ABSTRACT

STRUCTURAL REALIZATIONS OF
PROGRAM SCHEMATA

By

John G. Miles

Finite state machine theory can be employed to synthesize and detect common program structures ("controls" or "schemata") when these can be identified as sequential machines. The computation a program performs can be described as an interpretation or mapping on these structures. This dissertation proposes a program model and develops a number of structural realizations for common controls. The necessary constructions of appropriate interpretations on those controls to preserve program equivalence are described. A particular set of machine transformations is introduced which can be used to design optimal common control realizations.

STRUCTURAL REALIZATIONS OF PROGRAM SCHEMATA

By

John G. Miles

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

1971

To

Hall John Miles

and

Arline Miles

ACKNOWLEDGEMENTS

I wish to thank Professors Robert Barr, Richard Dubes, Julian Kateley and William Kilmer for serving on my doctorol committee and for many inspiring classroom sessions. I am especially grateful to my committee chairman, Dr. Carl Page, for his patient and sympathetic help in developing the ideas of this work.

I am deeply indebted to Dr. Richard Reid for the many opportunities made available to me while at MSU. Also, I would like to thank the Division of Engineering Research, under the able hand of John Hoffman, which provided both financial assistance and intellectual nurture during my graduate studies.

The price extracted for the completion of this venture would not have been met without the understanding, loyality and patience of my wife, Vicki.

Finally, my sincere thanks to Dr. Martin Keeney, whose friendship and encouragement in large measure sustained my efforts throughout my education.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd.)

LIST OF FIGURES

LIST OF FIGURES (Cont'd.)

LIST OF SYMBOLS AND TERMS

Ξ	Algorithm represented by a finite state machine
Σ^*	Set of all finite input sequences on an alphabet, Σ, 7.
$h(\cdot)$	Completely specified I/O function for a FSM,7.
$\leftrightarrow$	Denotes "corresponds to"
$\Longrightarrow$	Logical implication
ϵ	Set membership
$h_x(\cdot)$	I/O function formed by left translation, 7.
$\sigma_i \epsilon \Sigma$	i^{th} input symbol of a finite input alphabet
$x,y \epsilon \Sigma^*$	Finite length tapes
C	Computer representation, 14.
M	Location index set of computer memory, 14. Also a finite state machine depending on context.
B	Base set, 15.
D	State set of C, 15.
F	All functions on D, 15.
F^*	Set of finitely composed functions from F.
P	Set of all finite decision predicates on D, 16.
p_i^K	i^{th} K-ary decision predicate on D, 7, 16.
τ	Program schema or control, 16.
$\langle\ \rangle$	A tuple denoting an algebraic structure
$\delta(\cdot)$	State transition function, 16.
$\lambda(\cdot)$	Output function, 16.
I	Interpretation on a control, 16.

$\eta(\cdot)$	Map from control states to a computation sequence and decision predicate, 17.
$\gamma(\cdot)$	Map from states to decision predicates, 17.
$\zeta(\cdot)$	Map from output set to computation sequence, 17.
d, d_0	An initial state of D, 17.
$I[\tau]$	Interpreted schema or program, 17.
Δ, Y	Used variously to denote a finite set of output symbols for a FSM, 16, 18.
$\psi(\cdot)$	Assignment map for machine realization, 19.
$\alpha(\cdot)$	State to state subsets map, 19.
$\rho(\cdot)$	Input to input alphabet map, 19.
$\xi(\cdot)$	Output to output map, 19.
A-type	Denotes a particular machine realization, 20.
$\phi(\cdot)$	Machine or control homomorphism, 20.
$h_1(\cdot)$	State to State map of the homomorphism, 20.
$h_2(\cdot)$	Input to input alphabet map of the homomorphism, 20.
$h_3(\cdot)$	Output to output map of the homomorphism, 20.
$M\Big\vert_{\psi}^{\psi}, \tau\Big\vert$	Machine or control assignment restriction, 21.
H-type	Denotes a particular type of machine realization, 22.
ψ^*	Machine capability, 22.
ρ^*	Input alphabet to Input string map, 22.

R	A binary relation with varying attributes depending on context.
SP	Denotes an SP partition, 23.
L_M, L_τ	SP lattices of a machine or control, 24.
π^M, π^τ	An SP partition of a machine or control, 24.
S_M, S_τ	Semigroup accumulator of a machine or control, 28. Also the state set of a machine or control depending on context.
$M_1 \times M_2$, $\tau_1 \times \tau_2$	The direct product of two machines or controls, 28.
$\left(M_1 \times M_2\right)^c$, $\left(\tau_1 \times \tau_2\right)^c$	Connected direct product machine or control, 29.
Q_τ	The state set of a control.
$(Q_1 \times Q_2)^r$	Reachable set of states in a direct product machine or control, 29.
$\cong$	Isomorphism.
$rp_M(\cdot)$ $rp_\tau(\cdot)$	Response function of a machine or control, 31.
$rp_{s_0}(\cdot)$	Alternate designation of the response function with emphasis on the initial state.
Cyclic τ	A control whose states are all reachable from a single start state, 31.
¶	Program $\left(= I[\tau]\right)$ on an interpretated control, 33.

$V_{d_0}(\P)$	The value of a program, 33.
$S_{d_0}(\P)$	The valuation sequence on Σ, 34.
$\overline{\P}$	Initial configuration-free program, 34, 35.
$\tilde{S}_D(\P)$	Set of all valuation sequences for $\overline{\P}$, 35.
$\tilde{V}_D(\P)$	Set of all program values for $\overline{\P}$, 35.
$\equiv_V$	Program value equivalence, 48.
$\equiv_I$	Interpretation equivalence relation on controls, 49.
$\equiv_I(\tau)$	Control equivalence class under $\equiv_I$, 49.
R_I	Interpretation equivalence relation on control equivalence classes, 49.
Common Control, Γ	Common control for a class of controls, 50.
A_I	R_I relation between two controls, one of which is an A-type realization of the other, 52.
I^A	A-type interpretation construction, 54.
H_I	R_I relation between two controls, one of which is an H-type realization of the other, 63.
$\hat{\tau}$	Output-free control, or semi-control, 73.
$\lfloor^\pi$	Submachine of τ induced by SP partition, π, on Q_τ, 74.
G_τ	Set of minimum grids, 108.
g_i	A member of G_τ, 108.
$L_m[\tau]$	Set of m-state labeled extension of τ, 110.

q_i^e, s_j^e	Exit states, 117.
$\bar{p}_i^j$	Modified decision predicate due to a reordering transformation, 133.
i_τ	i^{th} reordering of the state transitions of the control, τ, 134.
$(n)_{10}$	Base ten representation of the number n. $(n)_2 \longleftrightarrow$ base two, 134.
i_{δ_τ}	State transition function of the reordered control, i_τ, 134.
T	Class of all FSM semi-controls with n states over a binary alphabet, 146.
T^k	A subset of T, each member of the set having at least one n-k+1 block SP partition, 146.
$r(q)$	Set of reachable states from state q by tapes $x \in \Sigma^*$, 149.
Θ	SP state cover, 157.
$R[\hat{\tau}_1 \hat{\tau}_2]$	Set of R-minimal direct product controls, 158.
$\Theta_{\hat{\tau}_1}$	The $\hat{\tau}_2$ induced cover on $\hat{\tau}_1$ for the direct product, $\hat{\tau}_1 \times \hat{\tau}_2$, 158.
$\|\Theta_\tau\|$	The sum of all elements in blocks of the cover, Θ_τ, 161.
A_i	State set generated by the reordering technique in Theorem 4.12, 177.
$\triangleright\triangleleft$	End of proof symbol.

CHAPTER I

PROGRAM SPECIFICATIONS AND MODEL

1.1 Introduction

In recent years some notions of a general theory of computability have begun to take shape in the hands of an increasing number of researchers. The breadth of the field is as extensive as the number of people working on topics ranging from formal language theory, to Turing machines and recursive function theory. The overall objectives are identical: What is a computation and is there a convenient model to describe how man and machine work together to accomplish it? The formalisms introduced are useful for studying powerful global aspects in these areas but are of little value to someone trying to program an algorithm, or finite process, in some programming language.

This dissertation will describe and investigate certain aspects in the process of programming an algorithm from its inception to a final implementation in some problem oriented language. The goal throughout will be to model and organize programs independent of any particular computer and programming language thus establishing a basis for "engineering" large systems programs. A mathematical

model will be used to investigate certain properties of programs such as one program's ability to simulate another, program equivalence, and program value (i.e. the set of computations performed). Primarily, the structure of programs defined by the model will be studied for possible inferences on these properties.

The ensuing discussion is particularly relevant in the treatment of algorithms which perform some kind of numerical computation. Simulation studies and engineering design and analysis methods rely heavily on such commonplace algorithms as root-finding and solution of simultaneous linear and nonlinear differential equations. The fact that the number of algorithms in this category is so vast and that most large programs used by engineers in analysis and design employ dozens of such schemes requires some means of systematically developing these programs, keeping them updated and monitoring their performance. It will be worthwhile if the identification and organization of primary aspects in the programming process contained herein leads to a more methodical approach in the design of such large programs.

1.2 Dissertation Objectives

The basic premise of this dissertation is that finite state machine (or automata) theory has considerable potential in dealing with the practical problems of program design. Structure and effect of programs, and information structures in general, can be thought of in terms of the structure and behavior of a finite state machine (FSM).

From the standpoint of finite state machine theory, generating structurally related controls or program schemata usually guarantees behavioral equivalence. A direct parallel could be drawn to program behavioral equivalence were it not for severe implementation problems arising as a result of different structural relationships. These difficulties are quite apart from FSM theoretic considerations. But to take advantage of the wealth of FSM theory, an intuitively appropriate model of a program and notions of program and control equivalence must be defined so as to create an environment where practical programming problems can be dealt with in terms of FSM concepts. In this dissertation such a program model is defined and used to construct structurally equivalent--hence value equivalent--programs. In addition, procedures will be introduced and employed in the optimization of such programs. Of major

importance throughout the development of these matters is the exposition of implementation difficulties and the conditions under which they occur. Whenever possible, proofs have been designed to be constructive in revealing these problems.

Section 1.3 introduces the notion of an algorithm and its realization as a finite state machine from an initial specification. This sets the stage for a presentation of a complete program model in Section 1.4. The material in Chapters II, III, and IV relies heavily on FSM concepts reviewed in Section 1.5, especially the classic notions of machine realizability.

Contained in the first section of Chapter II are definitions of program value and valuation sequences along with necessary and sufficient conditions relating valuation sequences and program values in a one to one manner. A program value equivalence relation is defined in the next section and leads to a partial ordering on program schemata and the notion of a common control for a class of program schemata. In Section 2.3 structural realization theorems are presented which show how appropriate interpretations are constructed to preserve program value.

Chapter III is concerned with the synthesis of three common controls of a particular structural type. Two realizations, S.P. partition (Section 3.1) and semigroup (Section 3.2) realizations, involve concepts which are not new. The composite realization (Section 3.4) is a deceptively simple model of what is an intuitively appealing, and quite direct, approach to constructing common controls. Section 3.3 contains a complete example of a practical use of semigroup common controls.

A fourth common control realization, utilizing direct products, is developed in the first section of Chapter IV. Section 4.2 introduces a new class of machine transformations. It is shown that while structural equivalence is not preserved under these transformations, program value is when the appropriate interpretations are constructed. Sections 4.3 and 4.4 investigate the use of these transformations in synthesizing optimal common controls of the type studied in previous sections.

1.3 Algorithm Specification

To model any physical device which is to interact with a given environment in some predetermined way, it is essential to specify the function and nature of the

interaction so that a designer can use the information directly in designing and modeling a suitable prototype. The same holds true for an algorithm which is to be programmed in some language and run on some class of computers. In this section it is assumed that a language and computer are available which have sufficient capability to describe and perform all necessary computations. All accessible programs, procedures, processes and data stored in working and secondary memory constitute the environment in which the algorithm--implemented as a program in the language--will function.

Given certain conditions on the environment, an algorithm, Ξ, is to direct computations on data in the environment until those conditions are met. To make this more precise, define a set of calculations which need be performed by Ξ as

$$F = \left\{ f_i \;\middle|\; R(f_i) \longleftarrow f_i \left[D(f_i) \right] \right\}$$

$D(f_i)$, $R(f_i)$: domain, range of f_i which are information structures in the computer

$R(f_i) \leftarrow f_i \left[D(f_i)\right]$: the results of applying f_i to some elements from $D(f_i)$ are placed in $R(f_i)$

and define an m-valued decision predicate

$$p^m : D(p^m) \rightarrow \left\{ \sigma_1, \ldots \sigma_m \right\} \triangleq \Sigma$$

F and p^m will be defined in greater detail in later sections. Then the algorithm might be specified by the 3-tuple $\Xi = \langle F, p^m, h \rangle$ where F, p^m are as above and h is the input-output function

$$h: \Sigma^* \longrightarrow F \; ; \qquad \Sigma^* = \text{set of all finite strings or words (free semigroup) on the alphabet } \Sigma$$

and is given ideally by an infinite set of pairs of the form

$$h : \begin{array}{l} (\sigma_1, f_{i_1}) \\ \quad \vdots \\ (\sigma_m, f_{im}) \\ (\sigma_1\sigma_1, f_{im+1}) \\ \quad \vdots \\ (\sigma_1\sigma_m, f_{i2m+1}) \\ \quad \vdots \end{array}$$

As such h is called <u>completely specified</u>.

Define a set of functions by <u>left translation</u> as follows:

for all $x \epsilon \Sigma^*$

$$h_x(y) \triangleq h(xy), \quad \text{for all } y \epsilon \Sigma^*$$

If the set $\left\{h_x\right\}_{x \epsilon \Sigma^*}$ is closed under left translation and is finite, then a finite state sequential machine, M, can be derived to produce the desired function, h, in the following manner:

$$M = \left\langle S, \Sigma, F, \delta, \lambda, s_0 \right\rangle$$

where

F: as above, called the output alphabet

Σ: as above, called the input alphabet

S: the state set of M

δ: $S \times \Sigma \rightarrow S$: transition function for M

λ: $S \times \Sigma \rightarrow F$: output function for M

and make the association

$$S \longleftrightarrow \left\{h_x\right\}_{x \epsilon \Sigma^*}$$

such that to each $s_i \epsilon S$ there corresponds one and only one

$$h_y \epsilon \left\{h_x\right\}_{x \epsilon \Sigma^*}$$

The state transition function is defined as follows:

for all $\sigma \epsilon \Sigma$:

$\delta(s_0, \sigma) = s_i$ iff $s_i \longleftrightarrow h_\sigma$, and $s_0 \longleftrightarrow h$

$\delta(s_i, \sigma) = s_j$ iff $s_i \longleftrightarrow h_\gamma$, $s_j \longleftrightarrow h_\xi$: $\gamma, \xi \epsilon \Sigma$ and $h_{\sigma\gamma} = h_\xi$

and the output function is defined

$$\lambda(s_0,\sigma) = f_i \text{ iff } (\sigma,f_i) \in h,\ s_0 \longleftrightarrow h$$

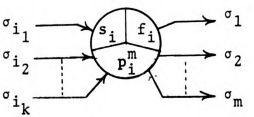
$$\lambda(s_i,\sigma) = f_j \text{ iff } (\sigma,f_j) \in h_\gamma,\ s_i \longleftrightarrow h_\gamma$$

The equivalent Moore machine is easily generated from this machine and it is this model which will be used in the sequel. Although the direct relationship of the output function and the I/O function, h, is lost, the Moore model associates a calculation with only a state which will be more convenient for the purposes at hand.

The (Moore machine) representation of the algorithm requires an m-way decision test at each state to direct it to further action. Thus a call to some procedure in the environment to initiate this test can be associated with each state in the program. When treating a class of programs, each with its own m-way decision test, it is sometimes advantageous to utilize a standardized decision logic. To this end partition each m-way test into (m-1) 2-way tests and associate with each state in a program a particular binary decision. The environment can then be thought of as simply returning to each program in the class true-false (1-0) response strings. The effect is the same as encoding the input strings in specifying h for each program. The

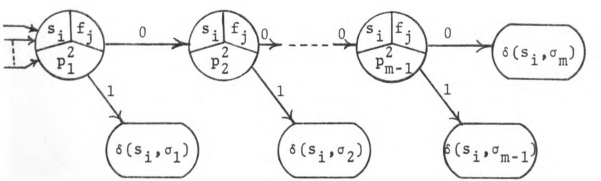
price paid is obviously an increase in the number of states in the program but in the conceptual development this is not of primary concern.

It is clear that the states may be split in a number of different ways. One method is illustrated in the following manner. Suppose a typical state in a Moore model representation of an algorithm is of the form



where: $\sigma_{i_k} \varepsilon \Sigma, |\Sigma| = m, k \leq m$

then its (m-1) 2-way decision equivalent is



With this breakdown and the state-only functional dependence of the output function in the Moore model, each state of the

program is characterized by a binary decision, p_i^2, and a calculation, f_i. Hence the correspondence

$$s_i \longleftrightarrow (f_i, p_i^2)$$

It is this association which will be exploited in the general model in section 1.4. In summary then, given initially a decision-calculation functional specification of an algorithm, $\Xi = \langle F, P, h \rangle$, a finite state machine--or control--can be developed where each state is associated with a calculation and a call for some decision, with the control responding to decision outcomes as inputs.

It should be noted that major difficulties in the transition from I/O function to machine synthesis arise from two considerations:

(1) h is usually incomplete in that some combinations (σ, f_i) either never occur or are "don't cares."

(2) h is defined over an infinite set, Σ^*, hence to completely specify h, a grammar or a Backus Naur Form representation together with some recursive definitions are required to identify, by common output, classes of strings in Σ^*.

Programmers bypass (2) entirely and circumvent (1) by, in effect, designing a flowchart--essentially a finite state machine--and filling don't cares by either arbitrary assignment, in which case it is assumed these combinations never occur, or by effecting a terminal procedure indicating an unprocessable abnormality.

This particular way of developing a finite state machine from an I/O function is in the spirit of that suggested in Assmus-Florentin [1968]. Other authors, notably Harrison-Grey [1966], have algorithms synthesizing machines from sequential functions. The concept is similar but the nature of the input-output function varies. Here only the last output is exhibited by the function h, when the machine processes an input string from Σ^*.

1.4 Program Model

Considerable attention has been given recently to developing models for asynchronous computation and parallel processing--see for example, Luconi [1967] and Karp-Miller [1966]. Most approaches have separately identified a control and an interpretation which together describe a specific computational procedure or program. The first attempts to apply this concept in programming appears to have been made by Ianov [1957] and Ianov [1958] and amplified in Rutledge [1964]. Ianov dis-

tinguished program control from interpretation and examined a particularly strong notion of equivalence of schemata--i.e. under all interpretations. Karp and Miller [1966] and [1967] have studied computational graphs--directed graphs with nodes representing computations and branches as transfer of control. Karp and Miller have recast their original studies on graphs along different lines reminiscent of Ianov's work. In the models of Luconi and Karp-Miller, queues are associated with branches leading into and out of nodes (operators) in the control and these stacks (called links by Luconi and the u-function by Karp and Miller) are used variously to store operator initializations, operands, results of operations, and certain operator or operand attributes. Because these authors have assumed a set of operations can be initiated arbitrarily and simultaneously, these stacks are used as working areas in each performance of an operation. Local conditions are then established on information in these stacks to investigate questions of performance (determinacy, equivalence, etc.) within and among programs.

Although the authors deal with considerations not directly applicable here, it is significant that the separation of the control and interpretation functions is suitable for studying computation within a multiprocessing environment. The viewpoint here is that isolating the control flow in a

computational procedure so that the specific operations to be performed are mapped onto the control to effect the procedure is analogous to, say, an electromechanical device with a fixed interconnection topology but whose network parameters or a portion of the component specification are arbitrary and are required for the characterization of the device.

Definitions of components of the model used in this thesis to exploit this separation are presented below. The model has three parts of which one is a computer model to underline the contention that in any program model it should be possible to relate all operations carried out in a program to a sequence of basic functions or operations inherent in the capability of some assumed class of machines.

Definition 1.1

This capability or computer representation is described by the 5-tuple

$$C = \langle M, B, D, F, P \rangle$$

where

M: the location index set of the total computer memory including core, secondary storage, special registers, etc.

$$M = \left\{ i \right\}_{i = 1,m}$$

$$B = \bigcup_{i=1,m} B_i$$

B_i: set of values cell $i \in M$ may assume. So if $i' \in M$ is a 12-bit register then $B_{i'} = \{0,1,\ldots 2^{12}-1\}$ and is appropriate for all 12-bit registers in M. B is called the base set of the computer*.

D: state set of C.

$D = \{g \mid g: M \rightarrow B\}$ where a state is some assignment of locations in M to values in the base set.

viz. $g(i) \in B_i \subseteq B$ for all $i \in M$

equivalently,

$$D = \underset{i \in M}{\times} B_i$$

F: set of all functions defined on D into D.

$F = \{f \mid f: D \longrightarrow D\}$

and

$|F| = D^D$

---

*This notation and definition is due to W.D. Maurer. See Maurer [1966] and [1968].

P: set of all finite decision predicates on D.

$$P = \bigcup_K \bigcup_i \left\{ p_i^K \mid p_i^K : D \longrightarrow \left\{ \sigma_1, \sigma_2, .., \sigma_K \right\} \right\}; 1 \leq K < \infty$$

Definition 1.2

A program schema, control or, simply, schema is defined as a finite state machine:

$$\tau = \langle S, \Sigma, \delta, \lambda, Y, s_0 \rangle$$

$$S = \left\{ s_i \right\}; \quad i=1,n$$

$$\Sigma = \left\{ \sigma_1, \sigma_2, \ldots \sigma_L \right\}$$

$\delta : S \times \Sigma \rightarrow S$

$\lambda : S \rightarrow Y$, a finite set of outputs

$s_0 \varepsilon S$

Definition 1.3

An interpretation on τ is a mapping

$$I : \tau \longrightarrow C$$

The mapping I is defined by the 4-tuple

$$I = \langle \eta, \gamma, \zeta, d_0 \rangle$$

where

η: $S \longrightarrow F^* \times p^L$

with

$\eta(s_0) = (f_0, p_0^L) \varepsilon F^* \times p^L$, and f_0, p_0^L defined for d_0

η: associates with each state a finite length sequence of computations, $\left\{f_i\right\} \varepsilon F^*$, and an L-ary decision, $p_i^L \varepsilon P^L$.

γ: $S \rightarrow P^L$

γ: represents the environmental response (the predicate value) to the L-ary decision predicate associated with states in S.

ζ: $Y \rightarrow F^*$

ζ: a projection of η onto F^* via

$$\zeta\left(\lambda(s_i)\right) = F^* \text{ projection of } (s_i)$$

$d_0 \varepsilon D$ an initial state.

Definition 1.4

The specific map, $I[\tau]$, will be called an interpreted schema, or a program.

Note that the γ mapping takes every state, $s_i \varepsilon S$, into an

L-ary decision predicate, thereby insuring τ is a completely defined machine. The key feature of the interpretation is that all elements in the control are related to appropriate entities in the machine class C, and as such define sequences of operations and decisions in terms of basic machine functions.

The three components (C,τ,I) serve as a basis for establishing a model which features (1) reference to a machine with sufficient capability and (2) separation of program control and interpretation. How these aspects are to be utilized must be postponed until some useful machine-theoretic concepts are reviewed.

1.5 Mathematical preliminaries

To adequately define realizability and equivalence between programs, certain concepts, definitions and theorems from automata and sequential machine theory are required*. Given two (Moore) sequential machines:

$$M_1 = \langle Q_1, \Sigma_1, \Delta_1, \delta_1, \lambda_1, q_0 \rangle \quad ; |Q_1| = n_1$$

$$M_2 = \langle Q_2, \Sigma_2, \Delta_2, \delta_2, \lambda_2, p_0 \rangle \quad ; |Q_2| = n_2$$

---

* Primary references for automata and sequential machine theoretic notions are the texts by Harrison [1965], Hartmanis-Stearns [1966], and Arbib [1969].

Definition 1.5

M_1 is equivalent to M_2 iff $\Sigma_1 = \Sigma_2 \triangleq \Sigma$

and

(1) for all $q \in Q$, there exists $p \in Q_2$ such that

$$\lambda_1(q,x) = \lambda_2(p,x), \text{ for all } x \in \Sigma^*$$

(2) for all $p' \in Q_2$, there exists $q' \in Q_1$ such that

$$\lambda_1(q',x) = \lambda_2(p',x), \text{ for all } x \in \Sigma^*$$

i.e. M_1 and M_2 exhibit the same input-output behavior.

Definition 1.6

M_1 realizes M_2 iff

there exists an assignment $\psi = (\alpha,\rho,\xi)$ of M_2 into M_1 denoted by

$$\psi: M_2 \rightarrow 2^{M_1}$$

such that

(1) $\alpha: Q_2 \xrightarrow{\text{into}} 2^{Q_1}$ (subsets of Q_1)

i.e. α maps Q_2 into disjoint subsets of Q_1

(2) $\rho: \Sigma_2 \xrightarrow{\text{into}} \Sigma_1$

(3) $\xi: \Delta_1 \xrightarrow{\text{into}} \Delta_2$

so that the following are satisfied

(a) $\delta_1\left[\alpha(p),\rho(x)\right] \subseteq \alpha\left[\delta_2(p,x)\right]$ for all $p\varepsilon Q_2$, for all $x\varepsilon\Sigma_2$

(b) $\xi\left[\lambda_1(q)\right] = \lambda_2(p)$ for all $q\varepsilon\alpha(p) \subseteq 2^{Q_1}$

It shall be advantageous to identify and use different concepts of realizability, hence call the above an A-type realization.

Definition 1.7

M_2 is a homomorphic image of M_1 iff there exists a 3-part homomorphism $\phi = (h_1,h_2,h_3)$ such that

(1) $h_1: Q_1 \xrightarrow{\text{onto}} Q_2$

(2) $h_2: \Sigma_1 \xrightarrow{\text{onto}} \Sigma_2$

(3) $h_3: \Delta_1 \xrightarrow{\text{onto}} \Delta_2$

so that

(a) $h_1\left[\delta_1(q,x)\right] = \delta_2\left[h_1(q),h_2(x)\right]$ for all $q\varepsilon Q_1$, for all $x\varepsilon\Sigma_1$

(b) $h_3\left[\lambda_1(q)\right] = \lambda_2\left[h_1(q)\right]$

Theorem 1.1 (Hartmanis and Stearns)

If M_1 realizes M_2 and M_2 is reduced (i.e. no input-output equivalent states) then M_2 is a homomorphic image of a submachine of M_1 when and only when the map ρ is 1-1. That is:

given M_1 realizes M_2 then by definition

$$\psi : M_2 \longrightarrow 2^{M_1} \quad \text{with} \quad \psi = (\alpha, \rho, \xi)$$

now iff ρ is 1-1

then there exists $\phi = (h_1, h_2, h_3)$ such that

$$\phi : M_1\Big|_{\psi} \longrightarrow M_2$$

whereby

$$M_1\Big|_{\psi} = \left\langle Q_1^{\psi}, \Sigma_1^{\psi}, \Delta_1^{\psi}, \delta_1^{\psi}, \lambda_1^{\psi}, q_0 \right\rangle$$

where

$$Q_1^{\psi} = \bigcup_q \left\{ \alpha(q) \;\middle|\; q \in Q_2 \right\}$$

$$\Sigma_1^{\psi} = \bigcup_x \left\{ \rho(x) \;\middle|\; x \in \Sigma_2 \right\}$$

$$\Delta_1^{\psi} = \Delta_1$$

$$\delta_1^{\psi} = \delta_1 \qquad \text{restricted to } Q^{\psi} \times \Sigma^{\psi}$$

$$\lambda_1^{\psi} = \lambda_1 \qquad \text{restricted to } Q^{\psi}$$

then $M_1\Big|_{\psi}$ is a submachine of M_1 and the functions

$$h_1(q) = p \qquad \text{where } q \varepsilon \alpha(p)$$

$$h_2 = \rho^{-1}$$

$$h_3 = \xi$$

map $M_1\Big|_{\psi}$ onto M_2.

Denote both these type realizations, H-type.

A third notion of realization which shall be referred to as C-type is defined as follows:

Definition 1.8

M_1 has the capability of M_2 iff given M_1 and M_2, there exists an assignment $\psi^* = (\alpha, \rho^*, \xi)$ of M_2 into M_1 denoted by $\psi^* : M_2 \longrightarrow 2^{M_1}$ such that

(1) $\alpha: Q_2 \xrightarrow{\text{into}} 2^{Q_1}$

(2) $\rho^*: \Sigma_2 \xrightarrow{\text{into}} \Sigma_1^*$

(3) $\xi: \Delta_1 \xrightarrow{\text{into}} \Delta_2$

and

$$\text{(a)}\quad \delta_1\left[\alpha(p),\rho^*(x)\right] \subseteq \alpha\left[\delta_2(p,x)\right] \text{ for all } p\in Q_2 \text{ for all } x\in\Sigma_2$$

$$\text{(b)}\quad \xi\left[\lambda_1(q)\right] = \lambda_2(p) \text{ for all } q\in\alpha(p) \subseteq 2^{Q_1}$$

The H-type realization requires that for any machine, M_1, to realize another machine, M_2, M_2 must represent a homomorphic image of a submachine of M_1--that is M_1 restricted to a subset of its states and subsets of the input and output alphabets with the transition and output functions defined over these restricted co-domains. When M_2 is a homomorphic image of M_1, M_2 is a partial computation of M_1.

In terms of machine structure, i.e. the states and transition function, these partial computations are most conveniently described by SP partitions--partitions (or equivalence relations) having the substitution property (or right congruence).

Definition 1.9

An SP partition, or right congruence relation, on the set of states of M_1 is an equivalence relation on Q_1 such that

$$q_i R q_j \text{ iff } \delta_1(q_i,\sigma)\ R\ \delta_1(q_j\sigma);\quad \text{for all } \sigma\in\Sigma$$

The blocks of the partition are the disjoint equivalence classes of the relation, R,

$$\pi = \left\{B_j\right\}_{j=1,m} \quad ; \quad \left\{B_j\right\}_{j=1,m} = \left\{R(q_i)\right\}_{i=1,n_1}$$

The machine defined by

$$M_\pi = \left\langle \left\{B_j\right\}, \Sigma_1, \delta_\pi \right\rangle$$

$$\delta_\pi(B_j,\sigma) = B_K \quad \text{iff} \quad \delta(q_i,\sigma)\varepsilon B_K \text{ for all } q_i \varepsilon B_j$$

provides a partial description of the way information "flows through" M_1. The SP lattice of M_1, L_{M_1}, is a hierarchial organization of these units of self-contained state transition information. Clearly then, all submachines represented by SP partitions on M_1 are homomorphic images of M_1.

Definition 1.10

The corresponding SP lattice is defined as a partial ordering on the SP partitions

$$L_{M_1} = \left\langle \left\{\pi_i^{M_1}\right\}, +, \circ, 0, I, \leq \right\rangle$$

where $\circ$ and $+$ are binary operations satisfying the idem-

potency, commutativity, associativity and absorption laws, and for which a greatest lower bound (glb) and a least upper bound (lub) can be defined for every pair $\left(\pi_i^{M_1}, \pi_j^{M_1}\right)$. The elements, 0 and I, are the glb and the lub for the set $\left\{\pi_i^{M_1}\right\}$. The reader is referred to Hartmanis and Stearns [1966] for the meaning of $\pi_i \circ \pi_j$ and $\pi_i + \pi_j$.

At this point it should be emphasized that state transition performance realization, i.e. realizability as in Definition 1.6 above but without the output mapping, λ, guarantees realization in the full (i.e. with output) sense for every machine in some class, $\left\{M_i\right\}$, by some state machine M_1 when M_1 takes on the output function of the M_i it is to realize and each M_i is a Moore machine.

It has been noted by many researchers that the semigroup of a machine can serve in exactly this capacity. The semigroup is defined below and a few salient properties will be exposed.

Given a state machine

$$M_1 = \left\langle Q_1, \Sigma_1, \delta_1 \right\rangle \quad ; \quad Q_1 = \left\{q_i\right\}_{i=1,n_1}$$

consider the input strings† of Σ^* as mappings of Q_1 into Q_1:

$$x : Q_1 \xrightarrow{\text{into}} Q_1 \quad \text{iff} \quad x(Q_1) = \left\{\delta(q_i,x)\right\}_{\substack{i=1,n_1 \\ x\in\Sigma^*}}$$

For an FSM there are only a finite number of such mappings--described by a finite partition of Σ^*,

$$\Sigma^* = \underset{x\in\Sigma^*}{U}\left\{f_x\right\} : \; x,y\in f_x = f_y \text{ iff } x(Q_1) = y(Q_1) \; ; \; x,y\in\Sigma^*$$

Definition 1.11

The semigroup of M_1, S_{M_1}', is this set of mappings which is closed under composition

$$S_{M_1}' = \left\langle \left\{f_x\right\}, \circ \right\rangle_{x\in\Sigma^*}$$

where

$$f_x \circ f_y[Q_1] = f_y\left[f_x(Q_1)\right] = f_z(Q_1) \; ; \; z = xy \; ; \; z,x,y\in\Sigma^*$$

Closure is certainly apparent and adding the identity element defines the related monoid where

$$S_{M_1}' = \left\langle \left\{f_x\right\}, \circ, \Lambda \right\rangle$$

†Here and in what follows, Σ^* will include the null word and is usually referred to as the free monoid on Σ.

with

$$f_x \circ \Lambda = \Lambda \circ f_x = f_x$$

An alternative mathematical characterization of S'_{M_1} is a congruence relation on Σ^*

$$R_S = \left\{f_x\right\} \; : \; xR_S y \quad \text{iff} \quad x(Q_1) = y(Q_1);\; x,y \in f_x = f_y$$

R_S is both right and left invariant because

$$\text{if } xR_S y,\; uR_S v \quad \text{then} \quad f_x \circ f_u = f_y \circ f_u = f_y \circ f_v$$

$$\text{and} \quad xu(Q_1) = yv(Q_1)$$

$$\text{and} \quad xuR_S yv$$

A state machine representing this semigroup is defined with the elements of the semigroup serving as both the input and state sets. Let

$$S'_{M_1} = \left\langle \left\{f_x\right\}, \left\{f_x\right\}, \delta \right\rangle$$

where

$$\delta(f_x, f_y) = f_x \circ f_y = f_{xy}$$

and denote by S_{M_1} the machine obtained from S'_{M_1} by restricting the input set to the generators of the semigroup, i.e. Σ_1. This machine is described by the system

$$S_{M_1} = \left\langle \left\{f_x\right\}_{x\in\Sigma^*} , \left\{f_\sigma\right\}_{\sigma\in\Sigma} , \delta \right\rangle$$

Both the semigroup machine (or accumulator) and the related semigroup will be so designated and either will be explicitly referred to in case of confusion.

S_{M_1} has all the power of S_{M_1}' since $S_{S_{M_1}}' \tilde{=} S_{M_1}'$ as is easily shown.

It will prove beneficial to relate S_{M_1} to a direct product of machines in the following fashion.

Definition 1.12

The direct product of two machines M_1 and M_2 is defined for

$$(\Sigma_1 = \Sigma_2 \triangleq \Sigma)$$

as

$$M_1 \times M_2 = \left\langle Q_1\ Q_2,\ \Sigma,\ \Delta_1\times\Delta_2,\ \delta_x,\ \lambda_x,\ (q_0,\ p_0)\right\rangle$$

where

$$\delta_x\Big([q_i,p_j],\sigma\Big) = \Big(\delta_1(q_i,\sigma),\delta_2(p_j,\sigma)\Big)$$

$$\lambda_x\Big([q_i,p_j],\sigma\Big) = \Big(\lambda_1(q_i),\lambda_2(p_j)\Big)$$

Definition 1.13

The connected machine, $(M_1 \times M_2)^c$, has as its state set

$$(Q_1 \times Q_2)^r \subseteq Q_1 \times Q_2$$

all pairs of states which can be reached by input strings on Σ from the initial state (q_0, p_0).

Now given the machine

$$M = \langle Q, \Sigma, \Delta, \delta, \lambda \rangle , \quad |Q| = n$$

define all possible machines by considering each state $q_i \in Q$ as a start state

$$\{M_i\}_{i=1,n} = \{ \langle Q, \Sigma, \Delta, \delta, q_i \rangle \mid q_i \in Q \}$$

then the following properties follow:

(i) $$S_M \cong \left(\underset{i=1}{\overset{n}{\times}} M_i \right)^c = \left\langle \left(\underset{n}{\times} Q \right)^r, \Sigma, \delta_{S_M} \right\rangle$$

(ii) $$S_{S_M} \cong \left(\underset{j=1}{\overset{n}{\times}} S_{M_j} \right)^c \cong S_M$$

where

$$S_{M_j} = \langle \{f_x\}, \Sigma, \delta_{S_M}, f_j \rangle \; ; \; f_j \in \{f_x\} ; \; |\{f_x\}| = m$$

(iii) $S_{\left(\times_{i=1}^{n} M_i\right)^c} \tilde{=} S_{\left(\times_{i=1}^{n} M_i\right)}$

that is the connected submachine of $\left(\times_{i=1}^{n} M_i\right)$, with $(q_1,\ldots q_n)$ as its starting state, computes as much as $\left(\times_{i=1}^{n} M_i\right)$.

One purpose of presenting these relationships* was to show that in one sense the concept of a semigroup of a FSM is identical with that of a direct product of certain algebras defined on the machine. The usefulness of this tie-in is apparent in noting that direct product algebras usually guarantee the existence of homomorphisms from the direct product onto each factor or subset of factors. The set of all such homomorphisms with an appropriate definition of some partial ordering, is isomorphic to the SP lattice of S_M.

The mathematical correspondence of S_M and $\left(\times_{i=1}^{n} M_i\right)$ also provides insight into the synthesis (and decomposition) of realization machines. Given any set of state machines with no initial states specified, the direct product machine

*These properties are actually paraphrasing the well known fact that, mathematically, the connected portion of any FSM is all that need be considered.

includes all connected submachines which could be defined by different sets of initial states. The semigroup of the direct product can realize any of these connected submachines alone but generally not two or more disconnected portions simultaneously. The SP lattice of the semigroup machine is the ordering of those machines and can be used to detect certain decompositions of any of them.

The response function and equiresponse relation and the notion of a cyclic machine conclude this section.

Definition 1.14

The response function of a machine M to any tape in Σ^* is the mapping

$$rp_M(x) = \delta_M(q_0,x) \quad x\epsilon\Sigma^* \; ; \qquad rp_M(x)\epsilon Q_M$$

Definition 1.15

The equiresponse relation of a machine M is a relation E_M, on Σ^* defined by

$$xE_My \text{ iff } rp_M(x) = rp_M(y)$$

E_M is a finite, right congruence relation and the equivalence classes are denoted by $E_M(x)$, $x\epsilon\Sigma^*$.

Definition 1.16

Machine M is cyclic if there exists some starting

state, q_0, such that

$$Q_M = \left\{ q_k \;\middle|\; \delta_M(q_0, x) = q_k, \text{ some } x \in \Sigma^* \right\}$$

CHAPTER II

CONTROL EQUIVALENCE AND STRUCTURAL REALIZABILITY

In this chapter the concepts of structural realizability, program value, evaluation sequences and program equivalence will be introduced and interrelated.

2.1 Program Value and Valuation Sequences

Definition 2.1

Given a program, $\P$, on a computer representation, C

$$\P = I\,[\tau]$$

$$\tau = \langle Q, \Sigma, Y, \delta, \lambda, q_0 \rangle$$

$$I = \langle \eta, \gamma, \zeta, d_0 \rangle$$

$$I : \tau \longrightarrow C = \langle M, B, D, F, P \rangle$$

then the value of $\P$ given d_0 is

$$V_{d_0}(\P) = \left\{ f_1, f_2, \ldots f_n \right\},\ f_i \varepsilon F^*$$

where for each f_i there is a state $q_i \varepsilon Q$ such that

$$\zeta\left[\lambda(q_i)\right] = f_i \ .$$

The value is defined only when n is finite.

Definition 2.2

For each $d_0 \epsilon D$ in an interpreted schemata, $\P$, with deterministic FSM control, the string in Σ^*, called the valuation sequence $S_{d_0}(\P)$ on Σ is the tape

$$S_{d_0}(\P) = \left\{\sigma_1, \sigma_2 \ldots \sigma_n\right\}, \ \sigma_i \epsilon \Sigma$$

such that if

$$V_{d_0}(\P) = \left\{f_1, f_2, \ldots f_n\right\}, \quad f_i \epsilon F^*$$

then

$$f_1 = \zeta \left(\lambda[\delta(s_0, \sigma_1)]\right)$$

$$f_2 = \zeta \left(\lambda[\delta(s_0, \sigma_1 \sigma_2)]\right)$$

$$\vdots$$

$$f_n = \zeta \left(\lambda[\delta(s_0, \sigma_1 \ldots \sigma_n)]\right)$$

Definition 2.3

Let $\P = I[\tau]$ be a program with a deterministic FSM control

$$\tau = \langle Q, \Sigma, Y, \delta, \lambda, q_0 \rangle$$

$$I = \langle \eta, \gamma, \zeta, d_0 \rangle$$

$$I : \tau \longrightarrow c$$

Define the initial configuration - free program, $\overline{\P}$,

where

$$\overline{\P} = \overline{\overline{I[\tau]}}$$

$$\overline{\tau} = \langle Q, \Sigma, Y, \delta, \lambda, q_0 \rangle$$

$$\overline{I} = \langle \eta, \gamma, \zeta \rangle, \text{ no initial configuration specified}$$

Notation:

Let $$\tilde{S}_D(\overline{\P}) = \left\{ S_d(\overline{\P}) \right\}_{d \in D}$$

be the set of all valuation sequences on Σ for all initial configurations for $\overline{\P}$.

Let $$\tilde{V}_D(\overline{\P}) = \left\{ V_d(\overline{\P}) \right\}_{d \in D}$$

be set of all program values for all initial configurations for $\overline{\P}$.

Now the valuation sequence can be looked upon as an encoding--in terms of strings on a finite alphabet--of the environmental responses governing the action of the program. Some rather obvious relationships between the valuation sequences and program values are described by the following theorems.

Theorem 2.1

Let ¶ be as in Definition 2.3, then for all $d \in D$, there exists at least one $S_d(\overline{¶})$ such that $V_d(\overline{¶})$ is always the interpreted behavior of τ under the input tape $S_d(\overline{¶})$. Moreover each $S_d(\overline{¶})$ is associated with a unique $V_d(\overline{¶})$.

Proof:

Clear from the assumption of τ being a deterministic FSM and Definition 2.3. $\triangleright\triangleleft$

Notation:

Let G be the map of Theorem 2.1,

$$G: \tilde{V}_D(\overline{¶}) \xrightarrow[1\text{-}1]{\text{into}} \tilde{S}_D(\overline{¶}) \quad \text{(1-1 by Axiom of Choice)}.$$

Let W be the map of Definition 2.2,

$$W: \tilde{S}_D(\overline{¶}) \xrightarrow{\text{into}} \tilde{V}_D(\overline{¶}).$$

Corollary 2.2

W is an onto map.

Proof:

From Definition 2.2,

$$W\,[\tilde{S}_D(\overline{¶})] \subseteq \tilde{V}_D(\overline{¶}).$$

From Theorem 2.1

$G\,[\hat{\mathcal{V}}_D(\overline{\P})] \subseteq \hat{\mathcal{S}}_D(\overline{\P})$, therefore

$W\,[\hat{\mathcal{S}}_D(\overline{\P})] = \hat{\mathcal{V}}_D(\overline{\P})$. ▷◁

The same is not necessarily true of G because of the way in which the single valuedness of G was imposed.

So

$$G \circ W\,[\hat{\mathcal{V}}_D(\overline{\P})] = \hat{\mathcal{V}}_D(\overline{\P})$$

but

$$W \circ G\,[\hat{\mathcal{S}}_D(\overline{\P})] \subseteq \hat{\mathcal{S}}_D(\overline{\P}).$$

Thus while each valuation sequence defines a unique program value there may be a number of such sequences associated with each distinct value.

This becomes an important consideration in the degree to which an interpreted control can model practical programs. More will be said about this later on. There are two situations in which a unique relation can be made between elements of $\hat{\mathcal{S}}_D(\overline{\P})$ and those of $\hat{\mathcal{V}}_D(\overline{\P})$.

Lemma 2.3

Given ¶ as in Definition 2.3, if

(1) the maps

$\lambda: Q \longrightarrow Y$

$\zeta: Y \longrightarrow F^*$

are injective (1-1, into)--so also the F^* projection of η is injective--and

(2) there is no state $q_i \varepsilon Q$ such that

$$\delta_\tau(q_i,\sigma_1) = \delta_\tau(q_i,\sigma_2) \text{ for all } \sigma_1,\sigma_2 \varepsilon \Sigma$$

(3) then W is bijective (1-1, onto).

Proof:

(1) Note that W is already onto by Theorem 2.1.

To show W is also injective,

for any

$$V_d(\overline{\P}) = \left\{f_1,f_2,\ldots f_n\right\}.$$

Since λ and ζ are injective then for each f_i in $V_d(\overline{\P})$ a unique state exists

$$q_i \varepsilon Q, \text{ such that } \zeta\left(\lambda(q_i)\right) = f_i.$$

Suppose there were two sequences

$$S_d(\overline{\P}) = \left\{\sigma_1,\sigma_2,\ldots \sigma_n\right\}$$

$$S_d'(\overline{\P}) = \left\{\sigma_1',\sigma_2',\ldots \sigma_n'\right\}$$

such that

$$W[S_d(\overline{\P})] = W[S_d'(\overline{\P})] = V_d(\overline{\P}) = \left\{f_1,\ldots f_n\right\}$$

where

$$\zeta\left(\lambda[rp_{q_0}(\sigma_1,\sigma_2,..\sigma_i)]\right) = \zeta\left(\lambda[rp_{q_0}(\sigma_1',\sigma_2',..\sigma_i')]\right) = f_i$$

for i = 1,n.

Since ζ and λ are injective, then

$$rp_{q_0}(\sigma_1,\sigma_2,..\sigma_i) = rp_{q_0}(\sigma_1',\sigma_2',..\sigma_i'); i = 1,n.$$

Now because this is true for all i = 1,n, and since τ is deterministic,

$$rp_{q_0}(\sigma_1) = rp_{q_0}(\sigma_1')$$

since by condition (2) above

$$\delta_\tau(q_0,\sigma_1) \neq \delta_\tau(q_0,\sigma_1'),\ \sigma_1 \neq \sigma_1' .$$

Similarly,

$$\delta_\tau(q_0,\sigma_1\sigma_2) = \delta_\tau\left(rp_{q_0}(\sigma_1),\sigma_2\right)$$

$$= \delta_\tau(q_i,\sigma_2) = \delta_\tau(q_i,\sigma_2')$$

hence

$$\sigma_2 = \sigma_2' .$$

By induction then,

$$\sigma_i = \sigma_i' ,\ i = 1,n .$$

Hence W is 1-1.

$\rhd\lhd$

The following example illustrates a nonbijective W when condition (1) is violated.

Example 2.1

Suppose δ_τ, λ_τ and ζ and Σ_τ were given by

δ_τ:

	0	1	λ_τ	$\zeta(Y_i)$
q_0	q_1	q_2	Y_1	f_1
q_1	q_0	q_3	Y_2	f_2
q_2	q_3	q_2	Y_3	f_3
q_3	q_0	q_2	Y_4	f_1

$$\Sigma_\tau = \{0,1\}$$

Clearly two sequences exist, {011}, {001} such that for some ¶ with this τ and ζ in its interpretation if

$$V_{d_0}(\overline{¶}) = \{f_2, f_1, f_3\}$$

then both

$$S_{d_0}(\overline{¶}) = \{011\}$$

$$S'_{d_0}(\overline{¶}) = \{001\}$$

are valuation sequences.

However, condition (1) alone is not necessary as can be seen by the next example.

Example 2.2

	0	1	λ_τ	$\zeta(Y_i)$
q_0	q_1	q_2	Y_1	f_1
q_1	q_1	q_2	Y_2	f_2
q_2	q_1	q_3	Y_3	f_3
q_3	q_1	q_3	Y_1	

Note λ_τ is not injective but it can be verified that no two distinct input tapes of the same length generate the same sequence, $\left(\{f_i\}\right)_{i\geq 1}$. But it should be clear that condition (2) is always necessary. These sufficient conditions are easy to check. The proof of this lemma and the last example indicate the somewhat more cumbersome but necessary and sufficent conditions.

Theorem 2.3

Given ¶ as in Definition 2.3 iff

(1) for the maps

λ_τ: $Q_\tau \longrightarrow Y$

ζ : $Y \longrightarrow F^*$

δ_τ: $Q_\tau x \Sigma_\tau \longrightarrow Q_\tau$

there is no state $q_i \varepsilon Q_\tau$ such that

$$\zeta\left[\lambda_\tau\left(\delta_\tau(q_i,\sigma_1)\right)\right] = \zeta\left[\lambda_\tau\left(\delta_\tau(q_i,\sigma_2)\right)\right] \text{ for all } \sigma_1,\sigma_2 \varepsilon \Sigma_\tau \text{ and } \sigma_1 \neq \sigma_2$$

then W is bijective.

Proof:

(1) Following the previous lemma: given (1) and

$$S_d(\overline{\P}) = \left\{\sigma_1 \ldots \sigma_n\right\} \qquad \sigma_i, \sigma_i' \varepsilon \Sigma_\tau$$

$$S_d'(\overline{\P}) = \left\{\sigma_1' \ldots \sigma_n'\right\}$$

$$W[S_d(\overline{\P})] = W[S_d'(\overline{\P})] = \left\{f_1, f_2 \ldots f_n\right\} = V_d(\overline{\P})$$

then

$$\zeta\left(\lambda_\tau[rp_{q_0}(\sigma_1)]\right) = \zeta\left(\lambda_\tau[rp_{q_0}(\sigma_1')]\right) = f_1$$

hence

$$\zeta\left(\lambda_\tau[\delta_\tau(q_0,\sigma_1)]\right) = \zeta\left(\lambda_\tau[\delta_\tau(q_0,\sigma_1)]\right)$$

and $\sigma_1 = \sigma_1'$ because of (1) above.

By induction then $\sigma_i = \sigma_i'$ for all i, hence W is bijective.

(2) If W is bijective, then there exists no two evaluation sequences $S_d(\overline{\P})$ and $S_d'(\overline{\P})$ for $V_d(\overline{\P})$ as above so that

$$\zeta\left(\lambda_\tau[rp_{q_0}(\sigma_1\ldots\sigma_i)]\right) = \zeta\left(\lambda_\tau[rp_{q_0}(\sigma_1'\ldots\sigma_i')]\right) = f_i$$

for all i = 1,n

where

$$(\sigma_1\ldots\sigma_i) \neq (\sigma_1'\ldots\sigma_i').$$

(3) Then in particular

$$\zeta\left(\lambda_\tau[rp_{q_0}(\sigma_1)]\right) \neq \zeta\left(\lambda_\tau[rp_{q_0}(\sigma_1')]\right)$$

for all $\sigma_1,\sigma_1'\varepsilon\Sigma_\tau,\sigma_1 \neq \sigma_1'$

and if $rp_{q_0}(\sigma_1) = q_1$,

then

$$\zeta\left(\lambda_\tau[\delta_\tau(q_0,\sigma_1)]\right) \neq \zeta\left(\lambda_\tau[\delta_\tau q_1,\sigma_1')]\right).$$

(4) By induction on (3),

$$\zeta\left(\lambda_\tau[rp_{q_0}(\sigma_1\ldots\sigma_{i-1},\sigma_i)]\right) = \zeta\left(\lambda_\tau[\delta_\tau(q_0,\sigma_1\ldots\sigma_{i-1},\sigma_i)]\right)$$

$$\zeta\left(\lambda_\tau[rp_{q_0}(\sigma_1 \ldots \sigma_{i-1}, \sigma_i)]\right) = \zeta\left(\lambda_\tau[\delta_\tau(q_{i-1}, \sigma_i)]\right)$$

$$\neq \zeta\left(\lambda_\tau[\delta_\tau(q_{i-1}, \sigma_i')]\right)$$

$$\sigma_i \neq \sigma_i' .$$

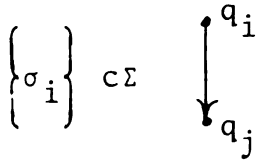
And since in general there is some path to every state in Q_τ required by some $d \in D$, this condition must hold for every state $\in Q_\tau$. (Note that condition (2) of Lemma 2.3 is always necessary for condition (1) of Theorem 2.3.) $\triangleright\triangleleft$

The reasons for presenting these Theorems are twofold. First, it is sometimes important to know if there is one and only one encoding (within an isomorphism) of the environmental response which characterizes any action of a program. For example, if a file structure and the control of any program manipulating that structure can be modeled by a common FSM and the conditions of Theorem 2.3 hold for both an interpretation of the FSM as the file structure, then every manipulation of the file possible by the program can be described in both effect and process by one input tape to the control. Secondly, if a programmer can fit his programs and information structures to this model,

he can bring a considerable amount of theory to bear in detecting common attributes even with only rudimentary knowledge of FSM theory.

It is worthwhile noting that when $\hat{S}_D(\overline{\pi})$ and $\hat{V}_D(\overline{\pi})$ are isomorphic and one program value is identified uniquely by one valuation sequence, all elements of the interpretation and control of $\overline{\pi}$ must remain constant except the initial configuration $d_0 \varepsilon D$. Given two programs $\overline{\pi}_1$, $\overline{\pi}_2$ with identical controls and input alphabets but different interpretations, it is certainly possible for one tape on Σ^* to be the valuation sequence for two different values $V_d(\overline{\pi}_1)$, $V_d(\overline{\pi}_2)$ even though the conditions of Theorem 2.3 are satisfied for each program. The point is that different interpretations on a common control relate environmental responses, to different decision predicates, with dissimilar computation sequences.

Also, it should be realized that the condition in Theorem 2.3 is a harsh one to impose. This is because it says that no program can be structured in the form



That is, there cannot be an unconditional transfer for some subset of input symbols from one set of statements

identified by q_i to another set q_j.

Therefore, since different interpretations on a common control and unconditional control transfers destroy the isomorphic relation between environmental response and program value, it is desirable to approach the notion of program equivalence from the point of view of its values and not the valuation sequences. Moreover, when equivalence preserving control transformations are introduced they have the property that even though $\tilde{S}_D(\bar{\pi}) \neq \tilde{V}_D(\bar{\pi})$, there is a specified mapping from $\tilde{S}_D(\bar{\pi})$ into $\tilde{S}_D(\bar{\pi}')$ where $\bar{\pi}$ and $\bar{\pi}'$ are value equivalent.

2.2 Program and Control Equivalence

A number of investigators have studied the question of equivalence between schemata. Generally, two schemata are equivalent if they have the same value under all interpretations. Patterson [1967] defined value as success, failure or divergence and, using a very general model for schemata, showed this problem is undecidable. Rutledge [1964] demonstrated that Ianov's model of schemata was equivalent to a finite machine and that Ianov's equivalence between two schemata--defined as identical value over all valuation sequences--was therefore decidable.

The difference between these two points of view is how much specificity about the program behavior is con-

sidered in the definition of equivalence. Patterson's schemata require an interpretation to specify decision predicates and assignment statements but allow different portions of memory to be operated upon by the assignment operators. In this case different paths through two interpreted schemata do not necessarily mean the schemata perform different functions. In general, the problem of deciding whether two such schemata are equivalent is unsolvable.

In contrast, the Ianov model assumed a specific input-output function for a schema and examined only "admissable interpretations"--that is, those assignments of decision predicates and computations which conformed to the I/O function of the schema. This restriction simplified the question of equivalence between two such schemata to one of behavioral equivalence in automata theory.

So decidability problems being what they are, can effective use be made of an interpreted control model of a program under the restricted notions of FSM equivalence and the total memory requirement for any assignment transaction? A sizable portion of the rest of this thesis is devoted to substantiating an affirmative reply to this question.

Definition 2.4

Two programs $\P_1$, $\P_2$ are V - equivalent, $\P_1 \underset{V}{\equiv} \P_2$ iff for all initial configurations $d \in D$,

$$V_d(\overline{\P}_1) = V_d(\overline{\P}_2).$$

Hence

$$\tilde{V}_D(\overline{\P}_1) = \tilde{V}_D(\P_2).$$

Notice the controls τ_1, τ_2 and the interpretations I_1, I_2 remain intact except for the initial configurations. As a consequence, program equivalence can be easily investigated from the point of view of FSM behavior. The path taken here is to not look at the behavior of the interpreted control but to examine the control structure itself and specify what the interpretations must be for structurally related controls to preserve program value. This point of view allows one to keep the concept of control separate from that of interpretation. As stated this is an equivalence relation among all programs possible for a given computer representation. Two binary relations on controls can now be defined in terms of program value equivalence.

Definition 2.5

Given two FSM deterministic controls τ_1, τ_2 and a com-

puter representation, C, τ_1 is <u>I-equivalent</u> to τ_2, $\tau_1 \underset{I}{\equiv} \tau_2$, iff

(1) for all interpretations on τ_1, $\{I_{1i}\}$, there exist interpretations, $\{I'_{2i}\}$, such that

$$I_{1i}\left[\tau_1\right] \underset{\bar{V}}{\equiv} I'_{2i}\left[\tau_2\right]$$

and

(2) for all interpretations on τ_2, $\{I_{2i}\}$, there exist interpretations, $\{I'_{1i}\}$, such that

$$I_{2i}\left[\tau_2\right] \underset{\bar{V}}{\equiv} I'_{1i}\left[\tau_1\right] .$$

Trivially then,

<u>Theorem 2.4</u>

The relation, $\underset{I}{\equiv}$, is an equivalence relation. ▷◁

<u>Definition 2.6</u>

Given the controls, τ_1 and τ_2, the corresponding I-equivalence classes, $\underset{I}{\equiv}(\tau_1)$ and $\underset{I}{\equiv}(\tau_2)$ are <u>R_I-related</u>, $\underset{I}{\equiv}(\tau_1) R_I \underset{I}{\equiv}(\tau_2)$, iff for all interpretations $\{I_i\}$ on any control $\tau \in \underset{I}{\equiv}(\tau_1)$, there exist interpretations $\{I'_{ij}\}$ for every control $\tau_j \in \underset{I}{\equiv}(\tau_2)$ such that

$$I_i\left[\tau\right] \underset{\bar{V}}{\equiv} I'_{ij}\left[\tau_j\right], \text{ for all i and all j.}$$

Theorem 2.5

The relation, R_I, is reflexive, transitive and anti-symmetric (a partial ordering).

Proof:

From Definition 2.5, R_I is reflexive. The transitivity of R_I follows from that of $\equiv_{\bar{V}}$. Also, according to Definition 2.5, if $\underset{I}{\equiv}(\tau_1) R_I \underset{I}{\equiv}(\tau_2)$ and $\underset{I}{\equiv}(\tau_2) R_I \underset{I}{\equiv}(\tau_1)$ then necessarily $\underset{I}{\equiv}(\tau_1) = \underset{I}{\equiv}(\tau_2)$. Hence, R_I is anti-symmetric. ▷◁

R_I is a partial ordering on the $\underset{I}{\equiv}$ equivalence classes. For convenience the equivalence class notation will be dropped and the abbreviated representation, $\tau_1 R_I \tau_2$, will be used. The constructions which follow will verify the R_I relation between representative controls, τ_1 and τ_2. The change in notation emphasizes this. The R_I relation provides a convenient means of identifying the common procedure or control.

Definition 2.7

A control, Γ, is a common control for a class of controls, $\{\tau_j\}$, j=1,m, iff $\tau_j R_I \Gamma$; j=1,m.

Common controls will be constructed and specific types of R_I relations identified in the following sections. The relation $\underset{I}{\equiv}$ is fundamental to Chapter IV.

2.3 Structural Realizability of Control

An important consequence of relating the control of one program to that of another by means of an H-, A- or C-type realization is that the two controls are R_I related. The next three theorems demonstrate this and are proved by constructing the appropriate interpretations.

For the theorems which follow, let two FSM deterministic controls be represented

$$\tau = \langle Q_\tau, \Sigma_\tau, Y_\tau, \delta_\tau, \lambda_\tau, q_0 \rangle$$

$$|Q_\tau| = n$$

$$|\Sigma_\tau| = L$$

$$\Gamma = \langle S_\Gamma, \Sigma_\Gamma, Y_\Gamma, \delta_\Gamma, \lambda_\Gamma, s_0 \rangle$$

$$|S_\Gamma| = m$$

$$|\Sigma_\Gamma| = K \geq L$$

and given a fixed computer representation

$$C = \langle M, B, D, F, P \rangle$$

then assume the set of all interpretations on τ is denoted by

$$\{I_j\} = \{ \langle \eta_j, \gamma_j, \zeta_j, d_{0_j} \rangle \}_{j \geq 1}$$

where

$$\eta_j : Q_\tau \rightarrow F^* \times P^L$$

$$\gamma_j : Q_\tau \rightarrow P^L$$

$$\zeta_j : Y_\tau \rightarrow F^*$$

$$d_{0_j} \in D .$$

Theorem 2.6

Given τ, C and Γ, if Γ is an A- type realization of τ, then $\tau R_I \Gamma$.

This particular relation will be denoted $\tau A_I \Gamma$.

Proof:

(1) If Γ is an A- type realization of τ then via Definition 1.6 there exists an assignment

$$\psi = (\alpha, \rho, \xi)$$

such that

$$\psi : \tau \rightarrow 2^\Gamma$$

where

$$\alpha : Q_\tau \xrightarrow{\text{into}} 2^{S_\Gamma}$$

$$\rho : \Sigma_\tau \xrightarrow{\text{into}} \Sigma_\Gamma$$

$$\xi : Y_\Gamma \rightarrow Y_\tau$$

and

$$\delta_\Gamma[\alpha(q),\rho(x)] \subseteq \alpha[\delta_\tau(q,x)] \text{ for all } q\varepsilon Q_\tau, \text{ for all } x\varepsilon\Sigma_\tau$$

$$\xi[\lambda_\Gamma(s)] = \lambda_\tau(q) \text{ for all } s\varepsilon\alpha(q) \subseteq 2^{S_\Gamma}, \; q\varepsilon Q_\tau.$$

(2) The submachine of Γ which will be considered is

$$\Gamma\Big|_\psi = \left\langle S^\psi,\ \Sigma_\Gamma^\psi,\ Y_\Gamma^\psi,\ \lambda_\Gamma^\psi,\ s_0 \right\rangle$$

where

$$S^\psi = U\left\{\alpha(q) \,|\, q\varepsilon Q_\tau\right\}$$

$$\Sigma_\Gamma^\psi = U\left\{\rho(x) \,|\, x\varepsilon\Sigma_\tau\right\}$$

$$Y_\Gamma^\psi = Y_\Gamma$$

$$\delta_\Gamma^\psi = \delta_\Gamma \text{ restricted to } S^\psi x\Sigma_\Gamma^\psi$$

$$\lambda_\Gamma^\psi = \lambda_\Gamma \text{ restricted to } S^\psi$$

$$s_0\varepsilon\alpha(q_0).$$

(3) Then for any interpretation

$$I_j = \left\langle \eta_j, \gamma_j, \zeta_j, d_{0_j} \right\rangle$$

A new interpretation can be formed

$$I_j^A = \left\langle \eta_j^A , \gamma_j^A , \zeta_j^A , d_{0_j} \right\rangle$$

Where

(a) $\zeta_j^A : Y_\Gamma^\psi \longrightarrow F^* \; ; \quad \zeta_j^A = \xi \circ \zeta_j$

This composition map is defined by

$$\zeta_j^A (y) = \xi \circ \zeta_j (y) = \zeta_j [\xi(y)] = \zeta_j [y'] = f \varepsilon F^*$$

where for any $y \varepsilon Y_\Gamma^\psi$,

there exists a $q \varepsilon Q_\tau$, and a $y' \varepsilon Y_\tau$

such that

$$\lambda_\tau (q) = y' \varepsilon \; Y_\tau , \; \zeta_j \left[\lambda_\tau (q) \right] = \zeta_j \left[y' \right] = f$$

and a $s \varepsilon \alpha(q) \subseteq 2^S$

where $\lambda_\Gamma (s) = y$

and $\xi \left[\lambda_\Gamma (s) \right] = \xi \left[y \right] = y'$.

Then

$$\xi \circ \zeta_j (y) = \zeta_j \left[\xi (y) \right] = \zeta_j \left[\xi \left(\lambda_\Gamma (s) \right) \right] = \zeta_j \left[y' \right] = f .$$

(b) $\gamma_j^A : S^\psi \longrightarrow P^K \; ; \quad \gamma_j^A = \gamma_j \circ \rho$

where for any $s \varepsilon S^{\psi}$, there exists $q \varepsilon Q_{\tau}$ such that

$$s \varepsilon \alpha(q) \subseteq 2^{S}$$

and

$$\gamma_j(q) = p^L : D \longrightarrow \left\{\sigma_1, \sigma_2 \ldots \sigma_L\right\} = \Sigma_{\tau}$$

for

p^L some L-ary decision predicate on D.

But

$$\rho : \Sigma_{\tau} \longrightarrow \Sigma_{\Gamma} = \left\{\sigma_1', \sigma_2' \ldots \sigma_K'\right\}, \quad K \geq L$$

hence

$$\rho(\sigma_i) \varepsilon \Sigma_{\Gamma} \text{ for all } \sigma_i \varepsilon \Sigma_{\tau}.$$

Consequently for <u>all</u> $s \varepsilon \alpha(q)$

$$\gamma_j \circ \rho(s) = \rho\left[\gamma_j(q)\right] = \rho\left[p^L\right] = p^K.$$

The statement $\rho\left[p^L\right] = p^K$ is interpreted to mean that K-ary decision predicate p^K which makes the same test as p^L but encodes the responses as a subset of its own alphabet based on the map ρ.

Hence if

$$p^L : D \longrightarrow \left\{\sigma_1, \sigma_2 \ldots \sigma_L\right\}$$

then

$$\rho\left[p^L\right] = p^K : D \longrightarrow \left\{\rho(\sigma_1), \rho(\sigma_2), \ldots \rho(\sigma_L)\right\} \subseteq \Sigma_\Gamma .$$

Finally

(c) $$\eta_j^A : S^\psi \longrightarrow F^* \times P^K$$

where

$$\eta_j^A (s) = (f, p^K)$$

iff

$$\gamma_j^A (s) = p^K$$

and

$$\zeta_j^A \left[\lambda_\Gamma(s)\right] = f .$$

(4) This completes the construction of I_j^A . It can be seen that

$$V_d\left(\overline{I_j[\tau]}\right) = V_d\left(\overline{I_j^A\ [\Gamma]}\right) \text{ for all } d \in D,\ j \geq 1$$

and hence

$$I_j[\tau] \underset{V}{\equiv} I_j^A\ [\tau] \Big|_{j \geq 1} \Longrightarrow \tau A_I \Gamma$$

by considering how the control Γ behaves under the interpretation I_j^A . When τ is started in

state q_0, there is some computation, f_0, and an L-ary decision, p_0^L, which is made and the outcome, say $\sigma_i \varepsilon \Sigma_\tau$, is the L-ary alphabet symbol corresponding to the environmental response to the test. Control τ transfers to state $\delta_\tau(q_0,\sigma_i)$. In the same fashion control Γ started in any state $s_0 \varepsilon \alpha(q_0)$, effects the computation f_0--via $\zeta_j^A(s_0) = f_0$--and makes the same decision test as p_0^L but encodes the symbol σ_i--via the map $\rho(\sigma_i) = \sigma_i' \varepsilon \Sigma_\Gamma$--as a symbol σ_i' in K-ary input alphabet. The control Γ transfers on this control to some state $s_i \varepsilon \alpha\left(\delta_\tau(q_i,\sigma_i)\right)$, since by definition of a A- type realization

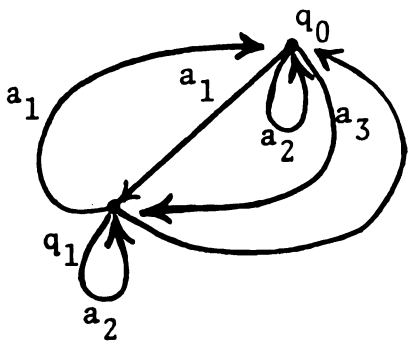
$$\delta_\Gamma\left(s_0,\rho(\sigma_i)\right) \varepsilon \alpha\left(\delta_\tau(q_0,\sigma_i)\right) \text{ for all } s_0 \varepsilon \alpha(q_0)$$

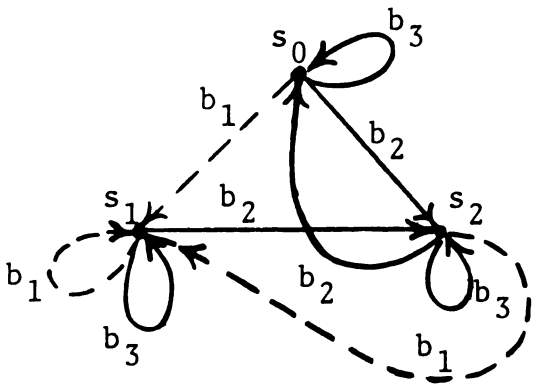
Continuing this process, both programs must give the same value for one interpretation on τ over all initial configurations $d \varepsilon D$. But the construction is "good" for any interpretation; hence the theorem is proved. [illegible]

Unfortunately, the notation clouds a deceptively simple process. All the information the programmer needs is there

but it is worthwhile to emphasize what and where it is by the following example.

Example 2.3





τ:

	Σ_τ			
	a_1	a_2	a_3	λ_τ
q_0	q_1	q_0	q_1	Z_1
q_1	q_0	q_1	q_0	Z_2

Figure 2.1

Two State Control, τ

Γ:

	Σ_Γ			
	b_1	b_2	b_3	λ_Γ
s_0	s_1	s_2	s_1	Y_1
s_1	s_1	s_2	s_1	Y_2
s_2	s_1	s_0	s_2	Y_3

Figure 2.2

An A-type Realization of τ

$$\Sigma_\Gamma = \{b_1, b_2, b_3\}$$

Now Γ is an A-type realization for τ since

$$\left.\begin{array}{ll} (1) & \alpha(q_0) = \{s_0, s_1\} \\ & \alpha(q_1) = \{s_2\} \end{array}\right\} \quad \alpha: Q_\tau \longrightarrow 2^{S_\Gamma}$$

$$\left.\begin{array}{ll} (2) & \rho(a_1) = \rho(a_3) = b_2 \\ & \rho(a_2) = b_3 \end{array}\right\} \quad \rho: \Sigma_\tau \longrightarrow \Sigma_\Gamma$$

$$\left.\begin{array}{ll} (3) & \xi(Y_1) = \xi(Y_2) = Z_1 \\ & \xi(Y_3) = Z_2 \end{array}\right\} \quad \xi: Y_\Gamma \longrightarrow Y_\tau$$

And it can be verified that

$$\delta_\Gamma\left[\alpha(q),\ \rho(a_i)\right] \subseteq \alpha\left[\delta_\tau(q, a_i)\right] \text{ for all } q \in Q_\tau \text{ and } a_i \in \Sigma_\tau$$

and also

$$\xi\left[\lambda_\Gamma(s)\right] = \lambda_\tau(q) \text{ for all } s \in \alpha(q).$$

Now suppose an interpretation, I, on τ was given by

(1) $\eta(q_0) = (f_0, p_0^3)$

$\eta(q_1) = (f_1, p_1^3)$

(2) $\gamma(q_0) = p_0^3$

$\gamma(q_1) = p_1^3$

(3) $\zeta(Z_1) = f_0$

$\zeta(Z_2) = f_1$.

Theorem 2.6 specifies the corresponding interpretation on Γ, I^A, to be

(1) $\zeta^A = \xi \circ \zeta$

$\xi \circ \zeta(Y_1) = \zeta\left[\xi(Y_1)\right]$

$= \zeta\left[Z_1\right] = f_0$

$\xi \circ \zeta(Y_2) = \zeta\left[\xi(Y_2)\right]$

$= \zeta\left[Z_1\right] = f_0$

$$\xi\circ\zeta(Y_3) = \zeta\left[\xi(Y_3)\right]$$

$$= \zeta\left[Z_2\right] = f_1$$

(2) $\gamma^A = \gamma\circ\rho$

$$\gamma^A(s_0) = \gamma\circ\rho(s_0) = \gamma\circ\rho(s_1)$$

$$= \rho\left[\gamma(q_0)\right] = \rho\left[p_0^3\right] = p_0^{3'}$$

where if

$$p_0^3(D) = a_1,\ \rho\left[p_0^3(D)\right] = p_0^{3'}(D) = b_2$$

or

$$p_0^3(D) = a_3,\ \rho\left[p_0^3(D)\right] = p_0^{3'}(D) = b_2$$

or

$$p_0^3(D) = a_2,\ \rho\left[p_0^3(D)\right] = p_0^{3'}(D) = b_3.$$

Similarly for p_1^3.

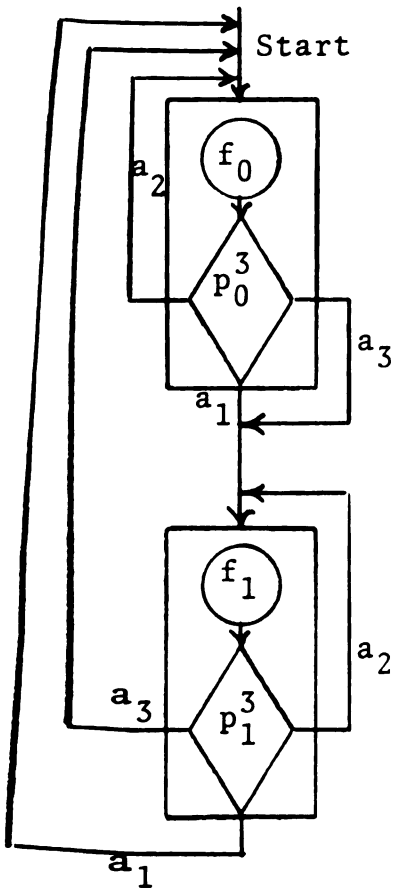
(3) Then

$$\eta^A(s_0) = (f_0, p_0^{3'})$$

$$\eta^A(s_1) = (f_0, p_0^{3'})$$

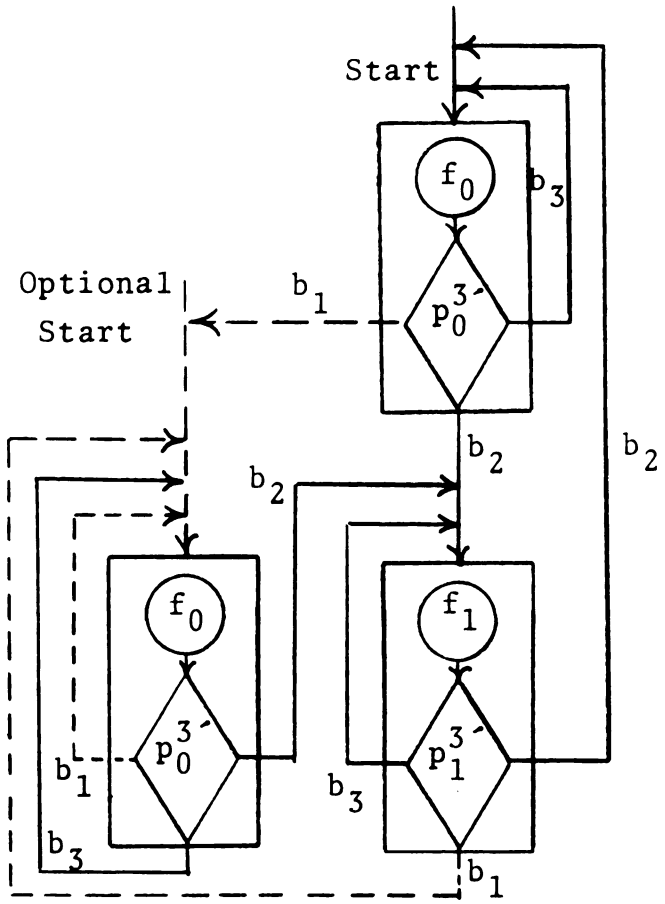
$$\eta^A(s_2) = (f_1, p_1^{3'}).$$

The program flow charts are shown in the figures that follow.



Flow Chart for $I[\tau]$

Fig. 2.3



Flow Chart for $I^A[\tau]$

Fig. 2.4

The solid lines in Fig. 2.4 and Fig. 2.2 indicate the only control flow paths allowable in the control Γ to structurally realize the control τ. The interpretation I^A on Γ specifies the decision predicates so as to insure that only those paths are traversed.

Note that Γ cannot be A_I related to τ since the

structure of τ is not rich enough to realize the machine Γ for all interpretations on Γ. Moreover τ is not actually a homomorphic image of any submachine in Γ since for this to be true the restricted input set of the submachine determined by $\rho(\Sigma_\tau) c \Sigma_\Gamma$ must be mapped back onto Σ_τ in a 1-1 manner. But in the example above $\{b_2, b_3\}$ cannot be mapped onto $\Sigma_\tau = \{a_1, a_2, a_3\}$ in this fashion. When ρ is 1-1, however, such a homomorphism exists and it simplifies the interpretations construction as shown in the next theorem.

Theorem 2.7

Given τ, C and Γ, if Γ is an H-type realization of τ, then

$\tau R_I \Gamma$ and will be denoted $\tau H_I \Gamma$.

Proof:

(1) Statements (1) and (2) in the proof of Theorem 2.7 are also valid here. The additional assumptions to be considered are that the map

$$\rho: \Sigma_\tau \longrightarrow \Sigma_\Gamma$$

is injective and that the control τ is a

reduced machine. Using these assumptions an interpretation I_j^H can be constructed from any I_j on τ as follows.

(2) From Theorem 1.1 and Definition 1.7, a 3 part homomorphism exists $\phi = (h_1, h_2, h_3)$ where

$$h_1: S^{\psi} \xrightarrow{\text{onto}} Q_{\tau}$$

$$h_2: \Sigma_{\Gamma}^{\psi} \xrightarrow{\text{onto}} \Sigma_{\tau}$$

$$h_3: Y_{\Gamma}^{\psi} \xrightarrow{\text{onto}} Y_{\tau}$$

and in terms of the existing map $\psi = (\alpha, \rho, \xi)$

$$h_1(s) = q \text{ for all } s \varepsilon \alpha(q) \subseteq S^{\psi}$$

$$h_2\left(\Sigma_{\Gamma}^{\psi}\right) = \rho^{-1}\left(\Sigma_{\Gamma}^{\psi}\right) = \Sigma_{\tau} \quad (\rho \text{ injective})$$

$$h_3\left(Y_{\Gamma}^{\psi}\right) = \xi\left(Y_{\Gamma}^{\psi}\right).$$

(3) Then the interpretation $I_j^H = \left\langle \eta_j^H, \gamma_j^H, \zeta_j^H, d_0 \right\rangle$ is constructed as follows:

(a) $\zeta_j^H: Y_{\Gamma}^{\psi} \longrightarrow F^*$

Then proceeding as in Theorem 2.6 but with

h_3 replacing ξ,

$$\zeta_j^H = h_3 \circ \zeta_j$$

where for any $y \varepsilon Y_\Gamma^\psi$, there exists a $q \varepsilon Q_\tau$, a $s \varepsilon S^\psi$, and a $y' \varepsilon Y_\tau$

such that

$$\lambda_\tau(q) = y' \varepsilon Y_\tau,\ \zeta_j\left[\lambda_\tau(q)\right] = \zeta_j\left[y'\right] = f \varepsilon F^*$$

and

$$h_1(s) = q;\quad s \varepsilon S^\psi$$

$$\lambda_\Gamma(s) = y;\ h_3\left[\lambda_\Gamma(s)\right] = h_3(y) = y'$$

Then

$$\zeta_j^H(y) = h_3 \circ \zeta_j(y) = \zeta_j\left[h_3(y)\right] = \zeta_j\left[h_3\left[\lambda_\Gamma(s)\right]\right]$$

$$= \zeta_j(y') = f$$

(b) $\gamma_j^H: S^\psi \longrightarrow p^K$

But here

$$\gamma_j^H = h_1 \circ \gamma_j \circ h_2^{-1}$$

where for all $s \varepsilon S^\psi$

$$h_1 \circ \gamma_j \circ h_2^{-1}(s) = h_2^{-1}\left[\gamma_j\left[h_1(s)\right]\right] = h_2^{-1}\left[\gamma_j(q)\right]$$

$$= h_2^{-1}\left[p^L\right]$$

and

$$h_2^{-1}\left[p^L\right] = \rho\left[p^L\right] = p^K$$

is (as before) that K-ary test equivalent to p^L. It encodes the response according to ρ but necessarily requires the same number of responses as p^L. If p^L were given by

$$p^L: D \longrightarrow \left\{\sigma_1\sigma_2\ldots\sigma_L\right\}$$

then

$$\rho\left[p^L\right] = h_2^{-1}\left[p^L\right]$$

$$= p^K : D \longrightarrow \left\{h_2^{-1}(\sigma_1), h_2^{-1}(\sigma_2)\ldots h_2^{-1}(\sigma_L)\right\}.$$

(c) $\eta_j^H: S^\psi \longrightarrow F^* \times p^K$

where

$$\eta_j^H(s) = (f, p^K)$$

iff

$$\gamma_j^H(s) = p^K$$

$$\zeta_j^H\left[\lambda_\Gamma(s)\right] = f.$$

$\triangleright\triangleleft$

This construction insures that the submachine $\Gamma|_\psi$ under the interpretation I_j^H, behaves as $I_j[\tau]$ for all $d \epsilon D$ and $j = 1,m$. The difference between these two realizations is brought out by the next Theorem.

Theorem 2.8

If

$$\tau H_I \Gamma \text{ then } \tau A_I \Gamma$$

but

$$\tau A_I \Gamma \not\Rightarrow \tau H_I \Gamma.$$

Proof:

This theorem is trivial based on the conditions necessary for A and H-type realizations.

The importance of the direction of the implication in Theorem 2.8 lies in the difference in complexity in interpretation construction. If $\tau A_I \Gamma$ then one must transform all decision predicates according to the rule

$$\rho(p^L) = p^K$$

In the A-type realization it may be necessary to encode the environmental responses to several distinct tests as one response in the predicate p^K. This is generally difficult to do efficiently. If $\tau H_I \Gamma$ then since ρ is 1-1 only the trivial encoding of picking some distinct symbol in Σ_Γ to correspond to every test made by p_i^L is required. For example, suppose the decision predicate p^K was implemented as an arithmetic expression in a computed GO TO statement,

```
GO TO (5,7,10,20) PK(Q)
```

Here, PK(Q) is a call to an external function which returns a value 1, 2, 3 or 4 thereby branching to "states" 5, 7, 10 or 20 respectively. If the decision predicate p^L appeared in some other computed GO TO statement as

```
GO TO (23,24,25) PL(Q)
```

then

$$\Sigma_\tau = \{1,2,3\}$$

$$\Sigma_\Gamma = \{1,2,3,4\}$$

And if

$$\rho: \Sigma_\tau \xrightarrow{1\text{-}1} \Sigma_\Gamma \quad \text{(injective)}$$

say, for example,

$\rho(1) = 2$

$\rho(2) = 4$

$\rho(3) = 1$

then the appropriate decision predicate to be used in a call for PK(Q) can be exactly the same as PL(Q) and the 1-1 encoding of the response can be effected by re-ordering the labels in the four branch GO TO statement above.

```
GO TO (7,20,5,10)
```

But if ρ is not injective then it probably will be necessary to change the external function to realize the more complicated encoding required by ρ. As might be expected the type of realization determines the relationship between valuation sequences.

Theorem 2.9

Given τ, Γ and C and $\tau A_I \Gamma$ or $\tau H_I \Gamma$, then for any l_j on τ, there exists a map ρ^* such that

$$\rho^*: \tilde{S}_D\left(\overline{I_j[\tau]}\right) \rightarrow \tilde{S}_D\left(\overline{I_j[\Gamma]}\right)$$

where ρ^* is defined by

$$\rho^*\left[S_d\left(\overline{I_j[\tau]}\right)\right] = \rho^*\left[\left(\sigma_1,\sigma_2,.,\sigma_n\right)\right] = \left\{\rho(\sigma_1),.,\rho(\sigma_n)\right\}$$

and

$$S_d\left(\overline{I_j[\tau]}\right),\ \rho^*\left[S_d\left(\overline{I_j[\tau]}\right)\right]$$

correspond to the same program value,

and for $\tau H_I \Gamma$, ρ^* is injective.

Proof:

This is clear from definitions of H- and A-type realizations. So one can always identify any sequence of environmental response encodings for a particular program with some sequence for another program whose control is an H- or A-type realization. ▷◁

In passing it is noted that $\tau R_I \Gamma$ for Γ a C-type realization of τ, and an interpretation construction could be given. But since $\rho:\Sigma_\tau \longrightarrow \Sigma_\Gamma^*$, the nature of the appropriate decision predicates to be mapped onto states in Γ by I^C is much more complicated than in either of the other two realizations.

The essential point in the development so far can now be made: the complexity of construction of program

equivalence preserving interpretations for a control common to some class of controls is determined solely by the structural relationships existing between the common control and schema in the class. As H-type realizations are by far the easiest to implement, most of what follows relies heavily on this type. Even with this restriction the applications in programming are quite diverse and interesting.

CHAPTER III

SYNTHESIZING PROGRAMS WITH COMMON CONTROLS

This chapter is concerned with three different ways of realizing H-type common controls and their applications in program design. Section 3.1 deals with a realization such that, when the stated conditions can be detected, a common control with interpretations can easily be constructed. Unfortunately the detection problem, though solvable with a simple algorithm, usually requires considerable effort. The sections 3.2 and 3.4 examine H-type common controls which can be constructed directly.

It will be assumed that some class of controls is given as

$$\left\{\tau_i\right\} = \left\{ \left\langle Q_{\tau_i}, \Sigma_{\tau_i}, Y_{\tau_i}, \delta_{\tau_i}, \lambda_{\tau_i}, q_{0_i} \right\rangle \right\} \; ; \; i = 1,m$$

along with the control

$$\Gamma = \left\langle S_\Gamma, \Sigma_\Gamma, Y_\Gamma, \delta_\Gamma, \lambda_\Gamma, s_0 \right\rangle$$

which is to be the common control for the class $\{\tau_i\}$,

$$\tau_i R_I \Gamma \; ; \; i = 1,m.$$

The nature of the R_I relation will be specified for various structural relationships between Γ and $\{\tau_i\}$. The following definition will be useful:

Definition 3.1

The semi-control $\hat{\tau}$, is defined to be the output-free state machine of τ:

if
$$\tau = \langle Q, \Sigma, Y, \delta, \lambda, q \rangle$$

then
$$\hat{\tau} = \langle Q, \Sigma, \delta, q \rangle$$

In this chapter the output function of Γ, λ_Γ, and the output set, Y_Γ, are distinct for each state in Γ. They merely serve as unspecified variables in 1-1 correspondence with $s_i \varepsilon S_\Gamma$ to make the composition maps in the previous chapter well defined. The theorems in this chapter involve $\hat{\Gamma}$ and there is no loss in generality.

3.1 SP Partition Realizations

An easy case for which a common control can be determined is when the $\hat{\tau}_i$ correspond to SP partitions on $\hat{\Gamma}$. If the SP lattice of $\hat{\Gamma}$ is denoted as

$$L_{\hat{\Gamma}} = \left\langle \left\{ \pi_j^{\hat{\Gamma}} \right\}, +, \cdot, 0, I, \leq \right\rangle$$

where

$$\pi_j^{\hat{\Gamma}} = \left\langle \left\{ B_k^j \right\}, \delta_{\hat{\Gamma}}, \Sigma_{\hat{\Gamma}} \right\rangle$$

and

$$B_k^j \subseteq 2^{S_{\hat{\Gamma}}} \text{ for all } k$$

$$\delta_{\hat{\Gamma}}(B_k^j,\sigma) = B_\ell^j \quad \text{iff} \quad \text{for all } s\in B_k^j,$$

$$\delta_{\hat{\Gamma}}(s,\sigma)\in B_\ell^j, \text{ for all } \sigma\in\Sigma_\Gamma.$$

Then the following is true.

Theorem 3.1

Assume $|\Sigma_{\hat{\tau}_i}| = |\Sigma_{\hat{\Gamma}}|$, i = 1,m. If for any $\hat{\tau}_i$, there is some $\pi_j^{\hat{\Gamma}}\in L_{\hat{\Gamma}}$ such that $\hat{\tau}_i \tilde{=} \rfloor\pi_j^{\hat{\Gamma}}$, then $\hat{\Gamma}$ is a common control for the class, $\{\hat{\tau}_i\}$, and $\hat{\tau}_i H_I \hat{\Gamma}$ for all i = 1,m.

Proof:

(1) The isomorphic relation $\hat{\tau}_i \tilde{=} \rfloor\pi_j^{\hat{\Gamma}}$ is interpreted to mean $\hat{\tau}_i$ is isomorphic to some submachine of $\hat{\Gamma}$. This submachine is a homomorphic image of $\hat{\Gamma}$ and the state mapping of this (three part) homomorphism defines the SP partition, $\pi_j^{\hat{\Gamma}}$.

(2) The state homomorphism identifying the blocks, $\{B_k^j\}$, of the partition, $\pi_j^{\hat{\Gamma}}$, is denoted (as before) by the $h_1^i(\cdot)$

mapping, under which the transition function, $\delta_\Gamma(\cdot)$, is preserved.

But

$$\hat{\tau}_i \overset{\sim}{=} \lfloor_{\pi_j^{\hat{\Gamma}}} \Rightarrow \{B_k^j\} \overset{\sim}{=} Q_{\tau_i}$$

hence

$$h_1^i: S \longrightarrow \{B_k^j\}.$$

(3) Also a 1-1 <u>onto</u> map exists because of the machine isomorphism

$$h_2^i: \Sigma_\Gamma \longrightarrow \Sigma_\tau$$

(4) The output map

$$h_3^i: Y_\Gamma \longrightarrow Y_{\tau_i}$$

is defined in the following manner:

for all $y \in Y_\Gamma$ such that $\lambda_\Gamma(s) = y$,

and

$$h_1^i(s) = q \in Q_{\tau_i},$$

$$\lambda_{\tau_i}(q) = y'$$

then

$$h_3^i(y) = y'.$$

(5) This three part homomorphism $\phi^i = \left(h_1^i, h_2^i, h_3^i\right)$ determines τ_i as a homomorphic image of the complete machine Γ (not a submachine) and from Theorem 2.7, $\tau_i \amalg_I \Gamma$. Since the map can be con-

structed for any τ_i under the conditions of this theorem, Γ is an H-type common control for the class $\{\tau_i\}$. $\triangleright\triangleleft$

It turns out that this particular type of structural relationship makes the job of constructing common control interpretations easier. The assumption $|\Sigma_{\tau_i}| = |\Sigma_\Gamma|$ implies that <u>exactly</u> the same decision predicates can be used if the responses are encoded in the 1-1 fashion via h_2^i. If $\gamma(q) = p^L$ for some I on τ_i, then

$$\gamma^H(s) = h_1^i \circ \gamma \circ h_2^{i-1}(s) = h_2^{i-1}\left[\gamma\left(h_1^i(s)\right)\right]$$

$$= h_2^{i-1}\left[\gamma(q)\right] = h_2^{i-1}\left[p^L\right]$$

where

$$p^L: D \longrightarrow \left\{\sigma_1, \sigma_2, .., \sigma_L\right\} = \Sigma_{\tau_i}$$

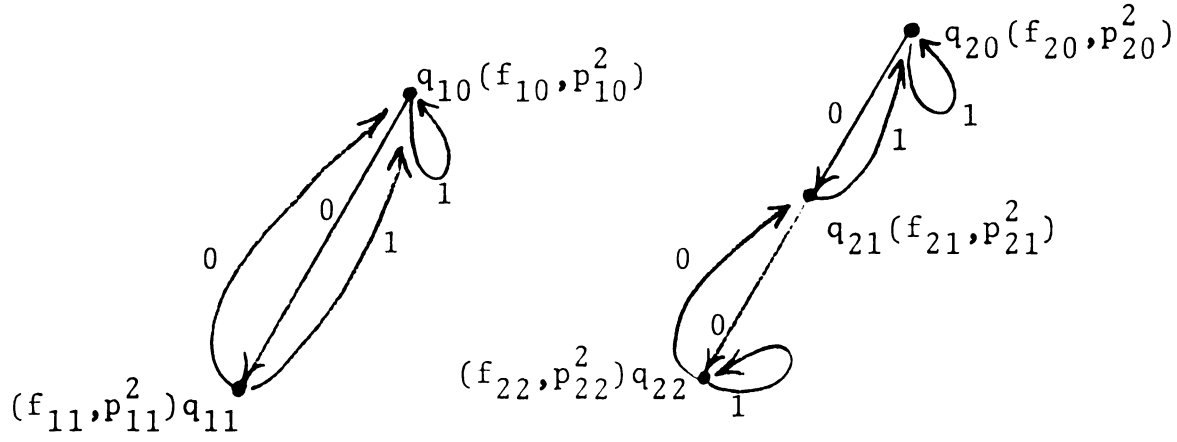
$$h_2^{i-1}\left[p^L\right]: D \longrightarrow \left\{h_2^{i-1}(\sigma_1), .., h_2^{i-1}(\sigma_L)\right\} = \Sigma_\Gamma$$

Also notice that in this case the submachine of Γ involved in the H-type realization is the machine itself. Thus it is necessary to seek out isomorphisms, if any, between elements of the class $\{\tau_i\}$ and partitions in the SP lattice

of Γ. A nice algorithm for generating the SP lattice of any FSM exists, (Hartmanis-Stearns [1966]), and is much more convenient than hunting for submachines in Γ. An example will suffice to show the ease with which proper interpretations can be constructed for the above situations.

Example 3.1

Consider the following interpreted controls.



δ_{τ_1}:

	0	1	η_1
q_{10}	q_{11}	q_{10}	(f_{10},p_{10}^2)
q_{11}	q_{10}	q_{11}	(f_{11},p_{11}^2)

(a)

δ_{τ_2}:

	0	1	η_2
q_{20}	q_{21}	q_{20}	(f_{20},p_{20}^2)
q_{21}	q_{22}	q_{20}	(f_{21},p_{21}^2)
q_{22}	q_{21}	q_{22}	(f_{22},p_{22}^2)

(b)

Figure 3.1

Interpreted controls τ_1 and τ_2.

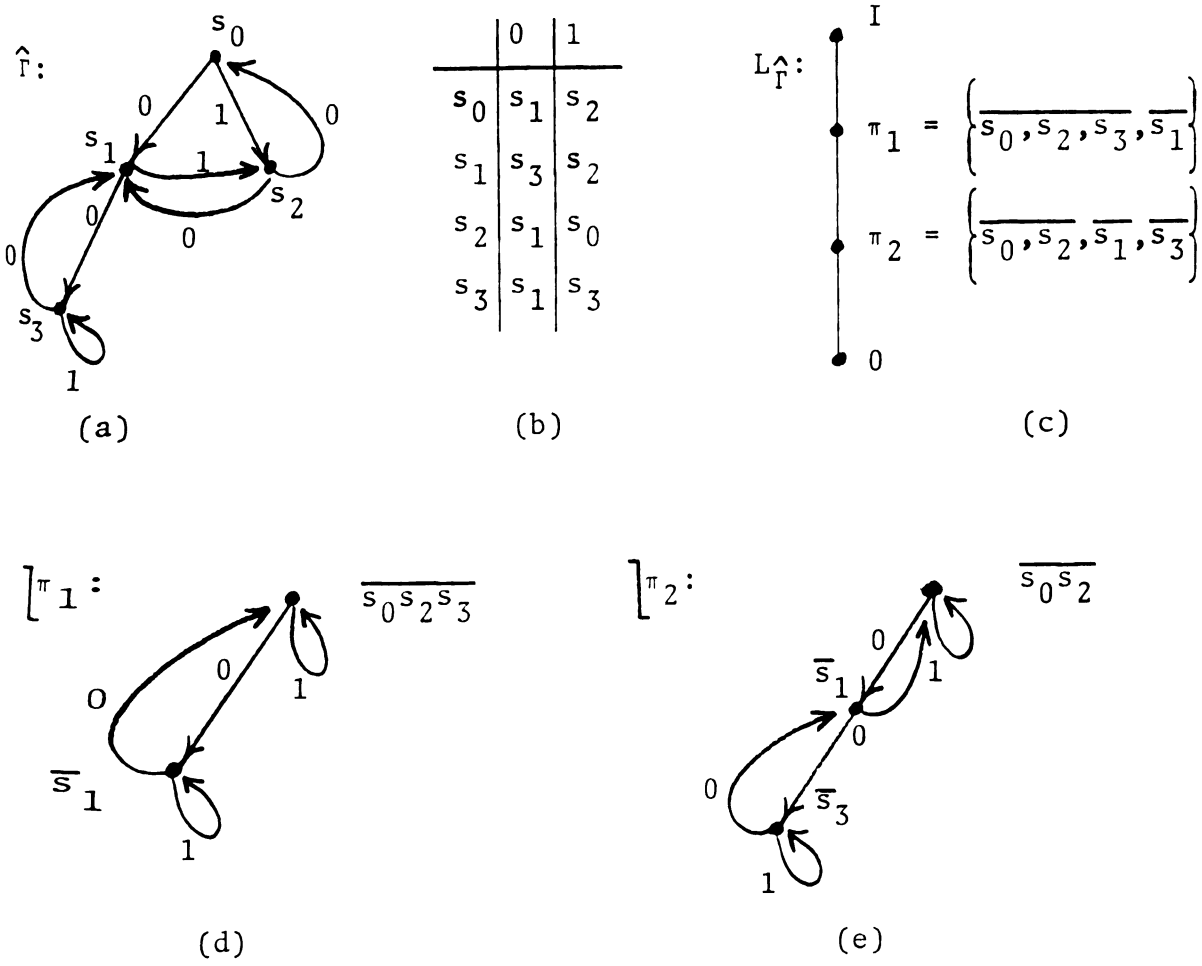


Figure 3.2

$\hat{\Gamma}$ and the SP lattice $L_{\hat{\Gamma}}$

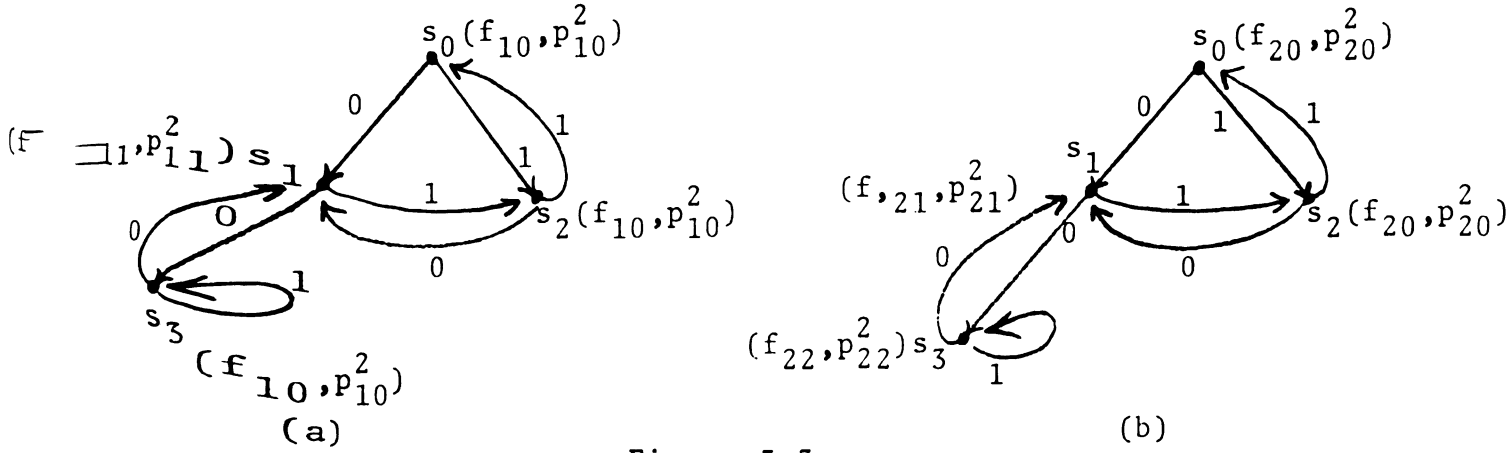


Figure 3.3

Interpretations on Γ to simulate $\P_1$ and $\P_2$

The appropriate computations and decision predicates for $\P_1$ and $\P_2$ defined by two interpretations on τ_1 and τ_2 are shown in Figure 3.1 (a) and (b) respectively. Examination of the two semi-controls corresponding to the SP partitions in $L_{\hat{\Gamma}}$ shows

$$\rceil_{\pi_1} \cong \tau_1$$

and

$$\rceil_{\pi_2} \cong \tau_2$$

with the (identity) maps

$$h_2^1: \Sigma_{\tau_1} \xrightarrow{\text{onto}} \Sigma_\Gamma$$

$$h_2^2: \Sigma_{\tau_2} \xrightarrow{\text{onto}} \Sigma_\Gamma .$$

The appropriate interpretations on Γ to simulate $\P_1$ and $\P_2$ are determined quite simply from Theorem 3.1 and appear as Figure 3.3 (a) and (b) respectively. The identity maps h_2^1, h_2^2 required no encoding of responses in either case, thus simplifying the implementation in some programming language. Also, the theorem does not require that the state homomorphism, h_1^i, map the starting state $s_0 \varepsilon S_\Gamma$ onto the starting state $q_{0_{\tau_i}}$. This complicates the situation because implementing the common control requires a reset input or

an entry point to some state $s_i \varepsilon S_\Gamma$ such that $h_1^i(s_i) = q_{0_{\tau_i}}$.

In the example, Γ is "synchronized" to both τ_1 and τ_2 and no such reset is necessary. So even though this type of H-realization is conceptually simple, the programmer is forced to accept responsibility of including entry points to $I_j^H[\Gamma]$ which synchronize with each $I_j[\tau_i]$. The next type of H-realization does not involve this synchronization problem but only at the expense of an increase in the number of states in the common control Γ.

3.2 Semigroup Realizations

Suppose a programmer devises a procedure with as many separate entry points as states (pairs of computations and decision predicates). If the procedure has a cyclic control τ, then all subprocedures identified by the entry points correspond to the class of controls

$$\left\{\tau_i\right\} = \left\{ \left\langle Q_\tau, \Sigma_\tau, Y_\tau, \delta_\tau, q_i \right\rangle \right\}, i = 1,m \; ;$$

$$\text{for } Q_\tau = \left\{q_i\right\} \; ; \; i = 1,m$$

Immediately then

Theorem 3.2

If $\hat{\Gamma}$ is the semi-control representing the semigroup accumulator of τ, then Γ is a common control for $\left\{\tau_i\right\}$

and

$$\tau_i H_I \Gamma \quad \text{for all} \quad i = 1,m.$$

Proof:

(1) From Definition 1.11, the states in $\hat{\Gamma}$ correspond to all maps on Q defined by input tapes in Σ_τ^*.

$$s_i \longleftrightarrow x \epsilon \Sigma_\tau^* \quad \text{such that} \quad \delta_\tau(Q_\tau, x) \subseteq Q_\tau$$

(2) So

$$\delta_\Gamma(s_i, \sigma) = s_j \; ; \; \sigma \epsilon \Sigma_\Gamma$$

iff

$$s_i \longleftrightarrow x \epsilon \Sigma_\tau^*$$

$$s_j \longleftrightarrow y \epsilon \Sigma_\tau^*$$

and

$$x \circ \sigma(Q_\tau) = \sigma[x(Q_\tau)] = y(Q_\tau).$$

(3) For any τ_i with initial state q_{0_i}, the three part homomorphism $\phi^i = \left(h_1^i, h_2^i, h_3^i\right)$ can be constructed as

(a) $h_1^i : S_\Gamma \longrightarrow Q_\tau$

$$h_1^i(s_0) = q_{0_i}$$

$$h_1^i(s_k) = \delta_\tau(q_{0_k}, z) \text{ where } s_k \longleftrightarrow z \epsilon \Sigma_\tau^*$$

(b) $h_2^i = I$: $\Sigma_\Gamma \longrightarrow \Sigma_\tau$

the identity map, since the semigroup accumulator is restricted to the input set Σ_τ.

(c) h_3^i: $Y_\Gamma \longrightarrow Y_\tau$

that is,

$$h_3^i(y) = y' \; ; \; \lambda_\Gamma(s) = y \varepsilon Y_\Gamma \text{ and } h_1^i(s) = q'$$

$$\text{and } \lambda_{\tau_i}(q') = y'.$$

(4) With the existence of a three part homomorphism demonstrated for any τ_i and the submachine of Γ identified as the whole machine, the construction of I^H on Γ for any I on some τ_i can proceed as in Theorem 2.7. Hence

$$\tau_i H_I \Gamma \quad ; \quad i = 1,m$$

and since this holds for any τ_i, Γ is an H-type common control for the class $\left\{\begin{matrix} i \\ \tau_i \end{matrix}\right\}$.

$\triangleright\triangleleft$

The utility of this realization is that it does not require a priori knowledge of its existence nor does it necessitate extensive search procedures to detect it. One simply generates the semigroup of the control τ. Then for

any start-state q_i, use the method above to get ϕ^i and an interpretation I^H on Γ for some I on τ_i. The decision predicates in I^H are exactly the same as in I and no encoding of responses is required because each h_2^i is an identity map. No synchronization is required to get Γ started correctly for any τ_i since $h_1^i(s_0) = q_{0_i}$ for all $i=1,m$. The price extracted for this is a tremendous increase in the number of states in Γ. The maximum size semigroup for any n-state machine is n^n. Even when, for reasons of program design, analysis and organization, it is desirable to use this type of realization it is ill-advised to use up to n^n states to simulate n, n-state machines. Chapter four will mention some aspects in the problem of reducing the semigroup size of τ using a set of transformations on τ.

3.3 Example of Semigroup Control Realization

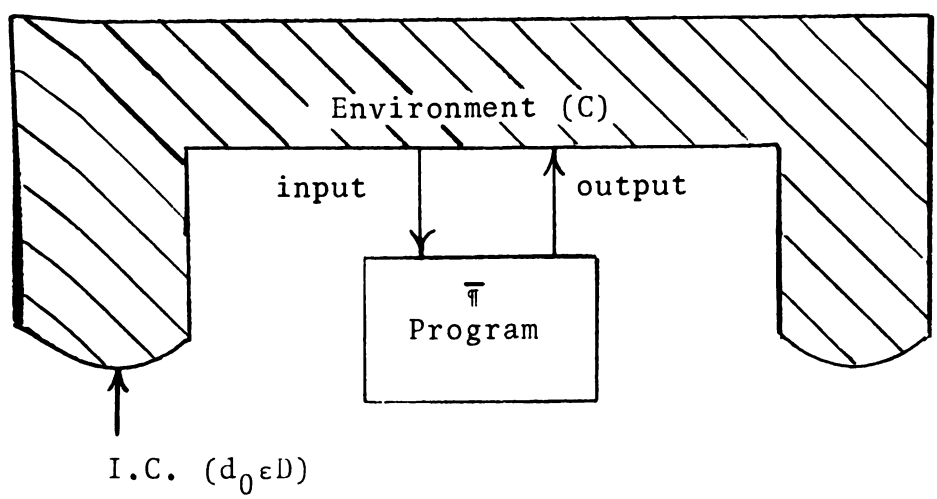
This section presents a complete example of developing a set of programs from an initial specification to the design of a common control for these programs and an appropriate interpretation on this control to simulate each. The method and type of realization is shown in Section 3.2.

This example will illustrate how a semigroup realization control can be constructed for a set of programs performing a typical housekeeping activity utilized by some file manage-

ment system. The activity shall be performed by a subprogram whose operational description is the following:

> Read and copy logical records comprising a set of files on a specified logical unit (disc, drum, tape, etc.) into a core buffer of specified length. Output the buffer to a second specified logical unit when (1) buffer is filled or (2) an end of file is received. Whenever an end file is indicated on input, after the buffer is output an end file is written on the second unit. Two end files read in succession indicate termination.

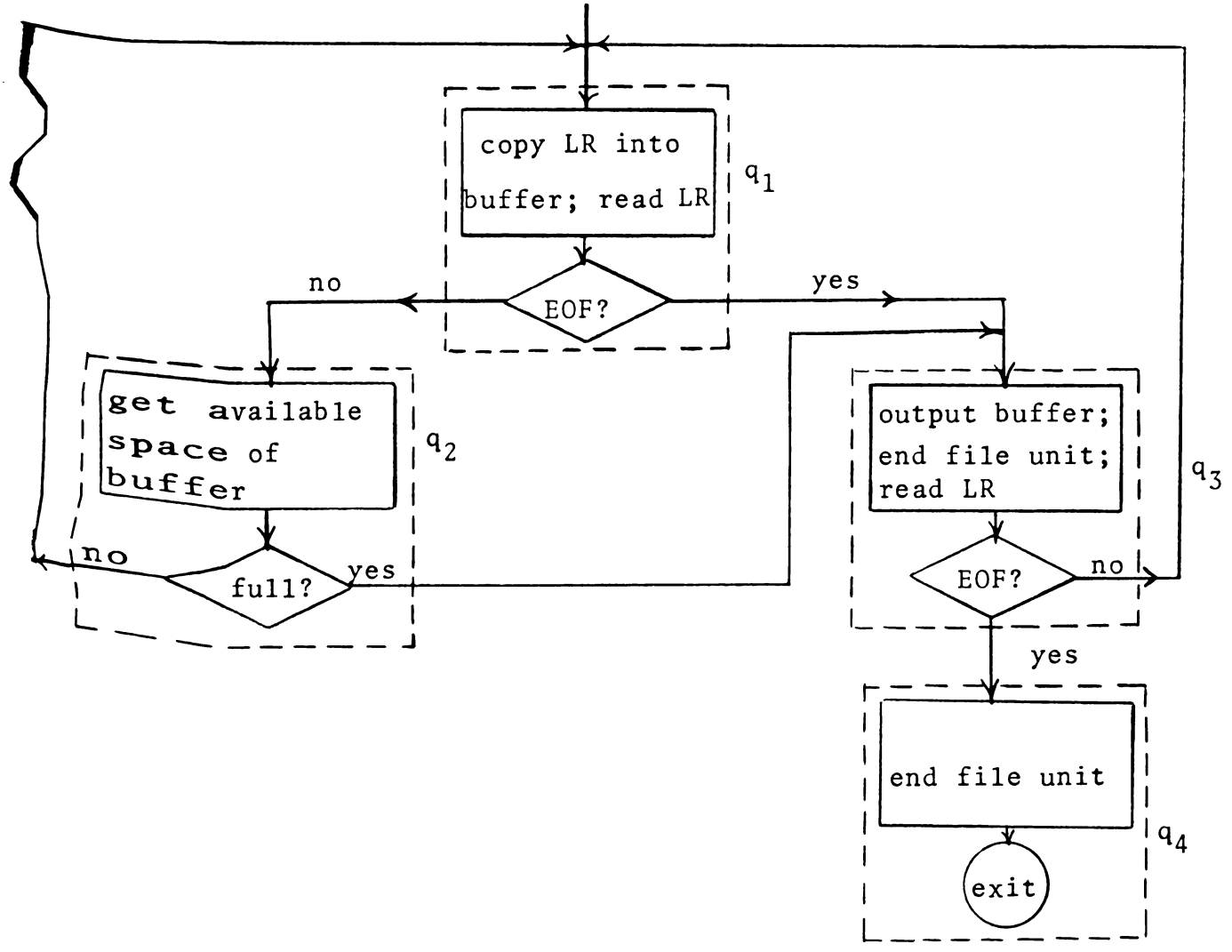
The subprogram will essentially direct the relevant operations which are considered part of its environment. A schematic of its relation to this environment is shown below.



Essentially, the environment consists of hardware and software functions belonging to the total computer representa-

tion, C. Thus the program output is a call to a subset of these functions--usually implemented as subprograms in the machine object code or microprograms--which activates these functions and to which the environment responds by sending an input to $\overline{\P}$ describing some portion of the effect of these calls. The initial configurations $d_0 \varepsilon D$ may or may *not* be set by the calling program of $\overline{\P}$ but are essential *in* ascertaining the program value, $V_d(\overline{\P})$.

One flow chart for $\overline{\P}$ might be as follows:



$$\overline{\P} = \overline{I[\tau]} \; ; \quad I = \langle \eta, \gamma, \zeta, d \rangle$$

where η associates with q_i the pair

$$\eta(q_i) = (f_i, p_i^2) \; ; \; i = 1,4$$

and

f_1: copy LR into buffer; read LR

f_2: get available space of buffer

f_3: output buffer; end file unit; read LR

f_4: end file unit

p_1^2: end file check

p_2^2: available space = zero

p_3^2: = p_1^2

p_4^2: null decision - exit

A state diagram of the control flow is given by Figure 3.4.

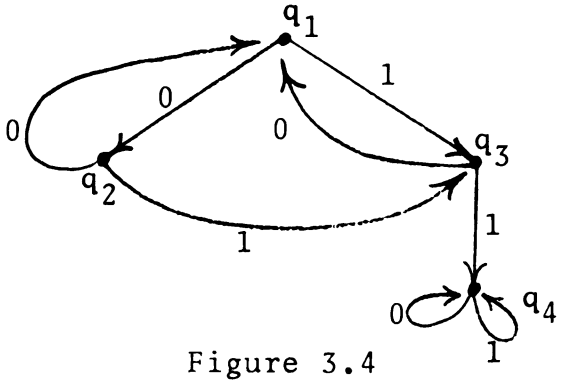


Figure 3.4

Flow Chart of $\overline{\P}$

The final state, q_4, is an exit and the inputs are superfluous but included for completeness. Conceivably a calling program for $\overline{\P}$ might enter $\overline{\P}$ at q_1, q_2, q_3 or q_4 depending upon various editing operations performed by the calling program. Multiple entry points conveniently meet this requirement. But to view a program as a "black box" with input-output lines over which control codes pass, it will be useful to consider the control entry as an initial condition. Moreover, it will be required that the initial conditions map onto an unspecified control flow diagram a sequence from the set $\left\{\left(f_i, p_i^2\right)\right\}$,

$i = 1,4$. Thus $\overline{\P}$ will be characterized by (1) its control flow diagram and (2) the initial conditions which link particular computations and decision functions with the states of the control. The interpreted control model of $\overline{\P}$ is

$$\overline{\P} = \overline{I(\tau)} \quad \text{(initial configuration-free program)}$$

where

$$\tau = \left\langle Q_\tau, \Sigma_\tau, Y_\tau, \delta_\tau, \lambda_\tau, q_1 \right\rangle$$

$$Q_\tau = \left\{q_i\right\};\ i = 1,4$$

$$\Sigma_\tau = \left\{0,1\right\} \qquad \text{(yes = 1; no = 0)}$$

$\lambda_\tau, \delta_\tau$ are given by the table

δ_τ :

	0	1	λ_τ
q_1	q_2	q_3	y_1
q_2	q_1	q_3	y_2
q_3	q_1	q_4	y_3
q_4	q_4	q_4	y_4

and

$$I = \langle \eta, \gamma, \zeta, d_0 \rangle$$

$$\eta: Q_\tau \longrightarrow F^* \times P^2 \quad : \eta(q_i) = (f_i, p_i^2) \quad i = 1,4$$

$$\gamma: Q_\tau \longrightarrow P^2 \quad : \gamma(q_i) = p_i^2 \quad i = 1,4$$

$$\zeta: Y_\tau \longrightarrow F^* \quad : \zeta(y_i) = f_i \quad i = 1,4$$

$$d_0 \in D$$

that is

d_0: some initial configuration of the environment insuring all variables, subprograms etc. are defined and available for the execution of ¶.

Examine now the four machines in Figure 3.5 obtained by using each state as a start state.

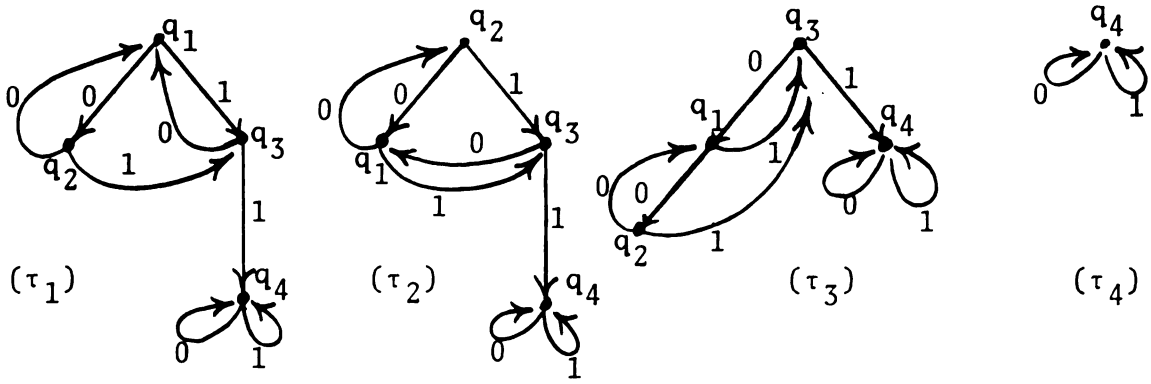


Figure 3.5

Start State Configurations

Clearly, the behavior (i.e. input-output sequences) of these machines is distinct for each configuration but certainly a common structure exists. Paraphrasing Saul Gorn*; "Structure (and behavior) is in the eyes of the beholder." So these machines can be distinguished as separate structural entities when it is realized that not one of the above structures can be programmed with a single entry point to effect the behavior of any of the other three. The semigroup machine of τ can, however, as will be shown. Define the control

$$\Gamma = \left\langle S_\Gamma, \Sigma_\Gamma, Y_\Gamma, \delta_\Gamma, \lambda_\Gamma, s_0 \right\rangle$$

*See the discussion between Gorn and Brzozowski in the article "Synthesis of Sequential Machines," in Systems and Computer Science, (p. 16), Hart, J.F., Takasu, S. ed., University of Toronto Press, 1967.

where

$$S = \{s_i\} \; ; \; i = 1,10$$

$$\Sigma_\Gamma = \{0,1\} = \Sigma_\tau$$

$$Y_\Gamma = \{Z_i\} \; ; \; i=1,10$$

where λ_Γ and δ_Γ are given by

δ_Γ:	0	1	λ_Γ
s_1	s_2	s_3	Z_1
s_2	s_4	s_5	Z_2
s_3	s_6	s_7	Z_3
s_4	s_2	s_5	Z_4
s_5	s_8	s_7	Z_5
s_6	s_9	s_3	Z_6
s_7	s_7	s_7	Z_7
s_8	s_{10}	s_5	Z_8
s_9	s_6	s_3	Z_9
s_{10}	s_8	s_5	Z_{10}

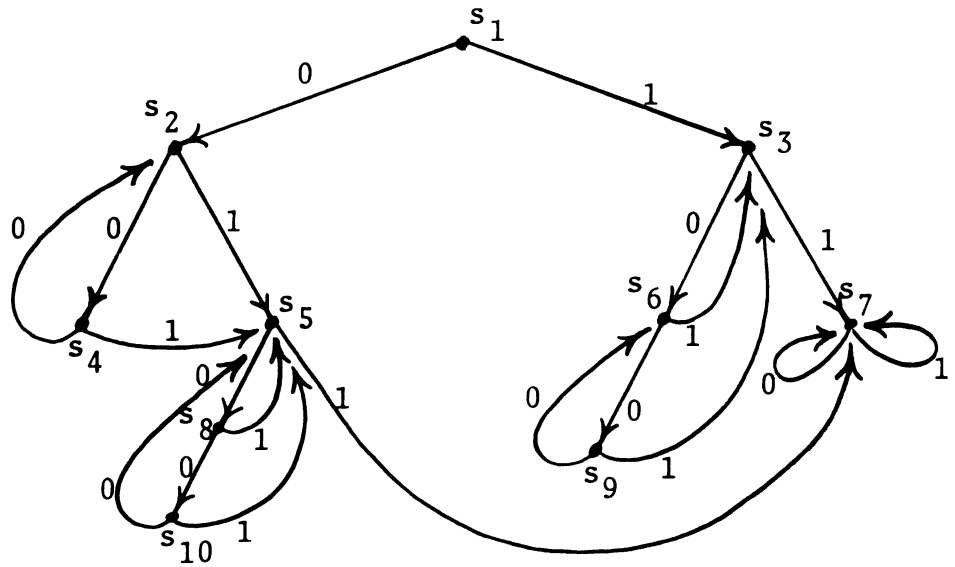


Figure 3.6

State Diagram of Semigroup Accumulator Γ

It can be verified that Γ is the semigroup machine for τ. Moreover, it is isomorphic to $(\times_{i=1}^{4} \tau_i)^c$ by the correspondence:

$s_1 \longleftrightarrow (q_1,q_2,q_3,q_4)$ and the corresponding input tape is null,

$s_2 \longleftrightarrow (q_2,q_1,q_1,q_4)$ and the corresponding input tape is 0,

$s_3 \longleftrightarrow (q_3,q_3,q_4,q_4)$ and the corresponding input tape is 1,

$s_4 \longleftrightarrow (q_1,q_2,q_2,q_4)$ and the corresponding input tape is 00,

$s_5 \longleftrightarrow (q_3,q_3,q_3,q_4)$ and the corresponding input tape is 01,

$s_6 \longleftrightarrow (q_1,q_1,q_4,q_4)$ and the corresponding input tape is 10,

$s_7 \longleftrightarrow (q_4,q_4,q_4,q_4)$ and the corresponding input tape is 11,

$s_8 \longleftrightarrow (q_1,q_1,q_1,q_1)$ and the corresponding input tape is 010,

$s_9 \longleftrightarrow (q_2,q_2,q_4,q_4)$ and the corresponding input tape is 100,

$s_{10} \longleftrightarrow (q_2,q_2,q_2,q_4)$ and the corresponding input tape is 0100.

The set of mappings $\{\phi^i\}$ verifying the Γ realization of $\{\tau_i\}$ is given by $\phi^i = \left(h_1^i, h_2^i, h_3^i\right)$; h_2^i is the identity map and h_1^i, h_3^i are defined as

$h_1^i(s_j)$ \ s_j	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
$h_1^1(s_j)$	q_1	q_2	q_3	q_1	q_3	q_1	q_4	q_1	q_2	q_2
$h_1^2(s_j)$	q_2	q_1	q_3	q_2	q_3	q_1	q_4	q_1	q_4	q_2
$h_1^3(s_j)$	q_3	q_1	q_4	q_2	q_3	q_4	q_4	q_1	q_4	q_2
$h_1^4(s_j)$	q_4	q_4	q_4	q_4	q_4	q_4	q_4	q_4	q_4	q_4

Note that since

$$\lambda_\Gamma(s_j) = Z_j;\ \lambda_\tau(q_k) = y_k$$

therefore

$$h_3^i(Z_j) = y_k \quad \text{iff} \quad h_1^i(s_j) = q_k.$$

Since $I = \langle \eta, \gamma, \zeta, d_0 \rangle$ is the common interpretation for all controls τ_i, and $\eta(q_i) = (f_i, p_i^2)$, $i = 1,4$ where f_i and p_i^2 are defined appropriately, the interpretations $I_i^H = \langle \eta_i^H, \gamma_i^H, \zeta_i^H, d_0 \rangle$ are defined by

(1) $\zeta_i^H(Z_j) = h_3^i \circ \zeta_i(Z_j) = \zeta_i\left[h_3^i(Z_j)\right]$

and can be read directly from the h_3^i, h_1^i above and the map $\zeta_i(y_k) = f_k$.

(2) $\gamma_i^H(s_j) = h_1^i \circ \gamma_i(s_j) = \gamma_i\left[h_1^i(s_j)\right]$

(also directly from h_1^i table).

(3) Combining 1 and 2 the table for η_i^H is

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
$\eta_1^H(s_j)$	$f_1p_1^2$	$f_2p_2^2$	$f_3p_3^2$	$f_3p_3^2$	$f_3p_3^2$	$f_1p_1^2$	$f_4p_4^2$	$f_1p_1^2$	$f_2p_2^2$	$f_2p_2^2$
$\eta_2^H(s_j)$	$f_2p_4^2$	$f_1p_1^2$	$f_3p_4^2$	$f_2p_2^2$	$f_3p_3^2$	$f_1p_1^2$	$f_4p_4^2$	$f_1p_1^2$	$f_4p_4^2$	$f_2p_2^2$
$\eta_3^H(s_j)$	$f_3p_4^2$	$f_1p_1^2$	$f_4p_4^2$	$f_2p_2^2$	$f_3p_3^2$	$f_4p_1^2$	$f_4p_4^2$	$f_1p_1^2$	$f_4p_4^2$	$f_2p_2^2$
$\eta_3^H(s_j)$	$f_4p_4^2$	$f_4p_4^2$	$f_4p_4^2$	$f_4p_4^2$	$f_4p_4^2$	$f_4p_4^2$	$f_4p_4^2$	$f_4p_4^2$	$f_4p_4^2$	$f_4p_4^2$

At this point all that remains is to implement Γ as a subprogram in FORTRAN, building in the control flow according to the transition function, δ_Γ, but parameterizing the computations and decisions made at each state. The control flow chart together with its parameterized subroutine implementation follow.

Each state Si is considered as a combination of a calculation and decision in Figure 3.7.

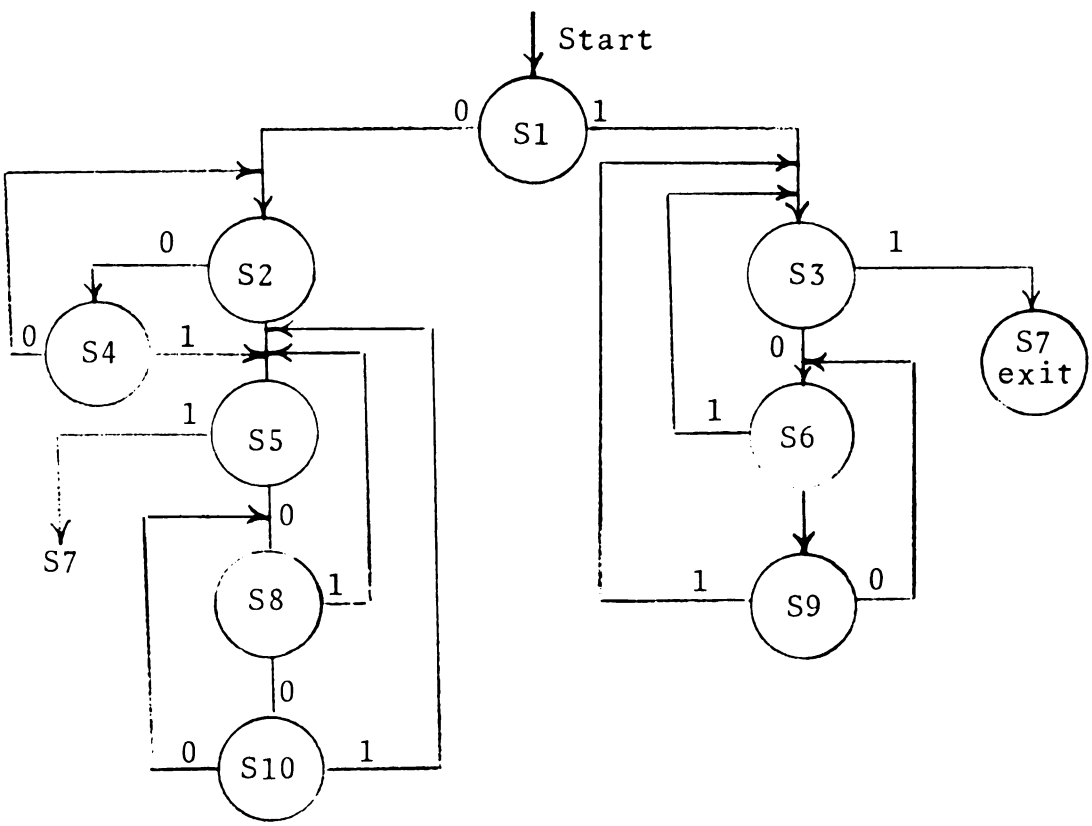


Figure 3.7

Γ Control Flow Chart

```
    SUBROUTINE GAMMA (S1,S2,S3,S4,S5,S6,S7,S8,S9,S10)
 10 IF (S1 (DUMMY))    500, 20, 30
 20 IF (S2 (DUMMY))    500, 40, 50
 30 IF (S3 (DUMMY))    500, 60, 70
 40 IF (S4 (DUMMY))    500, 20, 50
 50 IF (S5 (DUMMY))    500, 80, 70
 60 IF (S6 (DUMMY))    500, 90, 30
 70 IF (S7 (DUMMY))    500, 70, 70
 80 IF (S8 (DUMMY))    500, 100, 50
 90 IF (S9 (DUMMY))    500, 60, 30
100 IF (S10 (DUMMY))   500, 80, 50
500 RETURN
    END
```

(Note: the argument "DUMMY" and parentheses are required in FORTRAN to distinguish external function subprograms.)

Figure 3.7
(Cont.)
Parameterized Subroutine Implementation

A feasible set of functions and predicates could be incorporated into the following FORTRAN function subprograms:

```
      FUNCTION Q1 (DUMMY)
      COMMON A, B, N, MX, INU, IOU
      CALL COPY (A, B(N))
      CALL READU (INU, A)
      IF (EOF, INU) 10, 20
10    Q1 = 1.
      RETURN
20    Q1 = 0.
      RETURN
      END

      FUNCTION Q2 (DUMMY)
      COMMON A, B, N, MX, INU, IOU
      NDIF = MX - N
      IF (NDIF.LE.0) 10. 20
10    Q2 = 1.
      RETURN
20    Q2 = 0.
      RETURN
      END
```

```
      FUNCTION Q3 (DUMMY)
      COMMON A, B, N, MX, INU, IOU
      CALL OUTBUF (IOU, B, N)
      CALL ENDFILE (IOU)
      CALL READU (INU, A)
      IF (EOF, INU) 10, 20
10    Q3 = 1.
      RETURN
20    Q3 = 0.
      RETURN
      END

      FUNCTION Q4 (DUMMY)
      COMMON A, B, N, MX, INU, IOU
      CALL ENDFILE (IOU)
C     NULL DECISION - FORCES RETURN
      Q4 = - 1.
      RETURN
      END
```

Note:

A: temporary storage for logical record

B: output buffer

N: running index of used cells in B

MX: maximum length of B (= integral number of logical record lengths)

INU: number of the input logical unit

IOU: number of the output logical unit

In each function subprogram, Qi, the computation, f_i, and the predicate, p_i^2, is easily discerned. The interpretation mapping η (of I) is used directly to indicate the proper computations and decisions to be included. The "states" of GAMMA are defined when variables declared as external

functions in the calling program are used in the GAMMA call parameter list. Exactly how to line up the function programs Q1 to Q4 in the parameter list is revealed explicitly by the mappings $\left\{\eta_i^H\right\}_{i=1,4}$. The calling program would appear as:

```
      PROGRAM CALGAM
       .
       .
       .
      EXTERNAL Q1, Q2, Q3, Q4
       .
       .
       .
       .
C     CALL MACHINE TAUONE
      CALL GAMMA (Q1, Q2, Q3, Q1, Q3, Q1, Q4, Q1, Q2, Q2)
       .
       .
       .
       .
       .
C     CALL MACHINE TAUTWO
      CALL GAMMA (Q2, Q1, Q3, Q2, Q3, Q1, Q4, Q1, Q4, Q2)
       .
       .
       .
       .
       .
C     CALL MACHINE TAUTHREE
      CALL GAMMA (Q3, Q1, Q4, Q2, Q3, Q4, Q4, Q1, Q4, Q2)
       .
       .
       .
       .
       .
C     CALL MACHINE TAUFOUR
      CALL GAMMA (Q4, Q4, Q4, Q4, Q4, Q4, Q4, Q4, Q4, Q4)
       .
       .
       .
       .
       .
```

The subprogram Γ appears dependent upon three branch logic whereas the environment can only give the program strings of two symbol inputs. However, the output to the environment from any final state indicates termination which, as a practical matter in FORTRAN, cannot be accomplished except by proper program exit. The third branch to the return statement can be considered as an activity of the environment providing no "input" per se to the machine nor requiring any further machine response.

Since programs always require some set of terminal points or states, as a practical matter their implementation as statements in any programming language necessitates including fixed exit points in the procedure. One cannot, as a rule, terminate a program in a higher level language after any statement except by transferring control to one of these legitimate exit points. So while each control has a set of these states where an interpretation maps these states onto a "return" function, the control must be implemented so it can turn itself off by transferring to one of the legitimate exits. From the control point of view, when the final or terminal state is reached, the control is inactive until restarted at a later time. For any programming language including a set of processor interrupt commands, no artificial devices such as that above are needed to effect

termination. Without this feature, common controls must, in general, provide the facility for proper termination at any state. This is one implementation problem, among others, which is circumvented with the next type of realization.

3.4. Composite Realizations

This section introduces a model of the most natural way of combining a number of distinct programs. The fundamental idea is apparent from the following example.

Example 3.2

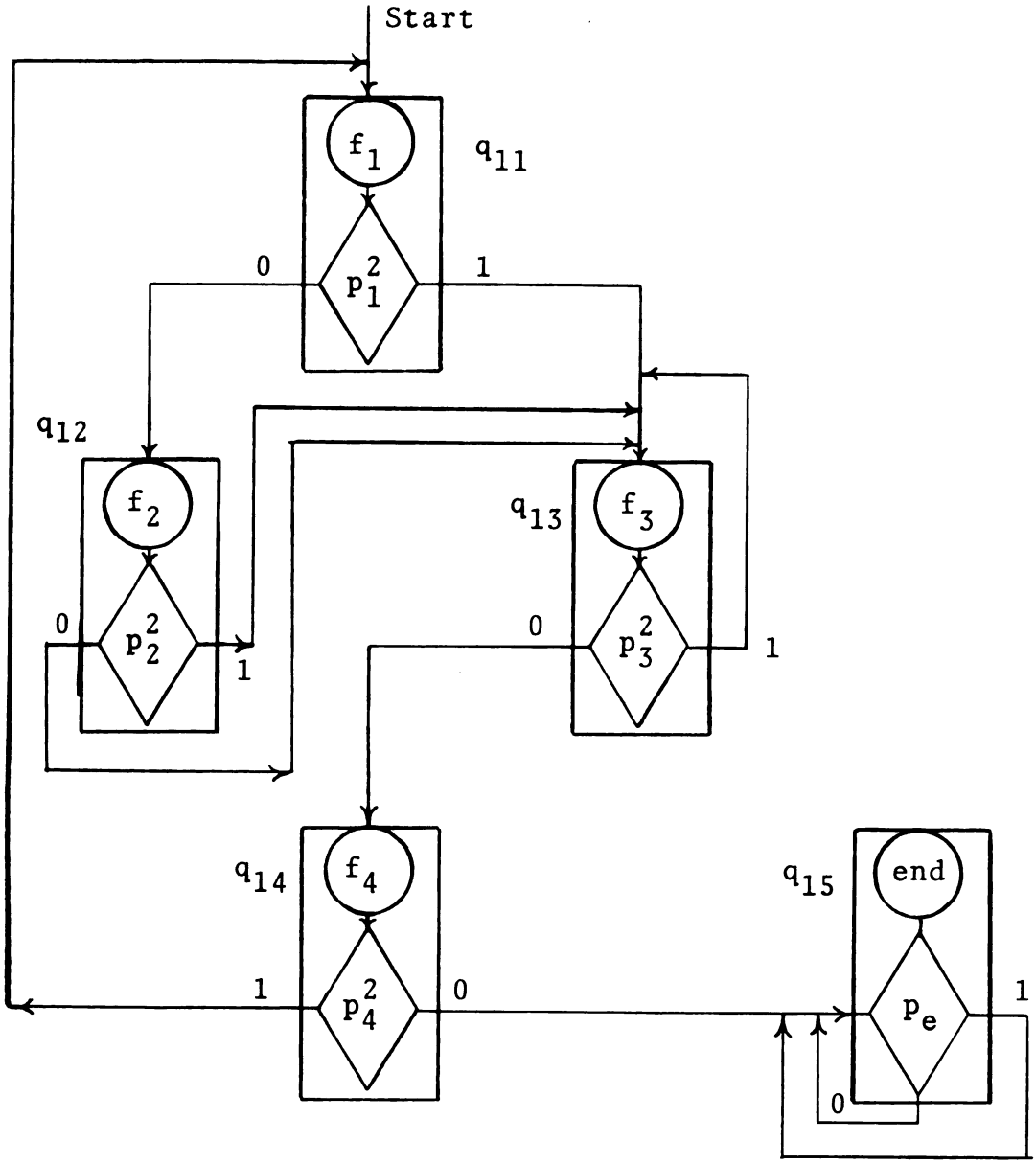
Let two programs $\mathcal{T}_1$, $\mathcal{T}_2$ whose flow charts appear in Figure 3.8, have the controls of Figure 3.9. Suppose interpretations are imposed on these controls by subroutine calls with external functions in the parameter list:

```
PROGRAM MAIN
  .
  .
  .
EXTERNAL Q11, Q12, Q13, Q14, Q21, Q22, Q23
  .
  .
  .
  .
  .
CALL PI (Q11, Q12, Q13, Q14)
  .
  .
  .
CALL P2 (Q21, Q22, Q23)
```

```
   SUBROUTINE P1 (Q1, Q2, Q3, Q4)
1  GO TO (2,3) Q1 (D)
2  GO TO (3,3) Q2 (D)
3  GO TO (4,3) Q3 (D)
4  GO TO (1,5) Q4 (D)
5  RETURN
   END

   SUBROUTINE P2 (Q1, Q2, Q3)
1  GO TO (2,3) Q1 (D)
2  GO TO (3,2) Q2 (D)
3  GO TO (1,4) Q3 (D)
4  RETURN
   END
```

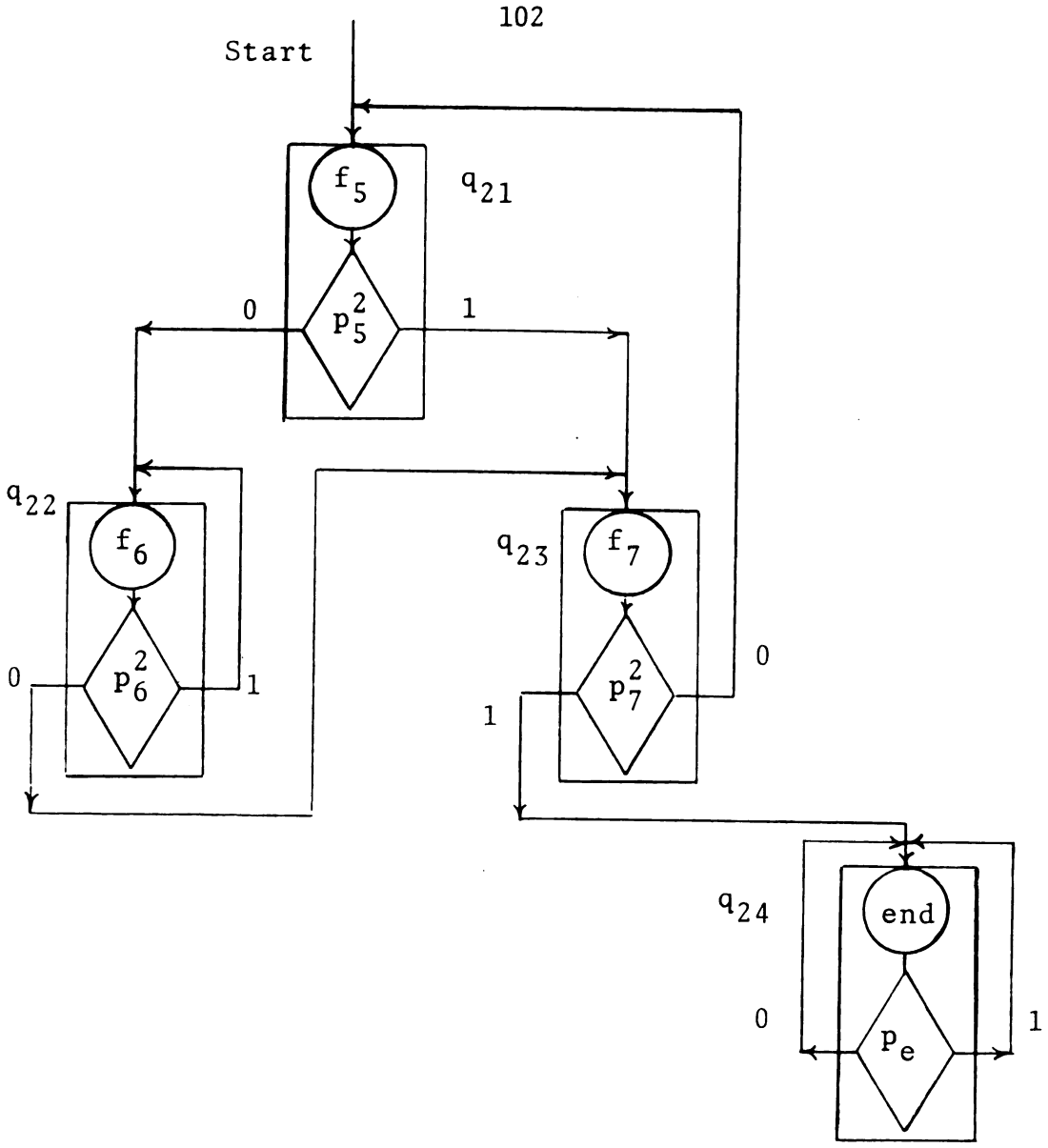
If one were asked to write a common control (in the above form) to implement both procedures, he might respond in the following fashion.



(a) π_1

Figure 3.8

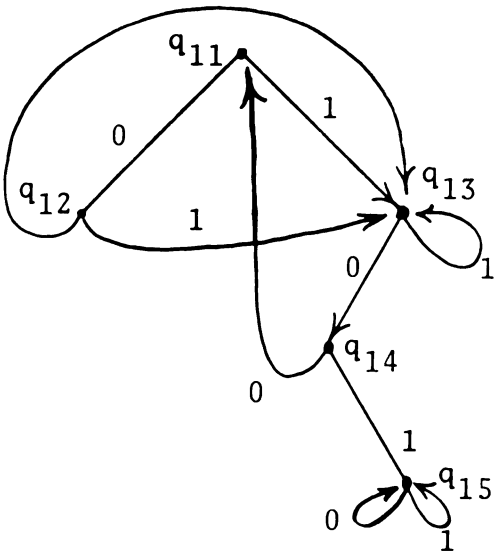
Flow Charts of π_1, π_2

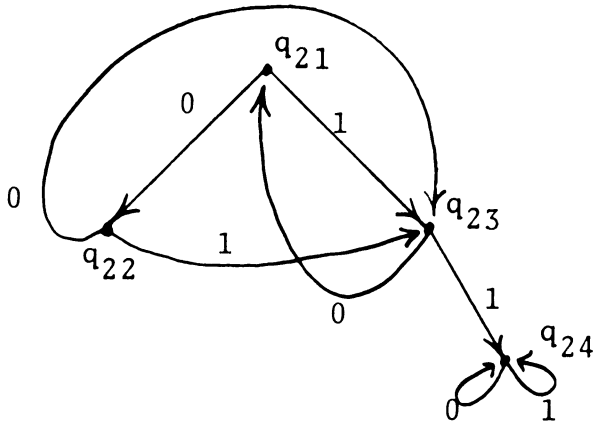


(b) $\overline{\P}_2$

Figure 3.8 (Continued)

Flow Charts of $\overline{\P}_1$, $\overline{\P}_2$





δ_{τ_1}:

	0	1
q_{11}	q_{12}	q_{13}
q_{12}	q_{13}	q_{13}
q_{13}	q_{14}	q_{13}
q_{14}	q_{11}	q_{15}
q_{15}	q_{15}	q_{15}

δ_{τ_2}:

	0	1
q_{21}	q_{22}	q_{23}
q_{22}	q_{23}	q_{22}
q_{23}	q_{21}	q_{24}
q_{24}	q_{24}	q_{24}

Figure 3.9

Controls τ_1, τ_2

```
      SUBROUTINE P12 (Q1, Q2, Q3, Q4)
  1   GO TO (2,3,2,3) Q1 (D)
  2   GO TO (3,3,3,2) Q2 (D)
  3   GO TO (4,3,4,5) Q3 (D)
  4   GO TO (1,5,2,3) Q4 (D)
  5   RETURN
      END
```

And the calls in MAIN

```
      CALL P12 (Q11, Q12, Q13, Q14)
      CALL P12 (Q21, Q22, Q23, Q21)
```

simulate P1 and P2 respectively. The decision predicates associated with each external function are modified so that the value of Qi(D) is 1 or 2 (3,4) for program $\overline{\pi}_1$ ($\overline{\pi}_2$). The common control, Γ, of subroutine P12 can be modeled by a composite construction using the two controls τ_1, τ_2. The control τ_1 has five states so Γ must have at least this many. Hence, let

$$S_\Gamma = \{s_i\} \quad i = 0,4$$

Now $|Q_{\tau_2}| = 4$ but there always is an n-state equivalent control (to τ_2) where n>4. One such 5-state control for τ_2 is pictured in Figure 3.10.

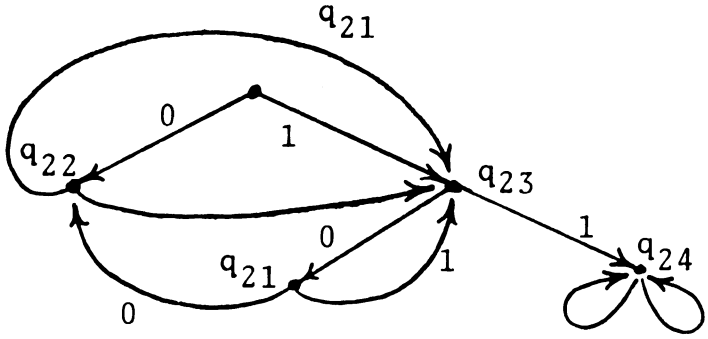


Figure 3.10

5-State Equivalent Control for τ_2

Let the input alphabet for Γ be

$$\Sigma_\Gamma = \{\sigma_{10}, \sigma_{11}, \sigma_{20}, \sigma_{21}\}$$

(where $\sigma_{10} = 1$, $\sigma_{11} = 2$, $\sigma_{20} = 3$, $\sigma_{21} = 4$ in the FORTRAN implementation) and the transition and output function be defined as :

$\lambda_\Gamma, \delta_\Gamma$:

	σ_{10}	σ_{11}	σ_{20}	σ_{21}	λ_Γ
s_0	s_1	s_2	s_1	s_2	y_0
s_1	s_2	s_2	s_2	s_1	y_1
s_2	s_3	s_2	s_3	s_5	y_2
s_3	s_0	s_5	s_1	s_2	y_3
s_4	s_4	s_4	s_4	s_4	y_4

Clearly the transition behavior of states of Γ under $\{\sigma_{10},\sigma_{11}\}$ is precisely that of τ_1 and similarly for τ_2 and the inputs $\{\sigma_{20},\sigma_{21}\}$. The construction simply makes a copy of each machine on a common grid by identifying separate inputs to each machine. Each control τ_i, is a homomorphic image of Γ via the maps

$$\phi^i \text{ where } \quad \phi^1 = \left(h_1^1,h_2^1,h_3^1\right),\ \phi^2 = \left(h_1^2,h_2^2,h_3^2\right)$$

and

	s_0	s_1	s_2	s_3	s_4
$h_1^1(s_i)$	q_{11}	q_{12}	q_{13}	q_{14}	q_{15}

	s_0	s_1	s_2	s_3	s_4
$h_1^2(s_i)$	q_{21}	q_{22}	q_{23}	q_{21}	q_{24}

$h_2^1\colon \Sigma_\Gamma^{\psi 1} \longrightarrow \Sigma_{\tau_1}$

$h_2^1(\sigma_{10}) = 0$

$h_2^1(\sigma_{11}) = 1$

$h_2^2\colon \Sigma_\Gamma^{\psi 2} \longrightarrow \Sigma_{\tau_1}$

$h_2^2(\sigma_{20}) = 0$

$h_2^2(\sigma_{21}) = 1$

$h_3^1\colon Y_\Gamma \longrightarrow Y_{\tau_1}$

$h_3^2\colon Y_\Gamma \longrightarrow Y_{\tau_2}$

where

$$h_3^i(y_j) = y'$$

$$\lambda_{\tau_i}\left[h_1^i(s_j)\right] = y';\ i = 1,2.$$

Γ is then a H-type realization for both τ_1 and τ_2, with the appropriate submachines (recall Theorem 1.1) of Γ described by

$$\Gamma\Big|_{\psi^i} = \left\langle S_\Gamma^{\psi_i},\ \Sigma_\Gamma^{\psi_i},\ Y_\Gamma,\ \delta_\Gamma^{\psi_i},\ \lambda_\Gamma^{\psi_i},\ s_0 \right\rangle$$

where

$$S_\Gamma^{\psi_i} = S_\Gamma$$

$$\Sigma_\Gamma^{\psi_1} = \left\{\sigma_{10},\sigma_{11}\right\}; \quad \Sigma_\Gamma^{\psi_2} = \left\{\sigma_{20},\sigma_{21}\right\}$$

$$\delta_\Gamma^{\psi_i}:\ S_\Gamma \times \Sigma_\Gamma^{\psi_i} \longrightarrow S_\Gamma$$

$$\lambda_\Gamma^{\psi_i}:\ S_\Gamma \longrightarrow Y_\Gamma\ .$$

The construction of the proper interpretations on Γ for Γ to be a common control for $\left\{\tau_1,\tau_2\right\}$ follows that of Theorem 2.7 and one implementation of these constructions is via the calls above. Recall that for some $I = \left\langle \eta,\gamma,\zeta,d \right\rangle$ on, say τ_1,

$$\gamma^H:\ S_\Gamma^{\psi_i} \longrightarrow P^4 \quad \text{since} \quad |\Sigma_\Gamma| = 4\ ,$$

if

$$\gamma^H(s) = h_1^1 \circ \gamma \circ \left(h_2^1\right)^{-1}(s) = \left(h_2^1\right)^{-1}\left(\gamma(q)\right) = \left(h_2^1\right)^{-1}(p^2) = p^4$$

where if

$$p^2:\ D \longrightarrow \left\{0,1\right\}$$

then

$$\left(h_2^1\right)^{-1}(p^2) = p^4\colon D \longrightarrow \left\{\left(h_2^1\right)^{-1}(0), \left(h_2^1\right)^{-1}(1)\right\} = \left\{\sigma_{10}, \sigma_{11}\right\}.$$

And the decision predicates defined by the maps in I^H make the same tests as those specified by I but encode the responses as a subset of Σ_Γ. Note also that the state homomorphisms mapped the starting state of S_Γ onto those of τ_1 and τ_2 as with the terminal states. This is always possible for interpreted controls with one exit point. Before the general construction method is presented, a means by which equivalent machines are generated is required.

Definition 3.2

For any cyclic control τ completely defined for some input alphabet Σ_τ, define the set of minimum grids, G_τ, as that set of pruned (i.e. all but tree branches eliminated) response trees of τ for which each node represents a distinct state in Q_τ and the path length from the root of the tree to that node is the smallest possible.

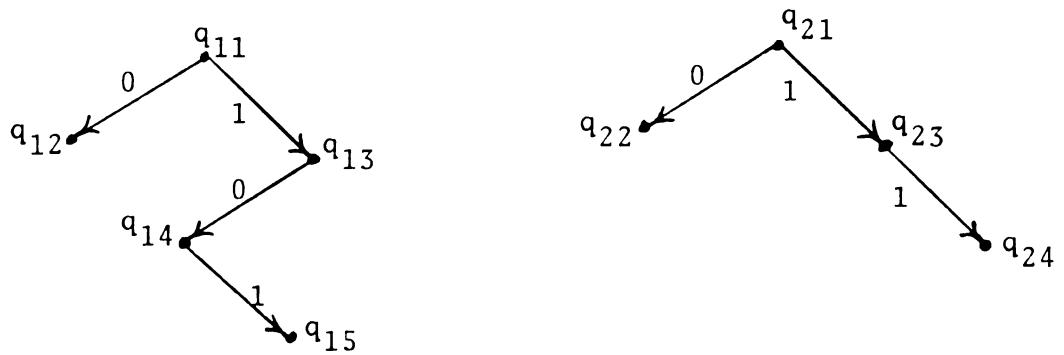
$$G_\tau = \left\{g_i\right\}, \quad g_i = \left\langle \left\{x_j^i\right\}_{j=1,n-1} \right\rangle$$

where to every $q_j \varepsilon Q_\tau$, there exists $x_j^i \varepsilon \Sigma_\tau^*$ such that

$$\delta_\tau(q_0, x_j^i) = q_j$$

Example 3.3

For τ_1, τ_2, in Example 3.2, G_{τ_1}, G_{τ_2} are unique and are shown in Figure 3.11 (a) and (b).



$$G_{\tau_1} = \{0,1,101\} \qquad G_{\tau_2} = \{0,1,11\}$$

Figure 3.11

Minimum Grids G_{τ_1}, G_{τ_2}

That is, there are no other trees on which the states of τ_1 and τ_2 can be mapped so that the path lengths from $q_{i1}\Big|_{i=1,2}$ are less than those above.

Example 3.4

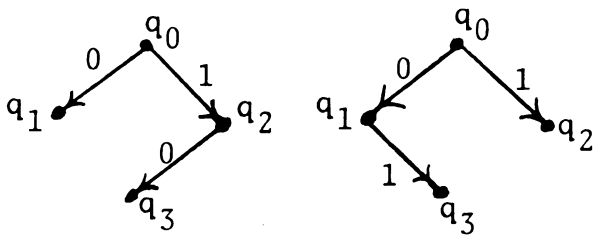
This example shows that G_τ is not necessarily unique for any τ. Consider the control, τ, in Figure 3.12.

δ_τ:

	0	1
q_0	q_1	q_2
q_1	q_0	q_3
q_2	q_3	q_2
q_3	q_0	q_3

(a)

G_τ:



(b)

Figure 3.12

Multiple Minimum Grids

The following theorem is trivial from the standpoint of finite state machine theory but it references a particular construction which shall be referred to in the sequel.

Theorem 3.3

For every cyclic n-state control, τ, there is a set of m-state controls $L_m[\tau]$, for any $m \geq n$ called the labeled extension of τ, where each machine in $L_m[\tau]$ is behaviorally equivalent to τ.

Proof:

There are a number of ways to construct such machines, for convenience $L_m[\tau]$ will be restricted to the method presented here.

(1) Pick any tree from G_τ and label from s_0 to s_{n-1} the nodes or states in some standard fashion starting with the root q_0 as s_0 and descending the tree from left to right numbering consecutively each encountered state. Redefine the

transition function δ_τ for this relabeling. Let this relabeled machine belong to $L_n[\tau]$ and define $L_n[\tau]$ to be all relabeled machines generated by G_τ.

(2) Since τ is cyclic at least n-1 entries in the state table exist which define any such tree. All other entries correspond to those arcs eliminated from the grid.

(3) Identify any such grid $g_i \varepsilon G_\tau$ as a subtree in the complete response tree over Σ_τ for the relabeled control τ. There are at least n-1 (and no more than n) nodes from which arcs can lead out of g_i and generate new subtrees. Label any node in the response tree not in the subtree g_i and reachable by some arc from a node s_j in g_i as s_n. Then the n+1-state equivalent machine to τ is constructed by redefining δ_τ as follows:

(a) If for node, s_j, $x_j^i \varepsilon \Sigma_\tau^*$, $x_j^i \varepsilon g_i \varepsilon G_\tau$

and $\delta_\tau\left(s_0, x_j^i\right) = s_j$; $j \leq n-1$ in the

relabeled n-state machine τ, and

if $x_j^i \sigma \notin g_i$, some $\sigma \varepsilon \Sigma_\tau$

then

$$\delta_\tau(s_0, x_j^i \sigma) = \delta_\tau(s_j, \sigma) = s_k; \; k \leq n-1 .$$

(b) Redefine δ_τ on state s_j as

$$\delta_{\tau'}(s_0, x_j^i\sigma) = \delta_{\tau'}(s_j, \sigma) = s_n$$

and define

$$\delta_{\tau'}(s_0, x_j^i\sigma\sigma_\ell) = \delta_{\tau'}(s_n, \sigma_\ell) = \delta_\tau(s_k, \sigma_\ell)$$

for all $\sigma_\ell \varepsilon \Sigma_\tau$

$$\lambda_{\tau'}(s_n) = \lambda_{\tau'}(s_k)$$

$\delta_{\tau'} = \delta_\tau$; $\lambda_{\tau'} = \lambda_\tau$ otherwise.

(c) Then $\tau' = \left\langle Q_{\tau'}, \Sigma_\tau, \delta_{\tau'}, \lambda_{\tau'}, Y_{\tau'}, s_0 \right\rangle$ where

$Q_{\tau'} = Q_\tau \cup \left\{s_n\right\}$; Q_τ: the relabeled states of τ

and

$$G_{\tau'} = \left\{g_i\right\},\ g_i' = g_i \cup \left\{x_j^i\sigma\right\},\ g_i \varepsilon G_\tau.$$

(d) Then τ is behaviorally equivalent τ, an $n+1$ state machine $\varepsilon L_{n+1}[\tau]$.

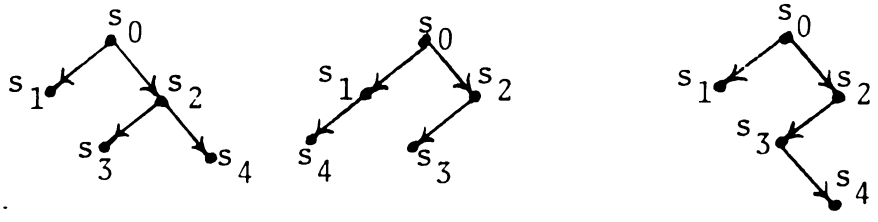
(e) Set $\tau = \tau'$; $G_\tau = G_{\tau'}$ and continue the process until $|Q_{\tau'}| = m$.

(4) $L_m[\tau]$ then contains those machines which can be constructed in this fashion, all of which are equivalent to τ.

Example 3.5

Consider the control and grids in Figure 3.12(a) and (b). A number of labeled extended 5-state equivalent machines with representative grids for this machine are given in Figure 3.13.

$G_{\tau'}$:



$\delta_{\tau'}$:

	0	1
s_0	s_1	s_2
s_1	s_0	s_3
s_2	s_3	s_4
s_3	s_0	s_2
s_4	s_3	s_4

	0	1
s_0	s_1	s_2
s_1	s_4	s_3
s_2	s_3	s_2
s_3	s_0	s_2
s_4	s_1	s_2

	0	1
s_0	s_1	s_2
s_1	s_0	s_3
s_2	s_3	s_2
s_3	s_0	s_4
s_4	s_3	s_2

Figure 3.13

5-State Equivalent Machines in $L_5[\tau]$

Theorem 3.4

A cyclic deterministic n-state control, τ, is the homomorphic image of any m-state labeled extension $\tau' \varepsilon L_m[\tau]$, $m \geq n$.

Proof:

(1) For every machine $\tau' \varepsilon L_n[\tau]$, each state s_i was a relabeling of $q_i \varepsilon Q_\tau$,

hence

$$h_1'(s_i) = q_i \; ; \; i = 0, n-1$$

h_2': $\Sigma_{\tau'} \longrightarrow \Sigma_\tau$; h_2': the identity transformation

h_3': $Y_{\tau'} \longrightarrow Y_\tau$; h_3': the identity transformation

(2) To generate some machine $\tau'' \varepsilon L_{n+1}[\tau]$, s_n was necessarily behaviorally equivalent to some s_j, $j = 0, n-1$

hence

$$h_1''(s_i) = q_i, \quad i = 0, n-1$$

$$h_1''(s_n) = h_1'(s_j) = q_j$$

$$h_2'' = h_2'$$

$$h_3''(y_i'') = y_i, \quad i = 0, n-1$$

$$h_3''(y_n'') = y_j, \; \lambda_{\tau''}(s_n) = y_n'', \; \lambda_\tau(q_j) = y_j$$

(3) Continuing in this way the 3 part homomorphism can be defined for any machine in $L_m[\tau]$ onto τ.

$\triangleright\triangleleft$

Theorem 3.3 provides the key construction used in the proof of the major theorem of this section.

Theorem 3.5

For any set of cyclic, deterministic, FSM, controls $\{\tau_i\}_{i=1,n}$ there is a common control Γ,

$$\Gamma = \langle S_\Gamma, \Sigma_\Gamma, Y_\Gamma, \delta_\Gamma, \lambda_\Gamma, s_0 \rangle$$

such that

(1) $|S_\Gamma| = m$, where there exists τ_j, $j = 1,n$, such that

$$|Q_{\tau_j}| = m \geq |Q_{\tau_k}| \text{ for all } k = 1,n.$$

(2) $$|\Sigma_\Gamma| = \sum_{i=1}^{n} |\Sigma_{\tau_i}| \; ; \; \Sigma_\Gamma = \bigcup_{i=1,n} \bigcup_{j=1,|\Sigma_{\tau_i}|} \{\sigma_{ij}\}.$$

(3) $\tau_i H_I \Gamma$ for all $i = 1,n$.

Proof:

(1) For any control, say τ_j, with the largest number of states

$$|Q_{\tau_j}| = m \geq |Q_{\tau_k}| \text{ for all } k = 1,n$$

pick any m-state labeled extension of τ_j from $L_m[\tau_j]$ and let this set of labels $\{s_i\}_{i=0,m-1}$ be the state set S_Γ.

This set of labels is common to all $L_m[\tau_i]$, $i = 1,n$, and for each τ_i, pick one m-state labeled extension $\tau_i \varepsilon L_m[\tau_i]$.

(2) Define the injective map h_2^{i-1}

$$h_2^{i-1}: \Sigma_{\tau_i} \xrightarrow[\text{into}]{1\text{-}1} \{\sigma_{ij}\} \quad j = 1, |\Sigma_{\tau_i}|.$$

(3) Then $\delta_\Gamma, \lambda_\Gamma$ are defined by

$$\delta_\Gamma(s_k, \sigma_{ij}) = \delta_{\tau_i}\left(s_k, h_2^i(\sigma_{ij})\right) \quad k = 1,m \qquad j = 1, |\Sigma_{\tau_i}|$$

$$\lambda_\Gamma(s_k) = Y_k \; ; \; k = 1,m.$$

(4) Consider each submachine of Γ of the following form

$$\Gamma\Big|_{\psi_i} = \left\langle S_\Gamma^{\psi_i}, \Sigma_\Gamma^{\psi_i}, \delta_\Gamma^{\psi_i}, \lambda_\Gamma^{\psi_i}, s_0 \right\rangle$$

$$S_\Gamma^{\psi_i} = S_\Gamma$$

$$\Sigma_\Gamma^{\psi_i} = \{\sigma_{ij}\}\Big|_{j = 1, |\Sigma_{\tau_i}|}$$

$$\delta_\Gamma^{\psi_i}: S_\Gamma \times \Sigma_\Gamma^{\psi_i} \longrightarrow S_\Gamma$$

$$\lambda_\Gamma^{\psi_i} = \lambda_\Gamma.$$

These submachines are isomorphic to the chosen $\tau' \varepsilon L_m[\tau_i]$ and since each τ_i is a homomorphic image of τ_i' from Theorem 3.4, replace the identity input alphabet map of that theorem with h_2^i above, hence verifying that τ_i is a homomorphic image of the submachine $\Gamma\Big|_{\psi_i}$ under the three part homomorphism $\phi^i = \left(h_1^i, h_2^i, h_3^i\right)$:

$$h_1^i: S_\Gamma^{\psi_i} \longrightarrow Q_{\tau_i}$$

$$h_2^i: \Sigma_\Gamma^{\psi_i} \longrightarrow \Sigma_{\tau_i}$$

$$h_3^i: Y_\Gamma^{\psi_i} \longrightarrow Y_{\tau_i}$$

(5) The conditions of Theorem 2.7 have been met, hence Γ is a common control for $\left\{\tau_i\right\}$ and

$$\tau_i H_I \Gamma \quad \text{for all} \quad i = 1,n. \qquad \triangleright\triangleleft$$

Remark: Now if only one exit point is identified in every program $I[\tau_i]$ it is easy to see how the relabeling process can be constrained so that for every such exit state $q_i^e \varepsilon Q_{\tau_i}$ then

$$h_1^i(s_{m-1}^e) = q_i^e \; ; \; i=1,m \quad \text{if} \quad s_{m-1}^e \leftrightarrow \text{exit state in } \Gamma.$$

This eliminates the necessity of introducing superficial transfers to the Γ exit states as in the Section 3.3 example. The example introducing this section illustrates this process. For every program with an n-state control and a number of exit states, one additional state can be added to form an n+1 equivalent control where all exits are made via transfer to this state. Hence this particular structural realization facilitates a simple implementation of any program $I[\tau_i]$ on Γ. The extension of the τ_i to m-state equivalents is advantageous not only in this regard but also to provide a completely defined control Γ over the augmented input alphabet Σ_Γ. The example of this section also shows the straightforward fashion in which the decision predicates are modified for any interpretation--a direct consequence of the injective map h_2^i characterizing the H-type structural realization.

CHAPTER IV

MINIMUM STRUCTURAL REALIZATIONS

Having examined some H-type common control realizations, Sections 4.1 and 4.2 introduce another H-type realization which also can be constructed for any set of cyclic controls.

4.1 Subdirect Product Realizations

Product algebras are common mathematical constructs of systems which preserve all the capability of their component algebras. Not surprisingly, a direct product of finite state machines is a useful device to construct H-type common control realizations. The semigroup realization was of this nature and the method in general is described by Theorem 4.1.

Theorem 4.1

Given a set of deterministic, cyclic, FSM controls; the control

$$\Gamma = \left(\underset{i=1}{\overset{n}{\times}} \tau_i \right)^c \quad \text{over} \quad \Sigma_\Gamma = \Sigma_{\tau_i}; \quad i = 1,n$$

$$= \left\langle \left\{ \left(q_{j_1}, q_{j_2}, .., q_{j_n} \right) \right\}^r, \Sigma_\Gamma, Y_\Gamma, \delta_\Gamma, \lambda_\Gamma, \left(q_{0_1}, ..., q_{0_n} \right) \right\rangle$$

is a H-type common control realization and

$$\tau_i H_I \Gamma \quad ; \qquad i = 1,n.$$

Proof:

An outline of the proof which is quite similar to that of Theorem 3.1 follows.

(1) It is clear that the component controls τ_i are homomorphic images of Γ since

$$S_\Gamma = \left\{\left(q_{j_1},\ldots,q_{j_n}\right)\right\}^r \quad ; \quad q_{j_i} \varepsilon Q_{\tau_i}$$

where there exists $x \varepsilon \Sigma_\Gamma^*$ such that

$$\delta_\Gamma\left[\left(q_{0_1},\ldots,q_{0_n}\right),x\right] = \left[\delta_{\tau_1}\left(q_{0_1},x\right),\ldots,\delta_{\tau_n}\left(q_{0_n},x\right)\right]$$

$$= \left(q_{j_1},\ldots,q_{j_n}\right) \varepsilon S_\Gamma$$

then $\phi^i = \left(h_1^i, h_2^i, h_3^i\right)$

where

(a) $h_1^i : S_\Gamma \rightarrow Q_{\tau_i}$

$$h_1^i\left(q_{j_1},q_{j_2},\ldots,q_{j_i},\ldots,q_{j_n}\right) = q_{j_i} \quad \text{for all } i = 1,n$$

$$\text{for all } j = 1,|Q_\Gamma|$$

(b) h_2^i = identity map for all i

(c) $h_3^i(y_k) = y_i'$ iff $\lambda_\Gamma(s_k) = y_k$ and

$$\lambda_{\tau_i}\left[h_1^i(s_k)\right] = y_i' \; ; \; s_k \in S_\Gamma$$

(2) The submachine of the homomorphism is Γ itself and the conditions of Theorem 2.7 have been met. $\triangleright\triangleleft$

The control $\left(\underset{i=1}{\overset{n}{\times}} \tau_i\right)$ is usually called the <u>subdirect</u> (or <u>direct</u>) <u>product</u> of the set $\{\tau_i\}$. Like the composite and semigroup realizations (a special case of the direct product), constructing the proper interpretations I_j^H on Γ to simulate some program $I_j[\tau_i]$ is particularly simple from the standpoint of choosing the correct computations, decision predicates and starting state (only one entry point to Γ is necessary). However unlike composite realizations, in direct product realizations the exit point problem must be anticipated and facilities for multiple exits in Γ provided. It is also apparent that while composite realizations minimize the number of states in the control Γ at the expense of augmenting the input alphabet, direct product realizations have just the opposite relationship. Another feature direct product realizations have is pointed out in the following theorem.

Theorem 4.2

Given a set of controls $\{\tau_i\}$ and common control Γ as in Theorem 4.1, if for each τ_i and $I_{ij} = \langle \eta_{ij}, \gamma_{ij}, \zeta_{ij}, d_{ij} \rangle$ on τ_i the map

$$W_{ij}: \tilde{S}_D\left(\overline{I_{ij}[\tau_i]}\right) \longrightarrow \tilde{V}_D\left(\overline{I_{ij}[\tau_i]}\right) \quad \text{is bijective}$$

then

$$W^H_{ij}: \tilde{S}_D\left(\overline{I^H_{ij}[\Gamma]}\right) \longrightarrow \tilde{V}_D\left(\overline{I^H_{ij}[\Gamma]}\right) \quad \text{is bijective.}$$

Proof:

(1) If W_{ij} is 1-1 and onto then by Theorem 2.3

$$\zeta_{ij}\left(\lambda_{\tau_i}\left[\delta_{\tau_i}(q_{k_i}, \sigma)\right]\right) \neq \zeta_{ij}\left(\lambda_{\tau_i}\left[\delta_{\tau_i}(q_{k_i}, \sigma')\right]\right)$$

for any $q_{k_i} \in Q_{\tau_i}$; for all $\sigma, \sigma' \in \Sigma_{\tau_i} = \Sigma_\Gamma$, $\sigma \neq \sigma'$

or

$$\delta_{\tau_i} \circ \lambda_{\tau_i} \circ \zeta_{ij}(q_{k_i}): \Sigma_{\tau_i} \longrightarrow F^* \quad \text{is injective.}$$

The same must be shown for δ_Γ, λ_Γ and ζ^H_{ij}.

(2) For any

$$s_k = \left(q_{k_1}, q_{k_2}, \ldots, q_{k_i}, \ldots, q_{k_n}\right)$$

$$s_m = \left(q_{m_1}, .., q_{m_i}, \ldots, q_{m_n}\right);\ \lambda_\Gamma[s_m] = y;\ \lambda_{\tau_i}[q_{m_i}] = z$$

$$s_{m'} = \left(q_{m'_1}, .., q_{m'_i}, .., q_{m'_n}\right);\ \lambda_\Gamma[s_{m'}] = y';\ \lambda_{\tau_i}[q_{m'_i}] = z'$$

such that

$$\delta_\Gamma(s_k,\sigma) = s_m, \quad h_1^i(s_m) = q_{m_i}$$

$$\delta_\Gamma(s_k,\sigma') = s'_m, \quad h_1^i(s'_m) = q'_{m_i}$$

then condition (1) says that

$$\zeta_{ij}\left[h_3^i\left(\lambda_\Gamma\left[\delta_\Gamma(s_k,\sigma)\right]\right)\right] \neq \zeta_{ij}\left[h_3^i\left(\lambda_\Gamma\left[\delta_\Gamma(s_k,\sigma')\right]\right)\right]$$

for $\sigma \neq \sigma'$

or

$$\zeta_{ij}\left[h_3^i\left(\lambda_\Gamma\left[s_m\right]\right)\right] \neq \zeta_{ij}\left[h_3^i\left(\lambda_\Gamma\left[s'_m\right]\right)\right]$$

because

$$h_3^i\left(\lambda_\Gamma\left[s_m\right]\right) = h_3^i(y) = z$$

$$h_3^i\left(\lambda_\Gamma\left[s'_m\right]\right) = h_3^i(y') = z'$$

and

$$\zeta_{ij}(z) \neq \zeta_{ij}(z')$$

But this is true for all s_k, so

$\zeta_{ij}\left[h_3^i\left(\lambda_\Gamma\left[\delta_\Gamma(s_k,\sigma)\right]\right)\right]$ is unique for distinct con-

trol input symbols, $\sigma\epsilon\Sigma_\Gamma$, or equivalently the map

$$\delta_\Gamma \circ \lambda_\Gamma \circ h_3^i \circ \zeta_{ij}(s_k) = \delta_\Gamma \circ \lambda_\Gamma \circ \zeta_{ij}^H(s_k) : \Sigma_\Gamma \longrightarrow F^*$$

is injective. ▷◁

This theorem provides a mathematical way of expressing the notion of simultaneity. For if the conditions in Theorem 4.2 hold and one considers the set

$$\chi = \bigcap_{i,j} \left\{ W^{-1}\left[\tilde{V}_D\left(\overline{I_{ij}^{H}[\Gamma]} \right) \right] \right\}_{i,j} \subseteq \left\{ \tilde{S}_D\left(\overline{I_{ij}^{H}[\Gamma]} \right) \right\}_{i,j}$$

then every valuation sequence in χ describes one and only one value for <u>each</u> of the programs $\overline{I_{ij}^{H}[\Gamma]}$ on some initial configuration $d \in D$. Hence one input tape to the control Γ can describe the simultaneous execution of any programs $\left\{ \overline{I_{ij}^{H}[\Gamma]} \right\}$, and consequently $\left\{ \overline{I_{ij}[\tau_i]} \right\}$ and their values $\left\{ V_d\left(\overline{I_{ij}[\tau_i]} \right) \right\}$ for some $\{d\} \in D$. This property cannot be true for composite realizations in general because the input alphabets of the individual controls are of different size and two different submachines in Γ usually cannot serve to simulate the same control. Composite realizations then cannot serve as models for parallel processing but could adequately model a set of programs time sharing a central processor in a multiprogramming environment. Direct product realizations can serve both causes but at the expense

of introducing many redundant states.

The following rather detailed example illustrates the direct product realization for two typical numerical analysis algorithms and introduces the idea behind the control transformation which will be discussed at length in later sections.

Example 4.1

Consider the following two flow charts for Runga-Kutta and modified Newton-Raphson algorithms in Figures 4.1 and 4.2.

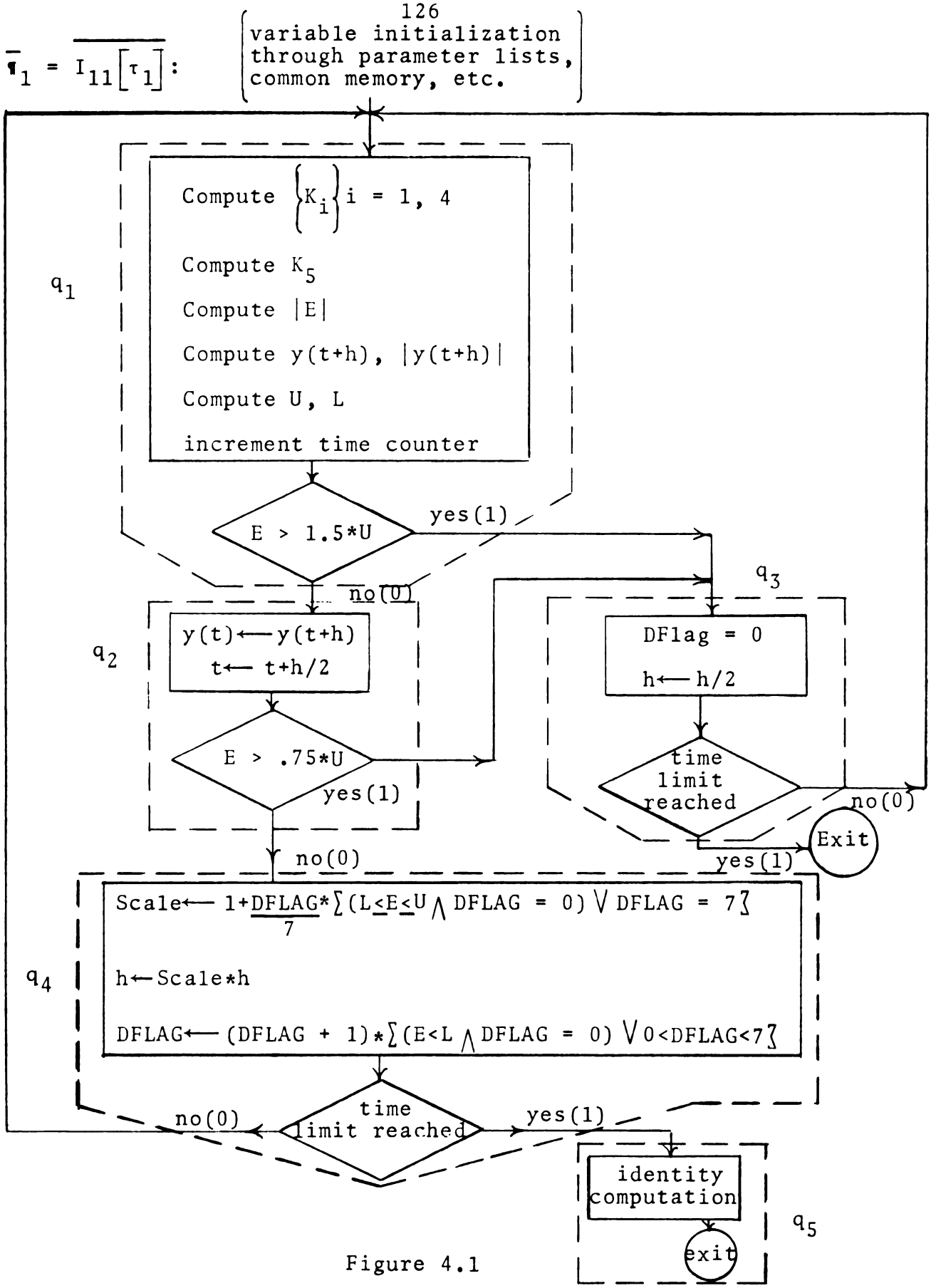


Figure 4.1

4th Order Runga-Kutta Formulation for the Numerical Solution of the Differential Equation y = f(t,y)

where

$$y(t+h) = y(t) + 1/6 \left| k_1 + 2k_2 + 2k_3 + k_4 \right|$$

$$k_1 = h*f\left(t, y(t)\right)$$

$$k_2 = h*f\left(t + h/2,\ y(t) + 1/2\ k_1\right)$$

$$k_3 = h*f\left(t + h/2,\ y(t) + 1/2\ k_2\right)$$

$$k_4 = h*f\left(t + h,\ y(t) + k_3\right)$$

$$k_5 = h*f\left(t + h,\ y(t+h)\right)$$

$$E = k_1 - 2k_3 - 2k_4 + 3k_5$$

$$U = u_1 + u_2*|y(t+h)|$$

$$L = m_1 + m_2*|y(t+h)|$$

where:

$u_1, u_2, m_1, m_2 \geq 0$, set initially for absolute or relative error control. The quantities

$$\{(L \leq E \leq U \wedge DFLAG = 0) \vee DFLAG = 7\}$$

and

$$\{(E < L \wedge DFLAG = 0) \vee 0 < DFLAG < 7\}$$

are binary-valued (0 or 1) decision predicates.*

---

* This flow chart is derived from the algorithm found in the SCEPTRE manual, Vol. II, Mathers [1967], p. 28-30.

$\overline{\pi_2} = \overline{I_{21}[\tau_2]}$:

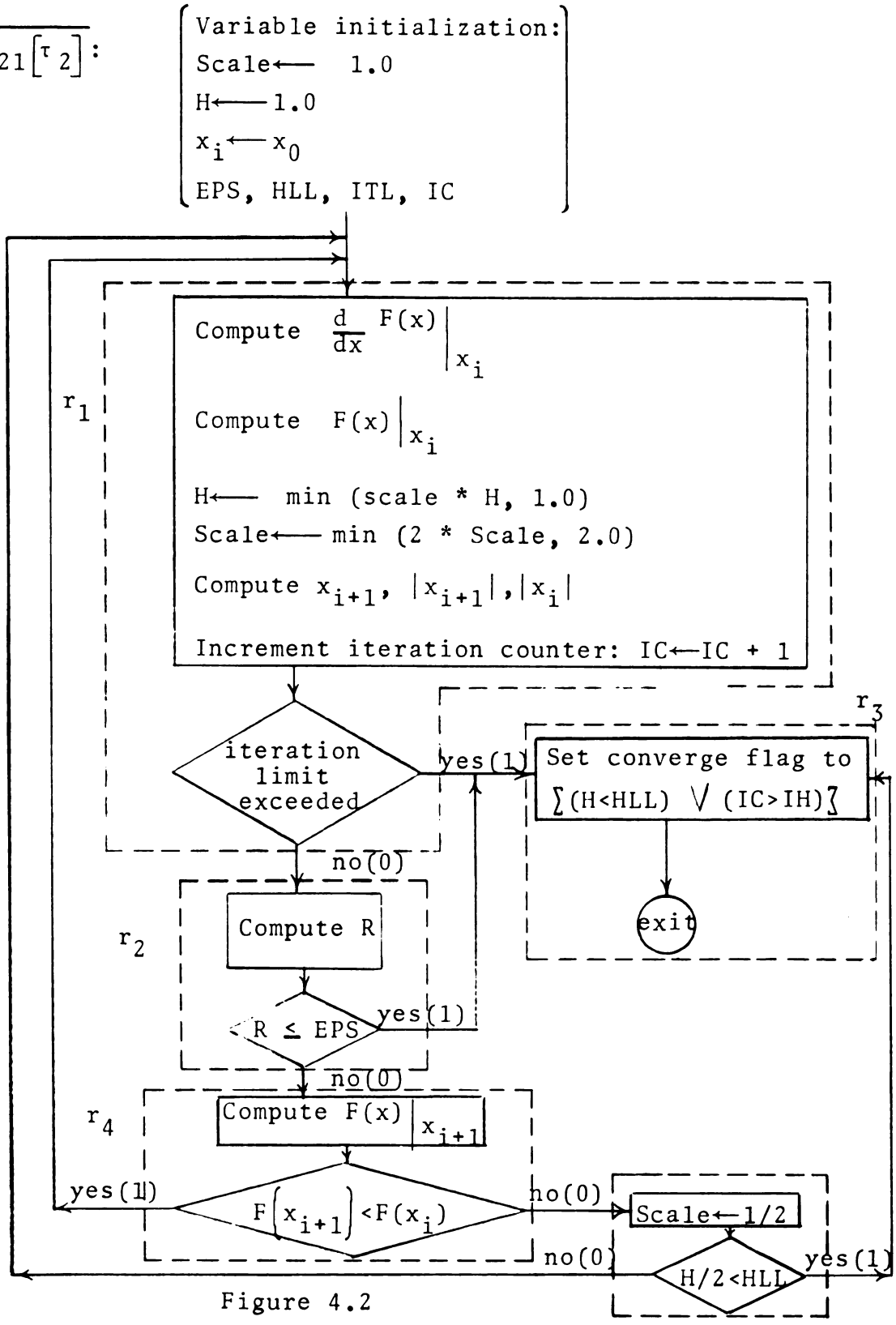


Figure 4.2

Newton-Raphson Scheme for Solution to Nonlinear Equation F(x)=0

The method uses the following convergence and improvement tests (governing convergence rate):

$$x_{i+1} = x_i - H * \frac{F(x_i)}{F'(x_i)}$$

if two improvements: $H \leftarrow 2*H$
if one improvement: $H \leftarrow H$
if no improvement: $H \leftarrow H/2$

where the improvement test is $F(x_{i+1}) < F(x_i)$.

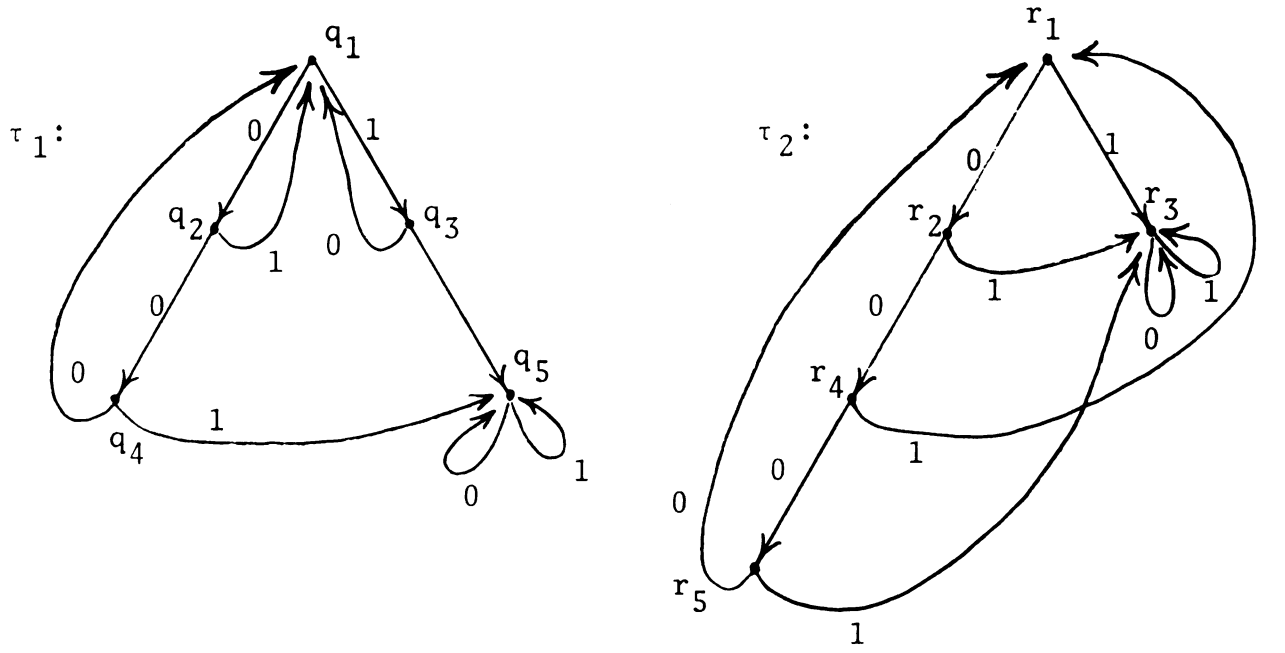
The convergence test used is $\frac{\left||x_{i+1}|-|x_i|\right|}{|x_{i+1}|+|x_i|} = R \leq$ EPS:(relative error).

The iteration limit = ITL, and the quantity

$$\{(H < HLL) \vee (IC > ITL)\}$$

is binary valued (0 or 1).

The controls τ_1 and τ_2 and the η_{11}, η_{21} maps for $I_{11}[\tau_1]$, $I_{21}[\tau_2]$ appear in Figure 4.3 (a) and (b).



δ_{τ_1}:	0	1	δ_{τ_1}	η_{11}
q_1	q_2	q_3	y_1	$(f_1 p_1^2)$
q_2	q_4	q_1	y_2	$(f_2 p_2^2)$
q_3	q_1	q_5	y_3	$(f_3 p_3^2)$
q_4	q_1	q_5	y_4	$(f_4 p_4^2)$
q_5	q_5	q_5	y_5	$(f_5 p_5^2)$

(a)

δ_{τ_2}:	0	1	δ_{τ_2}	η_{21}
r_1	r_2	r_3	z_1	$(f_6 p_6^2)$
r_2	r_4	r_3	z_2	$(f_7 p_7^2)$
r_3	r_3	r_3	z_3	$(f_8 p_8^2)$
r_4	r_5	r_1	z_4	$(f_9 p_9^2)$
r_5	r_1	r_3	z_5	$(f_{10} p_{10}^2)$

(b)

Figure 4.3

Controls for Runga-Kutta and Newton-Raphson Programs

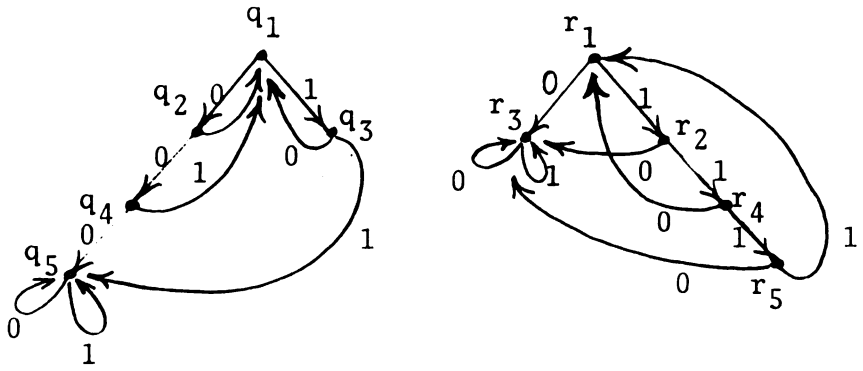
Now for

$$\Gamma = \tau_1 \times \tau_2 = \left\langle (Q_{\tau_1} \times Q_{\tau_2})^r, \Sigma, Y_\Gamma, \delta_\Gamma, s_0 \right\rangle \; ; \; s_0 = (q_1, r_1)$$

it can be shown that $\left| (Q_{\tau_1} \times Q_{\tau_2})^r \right| = 24$. But suppose the state transitions in τ_1 and τ_2 were changed as in Figure 4.4 (a) and (b). Then for these controls

$$\Gamma = \tau'_1 \times \tau'_2 = \left\langle (Q_{\tau'_1} \times Q_{\tau'_2})^r, \Sigma, Y_\Gamma, \delta_\Gamma, s_0 \right\rangle$$

resulting in the reduction, $\left| (Q_{\tau'_1} \times Q_{\tau'_2})^r \right| = 11$.



$\delta_{\tau'_1}$:

0	1	η'_{11}
q_1	q_2	q_3	$(f_1 p_1^2)$
q_2	q_4	q_1	$(f_2 p_2^2)$
q_3	q_1	q_5	$(f_3 p_3^2)$
q_4	q_5	q_1	$(f_4 \bar{p}_4^2)$
q_5	q_5	q_5	$(f_5 \bar{p}_5^2)$

(a)

$\delta_{\tau'_2}$:

0	1	η'_{21}
r_1	r_3	r_2	$(f_6 \bar{p}_6^2)$
r_2	r_3	r_4	$(f_7 \bar{p}_7^2)$
r_3	r_3	r_3	$(f_8 \bar{p}_8^2)$
r_4	r_1	r_5	$(f_9 \bar{p}_9^2)$
r_5	r_3	r_1	$(f_{10} \bar{p}_{10}^2)$

(b)

Figure 4.4

Transformed Controls τ'_1, τ'_2

The associated semi-control, $\hat{T}$, is shown in Figure 4.5, and

$\delta_{\hat{T}}$:

	0	1
s_1	s_2	s_3
s_2	s_4	s_5
s_3	s_5	s_6
s_4	s_7	s_5
s_5	s_2	s_8
s_6	s_9	s_{10}
s_7	s_7	s_7
s_8	s_2	s_7
s_9	s_7	s_{11}
s_{10}	s_7	s_9
s_{11}	s_7	s_6

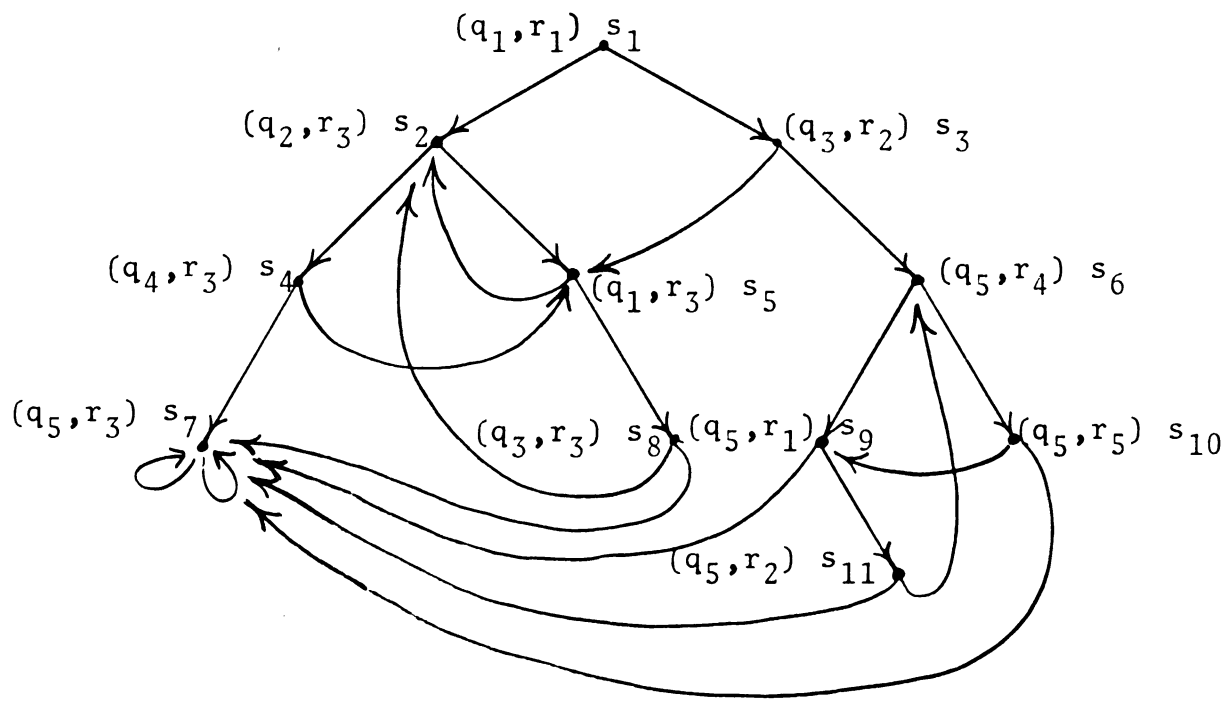


Figure 4.5

Direct Product of Transformed Controls

the maps h_1^1, h_1^2 are shown in Figure 4.6.

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}
$h_1^1(s_i)$	q_1	q_2	q_3	q_4	q_1	q_5	q_5	q_3	q_5	q_5	q_5
$h_1^2(s_i)$	r_1	r_3	r_2	r_3	r_3	r_4	r_3	r_3	r_1	r_5	r_2

Figure 4.6

State Homomorphisms of Γ onto τ_1 and τ_2

The appropriate maps η_{11}^H, η_{21}^H are read directly from Figure 4.6 and Figure 4.4 (a) and (b). The implementation of Γ in FORTRAN follows closely that of the example in section 3.3.

Notice the permutation of the entries in the transition tables for τ_1 and τ_2 in Figure 4.4. In τ_1, the transformed transition function, δ'_{τ_1}, reads as

$$\delta'_{\tau_1}(q_4, 0) = \delta_{\tau_1}(q_4, 1) = q_5$$

$$\delta'_{\tau_1}(q_4, 1) = \delta_{\tau_1}(q_4, 0) = q_1$$

Now if the binary decision predicate p_4^2 is changed to $\bar{p}_4^2$ so that

$$p_4^2(D) = 0 \Rightarrow \bar{p}_4^2(D) = 1$$

$$p_4^2(D) = 1 \Rightarrow \bar{p}_4^2(D) = 0$$

then the program $\overline{I_{11}[\tau\ 1]}$ has exactly the same value for any $d\varepsilon D$ and all valuation sequences for $I_{11}[\tau\ 1]$ are obtained from those of $I_{11}[\tau\ 1]$ with certain zeros replaced by ones and vice versa. This is the essence of the transformation which shall be considered in the following section.

4.2 Value Equivalent Control Transformations

For the rest of this chapter all controls will be defined over a binary input alphabet, {0,1}. The results and notations can be extended over an arbitrary number of symbols.

Definition 4.1

For any FSM control

$$\tau = \langle Q_\tau, \Sigma_\tau, Y_\tau, \delta_\tau, \lambda_\tau, q_0 \rangle \ ; \ |Q_\tau| = n, \ \Sigma_\tau = \{0,1\}$$

define a reordering of τ to be the control

$$i_\tau = \langle Q_\tau, \Sigma_\tau, Y_\tau, {}^{i}\delta_\tau, \lambda_\tau, q_0 \rangle \ ; \ 0 \leq i \leq 2^n-1$$

where if the n-bit binary representation of i is

$$(i)_{10} = b_{n-1} * 2^{n-1} + b_{n-2} * 2^{n-2} + \ldots b_1 * 2^1 + b_o * 2^0$$

and

$$(i)_2 = (b_{n-1}, b_{n-2}, \ldots b_1, b_0) \ ; \ b_i \varepsilon \{0,1\} \ ; \ 0 \leq i \leq 2^n-1$$

then if $b_j = 1$,

$$i_{\delta_\tau}(q_j,0) = \delta_\tau(q_j,1)$$

$$i_{\delta_\tau}(q_j,1) = \delta_\tau(q_j,0)$$

and if $b_j = 0$,

$$i_{\delta_\tau}(q_j,\sigma) = \delta_\tau(q_j,\sigma) \; ; \; \sigma \in \{0,1\}$$

Example 4.2

(1) Clearly

$$\tau = 0_\tau \cong 2^n - 1_\tau$$

(2) For controls τ_1, τ_2, in Figure 4.3 (a) and (b), the transformed controls represented in Figure 4.4 (a) and (b) would be denoted by ${}^{8}\tau_1$ and ${}^{27}\tau_2$ respectively. Since $(8)_{10} = (1000)_2$;

then

$$\left.\begin{aligned} {}^{8}\delta_{\tau_1}(q_4,0) &= \delta_{\tau_1}(q_4,1) = q_5 \\ {}^{8}\delta_{\tau_1}(q_4,1) &= \delta_{\tau_1}(q_4,0) = q_1 \end{aligned}\right\} \text{ since } b_4 = 1$$

and similarly for ${}^{27}\tau_2$, where $(27)_{10} = (11011)_2$.

Theorem 4.2

For any class of controls $\{\tau_i\}$ such that for some

$$\Gamma = \langle S_\Gamma, \Sigma_\Gamma, Y_\Gamma, \delta_\Gamma, \lambda_\Gamma, s_0 \rangle \ , \ |S_\Gamma| = n, \ \Sigma_\Gamma = \{0,1\}$$

where

$$\tau_i = {}^i\Gamma \qquad 0 \leq i \leq 2^n-1$$

then Γ is a common control for τ_i and

$$\tau_i \ R_I \Gamma \qquad 0 \leq i \leq 2^n-1.$$

Proof:

(1) Let I be any interpretation on τ_i. Define ${}^iI$ as that interpretation on Γ which is to be constructed so that

$$V_d\left(\overline{I[\tau_i]}\right) = V_d\left(\overline{{}^iI[\Gamma]}\right) \ ; \text{ for all } d\epsilon D.$$

(2) If

$${}^iI = \langle i_\eta, i_\gamma, i_\zeta, d \rangle$$

and

$$(i)_2 = \left(b_{n-1}, b_{n-2}, \ldots b_1, b_0\right)_2 \ ;$$

$$b_j \epsilon \{0,1\} \ ; \ j = 0, n-1$$

then set

(a) ${}^i\zeta = \zeta$

(b) ${}^i\gamma : S_\Gamma \longrightarrow P^2$

where if

$$(s_j) = p_j^2 \ \varepsilon \ P^2$$

then

$$i_\gamma(s_j) = p_j^2 \quad \text{if } b_j = 0$$

$$i_\gamma(s_j) = \overline{p}_j^2 \quad \text{if } b_j = 1$$

and if

$$p_j^2\ (D) = 0 \text{ then } \overline{p}_j^2\ (D) = 1$$

$$p_j^2\ (D) = 1 \text{ then } \overline{p}_j^2\ (D) = 0.$$

(c)
$$i_\eta(s_j) = (f_j, p_j^2) \text{ iff } i_\zeta\left[\lambda_\Gamma(s_j)\right] = f_j$$

$$\text{and } i_\gamma\ (s_j) = p_j^2.$$

(3) Now for any state, s_j, such that $b_j = 1$ and

$$rp_{s_0}\ (\sigma_1,..,\sigma_\ell) = s_j \ ; \ \sigma_k \varepsilon \{0,1\},\ 1 \leq k \leq \ell$$

let

$$\gamma(s_j) = p_j^2 \text{ and } p_j^2\ (D) = 0.$$

Now suppose

$$\delta_\Gamma(s_j,0) = s_m.$$

Correspondingly

$$\delta_{\tau_i}\ (s_j,0) = i_{\delta_\Gamma}(s_j,1) = s_m$$

but

$$i_{\gamma(s_j)} = \overline{p}_j^2 \ , \quad \overline{p}_j^2 \ (D) = 1$$

hence Γ will transfer to state s_m because the decision predicate associated with state s_j has encoded the same environmental response (to p_j^2) to correspond with the permuted transition function on s_j. Consequently if for some $d \epsilon D$,

$$\zeta \left(\lambda_{\tau_i} \left[rp_{s_0}(\sigma_1, \sigma_2, \ldots \sigma_\ell) \right] \right)$$

$$= {}^{i}\zeta \left(\lambda_{\Gamma} \left[rp_{s_0}(\sigma_1', \ldots \sigma_\ell') \right] \right) = f_\ell$$

and

$$\zeta \left(\lambda_{\tau_i} \left[rp_{s_0}(\sigma_1 \ldots \sigma_\ell, 0) \right] \right) = f_{\ell+1}$$

then

$${}^{i}\zeta \left(\lambda_{\tau_i} \left[rp_{s_0}(\sigma_1' \ldots \sigma_\ell', 1) \right] \right) = f_{\ell+1} .$$

(4) But this is true for any ℓ and corresponding m hence

$$V_d \left(\overline{I[\tau_i]} \right) = V_d \left({}^{i}\overline{I[\Gamma]} \right), \text{ for all } d \epsilon D.$$

▷◁

It is clear that the sets $\tilde{S}_D\left(\overline{I[\tau_i]}\right)$ and $\tilde{S}_D\left(^i\overline{I[\Gamma]}\right)$ should be related in a 1-1 fashion but the relationship in general is fairly complicated. The important fact is that any control $^0\tau$ is R_I related to any reordering $^i\tau$ and, since the converse is also true, they must be $\underset{I}{\equiv}$ equivalent. (see Definitions 2.5 and 2.6).

Theorem 4.3

For any n-state control, τ,

$$\tau \underset{I}{\equiv} {}^i\tau \; ; \; 0 \leq i \leq 2^n-1 .$$

$\rhd\lhd$

The remainder of this chapter will dwell heavily on the implications of reordered controls in the structural realizations previously discussed. Since the construction of interpretations is well specified for a common control realizing a class of reordered controls, attention is directed to the machine theoretic aspects of the reordering operation in hopes of gaining insight into suboptimal minimization procedures.

Underlying the discussion are fundamental concepts utilized in the machine decomposition theories of Zeiger and Krohn-Rhodes. (An excellent treatment of these topics is contained in Arbib [1969].)

4.3 Suboptimal SP Partitions, Semigroup and Composite Realizations

At the outset it must be stressed that while the reordering transformation of a control is a program value preserving operation, in general virtually no structural properties remain invariant. The SP lattice structure and the associated semigroup of the control, τ, changes drastically under certain reorderings. Figure 4.7 shows an n-state control, τ, and the reorderings up to i = 7. For i = 8 to i = 15, the diagrams are just reversed and the SP lattice and semigroups are isomorphic.

Theorem 4.4

For any n-state deterministic FSM, τ, the SP lattice and semigroups of ${}^{i}\tau$ and ${}^{2^n-1-i}\tau$ have the property

(1) for all $\pi_i \in L_{i_\tau}$, there exists a $\pi_{\acute{i}} \in L_{(2^n-1-i)_\tau}$ such that

$$\lfloor \pi_i \cong \lfloor \pi_{\acute{i}} \text{ as machines, and}$$

(2) the semigroup accumulators are isomorphic

$$S_{i_\tau} \cong S_{(2^n-1-i)_\tau}.$$

Proof:

(1) The proof is for the case ${}^{0}\tau$, ${}^{2^n-1}\tau$ since it must then also be true for ${}^{i}\tau$, ${}^{(2^n-1-i)}\tau$.

(2) Since

$$(2^n-1)_2 = (b_{n-1}, b_{n-2},..,b_o)_2$$

$$= (1,1,.....1)_2 \qquad b_j = 1 \; ; \; 0 \leq j \leq n-1$$

then

$${}^{2^n-1}\delta_\tau(Q_\tau,0) = {}^{0}\delta_\tau(Q_\tau,1)$$

$${}^{2^n-1}\delta_\tau(Q_\tau,1) = {}^{0}\delta_\tau(Q_\tau,0)$$

hence

$${}^{2^n-1}\tau \cong {}^{0}\tau$$

and isomorphic controls must have isomorphic semigroups and SP lattices.

▷◁

For an illustration of how reordering applies to SP partition realizations (recall Theorem 3.1) consider the following case . Given a control, τ, such that the semi-control, $\hat{\tau}$, is isomorphic to the SP partition, $\{\overline{13}, \overline{2}, \overline{4}\}$, of ${}^{3}\hat{\Gamma}$ in Figure 4.7, it is clear that no such partition could be uncovered from the trivial lattice, $L_{0\hat{\Gamma}}$. But if all lattices

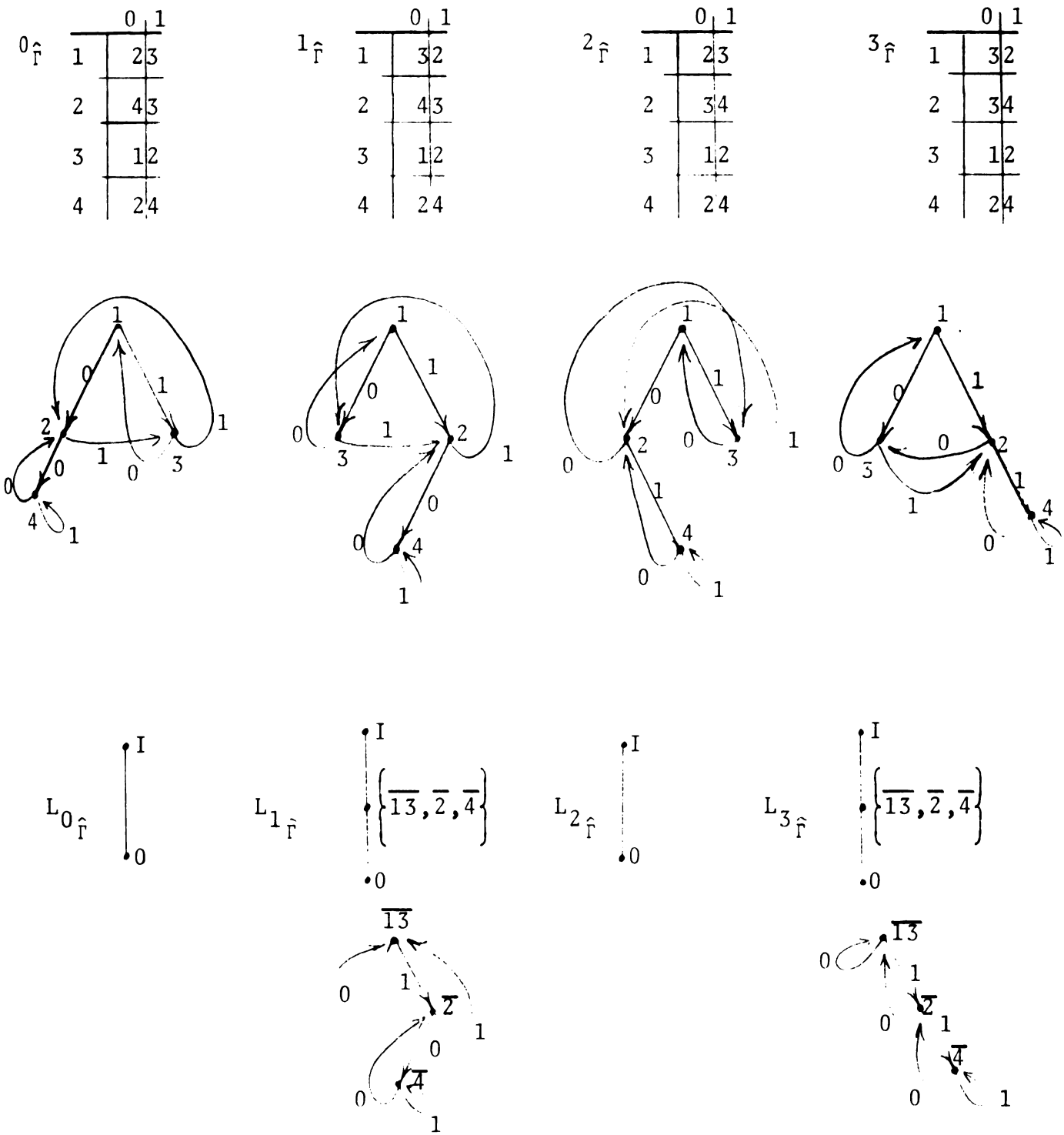


Figure 4.7

$\hat{\Gamma}$ and the Reorderings, ${}^{i}\hat{\Gamma}$, i = 0, 7.

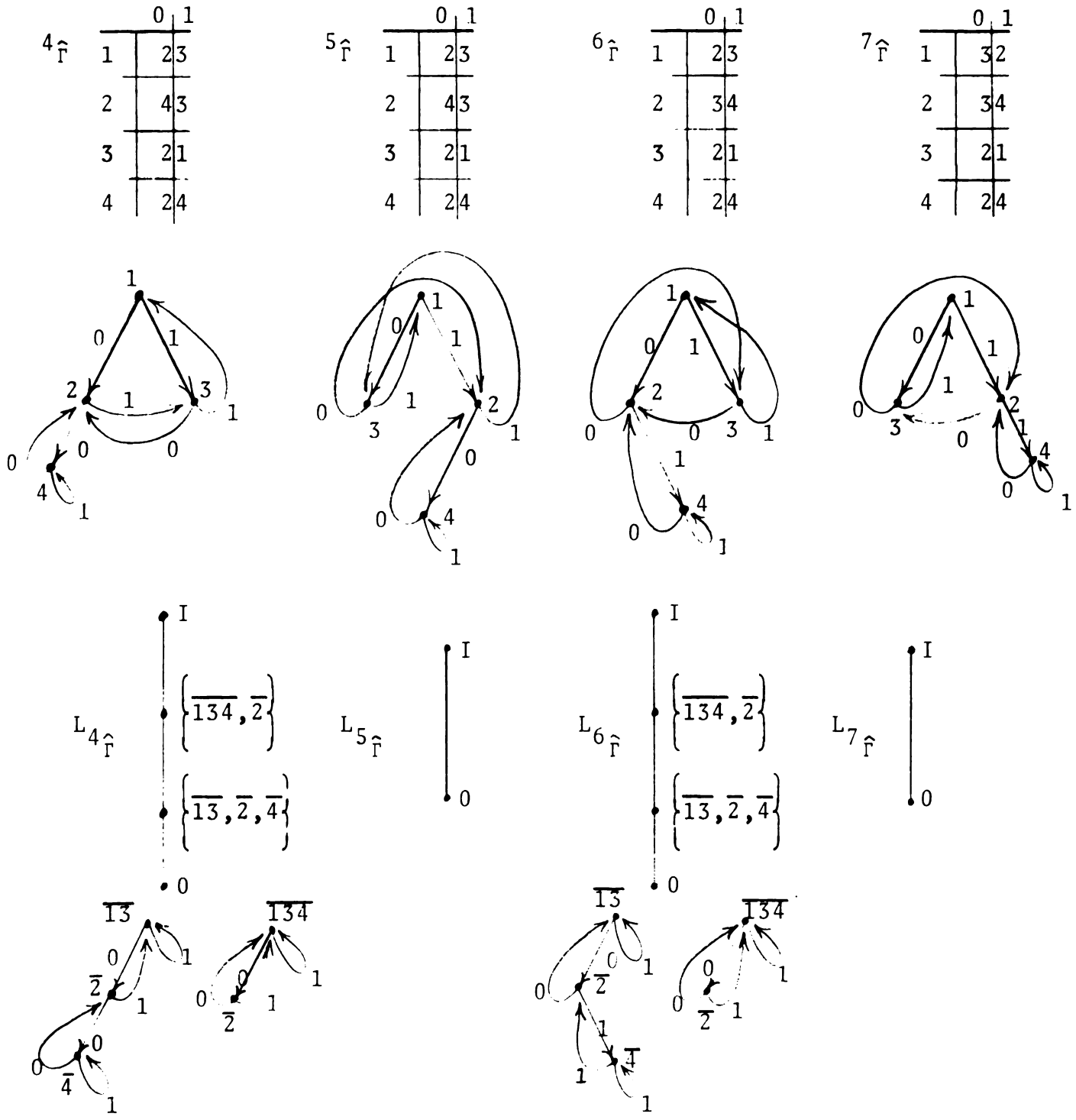


Figure 4.7 (Cont'd.)

$\hat{\Gamma}$ and the Reorderings, ${}^{i}\hat{\Gamma}$, i = 0, 7.

for ${}^{i}\hat{\Gamma}$ were searched, that very machine would be uncovered for the reordering, ${}^{3}\hat{\Gamma}$. Uncovering the lattices requires searching $(2^n-1)/2$ reordered controls because of Theorem 4.4. This procedure is tedious but well defined and the algorithm for SP partition generation only need be carried out to k-block partitions on $\hat{\Gamma}$ when trying to find an isomorphic machine to a k-state control, $\hat{\tau}$. The work can be cut down further if it is realized that when one k-block SP partition is found for some ${}^{i}\hat{\Gamma}$, ${}^{i}\delta_{\hat{\Gamma}}$ can be changed for every state in any of the blocks to generate a new reordering on ${}^{i}\hat{\Gamma}$ which has the same k-block SP partition in the SP lattice but whose machine structure differs from the partition in ${}^{i}\hat{\Gamma}$. For example ${}^{1}\hat{\Gamma}$ and ${}^{3}\hat{\Gamma}$ both contain the SP partition $\{\overline{13},\overline{2},\overline{4}\}$ in their SP lattice and their transition functions differ only on state $\overline{2}$. The same is true for ${}^{1}\hat{\Gamma}$ and ${}^{4}\hat{\Gamma}$, where ${}^{1}\delta_{\hat{\Gamma}} \neq {}^{4}\delta_{\hat{\Gamma}}$ on states $\overline{1}$ and $\overline{3}$. In all cases the associated machine structure for $\{\overline{13},\overline{2},\overline{4}\}$ is different. It is not possible to detect all k-block SP partitions by examining reorderings on just one k-block partition, but all those partitions which group the state set in the same way as the original can be uncovered by a set of reorderings. To illustrate this, note that all reorderings on ${}^{i}\hat{\Gamma}$ identifying the states $\{\overline{13},\overline{2},\overline{4}\}$ can be determined by reordering states 1 and 3,2, or 4 in any combination for the machine ${}^{1}\hat{\Gamma}$. The point is that reorderings on un-

covered SP partitions in $\hat{\Gamma}$ may produce the desired isomorphism to the given $\hat{\tau}$.

The case for semigroup realizations requires as much if not more work. One must determine that reordering on $\hat{\tau}$--for which each member of the set $\{\hat{\tau}_i\}$ is the control $\hat{\tau}$ for a different starting state in Q_τ--which produces the smallest semigroup accumulator, $\hat{\Gamma}$, realizing each semi-control, $\hat{\tau}_i$. If $\hat{\tau}$ has n states, its semigroup could be close to the maximum size, n^n. Consequently, determining the semigroup by ascertaining the closed set of state functions f_x: $Q_{\hat{\tau}} \longrightarrow Q_{\hat{\tau}}$ is not practical for large machines. There are two conditions which appear to be useful here.

It is well known that the right-congruent response relation on any finite machine refines the congruence relation associated with the semigroup of that machine. The power or computibility of the machine might be thought of as the rank of the (semigroup) congruence relation. It might be expected that as the number of SP partitions on an n state machine τ increases, for some reorderings ${}^{i}\tau$, then in some sense the semigroup sizes decrease. The following theorem indicates in what sense this is true for a restricted class of controls.

Let T : the class of all FSM semi-controls with n states over a binary alphabet.

Let $S_{\hat{M}}$: semi-group accumulator for control, $\hat{M} \in T$

$|S_{\hat{M}}|$ = number of states in $S_{\hat{M}}$.

Let $T^k \subseteq T$: that class of machines which have at least one n-k+1 block SP partition with one block having k elements, all other blocks having one element.

Then for all $\hat{M} \in T^k$, there exists a $\pi \in L_{\hat{M}}$ such that

$$\pi = \left\{ \{B_1, B_2, \ldots B_{n-k+1}\} \right.$$
$$\left. \{(\overline{q_{i_1}, q_{i_2}, .., q_{i_k}}), \underbrace{\overline{q}_{i_{k+1}}, \overline{q}_{i_{k+2}}, .., q_{i_n}}_{n-k \text{ blocks}}\} \right\}.$$

Theorem 4.5

$$\max_{\hat{M} \in T^2} |S_{\hat{M}}| \leq (n+2) * n^{n-2} = n^{n-1} + 2*n^{n-2}$$
$$\leq \max_{\hat{M} \in T} |S_{\hat{M}}| = n^n.$$

Proof:

(1) Now

$$S_{\hat{M}} \cong \left(\overset{\times}{i}\hat{M}_i\right)^C \qquad \text{where } \hat{M} = \left\langle Q_{\hat{M}}, \Sigma, \delta_{\hat{M}} \right\rangle$$

$$\text{and } \hat{M}_i = \left\langle Q_{\hat{M}}, \Sigma, \delta_{\hat{M}}, q_i \right\rangle$$

and the state set of S_M is isomorphic to

$$\left\{\left(\overset{n}{\underset{j=1}{\times}} Q_{\hat{M}}\right)\right\}^r = \left\{\left(q_{j_1}, q_{j_2}, \ldots, q_{j_n}\right)\right\}^r$$

where there exists a $x_j \varepsilon \Sigma^* = \{0,1\}^*$ such that for $i \leq j \leq n$,

$$\left(q_{j_1}, q_{j_2}, q_{j_n}\right) = \left(\delta_{\hat{M}}(q_1, x_j) \cdot\cdot\cdot\cdot\cdot \delta_{\hat{M}}(q_n, x_j)\right).$$

(2) But the maximum number of different n-tuples,

$\left(q_{j_1}, \ldots, q_{j_n}\right)$ is clearly $n \cdot n \cdot n \cdot\cdot\cdot\cdot n = n^n$

hence $|S_{\hat{M}}| \leq n^n$, as is known.

(3) Now suppose $\hat{M}$ has only <u>one</u> n-1 block SP partition denoted

$$\pi_{ij} = \left\{\overline{q_i q_j}, \overline{q_1}, \overline{q_2}, \ldots, \overline{q_{i-1}}, \overline{q_{i+1}}, \ldots, \overline{q_{j-1}}\,\overline{q_{j+1}}, \ldots, \overline{q_n}\right\}$$

and suppose

$$\delta_{\hat{M}}(q_i,0) = q_{i0} \ , \ \delta_{\hat{M}}(q_i,1) = q_{i1}$$

$$\delta_{\hat{M}}(q_j,0) = q_{j0} \ , \ \delta_{\hat{M}}(q_j,1) = q_{j1}$$

for the existence of π_{ij} only two cases are possible.

Case 1

If $q_{i0} \neq q_{j0}$, then $q_{i0} \in \{q_i,q_j\}$

and $q_{j0} \in \{q_i,q_j\}$.

Similarly if $q_{i1} \neq q_{j2}$, then $q_{i1} \in \{q_i,q_j\}$

and $q_{j1} \in \{q_i,q_j\}$.

Case 2

If $q_{i0} = q_{j0}$, then q_i may be arbitrary and similarly for q_{i1} and q_{j1}.

(4) In both cases if

$$\delta_{\hat{M}}(q_i,\sigma) \in (q_i,q_j)$$

$$\delta_{\hat{M}}(q_j,\sigma) \in (q_i,q_j) \quad \text{for } \sigma \in \{0,1\}$$

then the maximum number of different ordered pairs reachable from (q_i,q_j) is four, viz.

$$\left\{(q_i,q_j),(q_i,q_i),(q_j,q_j),(q_j,q_i)\right\}.$$

(5) For $q_{i0} = q_{j0} \triangleq q^0$

and $q_{i1} = q_{j1} \triangleq q^1$

define

$$r(q^0) \triangleq \left\{q_k \,\middle|\, \delta(q^0,x) = q_k \in Q_{\hat{M}} \; ; \; x \in \Sigma^*\right\}$$

and similarly for $r(q^1)$. The maximum number of different ordered pairs of states reachable from (q_i,q_j) is

$$\max \left| \; r(q^0) \cup r(q^1) \; \right| \leq n$$

since all pairs must be of the form (q_k,q_k).

(6) Now for $q_{i0} = q_{j0} \triangleq q^0 \in Q_{\hat{M}}$

and $q_{i1} \neq q_{j1}$

then $q_{i1} \in \left\{q_i,q_j\right\}$, $q_{j1} \in \left\{q_i,q_j\right\}$.

The reverse can also hold and still the maximum number of different ordered pairs of states reachable from (q_i,q_j) is

$$\max \left| r(q^0) \right| + \left|\{(q_i,q_j),\ (q_j,q_i)\}\right| \leq n+2 .$$

(7) Finally, since n+2 is the maximum number of reachable states from the pair (q_i,q_j), then the reachable states from the n-tuple, $(q_1,q_2,..,q_i,..,q_j,..,q_n)$, must be less than or equal to $(n+2)\cdot\underbrace{n\cdot n\cdot\cdots\cdot n}_{n-2}$.

(8) Thus for $\hat{M}$ with no n-1 block SP partitions

$$\max_{\hat{M}\in T} |S_{\hat{M}}| \leq n^n .$$

But if $\hat{M}$ has at least one n-1 block SP partition, then

$$\max_{\hat{M}\in T^2} |S_{\hat{M}}| \leq (n+2)\cdot n^{n-2} = n^{n-1}+2*n^{n-2}$$

$$< n^n .$$

▷◁

The argument can be generalized in the following theorem.

Theorem 4.6

$$\max_{\hat{M}\epsilon T^k} |S_{\hat{M}}| \leq (n+k^k-k)\cdot n^{n-k}$$

$$= (n-k)\cdot n^{n-k+k^k}\cdot n^{n-k} .$$

Proof:

(1) Suppose $\hat{M}$ has one n-k+1 block SP partition with one k element block (as in Theorem 4.5 for k=2). Then let

$$B_k = \left(\overline{q_{i_1}, q_{i_2}, \ldots, q_{i_k}}\right) .$$

Following Theorem 4.5, the worst case is

$$\delta_{\hat{M}}\left(q_{i_j}, \sigma\right) \notin B_k \text{ for all } j=1,k \; ; \; \sigma \in \{0,1\}$$

$$\delta_{\hat{M}}\left(q_{i_j}, \sigma'\right) \in B_k \text{ for all } j=1,k \; ; \; \sigma' \neq \sigma .$$

Then the maximum number of reachable states from the set B_k must be less than or equal to n, plus all arrangements on B_k, k^k, less the k sets previously counted (each element of the k-tuple is the same), i.e. $n+k^k-k$.

(2) For $\hat{M}$ with one such k-block SP partition

$$\max_{\hat{M}\in T^k} |S_{\hat{M}}| \leq (n+k^k-k)\cdot \underbrace{n\cdot n\cdot\cdots\cdot n}_{n-k}$$

$$= (n+k^k-k)\cdot n^{n-k}$$

$$= (n-k)\cdot n^{n-k} - k^k\cdot n^{n-k} < n^k.$$

▷◁

As an example of applying analytical techniques to some advantage, consider this bound as a function of k for fixed n.

Let $y = (n+k^k-k)\cdot n^{n-k}$

then

$$\ln(y) = (n-k)\ln(n) + \ln(n+k^k-k).$$

Now

$$\frac{d}{dk}\ln(y) = \frac{d}{dy}\ln(y)\cdot\frac{dy}{dk} = \frac{1}{y}\frac{dy}{dk}$$

$$= -\ln(n) + \left(\frac{1}{n+k^k-k}\right)\left[\frac{d}{dk}(n+k^k-k)\right].$$

Since

$$\frac{dk^k}{dk} = k^k\left(1+\ln(k)\right)$$

then

$$\frac{1}{y}\frac{dy}{dk} = -\ln(n) + \frac{k^k\left(1+\ln(k)\right)-1}{(n+k^k-k)}$$

$$\frac{dy}{dk} = (n+k^k-k)\cdot n^{n-k}\cdot\left[-\ln(n)+\frac{k^k\left[1+\ln(k)\right]-1}{(n+k^k-k)}\right]$$

and

$$\frac{dy}{dk} = 0 \quad \text{when} \quad \frac{k^k\left[1+\ln(k)\right]-1}{n+k^k-k} = \ln(n).$$

This indicates that for smaller k (larger n-k+1 block partitions) the maximum size of the semigroup tends to be reduced for fixed n.

While these bounds on $S_{\hat{M}}$ are very crude, they do indicate some things which should be taken into account when searching for minimal semigroups over a class of reordered controls. As a final comment on suboptimal semigroup realizations, a particularly convenient, though not necessarily minimal, form of transition function to look for in reordering controls is a permutation-reset (PR) machine.

Theorem 4.7

For any deterministic FSM control τ, if for any reordering i_τ,

$$\left.\begin{array}{l} i_{\delta_\tau}(Q,\sigma) = Q \\ \text{or} \\ i_{\delta_\tau}(Q,\sigma) = q, \text{ some } q\in Q \end{array}\right\} \text{ for all } \sigma\in\Sigma_\tau$$

Then the semigroup accumulator has the following bound on its states

$$|S_{i_\tau}| \leq n!+n.$$

Proof:

If every input is a permutation input or a reset input, the semigroup of state functions on Q_{i_τ} can be no more than the maximum number of permutations on Q_τ, n!, plus the number of all constant states, n. ▷◁

This approach is also useful when applied to a subset of states in Q. Since it is easy to examine the transition function over all Σ_τ to identify all those states which transfer to at least one state in common under any input, it is a simple matter to reorder τ so those states transfer to one of those common states under the same input. Whatever other transitions can be applied to reduce the size of other permutations or determine other reset inputs can further reduce the resultant semigroup. Sometimes this process results in a combinatorial semigroup where there are no simple groups which are subgroups of the semigroup. As an example of this, examine the transition tables of τ and its semigroup accumulator given in Figure 4.8 (a) and (b).

δ_τ:

	0	1
1	2	3
2	2	3
3	1	3

(a)

δ_{S_τ}:

	s_0	0 s_1	1 s_2	00 s_3	10 s_4
$,s_0$	s_0	s_1	s_2	s_3	s_4
$0,s_1$	s_1	s_3	s_2	s_3	s_4
$1,s_2$	s_2	s_4	s_2	s_3	s_4
$00,s_3$	s_3	s_1	s_2	s_3	s_4
$10,s_4$	s_4	s_3	s_3	s_3	s_4

(b)

Figure 4.8

FSM Control and its Combinatorial Semigroup

The semigroup accumulator is the control S_τ with states s_i, i=0,4 and input alphabet {0,1}. S_τ can, of course, serve as a common control for τ in any of its three possible starting states. By examining the state functions defined by the input tapes, {0,1,00,10}, it is easily seen that no subset of states is permuted by any tape. When this occurs there are no simple groups dividing the semigroup of τ and it is combinatorial. Contrast this with the control ${}^1\tau$ formed by reordering state 1 in τ above. Then S_{1_τ} has eleven states. For ${}^2\tau$, S_{2_τ} has

twenty-five states. But neither of these two reordered machines has a reset input--indicating that a good heuristic is to look for reorderings which uncover the largest number of reset inputs.

The reordering operation may also be used to reduce the input dependency of a composite control realization. For two controls τ_1, τ_2 and $|\Sigma_{\tau_1}| = |\Sigma_{\tau_2}| = m$, it is generally possible to reorder τ_1 and τ_2 so that a composite realization Γ exists for $^{r}\tau_1$, $^{s}\tau_2$ such that for some $s_k \varepsilon S_\Gamma$:

$$\delta_\Gamma(s_k,\sigma_{1j}) = \delta_\Gamma(s_k,\sigma_{2j}) \; ; \; 1 \leq j \leq m \; ; \sigma_{1j},\sigma_{2j} \; \varepsilon \; \Sigma_\Gamma .$$

Then the transition function δ_Γ for s_k need be defined for only m inputs instead of 2m inputs generally required in the construction. The resultant common control Γ appears incompletely defined but in the implementation of Γ the m-ary branching logic at state s_k precludes the necessity to change the decision predicates mapped onto that state by any interpretations, $\left\{I_i^H\right\}$. Unless the number of different controls and/or the individual input alphabets are large, the effort in searching through all reorderings may not significantly decrease computation or run times. But it is a useful device to keep in mind when keeping the number of different external procedures--identified by each

I_i^H on Γ--to a minimum.

4.4 Suboptimal Direct Product Realizations

As the example earlier in this chapter clearly indicates, to keep the number of states required for a direct product construction of a common control within a reasonable bound, the reordering operation becomes an extremely valuable tool. In this section some conditions are given which have proved useful in finding smaller direct products. The concept of an SP cover will be needed.

Definition 4.2

Given a deterministic FSM control, τ, an SP cover, Θ, on the state set Q_τ

$$\Theta = \left\{B_1, B_2, \ldots B_k\right\} \quad ; \; B_i \subseteq Q_\tau \,, \; i=1,k$$

satisfies the substitution property where for every $B_i \varepsilon \Theta$ and $\sigma \varepsilon \Sigma_\tau$, there exists a $B_j \varepsilon \Theta$ such that

$$\delta_\tau (B_i, \sigma) \subseteq B_j \,.$$

For exposition much of the following discussion will be limited to cyclic direct products of two semi-controls. Extensions to the general case can be made directly but only with a considerable investment in nctation.

Definition 4.3

Given two semi-controls $\hat{\tau}_1$, $\hat{\tau}_2$

where

$$\Sigma_{\hat{\tau}_1} = \Sigma_{\hat{\tau}_2} = \Sigma$$

$$\left|Q_{\hat{\tau}_1}\right| = \left|\left\{q_{11}, q_{12}, \ldots q_{1m}\right\}\right| = m$$

$$\left|Q_{\hat{\tau}_2}\right| = \left|\left\{q_{21}, q_{22}, \ldots q_{2n}\right\}\right| = n .$$

Define the set of R-minimal direct product controls

$$R\left[\hat{\tau}_1 \hat{\tau}_2\right] = \left\{r_{\hat{\tau}_1 \times s_{\hat{\tau}_2}}\right\} = \left\{\left\langle Q_{r_{\hat{\tau}_1 \times s_{\hat{\tau}_2}}}, \delta_{r_{\hat{\tau}_1 \times s_{\hat{\tau}_2}}}, \Sigma \right\rangle\right\}$$

where

$$\left|Q_{r_{\hat{\tau}_1 \times s_{\hat{\tau}_2}}}\right| \leq \left|Q_{i_{\hat{\tau}_1 \times j_{\hat{\tau}_2}}}\right| ; \quad 0 \leq i \leq 2^m - 1 \quad 0 \leq j \leq 2^n - 1 .$$

Definition 4.4

For the direct product of two semi-controls given as in Definition 4.3, define the $\hat{\tau}_2$ induced cover on $\hat{\tau}_1$ as a set cover on $\hat{\tau}_1$ induced by projecting $\hat{\tau}_1 \times \hat{\tau}_2$ onto $\hat{\tau}_2$. Then

$$\Theta_{\hat{\tau}_1} = \left\{B_1, B_2, \ldots B_n\right\} \; ; \; B_i \subseteq Q_{\hat{\tau}_1} \; ; \; 1 \leq i \leq n$$

where if

$$B_i = \left\{q_{1i_1}, q_{1i_2}, \ldots q_{1i_k}\right\} \; ; \; q_{1i_j} \in Q_{\hat{\tau}_1} \; ; \; 1 \leq j \leq k$$

then

$$\left(q_{1i_j}, q_{2i}\right) \in Q_{\hat{\tau}_1 \times \hat{\tau}_2} \quad ; \quad 1 \leq j \leq k.$$

Note that as a control, $\Theta_{\hat{\tau}_1}$, is isomorphic to $\hat{\tau}_2$ since it is the projection of the direct product machine onto $\hat{\tau}_2$.

Theorem 4.8

The induced covers, $\Theta_{\hat{\tau}_1}$ and $\theta_{\hat{\tau}_2}$, are SP covers.

Proof: (for $\Theta_{\hat{\tau}_1}$)

(1) Suppose

$$\delta_{\hat{\tau}_2}(q_{2i}, \sigma) = q_{2\ell} \quad ; \quad \sigma \in \Sigma \; ; \; q_{2i}, q_{2\ell} \in Q_{\hat{\tau}_2}$$

and for all $q_{1i_j} \in Q_{\hat{\tau}_1}$ such that

$$\left(q_{1i_j}, q_{2i}\right) \in Q_{\hat{\tau}_1 \times \hat{\tau}_2}$$

then

$$\delta_{\hat{\tau}_1 \times \hat{\tau}_2}\left[\left(q_{1i_j}, q_{2i}\right), \sigma\right] = \left(\delta_{\hat{\tau}_1}(q_{1i_j}, \sigma), q_{2\ell}\right).$$

(2) But B_i is exactly that set of q_{1i_j} in (1)

$$B_i = \left\{q_{1i_j}\right\} \; ; \; j=1, \; |B_i|$$

and if

$$B_\ell = \left\{q_{1\ell_j}\right\} \; ; \; j=1, \; |B_\ell|$$

then

$$\delta_{\hat{\tau}_1}(B_i, \sigma) = \left\{\delta_{\hat{\tau}_1}(q_{1i_j}, \sigma)\right\}_{j=1, |B_i|} \subseteq B_\ell .$$

(3) Since this is true for all i, $\Theta_{\hat{\tau}_1}$ has the substitution property. The proof is similar for $Q_{\hat{\tau}_2}$.

$\triangleright \triangleleft$

The useful feature about $\Theta_{\hat{\tau}_i}$ covers is that they provide a convenient means to describe $R\left[\hat{\tau}_1 \hat{\tau}_2\right]$.

Theorem 4.9

The direct product control for the reordered semi-controls $\hat{\tau}_1$, $\hat{\tau}_2$ (as above)

$$r_{\hat{\tau}_1} \times s_{\hat{\tau}_2} \in R\left[\hat{\tau}_1 \hat{\tau}_2\right]$$

is R-minimal iff

$$\left\| \Theta_{r_{\hat{\tau}_1}} \right\| = \left\| \left\{ B_1, B_2, \ldots B_n \right\} \right\| = \sum_{i=1}^{n} \left| B_i \right| \leq \left\| \Theta_{j_{\hat{\tau}_1}} \right\|$$

where $\Theta_{j_{\hat{\tau}_1}}$ is the $k_{\hat{\tau}_2}$ induced cover on $j_{\hat{\tau}_1}$ for any direct product, $k_{\hat{\tau}_1} \times j_{\hat{\tau}_2}$; $0 \leq k \leq 2^m - 1$, and $0 \leq j \leq 2^n - 1$.

Proof:

Clear when it is realized that $\left\| \Theta_{r_{\hat{\tau}_1}} \right\| = \left| Q_{r_{\hat{\tau}_1} \times s_{\hat{\tau}_2}} \right|$.

▷◁

It is apparent that the smallest direct product attainable for any two semi-controls can be achieved when one control is a homomorphic image of the other and thus must be isomorphic to some SP partition in its lattice. This fact is also readily expressible in terms of induced covers.

Theorem 4.10

Let $\hat{\tau}_1$, $\hat{\tau}_2$ be cyclic, deterministic, FSM semi-controls with $|Q_{\hat{\tau}_1}| = m$ and $|Q_{\hat{\tau}_2}| = n$. If $m \leq n$, then $s_{\hat{\tau}_2}$ is a homomorphic image of $r_{\hat{\tau}_1}$ and

$$r_{\hat{\tau}_1} \times s_{\hat{\tau}_2} \in R\left[\tau_1 \tau_2\right]$$

iff

$$\left\| \Theta_{r_{\hat{\tau}_1}} \right\| = m .$$

Proof:

(1) If

$$\left\| \Theta_{r_{\tau_1}} \right\| = \left\| \left\{ B_1, B_2, \ldots B_n \right\} \right\| = \sum_{i=1}^{n} |B_i| = m$$

then since τ_1 is cyclic

$$B_i \cap B_j = \phi \text{ (null)} .$$

(2) But then $\Theta_{r_{\hat{\tau}_1}}$ can be made isomorphic to some SP partition in $r_{\hat{\tau}_1}$ and since

$$\Theta_{r_{\hat{\tau}_1}} \tilde{=} s_{\hat{\tau}_2}$$

then for some $\pi \in L_{r_{\hat{\tau}_1}}$ (SP lattice of $r_{\hat{\tau}_1}$)

$$s_{\hat{\tau}_2} \tilde{=} \lfloor \pi .$$

(3) The argument is reversible to show necessity.

$\triangleright\triangleleft$

Unfortunately there is no assurance that any reorderings exist for $\hat{\tau}_1$ and $\hat{\tau}_2$ such that $\hat{\tau}_2$ is a homomorphic image of $\hat{\tau}_1$. A complete search is generally required. There is a weaker condition on the lattice structures which can sometimes be verified more easily.

Theorem 4.11

Given $\hat{\tau}_1$, $\hat{\tau}_2$ as before, if there exists a π_1, π_2 such that

$$\pi_1 \in L_{r_{\hat{\tau}_1}} \; ; \; 0 \leq r \leq 2^m-1 \; ; \; \pi_1 = \left\{a_1, a_2, \ldots a_k\right\} \; ; \; |a_i| = b_i$$

$$\pi_2 \in L_{s_{\hat{\tau}_2}} \; ; \; 0 \leq s \leq 2^n-1 \; ; \; \pi_2 = \left\{c_1, c_2, \ldots c_k\right\} \; ; \; |c_i| = d_i$$

and if $\pi_1 \tilde{=} \pi_2$ as controls under the transition functions $r_{\delta_{\hat{\tau}_1}}$, $s_{\delta_{\hat{\tau}_2}}$ where $a_i \leftrightarrow c_i$; i=1,k then

$$\left| Q_{r_{\hat{\tau}_1} \times s_{\hat{\tau}_2}} \right| \leq \sum_{i=1}^{n} b_i * d_i < m * n .$$

Proof:

(1) The lattice structure of $r_{\hat{\tau}_1} \times s_{\hat{\tau}_2}$ contains both $L_{r_{\hat{\tau}_1}}$ and $L_{s_{\hat{\tau}_2}}$ as sublattices. Hence it must contain a SP partition isomorphic to both π_1 and π_2. The (synchronized) projections of the direct product onto that SP partition structure identifies i=1,k disjoint blocks with no more than $b_i * d_i$ states in each.

(2) Then

$$\sum_{i=1}^{k} b_i = m \quad ; \quad \sum_{i=1}^{k} d_i = n$$

and

$$\sum_{i=1}^{k} b_i * d_i = \sum_{i=1}^{k-1} b_i * d_i + (m - \sum_{i=1}^{k-1} b_i)(n - \sum_{i=1}^{k-1} d_i)$$

$$= \sum_{i=1}^{k-1} b_i * d_i - m * \sum_{i=1}^{k-1} d_i - n * \sum_{i=1}^{k-1} b_i$$

$$+ \sum_{i=1}^{k-1} b_i * \sum_{i=1}^{k-1} d_i + m * n$$

or equivalently

$$\sum_{i=1}^{k} b_i*d_i = -m*\sum_{i=1}^{k-1} d_i - n*\sum_{i=1}^{k-1} b_i + 2*\left[\left(\sum_{i=1}^{k-1} b_i\right)\left(\sum_{i=1}^{k-1} d_i\right)\right]$$

$$- \left[\sum_{i=1}^{k-1} b_i * \sum_{i=1}^{k-1} d_i - \sum_{i=1}^{k-1} b_i*d_i\right] + m*n$$

$$= m*n - \sum_{i=1}^{k-1} d_i*\left(m - \sum_{i=1}^{k-1} b_i\right) - \sum_{i=1}^{k-1} b_i*\left(n - \sum_{i=1}^{k-1} d_i\right)$$

$$- \left[\sum_{i=1}^{k-1} b_i * \sum_{i=1}^{k-1} d_i - \sum_{i=1}^{k-1} b_i*d_i\right]$$

$$< m*n.$$

▷◁

Remark: Now the maximum value for $\sum_{i=1}^{k} b_i*d_i$ is achieved when

$$k=2 \; ; \; d_1=n-1 \; ; \; b_1=m-1$$

because

$$\sum_{i=1}^{k} b_i*d_i = m*n-b_1*(n-d_1)-d_1*(m-b_1)$$

$$= m*n-(n_1-1)-(n_2-1)$$

$$= m*n-(m-1)-(n-1).$$

On the other hand, if

$$k=n \; ; \; d_i=1 \quad \text{for} \quad 1 \leq i \leq n$$

then

$$\sum_{i=1}^{k} b_i*d_i = \sum_{i=1}^{k-1} b_i*d_i+(m-\sum_{i=1}^{k-1} b_i)(n-\sum_{i=1}^{k-1} d_i)$$

$$= \sum_{i=1}^{k-1} b_i*1+(m-\sum_{i=1}^{k-1} b_i)*(1)$$

$$= \sum_{i=1}^{k} b_i = m$$

as expected.

These are the upper and lower bounds for any two controls with common structures in their lattices. Unfortunately it is not necessarily true that $\sum_{i=1}^{k} b_i*d_i$ and the number of states in the direct product decrease as k increases between these ranges.

But it is sometimes the case that for any two controls there are reorderings $r_{\hat{\tau}1}$, $s_{\hat{\tau}2}$ such that for all $\sigma_i \neq \sigma_j$; $\sigma_i, \sigma_j \in \Sigma = \Sigma_{\hat{\tau}1} = \Sigma_{\hat{\tau}2}$,

$$r_{\delta_{\hat{\tau}1}}(Q_{\hat{\tau}1},\sigma_i) \cap r_{\delta_{\hat{\tau}1}}(Q_{\hat{\tau}1},\sigma_j) = \phi \text{ (null)}$$

and similarly

$$^{s}\delta_{\hat{\tau}_2}(Q_{\tau_2},\sigma_i) \cap {}^{s}\delta_{\hat{\tau}_2}(Q_{\tau_2},\sigma_j) = \phi \text{ (null)}$$

If this is the case then there is at least one SP partition structure common to both $L_{r_{\hat{\tau}_1}}$ and $L_{s_{\hat{\tau}_2}}$. So in the direct product, $^{r_{\hat{\tau}_1}}\times^{s_{\hat{\tau}_2}}$, if this is true for $|\Sigma|=2$, then k=2 and the direct product must have less than m*n-(m-1)-(n-1) states.

The upper bound on any $^{r_{\hat{\tau}_1}}\times^{s_{\hat{\tau}_2}} \in R\left[\hat{\tau}_1\hat{\tau}_2\right]$ can be tightened further for a number of particular configurations of the state transition tables. One case is presented in the concluding theorem as being rather typical and might indicate an approach to further investigation and classification of these special situations.

Consider the SP cover generated by the first step in the Zeiger decomposition of any semi-control, $\hat{\tau}$. Certainly the set

$$\left\{\delta_{\hat{\tau}}(Q_{\hat{\tau}},\sigma_1),\ \delta_{\hat{\tau}}(Q_{\hat{\tau}},\sigma_2)\right\} ;\quad |\Sigma_{\hat{\tau}}| = 2$$

is a SP cover on Q. For two such semi-controls over a common input alphabet, $\Sigma = \left\{\sigma_1,\sigma_2\right\}$,

if

$$r_{\hat{\tau}_1} \times s_{\hat{\tau}_2} \in R\left[\hat{\tau}_1 \hat{\tau}_2\right]$$

then

$$\left| Q_{r_{\hat{\tau}_1} \times s_{\hat{\tau}_2}} \right| \leq \left| r_{\delta_{\hat{\tau}_1}(Q_{\hat{\tau}_1}, \sigma_1)} \right| * \left| s_{\delta_{\hat{\tau}_2}(Q_{\hat{\tau}_2}, \sigma_1)} \right|$$

$$+ \left| r_{\delta_{\hat{\tau}_1}(Q_{\hat{\tau}_1}, \sigma_2)} \right| * \left| s_{\delta_{\hat{\tau}_2}(Q_{\hat{\tau}_2}, \sigma_2)} \right| .$$

The bound on states of controls in $R\left[\tau_1, \tau_2\right]$ can sometimes be tightened if the cardinal number of the set intersection of the elements of each of the covers

$$\left\{ \delta_{\hat{\tau}_1}(Q_{\hat{\tau}_1}, \sigma_1),\ \delta_{\hat{\tau}_1}(Q_{\hat{\tau}_1}, \sigma_2) \right\}$$

and

$$\left\{ \delta_{\hat{\tau}_2}(Q_{\hat{\tau}_2}, \sigma_1),\ \delta_{\hat{\tau}_2}(Q_{\hat{\tau}_2}, \sigma_2) \right\}$$

is reduced by reordering $\hat{\tau}_1$ and $\hat{\tau}_2$. That is, certain state transitions may be reversed to produce maximally disjoint SP covers of the above type. The following theorem presents a method--for a particular form of transition table--to accomplish this.

Theorem 4.12

If for the semi-control, $\hat{\tau}$, the following properties hold

1) $\delta_{\hat{\tau}}(Q,\sigma_1) = Q$; $|Q| = n$

$\delta_{\hat{\tau}}(Q,\sigma_2) = Q$

i.e. σ_1,σ_2 are permutation inputs

2) $\delta_{\hat{\tau}}(q,\sigma_1) \neq \delta_{\hat{\tau}}(q,\sigma_2)$; for all $q \in Q$

then there exists a reordering, r, on $\hat{\tau}$ such that

$$\left| r_{\delta_{\hat{\tau}}(Q_{\hat{\tau}},\sigma_i)} \right|_{i=1,2} = \frac{n}{2} \quad ; \quad n \text{ even}$$

$$\left| r_{\delta_{\hat{\tau}}(Q_{\hat{\tau}},\sigma_i)} \right|_{i=1,2} = \left[\frac{n}{2}\right] + 1 \quad ; \quad n \text{ odd}$$

where $\left[\frac{n}{2}\right]$ is the maximum integer $\leq \frac{n}{2}$.

Proof:

A method is described herein by which a set of states will be successively chosen for reordering thereby resulting in a transition table which has the desired properties.

(1) Choose those pairs of states with the following property:

Let

$$\delta_{\hat{\tau}}(q,\Sigma) = \left\{\delta_{\hat{\tau}}(q,\sigma_1),\delta_{\hat{\tau}}(q,\sigma_2)\right\}$$

then

$$A_1 = \left\{\left\{q_i,q_j\right\} \;\middle|\; \delta_{\hat{\tau}}(q_i,\Sigma) = \delta_{\hat{\tau}}(q_j,\Sigma)\right\}.$$

For every such pair (q_i,q_j) in A_1, if their transition table entries are not identical under each input then reorder one state of the pair so that

$$r_1\,\delta_{\hat{\tau}}(q_i,\sigma_1) = r_1\,\delta_{\hat{\tau}}(q_j,\sigma_1)$$

and

$$r_1\,\delta_{\hat{\tau}}(q_i,\sigma_2) = r_1\,\delta_{\hat{\tau}}(q_j,\sigma_2).$$

The encoding of which states were reordered, r_1, follows the procedure used earlier. If two states q_2,q_6 out of a set of eight were reordered,

then

$$(34)_{10} = (r_1)_{10} = (00100010)_2.$$

Now condition (2) of the hypothesis says that every state has exactly one predecessor state for both inputs σ_1,σ_2. Consequently,

$$\left|{}^{r_1}\delta_{\hat{\tau}}(A_1,\sigma_1)\right| = \frac{|A_1|}{2} = \left|{}^{r_1}\delta_{\hat{\tau}}(A_1,\sigma_2)\right|$$

Moreover

$$\left|{}^{r_1}\delta_{\hat{\tau}}(Q\text{-}A_1,\sigma_1)\right| = \left|\delta_{\hat{\tau}}(Q\text{-}A_1,\sigma_2)\right| = \left|Q\text{-}A_1\right|$$

and it is clear that only states from the set $Q\text{-}A_1$ need be considered further for possible reordering.

(2) From the set $Q\text{-}A_1$ pick two states q_i,q_j such that either

(a) $${}^{r_1}\delta_{\hat{\tau}}(q_i,\sigma_1) = {}^{r_1}\delta_{\hat{\tau}}(q_j,\sigma_1)$$

or

(b) $${}^{r_1}\delta_{\hat{\tau}}(q_i,\sigma_2) = {}^{r_1}\delta_{\hat{\tau}}(q_j,\sigma_2)$$

or

(c) $${}^{r_1}\delta_{\hat{\tau}}(q_i,\sigma_1) = {}^{r_1}\delta_{\hat{\tau}}(q_j,\sigma_2).$$

(Note that ${}^{r_1}\delta(\cdot,\Sigma) = {}^{r_1}\delta(\cdot,\Sigma)$ on $Q\text{-}A_1$.) In case (a) no reordering is necessary. For case (b) reorder q_i and q_j so that

$$r_{2}\delta_{\hat{\tau}}(q_i,\sigma_1) = r_{2}\delta_{\hat{\tau}}(q_j,\sigma_1)$$

$$= r_{1}\delta_{\hat{\tau}}(q_i,\sigma_2) = r_{1}\delta_{\hat{\tau}}(q_j,\sigma_2).$$

In case (c) reorder q_i or q_j so that

$$r_{2}\delta_{\hat{\tau}}(q_i,\sigma_1) = r_{2}\delta_{\hat{\tau}}(q_j,\sigma_1).$$

Here $r_2 = r_1$ + (encoding of reordered q_i and/or q_j)

Now set

$$A_2 = A_1 \cup \{q_i, q_j\}$$

then

$$\left| r_{2}\delta_{\hat{\tau}}(A_2,\sigma_1) \right| = \frac{|A_2|}{2}$$

$$\left| r_{2}\delta_{\hat{\tau}}(A_2,\sigma_2) \right| = \frac{|A_2|}{2} + 1$$

(3) Pick another state $q_k \in Q\text{-}A_2$ such that for some $q_i \in A_2\text{-}A_1$ either

(a) $r_{2}\delta_{\hat{\tau}}(q_i,\sigma_2) = r_{2}\delta_{\hat{\tau}}(q_k,\sigma_2)$

or

(b) $r_{2}\delta_{\hat{\tau}}(q_i,\sigma_2) = r_{2}\delta_{\hat{\tau}}(q_k,\sigma_1)$.

In case (a) no reordering is required and $r_3 = r_2$. For case (b) reorder q_k so that

$$r_{3_{\delta_{\hat{\tau}}}}(q_i,\sigma_2) = r_{3_{\delta_{\hat{\tau}}}}(q_k,\sigma_2) = r_{2_{\delta_{\tau}}}(q_k,\sigma_1)$$

where

$$r_3 = r_2 + (\text{encoding of reordered } q_k)$$

Now set

$$A_3 = A_2 \cup \{q_k\}$$

At this point A_3 is necessarily of odd cardinality and

$$\left| r_{3_{\delta_{\hat{\tau}}}}(A_3,\sigma_1) \right| = \left[\frac{|A_3|}{2} \right] + 1$$

$$\left| r_{3_{\delta_{\hat{\tau}}}}(A_3,\sigma_2) \right| = \left[\frac{|A_3|}{2} \right] + 1$$

Note that although $|A_3| = |A_2| + 1$, the cardinality of $r_{3_{\delta_{\hat{\tau}}}}(A_3,\sigma_2)$ does not increase. But that of $r_{3_{\delta_{\hat{\tau}}}}(A_3,\sigma_1)$ must increase since

$$r_{3_{\delta_{\hat{\tau}}}}(q_k,\sigma_1) \notin r_{3_{\delta_{\hat{\tau}}}}(A_2,\sigma_1)$$

because of condition (2) in the hypothesis

and because $^{r_3}\delta_{\hat{\tau}}(A_2,\sigma_1)$ consists of pairs of states. Also, at this point, both $^{r_3}\delta_{\hat{\tau}}(A_3,\sigma_1)$ and $^{r_3}\delta_{\hat{\tau}}(A_3,\sigma_2)$ have exactly one unpaired state.

(4) Select another state $q_m \in Q\text{-}A_3$ such that for $q_i \in A_3\text{-}A_2$, either

(a) $$^{r_3}\delta_{\hat{\tau}}(q_i,\sigma_1) = {}^{r_3}\delta_{\hat{\tau}}(q_m,\sigma_1)$$

or

(b) $$^{r_3}\delta_{\hat{\tau}}(q_i,\sigma_1) = {}^{r_3}\delta_{\hat{\tau}}(q_m,\sigma_2).$$

For case (a), $r_4 = r_3$. Otherwise reorder q_m so that

$$^{r_4}\delta_{\hat{\tau}}(q_i,\sigma_1) = {}^{r_4}\delta_{\hat{\tau}}(q_m,\sigma_1) = {}^{r_3}\delta_{\hat{\tau}}(q_m,\sigma_2)$$

$$r_4 = r_3 + (\text{encoded reordering of } q_m)$$

Then

$$A_4 = A_3 \cup \{q_m\},\ |A_4| \text{ is even}$$

and

$$\left|{}^{r_4}\delta_{\hat{\tau}}(A_4,\sigma_1)\right| = \frac{|A_4|}{2}$$

$$\left|{}^{r_4}\delta_{\hat{\tau}}(A_4,\sigma_2)\right| \leq \frac{|A_4|}{2} + 1$$

Since this step results in pairing the image of q_m under ${}^{r_4}\delta_{\hat{\tau}}(\cdot,\sigma_1)$ with the only unpaired image of q_i under ${}^{r_3}\delta_{\hat{\tau}}(\cdot,\sigma_1)$--added in step (3)--the cardinality of ${}^{r_4}\delta_{\tau}(A_4,\sigma_1)$ does not increase. Now if

$$\text{(i)} \quad {}^{r_4}\delta_{\hat{\tau}}(q_m,\sigma_2) \in {}^{r_4}\delta_{\hat{\tau}}(A_3,\sigma_2)$$

then

$$ {}^{r_4}\delta_{\hat{\tau}}(A_4,\sigma_2) = {}^{r_4}\delta_{\hat{\tau}}(A_3,\sigma_2)$$

also

$$\left| {}^{r_4}\delta_{\hat{\tau}}(A_4,\sigma_1) \right| = \frac{|A_4|}{2} = \left| {}^{r_4}\delta_{\hat{\tau}}(A_4,\sigma_2) \right|$$

and the process begins again at step (2).

If however,

$$\text{(ii)} \quad \left| {}^{r_4}\delta_{\hat{\tau}}(A_4,\sigma_2) \right| = \frac{|A_4|}{2} + 1$$

then there are exactly two unpaired states in ${}^{r_4}\delta_{\hat{\tau}}(A_4,\sigma_2)$--one of which can be matched by returning to step (3).

(5) Steps (2),(3),(4) are iterated according to the rules above until only one state remains.

That is for some i, $|Q|-|A_i| = 1$. A quick check of the procedure reveals that if $|Q|$ is odd, then step (3) will be the final step. Then the relationships at the end of this step,

$$\left|{}^{r_{i+1}}\delta_{\hat{\tau}}(A_{i+1},\sigma_1)\right| = \left[\frac{|A_{i+1}|}{2}\right] + 1$$

$$\left|{}^{r_{i+1}}\delta_{\tau}(A_{i+1},\sigma_2)\right| = \left[\frac{|A_{i+1}|}{2}\right] + 1$$

for $A_{i+1} = Q$, are those desired in the theorem. For $|Q|$ even, step (4) must terminate the process. But upon leaving step (3), exactly one unpaired state exists in both ${}^{r_i}\delta_{\tau}(A_i,\sigma_1)$ and ${}^{r_i}\delta_{\tau}(A_i,\sigma_2)$. Because of condition (2) in the hypothesis, the last state can be reordered so that its successor states match each of these. Hence the process must end with

$$\left|{}^{r_{i+1}}\delta_{\hat{\tau}}(A_{i+1},\sigma_1)\right| = \frac{|A_{i+1}|}{2} = \left|{}^{r_{i+1}}\delta_{\hat{\tau}}(A_{i+1},\sigma_2)\right|$$

with $A_i = Q$.

▷◁

It is worth emphasizing that because of the conditions of the theorem and the fact that the procedure builds up pairs of successor states in the images of A_i under ${}^{r_i}\delta_{\hat{\tau}}(\cdot,\Sigma)$, then at any point in the process short of the final step

$$ {}^{r_i}\delta_{\hat{\tau}}(A_i,\sigma_1) \cap {}^{r_i}\delta_{\hat{\tau}}(A_i,\sigma_2) = \phi $$

When $|Q|$ is even, this remains the case. For $|Q|$ odd, one and only one state belongs to both and

$$ \left| {}^{r}\delta_{\hat{\tau}}(Q,\sigma_1) \cap {}^{r}\delta_{\hat{\tau}}(Q,\sigma_2) \right| = 1 $$

To illustrate the method the details of reordering each of two controls are summarized in the examples which follow.

Example 4.3

Let a semi-control with an odd number of states be defined on a binary input alphabet by the transition function

$\delta_{\hat{\tau}}$:

	0	1
q_1	q_3	q_5
q_2	q_9	q_6
q_3	q_5	q_9
q_4	q_6	q_1
q_5	q_8	q_2
q_6	q_1	q_3
q_7	q_2	q_8
q_8	q_7	q_4
q_9	q_4	q_7

Applying the method to $\hat{\tau}$ results in the sequence:

after step 1: $A_1 = \{q_8, q_9\}$; $|A_1| = 2$

$$r_1 = (256)_{10} = (100000000)_2$$

$$\left| r_1{}_{\delta_{\hat{\tau}}}(A_1, \sigma_1) \right| = 1 = \frac{|A_1|}{2} = \left| r_1{}_{\delta_{\tau}}(A_1, \sigma_2) \right|$$

after step 2: $A_2 = A_1 \cup \{q_1, q_6\}$; $|A_1| = 4$

$$r_2 = (288)_{10} = r_1 + (000100000)_2 = (10010000)_2$$

$$\left| r_{2\,\delta_{\hat{\tau}}(A_2,\sigma_1)} \right| = 2 = \frac{|A_2|}{2}$$

$$\left| r_{2\,\delta_{\hat{\tau}}(A_2,\sigma_2)} \right| = 3 = \frac{|A_2|}{2} + 1$$

after step 3: $A_3 = A_2 \cup \{q_4\}$; $|A_3| = 5$

$$r_3 = r_2$$

$$\left| r_{3\,\delta_{\hat{\tau}}(A_3,\sigma_1)} \right| = 3 = \left[\frac{|A_3|}{2}\right] + 1$$

$$\left| r_{3\,\delta_{\hat{\tau}}(A_3,\sigma_2)} \right| = 3 = \left[\frac{|A_3|}{2}\right] + 1$$

after step 4: $A_4 = A_3 \cup \{q_2\}$; $|A_4| = 6$

$$r_4 = (290)_{10} = r_3 + (000000010)_2 = (100100010)_2$$

$$\left| r_{4\,\delta_{\hat{\tau}}(A_4,\sigma_1)} \right| = 3 = \frac{|A_4|}{2}$$

$$\left| r_{4\,\delta_{\hat{\tau}}(A_4,\sigma_2)} \right| = 4 = \frac{|A_4|}{2} + 1$$

after step 3: $A_5 = A_4 \cup \{q_3\}$; $|A_5| = 7$

$$r_5 = (294)_{10} = r_4 + (000000100)_2 = (100100110)_2$$

$$\left| r_5 \delta_{\hat{\tau}}(A_5, \sigma_1) \right| = 4 = \left[\frac{|A_5|}{2} \right] + 1$$

$$\left| r_5 \delta_{\hat{\tau}}(A_5, \sigma_2) \right| = 4 = \left[\frac{|A_5|}{2} \right] + 1$$

after step 4: $A_6 = A_5 \cup \{q_5\}$; $|A_6| = 8$

$$r_6 = r_5$$

$$\left| r_6 \delta_{\hat{\tau}}(A_6, \sigma_1) \right| = 4 = \frac{|A_6|}{2}$$

$$\left| r_6 \delta_{\hat{\tau}}(A_6, \sigma_2) \right| = 5 = \frac{|A_6|}{2} + 1$$

after step 3: $A_8 = A_7 \cup \{q_7\} = Q$

$$r_8 = (358)_{10} = r_7 + (001000000)_2 = (101100110)_2$$

$$\left| r_8 \delta_{\hat{\tau}}(A_8, \sigma_1) \right| = \left| \{q_3, q_6, q_7, q_8, q_9\} \right|$$

$$= 5 = \left[\frac{|A_8|}{2} \right] + 1$$

$$\left| r_{8_{\delta_{\hat{\tau}}}}(A_8, \sigma_2) \right| = \left| \{q_1, q_2, q_4, q_5, q_9\} \right|$$

$$= 5 = \left[\frac{|A_8|}{2} \right] + 1$$

and

$r_{8_{\delta_{\hat{\tau}}}}$:

	0	1
q_1	q_3	q_5
q_2	q_6	q_9
q_3	q_9	q_5
q_4	q_6	q_1
q_5	q_8	q_2
q_6	q_3	q_1
q_7	q_8	q_2
q_8	q_7	q_4
q_9	q_7	q_4

Example 4.4

For an illustration of a semi-control with an even number of states, consider $\hat{\tau}$ defined by

$\delta_{\hat{\tau}}$:

	0	1
q_1	q_2	q_3
q_2	q_3	q_4
q_3	q_4	q_5
q_4	q_5	q_6
q_5	q_6	q_1
q_6	q_1	q_2

The reordered semi-control obtained from applying the method to $\hat{\tau}$ is

$r_{6_{\delta_{\hat{\tau}}}}$

	0	1
q_1	q_2	q_3
q_2	q_4	q_3
q_3	q_4	q_5
q_4	q_6	q_5
q_5	q_6	q_1
q_6	q_2	q_1

$r_6 = (42)_{10} = (101010)_2$

$$\left| r_{6_{\delta_{\hat{\tau}}}(Q,\sigma_1)} \right| = \left| \{2,4,6\} \right| = 3 = \frac{|Q|}{2}$$

$$\left| r_{6_{\delta_{\tau}}(Q,\sigma_2)} \right| = \left| \{1,3,5\} \right| = 3 = \frac{|Q|}{2} .$$

In applying this theorem to obtain a bound on the direct product control, if two semi-controls, $\hat{\tau}_1$ and $\hat{\tau}_2$, have state transition functions which satisfy hypotheses (1) and (2), then the following is true:

if

$$\hat{\tau}_1 \times \hat{\tau}_2 \in R\left[\tau_1 \tau_2\right]$$

$$\left|Q_{\tau_1}\right| = n \; ; \quad \left|Q_{\tau_2}\right| = m$$

then

$$\left| Q_{r_{\hat{\tau}_1} \times s_{\hat{\tau}_2}} \right| \leq B$$

where

$$B = \left(\left[\frac{m}{2}\right] + 1\right) * \left(\left[\frac{n}{2}\right] + 1\right) \quad : \text{ m odd, n odd}$$

$$= \left(\left[\frac{m}{2}\right] + 1\right) * \left(\frac{n}{2}\right) \quad : \text{ m odd, n even}$$

$$= \left(\frac{m}{2}\right) * \left(\left[\frac{n}{2}\right] + 1\right) \quad : \text{ m even, n odd}$$

$$= \left(\frac{m}{2}\right) * \left(\frac{n}{2}\right) \quad : \text{ m even, n even.}$$

It is clear that the reordered controls resulting from applying the method to $\hat{\tau}_1$ and $\hat{\tau}_2$ can be used to generate the direct product if the above bound is satisfactory.

In practice the method applied to most any n-state, cyclic control over a binary alphabet produces an SP cover of the type mentioned above with no more than $\left[\frac{n}{2}\right] + 1$ states in each of the two blocks.

CHAPTER V

CONCLUSIONS AND OPEN PROBLEMS

This chapter presents a brief summary of the essential aspects of this dissertation and suggests possible areas for further investigation.

5.1 Summary

Chapter I applied the term, algorithm, to a particular binary relation on a regular set, Σ^*, and a set of finite outputs, Y. When this relation satisfied certain properties it represents the sequential relation of a finite state machine. This model was taken as the schemata or control of a program which associated "meaning"--via an interpretation--to the states, input symbols and outputs of the control.

The theorems in Section 2.1 establishing correspondences between program values and valuation sequences provided the setting for the rest of the dissertation: the importance and utility of a model of the control flow structure of a program. The important notion of a program-value equivalence relation led to a partial ordering, R_I, and an interpretation equivalence relation, $\equiv_I$, on controls. A common control was

one which stood in the relation R_I to any one member in a class of controls such that for any interpretation on a control belonging to the class, an appropriate interpretation could be constructed for the common control to insure program value equivalence. When the common control was an H- or A-type structural realization of the control class then Theorems 2.6 and 2.7 detailed the method of constructing the appropriate interpretations.

Sections 3.1 and 3.2 utilized the standard notions of SP partitions and semigroup accumulators to synthesize an H-type common control. The composite realization of Section 3.4 illustrated how new types of controls can be synthesized using the guidelines set forth in Theorem 2.7. In Chapter IV, Theorem 4.1 demonstrated that direct product controls were also H-type realizations. The important reordering transformation (Definition 4.1) was introduced and Theorem 4.2 showed that any two reorderings of the same control were R_I related and, more importantly, were value ($\underset{\sim}{\equiv}_V$) equivalent via Theorem 4.3. The significance of the reordering control transformation came into focus with Theorems 4.5 and 4.6. Here bounds were established on the states of the semigroup accumulator of a control by restructuring the SP lattice of the control or, in the case of Theorem 4.7, striving to

obtain, by reordering, a permutation-reset machine. After showing how isomorphic control substructures serve to reduce a direct product control, the chapter concluded with Theorem 4.12 which provided a method by which the cardinal number of a particular SP cover is reduced. This resulted in a tighter bound on the set of R-minimal direct product controls.

5.2 Conclusions

This dissertation presents a unified approach to program control analysis and synthesis. Some useful structural properties of program controls are identified and may be detected with certain finite state machine techniques. Common structural attributes so identified in a class of program controls are exploited in construction theorems to fabricate a new common control structure that inherits these attributes. This synthesized control can realize the computational activities of any control in the class for each member of a set of interpretations defined on that control.

The well known semigroup, direct product and homomorphic submachine structures are examined in detail to establish confidence in the methods and approach. A new composite realization is developed to suggest the possibility of creating a number of such constructs with the interpreted control model.

Perhaps more significantly, the approach leads to an optimization procedure which relies on a novel control transformation that preserves program value but not structural properties of controls. When this transformation is applied to any of the common control constructs above, certain economies are achieved. It has become apparent that the potential usefulness of this mechanism is far from being exhausted. Some possibilities are explored in the next section.

5.3 Open Questions

The examples presented in Chapters III and IV involving an implementation of an interpreted control utilized FORTRAN exclusively. This was to demonstrate that, in synthesizing common controls, changes required in decision predicates for non-isomorphic input alphabets or providing starting states or return points for each control realized by a common control can be carried out even in such an overworked language as FORTRAN. It is clear that pragmatic difficulties can be eased by employing a language with richer branching facilities--APL being a good example. The possibilities of simultaneously modeling the control flow of a set of algorithms programmed in some procedure oriented language and, say, a search of some nonlinear data structure represented by a program implemented in a list processing language seem intriguing. FORTRAN was utilized to suggest the feasibil-

ity, but not the extent, of implementing the type of controls developed in the preceding text.

In justifying the claim that an interpreted control can be of use in the modeling, analysis and synthesis of programs and their structures, considerable reliance was placed on rather standard notions of machine realizability. This facilitated the construction of the interpretation by way of composition mappings. The approach is a natural and common one and, not surprisingly, composition maps have found their way into more abstract notions of machine simulation, e.g. Herman [1968]. Are similar constructions possible for other types of machine realizations--for example machine capability (Hartmanis [1966])? How can the input mapping requirement

$$\rho^*: \Sigma_{\tau_i} \rightarrow \Sigma_\Gamma{}^*$$

be reflected in the decision predicates mapped onto Q_Γ by some interpretation? A satisfactory approach to this problem would permit the use of n-state universal machines to model any n-state program. On the other hand, can any of the various structural decomposition techniques suggest a more useful interpretation model so that a partial computation represented by a submachine of some control, τ, or its semigroup accumulator has a corresponding partial

interpretation?

Even restricted to the control and interpretation models employed in this dissertation, the reordering transformation opens up a number of interesting possibilities. It has been demonstrated that reordering can drastically alter program control structure while maintaining program value with corresponding changes in the decision predicates. This fact can be used advantageously if, say in the Zeiger decomposition of two or more controls, these controls can be reordered so as to maximize the number of PR machines they have in common. Unfortunately it does not follow that the direct product of these reordered controls is R-minimal. Can the reordering process be dictated somehow by the various steps in the Zeiger construction to produce R-minimal direct products? Short of exhaustive search what is the most effective method(s) of generating minimal common controls of any kind? An example of one heuristic which seems effective in minimizing the direct product of two semi-controls, $\hat{\tau}_1 \times \hat{\tau}_2$, is to minimize the direct product by searching all reorderings on $\hat{\tau}_2$, fixing this reordered machine and then minimizing over reorderings on $\hat{\tau}_1$. Continuing in this way a path is traced on the two dimensional grid of machine reorderings which often converges to a

R-minimal direct product. Typically it has been found this path has only four sides, i.e. a search was required over just $2*(2^m+2^n)$ machines where $|Q_{\hat{\tau}_1}| = m$ and $|Q_{\hat{\tau}_2}| = n$. Contrast this with the worst case of 2^{m+n} searches and there is strong motivation for developing new heuristics and, hopefully, convergent procedures.

BIBLIOGRAPHY

[1969] Arbib, M.A. Theories of Abstract Automata, Prentice Hall, Englewood Cliffs, New Jersey, 1969.

[1962] Armerding, G.W., FORTAB: A Decision Table Language for Scientific Computing Applications, Memorandum RM-3306-PR, Rand Corporation, Santa Monica, California, September 1962.

[1968] Assmus, E.F. and Florentine, J.J., "Algebraic Machine Theory and Logical Design," Algebraic Theory of Machines, Languages and Semigroups, Arbib, M.A. ed., Academic Press, New York, 1968.

[1964] Elgot, C.C. and Robinson, A., "Random-Access Stored-Program Machines, An approach to Programming Languages," JACM, Vol. 11, No. 4, pp. 365-399, 1964.

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

[1966] Harrison, M.A., and Gray, J.N., "The Theory of Sequential Relations," Information and Control, Vol. 9, No. 5, pp. 435-468, 1966.

[1966] Hartmanis, J., and Stearns, P.E. Algebraic Structure Theory of Sequential Machines, Prentice Hall, Englewood Cliffs, New Jersey, 1966.

[1968] Herman, G.T., "Simulation of One Abstract Computing Machine By Another," Comm. ACM., Vol. 11, No. 12, pp. 802, 813, December 1968.

[1958] Ianov, IU.,I., "On the Equivalence and Transformation of Program Schemes," Comm. ACM., Vol. I, pp. 8-12, October 1958.

[1958] _____, "On Matrix Program Schemes," Comm. ACM., Vol. I, pp. 283-286, December 1958.

[1969] Kaplan, D.M., "Regular Expressions and the Equivalence of Programs," Journal of Computer and Systems Sciences, Vol. 3, pp. 361-386, 1969.

[1967] Karp, R.M. and Miller, R.E., "Parallel Program Schemata: A Mathematical Model for Parallel Computations," IEEE Conference Record of 1967 Eighth Annual Symposium on Switching and Automata Theory, pp. 55-61, 1967.

[1966] _____, "Properties of a Model for Parallel Computations: Determinancy, Termination, Queueing," SIAM Journal of Applied Mathematics, Vol. 14, No. 6, pp. 1390-1411, November 1966.

[1968] Luconi, F.L. "Completely Functional Asynchronous Computational Structures," IEEE Conference Record of the 1967 Eighth Annual Symposium on Switching and Automata Theory, pp. 62-70, 1967.

[1967] Mathers, H.W., Sedore, S.R., Sents, J.R., Sceptre: Automated Digital Computer Program for Determining Responses of Electronic Circuits to Transient Nuclear Radiation," Technical Report No. AFWL-TR-66-126, Vol. II, February 1967.

[1966] Maurer, W.D., "A Theory of Computer Instructions," Journal of the Association for Computing Machinery, Vol. 13, No. 2, pp. 226-235, April 1966.

[1968] _____, Programming: An Introduction to Computer Languages and Techniques, Holden-Day, Inc. San Francisco, 1968.

[1965] Mesarovic, M.D., "New Directions in General Theory of Systems," Systems and Computer Science, Hart, J.F. and Takasu, S. eds., pp. 221-231, University of Toronto Press, Toronto, Ontario, 1965.

[1963] Myhill, J., "Finite Automata, Semigroups and Simulation," Lecture Notes, University of Michigan, Ann Arbor, Michigan, 1963.

[1965] Page, C.V., Equivalences Between Probabilistic Sequential Machines, Thesis, University of Michigan, Ann Arbor, Michigan, 1965.

[1967] Patterson, M.S., Equivalence Problems in a Model of Computation, Thesis, Trinity College, Cambridge, Massachusetts, 1967.

[1962] Pollack, S.L., DETAB-X: An Improved Business-Oriented Computer Language, Memorandum RM-3273-PR, Rand Corporation, Santa Monica, California, August 1962.

[1964] Rutledge, J.D., "On Ianov's Program Schemata," JACM, Vol. II, No. 1, pp. 1-9, January 1964.

]1965] Windeknecht, T.G., "Concerning an Algebraic Theory of Systems," Systems and Computer Science, Hart, J.F. and Takasu, S. eds., pp. 232-249, University of Toronto Press, Toronto, Ontario, 1965.

[1968] Zeiger, H.P., "Cascade Synthesis of Finite State Machines," Algebraic Theory of Machines Languages and Semigroups, Arbib, M.A. ed., Academic Press, New York, 1968.