S$, $B'' \in \tau_S$, $B' \cap B'' = s$;

$\Theta: \pi_O \times \tau_O \rightarrow O$ be defined by

$\Theta(z', z'') = z$ such that

$z' \in \pi_O$, $z'' \in \tau_O$, $z' \cap z'' = z \in O$.

Since $t' \odot t'' = T_0$, Θ is an injective function.

Φ and Θ are two surjective partial functions.

For each $(B', B'') \in \pi_S \times \tau_S$, $B' \cap B'' \neq \emptyset$ and $x \in I$,

$\Phi(B',B'')\delta_x$

$= s\delta_x$ {let $B'\cap B''=s$}

$= (B'\cap B'')\delta_x$ {calculus}

$\in B'\overline{\delta}_x \cap B''\overline{\delta}_x$ {$B'\cap B''\neq\emptyset$}

$\subseteq [B'\overline{\delta}_x]\pi_s \cap [B''\overline{\delta}_x]\tau_s$ {calculus}

$= B'\delta'_{\Psi(\cdot x)} \cap B''\delta''_{\Psi(x\cdot)}$ {M' and M''}

$= \Phi(B'\delta'_{\Psi(\cdot x)}, B''\delta''_{\Psi(x\cdot)})$ {defination of Φ}

$= \Phi(B',B'')\delta^*_{\Psi(x)}$ {defination of $M' \parallel M''$}

where $\Psi(\cdot x)$ denotes the first component and $\Psi(x\cdot)$ the second one of $\Psi(x)$, namely, $\Psi(x)=(\Psi(\cdot x),\Psi(x\cdot))$.

Since there certainly exist an $A'\in\pi_s$ and $A''\in\tau_s$ such that $[B'\overline{\delta}_x]\pi_s = A'$ and $[B''\overline{\delta}_x]\tau_s = A''$ and $|A'\cap A''| = 1$ indeed from $\pi_s\cdot\tau_s=\pi_s(0)$, in the sequence, it should be true that

$(B'\cap B'')\delta_x = B'\overline{\delta}_x\cap B''\overline{\delta}_x = [B'\overline{\delta}_x]\pi_s\cap[B''\overline{\delta}_x]\tau_s$

Thus,

$$\Phi(B',B'')\delta_x = \Phi((B',B'')\delta^*_{\Psi(x)})$$

Similarly,

$\Phi(B',B'')\lambda_x$

$= s\lambda_x$ {let $\Phi(B',B'')=s$}

$= (B'\cap B'')\lambda_x$ {$B'\cap B''=s$}

$= B'\overline{\lambda}_x\cap B''\overline{\lambda}_x$ {$B'\cap B''\neq\emptyset$, $\pi_o\cdot\tau_o= \pi_o(0)$}

$\subseteq [B'\overline{\lambda}_x]\pi_o\cap[B''\overline{\lambda}_x]\tau_o$ {$\pi_o\cdot\tau_o=\pi_o(0)$}

$= B'\lambda'_{\Psi(\cdot x)} \cap B''\lambda''_{\Psi(x\cdot)}$ {M' and M''}

$= \Theta(B'\lambda'_{\Psi(\cdot x)}, B''\lambda''_{\Psi(x\cdot)})$ {defination of Θ}

$= \Theta(B',B'')\lambda^*_{\Psi(x)}$ {defination of $M' \parallel M''$}

That is,

$$\Phi(B',B'')\lambda_x = \Theta((B',B'')\lambda^*_{\Psi(x)})$$

By Definition 4.3 we know

$$M \lhd M' \parallel M'' = M/t' \parallel M/t''$$

(End of Theorem 4.1)

Let us use an example to illustrate this theorem.

<u>**EXAMPLE 4.1**</u>

With Theorem 4.1 find a parallel full-decomposition, if it exists, for the machine shown in Fig. 4.4.

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

Fig. 4.4 Machine B

Calculating with a computer shows that trinities

$$t' = (\{\overline{1,5},\ \overline{2,4},\ \overline{3},\ \overline{6}\},$$
$$\{\overline{1,4,7},\ \overline{2,3,6},\ \overline{5,8}\},$$
$$\{\overline{1,2,3},\ \overline{4,5}\})$$
$$t'' = (\{\overline{1,4},\ \overline{2,3},\ \overline{5},\ \overline{6}\},$$
$$\{\overline{1,5,6},\ \overline{2,4},\ \overline{3,7,8}\},$$
$$\{\overline{1,5},\ \overline{2,4},\ \overline{3}\})$$

are orthogonal. Therefore, we use them to build the quotient machines B/t′ and B/t″. The quotient machine B/t′ is formed in Fig. 4.5 by making the following short notations:

$$\pi_I = \{\overline{1,5},\overline{2,4},\overline{3},\overline{6}\} = \{\beta_1,\beta_2,\beta_3,\beta_4\}$$
$$\pi_S = \{\overline{1,4,7},\overline{2,3,6},\overline{5,8}\} = \{\alpha_1,\alpha_2,\alpha_3\}$$
$$\pi_O = \{\overline{1,2,3},\overline{4,5}\} = \{\gamma_1,\gamma_2\}$$

In the same way, the quotient machine B/t″ is formed in Fig. 4.6 with the following short notations.

$$\tau_I = \{\overline{1,4},\overline{2,3},\overline{5},\overline{6}\} = \{y_1,y_2,y_3,y_4\}$$
$$\tau_S = \{\overline{1,5,6},\overline{2,4},\overline{3,8,7}\} = \{x_1,x_2,x_3\}$$
$$\tau_O = \{\overline{1,5},\overline{2,4},\overline{3}\} = \{z_1,z_2,z_3\}$$

	β_1	β_2	β_3	β_4
α_1	α_3/γ_2	α_1/γ_2	α_2/γ_2	α_3/γ_1
α_2	α_2/γ_1	α_1/γ_1	α_2/γ_1	α_1/γ_1
α_3	α_2/γ_2	α_2/γ_2	α_2/γ_2	α_1/γ_1

Fig. 4.5 Quotient machine B/t′

	y_1	y_2	y_3	y_4
x_1	x_1/z_2	x_2/z_1	x_3/z_1	x_1/z_3
x_2	x_3/z_2	x_1/z_2	x_3/z_2	x_3/z_2
x_3	x_1/z_1	x_3/z_1	x_1/z_1	x_1/z_3

Fig. 4.6 Quotient machine B/t″

If we make the following notations between machine B and B/t′ ∥ B/t″:

$\Psi: I \to \pi_I \times \tau_I$ $\Phi: \pi_S \times \tau_S \to S$ $\Theta: \pi_O \times \tau_O \to O$

are defined by,

for all $x \in I$; $(B', B'') \in \pi_S \times \tau_S$; $(z', z'') \in \pi_O \times \tau_O$

x	$\Psi(x)$	(B', B'')	$\Phi(B', B'')$	(z', z'')	$\Theta(z', z'')$
1	(b_1, y_1)	(α_1, x_1)	1	(γ_1, z_1)	1
2	(b_2, y_2)	(α_2, x_2)	2	(γ_1, z_2)	2
3	(b_3, y_2)	(α_2, x_3)	3	(γ_1, z_3)	3
4	(b_2, y_1)	(α_1, x_2)	4	(γ_2, z_2)	4
5	(b_1, y_3)	(α_3, x_1)	5	(γ_2, z_1)	5
6	(b_4, y_4)	(α_2, x_1)	6		
		(α_1, x_3)	7		
		(α_3, x_3)	8		

It is obvious that Ψ is an injective function and both Φ and Θ are surjective partial functions. By the definition we have

$B \lhd B/t' \parallel B/t''$

For example, let $(\alpha_3, x_3) \in \pi_S \times \tau_S$ be a present state in B/t′ ∥ B/t″; with the input $6 \in I$, $\Psi(6) = (\beta_4, y_1)$, the B/t′ ∥ B/t″ goes to

$$(\alpha_1, x_3)\delta^*_{\Psi(6)} = (\alpha_3, x_3)\delta^*_{(\beta_4, y_4)}$$

$$= (\alpha_3\delta'_{\beta_4}, x_3\delta''_{y_4})$$

$$= (\alpha_1, x_1)$$

$$\Phi((\alpha_3, x_3)\delta^*_{\Psi(6)}) = \Phi(\alpha_1, x_1) = 1$$

On the other hand, $\Phi(\alpha_2, x_3) = 8$

$$\Phi(\alpha_3, x_3)\delta_6 = 8\delta_6 = 1.$$

Therefore, $\Phi((\alpha_3, x_3)\delta^*_{\Psi(6)}) = \Phi(\alpha_3, x_3)\delta_6 = 1.$

A schematic representation for the full-decomposition of machine B is given in Fig. 4.7.

(End of Example 4.1)

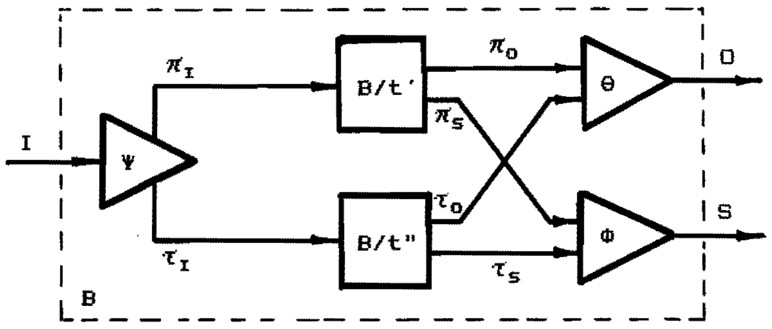


Fig. 4.7 B ⊲ B/t' ‖ B/t"

From Theorem 4.1, we can obtain a parallel full-decomposition M/t' ‖ M/t" which realizes the original a machine M. It should be noted that sometimes M/t' ‖ M/t" may be isomorphic to M. Here, we will study this special case of the theorem.

Firstly, we define some partitions and trinities which are permutable.

DEFINITION 4.5

Let S be a set and π and τ be partitions on S. The partitions π and τ are said to be permutable *if and only if*

$$\forall B' \in \pi \quad \forall B'' \in \tau: \quad |B' \cap B''| = 1$$

(End of Definition 4.5)

Thus, if π and τ are permutable, then any elements in a block of π are one permutation over all blocks of τ, and vice versa. For example,

let S = {1,2,3,4,5,6}. $\pi = \{\overline{1,3,6}, \overline{2,4,5}\}$ and $\tau = \{\overline{1,4}, \overline{2,3}, \overline{5,6}\}$

are permutatable. Obvious examples of permutable partitions are the trivial partitions: zero partition and identity partition.
For a pair of permutable partitions, we get the following property.

THEOREM 4.2

If π and τ are permutable partitions on S, then

i) $\pi \cdot \tau = \pi_S(O)$;

ii) $\pi + \tau = \pi_S(I)$.

Proof.

i) Since $|B' \cap B''| = 1$, any block B in $\pi \cdot \tau$ is a singleton. From the definition, $\pi \cdot \tau$ is a zero partition.

ii) Because any block B′ in π contains exactly an element of every block B″ in τ, the block in $\pi + \tau$ contains all elements of all blocks in π or τ.
Hence, $\pi + \tau$ is an identity partition.

(End of Theorem 4.2)

Partitions π and τ are called *complemeenary*, if they satisfy $\pi \cdot \tau = \pi_S(0)$ and $\pi + \tau = \pi_S(I)$. From the theorem, if π and τ are permutable, then, they are complementary. However, conversely, that π and τ are complementary does not imply that π and τ are necessarily permutable. For instance, if we change τ into

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

then, π and τ still are complementary, but they are not permutable.

We can extend the concept of 'permutable' to partition trinities.

<u>DEFINITION 4.6</u>

Let $t' = (\pi_I, \pi_S, \pi_O)$ and $t'' = (\tau_I, \tau_S, \tau_O)$ be two trinities on machine M. t′ and t″ are permutable *if and only if* π_I and τ_I, π_S and τ_S, and π_O and τ_O are permutable, respectively.

(End of Definition 4.6)

In the last part of this section, we will apply the concept of "permutable partition trinities" to test the isomorphic relation between a machine and its parallel full-decomposition.

<u>THEOREM 4.3</u>

A machine M is isomorphic to the parallel connection of two quotient machines M/t′ and M/t″ if t′ and t″ are permutable partition trinities.

Proof. From Theorems 4.1 and 4.2, we know that M/t′ ‖ M/t″ realizes M. Since t′ and t″ are permutable, there is no pair of states B′ in M/t′ and B″ in M/t″ which are disjoint. So are the pairs of inputs and outputs. It implies that the mappings of the triple (Φ, Ψ, Θ) are one-to-one. Hence the theorem.

(End of Theorem 4.3)

Again, we can take an example to interpret this theorem.

EXAMPLE 4.2

For the machine C shown in Fig. 4.8, a computer shows the following partition trinities.

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

Fig. 4.8 Machine C

$$t_1 = (\{\overline{1,3},\overline{2,4}\},\ \{\overline{1,5},\overline{2,6},\overline{3,7},\overline{4,8}\},\ \{\overline{1,6},\overline{2,5},\overline{3,8},\overline{4,7}\})$$

$$t_2 = (\{\overline{1,3},\overline{2,4}\},\ \{\overline{1,5},\overline{2,4,6,8},\overline{3,7}\},\ \{\overline{1,2,4,5,6,7},\overline{3,8}\})$$

$$t_3 = (\{\overline{1,3},\overline{2,4}\},\ \{\overline{1,3,5,7},\overline{2,6},\overline{4,8}\},\ \{\overline{1,2,3,5,6,8},\overline{4,7}\})$$

$$t_4 = (\{\overline{1,4},\overline{2,3}\},\ \{\overline{1,3,6,8},\overline{2,4,5,7}\},\ \{\overline{1,3,5,7},\overline{2,4,6,8}\})$$

$$t_5 = (\{\overline{1,3},\overline{2,4}\},\ \{\overline{1,2,5,6},\overline{3,4,7,8}\},\ \{\overline{1,2,5,6},\overline{3,4,7,8}\})$$

Inspecting the trinities, by using the definition of permutable, we get two partition trinities, t_4 and t_1, which are permutable and can be used for the isomorphic full-decomposition of machine C.

Now, we make substitutions on t_4 and t_1 and present the quotient machines in Fig. 4.9.

t_4 = $(\{i_1,i_2\}, \{s_1,s_2\}, \{y_1,y_2\})$

t_1 = $(\{j_1,j_2\}, \{q_1,q_2,q_3,q_4\}, \{z_1,z_2,z_3,z_4\})$

(End of Example 4.2)

	i_1	i_2
s_1	s_1/y_1	s_2/y_2
s_2	s_2/y_2	s_1/y_1

C/t_4

	j_1	j_2
q_1	q_1/z_1	q_2/z_3
q_2	q_2/z_2	q_1/z_4
q_3	q_3/z_3	q_2/z_2
q_4	q_4/z_4	q_1/z_1

C/t_1

Fig. 4.9 Quotient machines of C

Generally speaking, if a machine M is fully decomposible, such as M M/t' ‖ M/t"; then we can encode the input information in a binary code of N'+N" digits so that the component machine M/t' will operate only with the first digits and another component machine M/t" will operate only with the last N" digits. N' and N" can be calculated as follows

$$N' = \lceil \log_2 |\pi_I| \rceil$$

$$N'' = \lceil \log_2 |\tau_I| \rceil$$

where $\lceil x \rceil$ denotes the minimal integer larger than or equal to x. A similar coding can be obtained for the states and outputs. Its importance, in practice, is that combinational circuits for the mappings can be omitted.

For the machine C we can easily encode the inputs, states and outputs as follows.
For the inputs,

$$\lceil \log_2 |I| \rceil = \lceil \log_2 4 \rceil = 2$$

$$N' = \lceil \log_2 |\pi_I| \rceil = \lceil \log_2 2 \rceil = 1$$

$$N'' = \lceil \log_2 |\tau_I| \rceil = \lceil \log_2 2 \rceil = 1$$

$$N' + N'' = 2$$

I	bit1	bit2
1	0	0
2	1	1
3	1	0
4	0	1

where bit1=0 denotes i_1
bit1=1 denotes i_2
bit2=0 denotes j_1
bit2=1 denotes j_2

Similarly, for states,

$$N' = \lceil \log_2 |\pi_S| \rceil = \lceil \log_2 2 \rceil = 1$$

$$N'' = \lceil \log_2 |\tau_S| \rceil = \lceil \log_2 4 \rceil = 2$$

$$N' + N'' = 3$$

Let bit1 denote s_1 and s_2 on C/t_4, bits 2 and 3 denote q_1 through q_4 on C/t_1. The codes for the states of C are naturally formed in the following list

bit1	bit2	bit3	S
0	0	0	1
1	0	1	2
0	1	0	3
1	1	1	4
1	0	0	5
0	0	1	6
1	1	0	7
0	1	1	8

And the output codings are the same as listed above. Finally, a diagram of the realization of machine C is shown in the following figure.

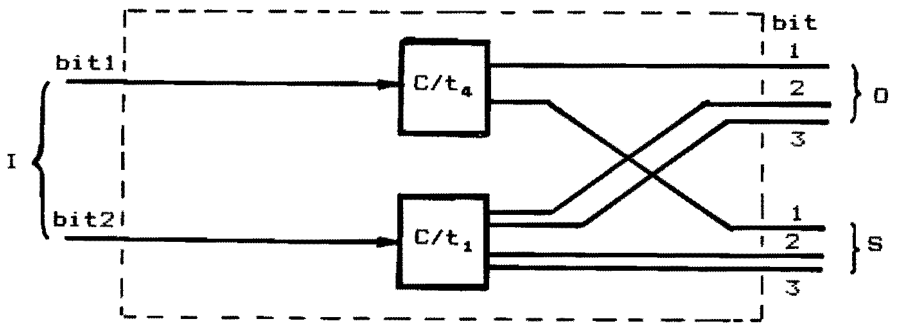


Fig. 4.10 $C = C/t_4 \| C/t_1$
with bit-wires of inputs, states and outputs.

CHAPTER 5

FORCED-TRINITY AND SERIAL FULL-DECOMPOSITION

From Chapter 4 we know that the parallel full-decomposition of sequential machines requires two partition trinities which satisfy the condition that their trinity product is a zero-trinity. In some cases this is a rigourous requirement. In this chapter, we will discuss the serial full-decomposition, that is, how to decompose a given machine into a network consisting of the serial connection of two machines with separate states, separate inputs, and separate outputs. It will be shown that the requirement for serial full-decomposition is weaker than that for parallel full-decomposition.

5.1 Forced-trinity

In this section, we study the relationship between a partition trinity and an image machine, which we call the physical property of a partition trinity. With the same aims, we study some tri-partitions that have a similar character to an PT, if we introduce some external conditions for them, which is called a forced-trinity. In the next section, it will be shown that a forced-trinity precisely describes a tail machine of a serial full-decomposition of a machine.

5.1.1 Physical Property of a Partition Trinity

DEFINITION 5.1

A sequential machine

$M' = (I', S', O', \delta', \lambda')$

is an *image machine* of the machine

$M = (I, S, O, \delta, \lambda)$

if and only if there exist three mappings:

i) Φ is a mapping of S onto S';

ii) Ψ is a mapping of I onto I';

iii) Θ is a mapping of O onto O';

such that $(\Phi, \Psi, \Theta): M \to M'$.

(End of Definition 5.1)

THEOREM 5.1

A partition trinity of a machine M determines an image machine of M. In other words, a partition trinity of a machine M corresponds to an image machine of M.

Proof.

Let $T = (\pi_I, \pi_S, \pi_O)$ be a partition trinity of the machine M, and $\{B_{Si}\}, \{B_{Ij}\}$ and $\{B_{Ok}\}$ be the sets of π_S , π_I and π_O, respectively. Because of the pair properties of a trinity, the machine M' constructed in the following way certainly exists:

$M' = (I', S', O', \delta', \lambda')$

where $I' = \pi_I$, $S' = \pi_S$, $O' = \pi_O$,

and for $s' \in S'$ and $x' \in I'$

$$s'\delta'_{x'} = [s'\overline{\delta}_{x'}]\pi_S \qquad (1)$$

$$s'\lambda'_{x'} = [s'\overline{\lambda}_{x'}]\pi_O \qquad (2)$$

The machine M' is well-defined because pair properties of π_I, π_S and π_O guarantee that,

for any q', q'' in S and z', z'' in I,

if q' and q'' in the same block of π_S and

z' and z'' in the same block of π_I, then

$$[q'\delta_{z'}]\pi_S = [q''\delta_{z''}]\pi_S \qquad (3)$$

$$[q'\lambda_{z'}]\pi_O = [q''\lambda_{z''}]\pi_O \qquad (4)$$

Now, we make three mappings:

$$\Phi: S \to S' \quad \text{by} \quad \Phi(s) = [s]\pi_S, \qquad (5)$$

$$\Psi: I \to I' \quad \text{by} \quad \Psi(x) = [x]\pi_I, \qquad (6)$$

$$\Theta: O \to O' \quad \text{by} \quad \Theta(y) = [y]\pi_O. \qquad (7)$$

Due to the partition property, Φ, Ψ and Θ are one-to-one ont. For any $s\in S$, $x\in I$, we have

$\Phi(s)\delta'_{\Psi(x)}$

$= [s]\pi_S\delta'_{[x]\pi_I}$ $\{(5),(6)\}$

$= [[s]\pi_S\bar{\delta}_{[x]\pi'_I}]\pi_S$ $\{(1)\}$

$= [s\delta_x]\pi_S.$ $\{(3),(\pi_S,\pi_S),(\pi_I,\pi_S)\}$

$= \Phi(s\delta_x)$ $\{(5)\}$

By the same argument, we have

$\Phi(s)\lambda'_{\Psi(x)} = \Theta(s\lambda_x)$

It shows that machine M′ is an image machine of M.

(End of Theorem 5.1)

We refer to Theorem 5.1 as the *physical property* of a partition trinity. From a partition trinity, we can obtain an image machine of the given sequential machine. An image machine has two important properties. Firstly, by using two combinational circuits, an image machine M′ can be simulated by its original machine M. Secondly, by using the connection of two or more image machines, the original machine M can be realized in the behaviours. From this point, an image machine is a component machine of the network which realizes the original machine (see example as follows). In this thesis, we are especially interested in the second property, which will be illustrated in the following sections.

EXAMPLE 5.1

We take the machines D and E shown in Figs. 5.1 and 5.2 as an example to illustrate Theorem 5.1.

	a	b
A	A/y	B/y
B	B/y	A/x

$I_1 = \{a,b\}$
$S_1 = \{A,B\}$
$O_1 = \{x,y\}$

Fig. 5.1 Machine D

	c	d	e	f
C	D/i	C/j	F/i	E/j
D	C/j	D/j	E/j	F/j
E	F/i	E/j	D/k	C/l
F	E/j	F/j	C/l	D/l

$S_2 = \{C,D,E,F\}$
$I_2 = \{c,d,e,f\}$
$O_2 = \{i,j,k,l\}$

Fig. 5.2 Machine E

For machine E, a partition trinity $T = (\pi_I, \pi_S, \pi_O)$,

$$\pi_I = \{\overline{c,d}, \overline{e,f}\},$$
$$\pi_S = \{\overline{C,D}, \overline{E,F}\},$$
and
$$\pi_O = \{\overline{k,l}, \overline{i,j}\},$$

is easily obtained by the trinity computation with machine E. Furthermore, based on the mappings defined in the proof of Theorem 5.1, we get an image machine M that is isomorphic to D. Therefore, for machine D we can simulate it by E, if we connect it in the way shown in Fig. 4.2.

On the other hand, using the method mentioned in Chapter 4 it is easily checked that image machine D is a component machine of a parallel decomposition of machine E. The network is shown in Fig. 5.3. *(End of Example 5.1)*

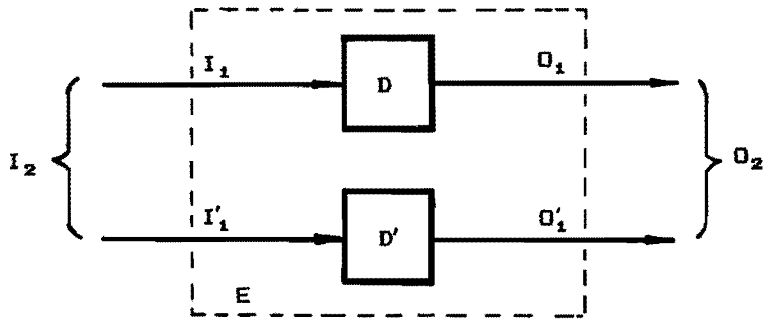


Fig. 5.3 Image machine D as a component machine of a parallel decomposion of E

5.1.2 Forced Trinity

Now, we turn our attention to some tri-partitions with a similar characteristic as an PT. If we substitute a tri-partition (τ_I, τ_S, τ_O) for its original machine, we get a smaller machine with $|\tau_S|$ states, $k \times |\tau_I|$ inputs and $|\tau_O|$ outputs, where k is a constant. Because the smaller machine, in fact, is not an image machine, but it looks like an image machine, and is obtained with some restrictions, such as to k. We refer to this kind of tri-partitions as a forced-trinity.

In order to make a precise description of a forced-trinity, firstly, we will give some definitions about the concept of machine vectors.

DFINITION 5.2

For a machine $M = (I,S,O,\delta,\lambda)$, the column vectors of its machine table are called state vectors or output vectors. Symbolically, they are defined by

$$V_i^s = S\vec{\delta}_i = (s_1\delta_i,\ s_2\delta_i,\ \ldots\ldots,\ s_n\delta_i) \qquad (5.1.a)$$

for a state vector and

$$V_i^o = S\vec{\lambda}_i = (s_1\lambda_i,\ s_2\lambda_i,\ \ldots\ldots,\ s_n\lambda_i) \qquad (5.1.b)$$

for an output vector, where $i \in I$; $n=|S|$; $s_k \in S$; $s_k \neq s_l$ if $k \neq l$; and S is considered as an n-arrangement in some order.
(End of Definition 5.2)

Note that a vector is an ordered n-tuple (or m-tuple, $m<n$, for a subvector) and the order is defined by the position of s_k. In this Chapter we write a vector by

V instead of $\vec{\delta}$ or $\vec{\lambda}$

in order to have a easy notation for developing properties of vectors.

If we substitute $s_k\delta_i$ by its block $[s_k\delta_i]$ of a state partition π and $s_k\lambda_i$ by its block $[s_k\lambda_i]$ of an output partition τ, we have

DEFINITION 5.3

The block vectors of a machine M are defined by

$$V_i^\pi = ([s_1\delta_i],\ [s_2\delta_i],\ \ldots,\ [s_n\delta_i]) \qquad (5.2.a)$$

and $$V_i^\tau = ([s_1\lambda_i],\ [s_2\lambda_i],\ \ldots,\ [s_n\lambda_i]) \qquad (5.2.b)$$

for state block vector and output block vector with partitions π and τ are on S and O of M.
(End of Definition 5.3)

Let π' be another state partition on S. Using partition π' we can divide a vector V into $|\pi|$ segments, each of which is called a subvector of V. A precise description is given as follows.

DEFINITION 5.4

Let B′ be a block of a partition π' on S. Vector $V^{s}_{B',i}$ resp. $V^{o}_{B',i}$ is referred to as subvector of V^{s}_{i} resp. V^{o}_{i} if

$$V^{s}_{B',i} = (s_1\delta_i,\ s_2\delta_i, \ldots\ldots,\ s_m\delta_i) \qquad (5.3.a)$$

resp. $$V^{o}_{B',i} = (s_1\lambda_i,\ s_2\lambda_i, \ldots\ldots,\ s_m\lambda_i) \qquad (5.3.b)$$

where $s_k \in B'$, $k=1\ldots m$, $m=|B'|$ and $s_k \neq s_l$ if $k \neq l$.

(End of Definiton 5.4)

Similarly, we can define subvectors of block vectors by

$$V^{\pi}_{B',i} = ([s_1\delta_i],\ [s_2\delta_i],\ \ldots\ldots,\ [s_m\delta_i]) \qquad (5.4.a)$$

resp. $$V^{\tau}_{B',i} = ([s_1\lambda_i],\ [s_2\lambda_i],\ \ldots\ldots,\ [s_m\lambda_i]) \qquad (5.4.b)$$

where $s_k \in B'$, $k=1\ldots m$, $m=|B'|$, and $s_k \neq s_l$ if $k \neq l$.

Usually, we refer to the state vector and output vector together in many problems. Therefore, we can make an abridged notation by combining (5.4.a) and (5.4.b), such as

$$V^{\pi/\tau}_{B',i} = ([s_1\delta_i]/[s_1\lambda_i],[s_2\delta_i]/[s_2\lambda_i],\ldots,[s_m\delta_i]/[s_m\lambda_i]) \quad (5.5)$$

for a convenient expression in the following sections.

DEFINITION 5.5

Two vectors are said to be equal, *if and only if*

$s_k\delta_i = s_k\delta_j$ for $V^{s}_{i} = V^{s}_{j}$

$s_k\lambda_i = s_k\lambda_j$ for $V^{o}_{i} = V^{o}_{j}$

$[s_k\delta_i]\pi = [s_k\delta_j]\pi$ for $V^{\pi}_{i} = V^{\pi}_{j}$ and $V^{\pi}_{B',i} = V^{\pi}_{B',j}$

$[s_k\lambda_i]\tau = [s_k\lambda_j]\tau$ for $V^{\tau}_{i} = V^{\tau}_{j}$ and $V^{\tau}_{B',i} = V^{\tau}_{B',j}$

for all $s_k \in S$.

(End of Definition 5.5)

For two blocks B′ and B″ with different number of elements in π', we can examine the relationship also with the concept of compatibility, which is defined by

DEFINITION 5.6

Two subvectors, $V^{\pi}_{B',i}$ and $V^{\pi}_{B'',j}$, are said to be compatible with respect to a state partition π'', $\pi' \cdot \pi'' = \pi_S(0)$, that is,

$$V^{\pi}_{B',i} \simeq V^{\pi}_{B'',j} \qquad (\pi'')$$

if and only if ,for all $s \in B'$ and $t \in B''$,

if $[s]\pi'' = [t]\pi''$, then

$[s\delta_i]\pi = [t\delta_j]\pi$ for a state parttion π;

or $[s\lambda_i]\pi = [t\lambda_j]\pi$ for an output partition π,

where $i,j \in I$; $B',B'' \in \pi'$.

(End of Definition 5.6)

Under this definition we can consider two vector operations of two compatible subvectors, which are shown as follows.

If $V^{\pi}_{B',i} \simeq V^{\pi}_{B'',j}$ (π'') and $\pi'' = \{B_1, B_2, \ldots, B_m\}$,

then $V^{\pi}_{B',i} + V^{\pi}_{B'',j} = (A_1, A_2, \ldots, A_m)$

where $A_k \in \pi$ for $k = 1 \ldots m$,

and $A_k = [s_k\delta_i]\pi$ if $s_k \in B'$ and $s_k \in B_k$; or

and $A_k = [t_k\delta_j]\pi$ if $t_k \in B''$ and $t_k \in B_k$; or

B_k = '–' otherwise;

and $V^{\pi}_{B',i} * V^{\pi}_{B'',j} = (A_1, A_2, \ldots, A_m)$

where $A_k \in \pi$, $k = 1 \ldots m$,

and $A_k = [s_k\delta_i]\pi = [t_k\delta_j]\pi$

if $s_k \in B'$, $t_k \in B''$ and $s_k, t_k \in B_k$, or

A_k = '–' otherwise.

When π is an output partition the vector operations are the same as we defined above and are omitted here.

Now, we are at a position to make a definition for forced-trinities.

DEFINITION 5.7

Let τ_S, τ_I and τ_O be partitions of a machine M on S, I and O, respectively. (τ_I,τ_S,τ_O) is called a forced-trinity (FT), *if and only if* either

i) there is an S-O pair (π_S,π_O) such that

$$\pi_S \cdot \tau_S = \pi_S(0)$$

and for all $i,j\in I$ and $B',B''\in\pi_S$,

$$[i]\tau_I = [j]\tau_I \text{ and } V^{\pi_O}_{B',i} \simeq V^{\pi_O}_{B'',j} \quad (\tau_S)$$

implies $\quad V^{\tau_S/\tau_O}_{B',i} \simeq V^{\tau_S/\tau_O}_{B'',j} \quad (\tau_S)$

in this case (τ_I,τ_S,τ_O) is an FT of type I; or

ii) there is a π_S such that

$$\pi_S \cdot \tau_S = \pi_S(0)$$

and for all $i,j\in I$; $B'\in\pi_S$

$$[i]\tau_I = [j]\tau_I \text{ implies } V^{\tau_S/\tau_O}_{B',i} \simeq V^{\tau_S/\tau_O}_{B'',j} \quad (\tau_S)$$

In this case' (τ_I,τ_S,τ_O) is an FT of type II;
where π_S and π_O are referred to forcing-partition (FP).
(End of Definition 5.7)

Because π_S and τ_O are two distinct types of partitions, we simply apply "(τ_I,τ_S,τ_O) with FP π_O or π_S" to state that (τ_I,τ_S,τ_O) is a FP of type I or of type II.

Besed on the definition, a procedure for determining a given (τ_I,τ_S,τ_O) whether or not it is an FT is outlined as follows.

<u>PROCEDURE 5.1</u>

1. Find an π_S such that $\pi_S \cdot \tau_S = \pi_S(0)$;
2. Initialize $\{V^{\tau_S/\tau_O}_{B,b}\}$, $B\in\pi_S$, $b\in\pi_I$, into empty vectors;
3. For all $B\in\pi_S$ do
4. For all $i\in I$ do
5. If $V^{\tau_S/\tau_O}_{B,i} \simeq V^{\tau_S/\tau_O}_{[B\lambda_i]\pi_O,[i]\tau_I} \quad (\tau_S)$,

   then $V^{\tau_S/\tau_O}_{[B\lambda_i]\pi_O,[i]\tau_I} \leftarrow V^{\tau_S/\tau_O}_{[B\lambda_i]\pi_O,[i]\tau_I} + V^{\tau_S/\tau_O}_{B,i}$;

   otherwise, go to 7;
6. (τ_I,τ_S,τ_O) is an FT with τ_S; go to 16;
7. If there is another π_S such that $\pi_S\cdot\tau_S=\pi_S(0)$, then repeat 1-5 for the new π_S; otherwise

```
8. Find a new $\pi_O$ such that $(\pi_S,\pi_O)$ is a pair;
9. Initialize $\{V^{\tau_S/\tau_O}_{B',b}\}$, $B'\in\pi_O$, $b\in\tau_I$, into empty vectors;
10. For all $B\in\pi_S$ do
11. For all $i\in I$ do
12. If $V^{\tau_S/\tau_O}_{B,i} \simeq V^{\tau_S/\tau_O}_{[B\lambda_i]\pi_O,[i]\pi_I}$ $(\tau_S)$
    then $V^{\tau_S/\tau_O}_{[B\lambda_i]\pi_O,[i]\tau_I} \leftarrow V^{\tau_S/\tau_O}_{[B\lambda_i]\pi_O,[i]\tau_I} + V^{\tau_S/\tau_O}_{B,i}$
    otherwise go to 14;
13. $(\tau_I,\tau_S,\tau_O)$ is a FP with $\pi_O$; go to 16;
14. If there is another $\pi_O$, then repeat 8-12 for the new it;
15. $(\tau_I,\tau_S,\tau_O)$ is not an FT;
16. Exit.
```

(End of Procedure 5.1)

With Procedure 5.1 we can obtain an FT with a FP, if they exist. But Theorems 5.2 and 5.3 present other ways to get an FT and its FP.

<u>THEOREM 5.2</u>

If (π_I,π_S) is an I-S pair and (π_I,π_O) is an I-O pair, then (π_I,π_S,π_O) is an FT with any FP τ_S such that $\pi_S\cdot\tau_S=\pi_S(0)$.

proof. The I-S pair (π_I,π_S) implies that

$$[s_k\delta_i]\pi_S = [s_k\delta_j]\pi_S$$

for all $s_k\in S$ and $i,j\in I$, such that $[i]\pi_I = [j]\pi_I$.
Hence, for any a FP τ_S, if $B'\in\tau_S$, then

$$V^{\pi_S}_{B',i} = V^{\pi_S}_{B',j}. \qquad (1)$$

The I-O pair (π_I,π_O) impies that

$$[s_k\lambda_i]\pi_O = [s_k\lambda_j]\pi_O$$

for all $s_k\in S$ and $i,j\in I$, such that $[i]\pi_I = [j]\pi_I$.
Therefore, for the τ_S, if $B'\in\tau_S$, then

$$V^{\pi_O}_{B',i} = V^{\pi_O}_{B',j}. \qquad (2)$$

Combining (1) and (2), we have

$$V^{\pi_S/\pi_O}_{B',i} = V^{\pi_S/\pi_O}_{B',j} \quad (\tau_S).$$

This shows that (π_I,π_S,π_O) is an FT with any FP τ_S.
(End of Theorem 5.2)

<u>THEOREM 5.3</u>

If (π_I,π_S,π_O) is an PT, then (π_I,π_S,π_O) is also an FT with any FP τ_S such that $\tau_S \cdot \pi_S = \pi_S(0)$.

Proof. That (π_I,π_S,π_O) is an PT implies that (π_I,π_S) is an I-S pair and (π_I,π_O) is an I-O pair. From Theorem 5.2 (π_I,π_S,π_O) is an FT with any FP τ_S.
(End of Theorem 5.3)

Under Definition 5.6 the

$$V^{\tau_S/\tau_O}_{\pi\times\tau_I}, \ \pi=\pi_S \quad \text{or} \quad \pi=\pi_O,$$

constructs a transition table of a machine, if we consider each block P_i of π_I as an input; each block Q_j of π_O as an output, and each block R_k of π_S as a state (they are virtually isomorphic mappings). If we refer to the image machine corresponding to a partition trinity as an *independent image machine*, then, we call the machine constructed by

$$\{V^{\tau_S/\tau_O}_{\pi\times\tau_I}\}$$

corresponding to a forced-trinity a *dependent image machine*. This machine can become a component machine of its original machine if some condition is satisfied, that is, it depends on the existence of some indepindent image machine. This will be shown in the following sections.

5.2 Serial Full-Decomposition

5.2.1 Serial Full-decomposition of a State Machine

In our first discussion of serial decomposition, we shall not be derectly concerned with the output of the machine, but are primarily interested in the problem of serial decomposition only with separate inputs and separate states.

<u>DEFINITION 5.8</u>

The *serial connection* of two state machines

$M_1 = (I_1, S_1, \delta^1)$ $\quad\quad$ $M_2 = (I_2, S_2, \delta^2)$

for which $I_2 = S_1 \times I_2$

is the state machine $M = M_1 \rightarrow M_2 = (I_1 \times I_2,\ S_1 \times S_2,\ \delta^*)$

where $\delta^*((s,t),(x_1,x_2)) = (\delta^1(s,x_1), \delta^2(t,(s,x_2)))$.

(End of Definition 5.8)

A diagram of this connection is shown in Fig. 5.4.

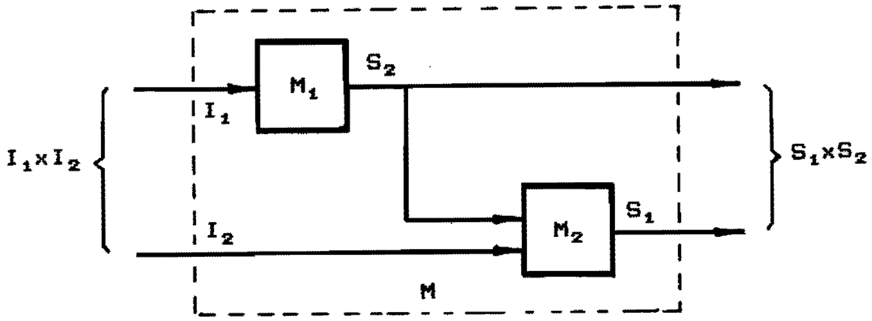


Fig. 5.4. Serial connection of state machines M_1 and M_2 with separate inputs.

<u>DEFINITION 5.9</u>

The state machine $M_1 \rightarrow M_2$ is a *serial full-decomposition* of state machine M if $M_1 \rightarrow M_2$ realizes M.

(End of Definition 5.9)

The serial full-decomposition is *nontrivial* if

$|S_1| < |S|$, $\quad |S_2| < |S|$,

$|I_1| < |I|$, and $|S_1 \times I_2| \leq |I|$.

<u>THEOREM 5.4</u>

The state machine $M = (S, I, \delta)$ has a nontrivial serial full-decomposition if there exist two partitions π_1 and π_2 on S and two partitions τ_1 and τ_2 on I which satisfy the following conditions:

i) (π_1, π_2) is an S-S pair, and

ii) (τ_1, π_1) is an I-S pair, and

iii) (τ_2, π_2) is an I-S pair, and

iv) $\pi_1 \cdot \pi_2 = \pi_S(0)$ and $\tau_1 \cdot \tau_2 = \pi_I(0)$.

Proof.

Given (τ_1,π_1) and (τ_2,π_2) on M, which satisfy

$(\pi_1,\pi_1) \quad \wedge \quad (\tau_1\cdot\tau_2 = \pi_I(0)) \quad \wedge \quad (\pi_1\cdot\pi_2=\pi_S(0))$ (0)

Let M_1 and M_2 be two machines which are constructed by

$M_1 = (\tau_1,\pi_1,\delta')$

$M_2 = (\pi_1\times\tau_2,\pi_2,\delta'')$

where τ_1,τ_2, and π_1,π_2 are considered as collections of blocks, each of which acts as an element of the inputs and outputs of machines M_1 and M_2 and δ' and δ'' are defined by

$\forall B'\in\pi_1 \;\; \forall \beta'\in\tau_1\text{:} \;\; B'\delta'_{\beta'} \;=\; [B'\bar{\delta}_{\beta''}]\pi_1$ (1)

and $\quad \forall B'\in\pi_1 \;\; \forall B''\in\pi_2 \;\; \forall\beta''\in\tau_2\text{:}$

$B''\delta''_{(B',\beta'')} \;=\; [(B''\cap B')\bar{\delta}_{\beta''}]\pi_2.$ (2)

Let $\quad \Psi\text{: } I \to \tau_1\times\tau_2$ be an injective function,

$\Phi\text{: } \pi_1\times\pi_2 \to S$ be a surjectioe partial function

defined by

$\forall i\in I\text{: } \Psi(i) \;=\; ([i]\tau_1,[i]\tau_2)$ (3)

and $\quad \forall(B',B'')\in\pi_1\times\pi_2,\; B'\cap B''\neq\emptyset\text{: } \Phi(B',B'')=B'\cap B''$ (4)

Since $\pi_1\cdot\pi_2=\pi_S(0)$ $\;|B'\cap B''|=1$, that is,

$\exists s\in S\text{: } \Phi(B',B'') \;=\; B'\cap B'' \;=\; s.$ (4')

Now, by the definitions of Φ and Ψ and definition of realization we have

$\forall(B',B'')\in\pi_1\times\pi_2 \;\; \forall x\in I\text{:}$

$\Phi((B',B''))\delta_x$

$= (B'\cap B'')\delta_x$ {(4)}

$= s\delta_x$ {(4')}

$= s\delta_x\cap s\delta_x$ {calculus}

$= (B'\cap B'')\delta_x \;\cap\; (B'\cap B'')\delta_x$ {(4')}

$\subseteq B'\bar{\delta}_x\cap(B'\cap B'')\delta_x$ {Prop.2.7}

$\subseteq B'\bar{\delta}_{[x]\tau_1}\cap(B'\cap B'')\bar{\delta}_{[x]\tau_2}$ {Prop.2.4}

$\subseteq [B'\bar{\delta}_{[x]\tau_1}]\pi_1 \;\cap\; [(B'\cap B'')\bar{\delta}_{[x]\tau_2}]\pi_2$ $\{(\tau_1,\pi_1),(\tau_2,\pi_2)\}$

$= \Phi(([B'\bar{\delta}_{[x]\tau_1}]\pi_1,\; [(B'\cap B'')\bar{\delta}_{[x]\tau_2}]\pi_2))$ {(4)}

$= \Phi((B'\delta'_{[x]\tau_1},\; B''\delta''_{(B',[x]\tau_2)}))$ {(1),(2)}

$= \Phi((B'\delta'_{\Psi(\cdot x)},\; B''\delta''_{(B',\Psi(x\cdot))}))$ $\{(3),\tau_1\cdot\tau_2=\pi_I(0)\}$

$= \Phi((B',B'')\delta^*_{\Psi(x)})$ {Def. 5.8}

It shows that serial connection of M_1 and M_2 realizes M by the definition of realization.
(End of Theorem 5.4)

The procedure for obtaining a serial full-decomposition of a given state machine may be explicitly outlined as follows.

<u>PROCEDURE 5.2</u>

1. Find an I-S pair (τ_1,π_1) such that (π_1,π_1) is an S-S pair;
2. Find an I-S pair (τ_2,π_2) such that
   $\pi_1 \cdot \pi_2 = \pi_S(0)$ and $\tau_1 \cdot \tau_2 = \pi_I(0)$;
3. Construct the machine M_1 using the pair (τ_1,π_1);
4. Construct the machine M_2 using the pair (τ_2,π_2) and partition π_1, and transfer the inputs into $S_1 \times I_2$.

(End of Procedure 5.2)

The following example illustrates this procedure.

<u>EXAMPLE 5.2</u>

Find a serial full-decomposition of the state machine shown in Fig. 5.5.

	1	2	3	4	5	6
1	3	4	3	4	1	2
2	3	3	3	3	1	1
3	2	1	4	3	4	3
4	2	2	4	4	4	4

Fig. 5.5 Machine F.

Step 1. We take the I-S pair (τ_1,π_1),
$\tau_1 = \{\overline{1,2},\overline{3,4},\overline{5,6}\}$ $\pi_1=\{\overline{1,2},\overline{3,4}\}$
It is easily checked that (π_1,π_1) is an S-S pair.

Step 2. I-S pair (τ_2,π_2),
$\tau_2 = \{\overline{1,3,5},\overline{2,4,6}\}$ $\pi_2 = \{\overline{1,3},\overline{2,4}\}$.
is suitable as second pair because it satisfies
$\tau_1 \cdot \tau_2 = \pi_I(0)$ and $\pi_1 \cdot \pi_2 = \pi_S(0)$.

Step 3. Let $\tau_1 = \{\overline{1,2},\overline{3,4},\overline{5,6}\} = \{a,b,c,d\}$ and
$\pi_1 = \{\overline{1,2},\overline{3,4}\} = \{A,B\}$.

Substitute {a,b,c} and {A,B} for $\{\overline{1,2},\overline{3,4},\overline{5,6}\}$ and $\{\overline{1,2},\overline{3,4}\}$ in machine F. We get a new transition table shown in Fig. 5.6 and delete the identical columns and rows. Finally, the machine F_1 is got and shown in Fig. 5.7.

	a	a	b	b	c	c
A	B	B	B	B	A	A
A	B	B	B	B	A	A
B	A	A	B	B	B	B
B	A	A	B	B	B	B

Fig. 5.6 Substitutions

	a	b	c
A	B	B	A
B	A	B	C

Fig. 5.7 Machine F_1.

Step 4. Let $\tau_2 = \{\overline{1,3,5},\overline{2,4,6}\}=\{e,f\}$ and

$\pi_2 = \{\overline{1,3},\overline{2,4}\}=\{C,D\}$.

By the substitutions (see Fig. 5.8(a)), transfer and deletion (see Fig. 5.8(b)), we obtain the machine F_2 shown in Fig. 5.8(c).

		e	f	e	f	e	f
A	C	C	D	C	D	C	D
A	D	C	C	C	C	C	C
B	C	D	C	D	C	D	C
B	D	D	D	D	D	D	D

(a) Substitutions

	A		B	
	e	f	e	f
C	C	D	D	C
D	C	C	D	D

(c) Transfer and deletion

	h	i	j	k
C	C	D	D	C
D	C	C	D	D

(c) Machine F_2

Fig. 5.8 The steps of constructing machine F_2.

The following mappings illustrate the isomorphic relation between machine F and machine $F_1 \rightarrow F_2$.

$S \rightarrow S_1 \times S_2$	$I \rightarrow I_1 \times I_2$	$I'_2 \rightarrow S_1 \times I_2$
$1 \rightarrow (A,C)$	$1 \rightarrow (a,e)$	$h \rightarrow (A,e)$
$2 \rightarrow (A,D)$	$2 \rightarrow (a,f)$	$i \rightarrow (A,f)$
$3 \rightarrow (B,C)$	$3 \rightarrow (b,e)$	$j \rightarrow (B,e)$
$4 \rightarrow (B,D)$	$4 \rightarrow (b,f)$	$k \rightarrow (B,f)$
	$5 \rightarrow (c,e)$	
	$6 \rightarrow (c,f)$	

(End of Example 5.2)

5.2.2 The Type I of Serial Full-Decomposition

We now begin by considering the problem of serial full-decomposition of a Mealy machine. Firstly, we develop the serial full-decomposition of type I where the outputs of the first machine are fed into the second machine as a part of inputs of it.

Furthermore, a systematic method for calculating the forced-trinities used in this type of serial full-decompositions will be discussed.

<u>DIFINITION 5.10</u>

The *serial connection of type I* of two machines

$$M_1 = (I_1, S_1, O_1, \delta^1, \lambda^1)$$
$$M_2 = (I'_2, S_2, O_2, \delta^2, \lambda^2)$$

for which $I'_2 = O_1 \times I_2$

is the machine $M = M_1 \rightarrow M_2 = (I_1 \times I_2, S_1 \times S_2, O_1 \times O_2, \delta^*, \lambda^*)$

where $\delta^*((s,t),(x_1,x_2)) = (\delta^1(s,x_1), \delta^2(t,(\lambda^1(s,x_1),x_2))$

$\lambda^*((s,t),(x_1,x_2)) = (\lambda^1(s,x_1), \lambda^2(t,(\lambda^1(s,x_1),x_2))$

(End of Definition 5.10)

<u>DEFINITION 5.11</u>

The machine $M_1 \rightarrow M_2$ is a *serial full-decomposition of type I* of machine M if the serial connection of type I of M_1 and M_2 realizes M.

(End of Definition 5.11)

<u>THEOREM 5.5</u>

A machine M has a nontrivial serial full-decomposition of type I if there exists a partition trinity (π_I, π_S, π_O) and a forced-trinity (τ_I, τ_S, τ_O) with forcing-partition τ which satisfy:

i) $\tau = \pi_O$, and

ii) $\pi_S \cdot \tau_S = \pi_S(0)$, $\pi_I \cdot \tau_I = \pi_I(0)$ and $\pi_O \cdot \tau_O = \pi_O(0)$.

Proof.

We show that when $t_p = (\pi_I, \pi_S, \pi_O)$ and $t_f = (\tau_I, \tau_S, \tau_O)$ satisfy the above conditions the serial connection of machines M′ constituted by t_p and M″ constituted by t_f realize M.

Let M′ and M″ be

$$M' = (\pi_I, \pi_S, \pi_O, \delta', \lambda')$$
$$M'' = (\pi_O \times \tau_I, \tau_S, \tau_O, \delta'', \lambda'')$$

where for $B' \in \pi_S$ $\mathcal{B}' \in \pi_I$

$$B'\delta'_{\mathcal{B}'} = [B'\overline{\delta}_{\mathcal{B}'}]\pi_S \qquad B'\lambda'_{\mathcal{B}'} = [B'\overline{\lambda}_{\mathcal{B}'}]\pi_O \tag{1}$$

and for $B'' \in \tau_S$, $\mathcal{B}'' \in \tau_I$, $y \in \pi_O$

$$B''\delta''_{(y,\mathcal{B}'')} = [\{s\delta_x \mid s \in B'', x \in \mathcal{B}'', s\lambda'_x \in y\}]\tau_S, \tag{2}$$

$$B''\lambda''_{(y,\mathcal{B}'')} = [\{s\lambda_x \mid s \in B'', x \in \mathcal{B}'', s\lambda'_x \in y\}]\tau_O. \tag{3}$$

Since t_p is a PT, (1) is well-defined, It means that $B'\overline{\delta}_{\mathcal{B}'}$ is located on one and only one block of π_S. So is $B'\overline{\lambda}_{\mathcal{B}'}$. For (2) and (3) they are well-defined too, because t_f is a FT which implies, for $s,t \in S$, $x_1,x_2 \in I$, if

$[s]\tau_S = [t]\tau_S$, $[x_1]\tau_I = [x_2]\tau_I$ and $[s\lambda_{x_1}]\pi_O = [t\lambda_{x_2}]\pi_O$,

then $[s\delta_{x_1}]\tau_S = [t\delta_{x_2}]\tau_S$ and $[s\lambda_{x_1}]\tau_O = [t\lambda_{x_2}]\tau_O$.

Thus, $B''\delta''_{(y,\mathcal{B}'')}$ resp. $B''\lambda''_{(y,\mathcal{B}'')}$ are indeed on *one and only one block* of τ_S resp. τ_O.

Let Ψ: $I \rightarrow \pi_I \times \tau_I$ be an injective function

Φ: $\pi_S \times \tau_S \rightarrow S$ be a surjective partial function

Θ: $\pi_O \times \tau_O \rightarrow O$ be a surjective partial function,

where $\Psi(x) = ([x]\pi_I, [x]\tau_I)$, (4)

$\Phi((B',B'')) = B' \cap B''$ if $B' \cap B'' \neq \emptyset$, (5)

and $\Theta((y',y'')) = y' \cap y''$ if $y' \cap y'' \neq \emptyset$, (6)

Due to the fact that t_p and t_f are orthogonal we know that Ψ, Φ and Θ are one-to-one and that

$$\Phi((B',B'')) \in S \quad \text{and} \quad \Theta((y',y'')) \in O. \tag{7}$$

Therefore, for $(B',B'') \in \pi_S \times \tau_S$, $B' \cap B'' \neq \emptyset$, $x \in I$

$\Phi((B',B''))\delta_x$

$= (B' \cap B'')\delta_x$ {(5)}

$= s\delta_x$ {(7), $(B' \cap B'' = s \in S$}

$= s\delta_x \cap s\delta_x$ {calculus}

$= (B' \cap B'')\delta_x \cap (B' \cap B'')\delta_x$ {calculus}

$\subseteq B'\overline{\delta}_x \cap (B' \cap B'')\delta_x$ {$B' \cap B'' \subseteq B'$}

$\subseteq B'\overline{\delta}_{[x]\pi_I} \cap B''\delta''_{(B'\lambda'_{[x]\pi_I},[x]\tau_I)}$ {$|B' \cap B''|=1$, (2)}

$= B'\delta'_{\Psi(.x)} \cap B''\delta''_{(B'\lambda'_{\Psi(.x)},\Psi(x.))}$ {(4), (2)}

$= \Phi(B'\delta'_{\Psi(.x)}, B''\delta''_{(y',\Psi(x.))}$ {(5), $B'\lambda'_{\Psi(.x)} = y \in \pi_O$}

$= \Phi((B', B'')\delta^*_{\Psi(x))}$ {Def. 5.10}

Similarly,

$$
\begin{aligned}
& \Phi((B',B''))\lambda_X \\
&= (B' \cap B'')\lambda_X && \{(5)\} \\
&= (B' \cap B'')\lambda_X \cap (B' \cap B'')\lambda_X && \{\text{calculus}\} \\
&\subseteq B'\lambda'_{\Psi(\cdot X)} \cap B''\lambda''_{(B'\lambda'_{\Psi(\cdot X)},\Psi(X\cdot))} && \{(4),(3),|B' \cap B''|=1\} \\
&= \Theta(B'\lambda'_{\Psi(\cdot X)}, B''\lambda''_{(y',\Psi(X\cdot))}) && \{(6)\} \\
&= \Theta((B', B'')\lambda^*_{\Psi(X)}) && \{\text{Def. } 5.10\}
\end{aligned}
$$

From the definition of realization we can conclude that $M' \rightarrow M''$ realizes M.

(End of Theorem 5.5)

<u>PROCEDURE 5.3</u>

1. Find a partition trinity (π_I,π_S,π_O);
2. Find a forced-trinity (τ_I,τ_S,τ_O) with forcing-partition τ such that
   i) $\tau = \pi_O$, and
   ii) $|\pi_O| \times |\tau_I| \leq |I|$;
3. Construct the machine M_1 based on partition trinity (π_I,π_S,π_O). In other words, construct the image machine corresponding to (π_I,π_S,π_O);
4. Construct the machine M_2 based on forced-trinity (τ_I,τ_S,τ_O). with FP π_S;
5. Connect machines M_1 and M_2 by the Definition 5.10.

(End of Procedure 5.3)

<u>EXAMPLE 5.3</u>

Consider the machine G given by the transition table in Fig. 5.9.

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

Fig. 5.9 Machine G

Step 1. It is easily checked that (π_I, π_S, π_O),

$\pi_S = \{\overline{1,2}, \overline{3,4}\}$,

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

$\pi_O = \{\overline{1,3}, \overline{2,4}\}$

is a partition trinity of machine G.

Step 2. We take tri-partition (τ_I, τ_S, τ_O),

$\tau_S = \{\overline{1,3}, \overline{2,4}\}$

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

$\tau_O = \{\overline{1,4}, \overline{2,3}\}$,

as a candidate of forced-trinity with forcing-prtition

$\pi_O = \{\overline{1,3}, \overline{2,4}\}$

Here, $|\pi_O| \times |\tau_I| = 2\times2 = 4 < |I| = 8$.

The thing left is to check (τ_I, τ_S, τ_O) whether or not it is a forced-trinity.

Firstly, we substitute {A,B}, {e,f}, and {x,y} for τ_S, τ_I, and τ_O in machine G, respectively. A set of block vectors for machine G is obtained as fllows:

V_1 = (A/y,A/x,B/x,B/y)

V_2 = (B/y,A/y,A/x,B/x)

V_3 = (A/y,A/x,A/y,A/x)

V_4 = (B/y,A/y,B/y,A/y)

V_5 = (B/x,B/y,A/y,A/x)

V_6 = (A/x,B/x,B/y,A/y)

V_7 = (B/x,B/y,B/x,B/y)

V_8 = (A/x,B/x,A/x,B/x)

Where V denotes V^{τ_S/τ_O}.

Secondly, we substitute $\pi_S = \{\overline{1,2}, \overline{3,4}\}$ with $\{\alpha, \beta\}$ to partition of states in machine G. We can divide the vectors above into the following subvectors:

$V_{\beta,1}$=(B/x,B/y)	$V_{\alpha,1}$=(A/y,A/x)
$V_{\beta,2}$=(A/x,B/x)	$V_{\alpha,2}$=(B/y,A/y)
$V_{\beta,3}$=(A/y,A/x)	$V_{\alpha,3}$=(A/y,A/x)
$V_{\beta,4}$=(B/y,A/y)	$V_{\alpha,4}$=(B/y,A/y)
$V_{\beta,5}$=(A/y,A/x)	$V_{\alpha,5}$=(B/x,B/y)
$V_{\beta,6}$=(B/y,A/y)	$V_{\alpha,6}$=(A/x,B/x)
$V_{\beta,7}$=(B/x,B/y)	$V_{\alpha,7}$=(B/x,B/y)
$V_{\beta,8}$=(A/x,B/x)	$V_{\alpha,8}$=(A/x,B/x)

Where V denotes V^{τ_S/τ_O} for short.

It is obvious that

$$V^{\pi_O}_{\alpha,1} \simeq V^{\pi_O}_{\beta,3} \simeq V^{\pi_O}_{\beta,3} \simeq V^{\pi_O}_{\beta,5} \; (\tau_S) \quad \text{implies}$$

$$V^{\tau_S/\tau_O}_{\alpha,1} \simeq V^{\tau_S/\tau_O}_{\beta,3} \simeq V^{\tau_S/\tau_O}_{\alpha,3} \simeq V^{\tau_S/\tau_O}_{\beta,5} \; (\tau_S);$$

$$V^{\pi_O}_{\beta,1} \simeq V^{\pi_O}_{\alpha,5} \simeq V^{\pi_O}_{\beta,7} \simeq V^{\pi_O}_{\alpha,7} \; (\tau_S) \quad \text{implies}$$

$$V^{\tau_S/\tau_O}_{\beta,1} \simeq V^{\tau_S/\tau_O}_{\alpha,5} \simeq V^{\tau_S/\tau_O}_{\beta,7} \simeq V^{\tau_S/\tau_O}_{\alpha,7} \; (\tau_S);$$

$$V^{\pi_O}_{\alpha,2} \simeq V^{\pi_O}_{\beta,4} \simeq V^{\pi_O}_{\alpha,4} \simeq V^{\pi_O}_{\beta,6} \; (\tau_S) \quad \text{implies}$$

$$V^{\tau_S/\tau_O}_{\alpha,2} \simeq V^{\tau_S/\tau_O}_{\beta,4} \simeq V^{\tau_S/\tau_O}_{\alpha,4} \simeq V^{\tau_S/\tau_O}_{\beta,6} \; (\tau_S);$$

$$V^{\pi_O}_{\beta,2} \simeq V^{\pi_O}_{\alpha,6} \simeq V^{\pi_O}_{\alpha,8} \simeq V^{\pi_O}_{\beta,8} \; (\tau_S) \quad \text{implies}$$

$$V^{\tau_S/\tau_O}_{\beta,2} \simeq V^{\tau_S/\tau_O}_{\alpha,6} \simeq V^{\tau_S/\tau_O}_{\alpha,8} \simeq V^{\tau_S/\tau_O}_{\beta,8} \; (\tau_S).$$

Hence, we get

$$\left\{ V^{\tau_S/\tau_O}_{\pi_S \times \tau_I} \right\} = \left\{ V^{\tau_S/\tau_O}_{\alpha,1} , \; V^{\tau_S/\tau_O}_{\beta,1} , \; V^{\tau_S/\tau_O}_{\alpha,2} , \; V^{\tau_S/\tau_O}_{\beta,2} \right\}.$$

This indicates that (τ_I, τ_S, τ_O) is a forced-trinity with forcing-partition π_O.

Step 3. Substitute $\pi_S = \{\overline{1,2}, \overline{3,4}\}$, $\pi_I = \{\overline{1,2}, \overline{3,4}, \overline{5,6}, \overline{7,8}\}$, and $\pi_O = \{\overline{1,3}, \overline{2,4}\}$ by $\{\alpha, \beta\}$, $\{a,b,c,d\}$ and $\{C,D\}$. An image machine G_1 of machine G is obtained and shown in Fig. 5.10.

Step 4. Listing the vectors in $\left\{ V^{\tau_S/\tau_O}_{\pi_S \times \tau_I} \right\}$ into a table with the

title in columns by the following way

titli of $V^{\tau_S/\tau_O}_{B',i} = ([B'\lambda_i]\pi_O, \; [i]\tau_I)$

and with titles in rows by the order

$B_1, B_2, \ldots, B_m, \; B_k \in \tau_S, \; k=1..m.$

The table reprents dependent image machine (tail machine) in a serial full-decomposition of the machine G, which is shown in Fig. 5.11.

Step 5. The serial connection of G_1 and G_2 is the same as Fig.2.7 except for changing M_1 and M_2 into G_1 and G_2.

	a	b	c	d
α	α/D	β/D	β/C	α/C
β	α/C	α/D	β/D	β/C

Fig. 5.10 Machine G_1

	(C,e)	(C,f)	(D,e)	(D,f)
A	B/x	A/x	A/y	B/y
B	B/y	B/x	A/x	A/y

Fig. 5.11 Machine G_2

From the partition trinity and forced-trinity that we apply here, we obtain the following isomorphic mappings between machine G and machine $G_1 \rightarrow G_2$.

Φ: $S \rightarrow S_1 \times S_2$	Ψ: $I \rightarrow I_1 \times I_2$	Θ: $O \rightarrow O_1 \times O_2$
$1 \rightarrow (\alpha,A)$	$1 \rightarrow (a,e)$	$1 \rightarrow (C,x)$
$2 \rightarrow (\alpha,B)$	$2 \rightarrow (a,f)$	$2 \rightarrow (D,x)$
$3 \rightarrow (\beta,A)$	$3 \rightarrow (b,e)$	$3 \rightarrow (C,y)$
$4 \rightarrow (\beta,B)$	$4 \rightarrow (b,f)$	$4 \rightarrow (D,y)$
	$5 \rightarrow (c,e)$	
	$6 \rightarrow (c,f)$	
	$7 \rightarrow (d,e)$	
	$8 \rightarrow (d,f)$	

For example, for 3 in S and 6 in I,

$\Phi(3) = (\beta,A)$, $\Psi(6) = (c,f)$,

$\delta(3,6) = 4$, $\lambda(3,6) = 4$,

$\Phi(4) = (\beta,B)$, $\Theta(4) = (D,y)$,

$\delta^1(\beta,C) = \beta$, $\lambda^1(\beta,C) = D$,

$\delta^2(A,(D,f)) = B$, $\lambda^2(A,(D,f) = y$,

Therefore,

$\Phi(\delta(3,4)) = (\delta^1(\beta,C),\ \delta^2(A,\lambda^1(\beta,C),f)))$,

$\Theta(\lambda(3,4)) = (\lambda^1(\beta,C),\ \lambda^2(A,\lambda^1(\beta,C),f)))$,

(End of Example 5.3)

From Definition 5.6, we know what a forced-trinity means and how to check a tri-partition to see whether or not it is a forced-trinity and what type of forced-trinity it is, if it is a forced-trinity. But it does not tell us how to find an FT easily. That is, to find an FT, if it exists, from the definition we have to take all the possible tri-partitions and check them against the definition. Does a way exist by which we can find all FT's directly, or by which we can see easily that no FT exists for the machine under the forcing of some given trinity?

In the last part of this section, we are going to discuss the problem.

For the sake of convenience, we recall the definition of a forced-trinity of type I here again.

For a given trinity (π_I,π_S,π_O), tri-partition (τ_I,τ_S,τ_O) is a forced-trinity under the force of the trinity *if and only if* for all $i,j\in I$ and $B',B''\in\pi_S$,

$$[i]\tau_I=[j]\tau_I \text{ and } V^{\pi_O}_{B',i}\simeq V^{\pi_O}_{B'',j} \quad (\tau_S)$$

imply $V^{\tau_S/\tau_O}_{B',i} \simeq V^{\tau_S/\tau_O}_{B'',j} \quad (\tau_S)$

Firstly, we analyse the condition $V^{\pi_O}_{B',i}\simeq V^{\pi_O}_{B'',j} \quad (\tau_S)$.

We know the following relationships hold for the Definition 5.5:

$$V^{\pi_O}_{B',i}\simeq V^{\pi_O}_{B'',j} \quad (\tau_S) \iff [s]\tau_S=[t]\tau_S \wedge [s\lambda_i]\pi_O=[t\lambda_j]\pi_O. \qquad (1)$$

Similarly, for $V^{\tau_S}_{B',i}\simeq V^{\tau_S}_{B'',j} \quad (\tau_S)$, we have:

$$V^{\tau_S}_{B',i}\simeq V^{\tau_S}_{B'',j} \quad (\tau_S) \iff [s]\tau_S=[t]\tau_S \wedge [s\lambda_i]\tau_S=[t\lambda_j]\tau_S \qquad (2)$$

Therefore, Definition 5.6(i) becomes that (τ_I,τ_S,τ_O) is a FT *if and only if* for all $i,j\in I$ and $s,t\in S$,

$$[i]\tau_I=[j]\tau_I \wedge [s]\tau_S=[t]\tau_S \wedge [s\lambda_i]\pi_O=[t\lambda_j]\pi_O$$

imply

$$[s]\tau_S=[t]\tau_S \wedge [s\delta_i]\tau_S=[t\delta_j]\tau_S. \qquad (3)$$

By the predicate calculus [19]

$$(A \wedge B \Longrightarrow A \wedge C) \iff (A \wedge B \Longrightarrow C),$$

the (3) becomes

$$[i]\tau_I=[j]\tau_I \wedge [s]\tau_S=[t]\tau_S \wedge [s\lambda_i]\pi_O=[t\lambda_j]\pi_O$$

imply $[s\delta_i]\tau_S=[t\delta_j]\tau_S. \qquad (3')$

Again, based on

$$(A \wedge B \Longrightarrow C) \iff (A \Longrightarrow (B \Longrightarrow C)),$$

(3') becomes

$$[s\lambda_i]\pi_O=[t\lambda_j]\pi_O$$

implies that

$$[i]\tau_I=[j]\tau_I \wedge [s]\tau_S=[t]\tau_S$$

imply $[s\delta_i]\tau_S=[t\delta_j]\tau_S. \qquad (4)$

The equation (4) indicates that, for all $y \in O$ which belong to the same block in π_O, we should check the corresponding entries to see whether they satisfy that,

for any $B \in \tau_S$, $A \in \tau_I$: $B\overline{\delta}_A \subseteq B'' \in \tau_S$. (5)

Before we discuss the procedure, we should make a precise definition on the partial machines produced by a given output partition π_O.

<u>DEFINITION 5.12</u>

Let π_O be a partition on output set of a machine M and y be any block in π_O. Then,

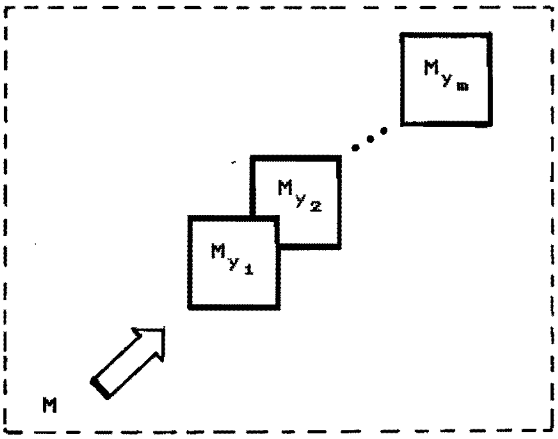
$$M_y = (I,S,\delta_y)$$

is called as a partial state machine with respect to y, for which, for any $s \in S$ and $i \in I$,

$$\delta_y(s,i) = \begin{cases} \text{don't care} & \text{if } \lambda(s,i) \notin y \\ \delta(s,i) & \text{if } \lambda(s,i) \in y \end{cases} \quad (6)$$

(End of Definition 5.12)

From the definition, we see that M_y is an incompletely specified machine and is part of the machine M. Thus, all of the partial machines produced by the blocks of π_O form the original machine M by piling them up together, if we see them transparently. Fig. 5.12 illustrates this idea.



$M = (I,S,O,\delta,\lambda)$, $\pi_O = \{y_1, y_2, \ldots\ldots, y_m\}$

$M_{y_i} = (I,S,\delta_{y_i})$, $i=1\ldots m$,

Fig. 5.12 Machine M and its partial machines

The following procedure describes the method for calculating FT's from partial machines.

<u>PROCEDURE 5.4</u>

1. For given $\pi_O = \{y_1, y_2, \ldots\ldots, y_m\}$ separate M into $\{M_{y_i}\}$.
2. From each M_{y_i} calculate partition pairs
$$P_i = \{(\tau_S,\tau_I) \mid \forall B_I \in \tau_I \wedge \forall B_S \in \tau_S:\ \delta_{y_i}(B_S,B_I) \subseteq B'_S \in \tau_S\}.$$
3. Calculate
$$P = \bigcap_{i=1}^{m} P_i$$
4. If $P=\emptyset$, return
"there is no FT with respect to π_O for M", exit.
5. Calculate the set of FT's based on P
$$FT's = \{(\tau_I,\tau_S,\tau_O) \mid (\tau_S,\tau_I) \in P \wedge \pi_O \cdot \tau_O = \pi_O(0)\}.$$
6. Exit.

(End of Procedure 5.4)

We should explain the step 2 more fully. When we do $\delta_y(B_S,B_I)$, $y \in \pi_O$, we must omit some $s \in B_S$, $x \in I$, such that $\delta_y(s,x)$ is undefined. After Chapter 7 we will see that (τ_I,τ_S) is a weak partition pair with some special features.

In this section, we considered two different ways of calculating a forced-trinity: one by vectors of a machine and the other by partial machines of the machine. With the former we can check given tri-partitions and build a tail machine easily, but it is not so easy to get all the FT's. In contrast, from the latter, we can simply calculate all the FT's, but it takes a very long time, due to the incompletely specified partial machines. In practice, we choose one, or both, of them to reach our goal.

To end this section, we give an example to explain the method mentioned above.

<u>EXAMPLE 5.4</u>

Using Procedure 5.3 calcultate FT's for the machine shown in Fig. 5.13 under the force of trinity

$t = (\pi_I,\ \pi_S,\ \pi_O)$ with

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

$\pi_S=\{\overline{1,2},\overline{3,4}\}$

$\pi_O=\{\overline{1,4},\overline{2,3}\}$

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

Fig. 5.13 Machine H

Step 1. Given $\pi_0 = \{\overline{1,4},\overline{2,3}\}$, the partial machines are $H_{(\overline{1,4})}$ and $H_{(\overline{2,3})}$ shown in Fig. 5.14 respectively.

Step 2. For machine $H_{(\overline{1,4})}$ we obtain $\tau = \{\overline{1,3},\overline{2,4}\}$

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

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

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

such that $\tau \times D \subseteq P_{(\overline{1,4})}$.

For machine $H_{(\overline{3,4})}$ it is obious that

$\tau' = \{\overline{1,3},\overline{2,4}\}$ and

$D' = \{\{\overline{1,3,5},\overline{2,4,6},\overline{7},\overline{8}\},\{\overline{1,3,5,7},\overline{2,4,6,8}\}\}$

such that $\tau' \times D' \subseteq P_{(\overline{2,3})}$

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

(a) $H_{(\overline{1,4})}$

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

(b) $H_{(\overline{2,3})}$

Fig. 5.14 Partial machines of H

Step 3. $\tau \times D \cap \tau' \times D'$

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

$\subseteq P_{(\overline{1,4})} \cap P_{(\overline{2,3})}$

Step 4. $P_{(\overline{1,4})} \cap P_{(\overline{2,3})} \neq \emptyset$ go to step 5.

Step 5. For $\pi_0 = \{\overline{1,4},\overline{2,3}\}$ there are two partitions

$\tau_0 = \{\overline{1,3},\overline{2,4}\}$

$\tau_0' = \{\overline{1,2},\overline{3,4}\}$

which are orthogonal to π_0.

Therefore, tri-partitions

$$\begin{cases} \tau_I = \{\overline{1,3,5,7},\overline{2,4,6,8}\} \\ \tau_S = \{\overline{1,3},\overline{2,4}\} \\ \tau_O = \{\overline{1,3},\overline{2,4}\} \end{cases}$$

and

$$\begin{cases} \tau_I = \{\overline{1,3,5,7},\overline{2,4,6,8}\} \\ \tau_S = \{\overline{1,3},\overline{2,4}\} \\ \tau_O' = \{\overline{1,2},\overline{3,4}\} \end{cases}$$

are forced-trinities with respect to π_0.

(End of Example 5.4)

5.2.3 The Type II of Serial Full-Decomposition

In type I of the serial full-decomposition, it should be noted that there is a problem of time delay. By the way of type I connection the first component machine has to compute its next state and output before the second component machine can compute its next state and output. Thus, if we assume that each machine computation requires a certain time interval, the output of the serial connection appears after two time intervals. This time delay increases with the number of serially connected machines and may be undesirable practical applications. On the other hand, the time delay requires the lasting time of input signals to be long enough for all machines to finish their operations correctly. In other words, the time delay limits the frequency of the input signals. For the reasons above, we must develop another type of serial full-decomposition for seqential machines.

DEINITION 5.13

The *serial connection of type II* of two machines

$$M_1 = (I_1,S_1,O_1,\delta^1,\lambda^1)$$
$$M_2 = (I'_2,S_2,O_2,\delta^2,\lambda^2)$$

for which $I'_2 = S_1 \times I_2$

is the machine $M = M_1 \rightarrow M_2 = (I_1 \times I_2, S_1 \times S_2, O_1 \times O_2, \delta^*, \lambda^*)$

where $\delta^*((s,t),(x_1,x_2)) = (\delta^1(s,x_1),\delta^2(t,(s,x_2)))$

$\lambda^*((s,t),(x_1,x_2)) = (\lambda^1(s,x_1),\lambda^2(t,(s,x_2)))$.

(End of Definition 5.13)

A schematic representation of type II serial connection is shown in Fig. 5.15.

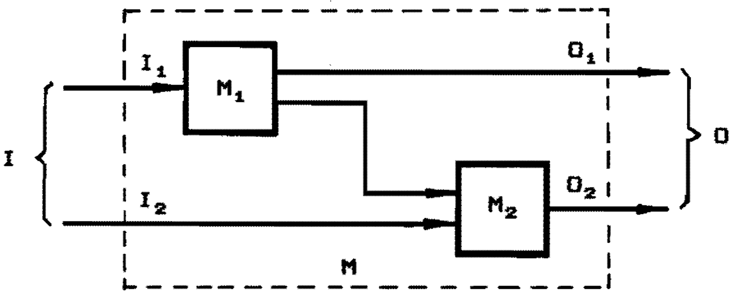


Fig. 5.15 Serial Connection of type II.

DEFINITION 5.14

The machine $M_1 \to M_2$ under the connection of type II is a *serial full-decomposition of type II* of machine M if $M_1 \to M_2$ realizes M.
(End of Definition 5.14)

THEOREM 5.6

The machine M has a nontrivial serial full-decomposition of type II if there exist a partition trinity (π_I, π_S, π_O) and a forced-trinity (τ_I, τ_S, τ_O) with forcing partition τ which statisfy:

i) $\tau = \pi_S$;

ii) Tri-partitions (π_I, π_S, π_O) and (τ_I, τ_S, τ_O) are orthogoal.

Proof. Let $t_p = (\pi_I, \pi_S, \pi_O)$ and
$t_f = (\tau_I, \tau_S, \tau_O)$ with π_S.
By the definition of FT t_f satisfies
$\pi_S \cdot \tau_S = \pi_S(0)$ and for all $i, j \in I$; $B' \in \pi_S$

$$[i]\tau_I = [j]\tau_I \Longrightarrow V^{\tau_S/\tau_O}_{B',i} \simeq V^{\tau_S/\tau_O}_{B'',j} \quad (\tau_S) \tag{1}$$

By the definition of compatible we have
$[i]\tau_I = [j]\tau_I \Longrightarrow$

$$([s]\tau_S = [t]\tau_S \Longrightarrow [s\delta_i]\tau_S = [t\delta_j]\tau_S \wedge [s\lambda_i]\tau_O = [t\lambda_j]\tau_O) \tag{2}$$

Based on the rule of predicate calculus, (2) becomes
$[i]\tau_I = [j]\tau_I \wedge [s]\tau_S = [t]\tau_S \Longrightarrow$

$$([s\delta_i]\tau_S = [t\delta_j]\tau_S \wedge [s\lambda_i]\tau_O = [t\lambda_j]\tau_O). \tag{3}$$

However, since $\pi_S \cdot \tau_S = \pi_S(0)$ if $s, t \in B' \in \pi_S$

$$[s]\tau_S = [t]\tau_S \text{ if and only if } s = t. \tag{4}$$

So, (3) is replaced by

$$[i]\tau_I = [j]\tau_I \Longrightarrow ([s\delta_i]\tau_S = [s\delta_j]\tau_S \wedge [s\lambda_i]\tau_O = [s\lambda_j]\tau_O) \tag{5}$$

which indicates that

(τ_I, τ_S) is an I-S pair, (6)

and (τ_I, τ_O) is an I-O pair. (6′)

Now, let $M' = (\pi_I, \pi_S, \pi_O, \delta', \lambda')$
and $M'' = (\pi_S \times \tau_I, \tau_S, \delta'', \lambda'')$

where
$$B'\delta'_{\beta'} = [B'\overline{\delta}_{\beta'}]\pi_S \tag{7}$$
$$B'\lambda'_{\beta'} = [B'\overline{\lambda}_{\beta'}]\pi_O \tag{7'}$$

for $B' \in \pi_S$ and $\beta' \in \pi_I$;

and
$$B''\delta''_{(B',\beta'')} = [(B' \cap B'')\overline{\delta}_{\beta''}]\tau_S \tag{8}$$
$$B''\lambda''_{(B',\beta'')} = [(B' \cap B'')\overline{\lambda}_{\beta''}]\tau_O \tag{8'}$$

for $B' \in \pi_S$, $B'' \in \tau_S$, $\beta'' \in \tau_I$.

t_p guarantees that (7) and (7') are well-defined. And so do (6) and (6') to (8) and (8').

Let Φ: $\pi_S \times \tau_S \rightarrow S$ defined by

$$\Phi((B',B'')) = B' \cap B'' \qquad (9)$$

Ψ: $I \rightarrow \pi_I \times \tau_I$ defined by

$$\Psi(x) = ([x]\pi_I, [x]\tau_I), \qquad (10)$$

Θ: $\pi_O \times \tau_O \rightarrow O$ defined by

$$\Theta((y',y'')) = y' \cap y''. \qquad (11)$$

Then, for $(B',B'') \in \pi_S \times \tau_S$, $B' \cap B'' \neq \emptyset$; $x \in I$,

$$\begin{aligned}
& \Phi((B',B''))\delta_x \\
= {} & (B' \cap B'')\delta_x && \{(9)\} \\
= {} & (B' \cap B'')\delta_x \cap (B' \cap B'')\delta_x && \{\text{calculus}\} \\
\subseteq {} & B'\overline{\delta}_x \cap (B' \cap B'')\overline{\delta}_x && \{B' \cap B'' \subseteq B'\} \\
\subseteq {} & [B'\overline{\delta}_{[x]\pi_I}]\pi_S \cap [(B' \cap B'')\overline{\delta}_{[x]\tau_I}]\tau_S && \{\text{calculus}\} \\
= {} & B'\delta'_{\Psi(\cdot x)} \cap B''\delta''_{(B',\Psi(x\cdot))} && \{(7),(8),(10)\} \\
= {} & \Phi(B'\delta'_{\Psi(\cdot x)}, B''\delta''_{(B',\Psi(x\cdot))}) && \{(9)\} \\
= {} & \Phi((B', B'')\delta^*_{\Psi(x\cdot)}); && \{\text{Def. 5.14}\}
\end{aligned}$$

and by the same argument we have

$$\begin{aligned}
& \Phi((B',B''))\lambda_x \\
= {} & (B' \cap B'')\lambda_x && \{(9)\} \\
= {} & (B' \cap B'')\lambda_x \cap (B' \cap B'')\lambda_x && \{\text{calculus}\} \\
\subseteq {} & B'\overline{\lambda}_x \cap (B' \cap B'')\overline{\lambda}_x && \{B' \cap B'' \subseteq B'\} \\
\subseteq {} & [B'\overline{\lambda}_{[x]\pi_I}]\pi_O \cap [(B' \cap B'')\overline{\lambda}_{[x]\tau_I}]\tau_O && \{\text{calculus}\} \\
= {} & B'\lambda'_{\Psi(\cdot x)} \cap B''\lambda''_{(B',\Psi(x\cdot))} && \{(7'),(8'),(10')\} \\
= {} & \Theta(B'\lambda'_{\Psi(\cdot x)}, B''\lambda''_{(B',\Psi(x\cdot))}) && \{(11)\} \\
= {} & \Theta((B', B'')\lambda^*_{\Psi(x\cdot)}) && \{\text{Def. 5.14}\}
\end{aligned}$$

Hence, machine $M' \rightarrow M''$ realizes M.

(End of Theorem 5.6)

Comparing Theorem 5.5 with Theorem 5.3, we see that serial full-decomposition of state machine is only a special case of the type II of serial full decomposition omitting the outputs of a sequential machine.

We now outline the procedure of finding a serial full-decomposition of type II of a given machine as follows.

PROCEDURE 5.5

1. Find a partition trinity (π_I, π_S, π_O);
2. Find a forced-trinity (τ_I, τ_S, τ_O) with forcing-partition τ; which satisfy:
   - i) $\tau = \pi_S$
   - ii) $(\pi_I, \pi_S, \tau_O) \odot (\tau_I, \tau_S, \tau_O) = (\pi_I(0), \pi_S(0), \pi_O(0))$;
   - iii) $|\tau| \times |\tau_I| \le |I|$;
3. Set up component machine M_1 based on (π_I, π_S, π_O);
4. Set up component machine M_2 based on (τ_I, τ_S, τ_O) and τ;
5. Connect M_1 and M_2 by the way given in Fig. 5.15.

(End of Procedure 5.5)

EXAMPLE 5.5

Find a serial full-decomposition of type II of machine J shown in Fig. 5.16.

	1	2	3	4	5	6
1	1/8	3/11	5/4	7/3	9/2	11/7
2	1/11	2/8	5/3	6/4	9/7	10/2
3	2/6	1/11	6/12	5/3	10/10	9/7
4	3/5	4/6	7/1	8/12	11/9	12/10
5	12/10	10/9	4/12	2/1	8/6	6/5
6	11/7	11/10	3/3	3/12	7/11	7/6
7	10/2	9/7	2/4	1/3	6/8	5/11
8	9/9	12/2	1/1	4/4	5/5	8/8
9	5/4	5/4	9/8	9/8	1/2	1/2
10	8/12	8/3	12/6	12/11	4/10	4/7
11	7/12	8/3	11/6	12/11	3/10	4/7
12	6/3	6/12	10/11	10/6	2/7	2/10

Fig. 5.16 Machine J.

The computation of partition trinity shows that (π_I, π_S, π_O) is a partition trinity of machine J, where

$$\pi_S = \{\overline{1,2,3,4},\ \overline{5,6,7,8},\ \overline{9,10,11,12}\},$$
$$\pi_I = \{\overline{1,2},\ \overline{3,4},\ \overline{5,6}\},$$
$$\pi_O = \{\overline{1,3,4,12},\ \overline{5,6,8,11},\ \overline{2,7,9,10}\}.$$

The image machine J_1 corresponding to (π_I, π_S, π_O) is shown in Fig. 5.17 with the substitutions of

$\pi_I = \{I, J, K\}$,

$\pi_S = \{M, N, P\}$,

and $\pi_O = \{e, f, g\}$.

	I	J	K
M	M/f	N/e	P/g
N	P/g	M/e	N/f
P	N/e	P/f	M/g

Fig. 5.17 Machine J_1

We choose the tri-partition (τ_I, τ_S, τ_O),

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

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

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

and $\tau = \{\overline{1,3,4,12},\ \overline{5,6,8,11},\ \overline{2,7,9,10}\}$ as the candidate of forced-trinity. It is obvious that $\tau = \pi_S$ and (τ_I, τ_S) is an I-S pair and (τ_I, τ_O) is an I-O pair.

	(M,a)	(M,b)	(N,a)	(N,b)	(P,a)	(P,b)
A	A/w	C/z	D/y	B/x	A/w	A/w
B	A/z	B/w	C/z	C/y	D/y	D/z
C	B/y	A/z	B/w	A/z	C/y	D/z
D	C/x	D/y	A/x	D/w	B/z	B/y

Fig. 5.18 Machine J_2.

In the following substitutions of

$\tau_S = \{A,B,C,D\}$,

$\tau_I = \{a,b\}$,

$\tau_O = \{x,y,z,w\}$, amd

$\tau = \{M,N,P\}$

and comparing of vectors, we obtain a dependent image machine J_2 (see Fig. 5.18). It can be shown that (τ_I, τ_S, τ_O) with τ is a forced-trinity. Therefore, the machine J_2 is a component machine of $J_1 \rightarrow J_2$ which is a serial full-decomposition of type II of machine J. The mappings are listed as follows:

$S \rightarrow S_1 \times S_2$	$I \rightarrow I_1 \times I_2$	$O \rightarrow O_1 \times O_2$
1 → (M,A)	1 → (I,a)	1 → (e,x)
2 → (M,B)	2 → (I,b)	2 → (g,w)
3 → (M,C)	3 → (J,a)	3 → (e,z)
4 → (M,D)	4 → (J,b)	4 → (e,w)
5 → (N,A)	5 → (K,a)	5 → (f,x)
6 → (N,B)	6 → (K,b)	6 → (f,y)
7 → (N,C)		7 → (g,z)
8 → (N,D)		8 → (f,w)
9 → (P,A)		9 → (g,x)
10 → (P,B)		10 → (g,y)
11 → (P,C)		11 → (f,x)
12 → (P,D)		12 → (e,y)

(End of Example 5.5)

CHAPTER 6

H- AND WREATH DECOMPOSITIONS

In this chapter, we shall discuss some special decompositions which are supplementary to the full-decomposition theory introduced in the previous chapters.

6.1 H-decompositions

From chapters 4 and 5 we know that for a given machine M, if its full-decomposition exists, there are then two machines, M_1 and M_2, which are constructed by two partition trinities (for a parallel full-decomposition) or one partition trinity and one forced trinity (for a serial full-decomposition). Hence,

$M \lhd M_1 \| M_2$ or $M \lhd M_1 \rightarrow M_2$

and there are three mappings:

$\Phi: \; S \rightarrow S_1 \times S_2; \quad \Psi: \; I \rightarrow I_1 \times I_2; \quad \Theta: \; O \rightarrow O_1 \times O_2$

where the mappings satisfy,

for i=1,2,

$$|S_i| < |S|; \quad |I_i| < |I|; \quad |O_i| < |O|.$$

However, we note that for some machines that are not fully decomposible, but there are some SP partitions on them. We are interested in looking for some decomposition for them. As a result, we found a type of decompositions that looked exactly like the full-decomposition introduced by Chapter 4.

For the new type of decompositions, we must introduce new mappings on input and output sets as follows

$$\Psi': \quad I \to I_1 \cup I_2 \qquad\qquad \Theta': \quad O \to O_1 \cup O_2$$

where $I_1 \cap I_2 = \emptyset$ and $O_1 \cap O_2 = \emptyset$. From the mappings, we know for each $i \in I$, either $\Psi'(i) \in I_1$ or $\Psi'(i) \in I_2$, which means the component machines M_1 and M_2 only can recognize parts of the inputs of the original machine M via the mapping, but together they can recognize all the inputs of M. In this way the two component machines work in a mutually exclusive way, such that for any an input i in I, only one component machine is in active state, if $\Psi'(i)$ in the input set of the component machine and another is in an inactive state. Therefore, the decomposition is called an H-decomposition due to its feature of half working.

6.1.1 H-connections

There are three main ways of connecting two machines to meet the above mappings corresponding to three modes of machines: state machines, Moore machines and Mealy machines. The connections are called H-connections and defined as follows.

<u>DEFINITION 6.1</u>

Let $M_i = (I_i, S_i, \delta^i)$, $i=1,2$, be two state machines. The H-connection of the two machines is defined by

$$M_1 \vee M_2 = (I_1 \cup I_2,\ S_1 \times S_2\ , \delta^{\vee})$$

where

$$\delta^{\vee}((s_1,s_2),\ i) = \begin{cases} (\delta^1(s_1,i),\ s_2) & \text{if } i \in I_1 \\ (s_1,\ \delta^2(s_2,i)) & \text{if } i \in I_2 \end{cases}$$

for all $(s_1,s_2) \in S_1 \times S_2$ and $i \in I_1 \cup I_2$, $I_1 \cap I_2 = \emptyset$.
(End of Definition 6.1)

We write $M_1 \vee M_2$ for the H-connected machine.

If M_i is a Mealy machine we have the following definition.

DEFINITION 6.2

The H-connection of two Mealy machines M_1 and M_2,

$$M_i = (I_i, S_i, O_i, \delta^i, \lambda^i),\ i=1,2,$$

is defined as follows

$$M_1 \vee M_2 = (I_1 \cup I_2,\ S_1 \times S_2,\ O_1 \cup O_2,\ \delta^{\vee},\ \lambda^{\vee})$$

where

$$\delta^{\vee}((s_1,s_2),\ i) = \begin{cases} (\delta^1(s_1,i),\ s_2) & \text{if } i \in I_1 \\ (s_1,\ \delta^2(s_2,i)) & \text{if } i \in I_2 \end{cases}$$

$$\lambda^{\vee}((s_1,s_2),\ i) = \begin{cases} (\lambda^1(s_1,i),\ s_2) & \text{if } i \in I_1 \\ (s_1,\ \lambda^2(s_2,i)) & \text{if } i \in I_2 \end{cases}$$

for all $(s_1,s_2) \in S_1 \times S_2$ and $i \in I_1 \cup I_2$, $I_1 \cap I_2 = \emptyset$.

(End of Definition 6.2)

The Definition 6.2 can also be used for Moore machines. However, we would like to introduce another definition for them due to the fact that each state in a Moore machine accompanies an output so that we can achieve greater output messages from the connected Moore machines.

DEFINITION 6.3

Let $M_i = (I_i, S_i, O_i, \delta^i, \lambda^i)$, $i=1,2$, be Moore machines. The H-connection of them is defined by

$$M_1 \vee M_2 = (I_1 \cup I_2,\ S_1 \times S_2,\ O_1 \times O_2,\ \delta^{\vee},\ \lambda^{\vee})$$

where $\delta^{\vee}$ is the same as that in Definition 6.1 and

$$\lambda^{\vee}((s_1,s_2)) = (\lambda^1(s_1),\ \lambda^2(s_2))$$

for all $(s_1,s_2) \in S_1 \times S_2$.

(End of Definition 6.3)

From the definition, we know that $M_1 \vee M_2$ presents a new and special work mechanism which shows the characteristics of parallel and mutually exclusive action states. We say it is working parallely since any one of the H-connected machines works independently, that is, its next states and outputs only depend on its present states, not on the states or outputs of another machine, in addition to inputs of the machine. The mutually exclusive is due to the fact that for any input in $I_1 \cup I_2$ only one of the H-connected machines can recognize it, so that it is enabled by the input and another one certainly does not know it so that it appears dummy to the input.

Figure 6.1 shows the structure of a H-connection $M_1 \vee M_2$. It looks exactly like a parallel full-decomposition in Chapter 4 except indicating $I_1 \cup I_2$.

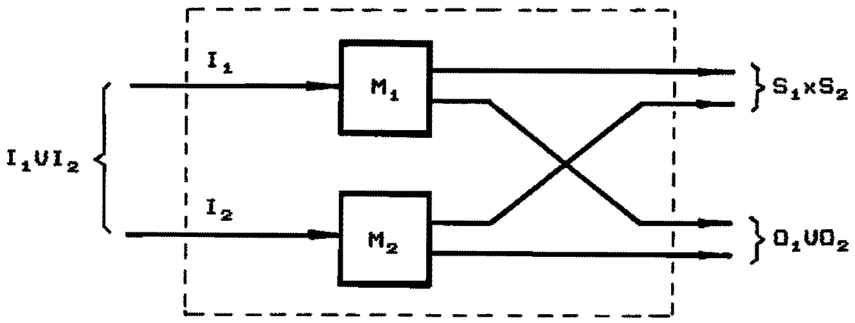


Fig 6.1 Structure of $M_1 \vee M_2$.

In the last part of this section, we are going to discuss some of the properties of H-connections of state machines.

<u>THEOREM 6.1</u>

If both M_1 and M_2, $M_i=(I_i,S_i,\delta^i)$, $i=1,2$, are permutation machines, then $M_1 \vee M_2$ is a permutation machine.

Proof. We know that in general M is a permutation *if and only if*

for any $s,t \in S$

$$s \neq t \Rightarrow s\delta_x \neq t\delta_x \qquad (1)$$

for all $x \in I$ of M.

Let (s_1,s_2) and (t_1,t_2) be any pair of present states in $M_1 \vee M_2$. If $(s_1,s_2) \neq (t_1,t_2)$, it implies

neither $s_1 = t_1$,

nor $s_2 = t_2$.

Therefore, for any $x \in I_1 \cup I_2$,

if $x \in I_1$, $(s_1,s_2)\delta_x^{\vee} = (s_1\delta_x^1,\ s_2)$, (2)

$(t_1,t_2)\delta_x^{\vee} = (t_1\delta_x^1,\ t_2)$. (3)

From (1) we know that if $s_1 \neq t_1$, $s_1\delta_x^1 \neq t_1\delta_x^1$

results in $(s_1,s_2)\delta_x^{\vee} \neq (t_1,t_2)\delta_x^{\vee}$. (4)

Otherwise, $s_2 \neq t_2$ results in the same situation. With the same reason (4) also is true for $x \in I_2$.

Hence, $M_1 \vee M_2$ is a permutation machine.

(End of Theorem 6.1)

<u>THEOREM 6.2</u>

For any $M_i = (I_i, S_i, \delta^i)$, $i=1,2$, the H-connection $M_1 \vee M_2$ never be a reset machine with a constant input mapping.

Proof. Since for any an input x in $I_1 \cup I_2$, it maps the preset states to the next states and keeps one machine inactive, this means the first (or second) components of the next states are the same as the components of the present states. The number of distinct elements in the compomemts are at least $|S_i|$ next states are distinct. Hence, machine $M_1 \vee M_2$ has not a column in the transition table with a constant next state.

(End of Theorem 6.2)

6.1.2 H-PAIRS

In order to analyse the condition of H-decompositions of a machine, we introduce a special partition pair -- H-pair as follows.

<u>DEFINITION 6.4</u>

Let π_I be an input partition with two blocks on a machine M, that is:

$$\pi_I = \{B_0, B_1\}$$

and π_S a partition on state set of M. (π_I, π_S) is a H-pair *if and only if* either for any $x_1 \in B_0$ and $x_2 \in B_1$,

$$B\overline{\delta}_{x_1} \subseteq B \quad \text{and} \quad B\overline{\delta}_{x_2} \subseteq B' \in \pi_S \tag{1}$$

for all $B \in \pi_S$; or for any $x_1 \in B_0$ and $x_2 \in B_1$.

$$B\overline{\delta}_{x_1} \subseteq B' \in \pi_S \quad \text{and} \quad B\overline{\delta}_{x_2} \subseteq B, \tag{2}$$

for all $B \in \pi_S$.

(End of Definition 6.4)

Because of the arbitrary of assumptions for the input blocks B_0 and B_1, (1) is sufficient for the definition of H-pairs. We call input block B_0 in π_I as keeping block and B_1 as acting block.

A H-pair of a machine dedicates the feature of half working of the machine. For the inputs in block B_0 they retain the next states unchanged with respect to partition π_S, but for others in B_1 they make the machine work as usual with respect to π_S. In other words, the feature obviously appears on the factor machine M/π_S of machine M.

A property on H-pairs is given in the following theorem.

THEOREM 6.3

If (π_I,π_S) is a H-pair, (π_S,π_S) then is an S-S pair.

Proof. Following the (1) we know for any $s,t\in S$, $[s]\pi_s=[t]\pi_s$ implies $[s\delta_x]\pi_S= [t\delta_x]\pi_S$ for all $x\in I$.

(End of Theorem 6.3)

In other words, Theorem 6.3 states: if (π_I,π_S) is a H-pair, π_S is an SP partition. We should mention it here that ,in general, a H-pair is not an I-S pair defined by Hartmanis although we have concerned the pair on the sets of inputs and states. If it is an I-S pair, we know the machine is possibly fully decomposible as a state machine and we can solve it with the concept in the previous reports. On the other hand, we should note that an I-S pair is not normally a H-pair. It means that H-pairs give completely a new concept induced by the new problem of decompositions of sequential machines.

Finally, a definition on H-pairs is given to end this section, which will be used in later sections.

DEFINITION 6.5

Two H-pairs, (π_I,π_S) and (τ_I,τ_S) are mutually complement if

i) $B_0 = A_1$ and $B_1 =A_0$,

ii) $\pi_S \cdot \tau_S = \pi_S(0)$

where $\pi_I = \{B_0,B_1\}$ and $\tau_I = \{A_0,A_1\}$.

We call (τ_I,π_S) a complement of H-pair (π_I,π_S) and vice versa.

(End of Definition 6.5)

It is obvious that, for an H-pair, its complement is not unique. In the definition, it is true that $\pi_I=\tau_I$, but they appear to be different functions in the H-pairs. We shall use one input partition to denote the complement H-pairs and indicate one block of it an acting block in a H-pairs and another block as an acting block in another H-pair.

6.1.3 H-decompositions

In this section we start by considering how to evaluate a given machine if it is H-decomposible or not and how to do the H-decomposition if it exists.

Firstly, we consider a state machine of which the H-decomposition is described by the following theorem.

<u>THEOREM 6.4</u>

State machine $M = (I,S,\delta)$ is H-decomposible if there are two complement H-pairs (π_I,π_S) and (π_I,τ_S).

Proof. Suppose (π_I,π_S) and (π_I,τ_S) are complement H-pairs on M and $\pi_I=\{B_0, B_1\}$, B_0 is the acting block of π_S and B_1 the one of τ_S. To construct M_1 and M_2, we take

$$M_1 = (B_0, \pi_S, \delta^1) \quad \text{and} \quad M_2 = (B_1, \tau_S, \delta^2)$$

where a block of π_S is as a state on M_1 and the same on M_2, and

$$s\delta^1_x = [s\overline{\delta}_x]\pi_S \qquad (1)$$

for all $s\in\pi_S$ and $x\in B_0$;

and

$$t\delta^2_x = [t\overline{\delta}_x]\tau_S \qquad (2)$$

for all $t\in\tau_S$ and $x\in B_1$.

Since π_S and τ_S are SP partitions (from Theorem 6.3) the definitions for δ^1 and δ^2 are well-defined.

Next, we should check whether the H-connection of M_1 and M_2 realizes M. For any $s\in S$ and $x\in I$ we have the partial functions:

$$\Phi: \quad \pi_S\times\tau_S \to S \qquad (3)$$

by $\quad \Phi(A,B) = s \quad$ if $A\cap B = s$

and $\quad \Psi: \quad I \to B_0\cup B_1 \qquad (4)$

by $\quad \Psi(x) = x$

where $\quad A\in\pi_S \quad B\in\tau_S$;

since $\quad \pi_S\cdot\tau_S = \pi_S(0)$, Φ is surjective.

By Definition 6.1 and Φ we have

$$\begin{aligned}
& \Phi((A,B))\delta_x \\
= \ & (A\cap B)\overline{\delta}_x && \{(3)\} \\
= \ & (A\cap B)\overline{\delta}_x \cap (A\cap B)\overline{\delta}_x && \{\text{calculus}\} \\
\subseteq \ & A\overline{\delta}_x \cap B\overline{\delta}_x && \{\text{Prop. } 2.7\} \\
\subseteq \ & [A\overline{\delta}_x]\pi_S \cap [B\overline{\delta}_x]\tau_S && \{(\pi_I,\pi_0),(\tau_I,\tau_0)\} \\
= \ & \begin{cases} ([A\overline{\delta}_x]\pi_S = A) \cap [B\overline{\delta}_x]\tau_S & x\in B_1 \quad \{(\pi_I,\pi_S)\} \\ [A\overline{\delta}_x]\pi_S \cap ([B\overline{\delta}_x]\tau_S = B) & x\in B_0 \quad \{(\tau_I,\tau_S)\} \end{cases} \\
= \ & \begin{cases} A \cap [B\overline{\delta}_x]\tau_S & x\in B_1 \\ [A\overline{\delta}_x]\pi_S \cap B & x\in B_0 \end{cases} && \{\text{substitutions}\} \\
= \ & \begin{cases} A \cap B\delta^2_x & x\in B_1 \\ A\delta^1_x \cap B & x\in B_0 \end{cases} && \{(1),(2)\} \\
= \ & \begin{cases} \Phi(A, B\overline{\delta}^2_x) & x\in B_1 \\ \Phi(A\overline{\delta}^1_x, B) & x\in B_0 \end{cases} && \{(3)\}
\end{aligned}$$

$= \Phi((A,B)\delta^{v}_{\Psi(x)})$ {(4),Def. 6.1}

It shows that $M_1 \vee M_2$ is a realization of M.

(End of Theorem 6.4)

We take an example to illustrate Theorem 6.4

EXAMPLE 6.1

For the machine K shown in Fig. 6.2 find a H-decomposition for it if it exists.

	a	b
1	3	2
2	4	1
3	1	4
4	2	3

Fig. 6.2 Machine K.

For the machine

$\pi_S = \{\overline{1,2},\overline{3,4}\}$

and $\tau_S = \{\overline{1,3},\overline{2,4}\}$

are two SP partitions such that $\pi_S \cdot \tau_S = \pi_S(0)$.

Since I={a,b} has two elements, the only partition is zero-πartition

$\pi_I(0) = \{a,b\}$

that can be used here.

For $(\pi_I(0), \pi_S)$ we have

$\{1,2\}\delta_a = \{3,4\}$ $\{3,4\}\delta_a = \{1,2\}$

and $\{1,2\}\delta_b = \{1,2\}$ $\{3,4\}\delta_b = \{3,4\}$

It means that {a} is an acting block and {b} is a keeping block for π_S. In the same way we know that $(\pi_I(0),\tau_S)$ is a H-pair too, and $(\pi_I(0),\pi_S)$ and $(\pi_I(0),\tau_S)$ are complementary.

Thus, Machine K is H-decomposible and the component machines are shown in Fig. 6.3.

	a
1	2
2	1

Machine K_1

	b
1	2
2	1

Machine K_2

Fig. 6.3 Component machines of $K_1 \vee K_2$

Machine K_1 is constructed from (π_I,π_S) and K_2 from (π_I,τ_S).
(End of Example 6.1)

In the example, the machine K has only two inputs. It is said that the machine is not fully decomposible. But we have obtained a H-decomposition with two same component machines. Therefore, under the concept of H-decompositions, a zero-partition is no longer a trivial partition, which differs from full-decomposition analysis in the previous chapters.

Now we present a theorem and an example to show the H-decomposition of Mealy machines.

<u>THEOREM 6.5</u>

A Mealy machine $M = (I,S,O,\delta,\lambda)$ has a H-decomposition if there exist two complements H-pairs (π_I,π_S) and (π_I,τ_S) such that $(\pi_S,\pi_O(O))$ is a restricted S-O pair with respect to one input block of π_I and $(\tau_S,\pi_O(O))$ is a restricted S-O pair with respect to another input block of π_I.

Proof. The concept of a restricted pair comes from Haring. A restricted pair with respect to some inputs means that the pair is defined only on the columns of those inputs of the transition table. A detailed description can be seen in [10]. By the conditions above, if we omit the outputs, M is H-decomposible, which is proved by Theorem 6.4. Here it is necessary only to consider how to keep a correct decomposition for the outputs of M.

Let β_0 denote the set of outputs which appear in the columns of inputs in block B_0 of π_I , and β_1 the set of outputs in the column of inputs in block B_1 of π_I. Then, $\beta_0 \cup \beta_1 = O$ and $\pi_O = \{\beta_0,\beta_1\}$. We construct the component machines of the H-decomposition

by $M_1= (B_0,\pi_S,\beta_0,\delta^1,\lambda^1)$

$M_2= (B_1,\tau_S,\beta_1,\delta^2,\lambda^2)$

where δ^1 and δ^2 are the same as those in the proof of Theorem 6.4, and

$$s\lambda_i^1 = [s\bar{\lambda}_i]\pi_O \qquad (1)$$

$$t\lambda_j^2 = [t\bar{\lambda}_j]\tau_O \qquad (2)$$

where $s\in\pi_S$, $t\in\tau_S$, $i\in B_0$, $j\in B_1$.

Since π_S and τ_S are output-consistent from that $(\pi_S,\pi_O(O))$ and $(\tau_S,\pi_O(O))$ are S-O pairs, (1) and (2) are well-defined. Let θ: $\beta_0 \cup \beta_1 \to O$ by $\theta(y) = y$. It is an one-to-one onto mapping. Both Φ and Ψ are the same as ones in Theorem 6.4. Thus, for all $s \in S$ and $i \in I$, $\Phi(s_1,s_2) = s$, $s_1 \in \pi_S$, $s_2 \in \tau_S$, and $\Psi(i) = i$, $\theta(\lambda(s,i)) = \lambda(s,i)$

On the other hand

$$
\begin{aligned}
& \Phi(s_1,s_2)\lambda_i \\
= \; & (s_1 \cap s_2)\lambda_i && \{(3) \text{ in Theo. } 6.4\} \\
\subseteq \; & \begin{cases} s_1\overline{\lambda}_i \\ s_2\overline{\lambda}_i \end{cases} && \{\text{Prop. } 2.7\} \\
\subseteq \; & \begin{cases} [s_1\overline{\lambda}_i]\pi_O \\ [s_2\overline{\lambda}_i]\tau_O \end{cases} && \{\text{calculus}\} \\
= \; & \begin{cases} s_1\lambda_i^1 \\ s_2\lambda_i^2 \end{cases} && \{(1),(2)\} \\
= \; & \begin{cases} \theta(s_1\lambda_{\Psi(i)}^1) \\ \theta(s_2\lambda_{\Psi(i)}^2) \end{cases} && \{(3)\} \\
= \; & \theta((s_1,s_2)\lambda_{\Psi(i)}^{\vee}) && \{\text{Def. } 6.2\}
\end{aligned}
$$

Hence, $M_1 \vee M_2$ is a H-decomposition of Mealy machine M.
(End of Theorem 6.5)

<u>EXAMPLE 6.2</u>

Find a H-decomposition for Mealy machine L shown in Fig. 6.4, if it is H-decomposible.

	i	j	k
1	3/2	2/e	1/b
2	5/a	4/c	2/b
3	1/b	5/e	1/a
4	6/a	1/d	4/b
5	2/b	6/c	2/a
6	4/b	3/d	4/a

Fig 6.4 Machine L.

By the careful examination of the machine table, we notice that, there are two SP partitions

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

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

which can form two H-pairs with input partition

$$\pi_I = \{\overline{i,k},\overline{j}\}$$

together. That is, (π_I,π_S) and (π_I,τ_S) are complementary H-pairs. Furthermore, we see that $(\pi_S,\pi_0(O))$ is a restricted S-O pair with respect to the input set {i,j} and $(\tau_S,\pi_0(O))$ is a restricted S-O pair with respect to {j}. Therefore, according to Theorem 6.5 there are

$$(B_0,\pi_S,\mathcal{B}_0) \text{ and } (B_1,\tau_S,\mathcal{B}_1),$$

to form machines

$$L_0 = (B_0,\pi_S,\mathcal{B}_0,\delta^0,\lambda^0)$$

and $$L_1 = (B_1,\tau_S,\mathcal{B}_1,\delta^1,\lambda^1)$$

where $B_0 = \{i,k\}$ $\qquad B_1 = \{j\}$

and $\mathcal{B}_0 = \{a,b\}$ $\qquad \mathcal{B}_1 = \{c,d,e\}$

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

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

The δ^0 and δ^1, λ^0 and λ^1 are shown by the machine tables in Fig. 6.5.

	i	k
1	2/a	1/b
2	1/b	1/a

Machine L_0

	j
1	2/e
2	3/c
3	1/d

Machine L_1

Fig. 6.5 Component machines

(End of Example 6.2)

The following theorem states the conditions for evaluating the H-decomposition of a Moore machine. The proof is the same as that in Theorem 6.5

THEOREM 6.6

For a Moore machine M,

$$M = M_1 \vee M_2$$

if there are two complement H-pairs (π_I,π_S) and (π_I,τ_S) which meet, there are two partitions π_0 and τ_0 on output of M

i) (π_S,π_0) is an S-O pair

and ii) (τ_S,τ_0) is an S-O pair

and iii) $\pi_0 \cdot \tau_0 = \pi(O)$.

Proof. With the same argument as that in the proof of Theorem 6.5
(End of Theorem 6.6)

To end this section, we give a simple way to discover if a given machine is not H-decomposible by Theorem 6.7.

THEOREM 6.7

Machine M is not H-decomposible if there is an input which maps all the present states into one state.

Proof. If there is a consistent input mapping on a machine M, from the definition of H-pairs, we know that there is no H-pair which considers the input as a keeping input. This implies that there are not two complementary H-pairs because one of them requires the input as a keeping input.
(End of Theorem 6.7)

This section is only an introduction to the H-decompositions of sequential machines. This work on the decompositions is just a beginning of the complete theory. Some problems remained that are worth further study, such as the H-decomposition of multi-submachines, and a systematic method to find H-pairs for a given machine.

6.2 Wreath Decompositions

Wreath product and decomposition of machines were presented and discussed by Holcombe [16]. The method of wreath decomposition was descibed by the semigroup theory. The decomposition theorem says that, if the transformation semigroup of a machine is decomposible wreathly, then the machine is decomposible too (Theorem 3.1.2 in [16]). Thus, the attention was paid to the study of semigroups of machines.

Since the wreath decomposition presents one part of the serial decomposition method, we do wish to take it as one part of full-decomposition theory. In this section, we will study the wreath decompositions of machines based on a partition pair and a partition trinity, which clearly shows the details of judgement and determinations of the inputs, states and outputs of component machines.

6.2.1 Wreath Connections

<u>DEFINITION 6.6</u>

Let $M_1 = (I_1, S_1, O_1, \delta^1, \lambda^1)$

and $M_2 = (I_2, S_2, O_2, \delta^2, \lambda^2)$

be Mealy machines. The wreath connection of M_1 and M_2 is

$$M_1 \circ M_2 = (I_1 \times I_2, S_1 \times S_2, O_1 \times O_2, \delta^0, \lambda^0)$$

where for $(s,t) \in S_1 \times S_2$, $(x,f) \in I_1 \times I_2^{S_1}$

$$(s,t)\delta^0_{(x,f)} = (s\delta^1_x,\ t\delta^2_{f(x)})$$

and $$(s,t)\lambda^0_{(x,f)} = (s\lambda^1_x,\ t\lambda^2_{f(x)})$$

where

$$f \in I_2^{S_1} = \{f: S_1 \to I_2\}$$

The definition can be depicted in Fig. 6.6.

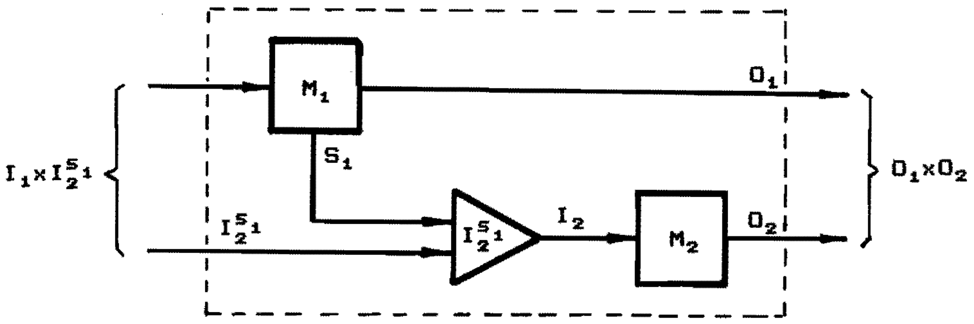


Fig. 6.6 $M_1 \circ M_2$

From the definition, we know that, on one hand, a wreath connection is greatly characterized by the mapping between two component machines. On the other hand, it describes one type of serial full-decompositions. The mapping to the tail machine is a set of all the functions from states of the front machine to inputs of the tail machine. A wreath connection looks very much like a serial full-connection of type II represented in Chapter 5, But, the difference appears in input assignment for the tail machine. In a serial connection, an input is mapped by only one element in the domain, while in a wreath connection, more than one are mapped. From the viewpoint of decomposition, the number of inputs on the tail machine by a wreath decomposition is less than that by a serial full-decomposition for the same machine.

6.2.2 Wreath Decompositions

We start with a definition and notation of compatible classes of machines before we deal with the description of wreath decomposotion.

Let $M = (I,S,O,\delta,\lambda)$ be a machine with distinct inputs. What distinct means is:

for any $i,j\in I$, $V_i^{\tau_S/\tau_O} = V_j^{\tau_S/\tau_O}$ implies $i = j$.

For the sake of simplicity, we will make this restriction, but it can be easily removed when applying the results of this section. Assume that V is a set of all the block vectors, $V_{B,i}$, where $B\in\pi_S$ and $i\in I$. Then a relation R on VxV is defined by $(v_1,v_2)\in V\times V$ v_1Rv_2 iff $v_1\simeq v_2$ (τ_S). The relation R obviously is reflexive, symmetric, and transitive. Therefore, R is an equivalence relation on S. By the relation R, vector set V can be divided into equivalence classes,each of which is defined by

$$[v] = \{v' \mid vRv'\} \tag{1}$$

Naturally, all of equivalence classes form a set

$$W = \{[v] \mid [v] \text{ is an equivalence class over } M\} \tag{2}$$

and we write an equivalence class [v] as

$$[(B,x)] = \{(B',x') \mid V_{(B,x)}^{\tau_S/\tau_O} \simeq V_{(B',x')}^{\tau_S/\tau_O} \quad (\tau_S)\} \tag{3}$$

The equivalence classes are also called compatible classes for an explicit meaning.

For any a machine M with distinct inputs, we can check whether or not it is wreath decomposible with the following theorem.

<u>THEOREM 6.8</u>

Machine M can be realized by some smaller machines M_1 and M_2 in wreath connection, if there exist an PT $t_p = (\pi_I,\pi_S,\pi_O)$ and an FT $t_f = (\tau_I,\tau_S,\tau_O)$ with π_S which satisfy

i) t_p and t_f are orthogonal, and

ii) $|W|^{|\pi_S|} = |\tau_I|$,

where W is a compatible class set.

Proof. Since the conditions for a wreath decomposition are very similar to those for a serial full-decomposition of type II except the extra condition (ii), most of steps followed are the same as those in the proof of Theorem 5.5. We simply state the procedure again here with some differences in the tail machine and in condition only (ii).

Suppose $M_1 = (\pi_I, \pi_S, \pi_O, \delta^1, \lambda^1)$

and $M_2 = (W, \tau_S, \tau_O, \delta^2, \lambda^2)$

where W is the set of all compatible classes over M and its blocks are elements of the input symbols of tail machine M_2. The definitions for δ^1 and λ^1 are the same as (7) and (7′) in Theorem 5.5, while ones for δ^2 and λ^2 are given as follows.

For $B'' \in \tau_S$ and $v \in W$

$$B''\delta^2_v = [(B' \cap B'')\overline{\delta}_f]\tau_S \qquad (8)$$

$$B''\lambda^2_v = [(B' \cap B'')\overline{\lambda}_f]\tau_O \qquad (9)$$

where $v = V^{\tau_S/\tau_O}_{(B',f)}$ and $B' \in \pi_S$ and $f \in \tau_I$.

With the (8) and (9) in mind we can naturally make the definitions on f in τ_I by

for any $f \in \tau_I$ and $B \in \pi_S$

$$f(B) = v \quad \text{if and only if} \quad v = V^{\tau_S/\tau_O}_{(B,f)}. \qquad (10)$$

Let $f(\pi_S) = (f(B_1), f(B_2), \ldots\ldots f(B_m))$

where $\pi_S = \{B_1, B_2, \ldots\ldots B_m\}$.

For f and f′ in τ_2,

$f(\pi_S) = f'(\pi_S)$

if and only if for all $B_i \in \pi_S$

$f(B_i) = f'(B_i)$

Because of distinct inputs on M

and for any $i, j \in I$ V^{τ_S/τ_O}_i and V^{τ_S/τ_O}_j

are compatible if $[i]\tau_I = [j]\tau_I$, we have that,

for any $f, f' \in \tau_I$,

$f(\pi_S) = f'(\pi_S)$ if and only if $f = f'$.

This states that by (10)

$\tau_I = \{ f \mid f: \pi_S \to W \}$

is equal to W^{π_S}, all the mappings from π_S to W, due to the condition (ii).

Now, let us make some relations Φ, Ψ and Θ by

$$\Phi: \pi_S \times \tau_S \to S \quad \text{by} \quad \Phi((B', B'')) = B' \cap B''; \qquad (11)$$

$$\Psi: I \to \pi_I \times \tau_I \quad \text{by} \quad \Psi(x) = (\beta', \beta'') \qquad (12)$$

$$\text{such that} \quad \beta' \cap \beta'' = x;$$

$$\Theta: \pi_O \times \tau_O \to O \quad \text{by} \quad \Theta((y', y'')) = y' \cap y''. \qquad (13)$$

Because of condition (i) both Φ and Θ are surjective partial functions and Ψ is an injective function.
For any $(B',B'') \in \pi_S \times \tau_S$, $B' \cap B'' \neq \emptyset$; $x \in I$,

$$
\begin{aligned}
& \Phi((B',B''))\delta_x \\
&= (B' \cap B'')\delta_x && \{(11)\} \\
&= (B' \cap B'')\delta_x \cap (B' \cap B'')\delta_x && \{\text{calculus}\} \\
&\subseteq (B' \cap B'')\overline{\delta}_{[x]\pi_I} \cap (B' \cap B'')\overline{\delta}_{[x]\tau_I} && \{\text{Prop. 2.7}\} \\
&\subseteq B'\overline{\delta}_{[x]\pi_I} \cap (B' \cap B'')\overline{\delta}_{[x]\tau_I} && \{\text{Prop. 2.7}\} \\
&= B'\delta'_{\mathcal{B}'} \cap (B' \cap B'')\overline{\delta}_{[x]\tau_I} && \{\mathcal{B}'=[x]\pi_I, (7) \text{ in Theo. 5.5}\} \\
&\subseteq B'\delta'_{\mathcal{B}'} \cap B''\delta''_{\mathcal{B}''(B'')} && \{\mathcal{B}''=[x]\tau_I, (8), |B' \cap B''| = 1\} \\
&= \Phi((B'\delta'_{\mathcal{B}'}, B''\delta''_{\mathcal{B}''(B'')})) && \{(11)\} \\
&= \Phi((B',B'')\delta^{o}_{(\mathcal{B}',\mathcal{B}'')}) && \{\text{Def. 6.6}\} \\
&= \Phi((B',B'')\delta^{o}_{\Psi(x)}) && \{(12)\}
\end{aligned}
$$

With the same argument we have

$$
\begin{aligned}
\Phi((B',B''))\lambda_x &= \Theta((B',B'')\lambda^{o}_{\Psi(x)}) \\
&= \Theta(B'\lambda'_{\Psi(\cdot x)}, B''\lambda''_{\Psi(x\cdot)(B')})
\end{aligned}
$$

Hence, $M' \circ M''$ realizes M correctly.
(End of Theorem 6.8)

In the above theorem, condition (ii) is a key for keeping the decomposition as a wreath decomposition. Since the inputs are not relevant to their symbol names, a mapping $f\colon \pi_S \to W$ is in the same situation as $\pi_S \times \tau_I \to W$. Thus, a wreath decomposition is just a special case of serial full-decompositions, where $|\tau_I| = |W|^{|\pi_S|}$.

The steps for a wreath decomposition are implicitly stated in the proof of the theorem. Here we list a procedure for applying the theorem to a wreath decomposition.

<u>PROCEDURE 6.1</u>

1. Find an PT $t_p = (\pi_I, \pi_S, \pi_O)$. If there is no, go to (9);
2. Find a tri-partition $t_f = (\tau_I, \tau_S, \tau_O)$ such that
   $t_p \odot t_f = T_0$. If there is no, go to (1);
3. Calculate compatible classes
   $W = \{[(B,f)]\}$
   to partition τ_S;

4. If t_f is an FT with π_S and $|W|^{|\pi_S|} = |\tau_I|$, then (5); otherwise go to (2);
5. Construct M_1 by t_p;
6. Construct M_2 by putting W in columns

   with the title v on the top of $V^{\tau_S/\tau_O}_{(B,f)}$ if

   $v = [V^{\tau_S/\tau_O}_{(B,f)}]$.

   The collection of v's is the input set of machine M";
7. The mappings of f's is listed by

   $f'(B') = v$ if $V^{\tau_S/\tau_O}_{(B',f')}$ in $[V^{\tau_S/\tau_O}_{(B,f)}]$
8. $M \lhd M' \circ M''$; exit.
9. There do not exist M' and M" such that $M' \circ M''$ realizes M; exit.

(End of Procedure 6.1)

In the case of a computer aided decomposition, we can take steps (2)-(5) in Procedure 5.1 instead of step (3) here. If meeting an input j of which $V^{\tau_S/\tau_O}_{(B,j)}$ is not compatible with the vector $V^{\tau_S/\tau_O}_{(B,f)}$, $j \varepsilon f \varepsilon \tau_I$, we stop the search immediately and go to step (2). In order to make the reader familiar with the theorem and the procedure, we give the following example.

EXAMPLE 6.3

Let us apply the procedure to machine N shown in Fig. 6.7

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

Fig. 6.7 Machine N

Step 1. Consider $t_p = (\pi_I, \pi_O, \pi_S)$

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

$\{\overline{1,2}, \overline{3,4}\}$

$\{\overline{1,3}, \overline{2,4}\})$

which is a partition trinity.

Step 2. Take $t_f = (\tau_I, \tau_S, \tau_O)$

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

$\{\overline{1,3},\overline{2,4}\}$

$\{\overline{1,4},\overline{2,3}\})$

It is apparent that $t_p \odot t_f = T_o$

Step 3. Substitute the blocks of partitions by symbols:

$\{I1,I2\} = (\{\overline{1,4,6,7}, \overline{2,3,5,8}\} = \pi_I$

$\{A1,A2\} = \{\overline{1,2},\overline{3,4}\} = \pi_S$

$\{C1,C2\} = \{\overline{1,3},\overline{2,4}\} = \pi_O$

$\{J1,J2,J3,J4\} = (\{\overline{1,8},\overline{3,6},\overline{4,5},\overline{2,7}\} = \tau_I$

$\{B1,B2\} = \{\overline{1,3},\overline{2,4}\} = \tau_S$

$\{D1,D2\} = \{\overline{1,4},\overline{2,3}\} = \tau_O$

In the following discussion,

V denotes V^{τ_S/τ_O}, for short.

Calculating the block vectors we have

$$\begin{aligned} & V_{(A1,1)} = V_{(A1,8)} = V_{(A1,J1)} \\ = & V_{(A2,1)} = V_{(A2,8)} = V_{(A2,J1)} \\ = & V_{(A1,4)} = V_{(A1,5)} = V_{(A1,J3)} \\ = & V_{(A2,2)} = V_{(A2,7)} = V_{(A2,J4)} \\ = & (B2/D1, B1/D2) \end{aligned} \quad (1)$$

$$\begin{aligned} & V_{(A1,2)} = V_{(A1,7)} = V_{(A1,J4)} \\ = & V_{(A2,3)} = V_{(A2,6)} = V_{(A2,J2)} \\ = & V_{(A2,4)} = V_{(A2,5)} = V_{(A2,J3)} \\ = & V_{(A1,3)} = V_{(A1,6)} = V_{(A1,J2)} \\ = & (B1/D1, B2/D1) \end{aligned} \quad (2)$$

The compatible classes are

$[(A1,J1)] = \{ V_{(A1,J1)}, V_{(A2,J1)}, V_{(A1,J3)}, V_{(A2,J4)}\}$;

$[(A1,J4)] = \{ V_{(A1,J4)}, V_{(A2,J2)}, V_{(A2,J3)}, V_{(A1,J2)}\}$.

Step 4. From (1) and (2) we know, for all $A \in \pi_S$ and $i,j \in I$

$V_{(B,i)} = V_{(B,j)}$ if $[i]\tau_I = [j]\tau_I$

Hence t_f is an FT with π_S.

On the other hand,

$W = \{[(A1,J1)],[(A1,J4)]\}$

$|W|^{|\pi_S|} = 2^2 = 4 = |\tau_I|$

Step 5. The machine N_1 can be formed by t_p, which is drawn in Fig. 6.8

	I1	I2
A1	A2/C1	A1/C1
A2	A1/C2	A2/C1

Fig. 6.8 N_1

Step 6. Columning the vectors, from compatible classes,

$V_{(A1,J1)}$ = (B2/D1, B1/D2)

and $V_{(A1,J4)}$ = (B1/D1, B2/D1)

and assigning the title v_1 and v_2, respectively,we construct the tail machine N_2 shown in Fig. 6.9.

	v1	v2
B1	B2/D1	B1/D1
B2	B1/D2	B2/D1

Fig. 6.9 N_2

The input set of N_2 is $\{v_1,v_2\} = W$

Step 7. The mapping set

$W^{\pi_S} = \{J1,J2,J3,J4\} = \tau_I$

is defined as the following table

A	J1(A)	J2(A)	J3(A)	J4(A)
A1	v1	v2	v1	v2
A2	v1	v2	v2	v1

Step 8. A careful checkness on $N_1 \circ N_2$ and N shows that

$N \lhd N_1 \circ N_2$

Note that machines N_1 and N_2 are isomorphic.

(End of Example 6.3)

In the above theorem, if we omit the ouput partitions, we can easily get a theorem for the wreath decomposition of state machines.

<u>THEOREM 6.9</u>

State machine $M = (I,S,\delta)$ can be decomposible in wreath connection if there exist two I-S pairs, (π_I,π_S) and (τ_I,τ_S) which satisfy

i) $(\pi_I,\pi_S)\cdot(\tau_I,\tau_S) = (\pi_I(0),\pi_S(0))$,

ii) (π_S,π_S) is an S-S pair, and

iii) $|W|^{|\pi_S|} = |\tau_I|$

where $W = \{[V^{\tau_S}_{(B,f)}]\}$

Proof. The proof is exactly the same as that for Theorem 6.8 without considering the output partitions and vectors.

(End of Theorem 6.9)

CHAPTER 7

FULL-DECOMPOSITION OF ISSM's

7.0 Introduction

In many practical design problems, the design specifications require only that a part of the transition table be specified; the rest is left blank or unspecified which is called a don't care (d for short). Moreover, even for a given completely specified machine, the first step in realizing it using digital components is to code the states in binary codes and also the input and output symbols, if they are not binary. In this case, some new blank or unspecified entries might be yielded if the number of symbols is not an integral power of 2. This generally results in an incompletely specified sequential machine (ISSM). Hence, we need to consider the problem of full-decomposition of this type of machines.

Based upon the concepts of weak partition pairs and extended partition pairs presented by Hartmanis for the purpose of state assignments of ISSM's, in this chapter, we will develop the concepts of weak partition trinities and extended partition trinities and use them to solve the problem of full-decomposition of ISSM's. In section 7.1, the definition and properties of weak partition trinities are presented and used for one approach for fully decomposing an ISSM. In section 7.2 we outline the main concepts of extended partition pairs and propose the extended partition trinities as another approach for the full-decomposition of ISSM's. Because of the similarity of discussions to that of partition trinities, we only give some general results here, without a detailed description.

7.1 APPROACH I: WPT

7.1.1 Weak Partition Pair (WPP)

Here, we simply outline the main concepts of weak partition pairs.

DEFINITION 7.1

Let $M = (I, S, O, \delta, \lambda)$ be a machine with d conditions and π and τ be partitions on S, ξ on I, and ω on O. Then, the *weak partition pairs* on M are defined by:

i) (π, τ) is a weak S-S pair, *if and only if*,
for all $s,t \in S$ and all $x \in I$,
$[s]\pi=[t]\pi \Rightarrow [s\delta_x]\tau=[t\delta_x]\tau$
whenever $s\delta_x$ and $t\delta_x$ are both specified.

ii) (ξ, τ) is a weak I-S pair, *if and only if*,
for all $a,b \in I$ and all $s \in S$,
$[a]\xi=[b]\xi \Rightarrow [s\delta_a]\tau=[s\delta_b]\tau$
whenever $s\delta_a$ and $s\delta_b$ are both specified.

iii) (π, ω) is a weak S-O pair, *if and only if*,
for all $s,t \in S$ and all $x \in I$,
$[s]\pi=[t]\pi \Rightarrow [s\lambda_x]\omega=[t\lambda_x]\omega$
whenever $s\lambda_x$ and $t\lambda_x$ are both specified.

iv) (ξ, ω) is a weak I-O pair, *if and only if*,
for all $s \in S$ and all $a,b \in I$,
$[a]\xi=[b]\xi \Rightarrow [s\lambda_a]\omega=[s\lambda_b]\omega$
whenever $s\lambda_a$ and $s\lambda_b$ are specified.

(End of Definition 7.1)

From the definition it is obvious that the following theorem holds.

THEOREM 7.1

If W is the set of all the WPP's on M with d conditions, then

i) $(\pi,\pi(I))$ and $(\pi(O),\pi)$ are in W.

ii) (π_1,τ_1) and (π_2,τ_2) are in W imply $(\pi_1 \cdot \pi_2, \tau_1 \cdot \tau_2)$ in W.

iii) (π_1,τ_1) in W implies $(\pi_1, \tau_1+\pi_2)$ in W.

(End of Theorem 7.1)

It states that the WPP's satisfy all except but the "+" postulate of a pair algebra, which is replaced by a weak form. It can be generalized in order to cover weak pairs. Although some properties are lost in a weak pair algebra, there is still a good possibility of developing the concept of PT-like based upon four WPP's which have some special characters, that is, the weak partition trinities to be discussed below.

7.1.2 Weak Partition Trinity

In the case of an ISSM, there certainly exist some unspecified entries in a machine table. Normally, we denote the entries by dashes. that is, for some $s \in S$ and $i \in I$,

$$s\delta_i = '-' \quad \text{or} \quad s\lambda_i = '-' \ .$$

if $s\lambda_i$ or $s\delta_i$ is unspecified. This causes a little changes for some operation results, such as

$$\{-\} \subseteq B\overline{\delta}_A \quad \text{or} \quad \{-\} \subseteq B\overline{\lambda}_A$$

where $B \subset S$ and $A \subset I$. During the discussions in this section, we keep this in mind.

<u>DEFINITION 7.2</u>

Let $M = (I,S,O,\delta,\lambda)$ be a machine with d conditions and π_S, π_I and π_O be partitions, separately, on S, I, and O. Then, tri-partition (π_I,π_S,π_O) is called a weak partition trinity (WPT), if and only if, for all $A \in \pi_S$, there exist a $B' \in \pi_S$ and a $Y \in \pi_O$, such that

$$B\overline{\delta}_A \subseteq B' \cup \{-\} \text{ and } B\overline{\lambda}_A \subseteq Y \cup \{-\}.$$

(End of Definition 7.2)

The definition naturally hints some connections between a WPT and WPP's, which are stated in theorems 7.2 and 7.3.

THEOREM 7.2

If (π_I,π_S,π_O) is an WPT on an ISSM, then (π_I,π_S), (π_I,π_O), (π_S,π_S) and (π_S,π_O) are WPP's on the ISSM.

Proof.

(π_I,π_S,π_O)

$\Leftrightarrow \forall A\in\pi_I\ \ \forall B\in\pi_S$

$\exists B'\in\pi_S\ \exists Y\in\pi_O$: {def. of WPT}

$B\overline{\delta}_A\subseteq B'\cup\{-\} \wedge B\overline{\lambda}_A\subseteq Y\cup\{-\}$

$\Rightarrow \forall s_1,s_2\in B\ \ \forall x1,x2\in A$: {calculus}

$(s_1\delta_{x1}\neq{}'-'\neq s_1\delta_{x2} \Rightarrow s_1\delta_{x1}\in B' \wedge s_1\delta_{x2}\in B')$

$\wedge\ (s_1\lambda_{x1}\neq{}'-'\neq s_1\lambda_{x2} \Rightarrow s_1\lambda_{x1}\in Y \wedge s_1\lambda_{x2}\in Y)$

$\wedge\ (s_1\delta_{x1}\neq{}'-'\neq s_2\delta_{x1} \Rightarrow s_1\delta_{x1}\in B' \wedge s_2\delta_{x1}\in B')$

$\wedge\ (s_1\lambda_{x1}\neq{}'-'\neq s_2\lambda_{x1} \Rightarrow s_1\lambda_{x1}\in Y \wedge s_2\lambda_{x1}\in Y)$

$\Rightarrow \forall s_1,s_2\in S\ \forall x1,x2\in I$: {calculus}

$([x1]\pi_I=[x2]\pi_I \wedge s_1\delta_{x1}\neq{}'-'\neq s_1\delta_{x2} \Rightarrow [s_1\delta_{x1}]\pi_S=[s_1\delta_{x2}]\pi_S)$

$\wedge\ ([x1]\pi_I=[x2]\pi_I \wedge s_1\lambda_{x1}\neq{}'-'\neq s_1\lambda_{x2} \Rightarrow [s_1\lambda_{x1}]\pi_O=[s_1\lambda_{x2}]\pi_O)$

$\wedge\ ([s_1]\pi_S=[s_2]\pi_S \wedge s_1\delta_{x1}\neq{}'-'\neq s_2\delta_{x1} \Rightarrow [s_1\delta_{x1}]\pi_S=[s_2\delta_{x1}]\pi_S)$

$\wedge\ ([s_1]\pi_S=[s_2]\pi_S \wedge s_1\lambda_{x1}\neq{}'-'\neq s_2\lambda_{x1} \Rightarrow [s_1\lambda_{x1}]\pi_O=[s_2\lambda_{x1}]\pi_O)$

$\Rightarrow (\pi_I,\pi_S) \wedge (\pi_I,\pi_O) \wedge (\pi_S,\pi_S) \wedge (\pi_S,\pi_O)$ {def. of WPP}.

(End of Theorem 7.2)

THEOREM 7.3

Let (π_I,π_S), (π_I,π_O), (π_S,π_S) and (π_S,π_O) be WPP's on an ISSM. Then, (π_I,π_S,π_O) is an WPT on the ISSM if

$\forall s_1,s_2\in S\ \ \forall x1,x2\in I$:

$[s_1]\pi_S=[s_2]\pi_S \wedge [x1]\pi_I=[x2]\pi_I$

$\Rightarrow [s_1\delta_{x1}]\pi_S=[s_2\delta_{x2}]\pi_S \wedge [s_2\delta_{x1}]\pi_S=[s_1\delta_{x2}]\pi_S$ (1)

$\wedge\ [s_1\lambda_{x1}]\pi_I=[s_2\lambda_{x2}]\pi_I \wedge [s_2\lambda_{x2}]\pi_O=[s_1\lambda_{x2}]\pi_O$ (2)

where $s_i\delta_{xj}$ and $s_i\lambda_{xj}$, i,j=1,2, are specified.

Proof. (π_I,π_S), (π_I,π_O), (π_S,π_S) and (π_S,π_O)

imply that $\forall s_1,s_2 \in S \quad \forall x1,x2 \in I$:

$$([x1]\pi_I=[x2]\pi_I \Rightarrow [s_1\delta_{x1}]\pi_S=[s_1\delta_{x2}]\pi_S) \tag{3}$$

$$\wedge \quad ([x1]\pi_I=[x2]\pi_I \Rightarrow [s_1\lambda_{x1}]\pi_O=[s_2\lambda_{x2}]\pi_O) \tag{4}$$

$$\wedge \quad ([s_1]\pi_S=[s_2]\pi_S \Rightarrow [s_1\delta_{x1}]\pi_S=[s_2\delta_{x2}]\pi_S) \tag{5}$$

$$\wedge \quad ([s_1]\pi_S=[s_2]\pi_S \Rightarrow [s_1\lambda_{x1}]\pi_O=[s_2\lambda_{x2}]\pi_O) \tag{6}$$

whenever $s_i\delta_{xj}$ and $s_i\lambda_{xj}$, $i,j=1,2$, are specified.

Combining (1),(3) and (5), we have

$$\forall s_1,s_2 \in S \quad \forall x1,x2 \in I:$$

$$[s_1]\pi_S=[s_2]\pi_S \wedge [x1]\pi_I=[x2]\pi_I$$

$$\Rightarrow \quad [s_1\delta_{x1}]\pi_S=[s_1\delta_{x2}]\pi_S=[s_2\delta_{x1}]\pi_S=[s_2\delta_{x2}]\pi_S \tag{7}$$

whenever $s_i\delta_{xj}$, $i,j=1,2$, are specified.

Combining (2),(4) and (6), we obtain

$$\forall s_1,s_2 \in S \quad \forall x1,x2 \in I:$$

$$[s_1]\pi_S=[s_2]\pi_S \wedge [x1]\pi_I=[x2]\pi_I$$

$$\Rightarrow \quad [s_1\lambda_{x1}]\pi_O=[s_1\lambda_{x2}]\pi_O=[s_2\lambda_{x1}]\pi_O=[s_2\lambda_{x2}]\pi_O \tag{8}$$

whenever $s_i\lambda_{xj}$, $i,j=1,2$, are specified.

Moreover,(7) and (8) mean that

$$\forall A \in \pi_I \quad \forall B \in \pi_S \quad \exists B' \in \pi_S \ \exists Y \in \pi_O:$$

$$B\bar{\delta}_A \subseteq B' \cup \{-\} \ \wedge \ B\bar{\lambda}_A \subseteq Y \cup \{-\}$$

Namely, (π_I,π_S,π_O) is an WPT.

(End of Theorem 7.3)

Like a partition trinity, a weak partition trinity gives the dependences of all information flows on an ISSM. Many properties of partition trinities remain in WPT's except the trinity operation $\oplus$ rules out because of the limited properties of WPP's. Therefore, we study here some simple properties that are used in the study of full-decomposition of an ISSM.

<u>THEOREM 7.4</u>

If (π_I,π_S,π_O) is an WPT on a machine M with d conditions, τ_I on I and $\tau_I \le \pi_I$, and τ_O on O and $\tau_O \ge \pi_O$, then

i) (τ_I,π_S,π_O) is an WPT on M,

ii) (π_I,π_S,τ_O) is an WPT on M, and

iii) (τ_I,π_S,τ_O) is an WPT on M.

Proof. (π_I,π_S,π_O) is an WPT

$\Leftrightarrow \forall A\in\pi_I\ \forall B\in\pi_S\ \exists B'\in\pi_S\ \exists Y\in\pi_O$:

$B\overline{\delta}_A \subseteq B' \cup \{-\} \wedge B\overline{\lambda}_A \subseteq Y \cup \{-\}$. (1)

i) $\tau_I \le \pi_I$

$\Rightarrow \forall A'\in\tau_I\ \exists A\in\pi_I$: $A' \subseteq A$ {def. of ≤}

$\Rightarrow \forall A'\in\tau_I$:

$B\overline{\delta}_{A'} \subseteq B\overline{\delta}_A \wedge B\overline{\lambda}_{A'} \subseteq B\overline{\lambda}_A$ {Prop. 2.4}

$\Rightarrow \forall A'\in\tau_I\ \forall B\in\pi_S\ \exists B'\in\pi_S\ \exists Y\in\pi_O$:

$B\overline{\delta}_{A'} \subseteq B' \cup \{-\} \wedge B\overline{\lambda}_{A'} \subseteq Y \cup \{-\}$. {calculus,(1)}

$\Rightarrow (\tau_I,\pi_S,\pi_O)$ is an WPT. {def. of WPT}

ii) The same as (i).

iii) $\tau_I \le \pi_I \wedge \tau_O \ge \pi_O$

$\Rightarrow (\tau_I,\pi_S,\pi_O)$ is an WPT {(i), (1)}

$\wedge\ \tau_O \ge \pi_O$

$\Rightarrow (\tau_I,\pi_S,\tau_O)$ is an WPT. {(ii)}

(End of Theorem 7.4)

Theorem 7.4 provides one way of computing WPT's. Also, the WPT from which we can get a set of WPT's is called a basic WPT's. It is better to calculate basic WPT's first, than use the theorem to produce all other WPT's. Usually, it is faster and simpler than one by one computation according to the definition of WPT's.

THEOREM 7.5

A WPT of a machine with d conditions corresponds to an image machine of the machine.

Proof. Using the same procedure as in the proof of Theorem 5.2 in Chapter 5, besides doing all argumentation under the condition that $s\delta_x$ or $s\lambda_x$ is specified.

(End of Theorem 7.5)

Similarly to partition trinities, we refer to the theorem as a physical property of the WPT, because it presents a component machine in parallel or series decomposition of an ISSM.

When dealing with serial full-decompositions in chapter 5, we presented the concept of forced-trinity. Similarly, we must consider that concept here again in order to obtain the serial full-decompositions of ISSM's. Because of d conditions, we refer to it as a forced weak trinity (WPT) with some restraints below for the definitions and operations from ones of FT.

i) If $s\delta_i$, $s\in S$ and $i\in I$, is not specified, a dash '−' is put in a vector or a block vector instead of $s\delta_i$ or $[s\delta_i]$, such as in Def. 5.4.

ii) Whenever we deal with $s\delta_i$ and $t\delta_i$, $s,t\in S$ and $i,j\in I$, we must make sure that both $s\delta_i$ and $t\delta_i$ are specified, as in Defs. 5.5, 5.6 and vector operations on compatible subvectors.

The above restraints also apply to the output vectors and operations. With this in mind, we can consider full-decompositions of ISSM's by directly applying similar methods to those Chapters 4 and 5.

7.1.3 Approach I of the full-Decomposition of ISSM's

Now we start by considering the problem of full-decomposition of an incompletely specified sequential machine.

Because of its similarity of discussions with the full-decompositions of completely specified sequential machines, we only need give here the decomposition theorems without proof since they are the same as those for partition trinities.

<u>THEOREM 7.6</u>

A machine $M = (I,S,O,\delta,\lambda)$ with d conditions has a nontrivial parallel full-decomposition if there are two WPT's, (π_I,π_S,π_O) and (τ_I,τ_S,τ_O), such that

$$(\pi_I,\pi_S,\pi_O) \odot (\tau_I,\tau_S,\tau_O) = (\pi_I(0),\pi_S(0),\pi_O(0)).$$

(End of Theorem 7.6)

<u>THEOREM 7.7</u>

A machine $M = (I,S,O,\delta,\lambda)$ with d conditions can be decomposed into a serial connection form of type I, if there exist one WPT (π_I,π_S,π_O), as well as, a forced-WT (τ_I,τ_S,τ_O) with a forcing-πartition τ which satisfy

i) $\tau = \pi_O$, and

ii) $(\pi_I,\pi_S,\pi_O) \odot (\tau_I,\tau_S,\tau_O) = (\pi_I(0),\pi_S(0),\pi_O(0))$

(End of Theorem 7.7)

THEOREM 7.8

A machine $M = (I,S,O,\delta,\lambda)$ with d conditions can be decomposed into a serial connection form of type II if there exist one WPT (π_I,π_S,π_O), as well as, a forced-WT (τ_I,τ_S,τ_O) with τ which satisfy

i) $\tau = \pi_O$;

ii) (τ_I,τ_S) and (τ_S,τ_O) are WPP's;

iii) $(\pi_I,\pi_S,\pi_O) \odot (\tau_I,\tau_S,\tau_O) = (\pi_I(0),\pi_S(0),\pi_O(0))$

(End of Theorem 7.8)

An example is given below in order to illustrate the procedures for decomposing an ISSM using these theorems.

EXAMPLE 7.1

Find a full-decomposition of the machine P shown in Fig. 7.1 in which a don't care condition is denoted by a dash.

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

Fig. 7.1 Machine P

Step 1. For Machine P, computation shows that there are more than two WPT's which satisfy the conditions of parallel full-decomposition given in the Theorem 7.6. Therefore, we choose the largest WPT1 and WPT2 for two component machines.

$WPT1=(\pi_I,\pi_S,\pi_O)$

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

$WPT2=(\tau_I,\tau_S,\tau_O)$

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

Step 2. Construct an image machine corresponding to WPT1. Generally speaking, an image machine corresponding to an WPT can be constructed in two steps:

i) Symbol assignments.

To assign the symbols for the blocks of WPT1, we take WPT1 = ({a,b,c,d},{A,B},{α,β})

Hence, the component machine A1 has the input, state, and

output sets I_1, S_1 and O_1 as the assignment for WPT1.

ii) Determine the machine functions δ^1 and λ^1.

For all x in I_1 and s in S_1,

either

$$s\delta^1_x = [s\bar{\delta}_x - \{-\}]\pi_S \quad \text{if } s\bar{\delta}_x \neq \{-\}$$
$$\text{and } s\lambda^1_x = [s\bar{\lambda}_x - \{-\}]\pi_O \quad \text{if } s\bar{\lambda}_x \neq \{-\}$$

or

$$s\delta^1_x = '-' \quad \text{if } s\bar{\delta}_x = \{-\}$$
$$\text{and } s\lambda^1_x = '-' \quad \text{if } s\bar{\lambda}_x = \{-\}$$

In this way, all entries for Machine P_1 are defined and shown in Fig. 7.2

	a	b	c	d
A	B/β	A/α	B/α	A/β
B	B/α	A/β	A/β	B/α

Fig. 7.2 Machine P_1

Step 3. Construct an image machine corresponding to WPT2.

With the same procedure, we can easily obtain the image machine P_2 based on WPT2 shown in Fig. 7.3, where

$C = \overline{1,5,}$
$D = \overline{2,4,}$
$E = \overline{3,7,}$
$F = \overline{6};$
$e = \overline{1,4,5,6,}$
$f = \overline{2,3,7,}$
$x = \overline{1,4,}$
$y = \overline{2,3,}$
$z = \overline{5,6,}$
$w = \overline{7};$

	e	f
C	C/w	E/x
D	F/z	F/y
E	D/y	C/z
F	E/x	D/w

Fig. 7.3 Machine P_2

Step 4. The mapping between machines P and $P_1 \| P_2$.

$S \rightarrow S_1 \times S_2$	$I \rightarrow I_1 \times I_2$	$O \rightarrow O_1 \times O_2$
$1 \rightarrow (A,C)$	$1 \rightarrow (a,e)$	$1 \rightarrow (\alpha,x)$
$2 \rightarrow (B,D)$	$2 \rightarrow (b,f)$	$2 \rightarrow (\alpha,y)$
$3 \rightarrow (B,E)$	$3 \rightarrow (c,f)$	$3 \rightarrow (\beta,y)$
$4 \rightarrow (A,D)$	$4 \rightarrow (d,e)$	$4 \rightarrow (\beta,x)$
$5 \rightarrow (B,C)$	$5 \rightarrow (b,e)$	$5 \rightarrow (\alpha,z)$
$6 \rightarrow (A,F)$	$6 \rightarrow (c,e)$	$6 \rightarrow (\beta,z)$
$7 \rightarrow (A,E)$	$7 \rightarrow (d,f)$	$7 \rightarrow (\alpha,w)$

(End of Example 7.1)

In this example, we show the decomposition procedure in detail for a good understanding of the properties of WPT's. However, in practice, it can be done in a simple way instead of calculating all sets of $s\bar{\delta}_x$ or $s\bar{\lambda}_x$. After giving the block symbols, we can list the table of an image machine for the new inputs with the input block symbols and for the present state with the state block symbols. The next states and outputs can be filled by finding a state in the corresponding present state block and one input in the corresponding input block. The blocks of the next state and output of the state and input in the original machine table should be the entries in the image machine table. In fact, this just is the computation of δ and λ on the blocks. For example, for the machine P_2

$$c\delta_\alpha^2 = [1\delta_1]\tau_S = c$$

$$c\lambda_\alpha^2 = [5\lambda_1]\tau_O = \omega.$$

Correctness is ensured by examming the properties of the weak partition trinities.

7.2 Approach II: EPT

7.2.1 Extended Partition Pair (EPP)

In the concept of WPT's, we ignored the occurences of d conditions. In that situation, trinity operation $\oplus$ is ruled out, so that one operation is lost in the WPT algebra. In approach II, we give each d condition a separate name, and then keep a careful record of it. A machine with labelled d conditions is given by a machine table where values of δ may be from a set C of labels and some values of λ may be from a set D of labels. Under this consideration, the concept of an extended partition pair is naturally obtained.

<u>DEFINITION 7.3</u>

Let $M = (I,S,O,\delta,\lambda)$ be a machine with labelled d conditions C and D and π be partition on S, τ on SUC, ξ on I, and w on OUD. Then, the extended partition pairs (EPP's) on M are defined by

i) (π, τ) is an S-SUC pair *if and only if*,

for all $s,t \in S$ and all $x \in I$,

$[s]\pi=[t]\pi \Longrightarrow [s\delta_x]\tau=[t\delta_x]\tau$;

ii) (ξ, τ) is an I-SUC pair *if and only if*

for all $a,b \in I$ and all $s \in S$,

$[a]_\xi=[b]_\xi \Longrightarrow [s\delta_a]\tau=[s\delta_b]\tau$;

iii) (π, ω) is an S-OUD pair *if and only if*

for all $s,t \in S$ and all $x \in I$,

$[s]\pi=[t]\pi \Longrightarrow [s\lambda_x]\omega=[t\lambda_x]\omega$;

vi) (ξ, ω) is an I-OUD pair *if and only if*

for all $a,b \in I$ and all $s \in S$,

$[a]_\xi=[b]_\xi \Longrightarrow [s\lambda_a]\omega=[s\lambda_b]\omega$;

(End of Definition 7.3)

Now, we take the machine Q shown in Fig. 7.4 as an example to illustrate the concept of EPP.

	1	2	3	4	5	6
1	7/1	5/6	2/5	7/2	3/1	3/2
2	7/4	4/3	d_3/3	7/5	6/4	6/5
3	9/d_1	5/2	2/1	6/4	4/1	8/2
4	6/4	2/3	5/3	6/5	8/d_2	4/3
5	2/5	3/2	3/1	5/6	9/5	1/6
6	2/1	7/4	2/4	5/2	4/1	8/2
7	2/4	8/2	4/1	7/4	9/4	6/4
8	d_2/5	7/2	d_1/1	1/6	3/1	3/2
9	5/3	7/5	7/4	2/3	8/3	4/3

Fig. 7.4 Machine Q

In the machine

$C = \{d_1,d_2,d_3\}$ $SUC = \{1,2,3,4,5,6,7,8,9,d_1,d_2,d_3\}$

$D = \{d_1,d_2\}$ $OUD = \{1,2,3,4,5,6,d_1,d_2\}$

Observe that,

$$(\pi_1,\tau_1)=(\{\overline{1},\overline{2,7},\overline{3},\overline{4},\overline{5},\overline{6},\overline{7},\overline{8},\overline{9}\},\{\overline{1},\overline{2,7,d_1},\overline{3},\overline{4,8,d_3},\overline{5},\overline{6,9,d_2}\})$$

and

$$(\pi_2,\tau_2)=(\{\overline{1,7},\overline{2},\overline{3},\overline{4},\overline{5},\overline{6},\overline{7},\overline{8},\overline{9}\},\{\overline{1},\overline{2,4,7,d_2},\overline{3,6,9,d_2},\overline{5,8,d_2}\})$$

are EPP's. The partition operations of multiplication and addition hold on the set of all EPP's such as

$$(\pi_1\cdot\pi_2,\ \tau_1\cdot\tau_2)$$
$$=(\{\overline{1},\overline{2},\overline{3},\overline{4},\overline{5},\overline{6},\overline{7},\overline{8},\overline{9}\},\{\overline{1},\overline{2,7,d_1},\overline{3},\overline{4},\overline{5},\overline{6,9,d_2},\overline{8,d_3}\})$$

and

$$(\pi_1+\pi_2,\ \tau_1+\tau_2)$$
$$=(\{\overline{1,2,7},\overline{3},\overline{4},\overline{5},\overline{6},\overline{8},\overline{9}\},\{\overline{1},\overline{2,4,5,7,8,d_1,d_3},\overline{3,6,9,2,d_2}\})$$

are also EPP's. More generally we have the next lemma.

<u>LEMMA 7.1</u>

The set of all extended partition pairs on a machine with labelled d conditions is a pair algebra.

Proof. The proof for PP algebra carries over word for word except that set SUC or OUD is used instead of S or O.

(End of Lemma 7.1)

Now we have the m operator and M operator with all pair algebra results at our disposal. That is, on the algebra of extended pairs, we have m and M operations on the pairs of S-SUC, I-SUC, S-OUD and I-OUD.

In the following discussions, when we refer to $\overline{\tau}$ as the *restriction of τ to* S, we mean
for all s, t$\in$S, $\overline{\tau}$ on S, and τ on SUC,

$$[s]\overline{\tau} = [t]\overline{\tau} \iff [s]\tau = [t]\tau$$

In the same way, we have the restriction $\overline{\omega}$ of ω to O defined by for all $\alpha,\beta\in$O, $\overline{\omega}$ on O, and ω on OUD,

$$[\alpha]\overline{\omega} = [\beta]\overline{\omega} \iff [\alpha]\omega = [\beta]\omega.$$

7.2.2 Extended Partition Trinity

Under the definition of extended partition pairs, the concept of an extended partition trinity is naturally obtained and is simply described here. It is another useful tool for studying the full-decomposition of ISSM's.

DEFINITION 7.4

Let $M = (I,S,O,\delta,\lambda)$ be a machine with labeled d conditions C and D and π_S be a partition on SUC, π_I on I, and π_O on OUD. Then, tri-partition (π_I,π_S,π_O) is called an extended partition trinity (EPT), *if and only if*, for all $B\epsilon\overline{\pi}_S$ and $A\epsilon\pi_I$, there exist a $B'\epsilon\pi_S$ and a $Y\epsilon\pi_O$ such that

$$B\overline{\delta}_A \subseteq B' \quad \text{and} \quad B\overline{\lambda}_A \subseteq Y$$

where $\overline{\pi}_S$ is the restriction of π_S to S

(End of Definition 7.4)

Like Theorem 3.2, we have a similar result for ISSM's.

THEOREM 7.8

A tri-partition (π_I,π_S,π_O) on a machine with labelled d conditions is an EPT if and only if (π_I,π_S), (π_I,π_O), $(\overline{\pi}_S,\pi_S)$, and $(\overline{\pi}_S,\pi_O)$ are EPP's.

Proof. The proof is exactly the same as that in Theorem 3.2 except we have to pay attention to restricted partitions sometimes. So, we omit it here.

(End of Theorem 7.8)

With the definition and the theorem in mind, we can prove that the trinity operations of ⊙ and ⊕ are closed within the set of all EPT's of an ISSM. This just is the advantage of EPT's over WPT's because the operation ⊕ holds. Therefore, we can study the EPT's by a similar manner as that on PT algebra. All of these will be referred to in later discussions without writing out their formal forms.

7.2.3 The Full-Decomposition of ISSM's By EPT's

The concept of EPT algebra presents another approach for the full-decomposition of an ISSM. Similarly, we can develop some decomposition theorems on the parallel full-decomposition and serial full-decomposition of ISSM's by applying EPT's.

Here, we give the decomposition theorems without detailed description or proof which can be easily derived in a similar way to those in the previous chapters. Finally, an example of serial full-decomposition of type I of an ISSM is given to illustrate the special characteristics of decomposition of ISSM's in this approach.

THEOREM 7.9

let $M = (I,S,O,\delta,\lambda)$ be a machine with labelled d conditions C and D. Then,

a) M has a nontrivial parallel full-decomposition if there exist two EPT's
(π_I,π_S,π_O) and (τ_I,τ_S,τ_O) such that
$(\pi_I,\overline{\pi}_S,\overline{\pi}_O) \odot (\tau_I,\overline{\tau}_S,\overline{\tau}_O) = (\pi_I(0),\pi_S(0),\pi_O(0))$;

b) M has a nontrivial serial full-decomposition of type I if there are an EPT (π_I,π_S,π_O) and a forced-EPT (τ_I,τ_S,τ_O) with τ which satisfy

i) $\tau = \pi_O$ and
ii) $(\pi_I,\overline{\pi}_S,\overline{\pi}_O) \odot (\tau_I,\overline{\tau}_S,\overline{\tau}_O) = (\pi_I(0),\pi_S(0),\pi_O(0))$;

c) M has a nontrivial serial full-decomposition of type II if there exist an EPT (π_I,π_S,π_O) and a forced-EPT (τ_I,τ_S,τ_O) with τ which satisfy

i) $\tau = \pi_S$,
ii) (τ_I,τ_S) and (τ_I,τ_O) are EPP's, and
iii) $(\pi_I,\overline{\pi}_S,\overline{\pi}_O) \odot (\tau_I,\overline{\tau}_S,\overline{\tau}_O) = (\pi_I(0),\pi_S(0),\pi_O(0))$,

where
$\overline{\pi}_S$ is the restriction of π_S to S;
$\overline{\pi}_O$ is the restriction of π_O to O;
$\overline{\tau}_S$ is the restriction of τ_S to S;
$\overline{\tau}_O$ is the restriction of τ_O to O.

(End of Theorem 7.9)

EXAMPLE 7.2

Consider the incompletely specified sequential machine B shown in Fig. 7.4 and find a full-decomposition of it.

In this example an V represents an V^{τ_s/τ_o} for short.

Step 1. Compute the EPT's.
By the computation of EPT's on a computer, the machine has totally seven nontrivial EPT's listed below:

$$\begin{aligned}\mathrm{EPT1} = (&\{\overline{1,4},\overline{2,3},\overline{5,6}\},\\ &\{\overline{1,6,9,d_2},\overline{2,5,7,d_1},\overline{3,4,8,d_3}\},\\ &\{\overline{1,2,3,d_2},\overline{4,5,6,d_1}\});\end{aligned}$$

$$\begin{aligned}\mathrm{EPT2} = (&\{\overline{1},\overline{4},\overline{2,3},\overline{5,6}\},\\ &\{\overline{1,6,9,d_2},\overline{2,5,7,d_1},\overline{3,4,8,d_3}\},\\ &\{\overline{1,2,3,d_2},\overline{4,5,6,d_1}\});\end{aligned}$$

$$\begin{aligned}\mathrm{EPT3} = (&\{\overline{1},\overline{4},\overline{2,3},\overline{5},\overline{6}\},\\ &\{\overline{1,6,9,d_2},\overline{2,5,7,d_1},\overline{3,4,8,d_3}\},\\ &\{\overline{1,2,3,d_2},\overline{4,5,6,d_1}\});\end{aligned}$$

$$\begin{aligned}\mathrm{EPT4} = (&\{\overline{1},\overline{4},\overline{2},\overline{3},\overline{5,6}\},\\ &\{\overline{1,6,9,d_2},\overline{2,5,7,d_1},\overline{3,4,8,d_3}\},\\ &\{\overline{1,2,3,d_2},\overline{4,5,6,d_1}\});\end{aligned}$$

$$\begin{aligned}\mathrm{EPT5} = (&\{\overline{1,4},\overline{2},\overline{3},\overline{5,6}\},\\ &\{\overline{1,6,9,d_2},\overline{2,5,7,d_1},\overline{3,4,8,d_3}\},\\ &\{\overline{1,2,3,d_2},\overline{4,5,6,d_1}\});\end{aligned}$$

$$\begin{aligned}\mathrm{EPT6} = (&\{\overline{1,4},\overline{2},\overline{3},\overline{5},\overline{6}\},\\ &\{\overline{1,6,9,d_2},\overline{2,5,7,d_1},\overline{3,4,8,d_3}\},\\ &\{\overline{1,2,3,d_2},\overline{4,5,6,d_1}\});\end{aligned}$$

$$\begin{aligned}\mathrm{EPT7} = (&\{\overline{1,4},\overline{2,3},\overline{5},\overline{6}\},\\ &\{\overline{1,6,9,d_2},\overline{2,5,7,d_1},\overline{3,4,8,d_3}\},\\ &\{\overline{1,2,3,d_2},\overline{4,5,6,d_1}\});\end{aligned}$$

Unfortunately, within this set there do not exist two EPT's such that their trinity product is a zero trinity. This means that we cannot find a parallel full-decomposition of the machine. But, for the existence of EPT's, it may be possible to find a serial full-decomposition. We now try to do so.

We take the largest EPT in qusetion, EPT1=(π_I,π_S,π_O), because a larger EPT usually gives us a simpler image machine.

Step 2. Find a forced-EPT.

We take tri-partition

$$(\tau_I,\tau_S,\tau_O) = (\{\overline{1,3,5},\ \overline{2,4,6}\}$$
$$\{\overline{1,5,8,d_3},\overline{2,4,9,d_2},\overline{3,6,7,d_1}\}$$
$$\{\overline{1,4,d_1},\overline{2,5},\overline{3,6,d_2}\})$$

as a candidate and examine if it is a forced-EPT under the forcing-partition

$$\bar{\pi}_S = \{\overline{1,6,9},\overline{2,5,7},\overline{3,4,8}\}.$$

Let
$$\tau_S = \{\overline{1,5,8,d_3},\overline{2,4,9,d_2},\overline{3,6,7,d_1}\} = \{A,B,C\}$$
$$\tau_O = \{\overline{1,4,d_1},\overline{2,5},\overline{3,6,d_2}\}) = \{x,y,z\}$$
$$\tau_I = (\{\overline{1,3,5},\overline{2,4,6}\} = \{a,b\}$$
$$\bar{\pi}_S = \{\overline{1,6,9},\overline{2,5,7},\overline{3,4,8}\} = \{M,N,P\}$$
$$\tau = \pi_O = \{\overline{1,2,3,d_2},\overline{4,5,6,d_1}\} = \{\alpha,\beta\}$$

Substituting them into the transition table of machine B, we have

$$V_{M,1}=V_{M,5}=V_{N,3}=V_{P,3}=V_{P,5}=(C/x,B/z,A/x)$$
$$V_{M,2}=V_{N,4}=V_{N,6}=V_{P,4}= (A/z,C/y,C/x)$$
$$V_{M,1}=V_{N,3}=V_{N,5}=V_{P,1}= (B/y,C/x,B/x)$$
$$V_{N,2}=V_{M,4}=V_{M,6}=V_{P,2}=V_{P,6}= (C/y,B/z,A/y)$$

which satisfy

i) $\pi_S \cdot \tau_S = \pi_S(0)$

ii) for any $i,j \in I$, $B',B'' \in \pi_S$,

$$[i]\tau_I = [j]\tau_I \wedge V^{\pi_O}_{B',i} \simeq V^{\pi_O}_{B'',j} \quad (\tau_S)$$
$$\Rightarrow V_{B',i} \simeq V_{B'',j} \quad (\tau_S)$$

where

$$V_{\pi_S \times a} = \{V_{M,1},V_{N,1}\}$$
$$V_{\pi_S \times b} = \{V_{M,2},V_{N,2}\}.$$

It is said that (τ_I,τ_S,τ_O) is a forced-EPT under the forcing-partition $\tau=\pi_O$.

Step 3. Set up image machine Q_1.
By the substitution of

$$\pi_S = \{\overline{1,6,9,d_2},\overline{2,5,7,d_1},\overline{3,4,8,d_3}\} = \{M,N,P\}$$
$$\pi_I = \{\overline{1,4},\overline{2,3},\overline{5,6}\} = \{m,n,p\} \text{ and}$$
$$\pi_O = \{\overline{1,2,3,d_2},\overline{4,5,6,d_1}\} = \{\alpha,\beta\}$$

and the computation of δ^1 and λ^1 on the blocks, such as

$$M\delta^1_m = [M\bar{\delta}_m]\pi_S = N,$$

$$M\lambda^1_m = [M\bar{\lambda}_m]\pi_O = \alpha,$$

and so on, the image machine Q_1 is obtained is shown in Fig. 7.5.

	m	n	p
M	N/α	N/β	P/α
N	N/β	P/α	M/β
P	M/β	N/α	P/α

Fig. 7.5 Machine Q_1

Step 4. Set up image machine Q_2.

The four vectors obtained in step 2 will construct the image machine of the forced-EPT with the following output assignments in the inputs:

$V_{M,1} \rightarrow (\alpha,a)$ because $M\bar{\lambda}_1 \subseteq \alpha$ and $1 \epsilon a$

$V_{M,2} \rightarrow (\beta,b)$ because $M\bar{\lambda}_2 \subseteq \beta$ and $2 \epsilon b$

$V_{N,1} \rightarrow (\beta,a)$ because $N\bar{\lambda}_1 \subseteq \beta$ and $1 \epsilon a$

$V_{N,2} \rightarrow (\alpha,b)$ because $N\bar{\lambda}_2 \subseteq \alpha$ and $2 \epsilon b$

the image machine Q_2 is shown in Fig. 7.6.

	(α,a)	(α,b)	(β,a)	(β,b)
A	C/x	C/y	B/y	A/z
B	A/z	B/z	C/x	C/y
C	B/x	A/y	B/x	C/x

Fig. 7.6 Machine Q_2

Step 5. The mappings between machine Q and machine $Q_1 \rightarrow Q_2$ are listed as follows.

$S \rightarrow S_1 \times S_2$		$I \rightarrow I_1 \times I_2$		$O \rightarrow O_1 \times O_2$	
1 →	(M,A)	1 →	(m,a)	1 →	(α,x)
2 →	(N,B)	2 →	(n,b)	2 →	(α,y)
3 →	(P,C)	3 →	(n,a)	3 →	(α,z)
4 →	(P,B)	4 →	(m,b)	4 →	(β,x)
5 →	(N,A)	5 →	(p,a)	5 →	(β,y)
6 →	(M,C)	6 →	(p,b)	6 →	(β,z)
7 →	(N,C)				
8 →	(P,A)				
9 →	(M,B)				

(End of Example 7.2)

CHAPTER 8

COMPUTER AIDED DECOMPOSOTIONS

During the study of the decomposition of sequential machines there was an extensive support of a computer. This helped the rapid progress of this study. In this chapter, we will discuss a series of algorithms for the decompositions of sequential machines. The algorithms are applied in a program package in which we can calculate most of the functions and properties, such as partitions, partition pairs, partition trinities, and full-decompositions of sequential machines (see Appendix).

In Section 8.1, we will describe the data structure used. Section 8.2 discusses the algorithms for basic operations in the decomposition theory.

8.1 Data Structure

In the study of machine decompositions, the only input data was a table which described the state transitions and outputs of a machine. For the table, we made the following stipulations for the programming.

Expressing Form

For the sake of simplifying the program design and management, we defined the data of state, input and output with an expression form as follows:

State set: $S = \{0,1,2,...,NS\}$
Input set: $I = \{1,2,...,NI\}$
Output set: $O = \{0,1,2,...,NO\}$

where NS is the number of states;
NI is the number of inputs;
NO is the number of outputs.

The element 0 denoted a "don't care" condition. Also, NS, NI and NO ware used as global variables to express the numbers of states, inputs and outputs for different sizes of machines within whole descriptions of algorithms and all programs.

Storage Form

We arranged two arrays δ[] and λ[], with sizes NSxNI, to record the next states and outputs of any machine to be studied. The arrays were set up by a special procedure in one of two ways, one from a keyboard input and another from a floppy disk input. In the mode of keyboard input, the procedure accepted the data and wrote it on the disk and in arrays of memory. In the mode of a floppy disk input, the procedure read the data from floppy disk into the arrays of memory. The data on floppy disk was also written in the Editor mode and was of the following format:

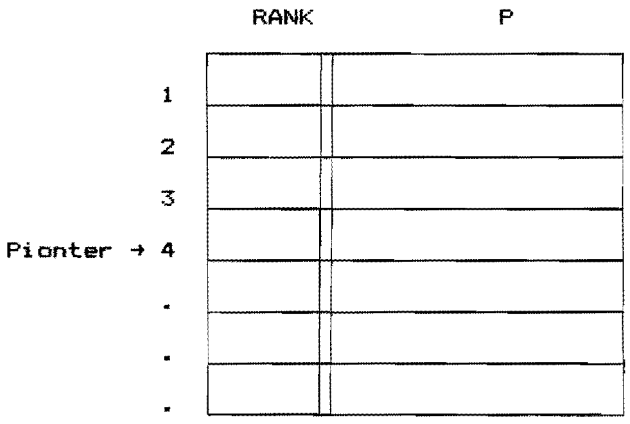
machine type
basic parameters
(next state, output)

where the machine type was a number expressing Moore machine with 0 or Mealy machine with 1; basic parameters were composed of three integer numbers NS, NI and NO in order; the last part (2x)NSxNI numbers of the next states and outputs separated by a space and positioned according to the original machine table. The advantage of the design was that we could make use of Editor mode to input the data off-line.

Dynamic storage form

Based on the arrays a running program produced derived data or results, such as partitions, partition pairs, or partition trinities. And some of these data might be used as input data for another program with other functions. Therefore, a dynamic data structure should be arranged for this kind of requirement. For simplicity we chose partitions as the cells of the dynamic data structure. Other forms of data could be obtained by combining cells in a particular program. For instance, two cells consisted of a partition pair and three cells for partition trinity. In practice, we used the following two types of structures.

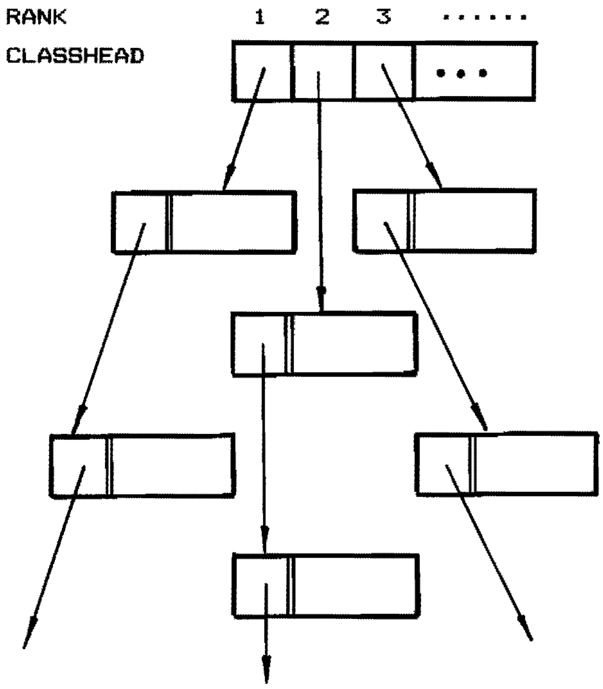
A. Ordered linked list.



In this type of structure, each item consists of two parts. One part was P, which was an integral array to express a partition π. Another part was RANK, which gave the number of blocks in the partitions. Because there were many comparison operations of partitions in a program and because of the property that two partitions with unidentical RANK numbers were certainly not equivalent, it was shown that the arrangement of RANK made a large benefit in simplifying programming and fast computation. There was a pointer to keep the position of the last used item.

B. Classed linked list.

In some programs, we used another type of structure while the number of items was very large so that the computation was time-consuming. We noted that, for any given machine, the number of different ranks was equal to NS (for state partition), NO (for output partition) or NI (for input partition). In order to speed up the procedure of searching partitions for the same rank, we made a classed link instead of RANK, which was shown as follows:



In the structure, part P is the same as in A. But part of RANK recorded the next position of partition in the class (RANK $\neq 0$) or the end of link of the class (RANK $= 0$). CLASSHEAD gave the first item in a class (by the content) and the number of blocks in the class (by index). There was also a pointer to indicate the next cell available for storing a new partition, partition pair, or PT. The description of data structure on P will be given in a special section later (see Section 8.2.1).

8.2 Algorithms of Basic Operations

Like in any mathematical system, there are also some basic operations in the algebraic theory of machine decompositions. They are partition addition, partition multiplication,

$\pi^m_{s,t}$, $m(\pi)$, $M(\pi)$, etc. All the other operations,

such as partition pair operations and partition trinity operations are built by the basic operations. In this section, we give a general description of the basic operations and discuss their computer algorithms.

8.2.1 Partition Function

In the study aided by a computer, we must look for a better form of storage and representation of the data (here, partitions) because it effects the computation complexity directly (space and time).

A direct way is to use a set to represent the partition, since the partition is a set of blocks each of which is a subset. In this way, for a partition on a set S which has N distinct elements, we define the following types:

```
block     = set of 1..N
partition = array [1..N] of block
```

Since a partition may contain N blocks (zero partition) and a block may contain N elements (identity partition) we have to define it with N. Thus, a partition takes $N \times N = N^2$ bits if we use one bit to represent one element in S. It is obvious that a partition needs too much space to do computations when set S is larger.

On the other hand, we consider an operation of partitions, say partition addition, under the above representation to examine the time complexity.

Let $\pi_1 = \{B_{1,1}, B_{1,2}, \ldots, B_{1,n}\}$;
$\pi_2 = \{B_{2,1}, B_{2,2}, \ldots, B_{2,m}\}$;

Firstly, we should do set addition on any two blocks in the two partition if they have at least one common element, Symbolically it is inductively described as follows:

Let $B'_{i,o} = B_{1,i}$

and for any j, $0<j<m$, let

$$B'_{i,j+1} = \begin{cases} B'_{i,j} \cup B_{2,j} & \text{if } B'_{i,j} \cap B_{2,j} \neq \emptyset \\ B'_{i,j} & \text{if } B'_{i,j} \cap B_{2,j} = \emptyset \end{cases} \qquad (1.1)$$

Since it is possible that there will be common elements in two different $B'_{k,m}$ and $B'_{i,m}$ of $\{B'_{i,m}\}$, we have to do a check and additions on $\{B'_{i,m}\}$ again, as in the above procedure, that is, let $B''_{i,o} = B'_{i,m}$
and for any $0<j\leq n$,

$$B''_{i,j+1} = \begin{cases} B''_{i,j} \cup B'_{j,m} & \text{if } B''_{i,j} \cap B'_{j,m} \neq \emptyset \\ B''_{i,j} & \text{if } B''_{i,j} \cap B'_{j,m} = \emptyset \end{cases} \qquad (1.2)$$

The similar procedure of (1.2) on the set $\{B''_{i,n}\}$ must be repeated until one of the following conditions is satisfied:

i) $B''_{k,n} \cap B''_{l,n} = \emptyset$; (1.3)

ii) $B''_{k,n} \cap B''_{l,n} = B''_{k,n} \wedge B''_{K,n} \cap B''_{l,n} = B''_{i,n}$

for any k,l ($k\neq l$), $1\leq k,l\leq n$.
Then, for any i, $B''_{i,n}$ is a block of $\pi_1+\pi_2$, that is,

$$\pi_1 + \pi_2 = \{B''_{i,n}\}$$

Here, to get $\{B'_{i,m}\}$ we have to do more than nxm times of set operations, and for $\{B'_{i,n}\}$ more than nxn times of set operations. Totally, to get $\pi_1 + \pi_2$ it takes

$$n \times m + k \times n \times n \simeq kN^2$$

times set operations where k represents the times we repeat the procedure on $\{B''_{i,n}\}$ for satisfying (1.3).

It is obvious that, as N becomes larger, the computation time will be so long that it is unacceptable in the cases when we must do a lot of partition additions on a larger set of partitions. The conclusion is that a better representation of the partitions is requred.

In the following discussion, we first study the mechanism of the structure of a partition and finally derive the general definition of a partition function.

Let $\tau_{i,j} = \{\overline{1},\overline{2},\ldots,\overline{i,j},\ldots,\overline{N}\}$ be a minimal partition on which only elements i and j belong to the same block. Then, for any a partition τ on S, we have

$$\tau = \sum \{\tau_{i,j} \mid [i]\tau=[j]\tau\} \tag{1.4}$$

where $\sum$ denotes repeated partition additions.

In (1.4), there are totally $N+C_N^2$ $\tau_{i,j}$ we have to examine.But if a check is made on $\{\tau_{i,j} \mid [i]\tau=[j]\tau\}$, we know, for any $i,j\in S$,

$[i]\tau=[j]\tau$ implies

$$\begin{array}{rl} \text{i)} & \tau_{i,i} \in \{\tau_{i,j} \mid [i]\tau=[j]\tau\}, \\ \text{ii)} & \tau_{j,j} \in \{\tau_{i,j} \mid [i]\tau=[j]\tau\}, \\ \text{iii)} & \tau_{i,j} \in \{\tau_{i,j} \mid [i]\tau=[j]\tau\}, \\ \text{vi)} & \tau_{j,i} \in \{\tau_{i,j} \mid [i]\tau=[j]\tau\}. \end{array} \tag{1.5}$$

But, for any $i,j\in S$ and for any π on S,

$$\begin{array}{l} \pi + \tau_{i,i} = \pi + \tau_{j,j} = \pi, \\ \pi + \tau_{i,j} = \pi + \tau_{j,i}. \end{array} \tag{1.6}$$

It is true that some of them are redundant. They are $\tau_{i,i}$, $\tau_{j,j}$, one of $\tau_{i,j}$ and $\tau_{j,i}$ and one of $\tau_{i,j}$, $\tau_{i,k}$ and $\tau_{i,k}$ to calculate τ by (1.4). In this case, we see that the additions, $\pi+\tau_{i,j}$ are trivial. Therefore, we need to make some restrictions to (1.4) in order to reduce the redundant information units. It is obvious that the restriction

$$i \neq j$$

can cut down (i) and (ii). And because S is defined as a set of integers, the restriction

$$i \geq j$$

can cut down (iv). Thus, (1.4) becomes

$$\tau = \sum \{\tau_{i,j} \mid i>j \wedge [i]\tau=[j]\tau\}. \tag{1.7}$$

In order to ensure the minimum amount of numbers of $\tau_{i,j}$ for building a partition, we consider the following lemma first.

LEMMA B.1

Let B be a block of τ on S and let B have m distinct elements, i.e. $|B|=m$. Then, we need at least m−1 $\tau_{i,j}$ to build B. In other words,

$$\tau_B = \{\overline{1},\overline{2},\ldots,\overline{B},\ldots,\overline{N}\}$$

$$= \sum^{m-1} \{\tau_{i,j} | i>j \wedge i,j\in B\} \qquad (1.8)$$

where $\sum^{m-1}$ means m−1 partition additions have to be done nontrivially.

Proof.

1) It is obvious that when n=1, $\tau_B=\pi(0)$, we need nothing to do it;

2) For n=2, n−1=1, since

$$\tau_B = \{\overline{1},\overline{2},\ldots,\overline{i,j},\ldots,\overline{N}\} = \tau_{i,j},$$

(1.8) holds.

3) Assume when n=m−1 (1.8) holds, that is

$$\tau'_B = \{\overline{1},\overline{2},\ldots,\overline{B'},\ldots,\overline{N}\}$$

$$= \sum^{m-1-1} \{\tau_{i,j} | i>j \wedge i,j\in B\}.$$

Then, for n=m−1, suppose

$B-B'=\{k\}$, i.e. $B=B'\cup\{k\}$. .

We know that one more minimal partition is enough to build G_B form τ'_B since, for some $i\in B'$,

$$\tau_B = \tau'_B + \tau_{i,k} \quad \text{if} \quad i>k;$$

$$\tau_B = \tau'_B + \tau_{k,i} \quad \text{if} \quad i<k.$$

Thus,

$$\tau_B = \sum^{m-1} \{\tau_{i,j} | i>j \wedge i,j\in B\}$$

(End of Lemma B.1)

Based upon the Lemma, we have

THEOREM 8.1

Let τ be a partition on S, then there are at least $N-|\tau|$ $\tau_{i,j}$ to build τ that is,

$$\tau = \sum^{N-|\tau|} \{\tau_{i,j} | i>j \wedge [i]\tau=[j]\tau\}. \tag{1.9}$$

proof:

Case 1: $\tau = \pi(O)$, $N-|\tau| = 0$, we need not do anything for τ;

Case 2: $\tau = \pi(I)$, $N-|\tau| = N-1$, following Lemma 8.1

Case 3: $\tau \neq \pi(O)$, and $\tau \neq \pi(I)$. From Lemma 8.1, for each block B_k in τ, we need $|B_k|-1$ of $\tau_{i,j}$. Thus, for τ we need

$$\sum_{k=1}^{|\tau|} (|B_k| - 1) = \sum_{k=1}^{|\tau|} |B_k| - |\tau| = N-|\tau|$$

pieces of $\tau_{i,j}$.

(End of Theorem 8.1)

If we consider each $\tau_{i,j}$ as an *information unit*, from the theorem a corollary is obtained.

COROLLARY 8.1

To represent any a partition τ on S by $\tau_{i,j}$ we need at least N information units.

Proof:

From Theorem 8.1, we know that, for any non-zero partition, we need at least $N-|\tau|$ information units. But for the zero partition $\pi(O)$,

$$\pi(O) = \sum \{\tau_{i,j} | i=j\}$$

which needs N information units to represent it.

Thus, in order to represent any partition on S by $\tau_{i,j}$, we have to have at least N information units.

(End of Corollary 8.1)

Now, we should consider how to select $N-|\tau|$ $\tau_{i,j}$ which perfectly construct τ. Firstly, examining (1.8) we know in $\{\tau_{i,j} | i>j \wedge i,j\in B\}$ there are

$$\sum_{k=1}^{m} (k-1)$$

distinct $\tau_{i,j}$. But for some $i,j,k \in B$, $i \neq j \neq k$, there exists certainly an order on i,j and k. Suppose the order is

$$i > j > k.$$

Then, clearly,

$$\tau_{i,j},\ \tau_{i,k},\ \tau_{j,k} \in \{\tau_{i,j} \mid i > j \wedge i,j \in B\}.$$

Since

$$\tau_{i,j} + \tau_{i,k} + \tau_{j,k} = \tau_{i,j} + \tau_{j,k},$$

or $\tau_{i,j} + \tau_{i,k} + \tau_{j,k} = \tau_{i,j} + \tau_{i,k},$

or $\tau_{i,j} + \tau_{i,k} + \tau_{j,k} = \tau_{j,k} + \tau_{i,k}.$

This means that one information unit is redundant. In order to remove the redundant one we must introduce the restriction "*only take one i in* $\{\tau_{i,j} \mid i > j \wedge i,j \in B\}$", which is realized by

$$\tau_B = \sum_{i=1}^{N} \{\tau_{i,j} \mid i,j \in B \wedge i = j\}.$$

Because blocks of τ are disjointed (1.7) becomes

$$\tau = \sum_{i=1}^{N} \{\tau_{i,j} \mid i > j \wedge [i]\tau = [j]\tau\}.$$

This states that we only take the $\tau_{i,j}$ with different i to build τ.

With the N information units for any non-zero partition τ, there are $|\tau|$ redundant information units. For them, we only take those $\tau_{i,j}$ such that i is the minimum element in the block which contains i in order to make it coincident on both non-zero and zero partitions. Therefore, any partition can be built by

$$\tau = \sum_{i=1}^{N} \{\tau_{i,j} \mid (i > j) \wedge [i]\tau = [j]\tau \vee (i = j) \wedge (i = \min(B(i)))\} \qquad (1.10)$$

where $i=\min(B(i))$ means that i is the smallest element in the block containing i. Although we have $|\tau|$ redundant units for the representation of non-zero partitions, we will see later that it is very convenient for the operations of partitions.

So far, we have divided any partition on S into N information units each of them meets $i \geq j$. Since, in each $\tau_{i,j}$ only two parameters, i and j, are involved, we can use a very simple form of representation to indicate the character of $\tau_{i,j}$: only i and j in a block. An obvious way to do this is to use an array in which the index is i and the value of index i is j. Consequently, a function is defined as follows:

<u>DEFINITION 8.1</u>

Let π be a partition on S. P_π is a p-function of π if P_π maps S into S by the following rule:

$P_\pi(s)=s$ *if and only if* $\forall t\in S$: $[s]\pi=[t]\pi \Rightarrow t\geq s$;
$P_\pi(s)\neq s$ *if and only if* $\exists P_\pi(s)\in S$: $[s]\pi=[P_\pi(s)]\pi \Rightarrow s\geq P_\pi(s)$.

(End of Definition 8.1)

If we make a comparison on a partition of a set and an undirected graph, we know fortunately that the p-function is equivalent to the f-function invented by Rem[4]. This is because, if we consider the elements of a set of a sequential machine as the vertices of an undirected graph, a block of the partition just is a connected subgraph. Therefore, the Rem algorithm can be directly used later in the discussions of algorithms of basic operations in machine decomposition theory.

By definition, a p-function of a partition π portraies vividly the block characters of the partition with the following properties:

1) for any $s\in S$, $1\leq P_\pi(s)\leq s$;
2) any block *has one and only one* identifying element s with $P_\pi(s)=s$;
3) for any $s,t\in S$
   $[s]\pi = [t]\pi$ *if and only if* $id(s)=id(t)\,(P_\pi)$;
4) π is zero partition *if and only if* $id(P_\pi)^{\#} = N$;
5) π is identity partition *if and only if* $id(P_\pi)^{\#} = 1$;
6) for any π, τ on S

   $\pi\geq\tau \Rightarrow id(P_\pi)^{\#} \leq id(P_\tau)^{\#}$;

7) π has more than one different p-function *if and only if* $\max|B_i| >2$, $B_i\in\pi$;

where i) an identifying element is an element s such that $P_{\pi}(s)=s$;

ii) id(s) denotes the identifying element which comes from that there is a finite sequence of 1..i, $1\le i\le|s|$,

$$id(s)= P_{\pi}^{i}(P_{\pi}^{i-1}(....(P_{\pi}^{1}(s))...))$$

such that $P_{\pi}^{i+1}(s)=P_{\pi}^{i}(s)$;

iii) $id(P_{\pi})^{\#}$ denotes the number of distinct identifying elements in P_{π}.

From the definition, we know that a partition function takes NxL bits, where L is the length of words in a computer, and that where N>L, a partition function gives a great advantage for the space requirement. We should also note that in the case of using packed array, a partition function only takes $N\times\log_2 N$ bits for its storage. An implementation of partiton functions is defined with the following two types:

```
STYPE = 1..N
PTYPE = array[1..N] of STYPE.
```

8.2.2 Partition Addition

8.2.2.1 A Method for $\pi_1+\pi_2$ by Hand

A method for calculating the partition sum $\pi_1+\pi_2$ by hand, like normal form on compact computation on decimal numbers, is presented here. In this method, firstly, we draw a table in which each column denotes an element of the set and each line denotes a block of π_1 or of π_2. Secondly, we fill entries in the table in this way: if element j belongs to some block i, then we put a x in column j on the row in which the block is located. Thirdly, we calculate the partition sum by the following procedure:

PROCEDURE 8.1

1. Take a column i without any symbol of its head; put a line on column i and a new symbol on the head of column i;
2. If row j has a x on column i, put a line on row j;
3. If column k ($k \neq i$) has a x on row j, put the same symbol on the head of column k;
4. For all rows with a x on column i, repeat 2 and 3 again;
5. For all columns with a x on row j, repeat 3 and 4 again;
6. Repeat 1-5 until the heads of all columns have symbols;
7. The elements with the same symbols form a block of $\pi_1+\pi_2$

(End of Procedure 8.1)

To illustrate the procedure an example is given as follows:

EXAMPLE 8.1

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

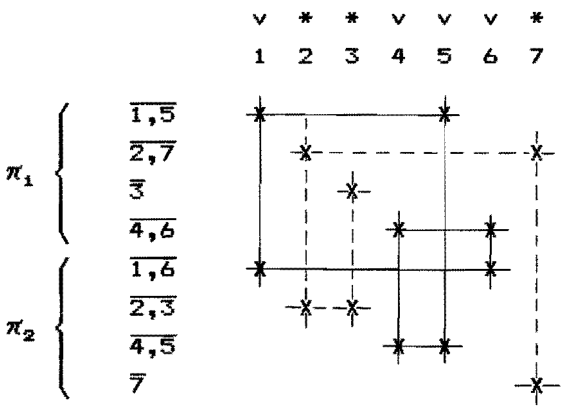
$\pi_2 = \{\overline{1,6},\overline{2,3},\overline{4,5,7}\}$

be two partitions on the set

$S = \{1,2,3,4,5,6,7\}$

By Procedure 8.1, a compact form for calculating partition $\pi_1+\pi_2$ is given in Fig. 8.1.

(End of Example 8.1)



$\pi_1+\pi_2 = \{\overline{1,4,5,6},\overline{2,3,7}\}$

Fig. 8.1 $\pi_1+\pi_2$

In the table, each vertical line indicates the blocks with common elements and each horizontal line indicates a subset of block of $\pi_1+\pi_2$. Since we check all subsets of the block, a correct result is obtained. Because we do possibly many partition additions on a small set during a study, the method mentioned above presents a convenient and reliable way to do them by hand on paper.

A little more should be added when we calculate the partition sum of more than two partitions the procedure shows a big advantage for a convenient computation.

8.2.2.2 Partition Sum P_1+P_2

Now, we consider how to do partition addition based on two P-functions. This means that from P_1 and P_2 of π_1 and π_2, respectively, how to do we can get a P_3 which is a p-function of $\pi_3 = \pi_1 + \pi_2$.

By the concept of information units we know, for any π, τ on S

$$\pi+\tau = \pi + \sum_{i=1}^{N} \{\tau_{i,j} \,|\, (i>j) \wedge ([i]\tau=[j]\tau) \vee (i=j) \wedge (i=\min(B(i)))\}$$

Since $\pi+\tau+\pi = \pi+\tau$, we also know

$$\pi+\tau = \sum_{i=1}^{N} \{\pi+\tau_{i,j} \,|\, (i>j) \wedge ([i]\tau=[j]\tau) \vee (i=j) \wedge (i=\min(B(i)))\}$$

This states that we merge continually two blocks B(i) and B(j) in π if i and j are in the same block of τ. Comparing with an undirected graph G, $\pi+\tau_{i,j}$, here, is equivalent to *"making a new edge between vertices i and j to the graph G"*. For this, Rem presented a beautiful algorithm [4] based on f-function representation of a graph, which can be directly used in our problem and is described as follows:

```
Algorithm  NEWEDGE (var P:PTYPE; s,t:STYPE);
input:  p-function P of $\pi$ and elements s and t of $\tau_{s,t}$;
output: p-function P of $\pi+\tau_{s,t}$;
procedure:
  begin var $s_0,t_0,s_1,t_1$: STYPE;
    $s_0,t_0$ := s,t;
    $s_1,t_1$ := P(s),P(t);
    do $s_1<t_1$ → P($t_0$) := $s_1$; $t_0,t_1$ := $t_1$,P($t_1$);
     ‖ $t_1<s_1$ → P($s_0$) := $t_1$; $s_0,s_1$ := $s_1$,P($s_1$);
    od
  end
```

To honour the inventor, we give the name NEWEDGE for its application in our problem. NEWEDGE realizes the merge of two blocks which contain elements s and t respectively by reassigning the values of p-function of elements from s to id(s) and t to id(t) and finally meeting

$$id(s) = id(t) = min(id(s),id(t)) .$$

Secondly, we consider how to use NEWEDGE to calculate $\pi+\tau$.
For $P_3 = P_1+P_2$ we initialize it into P_1, that is

$P_3= P_1$

realized by

```
i := 1;
do i≤N → i,$P_3$(i) := i+1,P(i) od.
```

In order to do $\pi + \tau_{i,j}$ we call the procedure NEWEDGE by

NEWEDGE(P_3,i,P_2(i)).

But, because there is some redundant $\tau_{i,j}$ in P_2 on which $\pi+\tau_{i,j}$ is trivial, we should give up the operation on $\tau_{i,j}$.
This is done by

```
if i≠ $P_2$(i) →  NEWEDGE($P_3$,i,$P_2$(i)) fi.
```

The procedure has to be repeated for all

$$\tau_{i,j}\epsilon\{\tau_{i,j}|(i>j) \wedge [i]\tau=[j]\tau\},$$

which is realized by examining all P_2(i) in P_2, that is,

```
do I≤N → if i≠$P_2$(i) → NEWEDGE($P_3$,i,$P_2$(i)) fi od.
```

Finally, a completed algorithm for calculating P_1+P_2 is obtained as follows:

```
Algorithm  SUMP(var $P_1$,$P_2$,$P_3$: PTYPE; N: STYPE);
input : Partition functions $P_1$ and $P_2$, partition N;
output: Partition sum $P_3$= $P_1$+$P_2$;
procedure:
  begin
    begin var i : integer;
      i := 1;
      do i≤N → i, $P_3$(i) := i+1, P(i) od
    end
    begin var i : integer;
      i := 1;
      do i≤N →  if i≠$P_2$(i) → NEWEDGE($P_3$,i,$P_2$(i)); i :=i+1;
                 ▯ i=$P_2$(i) → i := i+1
                fi
      od
    end
  end
```

In the algorithm a variable N is arranged by making it suitable to any type of partitions, such as state partitions, input partition or output partition, on which N=NS, N=NI or N=NO.

8.2.3 Partition Product $P_1 \cdot P_2$

Let π, τ be partitions on S. then, based on the definition of partition product, we have

$$\pi \cdot \tau = \sum_{i=1}^{N} \{\pi_{i,j} \mid (i>j) \wedge [i]\pi=[j]\pi\} \cdot \sum_{i=1}^{N} \{\tau_{i,j} \mid (i>j) \wedge ([i]\tau=[j]\tau\}$$

$$= \sum_{i=1}^{N} \{\pi_{i,j} \mid (i>j) \wedge ([i]\pi=[j]\pi) \wedge ([i]\tau=[j]\tau)\}$$

It tells us the main thing to do in the operation is to judge each $\pi_{i,j}$ if there is a $\tau_{i,j}$ in τ. Consequently, we should develop a function to do this.
As we know, for any i,j∈S,

$[i]\pi = [j]\pi$ *if and only if* id(i)=id(j)(Pπ)

in Pπ. Therefore, a function IJLINKED is written easily as follows:

```
Algorithm  IJLINKED(var P: PTYPE; I,J: STYPE) : Boolean;
input : p-function P; elements I and J ,
output: Boolean function IJLINKED = true if id(I)=id(J)
          else IJLINKED = false
procedure:
  begin var $I_0$,$J_0$ : STYPE;
    $I_0$,$J_0$ := I,J;
    do $I_0 \neq P(I_0) \rightarrow I_0 := P(I_0)$ od;
    do $J_0 \neq P(J_0) \rightarrow J_0 := P(J_0)$ od;
    IJLINKED := ($I_0$=$J_0$)
  end
```

Now, using the function we can write down the procedure for calculating $P_1 \cdot P_2$.

```
Algorithm  XP(var $P_1$,$P_2$,$P_3$: PTYPE; N: STYPE);
input : p-functions $P_1$ and $P_2$; partition type N;
output: $P_3=P_1 \cdot P_2$
procedure:
  begin
    begin var i: integer;
      i := 1;
      do $i \leq N \rightarrow i,P_3(i) := i+1,i$ od
    end;
    begin var i,j: integer;
      i := 1;
      do $i \leq N-1 \rightarrow j := i+1$;
         do $j \leq N \rightarrow$
            if $id(i)=id(j)(P_1) \wedge id(i)=id(j)(P_2) \rightarrow$
                NEWEDGE ($P_3$,i,j); j := j+1
             ▯ $id(i) \neq id(j)(P_1) \vee id(i) \neq id(j)(P_2) \rightarrow j := j+1$
            fi
         od; i := i+1
      od
    end
  end
```

The relation $id(i)=id(P_1(i))(P_2)$ is done by

IJLINKED(P_2,i,P_1(i)).

To understand the algorithm conveniently we write IJLINKED(P_2,i,P_1(i)) by the form of id(i)=id(P_1(i))(P_2).

8.2.4 $\pi^m_{s,t}$

<u>DEFINITION 8.2</u>

State pair (s',t') is a relative state pair of state pair (s,t) *if and only if* there exists a $x \in I^*$ such that

$$(s,t)\vec{\delta}_x = (s',t'). \tag{4.1}$$

(End of Definition 8.2)

For any pair (s,t), its relative pairs form a set $R_{s,t}$,

$$R_{s,t} = \{(s',t') \mid (s',t') \text{ is a relative pair of } (s,t)\}. \tag{4.2}$$

The pair (s,t), obviously, is in $R_{s,t}$ since for an empty input ε

$$(s,t)\vec{\delta}_\varepsilon = (s,t).$$

by Property 2.11.
Then, for any $s,t \in S$, their smallest SP partition $\pi^m_{s,t}$ is calculated by

$$\pi^m_{s,t} = \sum \{ \pi_{i,j} \mid (i,j) \in R_{s,t} \}. \tag{4.3}$$

Now, The things to do are to find (s',t') and to record it in $R_{s,t}$. We define a p-function P to record $R_{s,t}$ with the initial value

P = a p-function of $\pi_{s,t}$.

When a (s',t') is obtained, it is recorded by

NEWEDGE(P,s',t').

Once we get all $(s',t') \in R_{s,t}$, the final value of P just is a p-function of $\pi^m_{s,t}$, that is,

P = a p-function of $\pi^m_{s,t}$

To find a $(s',t') \in R_{s,t}$, we start from (s,t), for all $i \in I$, the next states

$$(s,t)\vec{\delta}_i \in R_{s,t}.$$

Generally speaking, if $(s',t') \in R_{s,t}$, for all $i \in I$, $(s',t')\vec{\delta}_i \in R_{s,t}$

and for any $(s',t')\vec{\delta}_i$ we must record it in P by

NEWEDGE(P,δ[s',i],δ[t',i]

where δ[] denotes the array for transition table of a machine; another thing to do is to find continuely that for all $j \in I$,

$$(s',t')\vec{\delta}_i\vec{\delta}_j \in R_{i,j}.$$

The procedure should be repeated until all (s',t') are checked on all $j \in I$. Consequently, a recursive procedure is yielded as follows.

```
Algorithm  NEWPAIR(var P: PTYPE; $s_0,t_0$: STYPE);
input : states s and t; array δ;
output : p-function P of $R_{s,t}$
Procedure :
begin var i: integer;
  i := 1;
  do  $i \leq NI$ →
        if δ[$s_0$,i]≠δ[$t_0$,i] ∧ id(δ[$s_0$,i])≠id(δ[$t_0$,i])(P) →
                NEWEDGE(P,δ[$s_0$,i],δ[$t_0$,i]);
                NEWPAIR(P,δ[$s_0$,i],δ[$t_0$,i]);
                i := i+1;
         | δ[$s_0$,i]=δ[$t_0$,i] ∨ id(δ[$s_0$,i])=id(δ[$t_0$,i])(P) →
                i := i+1;
        fi
  od
end
```

Here the restriction δ[s_0,i]≠δ[t_0,i] is presented from

$$\pi+\tau_{i,i} = \pi$$

and id(δ[s_0,i])≠id(δ[t_0,i])(P) from

$$\pi+\tau_{i,j}+\tau_{j,k}=\pi+\tau_{i,j}+\tau_{j,k}+\tau_{i,k}$$

which guarantee that for any $s,t \in S$, the NEWPAIR is called

$$NS - |\pi^m_{s,t}|$$

times. Thus, the maximum number of calling NEWPAIR is NS−1 only when $\pi^m_{s,t} = \pi(I)$.
So far, an algorithm for $\pi^m_{s,t}$ is written easily based on the

procedure NEWPAIR.

```
Algorithm $\pi^m_{s,t}$(var P: PTYPE; s,t: STYPE);
input: states s and t; array $\delta$[];
output: a p-function P of $\pi^m_{s,t}$;
procedure:
  begin var i: integer;
    i :=1;
    do $i \le NS \to i,P(i) := i+1,i$ od;
    if $s>t \to P(s) := t$;
     ▯ $t>s \to P(t) := s$;
    fi;
    NEWPAIR(P,s,t)
  end
```

8.2.5 m(π)

To compute m(π) we consider first

$$\pi = \sum_{i=1}^{N} \{\pi_{i,j} \mid i>j \wedge [i]\pi=[j]\pi\} \tag{5.1}$$

By Theorem 3.1 in [15],

$$m(\pi_1+\pi_2) = m(\pi_1)+m(\pi_2)$$

We have

$$m(\pi)=m(\sum_{i=1}^{N} \{\pi_{i,j} \mid i>j \wedge [i]\pi=[j]\pi\})$$

$$= \sum_{i=1}^{N} \{m(\pi_{i,j}) \mid i>j \wedge [i]\pi=[j]\pi\} \tag{5.2}$$

Now, the problem is how to do for $m(\pi_{i,j})$ after easily getting $\pi_{i,j}$ in p-function of π. In $\pi_{i,j}$ there are only two elements, i and j, that should be considered. According to the definition of m operation it is obvious that

$$m(\pi_{i,j}) = \sum_{i=1}^{NS} \{\tau_{i\delta_k, j\delta_k} | \text{for all } k\in I\} \qquad (5.3.a)$$

$$m(\pi_{i,j}) = \sum_{i=1}^{NI} \{\tau_{k\delta_i, k\delta_j} | \text{for all } k\in S\} \qquad (5.3.b)$$

$$m(\pi_{i,j}) = \sum_{i=1}^{NI} \{\tau_{k\lambda_i, k\lambda_j} | \text{for all } k\in S\} \qquad (5.3.c)$$

$$m(\pi_{i,j}) = \sum_{i=1}^{NS} \{\tau_{i\lambda_k, j\lambda_k} | \text{for all } k\in I\} \qquad (5.3.d)$$

for S-S, I-S, I-O, or S-O respectively.
Let P_1 be a p-function of π and P_2 be a p-function of $m(\pi)$.
Then, for (5.3) we can realize them by

```
k := 1
if S-S pair → do k≤NI → NEWEDGE($P_2$,δ[i,k],δ[$P_1$(i),k]);
                        k := k+1
              od
 | I-S pair → do k≤NS → NEWEDGE($P_2$,δ[k,i],δ[k,$P_1$(i)]);
                        k := k+1
              od
 | S-O pair → do k≤NI → NEWEDGE($P_2$,λ[i,k],λ[$P_1$(i),k]);
                        k := k+1
              od
 | I-O pair → do k≤NS → NEWEDGE($P_2$,λ[k,i],λ[k,$P_1$(i)]);
                        k := k+1
              od
fi
```

where λ[] expresses the array for output table of a machine.
If the computations are repeated for all i in P_1, a p-function P_2 of $m(\pi)$ is obtained finally, which is described by the following algorithm:

```
Algorithm m(π) (var P₁,P₂: PTYPE; PT: string);
input:  p-function of π; pair type PT; arraies δ[] an λ[]
output: p-function of m_{S-S}(π), m_{I-S}(π), m_{S-O}(π), or m_{I-O}(π)
procedure:
begin var i,j,k,n₁,n₂,n₃: integer;
 if PT='S-S'→ n₁,n₂,n₃, := NS,NI,NS
  ▯ PT='I-S'→ n₁,n₂,n₃, := NI,NS,NS
  ▯ PT='S-O'→ n₁,n₂,n₃, := NS,NI,NO
  ▯ PT='I-O'→ n₁,n₂,n₃, := NI,NS,NO
 fi;
 i := 1;
 do i≤n₃ → i,P₂(i) := i+1,i od;
 i := 1;
 do i≤n₁ →
    if i≠P₁(i) → k := 1;
       do k≤n₂ →
          if PT = 'S-S'→
             if δ[i,k]≠δ[P₁(i),k] → NEWEDGE(P₂,δ[i,k],δ[P₁(i),k])
              ▯ δ[i,k]=δ[P₁(i),k] → skip
             fi
           ▯ PT = 'I-S'→
             if δ[k,i]≠δ[k,P₁(i)] → NEWEDGE(P₂,δ[k,i],δ[k,P₁(i)])
              ▯ δ[k,i]=δ[k,P₁(i)] → skip
             fi
           ▯ PT = 'S-O'→
             if λ[i,k]≠λ[P₁(i),k] → NEWEDGE(P₂,λ[i,k],λ[P₁(i),k])
              ▯ λ[i,k]=λ[P₁(i),k] → skip
             fi
           ▯ PT = 'I-O'→
             if λ[k,i]≠λ[k,P₁(i)] → NEWEDGE(P₂,λ[k,i],λ[k,P₁(i)])
              ▯ λ[k,i]=λ[k,P₁(i)] → skip
             fi
          fi;
          k := k+1
       od
     ▯ i=P₁(i) → skip
    fi;
    i := i+1
 od
end
```

8.2.6 $M(\pi)$

To compute $M(\pi)$ means that for a given partition π to make sure each $\tau_{i,j}$ such that

$$\tau = M(\pi) = \sum_{i=1}^{N}\{\tau_{i,j} \mid i>j \wedge [i]_{M(\pi)}=[j]_{M(\pi)}\} \qquad (6.1)$$

Under the case of using p-function it is for every i in P_2 of $M(\pi)$ to find one and only one j such that

$i>j$ and $[i]_{M(\pi)} = [j]_{M(\pi)}$.

For the restriction $i>j$ it is guaranteed by searching some j less than i. But, for $[i]_{M(\pi)}=[j]_{M(\pi)}$, by the definition of $M(\pi)$, it means for all $k \in NI$, $[i\delta_k]\pi=[j\delta_k]\pi$, (for M_{S-S}). That is

$[i]_{M(\pi)}=[j]_{M(\pi)}$ iff $[i\delta_k]\pi=[j\delta_k]\pi$

for all $k \in NI$, which is translated by

$P_2(i) = j$ iff $id(\delta[i,k])=id(\delta[j,k])(P_1)$

for all $k \in NI$.

Similarly, we can establish the judgements for other types of M operations as follows:

For $M_{I-S}(\pi)$

$P_2(i) = j$ iff $id(\delta[k,i])=id(\delta[k,j])(P_1)$

for all $k \in NS$;

for $M_{S-O}(\pi)$

$P_2(i) = j$ iff $id(\lambda[i,k])=id(\lambda[j,k])(P_1)$

for all $k \in NI$;

for $M_{I-O}(\pi)$

$P_2(i) = j$ iff $id(\lambda[k,i])=id(\lambda[k,j])(P_1)$

for all $k \in NS$.

When a k is found, so that $id(\delta[i,k]) \neq id(\delta[j,k])(P_1)$ the checks for other k's should be stopped. We give a controlling boolean variable EQ to record it provided EQ is false we can stop the checking immediately.

Also because only one j is needed for the $P_2(i)$ we give another controlling boolean variable FIND to indicate if or if not $[i]\pi=[j]\pi$. Once FIND is true we can stop the searching for other smaller j immediately.

With the considerations above an algorithm is naturally yielded as follows:

```
Algorithm M($\pi$) (var $P_1$,$P_2$: PTYPE; PT: string);
*input:*   p-function $P_1$ of $\tau$, pair type PT; $\delta$[] or $\lambda$[]
*output:*  p-function $P_2$ of $M_{S-S}(\pi)$,$M_{I-S}(\pi)$, $M_{S-O}(\pi)$ or $M_{I-O}(\pi)$
*procefure:*
begin var i,j,k,$n_1$,$n_2$,$n_3$:integer; FIND,EQ: boolean;
  if PT = 'S-S' → $n_1$,$n_2$,$n_3$ :=NS,NI,NS
   ‖ PT = 'I-S' → $n_1$,$n_2$,$n_3$ :=NI,NS,NS
   ‖ PT = 'S-O' → $n_1$,$n_2$,$n_3$ :=NS,NI,NO
   ‖ PT = 'I-O' → $n_1$,$n_2$,$n_3$ :=NI,NS,NO
  fi;
  i := 0;
  do $i \le n_3$ → i := i+1; $P_2$(i) := i od;
  i := $n_1$+1;
  do i>2 →
     i := i-1
     FIND,j := false,i
     do j>1 ∧ not FIND →
        j := j-1;
        k,EQ := 0,true;
     do k<$n_2$ ∧ EQ →
          k := k+1;
          if PT='S-S' →
              if $\delta$[i,k]≠$\delta$[j,k] → EQ:=id($\delta$[i,k])=id($\delta$[j,k])($P_1$)
               ‖ $\delta$[i,k]=$\delta$[j,k] → skip
              fi
           ‖ PT='I-S' →
              if $\delta$[k,i]≠$\delta$[k,j] → EQ:=id($\delta$[k,i])=id($\delta$[k,j])($P_1$)
               ‖ $\delta$[k,i]=$\delta$[k,j] → skip
              fi
           ‖ PT='S-O' →
              if $\lambda$[i,k]≠$\lambda$[j,k] → EQ:=id($\lambda$[i,k])=id($\lambda$[j,k])($P_1$)
               ‖ $\lambda$[i,k]=$\lambda$[j,k] → skip
              fi
           ‖ PT='I-O' →
              if $\lambda$[k,i]≠$\lambda$[k,j] → EQ:=id($\lambda$[k,i])=id($\lambda$[k,j])($P_1$)
               ‖ $\lambda$[k,i]=$\lambda$[k,j] → skip
              fi
          fi;
          if PT='S-S' → EQ := id($\delta$[i,k])=id($\delta$[j,k])($P_1$)
           ‖ PT='I-S' → EQ := id($\delta$[k,i])=id($\delta$[k,j])($P_1$)
        ▯ PT='S-O' → EQ := id(λ[i,k])=id(λ[j,k])(P_1)
        ▯ PT='I-O' → EQ := id(λ[k,i])=id(λ[k,j])(P_1)
       fi;
       if k=n_2 ∧ EQ → P_2(i) := j; FIND := TRUE
        ▯ k≠n_2 ∨ not EQ → skip
       fi
     od
   od
end
```

8.2.7 Relation Operations

Since many comparisons may be made for two partitions, two pairs, or two trinities, it is essential to establish some algorithms for them.

Because the comparisons of pairs or trinities are, in the final analysis, built up by those of partitions, we only consider here the algorithms for partitions. relations on the representation of p-functions.

Let π and τ are partitions on set S, and

$$\pi = \sum_{i=1}^{N} \{\pi_{i,j} | i>j \wedge [i]\pi=[j]\pi\}$$

$$\tau = \sum_{i=1}^{N} \{\tau_{i,j} | i>j \wedge [i]\tau=[j]\tau\}$$

Then, it is obvious to know that, for relation $\pi\leq\tau$,

$$\pi \leq \tau \quad \text{iff } \pi_{i,j}\in\pi \Longrightarrow \tau_{i,j}\in\tau$$

for all $\pi_{i,j}$, $i>j \wedge [i]\pi=[j]\pi$, in π.

With p-functions it is established by

$$P_1>P_2 \qquad \text{iff } P_1(i)\neq i \Longrightarrow id(i)=id(P_1(i))(P_2)$$

for all $i\in S$.

Thus, the algorithm for $P_1 \leq P_2$ is shown below.

```
Algorithm P1LTP2(var $P_1$,$P_2$: PTYPE; N: integer): boolean
*input* :  p-functions $P_1$ and $P_2$; partition type N;
*output*:  P1LTP2 := 1,true;
*procedure*:
  begin var i: integer;
    i,P1LTP2 := 1,true;
    do i≤N ∧ P1LTP2 →
       if i≠$P_1$(i) → P1LTP2 := id(i)=id($P_1$(i))($P_2$)
        ▯ i=$P_1$(i) → skip
       fi;
       i := i+1
    od
  end
```

Having the algorithm P1LTP2, other algorithms of relation operations are easily written down as follows:

```
Algorithm P1LEP2(var $P_1$,$P_2$: PTYPE; N: integer): boolean;
*input*:  p-function $P_1$ and $P_2$; partition type N;
*output*: P1LEP2=true if $P_1$<$P_2$
*procedure*:
  begin
    P1LEP2 := P1LTP2($P_1$,$P_2$,N) ∧ not(P1LTP2($P_2$,$P_1$,N))
  end
```

```
Algorithm P1EQP1(var $P_1$,$P_2$: PTYPE; N: integer): boolean;
*input*:  p-fuction $P_1$ and $P_2$; partition type N;
*output*: P1EQP2=true if $P_1$=$P_2$
*procedure*:
  begin
    P1EQP2 := P1LTP2($P_1$,$P_2$,N) ∧ P1LTP2($P_2$,$P_1$,N)
  end
```

8.2.8 m′(π) and M′(π)

Because of the existence of "don't care" conditions, the algorithms for computing m and M operations on an incompletely specified machine are ruled out. Here we consider the algorithms on weak pairs.

Let m′ denote the weak n-operation and M′ the weak M-operation.

Then for a partition P, there are four $m'(\pi)$ and four $M'(\pi)$ as follows:

$$\begin{array}{cccc} m'_{S-S}(\pi) & m'_{I-S}(\pi) & m'_{S-O}(\pi) & m'_{I-S}(\pi) \\ M'_{S-S}(\pi) & M'_{I-S}(\pi) & M'_{S-O}(\pi) & M'_{I-S}(\pi) \end{array}$$

According to the definition of sets on a machine mentioned before, the only difference between incompletely specified and completely specified machines is that there are some zero entries in the δ and λ tables. Therefore, we should have a special treatment to the zero entries, just like

$$m'_{S-S}(\pi_{i,j}) = \Sigma\ \{\tau_{i\delta_k, j\delta_k} | i\delta_k \neq j\delta_k, \text{ for all } k\in I\}$$

for weak m-operation, and

$$[i]_{M'_{S-S}(\pi)} = [j]_{M'_{S-S}(\pi)} \text{ iff } i\delta_k \neq j\delta_k \neq 0 \Rightarrow \delta[i\delta_k]\pi = [j\delta_k]\pi\ \forall k\in I$$

for weak M-operation.

With the representations of p-functions the treatments above are easily to do in Algorithms $m(\pi)$ and $M(\pi)$ by simply changing the restrictions such as $i\delta_k \neq j\delta_k \neq 0$, which are shown below.

For $M'(\pi)$, in Algorithm $M(\pi)$,

```
if δ[i,k]≠δ[j,k] becomes
   if δ[i,k]≠δ[j,k] ∧ δ[i,k]≠0 ∧ δ[j,k]≠0;
if δ[k,i]≠δ[k,j] becomes
   if δ[k,i]≠δ[k,j] ∧ δ[k,i]≠0 ∧ δ[k,j]≠0;
if λ[i,k]≠λ[j,k] becomes
   if λ[i,k]≠λ[j,k] ∧ λ[i,k]≠0 ∧ λ[j,k]≠0;
if λ[k,i]≠λ[k,j] becomes
   if λ[k,i]≠λ[k,j] ∧ λ[k,i]≠0 ∧ λ[k,j]≠0;
```

and for $m'(\pi)$, in Algorithm $m(\pi)$,

```
if δ[i,k]≠δ[j,k] becomes
   if δ[i,k]≠δ[P_1(i),k] ∧ δ[i,k]≠0 ∧ δ[P_1(i),k]≠0;
if δ[k,i]≠δ[k,j] becomes
   if δ[k,i]≠δ[k,P_1(i)] ∧ δ[k,i]≠0 ∧ δ[k,P_1(i)]≠0;
if λ[i,k]≠λ[j,k] becomes
   if λ[i,k]≠λ[P_1(i),k] ∧ λ[i,k]≠0 ∧ λ[P_1(i),k]≠0;
if λ[k,i]≠λ[k,j] becomes
   if λ[k,i]≠λ[k,P_1(i)] ∧ λ[k,i]≠0 ∧ λ[k,P_1(i)]≠0;
```

Thus, the complete descriptions of the Algorithms $m'(\pi)$ and $M'(\pi)$ are omitted here.

CHAPTER 9

EPILOGUE

We conclude this thesis with a short summary of the results obtained in preceding chapters and some opinion on further study of the full-decomposition theory.

Up to now, the discussions in this thesis are mainly located on the following aspects:

- Partition trinities which present a suitable representation for the information between input and output, and between present state and next state simultaneously (Chapters 3-7).

- Trinity algebra of a machine, such that we can directly apply many of the abstract tools that have been developed in algebra theory (Chapter 3).

- Parallel full-decompositions examed by PT's (Chapter 4)

- Serial full-decompositions detected by a PT and a FT (Chapter 5).

- H-decompositions based on so-called H-pairs (Chapter 6).

- Wreath decompositions set up by partition trinities (Chapter 6).

- Basic algorithms for doing decompositions and analyses with a computer (Chapter 8).

Moreover, we think the work appeared in this thesis is only an introduction to the trinity algebra and full-decomposition theory of machines. We still have some motivation on this subject with the following aspects:

. Specified decompositions. Let M_S be a machine and M be any machine to decompose. The decomposition to make, for some machine M′ amd some connection ω,

$$M \lhd M_S \ \omega \ M'$$

is called a specified decomposition. In other words, we specify a machine that should be a component machine of a decomposition. The decomposition is very significant in a situation where the specified machine M_S is corresponding to an avilable IC.

. The primary package DASM, Decompositions and Analyses of Sequential Machines, served as a tool for our study on machine decompositions and runs on ALTOS in the level of experiments. To develop a large package from it running on a large machine, say VAX, for a general application is necessary and possible. Of course, there will be some techniques to be considered for gaining speed and managements.

. Although having paid certain attention to mathematical description on trinity algevra we are still not satisfied with the description on it. Maybe it will be done by a mathematician who is interested in the trinity algebra.

. To expand the trinity algebra based on a set system is useful and possible.

. To develop the application of trinity algebra to complex decompositions in order to set up a more complete full--decomposition theory of machines.

APPENDIX

DASM

The programme package DASM (<u>D</u>ecompositions and <u>A</u>nalyses of <u>S</u>equential <u>M</u>achines) was primaryly designed and used as a valuable tool during the study of the subject of this thesis. Here, we gave a brief summary of DASM functions and the environment in which DASM was used.

LANGUAGE : PASCAL;
OPERATING SYSTEM: UCSD;
COMPUTER : ALTOS;
FUNCTIONS:

1) Basic operations:
   partition: $\pi^{m}_{s,t}$, $\pi_1 \cdot \pi_2$, $\pi_1 + \pi_2$;
   pair : $m(\pi)$, $M(\pi)$, $P_1 * P_2$, $P_1 + P_2$;
   trinity : $t_1 \odot t_2$, $t_1 \oplus t_2$;
2) SP partitions;
3) Partition pairs: S-S, S-O, I-S, I-O;
4) State decomposition of machines:
   parallel or serial;
5) Partition trinities;
6) Full-decompositions of machines;
7) Assignment of states of machines;
8) Simulation of machines;
9) Analyses and decomposition of ISSM's.

RUNNING:

Once the diskette DASM was put in drive A of ALTOS, the system automaticly went to DASM state. The functions mentioned above could be recalled under the guidance of the menu display along the top line of the screen.

The main command line on such a guide line was like like

```
*DASM(1984): D(ecomp. F(ull-decomp. I(SSM T(rinity ?[HYB 84.01].
```

Typing a guestion mark '?' would cause a display of the rest function commands:

```
*DASM(1984): P(air S(P-partition A(ssignment M(sinulation H(elp
             Q(uit [HYB 84.01]
```

In this situation, typing any capital letter in the command line can get a certain function while DASM goes to a sublevel. For example, typing 'D' change the guide line into

```
>Decomp: P(arallel, S(erial, Q(uit [HYB 84.01].
```

Furthermore, pressing 'P' causes DASM to make a parallel decomposition of a machine. In this way, we can enter or leave any level. The parameters needed for a particular calculation are input entered as an interactive mode. Also, the results can be put into a device, such as a printer, a screen, or a diskette according to the instruction from a user. An 'H' command in main level represents some explanation for using this package.

A detailed description of DASM will be presented in a seperate documentation accompanying the final version of DASM later.

REFERENCES

[1] Abdullaev, D.A. and D. Yunusov
DECOMPOSITION OF SYMMETRICAL BOOLEAN FUNCTIONS.
Autom. Control & Comput. Sci., Vol. 9, No. 2(1975), p. 11-12.
Transl. of: Avtom. & Vychisl. Tekh., Vol. 9, No. 2(1975), p. 12-13.

[2] Cioffi, G. and E. Costantini, S. de Jiulio
A NEW APPROACH TO THE DECOMPOSITION OF SEQUENTIAL SYSTEMS.
Digital Processes, Vol. 3(1977), p. 35-48.

[3] Cioffi, G. and S. De Julio, M. Lucertini
OPTIMAL DECOMPOSITION OF SEQUENTIAL MACHINES VIA INTEGER NON-LINEAR PROGRAMMING: A computational algorithm.
Digital Processes, Vol. 5(1979), p. 27-41.

[4] Dijkstra, E.W.
A DISCIPLINE OF PROGRAMMING.
Englewood Cliffs, N.J.: Prentice-Hall, 1976.
Prentice-Hall series in automatic computation

[5] Eilenberg, S.
AUTOMATA, LANGUAGES, AND MACHINES. Volume A.
New York: Academic Press, 1974.
Pure and applied mathematics: A series of monographs and textbooks

[6] Eilenberg, S.
AUTOMATA, LANGUAGES, AND MACHINES. Volume B.
New York: Academic Press, 1976.
Pure and applied mathematics: A series of monographs and textbooks

[7] Enin, S.V. and P.N. Bibilo
JOINT DECOMPOSITION OF A SYSTEM OF VECTOR BOOLEAN FUNCTIONS.
Autom. Control & Comput. Sci., Vol. 13, No. 1(1979), p. 14-20.
Transl. of: Avtom. & Vychisl. Tekh., Vol. 13, No. 1(1979), p. 16-22.

[8] Friedman, A.D. and P.R. Menon
THEORY & DESIGN OF SWITCHING CIRCUITS.
Woodland Hills, Cal.: Computer Science Press, 1975.
Digital system design series

[9] Ginzburg, A.
ALGEBRAIC THEORY OF AUTOMATA.
New York: Academic Press, 1968.
ACM monograph series

[10] Haring, D.R.
SEQUENTIAL-CIRCUIT SYNTHESIS: State assignment aspects.
Cambridge, Mass.: MIT Press, 1966.
Research monograph, No. 31.

[11] Hartmanis, J.
ON THE STATE ASSIGNMENT PROBLEM FOR SEQUENTIAL MACHINES I.
IRE Trans. Electron. Comput., Vol. EC-10(1961), p. 157-165.

[12] Hartmanis, J.
LOOP-FREE STRUCTURE OF SEQUENTIAL MACHINES.
Inf. & Control, Vol. 5(1962), p. 25-43.

[13] Hartmanis, J.
FURTHER RESULTS ON THE STRUCTURE OF SEQUENTIAL MACHINES.
J. Assoc. Comput. Mach., Vol. 10(1963), p. 78-88.

[14] Hartmanis, J. and R.E. Stearns
PAIR ALGEBRA AND ITS APPLICATION TO AUTOMATA THEORY.
Inf. & Control, Vol. 7(1964), p. 485-507.

[15] Hartmanis, J. and R.E. Stearns
ALGEBRAIC STRUCTURE THEORY OF SEQUENTIAL MACHINES.
Englewood Cliffs, N.J.: Prentice-Hall, 1966.
Prentice-Hall international series on applied mathematics

[16] Holcombe, W.M.L.
ALGEBRAIC AUTOMATA THEORY. Cambridge University Press, 1982.
Cambridge studies in advanced mathematics, Vol. 1.

[17] INTEGRATED CIRCUITS. Part 10: Signetics Integrated Fuse Logic (IFL).
Eindhoven: Philips Electronic Components and Materials Division (ELCOMA), May 1983.
Philips data handbook, IC 10.

[18] Kammozev, N.F. and A.N. Sychev
SPECTRAL METHOD OF DECOMPOSITION OF BOOLEAN FUNCTIONS.
Autom. Control & Comput. Sci., Vol. 13, No. 2(1979), p. 46-50.
Transl. of: Avtom. & Vychisl. Tekh., Vol. 13, No. 2(1979), p. 54-58.

[19] Krohn, K. and J. Rhodes
ALGEBRAIC THEORY OF MACHINES I: Prime decomposition theorem for finite semigroups and machines.
Trans. Amer. Math. Soc., Vol. 116(1965), p. 450-464.

[20] Lew, A.
COMPUTER SCIENCE: A mathematical introduction.
Englewood Cliffs, N.J.: Prentice-Hall, 1985.
Prentice-Hall international series in computer science

[21] Mealy, G.H.
A METHOD FOR SYNTHESIZING SEQUENTIAL CIRCUITS.
Bell Syst. Tech. J., Vol. 34(1955), p. 1045-1079.

[22] Moore, E.F.
GEDANKEN-EXPERIMENTS ON SEQUENTIAL MACHINES.
In: Automata Studies. Ed. by C.E. Shannon and J. McCarthy.
Princeton University Press, 1956.
Annals of mathematical studies, Vol. 34. P. 129-153.

[23] Pottosin, Yu.V. and E.A. Shestakov
APPROXIMATE ALGORITHMS FOR PARALLEL DECOMPOSITION OF AUTOMATA.
Autom. Control & Comput. Sci., Vol. 15, No. 2(1981), p. 24-31.
Transl. of: Avtom. & Vychisl. Tekh., Vol. 15, No. 2(1981), p. 31-38.

[24] Pottosin, Yu.V. and E.A. Shestakov
DECOMPOSITION OF AN AUTOMATION INTO A TWO-COMPONENT NETWORK WITH CONSTRAINTS ON INTERNAL CONNECTIONS.
Autom. Control & Comput. Sci., Vol. 16, No. 6(1982), p. 24-31.
Transl. of: Avtom. & Vychisl. Tekh., Vol. 16, No. 6(1982), p. 25-32.

[25] Shen, V.Y. and A.C. McKellar
AN ALGORITHM FOR THE DISJUNCTIVE DECOMPOSITION OF SWITCHING FUNCTIONS.
IEEE Trans. Comput., Vol. C-19(1970), p. 239-248.

[26] Shvartsman, M.I.
OUTPUT DECOMPOSITION FOR COMBINATIONAL PLA-STRUCTURES.
Autom. Control & Comput. Sci., Vol. 15, No. 6(1981), p. 9-14.
Transl. of: Avtom. & Vychisl. Tekh., Vol. 15, No. 6(1981), p. 12-17.

[27] Sorokin, B.L.
DECOMPOSITION METHOD OF SYNTHESIZING CIRCUITS BASED ON PROGRAMMABLE LOGIC ARRAYS.
Autom. Control & Comput. Sci., Vol. 16, No. 4(1982), p. 47-52.
Transl. of: Avtom. & Vychisl. Tekh., Vol. 16, No. 4(1982), p. 50-55.

[28] Thayse, A.
A FAST ALGORITHM FOR THE PROPER DECOMPOSITION OF BOOLEAN FUNCTIONS.
Philips Res. Rep., Vol. 27(1972), p. 140-150.

[29] Yoeli, M.
THE CASCADE DECOMPOSITION OF SEQUENTIAL MACHINES.
IRE TRans. Electron. Comput., Vol. EC-10(1961), p. 587-592.

Samenvatting.

Het proefschrift behandelt het decomponeren van sequentiële machines in kleinere machines. Traditioneel zijn deze decomposities gericht op het minimaliseren van het aantal toestanden. In de hier behandelde theorie minimaliseren we ook het aantal inputs en outputs (verbindingsdraden) in de decompositie. We spreken dan van een totale decompositie ("full-decomposition").

Totale decomposities ontlenen hun belang aan de komst van complexe geïntegreerde schakelingen (VLSI), waarin het aantal verbindingsdraden een belangrijke beperkende factor vormt.

De theorie van totale decomposities is gebaseerd op de wiskundige begrippen partitie-triniteit en triniteits-algebra, welke in dit proefschrift worden geïntroduceerd. Evenals in de traditionele decompositie-theorie onderscheiden we parallelle en seriële decomposities. Voor de laatstgenoemde decomposities wordt het begrip geforceerde triniteit ("forced-trinity") ingevoerd. De theorie wordt verder uitgebreid met H-decomposities - een variant van de parallelle decompositie - en kransdecomposities. We laten zien dat het merendeel van de theorie ook kan worden toegepast op onvolledig gespecificeerde machines.

Tenslotte presenteren we een aantal algoritmen, die gebruikt kunnen worden bij het analyseren van machines en het berekenen van decomposities van machines.

CURRICULUM VITAE

De schrijver van dit proefschrift werd op 12 april 1952 te Shaanxi in de Volksrepubliek China geboren.

Hij beëindigde de Wugong Middelbare School met een eindexamen in 1968. In 1972 begon hij zijn universitaire studie in de afdeling elektronica van de Xian Jiaotong Universiteit. Deze studie werd in 1975 afgesloten. In de daarop volgende jaren werkte hij op het Instituut der Shaanxi Dynamic. Hij hervatte zijn studie op de Xian Jiaotong Universiteit in 1978, waar hij in 1981 de M.Sc. graad onder leiding van Prof. Zheng Shouqi verkreeg. Tot 1982 werkte hij als docent op dezelfde universiteit. Sinds 1983 is hij research fellow in de afdeling der Elektrotechniek van de Technische Hogeschool te Eindhoven in de Vakgroep Digitale Systemen (voorzitter Prof.ir. A. Heetman).

STELLINGEN

[1] With the development of integrated circuit technology, the decomposition theory of machines must include decomposition related to pins of IC's, in addition to internal components (Chapters 1,2).

[2] For any sequential machine, there is a trinity lattice and a trinity algebra for it(Chapter 3).

[3] If there are two orthogonal partition trinities for a machine, then, that machine can be decomposed into the interconnection of two smaller machines which can work independently or in parallel with separate inputs and outputs(Chapter 4).

[4] A partition trinity and a forced-trinity in which the trinity product is zero trinity show that the machine is of a serial full-decomposition. That is, there are two smaller machines with distinct inputs and outputs and one of them takes a message from the other (Chapter 5).

[5] The minterm-vector method provides an approach to prepare a numerical algorithm for fault diagnosis and a new way of calculating Boolean differences on a computer.

Hou Yibin & Zheng Shouqi: A Minterm-vector Method for Diagnosting Faults in Combinational Networks, Journal of Xian Jiaotong Univ. Selected Paper of Scientific Research (in English), pp. 157-161, 1981

[6] During the next ten years, computer security will be one of the most important subjects.

Harold Lorin: Emerging Security Requirments, Computer Communications, pp. 293-298, Vol.8, No.6, December, 1985.

[7] "Structured programming" is the inevitable outcome of "structured design thought" that exists in all engineering design areas.

1. O,J. Dahl, E.W. Dijkstra and C.A.R. Hoare: Structured Programming, Academic Press, London, 1972.
2. V.R. Basili & T. Baker: Structured Programming, IEEE Computer Society, IEEE catalog No. 75ch1049-6, 1975.

[8] Unlike society, science has no national boundaries; it is a bridge for friendship while friendship is a wing of science.

1. Claude Bernard: "Art is I, science is we."
2. THE: Statement of Intent between the Eindhoven University and the Xian Jiaotong University, TH Berichten, Nr.14, p.5, 16 november, 1984.

[9] The number of operational symbols in discrete mathematics is insufficient for describing complex systems. Thus, it never ends to create new symbols.

1. J.P. Tremblay & R. Manohar: Discrete Mathematical Structures with Applications to Computer Science, McGraw-Hill, 1975.
2. A. Lew: Computer Science: A Mathematical Introduction, Englewood Cliffs, N.J.: Prentice-Hall, 1985.

[10] Language problems consume much time, but, in the Tomorrow of Mankind, all the people will speak the same language.

[11] A personal computer is not only an interesting asset, but it can also be tiring to use.