AD 669812

ANALYSIS OF IMPLEMENTATION ERRORS IN DIGITAL COMPUTING SYSTEMS

Technical Report No. 6

March, 1968

JUN 10 1968

This document has been approved for public release and sale; its distribution is unlimited.

Computer Systems Laboratory
Washington University
St. Louis, Mo.

Reproduced by the
CLEARINGHOUSE
for Federal Scientific & Technical
Information Springfield Va. 22151

98

DISCLAIMER NOTICE

THIS DOCUMENT IS THE BEST QUALITY AVAILABLE.

COPY FURNISHED CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

ANALYSIS OF IMPLEMENTATION ERRORS IN DIGITAL COMPUTING SYSTEMS

Robert M. Keller and Donald F. Wunn

TECHNICAL REPORT NO. 6

March, 1968

Computer Systems Laboratory
Washington University
St. Louis, Missouri

This work has been supported by the Advanced Research Projects Agency of the Department of Defense under contract SD-302 and by the Division of Research Facilities and Resources of the National Institutes of Health under Grant FR-00218.

"Errors, like straws, upon the surface flow;
He who would search for pearls must dive below."

Dryden, *All for Love*, Prologue

ABSTRACT

This report discusses problems encountered with control networks in highly restructurable digital systems. In particular the treatment of implementation errors is covered with emphasis on concurrent processing. The implementation of concurrent processing networks may result in errors which will be quite complex to detect and systematic methods are warranted. A model representing a particular type of computing system is presented, and methods for introducing concurrent control into the model discussed. The automatic detection of a certain class of errors caused by improper design of these systems is investigated. Graph theoretic representation is employed in demonstrating several error detection techniques. The properties of these techniques are compared and it is concluded that one technique, of those investigated is of sufficient generality, thoroughness, and simplicity in implementation to be used for automatic error analysis.

TABLE OF CONTENTS

No. Page

1. Introduction 1
   1.1 Motivation 1
   1.2 Asynchronous Concurrent Mode as Compared with Other Modes 1
      1.2.1 Levels 1
      1.2.2 Synchronous versus Asynchronous 1
      1.2.3 Serial versus Concurrent 2
   1.3 Domain of Interest 2
      1.3.1 Historical Development 2
      1.3.2 Current Research 2
   1.4 Problems Introduced by Allowing Explicit Concurrency 3
2. A Model of the Type of Computing System to be Analyzed 4
   2.1 Signals, Paths, Processes, and Memories 4
      2.1.1 Data Signals and Memories 4
      2.1.2 Processes which Transform Data 4
      2.1.3 Control Signals 6
      2.1.4 General Processes 6
      2.1.5 Summary of Elements in the Model and an Example 6
   2.2 Process Networks 8
      2.2.1 Sequential Process Networks 10
         2.2.1.1 Synchronous and Asynchronous Sequential Processes 10
         2.2.1.2 Concepts Encountered in Sequential Asynchronous Networks 10
         2.2.1.3 Decision and Merge Processes 10
      2.2.2 An Example of Asynchronous Sequential Control 12
      2.2.3 Concurrent Process Networks 12
   2.3 Application to Computing Systems 16
      2.3.1 Application at the Organizational Level 16
      2.3.2 Application at the Program Level 16
3. Graph Theoretic Concepts 18
   3.1 Definitions 18
   3.2 Matric Representations 20
4. Errors in Networks of Concurrent Processes with Asynchronous Control 26
   4.1 General Types of Errors 26
      4.1.1 Infinite Duration 26
      4.1.2 Regeneration 29
      4.1.3 Indeterminacy 29
      4.1.4 Summary of Implementation Errors 34
   4.2 Detection of Implementation Errors 34
      4.2.1 Simulation 37
         4.2.1.1 Simulation with Trial Data 43

TABLE OF CONTENTS
continued

No.		Page
	4.2.1.2 Monte Carlo Simulation	43
	4.2.1.3 Exhaustive Simulation	43
	4.2.2 Topological Analysis	44
	4.2.3 Symbolic Analysis Using Algebraic Expressions	44
	4.2.4 The State Transition Method	53
	4.2.5 A Summary of the Techniques of this Study	59
5.	Summary and Conclusion	61
6.	Acknowledgement	62
7.	Appendix	63
8.	References	76
9.	Bibliography	79

LIST OF FIGURES

No. Page

1. Representation of data signal paths and memory 5
2. Process with initiation and completion signals 7
3. Example of a process showing the existence of signals on control and data paths 9
4. Representation of decisions 11
5. Representation of merges 11
6. Separable process which computes n-factorial 13
7. Possible subprocesses for Figure 6 14
8. Concurrent asynchronous control of processes T, U, and V with the precedence relation $T < V$, $U < V$ 15
9. Example of a graph 19
10. A strongly connected graph 21
11. A subgraph of the graph in Figure 10. This subgraph is separable 21
12. The maximal strongly connected subgraph of the subgraph in Figure 11 21
13. A minimal strongly connected subgraph of the subgraph in Figure 12 21
14. Input and output matrices for the graph of Figure 9 22
15. Arc-node matrix for the graph of Figure 9 23
16. The connection and reachability matrices for the graph of Figure 9 25
17. A network with an infinite duration due to an algorithm 27
18. Networks which may have infinite duration errors because of incorrect algorithm 27
19. Networks having infinite duration because of incorrect use of concurrent-control elements 28
20. Regeneration by branch within a strongly connected subgraph 28
21. Possible regeneration caused by a hazard 30
22. Networks with resudual control 31
23. Incorrect implementation for $A < B$ 32
24. Correction of the error in Figure 23 33
25. Error-free graphs 35
26. Examples of graphs with errors 36
27. A graph displaying the maximum number of combinations for two decisions 38
28. A graph with two decisions and less than the maximum number of combinations 38
29. A 3-decision graph with four combinations 39
30. Indication of combinations for a strongly connected graph 40
31. Combinations for a graph with a strongly connected subgraph 40
32. Control for a floating-point arithmetic unit 41
33. Exhaustive test of an error-free network 45
34. Exhaustive test of a network with errors 46
35. Exhaustive test of a network with undetectable errors 47
36. Symbolic analysis showing a hazard 49
37. Symbolic analysis of an infinite duration case 50
38. Symbolic analysis of an error-free case 51
39. Symbolic analysis of an error case 52

LIST OF FIGURES
continued

No. Page

40. Some simple error-free networks and their corresponding state transition graphs 54
41. Partial state transition graph for a branch 55
42. Partial state transition graphs for decisions, merges, and rendezvous 56
43. State transition graphs for error cases 57
44. State transition graphs for error cases 58

BLANK PAGE

ANALYSIS OF IMPLEMENTATION ERRORS IN DIGITAL COMPUTING SYSTEMS SUPPORTING ASYNCHRONOUSLY-CONTROLLED CONCURRENT PROCESSES

1. INTRODUCTION

This report is concerned with digital computing systems supporting asynchronously-controlled concurrent processes. Systems of this variety present a departure from techniques of conventional usage. Certain methods may be used to provide explicit concurrent control in these systems. If the methods are incorrectly applied, a number of different errors, which are unlike those encountered in conventional systems, may result. Presented here is a discussion of explicit concurrent control methods and an investigation of techniques for automatic detection of errors introduced in using these methods. Several solutions are demonstrated and their relative merits evaluated.

1.1 MOTIVATION

The desirability for increased speed in computer systems has focused interest on two major areas:

1. Digital electronics
2. Computer organization

In the first area, the goal is development of electronic switching networks, ferrite core memories, and other components capable of operating at extremely high speeds.[1] The second, which is largely independent of the first, involves efforts toward the effective usage of existing components. It is this latter area which will be of concern here.

1.2 ASYNCHRONOUS CONCURRENT MODE AS COMPARED WITH OTHER MODES

1.2.1 LEVELS

Before determining whether a particular computer falls into the asynchronous concurrent category, the qualification of *level* must be made. Three levels will be considered:

1. The logic level
2. The organizational level
3. The program level

The logic level is that at which the elementary entities are gates, flip-flops, clocks, etc. The organizational level has as elements registers, memories, and other units constructed of logic elements. It may also include arithmetic units, input-output controllers, or even an entire processing unit. At the program level the elements are instructions written in a sequence which describes the operations to be performed by a computer.

1.2.2 SYNCHRONOUS VERSUS ASYNCHRONOUS

Synchronous means that operations are controlled by a clock with a fixed period. Processes at the logic level in most conventional computers are synchronous. The reason for this is that at the logic level, synchronous control is easier to use in design.

Contrarily, at the program level, processes usually operate asynchronously. The execution time of instructions in most computers varies depending on the type of operation or the amount of data being manipulated.

1.2.3 SERIAL VERSUS CONCURRENT

Concurrent means that processes occur simultaneously, while serial implies one process proceeding after another in a particular order. In contrast to the examples in the previous section for conventional computing systems, at the logic level concurrent processes do occur, while at the program level they do not. Some qualification needs to be made concerning the latter statement. Most contemporary computers do provide for concurrency of input and output operations with other types of operations. However, the program generally does not have absolute control of these operations. It may be said that the programmer does not normally have the option of explicitly declaring concurrency.

1.3 DOMAIN OF INTEREST

The processes to be considered in this research will be entirely at the organization or program levels. A model will be proposed which is adequate for the representation of processes at either level and its applicability to existing computers demonstrated. The model is particularly suited to organization or programming of the class of computers originally proposed by von Neumann[2], in which the greater percentage of existing computers are included. No attempt is made to show its adequacy for various computers such as SOLOMON[3], the Holland Machine[4], and other computers which are described as *highly parallel, distributed logic,* etc. For a cross sectional description comparing various types of concurrent processors, see Murtha[5].

1.3.1 HISTORICAL DEVELOPMENT

Examination of the characteristics of computers since the first large-scale computer, the Harvard Mark I Calculator[6] in 1944, yields an interesting picture regarding concurrent processing. The successor to the Mark I, the ENIAC (Electronic Numerical Integrator and Computer)[7], was capable of sustaining concurrent processes. This feature was made possible by the use of wired programs.

With the introductions of EDVAC (Electronic Discrete Variable Automatic Computer)[2], which was the first stored program machine, problems with the control increased, and thus, attention was drawn away from concurrent processing.

As the use of electronic computing increased, it became apparent that certain functions of a computer, e.g multiplication, division, and certain input and output operations, consumed a disproportionate amount of time in comparison to other operations. Consequently, during operations such as these, part of the components of the computer remained idle. This renewed interest in applying asynchronous control and concurrency to more effective utilization of this idle time. Several machines then appeared which allowed multiplication and division to proceed simultaneously and autonomously[8]. Another step was the introduction of an input-output overlap feature in the UNIVAC I. This feature, which allowed input and output operations to proceed autonomously and concurrently with a program, is present in most commercial and scientific computers presently manufactured. This idea was then extended to permit other types of instructions to be executed simultaneously by the interconnection of two or more computers.

1.3.2 CURRENT RESEARCH

Currently, many existing and planned computer systems are incorporating concurrent asynchronous control. Unfortunately, few of these allow explicit specification of concurrency at the program level, and some give this

privilege to a supervisory program only.

At the organizational level, the trend toward more flexibility and modularity of units has offered a growing opportunity for development of new approaches for concurrent structures. The *fixed-plus-variable* computer proposed by Estrin[9] was a major step in this direction. Estrin suggested that a standard computer be combined with a network of computer components under common control of a supervisor. The network could be *restructured* for particular problems to yield an increase in program running speed.

Another significant advancement, consisting of a collection of autonomously operating modules, was proposed by Clark[10]. These modules, called *macromodules* were to be designed in such a way as to eliminate the electronic engineering details present in conventional computers and thus provide a means of organizing computing systems by considering only the functions to be performed. This project is currently in the development stage.

1.4 PROBLEMS INTRODUCED BY ALLOWING EXPLICIT CONCURRENCY

The provision for explicit asynchronous control of concurrent processes has introduced problems not encountered in computing systems of other types. Some of these problems have been discussed in the literature and generally deal with questions of how to use this type of system most effectively.

The problem of scheduling processes, deciding which processes are handled by which units of the system and at what time, is considered in 11, 12, 13, 14. The effects on the specification of algorithms is investigated in 14, 15, 16, 17 and the effect on program-language compilers in 18, 19, 20. Discussions of interrupt handling, memory usage, and other problems peculiar to certain systems may be found in 21, 22, 23.

This report concentrates on the problem of detecting certain types of errors which may be introduced in implementing concurrent computing systems. These will be called implementation errors. A general approach applicable to a large class of computers is used, and examples are presented illustrating the method as utilized in macromodular constructions. Implementation errors have been previously discussed in 24, 25, 26.

2. A MODEL OF THE TYPE OF COMPUTING SYSTEM TO BE ANALYZED

Prior to considering implementation errors, it is necessary to present a model of the computing system to be analyzed. The model may be used to represent certain computers at either the organizational level or the program level. The basic elements of the model are the *signal*, the *process*, the *signal path*, and the *memory*.

2.1 SIGNALS, PATHS, PROCESSES, AND MEMORIES

The definitions of signal and process are of a recursive nature i.e., signals are responsible, among other things, for initiating processes; but, processes may be said to create signals. To simplify definitions, the signals are classified into two types: *data* signals and *control* signals. The signal path, being a medium for a signal, will be introduced with the signals. The order of the subjects in the following discussion will be:

1. Data signals and memories
2. Processes which transform data signals
3. Control signals
4. Processes in general

2.1.1 DATA SIGNALS AND MEMORIES

A *data signal* is an entity which conveys information by assuming one of a number of possible values. It exists in a medium known as a *path*. The value of the data signal may be recorded by an element known as a *memory*. After the value of a data signal is recorded by memory, the signal ceases to exist. The memory element has the property that it subsequently creates data signals having the value which the memory last recorded. Only one value is retained at any one time. Signals are recreated by a memory whenever they are requested by a process.

The signals whose values may be recorded by a memory are restricted to certain paths associated with the memory. Similarly, signals may be created only on paths associated with the memory. A memory and associated paths is represented schematically in Figure 1. The memory is represented by a rectangle while the paths are represented by arrows. The arrow is directed *into* a memory if the memory records the value of a signal on the path. The arrow is directed *from* the memory if the memory creates signals on that path.

For the particular systems which will be modeled, it is required that a path support only one data signal at any instant of time. For contrast, a theoretic model not having this restriction is described by Karp and Miller[27] and Reiter[28,29].

2.1.2 PROCESSES WHICH TRANSFORM DATA SIGNALS

There are various types of processes, one of which functions to transform data signals. By *transform*, it is meant that some data signals may be created whereas others are destroyed. When destroying a signal, a process may inspect its value, which may have an effect on the subsequent action of the process. The process may create data signals, the value of which depends on data signals previously inspected. Thus, the transformation spoken of is really a mapping from the set of all possible data signals into itself.

As with a memory element, a particular process may be allowed to transform only a certain set of signals. This set is determined by a set of *paths* associated with the process. The paths may connect to memories and are represented by arrows, which are the same as those arrows described for memories in the preceding section. The arrow is directed into a process if the process requests data from a memory, inspects, and destroys the data signal on that path. The arrow is directed outward from a process if a data signal may be created on that path.

It should be mentioned that a process may transform data only intermittently. When a process is trans-

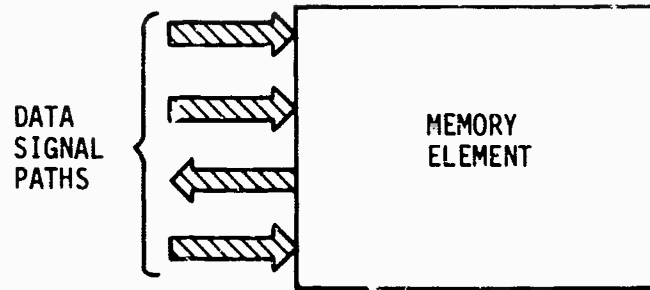


FIGURE 1. REPRESENTATION OF DATA SIGNAL PATHS AND MEMORY

forming signals, it is said to be *active*. Otherwise, it is *inactive*. Processes which do transform continuously are called *continuous processes* while those which do not are called *discrete processes*. The processes to be considered will be implicitly discrete unless specified otherwise.

2.1.3 CONTROL SIGNALS

If a process is discrete, i.e., it is active only at certain times, it is necessary to provide a means of rendering it active, or initiating it. This is accomplished by another type of signal, the *control signal*. In contrast to data signals, the control signal simply exists or it does not. There is no associated value.

The control signal, unlike the data signal, may initiate a process spontaneously. Once it has done so, it is destroyed. When a control signal has this effect on a process, the process is said to *accept* the signal. Also, when an existing process has completed its transformation, it ceases to be active and creates a control signal indicating its completion. This signal may then be used to initiate other processes.

As with data signals, only certain control signals are associated with any process. These exist on particular paths, and *only one control signal may exist on a path at any given time*. The paths are represented by arrows which are lighter and thinner than those representing data paths. The arrow is directed inward if a signal on the path initiates the process and outward if the process creates a signal on the path, as in Figure 2. The data paths are not always shown if explicit reference is made to memory elements inside the figure representing the process.

2.1.4 GENERAL PROCESSES

In the preceding sections, a process was described as functioning to transform data upon the acceptance of an initiation signal and to return a completion signal at the end of the transformation. Now that control has been defined, a more general definition of *process* may be given.

A process may accept control signals on more than one path to it and the existence of signals on these paths may affect the process. Also, a process may create more than one control signal, which may initiate other processes. It is not necessary that the signal which initiates the process always be on the same path. Control signals accepted by a process are called *input control signals*. That which initiates the process is the *primary input control signal*, while others are known as *secondary input control signals*. Similarly, a single control signal is created which indicates that the process no longer exists. This will be called the *primary output control signal*, while others are known as *secondary output control signals*. The *primary output control signal* is not generally required to be on a particular path. To simplify discussion, if there is more than one input control, *initiation signal* may be used to mean *primary input control signal* and if there are multiple output controls, *completion signal* may be used to mean *primary output control signal*. In a similar manner, data signals will be described as *input* or *output* with respect to a process, depending on whether they are destroyed or created by that process.

2.1.5 SUMMARY OF ELEMENTS IN THE MODEL AND AN EXAMPLE

A summary of the concepts introduced in sections 2.1.1 through 2.1.4 is now presented. The elements of the model are:

1. *Signals* – are accepted and created by processes and memories, and provide for intercommunication.
   - A. *Data signals* – convey *values*
     1. *Input data signals* – are requested, inspected, and destroyed by processes and their values are recorded by memories.
     2. *Output data signals* – are created by processes or memories.

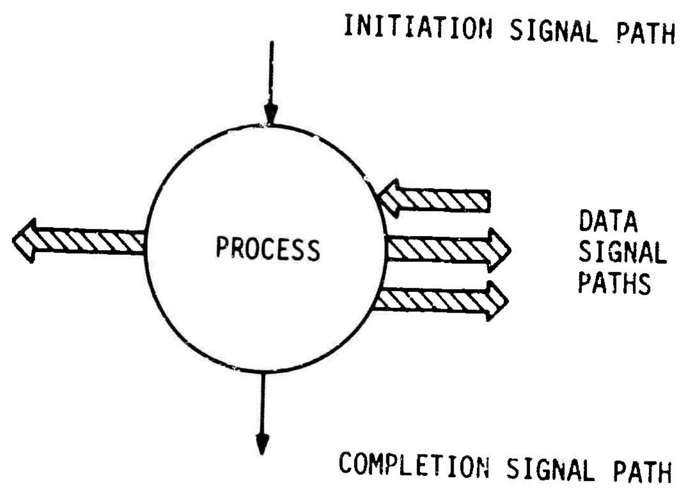


FIGURE 2. PROCESS WITH INITIATION AND COMPLETION SIGNALS

B. *Control signals* – control processes and have only a single value.
   1. *Control input signals* – are accepted by processes.
      a. *Primary control input* or *initiation signals* – cause the activation of a process.
      b. *Secondary control input signals* – control processes but do not initiate them.
   2. *Control output signals* – are created by processes.
      a. *Primary control output* or *completion signals* – indicate that a process is no longer active.
      b. *Secondary control output signals* – are produced by a process prior to completion.

II. *Paths* – are media in which signals exist.
   A. *Control paths* – may sustain control signals only.
   B. *Data paths* – may sustain data signals only.

III. *Processes* – have initiation and completion signals and the ability to accept and create control and data signals.

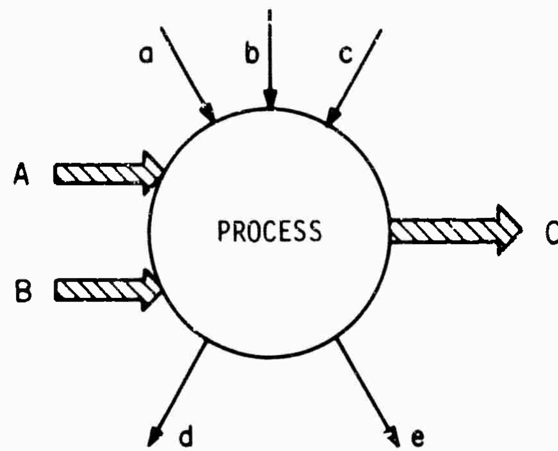
IV. *Memories* – record the most current value of a certain set of data signals and recreate signals having this value.

An example of a general process is shown in Figure 3. The timing diagram indicates the presence of signals on various paths. Those which are due to the process are indicated by solid lines, while those from some external source are indicated by dashed lines. In this example, the initiation signal will always be on path *a* and the completion signal will be on either path *d* or *e*, by assumption. A secondary control input may be on either path *b* or *c*, but not both. The process may be described as follows: After initiation by the signal on path *a*, the process waits for a signal on *b* or *c*. If a signal occurs on *b*, the data signals *A* and *B* are compared. If *A* and *B* have the same value, this value is given to a signal created on path *C* and a completion signal is created on *d*. If the values of *A* and *B* are different, a signal is created on *C* with a predetermined value and a completion signal is created on *d*. If a control signal appears on *c* instead of *b*, the data signals on *A* and *B* are destroyed and ignored. The predetermined value is assigned to a signal on *C* and the completion reported on *e*.

The timing diagram in Figure 3 shows two example cases. In the first, an input appears on *b* and the signals on *A* and *B* have the same value, thus the completion is reported on *d*. In the second, an input appears on *c*, thus completion is reported on *e*.

2.2 PROCESS NETWORKS

Networks of processes will now be discussed. A network of processes is a set of processes and memories interconnected by data and control signal paths. First, asynchronous and synchronous sequential networks will be compared. *Sequential* means that only one process is active at any one time and thus, the processes occur in a sequence, one after another. Following the discussion of sequential processes, *concurrent* processes will be investigated and their advantages described. In concurrent process networks, more than one process may be active at any time. The terms *synchronous* and *asynchronous* will be applied to concurrent networks also, which leads to the type of network with which this report is mainly concerned, asynchronous concurrent process networks.



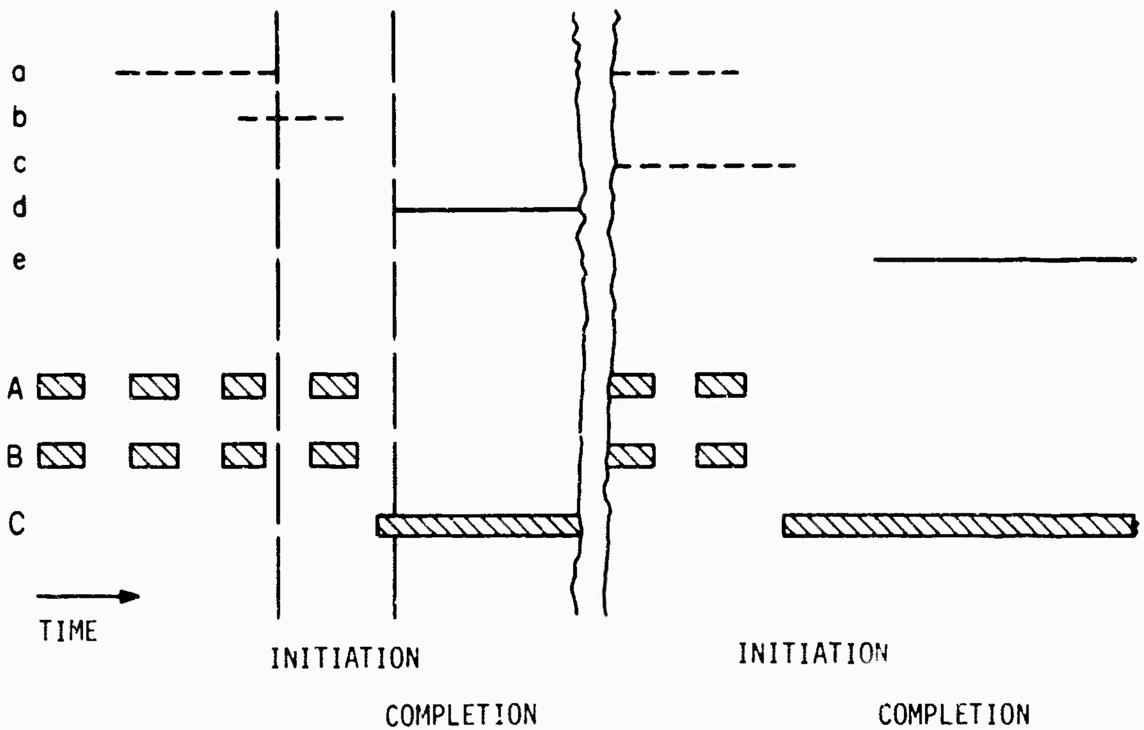


FIGURE 3. EXAMPLE OF A PROCESS SHOWING THE EXISTENCE OF SIGNALS ON CONTROL AND DATA PATHS

2.2.1 SEQUENTIAL PROCESS NETWORKS

2.2.1.1 SYNCHRONOUS AND ASYNCHRONOUS SEQUENTIAL PROCESSES

Sequential processes occur one after another in some prescribed order. Sequential processes may be of two types: synchronous and asynchronous. Synchronous processes are initiated at definite instances in time by control signals from a *clock*. The completion signal of a synchronously controlled process is of no consequence, since the initiation proceeds strictly by the clock, regardless of whether the previous process is complete or not. Consequently, the initiation signals produced by the clock must be spaced far enough apart to allow the preceding process to be completed. If the period of activity of a process is variable, then the clock interval must be at least as large as the maximum period. If the variation in the length of activity of a process is great, and the length tends to be much less than the maximum a large percentage of the time, then there is a considerable length of time where the system is idle. Asynchronous sequencing can be introduced to minimize this idle time.

In asynchronous sequencing, the completion signal of one process is used to initiate the next process. The sequencing continues in a chain-like manner, and there is no idle time between completion of one process and initiation of another.

2.2.1.2 CONCEPTS ENCOUNTERED IN SEQUENTIAL ASYNCHRONOUS NETWORKS

Introduced now will be some terms which describe asynchronous sequential processes and networks. Any process may consist of subprocesses which, themselves are processes. The *subprocesses* communicate among themselves with the same types of signals and also accept and create signals outside of the process. A process is said to be *separable* if its only control paths are a single initiation path and a single completion path. Thus, the effect of a separable process is strictly transformation of data. A separable process will be represented by a rectangle.

A *null process* is a separable process which has no effect on data. A null process is identical to a single control path. An asynchronous sequential network may simply be a chain of separable processes which is also a separable process. It may also be more complex if decisions and merges are introduced as described below.

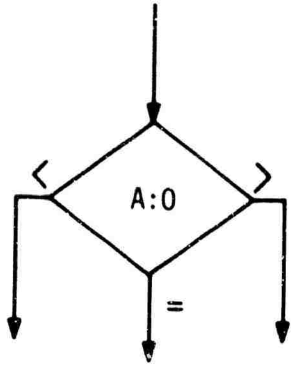
2.2.1.3 DECISION AND MERGE PROCESSES

In networks with synchronous control, certain values of data may cause certain processes not to be initiated. Selective initiation is accomplished in asynchronous networks by a special process, the decision.

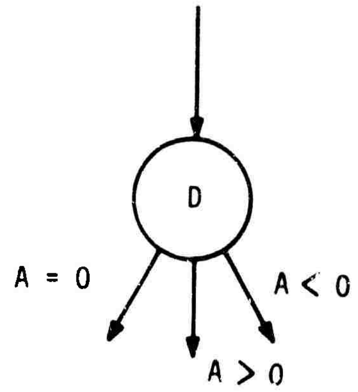
A *decision* is defined as a process with a single control input path but multiple control output paths. An output control signal is produced or only one of these paths. The path selected depends on data input signals. Thus, the decision process *decides* on which path the completion signal will be created. A decision with n output paths will be termed an *n-way decision*.

The decision is usually represented on a conventional flowchart as a diamond shape with a specification of the way in which a choice is made indicated inside the diamond and on the control output paths. The data input paths are usually implicit. In analyses where the data is not of concern, a decision may simply be represented as a circle containing the letter *D*. See Figure 4 for both of these representations. The introduction of decisions produces control signals which will exist on only one of a number of possible paths To recombine these possibilities into a signal on only one path, the merge element is required.

A process is said to be an *n-way merge* if it has n input signal paths and a single output signal path, and has the property of creating its completion signal upon acceptance of an initiation signal on *any* control

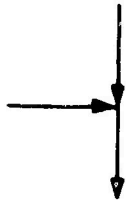


3-WAY CONVENTIONAL FLOWCHART

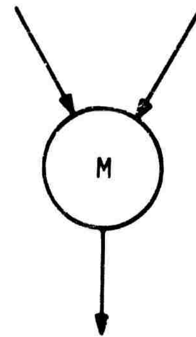


3-WAY, THIS RESEARCH

FIGURE 4. REPRESENTATION OF DECISIONS



CONVENTIONAL



THIS RESEARCH

FIGURE 5. REPRESENTATION OF MERGES

input. The merge is represented by a circle containing an *M*, as in Figure 5. Merging is shown on a conventional flowchart as simply the junction of two paths.

2.2.2 AN EXAMPLE OF ASYNCHRONOUS SEQUENTIAL CONTROL

Figure 6 shows a separable process which computes n! from a memory element which has recorded n($\geq$1) and puts the result into a second memory element. In Figure 7, subprocesses are shown which achieve the result using elementary processes which assign, add, and multiply the values of data signals to produce an output data signal. The numeric value *1* is assumed to be built into the processes requiring it. Upon accepting the initiation signal, the separable process proceeds autonomously until the computation is complete at which time the result will be recorded and the completion signal produced.

2.2.3 CONCURRENT PROCESS NETWORKS

Introduced here is the concept of concurrent processes, in which the restriction of a strict sequence, as in sequential process networks is removed. The synchronous mode applies also to concurrent processes. In fact, the processes at the logic level in conventional computers are synchronous concurrent. However, the problem at the program or organizational level in requiring that processes be synchronous is the same as for the sequential case; namely, that there is generally a large amount of idle time.

In considering ways in which processes may concur, certain restrictions must be observed, First, there must be a definite ordering between certain pairs of processes, i.e., one must occur before the other. Second, certain sets of processes may not occur at the same time if the data signals of one process are required for use by the second process. This is due to the required use of the input signals to a second process. Examples of this may be found in Figure 7, the n-factorial example.

The ordering between two processes may be expressed as a binary relation, <. If A and B are two processes, then A < B means *A must precede B*. A relation of this sort is known as a predecent relation.[30] If neither A < B nor B < A, then A and B may concur, which will be written A≡B.

Two processes that will be used specifically for the control of concurrent processes are now introduced. These are the branch and the rendezvous. Suppose there are three separable processes T, U, and V which must occur with the following restrictions: T < V and U < V. T and U may be allowed to concur, but both must precede V. To do this, a process known as an *n-way branch* is introduced. It involves control signals only, with one control input and n control outputs. When the initiation signal is accepted, output control signals are created on *all n* of the output control paths. Using a two-way branch with connections from the output control paths to the input control paths of T and U, an initiation signal applied at the input of the branch causes the concurrent activation of both T and U.

It is required that both T and U be complete before initiating V and for this purpose the rendezvous process is introduced. A process is an *n-way rendezvous* if it has n control inputs, a single control output, and no data paths. An initiation signal may occur on any one of the input paths, but no completion signal is given until all signals have been accepted on all input paths. By connecting the completion paths of T and U to a 2-way rendezvous, the rendezvous does not report completion until both T and U are complete.

A schematic of this entire network is presented in Figure 8. The branch and rendezvous are represented by circles with the letters B and R, respectively. The examples demonstrated so far have been simple. Even with only four elements: branch, decision, rendezvous, and merge, together with separable processes, the networks which may be constructed can be quite complex. This will be supported in Chapter Four, where it is shown that errors may be inadvertently introduced when implementing these concurrent process networks, and automatic means of detecting them are investigated.

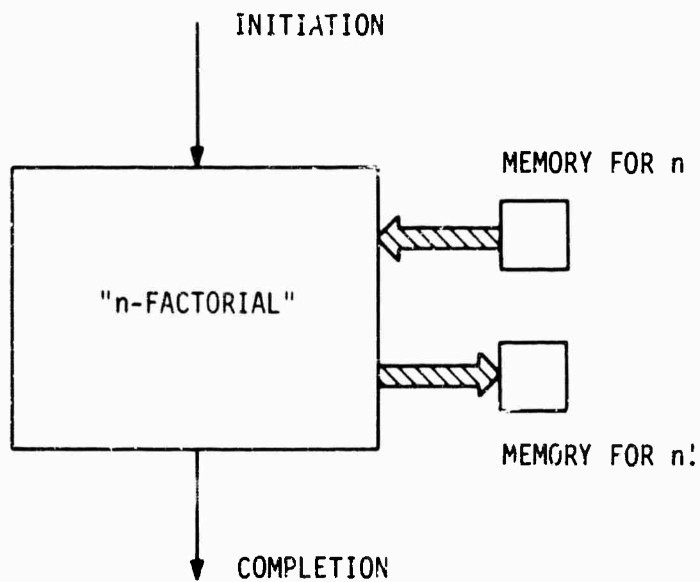


FIGURE 6. SEPARABLE PROCESS WHICH COMPUTES n-FACTORIAL

INITIATION

1 → A
1 → B

MERGE

M

A × B
→ B

A + 1
→ A

A < n

D

DECISION

A = n

B → n!

A

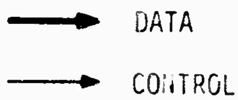
B

n

n!

COMPLETION

FIGURE 7. POSSIBLE SUBPROCESSES FOR FIGURE 6



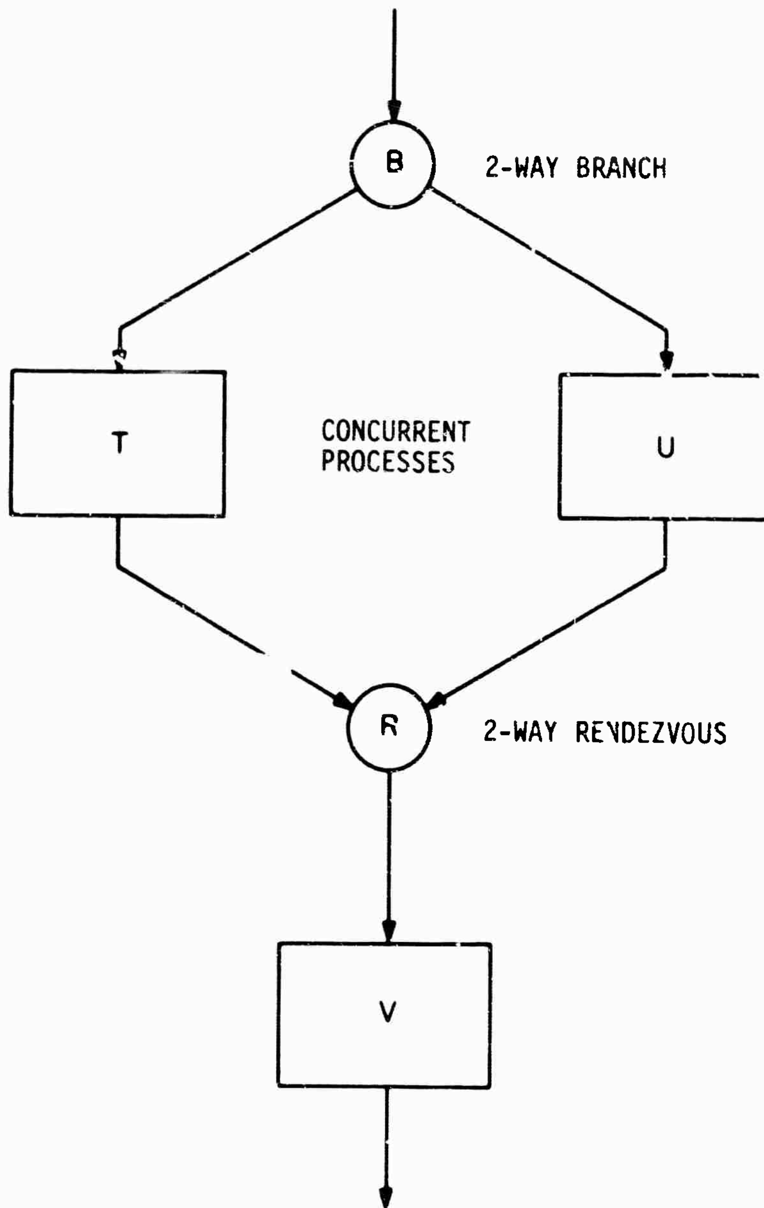


FIGURE 8. CONCURRENT ASYNCHRONOUS CONTROL OF PROCESSES T, U, AND V WITH THE PRECEDENCE RELATION $T < V$, $U < V$

2.3 APPLICATION TO COMPUTING SYSTEMS

It has been stated that the model presented applies to computers at both the organizational level and the program level. The validity of this statement will now be demonstrated.

2.3.1 APPLICATION AT THE ORGANIZATIONAL LEVEL

The terms *branch, rendezvous,* and *merge* are from the description of *macromodular systems* by Clark, et al.[13,32,39] The model was strongly influenced in other ways by macromodular systems, since these systems present what is probably the first major steps in separating the functions of processes at the organizational level from the engineering details of these processes.

A macromodular system has electronic units known as *macromodules.* These correspond to the processes in the model. There are also *data cables* and *control cables* corresponding to the signal paths of the model. Control and data signals are electrical signals on the cables.

A few specific types of macromodules will now be mentioned. The memory modules are of two forms: registers, constructed of flip-flops, and core memory. Associated with registers are several function units which perform logical, arithmetic, and shift operations. There are also *gates* which transfer data between registers. All of these processes are separable.

Affecting control are branch, rendezvous, and merge units, the function of which is identical to the corresponding two-way units of the model. The decision process appears in two forms: a *detector* which compares signals from two registers under a *mask* signal from a third register and creates a control signal on one of two control output paths, depending on whether or not the comparands are equal under the mask; and a *decoder,* which decodes three bits of data signal to select one of eight possible control output paths.

Since the control cables for any process may be wired into only one sequence, *call units* are provided. Call units effectively allow a process to be used as a subprocess within several different processes.

2.3.2 APPLICATION AT THE PROGRAM LEVEL

Several techniques have been proposed for the inclusion of explicit concurrent control into procedure oriented program languages.[33-41] This area is usually found in the available literature classified by terms such as *parallel programming, multiprocessing,* and *multiprogramming.* The control of processes asynchronously at this level is accomplished in various ways, the discussion of which is not pertinent here. The general scheme may be described as two or more processing units executing instructions simultaneously and communicating via a common core memory. Examples of existing and proposed machines for this purpose may be found in [42,43,44].

The languages utilized are similar to Fortran or Algol, with the addition of several statements which serve to specify concurrency. The control signals in the model correspond to the sequencing of instructions in these languages and the flow of data corresponds to assigning values to variables. One type of instruction introduced is analogous to the branch. This is the *FORK* instruction, by which a label is given instructing the computer to begin a concurrent sequence at that statement with the label. The statement corresponding to the rendezvous is written as *JOIN,* indicating that the control of the sequence containing the join statements referencing a particular label will meet at a statement with that label. The corresponding machine language instructions to accomplish this have also been described in the referenced literature. A less flexible method, which is equivalent to requiring that all concurrent process existing at once be controlled by the same branch-rendezvous pair, has been suggested using the statements *DO TOGETHER, AND* and *PARALLEL FOR.* This specifies that certain sequences are to be executed concurrently, e.g., the *DO-group* of Fortran or the *block* in Algol. The reason these schemes are less flexible is there can be no transfer out of the sequences or among them.

A third and more flexible way has also been proposed and has been included in the definition of a language which is currently being implemented.[41] This technique may be described as using certain special data signals which may be called *flag* or *semaphore* quantities, or *events*. Briefly, flag quantities may be tested for a particular value and depending on that value, the completion of the testing process may be reported or it may be delayed until the flag does assume that particular value. This, coupled with the ability to *terminate* control (i.e., destroy it without creating any other control signals), may be used to function as the rendezvous or in several other ways which are generally unachievable with only the branch, rendezvous, decision, and merge elements.

3. GRAPH THEORETIC CONCEPTS

In discussing networks of processes, it is desirable to have a concise language available for describing them. Since an automatic analysis of networks is sought, it is also desirable to have a convenient way of representing such networks to a computer. The branch of mathematics known as *graph-theory* is well-suited to this purpose.

Thorough discussions on the theoretic aspects with some applications are given in Berge[45], Ore[46], and Harary, et al.[47]. Applications of graph theory to processes in digital computers may be found in [24-26,48-59].

3.1 DEFINITIONS

The definition of a graph, as presented here, is similar to Harary's definition of a *net*. A *graph*, P, is a system (N, A, f, g) where

N is a finite set of elements called *nodes*
A is a finite set of elements called *arcs*
f is a mapping of A into $N \vee \{\phi\}$
g is a mapping of A into $N \vee \{\phi\}$
ϕ is a special element distinct from any element of N.

A graph may be schematically represented by a diagram as shown in Figure 9 which immediately suggests its usefulness in describing the interconnection of processes. The arrows represent the arcs and the circles represent the nodes. The functions f and g are defined as follows:

Let c be an arc, n be a node. Then
f(c) = n if and only if the head of c connects to n.
c is then said to be an *input arc* with respect to n.

g(c) = n if and only if the tail of c connects to n.
c is then said to be an *output arc* with respect to n.

If either f(c) = r or g(c) = n, then c is said to be *incident with* n.

The functions f and g for the graph in Figure 9 are defined below the graph.

If n and m are two nodes and c is an arc such that n = g(c) and m = f(c), then n is said to *connect to* m while m is said to *connect from* n. In either case, n and m are said to be *connected*. The arc, c, may be represented by an ordered pair of nodes (m,n).

The *out-degree* of a node, n, is the number of arcs, c, for which f(c) = n. The *in-degree* of a n is the number of arcs, d, for which g(d) = n.

If b and c are nodes, there is said to exist a *semipath* between b and c if one of the following holds:

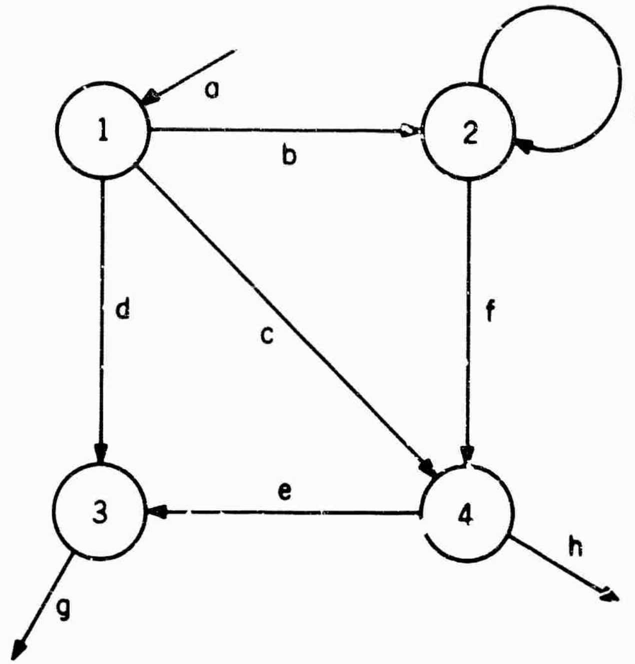
1. b and c are connected
or 2. b is connected with a node d and there is a semipath between d and c.

If b and c are nodes, then there is said to exist a *path from* b *to* c if one of the following holds:

1. b connects *to* c
or 2. b is connected to a node d and there is a *path from* d to c.

In this case c is said to be *reachable* from b, or b *reaches* c.

Path should not be confused with *signal path* from Chapter 2.



$N = (1, 2, 3, 4)$

$A = (a, b, c, d, e, f, g, h, i)$

$g(b) = g(c) = g(d) = 1$

$g(f) = g(i) = 2$

$g(g) = 3$

$g(e) = g(h) = 4$

$g(a) = \emptyset$, THE SPECIAL ELEMENT

$f(a) = 1$

$f(b) = f(i) = 2$

$f(d) = f(e) = 3$

$f(c) = f(f) = 4$

$f(g) = f(n) = \emptyset$

FIGURE 9. EXAMPLE OF A GRAPH

A graph is said to be *weakly connected* if between any two nodes there exists a semipath. The graph of Figure 9 is weakly connected. Similarly, a graph is said to be *strongly connected* if between any two nodes there exists a path. Thus, a strongly connected graph is weakly connected but the converse does not necessarily hold. In cases where the converse does not hold, the graph is said to be *strictly weakly connected*.

A *subgraph* of a graph P = (N, A, f, g) is a graph, Q = (N′, A′, f′, g′), where N′ is a subset of N and A′ is the set of arcs incident with the nodes N′. Thus f′ and g′ are restrictions of the mapping f to A″ where $c \in A''$ if and only if $f(c) \in N'$ and g to A‴ where $d \in A'''$ if and only if $g(d) \in N'$.

An arc of a subgraph is said to be *input* with respect to that subgraph if it is input to some node in that subgraph but is *not* output to any node in that subgraph. An arc of a subgraph is said to *output* with respect to that subgraph if it output to some node in that subgraph but is not input to any node in the subgraph.

A subgraph is defined to be *separable* if it has only one input arc and one output arc.

A subgraph is said to be *minimal* of a property L if the removal of any connected node results in a subgraph which does not have property L. A subgraph is said to be *maximal* of a property L if the addition of any connected node results in a subgraph which does not have property L. Thus a maximal strongly connected subgraph is one in which the property of strong-connectedness is lost when any node connected to the subgraph is added.

A node is said to be self-connected if it connects to itself.

A set of arcs $(a_1, a_2 \cdots a_n)$ are said to be *parallel* if $f(a_1) = f(a_2) = \cdots = f(a_n)$ and $g(a_1) = g(a_2) = \cdots = g(a_n)$. Examples are shown in Figures 10 through 13.

3.2 MATRIC REPRESENTATION

Matric notation has been shown to be a convenient representation for graphs, especially if the matrices are to be manipulated by computer.

The mappings f and g may be represented by allowing each column of a matrix to correspond to a node and and each row to an arc, and letting the (i,j)th entry assume the value *1* if the arc corresponding to row i maps into the node which corresponds to column j. Denote by F and G the matrices for the mappings f and g. F and G will be respectively called the *input* and *output* matrices. The matrices F and G for f and g of Figure 9 are shown in Figure 14.

Other useful matrices may be derived from the input and output matrices. The first, known as the arc-node matrix, A, can be used to represent both F and G provided that there is no arc which is self-connected. By definition A = F − G. If no nodes are self connected, there will be no entries in F and G which are both *1*, but if there are nodes which are self connected, there will be such entries. Identical entries result in the corresponding entry of A being *0* which is indistinguishable from no connections at all to that particular node. If such entries do not occur, F and G can be obtained from A. Figure 15 shows the arc-node matrix for the graph in Figure 9. Notice the (i,2) entry.

An interesting algorithm is presented by Wann[24] using the arc-node matrix, A, in testing a subgraph for separability. It may be stated as follows: A subgraph consisting of a particular set of nodes is separable if and only if the sum of the corresponding columns of the arc-node matrix contains a single +1 and a single −1 entry.

Two other matrices which may also be computed from F and G are as follows:

The *node-node* or *connection matrix*, $C = {}_tGF$ (where *t* indicates *transpose of*) defined by $C_{i,j}$ = the number of arcs input to node j and output from node i.

The *arc-arc matrix*, $D = F_tG$, defined by $D_{i,j} = 1$ if arc i is input to a node from which arc j is output, and 0 otherwise.

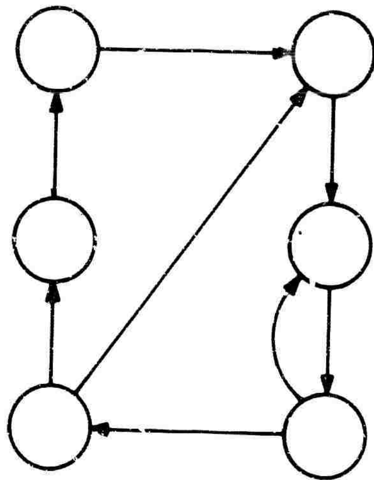


FIGURE 10. A STRONGLY CONNECTED GRAPH

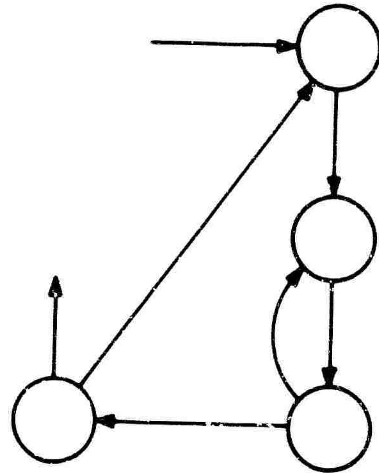


FIGURE 11. A SEPARABLE SUBGRAPH OF THE GRAPH IN FIGURE 10

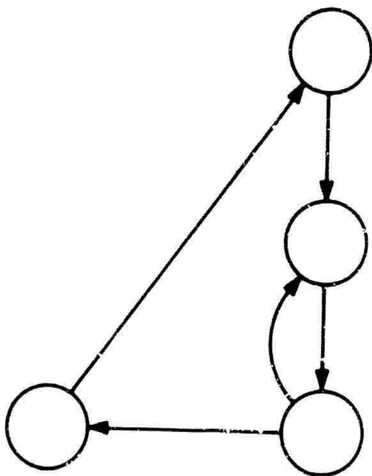


FIGURE 12. THE MAXIMAL STRONGLY CONNECTED SUBGRAPH OF THE GRAPH OF FIGURE 11

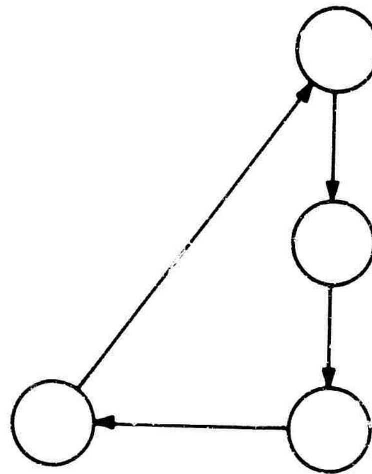


FIGURE 13. A MINIMAL STRONGLY CONNECTED SUBGRAPH OF THE GRAPH OF FIGURE 12

$$
F = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{array}
\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\
\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0
\end{array}\right] \end{array}
$$

$$
G = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{array}
\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\
\left[\begin{array}{cccc}
0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0
\end{array}\right] \end{array}
$$

Figure 14. Input and output matrices for the graph of Figure 9.

$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{array}\right] \end{array}$$

Figure 15. Arc-node matrix for the graph of Figure 9.

Another matrix, the *reachability matrix*, $\bar{R}$ is defined as $R_{ij} = 1$ if there is a path from node i to node j, and 0 otherwise. The reachability matrix may be computed from the connection matrix as follows:

> C^1, the first power of C, gives, for any two nodes, the number of paths from one to the other of length 1. (The *length* of a path between two nodes being the number of arcs traversed in tracing from one node to the other.) It can be shown that C^n, the *nth* power of C, gives for any two nodes, the number of paths from one to the other of length n. Define a function, W, as $W(x) = 0$ otherwise. Then $(W(C^n))_{ij} = 1$ if there is any path from i to j of length n, and 0 otherwise.
> Thus $W(C) \vee W(C^2)$, where $\vee$ is the Boolean sum, gives all paths of length 1 or 2.
> Similarly, $\bigvee_{i=1}^{n} W(C^i)$ gives paths of length 1, or 2, or . ., or n. For any finite graph, all paths which are greater than a certain length, say p, necessarily include a loop, thus the Boolean sum $\bigvee_{i=1}^{p} W(C^i)$ will be identical to $\bigvee_{i=1}^{n} W(C^i)$ for any $n \geq p$. The point here being that to determine the reachability matrix, only a finite number of matrices need be summed.

An equivalent method for computing the reachability matrix, which is computationally more efficient, is given in [60] Other useful algorithms, such as one for the determination of strongly-connected subgraphs from the reachability matrix, are given by Ramamoorthy.[57] The connection matrix for the graph of Figure 9, and the construction of the reachability matrix are exhibited in Figure 16.

In succeeding sections, process networks will be represented by graphs, and graph-theoretic terminology will be employed in their descriptions. The analysis will be concerned mainly with control. Consequently, data paths will not be shown. The nodes of graphs will represent non-separable processes, particularly branches, merges, decisions, and rendezvous. The arcs will represent control signal paths. Arcs will also be used to represent separable processes, since a separable process has only one input and one output control path. The terms *graph* and *network* will be used interchangeably. The description of an arc as being *active* means that a signal exists on the corresponding control path.

$$C = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$C^2 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = W(C^2)$$

$$C^n = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = W(C^n) \qquad n \geq 2$$

$$R = C \vee C^2 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 16. The connection and reachability matrices for the graph of Figure 9.

4. ERRORS IN NETWORKS OF CONCURRENT PROCESSES WITH ASYNCHRONOUS CONTROL

4.1 GENERAL TYPES OF ERRORS

Process networks which are employed in the solutions of computational problems are generally separable. The solution begins with the introduction of an initiation signal to the separable process. The data, initially in memory elements, is inspected by the process and data produced indicating the results. At the completion of the solution steps, a single completion signal is produced.

Several types of errors may occur in such a separable process network. The general characteristic of an error is that the desired result is not produced

Errors may be classified into the ways in which they are produced:

1. A process physically malfunctions
2. The solution steps of an algorithm are incorrectly specified
3. Concurrent control is incorrectly specified

The first of these is of no concern here. The second will be called an *algorithm error*, but detection of this type of error will not be considered because of its general infeasibility. The third will be called an *implementation error* because it is introduced by implementation of an algorithm as a concurrent process network.

The following properties are postulated as being desirable *for separable processes*, the lack of them being an error:

1. *Finite duration* – After initiation, a separable process must complete within a finite period of time.
2. *Non-regeneration* – Once initiated, a separable process will create only one output control signal.
3. *Determinacy* – A separable process, for any activation, will always produce the same output data if the input data is the same.

Sequential networks are always non-regenerative and determinate, but may not be of finite duration if the control of iteration is specified incorrectly. In networks of concurrent processes, all of these properties may be lacking due to improper specification of concurrent control. As a clarifying point, it might be mentioned that such errors are dynamic. For some data, the network may function normally while for other data it may malfunction in different ways. A network will be said to have certain types of errors if it is *possible* for the network to malfunction in certain ways. The means by which each of these errors are introduced into networks is now investigated.

4.1.1 INFINITE DURATION

The name given to the error in process networks which do not complete in a finite length of time is *infinite duration*. It was mentioned that infinite duration may be due to an error in an algorithm for graphs such as in Figure 17. It is re-emphasized that strongly connected subgraphs, as in Figure 18, do not necessarily imply an error, but that data must be considered before determining if the network is in error. Again it is mentioned that such algorithm errors are infeasible to detect.

Infinite duration caused by introduction of concurrent control is generally the result of processes internal to the network which are not able to report completion. This is the case in a network in which only one input to a rendezvous ever becomes active, as in Figure 19.

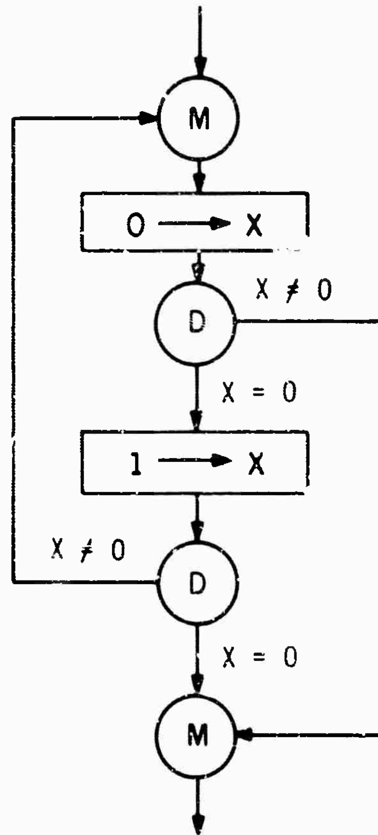


FIGURE 17. A NETWORK WITH AN INFINITE DURATION DUE TO AN ALGORITHM

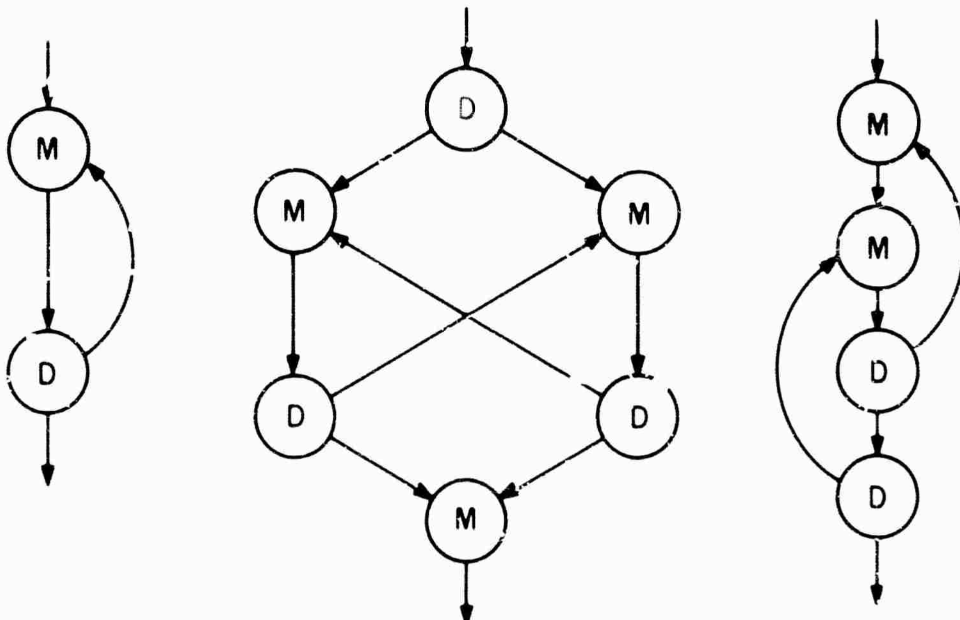


FIGURE 18. NETWORKS WHICH MAY HAVE INFINITE DURATION ERRORS BECAUSE OF INCORRECT ALGORITHM

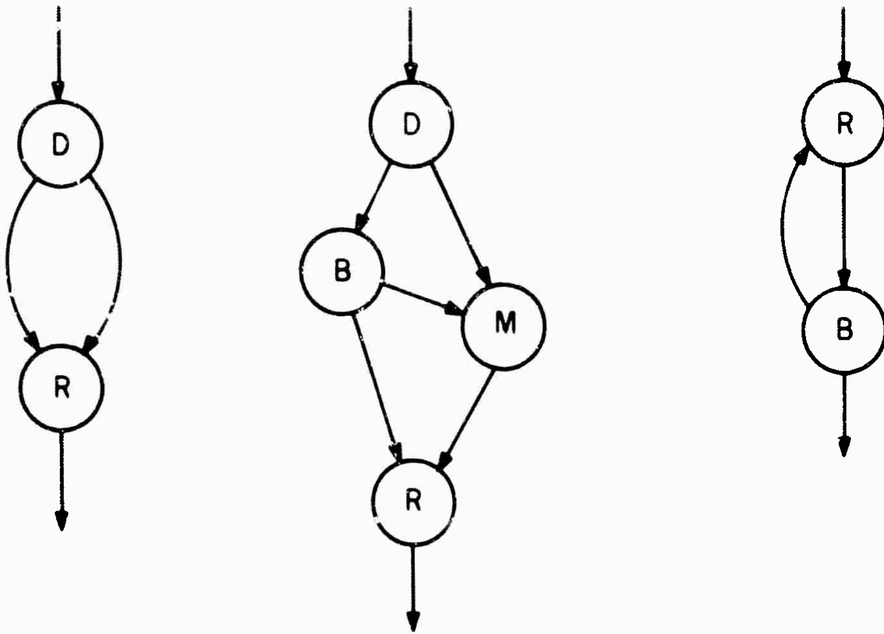


FIGURE 19. NETWORKS HAVING INFINITE DURATION BECAUSE OF INCORRECT USE OF CONCURRENT-CONTROL ELEMENTS

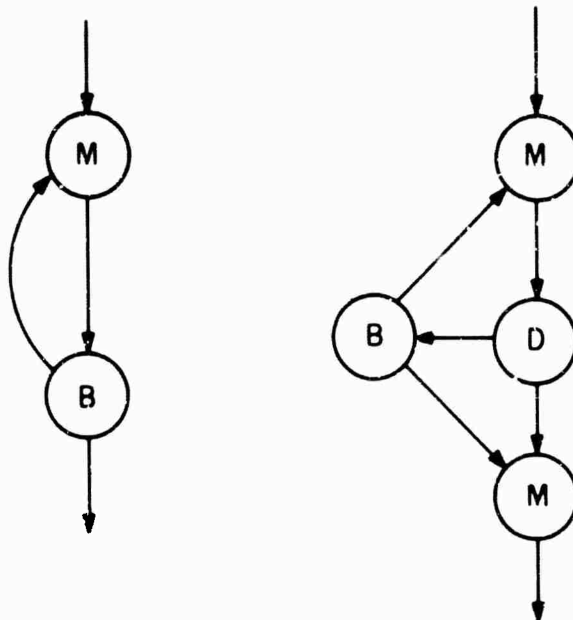


FIGURE 20. REGENERATION CAUSED BY A BRANCH WITHIN A STRONGLY CONNECTED SUBGRAPH

4.1.2 REGENERATION

Networks which are *regenerative* may produce multiple output control signals after being initiated only once. This may be caused in two ways. The first is by allowing a branch to produce an output from a strongly connected subgraph. This is shown in Figure 20. It should be mentioned that not *every* strongly connected subgraph with a branch implies an error.

Regeneration is also produced by what will be called a *hazard*, due to its similarity to the hazard in switching networks (cf. McCluskey.[61]) The hazard is found by consideration of the merge process. Suppose there is a 2-way merge with input arcs, a and b, and output arc c. The arc c is the output arc of a separable process, P, as shown in Figure 21. Suppose it is possible that a and b may have signals simultaneously. Because of this possibility, one of two phenomena may occur: (1) If a and b have signals which overlap in time, the merge receives two initiation signals and the result is unpredictable, since, by definition, a merge is initiated by a signal on only one of its input arcs. (2) If a and b do not overlap, the process P may report completion twice.

In summary, the possibility of more than one input signal to a merge may cause either of these problems, and will be identified as a hazard. The hazard is also responsible for producing indeterminacy, as will be seen in the following section.

4.1.3 INDETERMINACY

A network is said to be *indeterminate* if different output data are produced in two or more different activations of the network for the same input data. Three ways in which a network may be indeterminate are: (1) by the failure to observe constraints on processes, (2) by the process reporting completion with some residual control signals still present within the network, and (3) by the failure to observe precedence requirements in designing the control.

Failure to observe constraints on processes occurs when, as in the previous discussion of hazards, a process is initiated twice. Similarly, two consecutive signals to the same input of a rendezvous is a violation of the constraints for this process.

It is possible for a process to report completion only once but, for some control to remain active within the network. This occurs when a rendezvous has accepted a single control input and the network containing the rendezvous has reported completion. When the separable process is activated a second time, a control signal to the other input will cause the rendezvous to report completion. Thus, even though the data may be the same, the results could be different for two successive activations. Networks which may report completion while rendezvous remain active are said to possess *residual control*. Examples of residual control are shown in Figure 22.

Failure to observe precedence relations, as mentioned in section 2.2.3, may cause erroneous output data, even though the errors in control previously discussed are not present. Since no assumptions are made about relative times of processes in asynchronous control networks, a possible variation in time of processes which are active concurrently may cause varying results for identical data. For example, consider two processes, A and B, where $A < B$ is a requirement. If the network is implemented as in Figure 23, where C and D represent other processes unrelated to A or B, an assumption that C will last longer than A may not be supported, and the output data from A which is required for input to B may not have been set when B requires it. Thus, B may reference the data which was previously in memory elements, and erroneous results produced.

The error exemplified in the preceding paragraph may be detected by observing the possibility of A and B concurring whereas it is required that A precede B. A correct implementation appears in Figure 24. An error

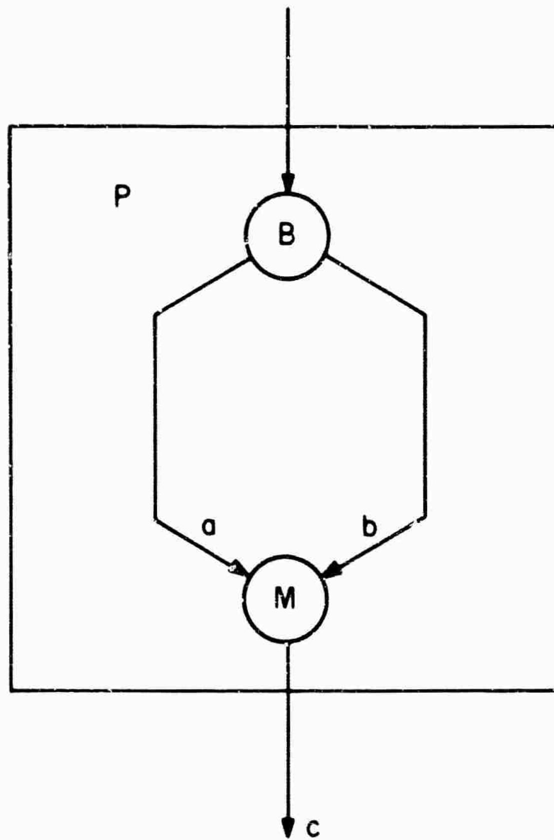


FIGURE 21. POSSIBLE REGENERATION CAUSED BY A HAZARD

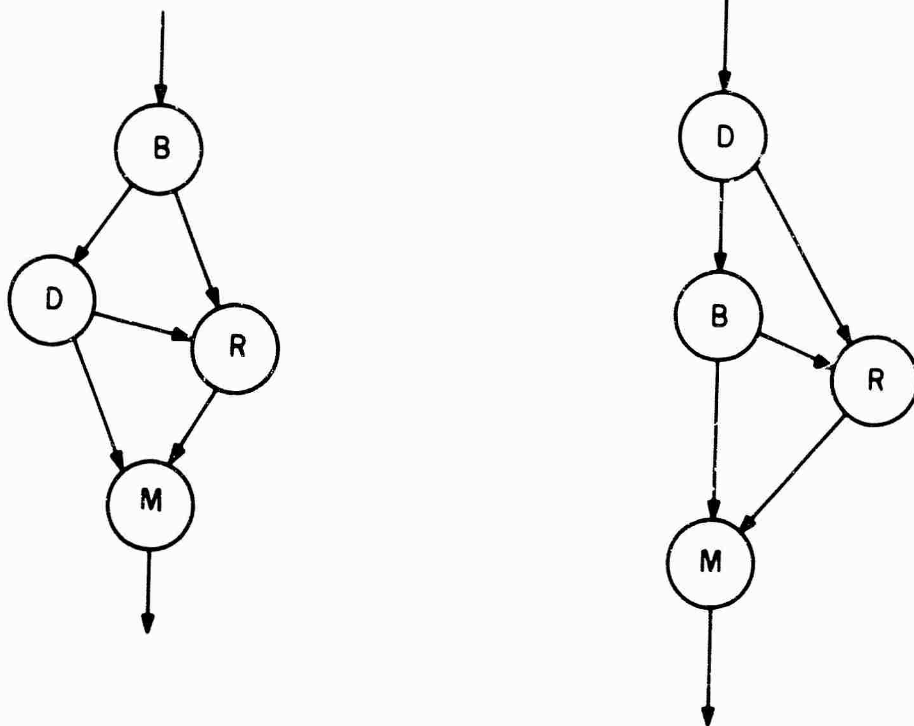


FIGURE 22. NETWORKS WITH RESIDUAL CONTROL

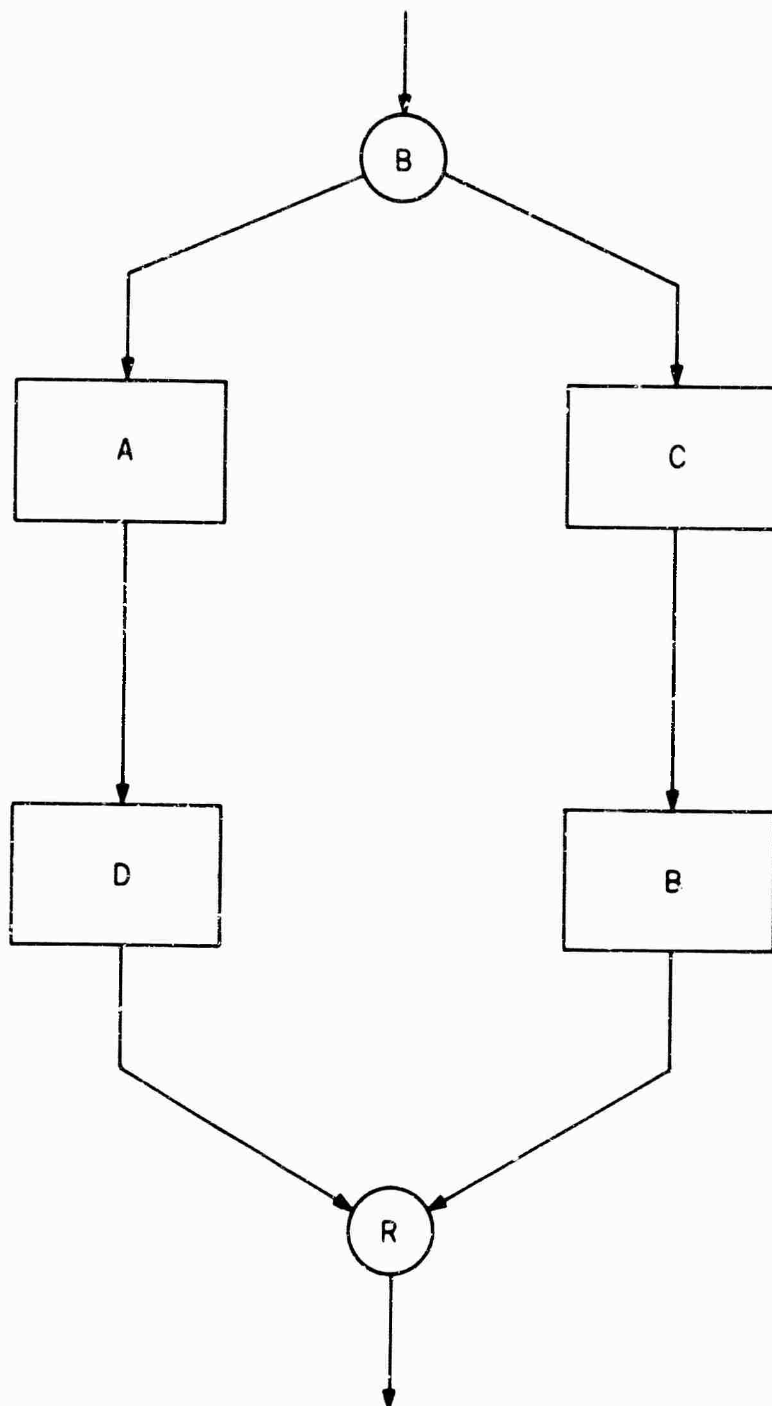


FIGURE 23. INCORRECT IMPLEMENTATION FOR A < B

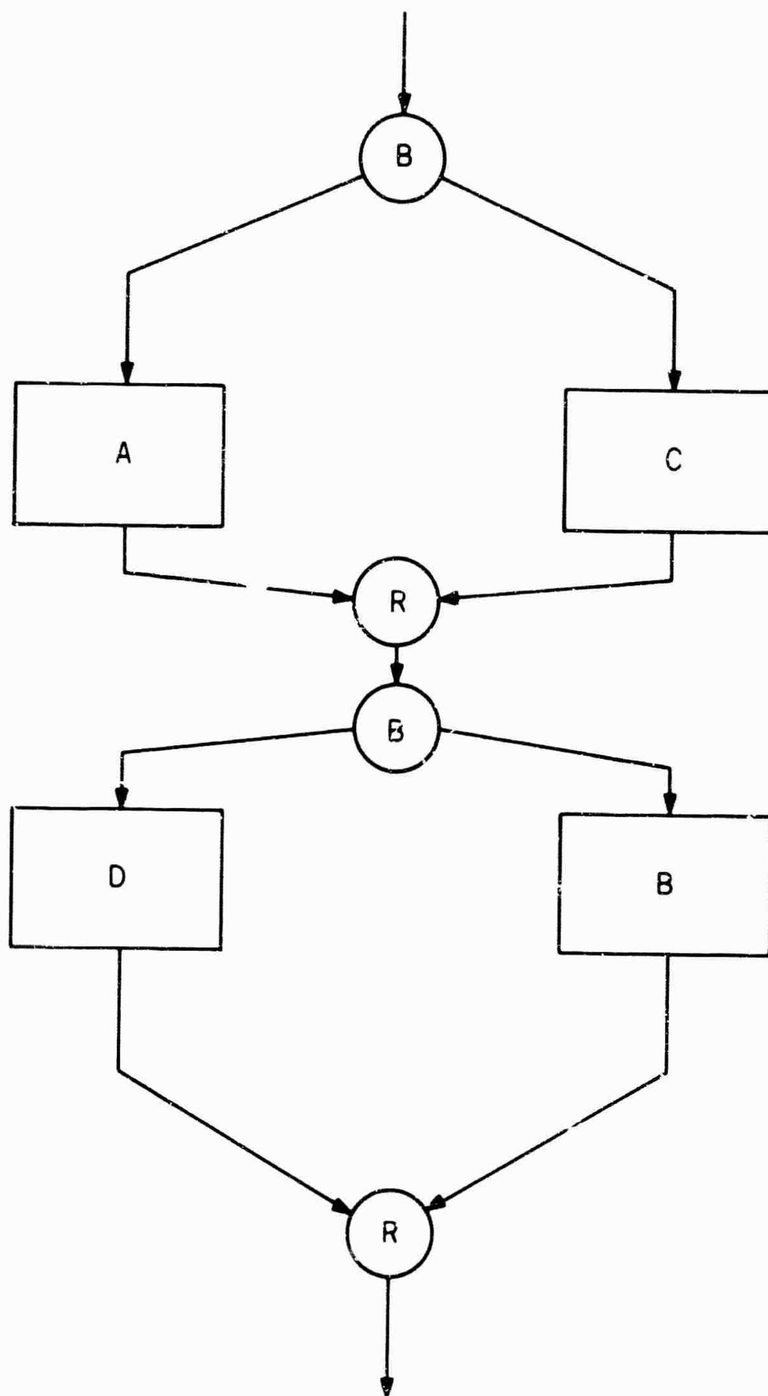


FIGURE 24. CORRECTION OF THE ERROR IN FIGURE 23

of this kind will be called a *race* because the time duration of the processes affect the results. The insertion of processes to eliminate races is called *interlocking*. This example has presented a very simple case of interlocking. More complex interlock schemes may be devised to allow more freedom and still meet precedence requirements. These are discussed by Littlefield.[62]

4.1.4 SUMMARY OF IMPLEMENTATION ERRORS

1. *Infinite duration* – The process does not complete within a finite time after initiation. Infinite duration is produced by the impossibility of completion of an active rendezvous.
2. *Regeneration* – The process produces multiple output control signals after a single initiation. Regeneration is caused by:
   a) Certain strongly connected networks with branches connecting to output arcs
   b) Hazards
3. *Indeterminacy* – The process produces erratic results. Indeterminacy is produced by:
   a) Hazards
   b) Residual control
   c) Races

In considering detection of errors in networks, it is helpful to regroup the sources of errors into those categories which are similar. The regrouping is shown below with short names provided for simplicity of discussion.

1. *Incomplete rendezvous* – the only source of infinite duration, and the source of residual control producing indeterminacy
2. *Reentered branch* – the branch in certain strongly connected subgraphs which produces regeneration
3. *Hazard* – the source of some regeneration and indeterminate cases
4. *Races* – the source of indeterminacy by violation of precedence requirements

For comparison, additional examples of error and error-free cases are shown in Figures 25 and 26. It should be noted that these errors display the imcomplete rendezvous, reentered branch, and hazard only. The race may appear in any network with concurrent processes. It should also be noted that no assumption is made about the dependency of various decision elements upon data.

4.2 DETECTION OF IMPLEMENTATION ERRORS

One method for detecting errors is to construct the network and perform a number of trial activations. *Construct* implies connecting the electronic units and making the necessary connections if the organizational level is being considered. Depending upon the flexibility of components, this task may be quite time consuming. At the program level, *construct* means writing the program and putting it into form for input to the computer system. The trial implementation has the advantage that algorithm errors as well as implementation errors may be checked. It has the following disadvantages:

1. Construction of the network is usually a lengthy task at the organizational level.
2. It is difficult, if not impossible, to devise trial data which tests the network with sufficient thoroughness.

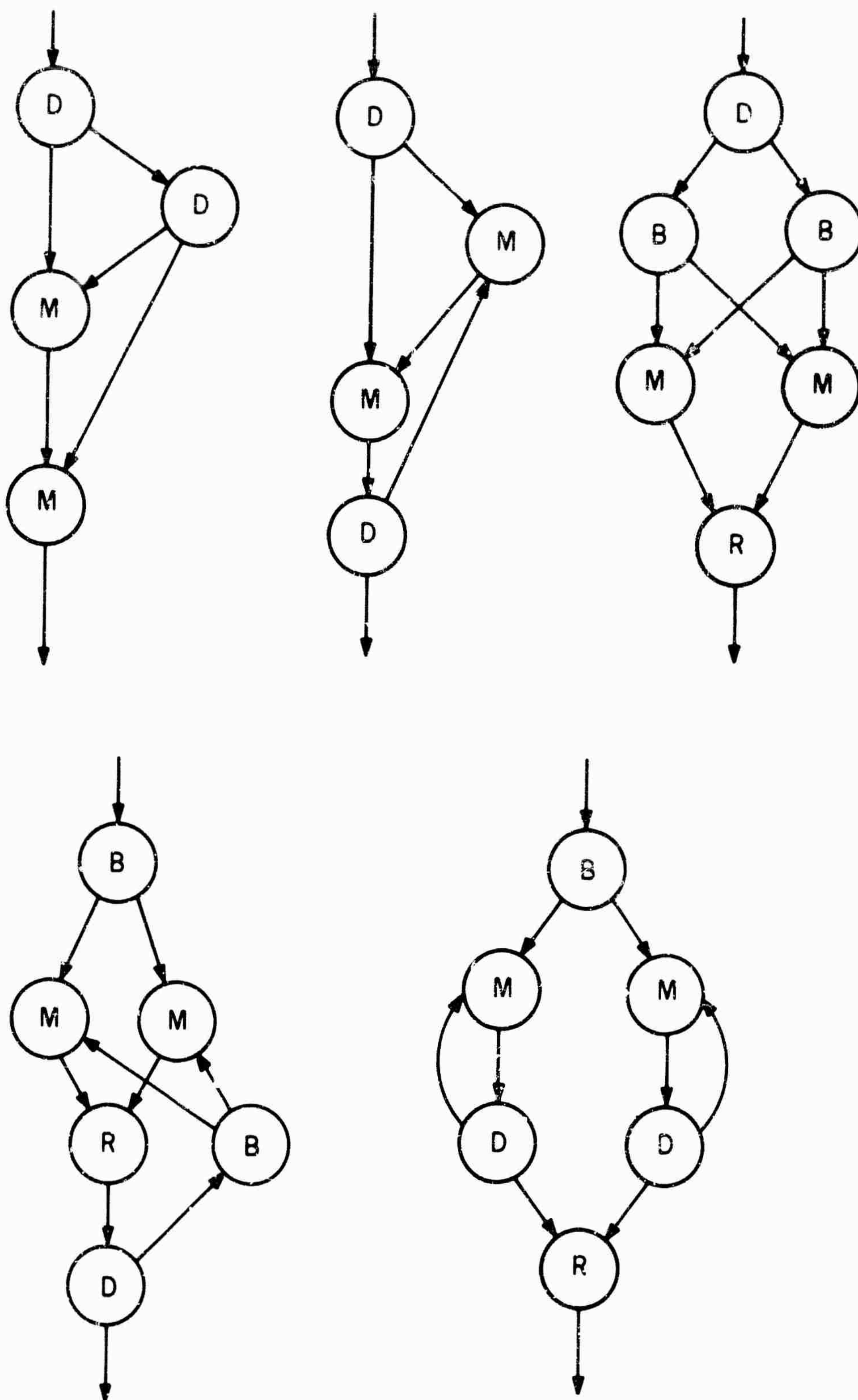


FIGURE 25. ERROR-FREE GRAPHS

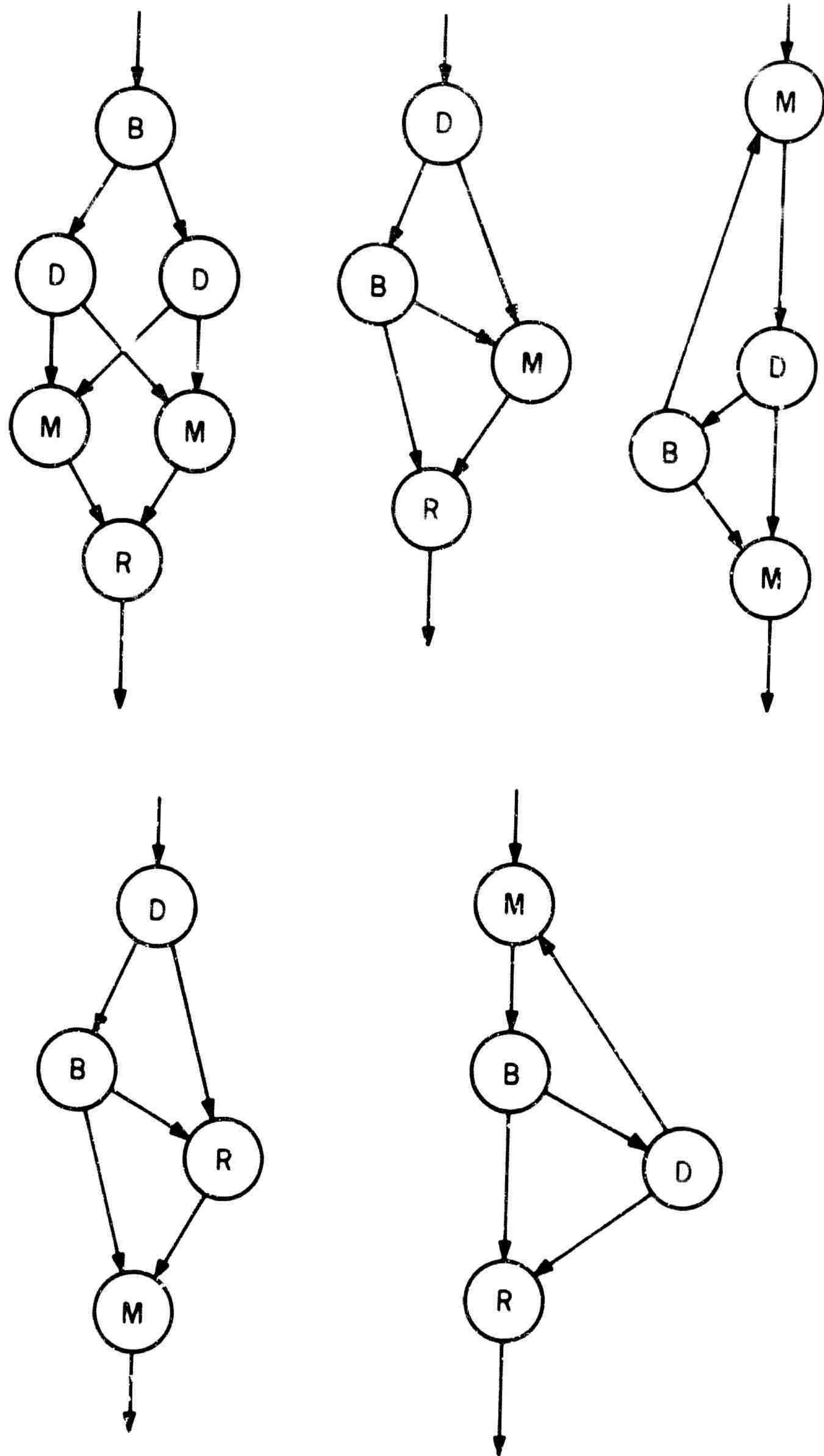


FIGURE 26. EXAMPLES OF GRAPHS WITH ERRORS

3. Errors due to races may not occur at all during a test but may occur during some subsequent use of the network.
4. The amount of time required to perform a sufficient number of tests may be prohibitive.

Some comment may be made as to what a *sufficient number* of tests implies. Regardless of how many trial activations are performed with consistent results, there is always a possibility of a race. Thus, a *sufficient number* implies that races are not being considered.

The term *combination* is used to indicate a particular set of arcs on which control signals appear during the activation of a separable network. It may be observed that, in an error-free network, there may be several unique combinations, the number of which depends upon the number of decisions and their degree. The maximum number of combinations is $\prod_{i=1}^{n}$ where d_i is the out-degree of the ith decision and n is the number of decisions. For instance, a graph is shown in Figure 27 with two 2-way decisions. Four combinations are shown in the accompanying diagrams. Figure 28 depicts a graph with two decisions and less than the maximum number of combinations. A lower bound on the number of combinations for a given number of decisions depends on the manner in which the nodes are connected. The configuration yielding the fewest combinations is the *tree-structure,* as shown in Figure 29. The number of combinations for such a tree is a complex function of the number of decisions and their out-degrees, cf. Iverson.[63] For a graph composed of n 2-way decisions, the bounds are n+1 and 2^n.

Another point that might be mentioned concerning combinations is that in graphs with strongly connected subgraphs, a fixed result for each decision does not always produce an output control, but if an alternate is provided for the second encounter of a decision, it may be possible to produce an output control. In this case, combinations are indicated as in Figure 30, where the number at the output arcs of the decision indicate the order in which the outputs are used. A second example of this is shown in Figure 31.

The point demonstrated by the above is that for networks of considerable complexity, the number of combinations may be very large. A practical example is the control network for a floating-point arithmetic unit[64] as shown in Figure 32.

In view of the disadvantage of trial implementation of networks, a method is desired which will test a network and which eliminates these disadvantages. A method which is suitable for implementation on a computer is also desirable. Four areas of approach have been investigated in this research.

1. Simulation
   a. Trial data test
   b. Monte Carlo test
   c. Exhaustive test
2. Topological analysis
3. Symbolic analysis using algebraic expressions
4. State transitions

These methods are described in the following sections.

4.2.1 SIMULATION

It is possible to simulate concurrent process networks on sequential digital computers and detect certain errors. The simulation of sequential processes is simple to accomplish since all that need be done is to implement a program to perform the desired data operations. The flow of control is the same as the execution of program steps. While decisions and merges are found in conventional programs for sequential computers,

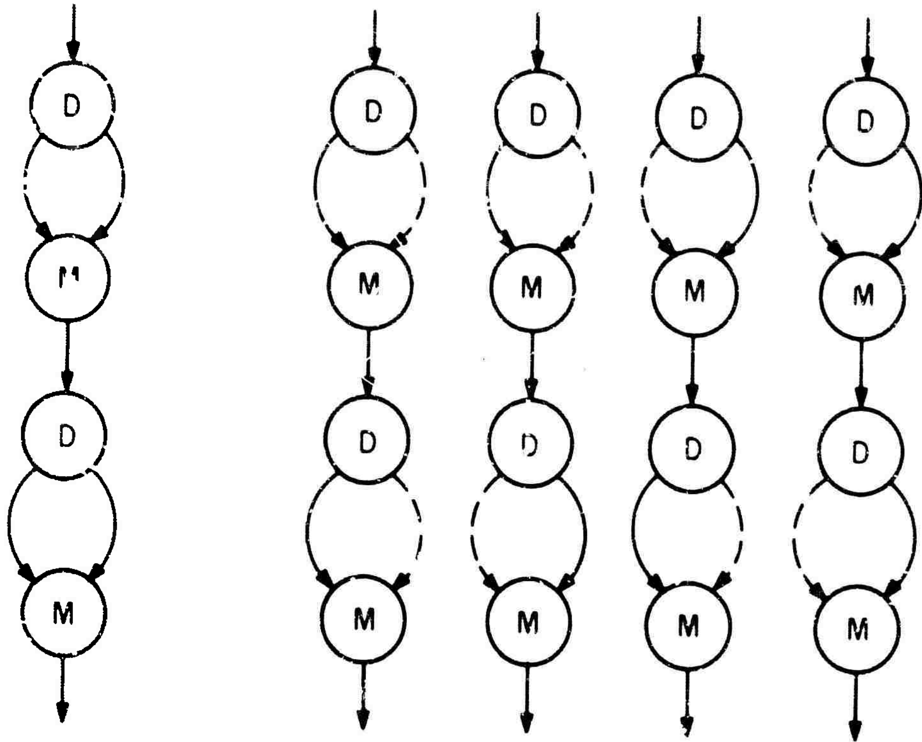


FIGURE 27. A GRAPH DISPLAYING THE MAXIMUM NUMBER OF COMBINATIONS FOR TWO DECISIONS

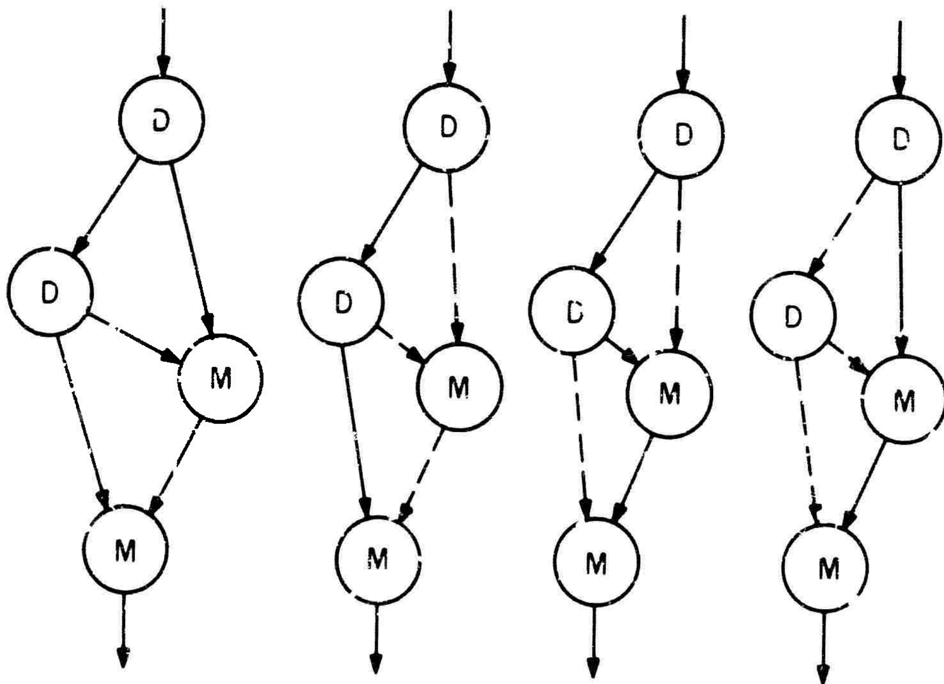


FIGURE 28. A GRAPH WITH TWO DECISIONS AND LESS THAN THE MAXIMUM NUMBER OF COMBINATIONS

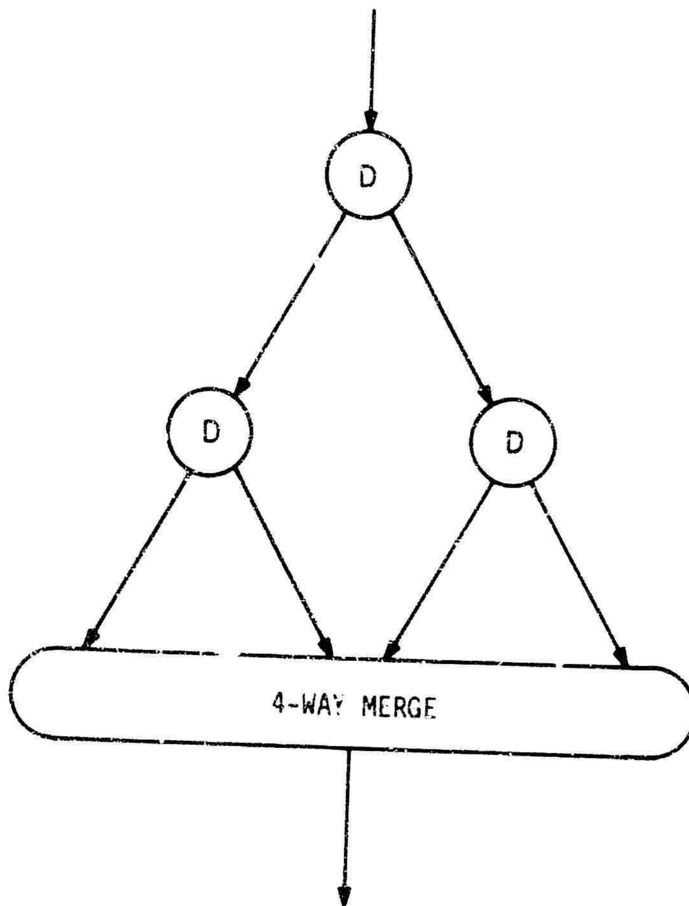


FIGURE 29. A 3-DECISION GRAPH WITH FOUR COMBINATIONS

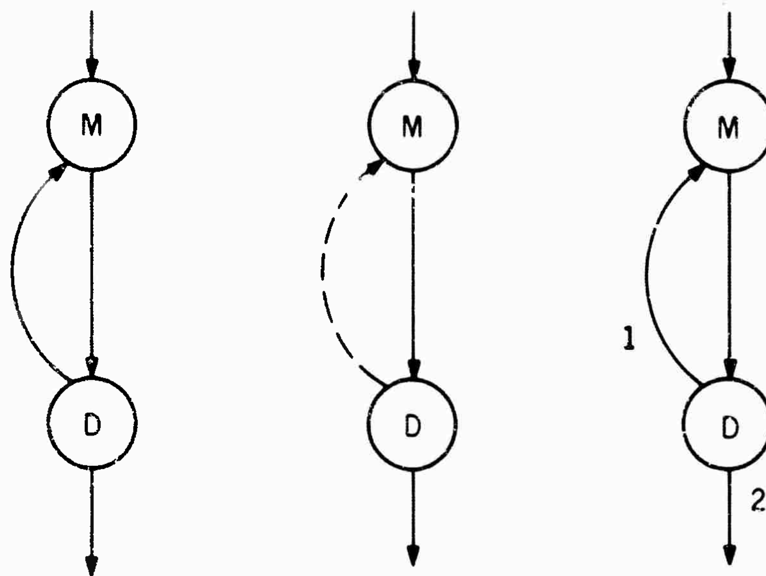


FIGURE 30. INDICATION OF COMBINATIONS FOR A STRONGLY CONNECTED GRAPH

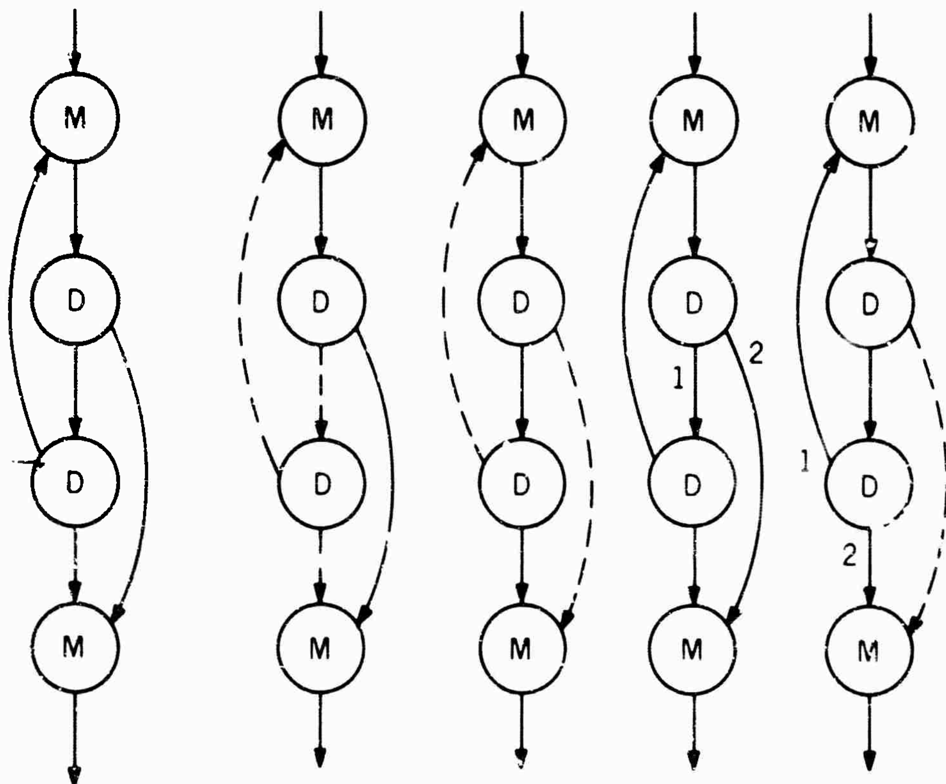


FIGURE 31. COMBINATIONS FOR A GRAPH WITH A STRONGLY CONNECTED SUBGRAPH

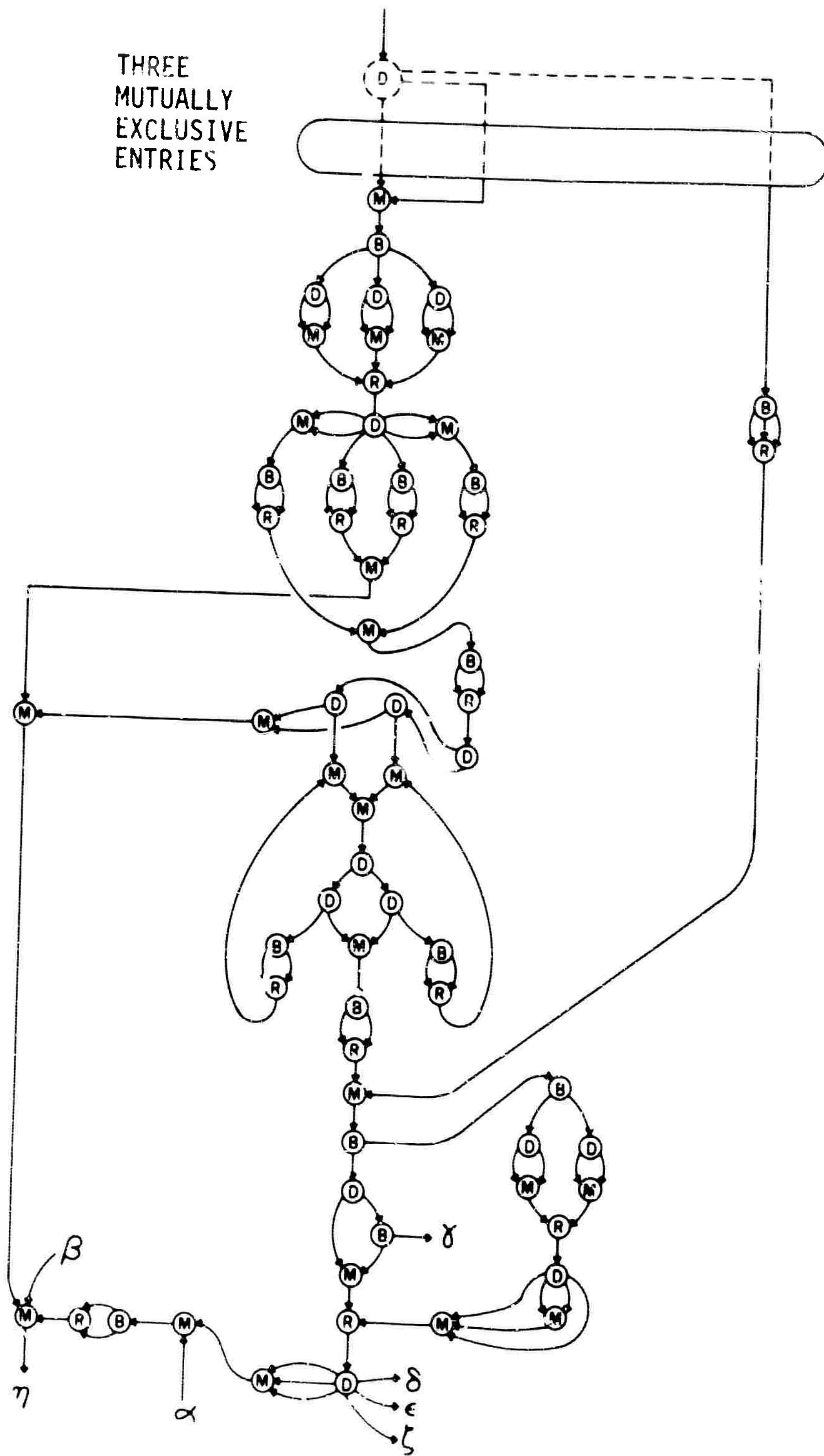


FIGURE 32. CONTROL FOR A FLOATING-POINT ARITHMETIC UNIT (CONTINUED ON FOLLOWING PAGE)

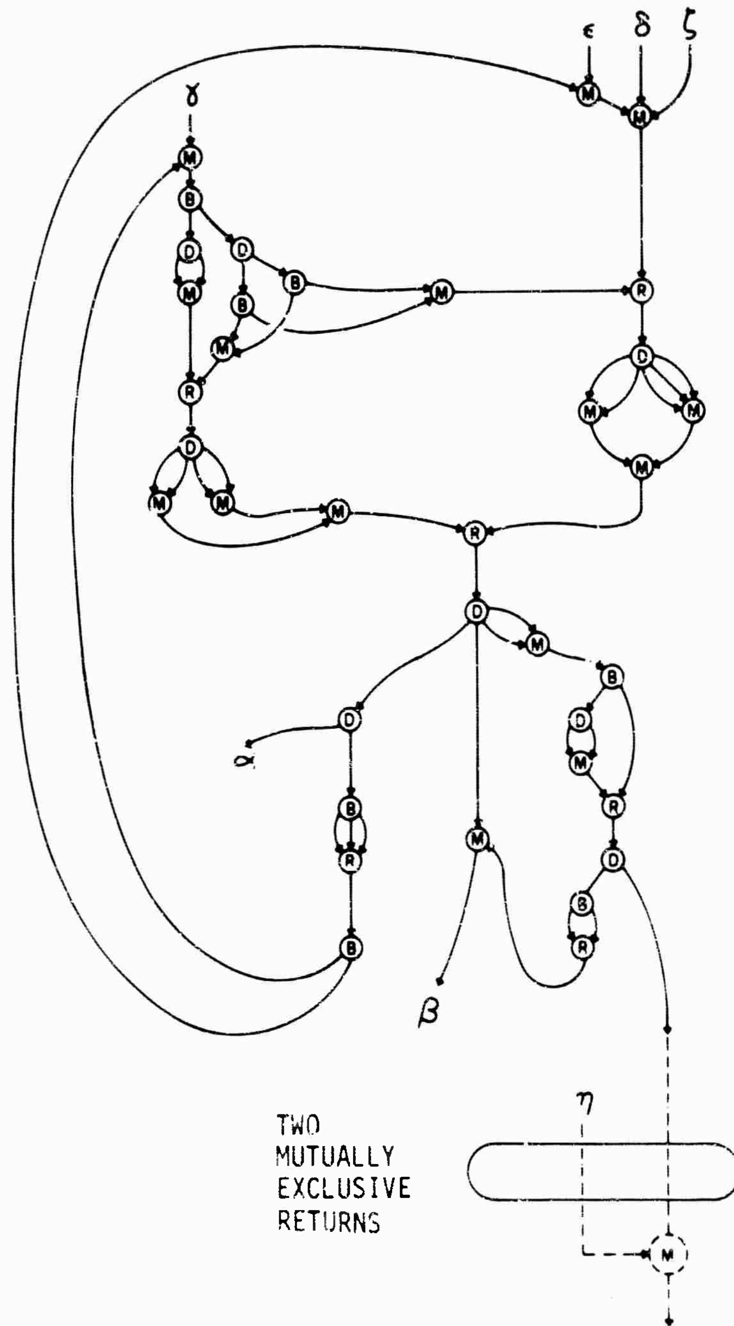


FIGURE 32(CONTINUED).

the branch and rendezvous are not. To simulate branch and rendezvous, certain records must be kept. n bits are associated with each n-way branch or rendezvous. Each bit corresponds to a particular output arc for a branch or input arc for a rendezvous. All bits are initially zero. When a branch unit is encountered during simulation, one arc is selected on which simulated control is to proceed. The bits corresponding to the other arcs of the branch are set to 1. When a particular input to a rendezvous is encountered, the bit corresponding to that input is set to 1. Next, all of the other bits of the same rendezvous are compared. If all are 1, then all inputs to the rendezvous have been accepted and the bits are reset to zero, the control proceeding to the output of the rendezvous. If all bits are not 1, then control cannot proceed to the output. Instead, the bits of branches are checked until one is found which is a 1, indicating that control may proceed on the corresponding arc. The bit is then reset to a 0.

When a simulated signal is present on the input to a merge, the control simply proceeds to the output. Similarly, after the output arc is selected by a decision, control proceeds to that arc.

Error checking in simulation will now be described. Some races may be detected by varying the order in which output arcs are chosen at a branch, but a test of all of these ways for every combination is infeasible.

All hazards are not checked, because this too would require stepping the control through the network in every possible way for each combination. Reentered branches may be found by examining the bits corresponding to the branch or rendezvous when encountered by control and this may indicate a hazard or regeneration. Hazards cannot be located by checking for reentered merges or decisions because this reentry is perfectly legitimate, as in strongly connected subgraphs. Thus, some hazards will escape detection.

Two types of errors may be checked when simulated control proceeds to the output arc of the graph in question. At this time, all branch bits may be checked, and if any are 1, active control arcs are implied. This could ultimately produce regeneration or other errors The existence of incomplete rendezvous is determined by examining the bits of each rendezvous for the value 1.

Three methods were investigated in the a[illegible] of simulation: the trial data test, the Monte Carlo test, and and the exhaustive test. The distinction between these is presented in sections 4.2.1.1 through 4.2.2.3.

4.2.1.1 SIMULATION WITH TRIAL DATA

This method attempts to provide data which would be typical for usage by the physical network for problem solution. The method has the advantage that it also provides checking for errors in algorithms, but it is generally unlikely that all possible combinations will be tested, especially with a large number of decisions in the graph.

4.2.1.2 MONTE CARLO SIMULATION

Application of this technique, suggested by Ellis,[65] simulates control only. A random number is generated each time a decision is encountered to determine which output arc is to be followed in the simulation. The Monte Carlo technique has the disadvantage that there is always a finite probability that some combination will not be tested. Also, not all combinations are equally likely to be tested.

4.2.1.3 EXHAUSTIVE SIMULATION

The exhaustive technique also simulates control only. All combinations of a network are tested. This is done by keeping a record in the simulation of the output arc selected for each decision, and simulating control for each possible combination of decisions. Exhaustive simulation generally results in more simulations than necessary, since the number of distinct combinations is usually somewhat less than the upper bound, i.e., the

product of the out-degree of all decision nodes. However, since it is not possible to tell a priori whether a combination has been tested, the exhaustive test necessarily simulates the upper bound of combinations.

Examples of the exhaustive test are presented in Figures 33 through 35. Two-way elements are assumed. Branches and rendezvous each have two bits associated with them, as previously explained. Three bits are associated with the decision. Two of these bits indicate on which arc the decision is to produce a control output. The other bit is 0 if control has not previously entered the decision, and 1 otherwise. The purpose of this bit is to provide a means for control to leave a strongly connected subgraph, rather than proceed in a loop indefinitely. Figure 33 is an error-free case and Figure 34 is not. Figure 35 gives a case with a hazard which is not detected by exhaustive testing.

4.2.2 TOPOLOGICAL ANALYSIS

Some errors may be detected by a topological analysis of the network. Topological analysis involves examining the interconnection of nodes in the network. Certain rules have been found which govern proper network construction, but no techniques using topology alone have been able to locate all errors. Several of these rules may be stated here. These will be justified in the appendix.

1. Parallel arcs from a branch to a merge are in error.
2. Parallel arcs from a decision to a rendezvous are in error.
3. Separable graphs composed entirely of decisions and merges are error free.
4. Strongly connected graphs composed entirely of branches and rendezvous are in error.

The main problem with topological analysis is that, generally, no inspection of subgraphs of any given number of nodes always yields a definite conclusion as to errors. Some topological *reductions* aid in simplifying the problem, however. An example of such a reduction is the technique explained in Chapter 3 for determining separable subgraphs.

4.2.3 SYMBOLIC ANALYSIS USING ALGEBRAIC EXPRESSIONS

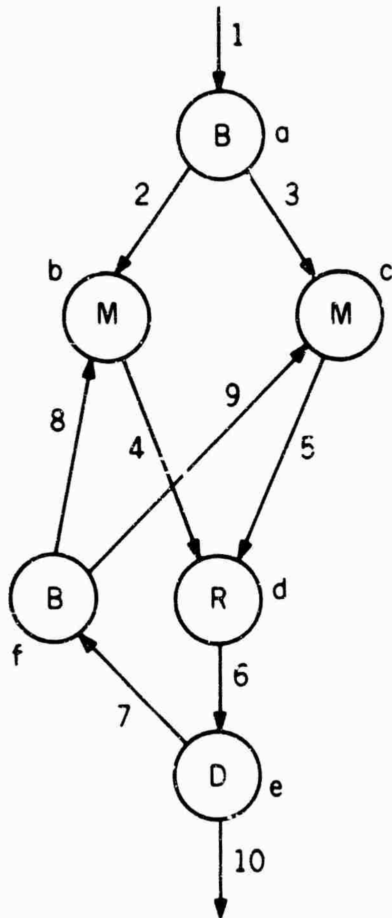
The following method, which was derived from suggestions by Stucki,[66] indicates some errors. The method is heuristic and is suggested by observing that the outputs of the rendezvous and merge are functions of their inputs in a manner which is similar to the Boolean functions of gates at the logic level. Specifically, the merge is an *exclusive-or* in the sense that it produces an output if either but not both inputs are present. Likewise, the rendezvous is an *and* in the sense that both inputs are required to produce an output. Expressions are synthesized, for each arc, which indicate the possibility of a control appearing on that arc as a function of the results of decisions.

Assume, for simplicity in illustration, that a graph is constructed only of two-way nodes. Two sets of symbols, a_1, a_2, a_3, ... and x_1, x_2, x_3, ... are used. The a's represent expressions on arcs, x_i and $\overline{x}_i$ represent the two possible completion arcs of the *ith* decision.

Expressions are formed for the output of arcs given their input arc expression by the following rules:

1. If a_k is the input to the *ith* decision, the outputs are labelled $a_k x_i$ (a_k *followed by* x_i) and $a_k \overline{x}_i$ (a_k *followed by "not"* x_i).
2. If a_k is the input to a branch, the outputs are both labeled a_k.
3. If a_i and a_k are inputs to a rendezvous, then one of the following is applied:

   a) If a_i and a_k are identical, then the output arc is labeled a_i.



ACTIVE ARC	BRANCH				REND.		DECISION		
NODE	a		f		d		e		
ARC	2	3	8	9	4	5	7	10	*

BEGIN TEST OF FIRST COMBINATION

1	0	0	0	0	0	0	0	1	0
2	0	1	0	0	0	0	0	1	0
4	0	1	0	0	1	0	0	1	0
3	0	0	0	0	1	0	0	1	0
5	0	0	0	0	1	1	0	1	0
6	0	0	0	0	0	0	0	1	0
10	0	0	0	0	0	0	0	1	1

END TEST OF FIRST COMBINATION

NO ERRORS

BEGIN TEST OF SECOND COMBINATION

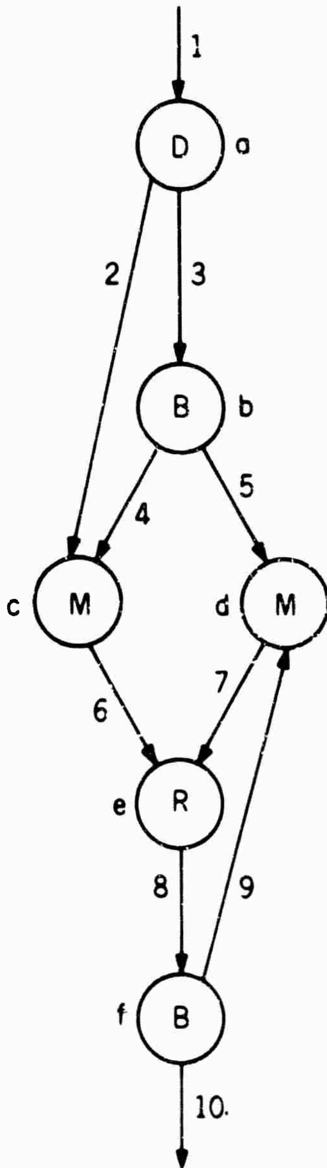
1	0	0	0	0	0	0	1	0	0
2	0	1	0	0	0	0	1	0	0
4	0	1	0	0	1	0	1	0	0
3	0	0	0	0	1	0	1	0	0
5	0	0	0	0	1	1	1	0	0
6	0	0	0	0	0	0	1	0	0
7	0	0	0	0	0	0	1	0	1
8	0	0	0	1	0	0	1	0	1
4	0	0	0	1	1	0	1	0	1
9	0	0	0	0	1	0	1	0	1
5	0	0	0	0	1	1	1	0	1
6	0	0	0	0	0	0	1	0	1
10	0	0	0	0	0	0	1	0	1

END TEST OF SECOND COMBINATION

NO ERRORS

FIGURE 33. EXHAUSTIVE TEST OF AN ERROR-FREE NETWORK

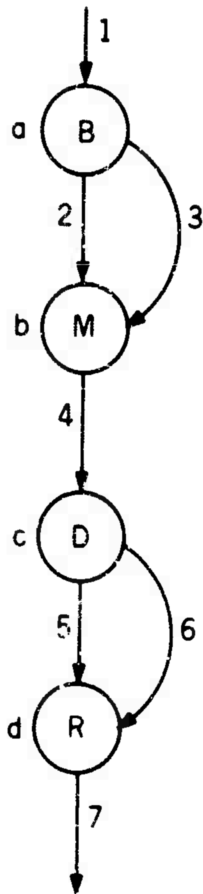
* - INDICATES DECISION ENTERED



ACTIVE ARC	BRANCH				REND.		DECISION		
NODE	b		f		e		a		
ARC	4	5	9	10	6	7	2	3	*
BEGIN TEST OF FIRST COMBINATION									
1	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	1	0
4	0	1	0	0	0	0	0	1	0
6	0	1	0	0	1	0	0	1	0
5	0	0	0	0	1	0	0	1	0
7	0	0	0	0	1	1	0	1	0
8	0	0	0	0	0	0	0	1	0
10	0	0	1	0	0	0	0	1	0
END TEST OF FIRST COMBINATION									
ERROR - ARC 9 ACTIVE									
BEGIN TEST OF SECOND COMBINATION									
1	0	0	0	0	0	0	1	0	0
2	0	0	0	0	0	0	1	0	0
6	0	0	0	0	0	0	1	0	0
END TEST OF SECOND COMBINATION									
ERROR - NON COMPLETION									

FIGURE 34. EXHAUSTIVE TEST OF A NETWORK WITH ERRORS

* - INDICATES DECISION ENTERED



ACTIVE ARC	BRANCH	REND.	DECISION
NODE	a	d	c
ARC	2 3	5 6	5 6 *

BEGIN TEST OF FIRST COMBINATION

1	0 0	0 0	0 1 0
2	0 1	0 0	0 1 0
4	0 1	0 0	0 1 0
6	0 1	0 1	0 1 1
3	0 0	0 1	0 1 1
4	0 0	0 1	0 1 1
5	0 0	1 1	0 1 1
7	0 0	0 0	0 1 1

END TEST OF FIRST COMBINATION

NO ERRORS

BEGIN TEST OF SECOND COMBINATION

1	0 0	0 0	1 0 0
2	0 1	0 0	1 0 0
4	0 1	0 0	1 0 0
5	0 1	1 0	1 0 1
3	0 0	1 0	1 0 1
4	0 0	1 0	1 0 1
6	0 0	1 1	1 0 1
7	0 0	0 0	1 0 1

END TEST OF SECOND COMBINATION

NO ERRORS

FIGURE 35. EXHAUSTIVE TEST OF A NETWORK WITH UNDETECTABLE ERRORS

* - INDICATES DECISION ENTERED

b) If a_i and a_k are known to be mutually exclusive, then an *incomplete rendezvous* is indicated.

c) If it is not known at the time whether a_i and a_k are identical or exclusive, then the output is labeled $a_i * a_k$ (a_i *and* a_k).

4. If a_i and a_k are inputs to a merge then one of the following is applied:

a) If a_i and a_k are mutually exclusive then a combination is made according to the usual rule for Boolean expressions($a_i x_i + a_i \bar{x}_i = a_i$, where + means *exclusive or*).

b) If a_i and a_k are known not to be mutually exclusive, a *hazard* is indicated.

c) If it is not known at the time whether a_i and a_k are exclusive, the output is labeled $a_i + a_k$ (a_i *or* a_k).

Figures 36 and 37 indicate the steps involved in performing symbolic analysis. The input arc to the separable graph is labeled a_o. Whenever all inputs to an node are labeled, the output arcs may be labeled. This is continued until it is impossible to proceed further due to an error, or until the output of the graph is labeled. This label specifies which decision combinations produce the output control signal.

In strongly connected graphs, there are one or more stages at which the labeling is not complete, but can proceed no further because no nodes remain with all inputs labeled. At this stage, a decision or branch is selected, and its input is assigned a unique label a_i. The labeling then continues, new labels being assigned as needed. Because of the introduction of these labels, there will be instances in which the input of a branch or decision *would be* labeled, but has already been assigned a label a_k. Let L represent the label which would be assigned if the arc were not previously labeled a_k. One of the following is then applied:

1. If L is the input to a branch, then:

a) If L is in terms of a_k, a *reentered branch* error is indicated;

b) If L is not in terms of a_k, all occurrence of a_k in all expressions are replaced by L.

2. If L is the input to a decision, then:

a) If L is in terms of a_k, an iteration is indicated. The situation is similar to a recursive definition, $a_k = L$ where $L = a_k x_i + a_i$, i.e., $a_k = a_i$ or a_k followed by x_i. All occurrences of a_k are then replaced by $a_i \# x_i$ meaning a_i followed by *any number of* x_i.

b) If L is not in terms of a_k, all occurrences of a_k in all expressions are replaced by L.

The analysis continues until an error is encountered, making continuation impossible, or until arcs are in terms of a_o and x's only. In the latter case, the network is not necessarily error-free, but any errors present remain undetected. Figures 38 and 39 show examples with strongly connected graphs.

The symbolic analysis technique has the disadvantage that it is cumbersome to implement on a computer, in addition to not being able to detect all errors. It does have the advantage of providing an expression indicating the condition for the existence of a signal on any arc as a function of the outcomes of decisions.

It is not implied that *no* symbolic analysis is useful or is able to detect all errors. Possibly some more thorough technique may be found. The symbolic analysis presented here is representative of several which

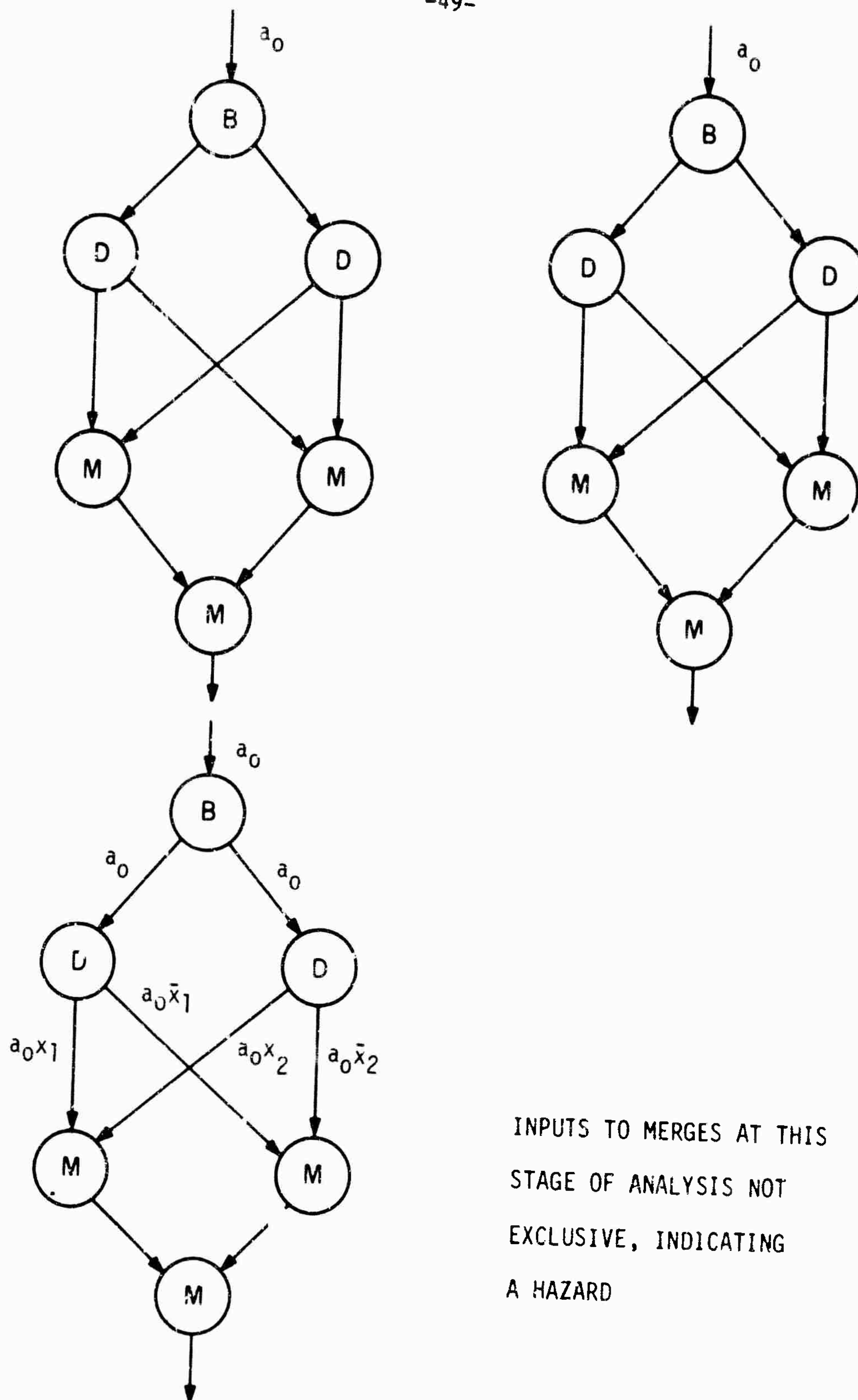


FIGURE 36. SYMBOLIC ANALYSIS SHOWING A HAZARD

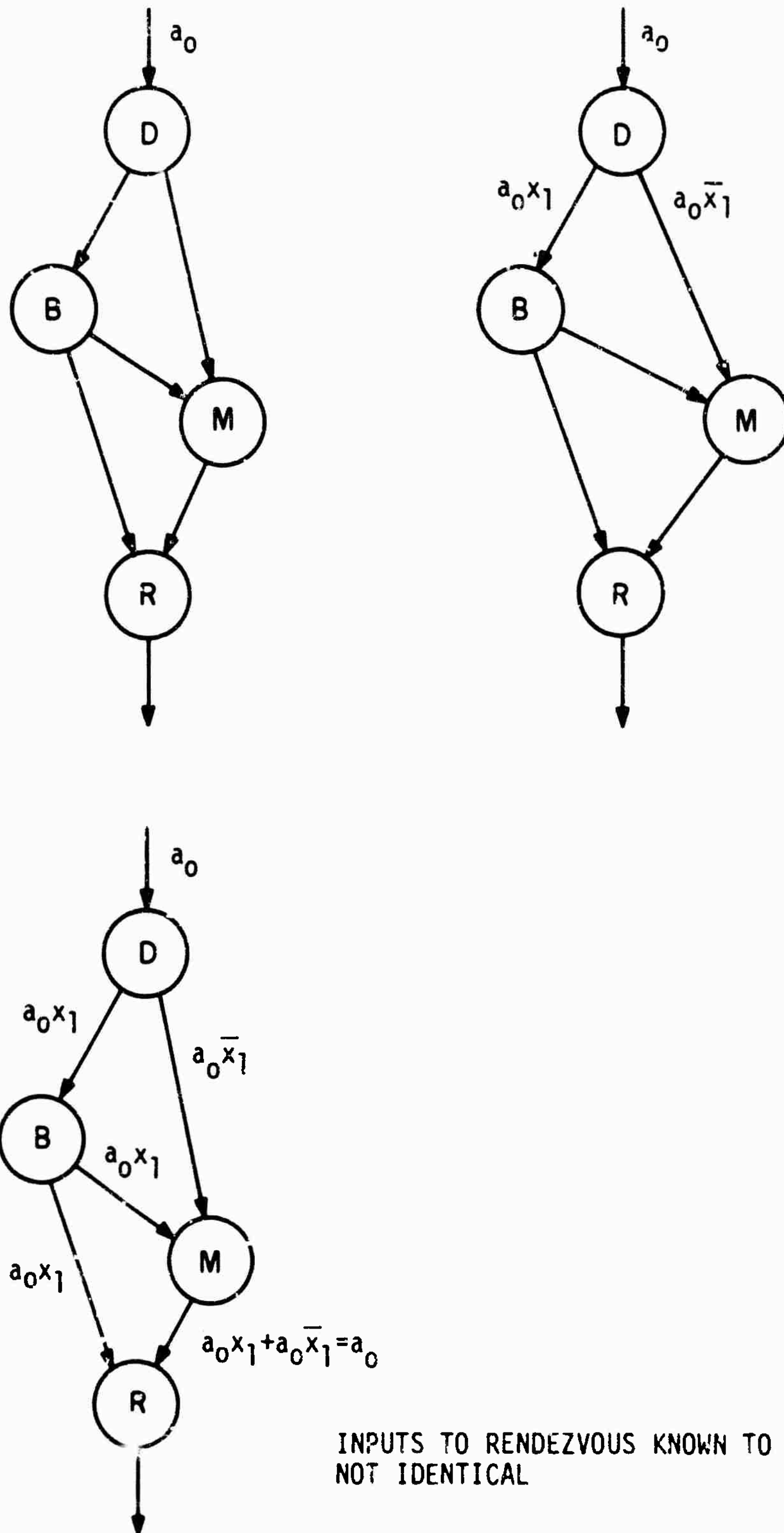


FIGURE 37. SYMBOLIC ANALYSIS OF AN INFINITE DURATION CASE

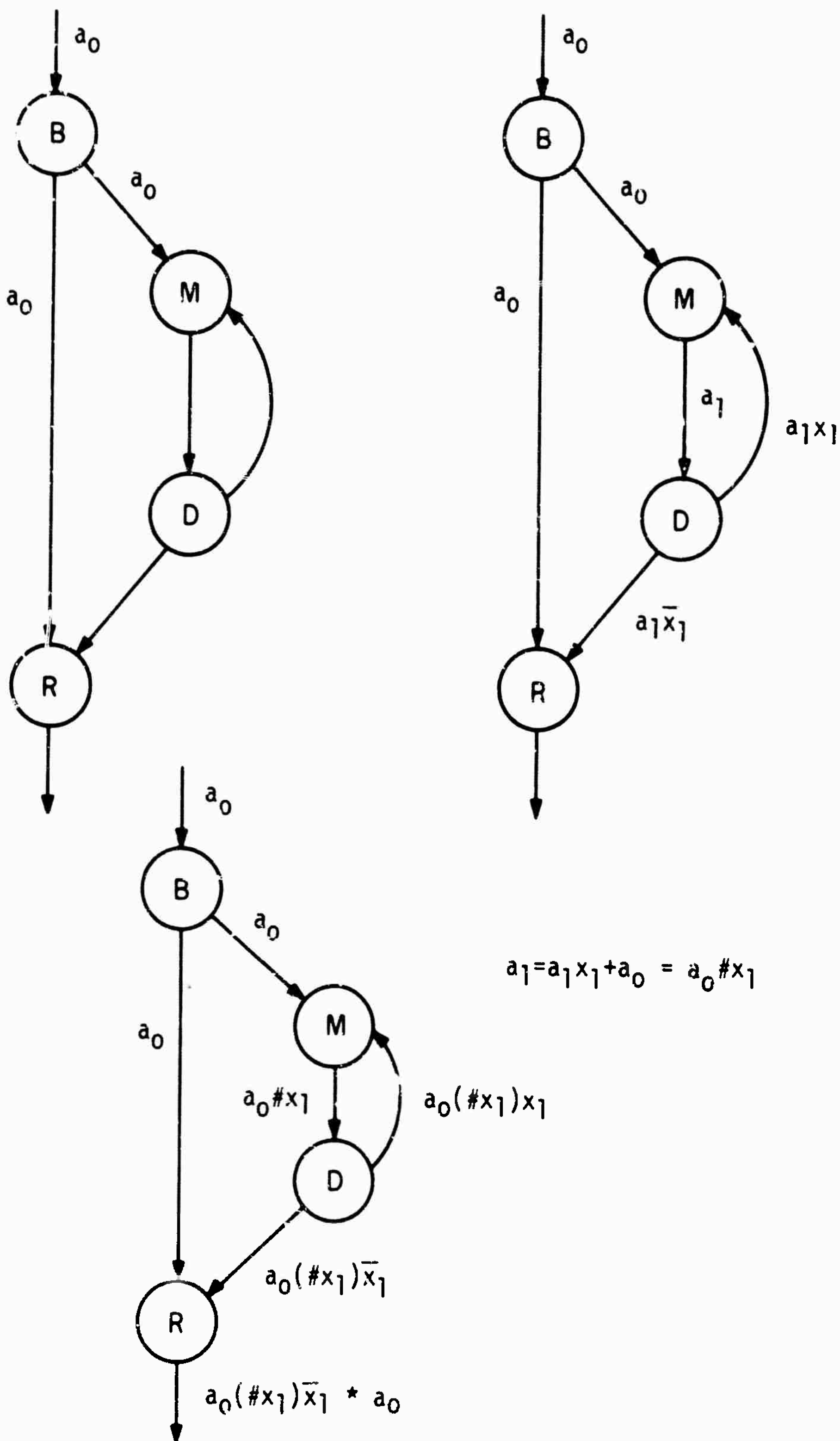


FIGURE 38. SYMBOLIC ANALYSIS OF AN ERROR-FREE CASE

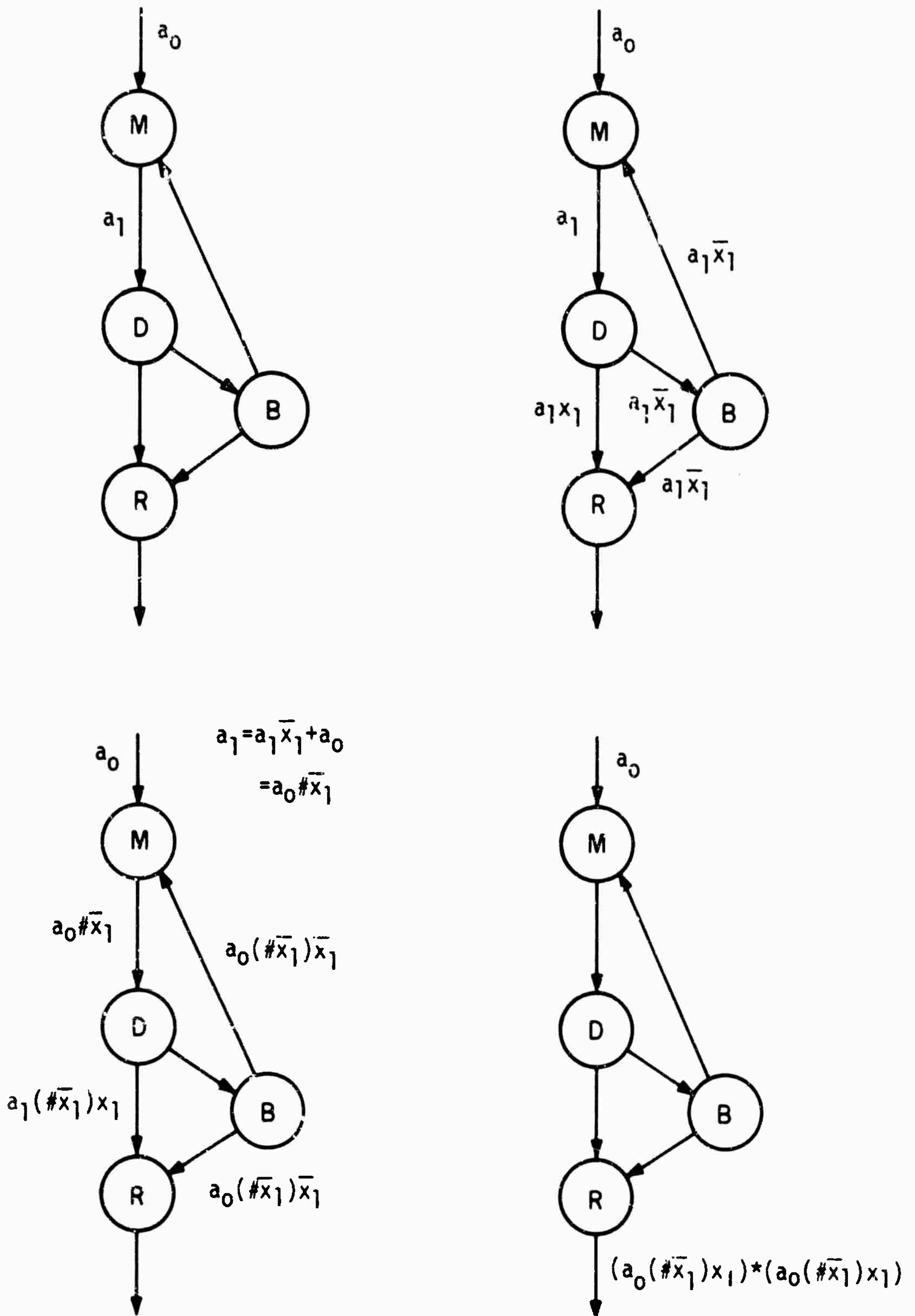


FIGURE 39. SYMBOLIC ANALYSIS OF AN ERROR CASE

were investigated in this study. A problem which is likely to be common to many symbolic approaches is the difficulty in computer implementation.

4.2.4 THE STATE TRANSITION METHOD

The final technique to be presented will be called the state transition method, due to a similarity to the representation of a sequential switching network as a graph displaying transitions between states.[61] It is suitable for digital computer implementation and can detect all of the implementation errors discussed, including some races. It also appears to be extendable to other types of control processes, such as the flag described in 2.3.4. The state of a network will be defined, followed by an explanation of error indication by the consideration of possible states, and possible transitions between them.

A label is assigned to each arc of the graph in consideration. A *state* of the network is a list of those arcs which may be active at a particular instant of time, e.g., *a, b, c,* where a, b, and c are arcs. A *transition* between states is the change of a network from one state to another. If the states of a network are represented as nodes of a second graph, the *possible* transitions may be represented as the arcs of this graph, which will be called the *state transition graph* of the network. Figure 40 indicates the state transition graphs for some simple error-free networks.

The *initial state* is defined to be that state of a separable network consisting of only the input arc of the network. The final state is that state consisting of only the output arc. A set of arcs, a_1, a_2, ... a_n, is said to be a partial state of a state, B = b_1, b_2, ... b_m, if and only if $a_i \in B$ for i = 1, 2, ... n.

The *partial states* of a state are used in determination of possible state transitions by considering each node having the arcs of a state as input arcs. For example, if *a* is a partial state of *a*, e_1, e_2, e_3, ..., and *a* is the input to a branch with outputs b and c, the transition from *a*, e_1, e_2, e_3, ..., *to* a state *b,c,*e_1,e_2,e_3, ..., may occur. The possible transition for the branch is indicated in Figure 41. A graph such as the one in Figure 41 will be called a *partial state transition graph.* The partial state transition graphs for the decision, merge, and rendezvous are shown in Figure 42. Note that the decision has two possible transitions. Note also that the transition graph for the merge has an isolated node representing the state *a,b.* If this state is possible, then a *hazard* is indicated. States or partial states which are identifiable as errors will be called *error states.*

Error states which indicate other types of errors are now explained. A state in which the output arc appears with other arcs is an error, because it indicates that a completion signal is produced while some arc may still be active within the network. If the output arc is removed from this state and the construction of the transition graph continued, then *regeneration* is indicated if the output arc eventually appears again in a state. Otherwise, *residual control* is indicated. If there are states, other than the final state, from which no transition may occur, *infinite duration* is indicated in these states. An attempt to form a transition to a state having two arcs which are the same is an indication of a *hazard.*

Since the transition graph indicates which arcs may be active simultaneously, consideration of the data in the separable processes represented by these arcs will indicate any races, if it is known exactly which memory elements will be used, and which will be altered. Figures 43 and 44 exemplify some networks with errors. The error-state nodes are indicated by dashed lines.

The following algorithm presents the steps in constructing the state transition graph. The network graph is assumed, with initial and final states known. The state graph initially contains a single node for the initial state.

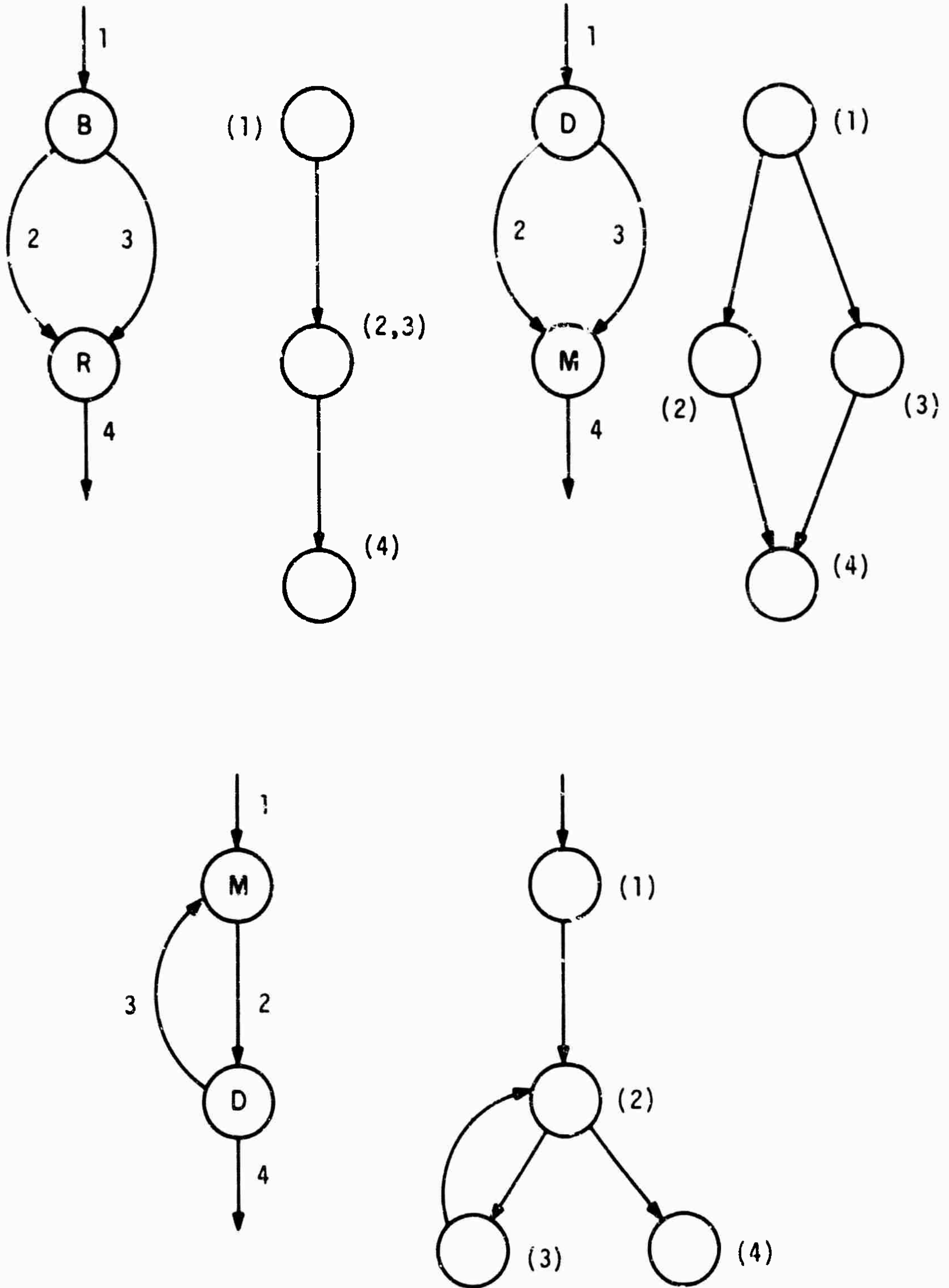
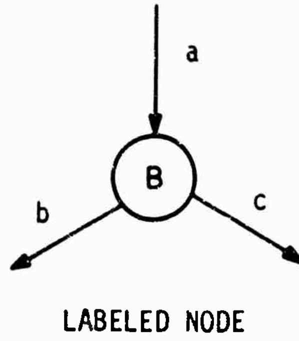
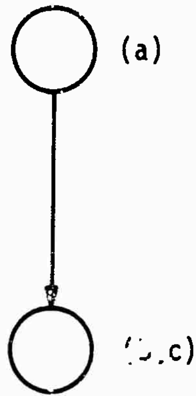


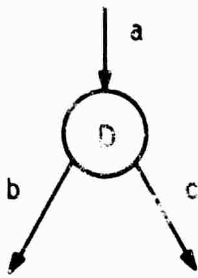
FIGURE 40. SOME SIMPLE ERROR-FREE NETWORKS AND THEIR CORRESPONDING STATE TRANSITION GRAPHS

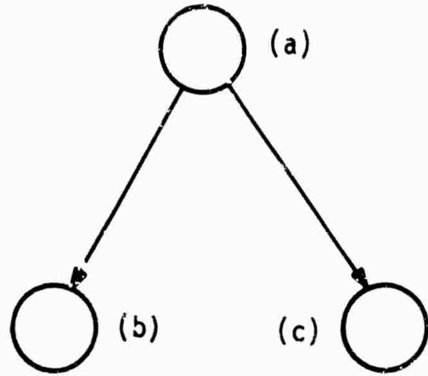


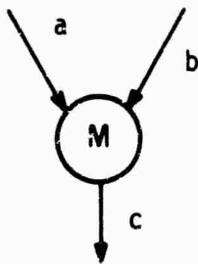


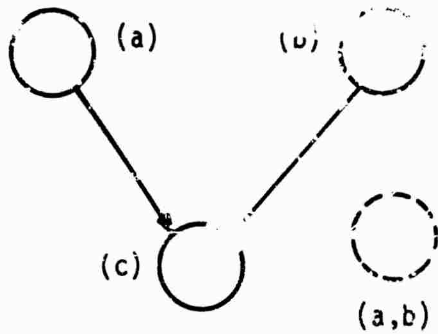
TRANSITION GRAPH

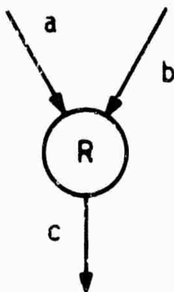
FIGURE 41. PARTIAL STATE TRANSITION GRAPH FOR A BRANCH











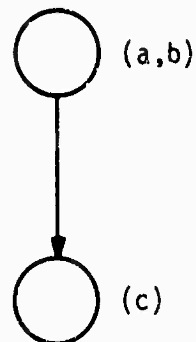
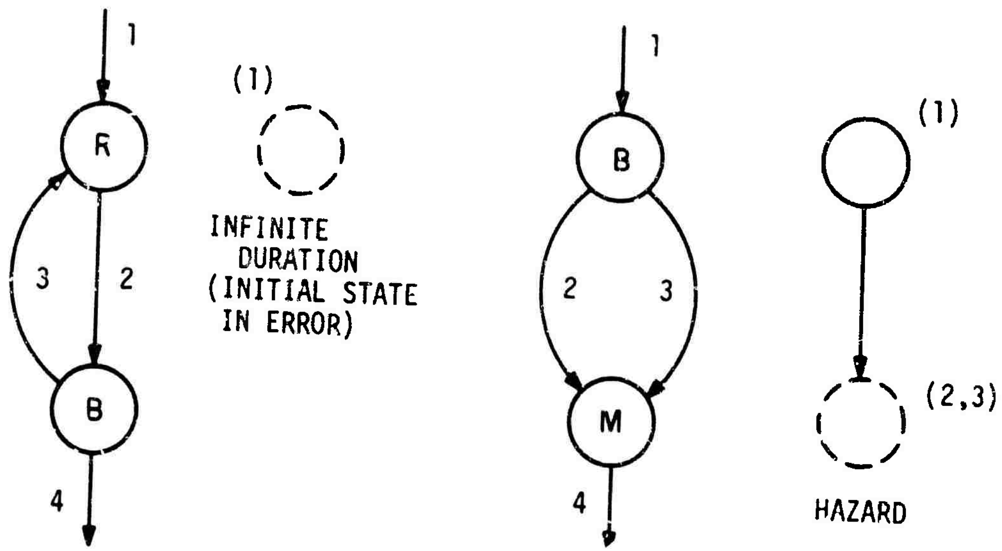


FIGURE 42. PARTIAL STATE TRANSITION GRAPHS FOR DECISIONS, MERGES, AND RENDEZVOUS



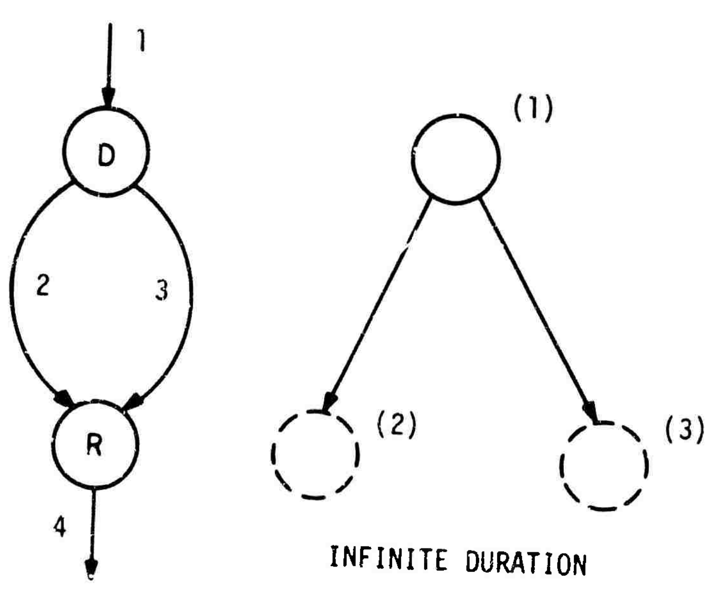
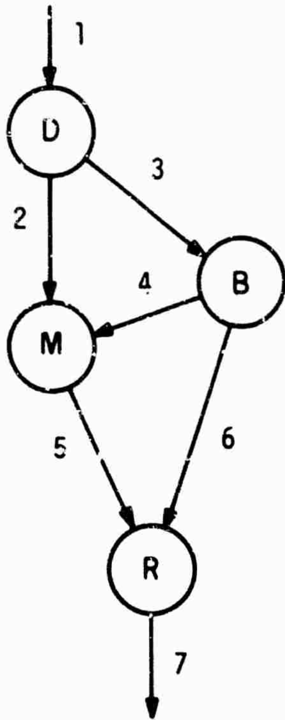
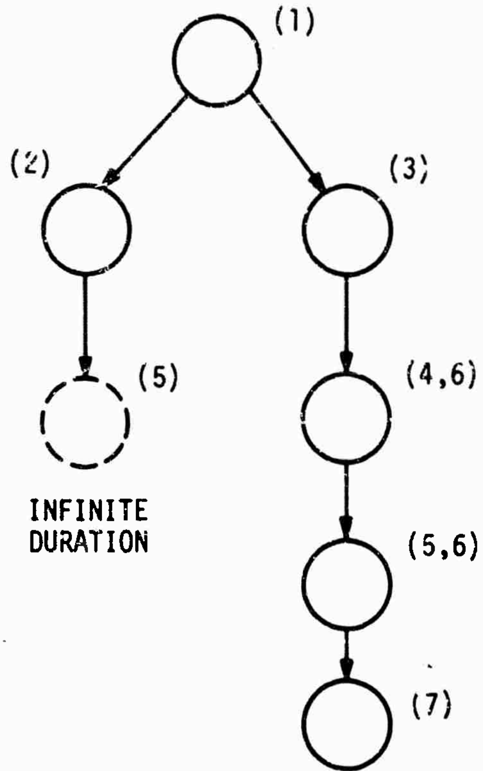
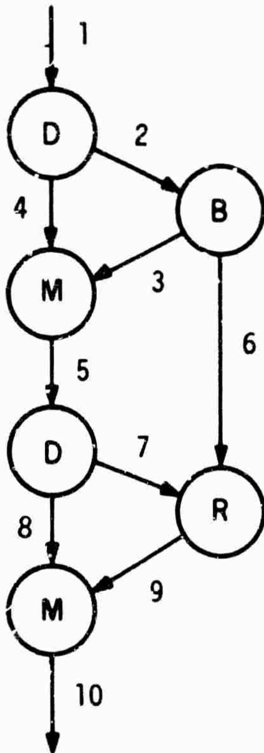


FIGURE 43. STATE TRANSITION GRAPHS FOR ERROR CASES







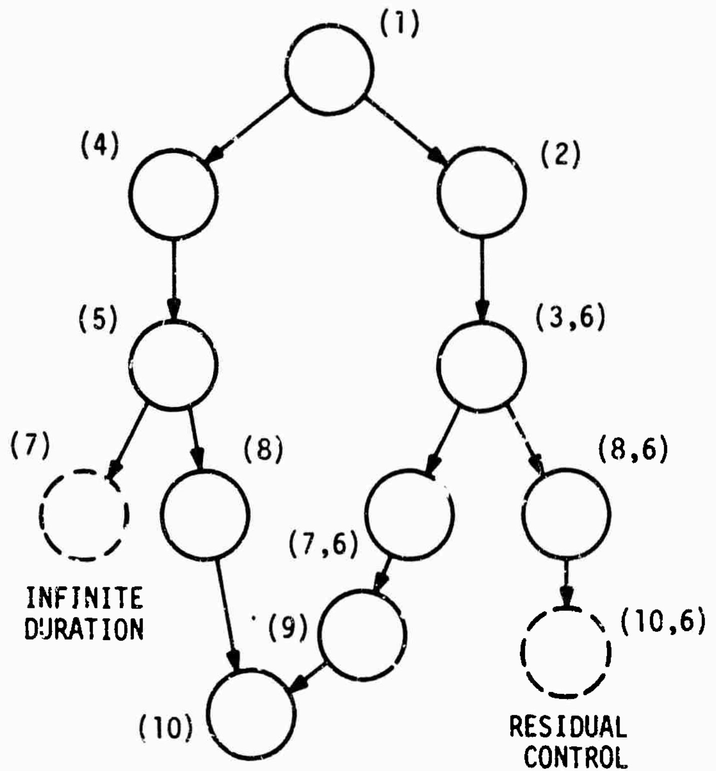


FIGURE 44. STATE TRANSITION GRAPHS FOR ERROR CASES

1. If all nodes have been considered for transitions, the state graph is complete. Stop.
2. Select a state for which all transitions have not been considered. Call this state s.
3. Select an arc in s, and call this arc c. Remove all arcs from s which are input to the same node to which c is input. Call these arcs p, a partial state.
4. Compare the arcs of p to the partial state diagram for possible transitions or error states.
5. If possible transitions exist, check to see if the next state formed is already in the state graph. If it is, connect the next state to state s by an arc. If it is not, add the new state to the graph and connect it to s by an arc.
6. Repeat steps 3 through 5 until all distinct nodes connected to arcs of s have been considered. Then go to step 1.

Step 4 of the algorithm may be elaborated by specifying the procedure for each of the four types of nodes considered. If other types of processes are involved, appropriate procedures for them must be formulated using the state transition graphs. The procedures for branch, rendezvous, decision, and merge are now given:

1. Branch – If the input arc of a branch is a partial state of p, a possible transition is to a state with the input arc replaced by the output arcs of the branch, and all other arcs in p unchanged.
2. Rendezvous – If *all* of the input arcs of a rendezvous form a partial state of p, a possible transition is to a state with these arcs replaced by the output arc of the rendezvous, and all other arcs in p unchanged.

   If some, but not all, of the input arcs of a rendezvous form a partial state of p, and there are no other possible transitions, an incomplete rendezvous error is indicated.
3. Decision – If the input arc of a decision is a partial state of p, possible transitions are to states with the input arc of the decision replaced by a single output arc of the decision.
4. Merge – If, at most, *one* of the input arcs of a merge is in any partial state of p, a possible transition is to a state consisting of p with the input arc to the merge replaced by the output arc. If more than one input arc to a merge is a partial state, a hazard is indicated.

For any element, if a transition is made in such a way that the new state has the same arc twice, an error is indicated.

4.2.5 A SUMMARY OF THE TECHNIQUES OF THIS STUDY

Table 1 compares the techniques presented as to their thoroughness in detection of various errors. It appears at this time, that the state transition method is the most complete and also is relatively simple to implement. Some topological reduction techniques, as described in the appendix, can be used to enhance the power of this, or any other method.

Table 1. A comparison of the error detection methods investigated

Method	Errors Detected: Incomplete Rendezvous	Reentered Branches	Hazards	Races	Comments
1 Simulation					
a) Trial Data	S	S	S	S	Also checks for some algorithm errors
b) Monte Carlo	S	S	S	N	Not all combinations are tested with equal likelihood
c) Exhaustive	A	A	S	N	Testing time may be prohibitive
2. Topological	S	S	S	N	Can be applied during reduction of graph
3. Symbolic	S	S	S	N	Cumbersome to implement by computer program
4. State transition	A	A	A	S	Simple to implement, extendable to other types of processes than decisions, merges, branches, and rendezvous

A -- All errors of this type detected
S -- Some errors of this type detected
N -- No errors of this type detected

5. SUMMARY AND CONCLUSION

A model suitable for representation of particular types of computing systems has been presented and techniques for introducing concurrent asynchronous control into the model discussed.

Graph theoretic terminology was introduced, and its applicability to description of the model demonstrated.

Several matric representations for graphs were presented as a possible means for representing the model in a computer for which automatic analysis techniques may be implemented.

The possible introduction of errors into computing systems by improper implementation of concurrent asynchronous control networks was illustrated, and various types of errors were classified. Several methods for detection of implementation errors were investigated and compared.

It is concluded that state-transition method, in combination with topological reductions, is the most satisfactory method of those investigated due to several advantages: It detects all implementation errors in the type of networks considered, it is extendable to other types of processes, and it may be implemented by a digital computer program.

6. ACKNOWLEDGEMENT

The authors would like to acknowledge tne suggestions and comments of Robert Ellis and Mishell Stucki of the Computer Systems Laboratory, that proved to be of immeasurable assistance in this investigation.

APPENDIX 7.1

REDUCTION TECHNIQUES AND SOME THEOREMS CONCERNING TOPOLOGY

As mentioned in section 4.2.2, some topological reductions may be applied to reduce the complexity of the error analysis problem. In the course of these reductions, some simple tests may be made to determine if the network contains errors. These reductions and tests will be presented, and derived by consideration of partial state transition diagrams. It should be noted that the reductions remove some arcs and hence, if races are to be checked for, this should be done before applying any reductions. The following paragraphs do not apply to detection of races.

The *input boundary states* of a graph *not necessarily separable* are those states containing only input arcs to the graph. A similar definition applies for *output boundary states*. In considering the state transition graph for a network, each input boundary state ultimately reaches, *in the graph-theoretic nomenclature*, either an output state or an error state. Two graphs are said to be *A-equivalent* if they have the same reachability relations among their respective error and boundary states. The state graphs of two A-equivalent networks are shown in Figure 7.7.1.

Postulate - If P is a graph and Q a subgraph of P, P may be analyzed for errors by replacing Q with any A-equivalent subgraph.

Theorem 1 - All weakly connected graphs composed of only one type *branch, decision, merge, or rendezvous* of 2-way elements are A-equivalent to a single element of the same degree as the former graph.

The proof is given for merges. The others follow analogously. The method of mathematical induction is employed. Assume that the theorem holds for a graph of 2-way merge nodes of degree *n,1*. This network is shown with its transition graph in Figure 7.1.2. Since the 2-way merge network is assumed equivalent to an n-way merge, its state graph is that of the n-way merge. A 2-way merge is then added, producing a network of degree *n + 1, 1*. The transition graph for this augmented network is shown in Figure 7.1.3. Figure 7.1.4 displays a second state graph with the same reachability among boundary and error states as that in Figure 7.1.3. The second graph is identical to a state transition graph for a merge node of degree *n + 1, 1*.

For the case $n = 2$, the theorem holds trivially since a network of degree *2,1* is identical to a single 2-way merge. The truth of the theorem has been shown for a single 2-way merge element and the assumption of truth for a network of degree *n,1* has been shown to imply its truth for a network of degree *n + 1, 1*. Thus, by induction, the theorem is true for all merge networks of degree *n,1* where $n \geq 2$.

Theorem 2 - Any two weakly connected graphs composed of one type of node *decision, merge, branch, or rendezvous* and of the same degree are A-equivalent.

Proof - Any such graphs are A-equivalent to a graph of 2-way nodes and of the same degree. Therefore, they are equivalent to each other.

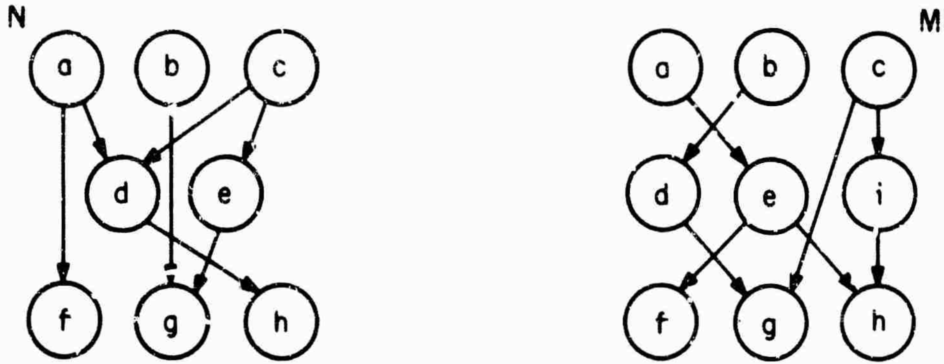
Theorem 3 - Any separable graph which is error-free is A-equivalent to a null process.

Proof - A separable error-free graph has no error state. There is a single input boundary state which reaches a single output boundary state. This is equivalent to the graph of a null process.

Theorem 4 - A separable graph, the input arc of which is connected to a rendezvous, is in error.

The proof follows from the fact that the initial state can make no transitions.

Theorem 5 - Any parallel arcs from a subgraph composed of decisions to a subgraph composed of merges, or from a subgraph composed of branches to a subgraph composed of rendezvous, may be replaced with a single arc with A-equivalence being preserved.



a, b, c, f, g, h ARE BOUNDARY STATES

R(N) =

	a	b	c	f	g	h	d	e
a	0	0	0	1	0	1	1	0
b	0	0	0	0	1	0	0	0
c	0	0	0	0	1	1	1	1
f	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	0
d	0	0	0	0	0	1	0	0
e	0	0	0	0	1	0	0	0

R(M) =

	a	b	c	f	g	h	d	e	i
a	0	0	0	1	0	1	0	1	0
b	0	0	0	0	1	0	1	0	0
c	0	0	0	0	1	1	0	0	1
f	0	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	0		0
h	0	0	0	0	0	0	0	0	0
d	0	0	0	0	1	0	0	0	0
e	0	0	0	1	0	1	0	0	0
i	0	0	0	0	0	1	0	0	0

FIGURE 7.1.1 TWO A-EQUIVALENT STATE GRAPHS AND THEIR REACHABILITY MATRICES

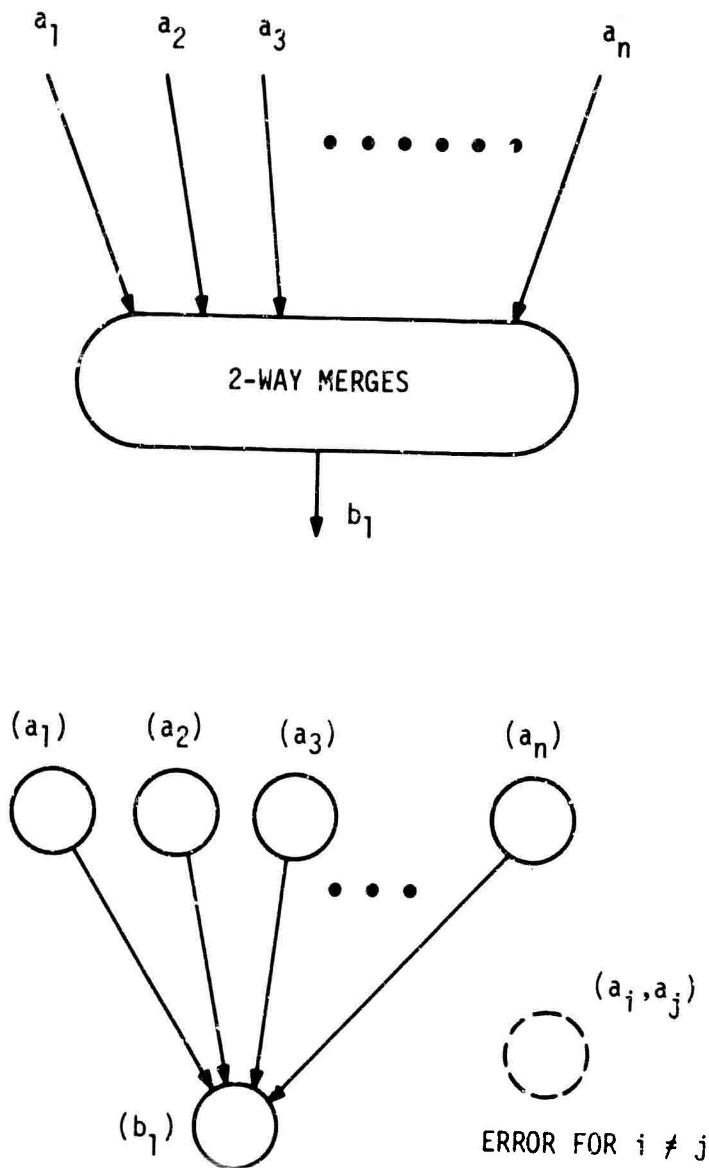
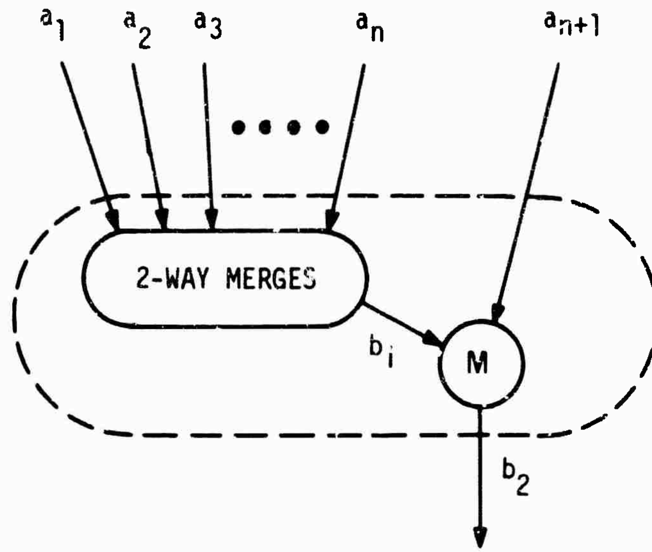


FIGURE 7.1.2 TWO-WAY MERGE NETWORK OF DEGREE (n,1) AND ITS TRANSITION GRAPH ASSUMING A-EQUIVALENCE TO AN n-WAY MERGE



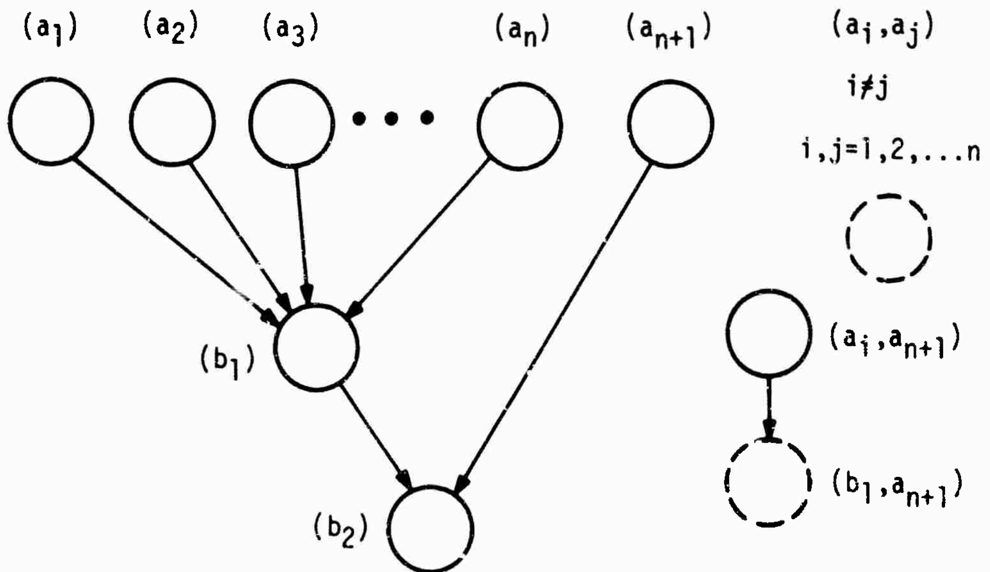


FIGURE 7.1.3 THE RESULT OF CONNECTING A 2-WAY MERGE TO THE NETWORK OF FIGURE 7.1.2

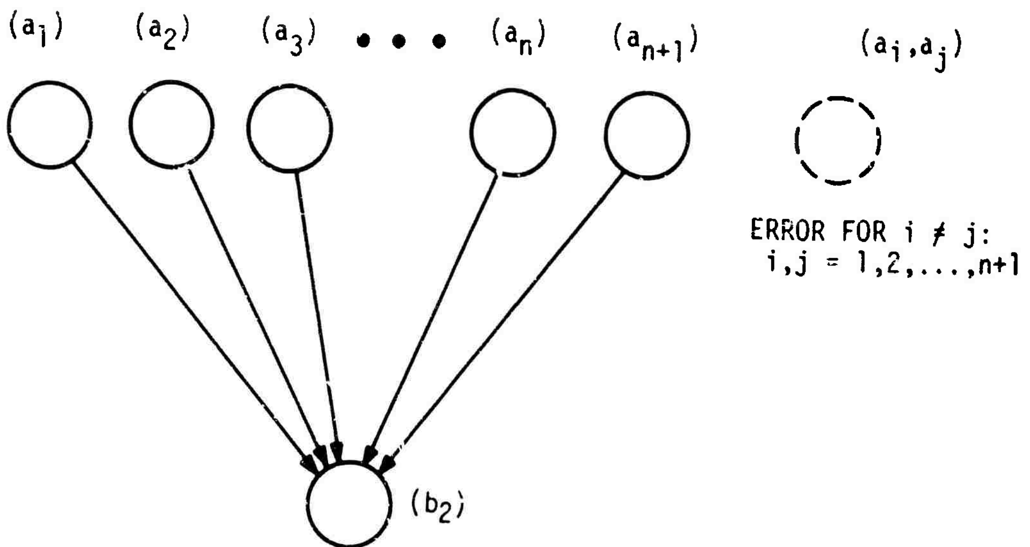


FIGURE 7.1.4 STATE DIAGRAM WITH THE SAME REACHABILITY AMONG BOUNDARY AND ERROR STATES AS THAT IN FIGURE 7.1.3

Proof - The two graphs are equivalent to graphs composed of only 2-way elements. The 2-way elements may be grouped in such a way that the arcs between these elements are parallel. Parallel arcs between a 2-way decision and a 2-way merge, or between a 2-way branch and a 2-way rendezvous, are known to constitute separable error-free graphs. These arcs may be replaced by a null process, i.e., a single arc.

Theorem 6 - Any parallel arcs from a subgraph composed of decisions to a subgraph composed of rendezvous, or from a subgraph composed of branches to a subgraph composed of merges, indicate an error.

Theorem 7 - A strongly connected graph consisting only of branches and rendezvous is in error.

Proof - The graph may be divided into maximal subgraphs consisting of either all branches or rendezvous. All input arcs to the graph must be input arcs to rendezvous, since if the arcs are inputs to branches, the graph is not strongly connected. At least one input arc to each of these rendezvous must not be an input arc of the subgraph, for if it were, the subgraph would not be strongly connected. It is never possible for any of these rendezvous to report completion, since every one requires control from inside the graph, but this control cannot be present, without at least one rendezvous reporting completion. Therefore, the subgraph is in error.

Use of these theorems will now be illustrated in the reduction of the floating-point arithmetic unit of Figure 32. A suggested procedure for application of the theorems follows, although no attempt is made to show that it is optimal:

1. Form all maximal subgraphs of a single type of elements.
2. Check for any parallel arcs among these subgraphs. If some arcs are parallel and indicate errors, then the procedure is stopped. If arcs are parallel and may be replaced by a single arc, then replace them. If no arcs are parallel, then go to step 3, otherwise go to 1.
3. Check the remainder of the graph by the state transition method.

Figures 7.1.5 through 7.1.8 illustrate the application.

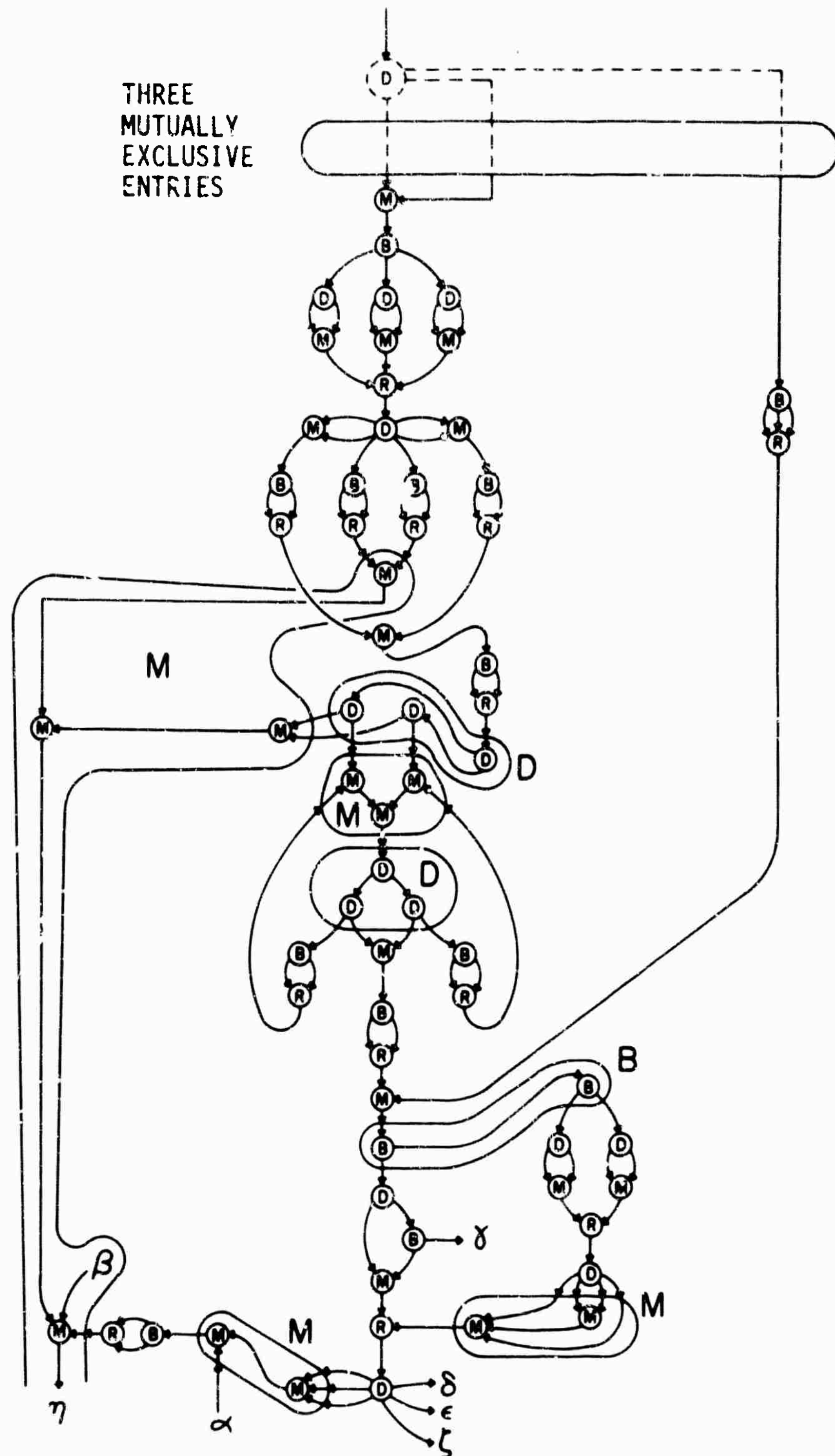


FIGURE 7.1.5 REDUCTION ALGORITHM AS APPLIED TO FIGURE 32.
STEP 1 - FORMATION OF MAXIMAL SUBGRAPHS OF EACH TYPE NODE
(CONTINUED ON FOLLOWING PAGE)

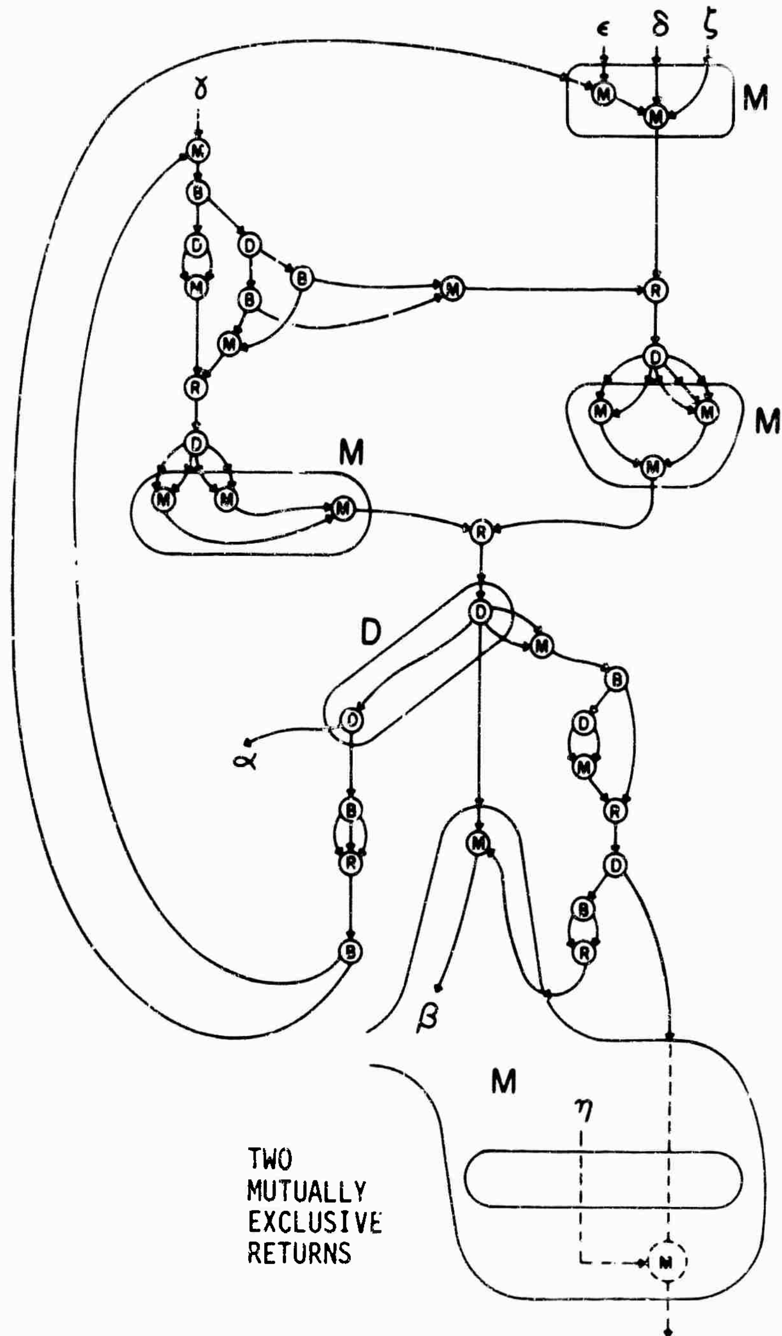


FIGURE 7.1.5 (CONTINUED)

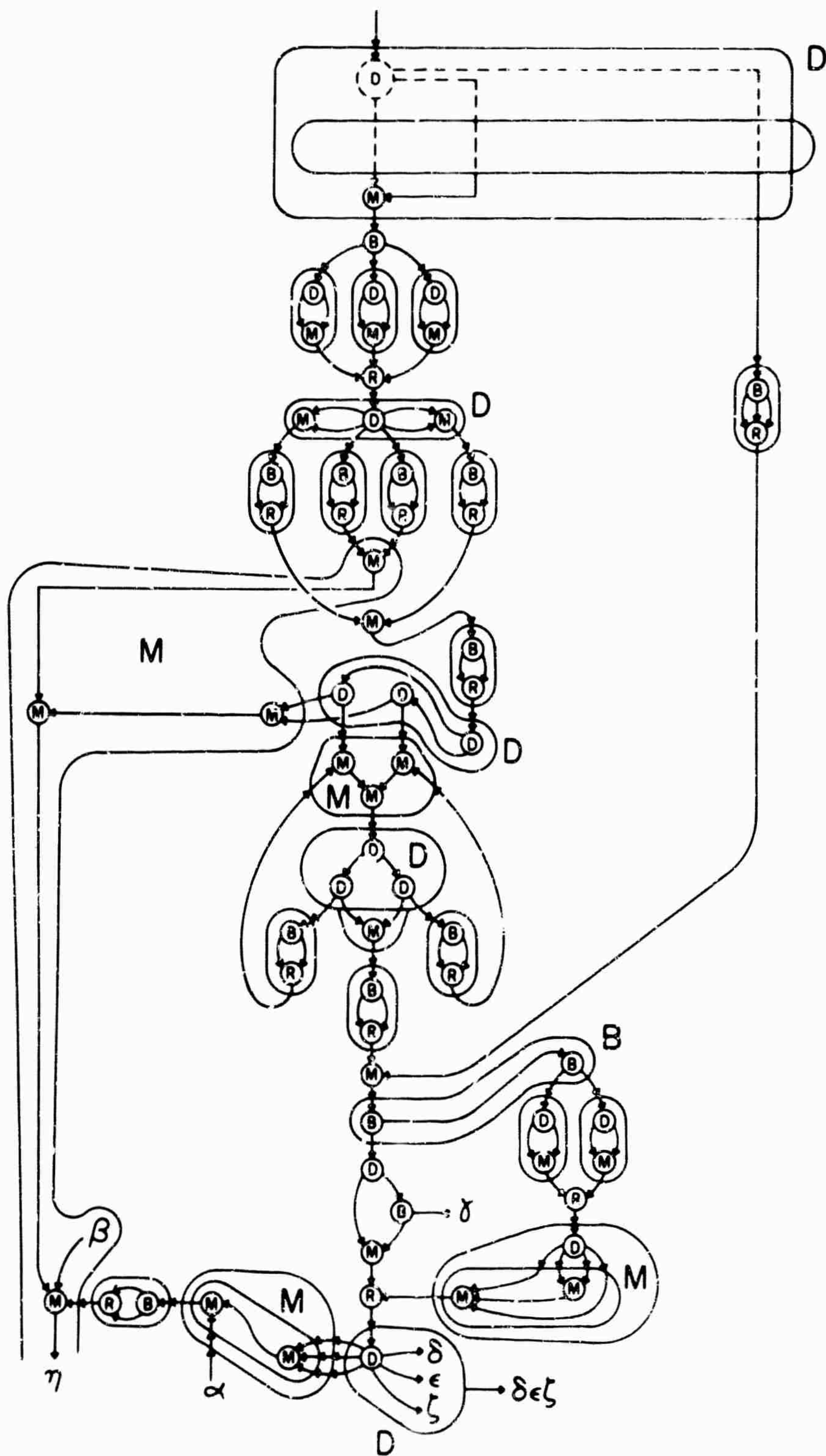


FIGURE 7.1.6 REDUCTION ALGORITHM AS APPLIED TO FIGURE 32.
STEP 2 - ELIMINATION OF PARALLEL ARCS (CONTINUED ON FOLLOWING PAGE)

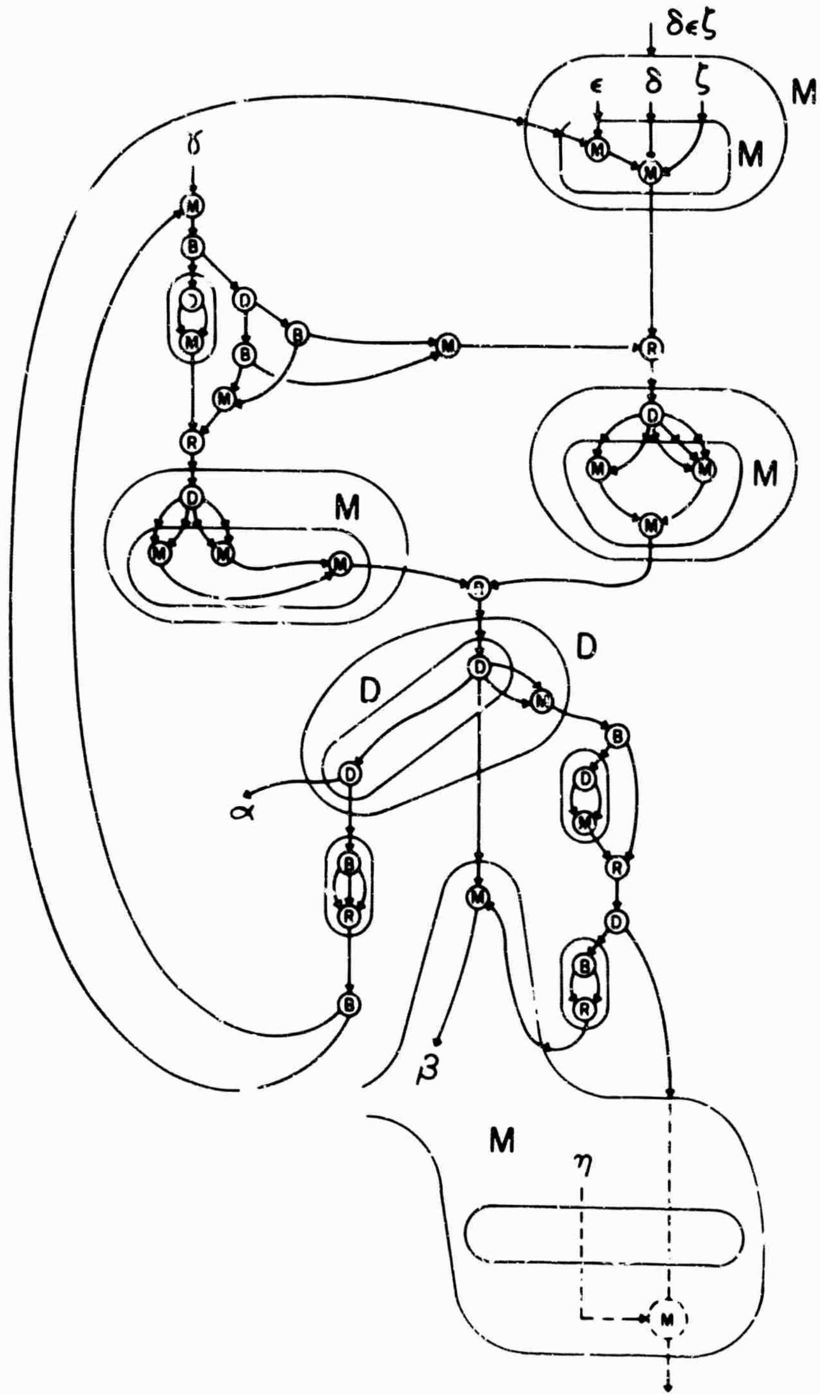


FIGURE 7.1.5 (CONTINUED)

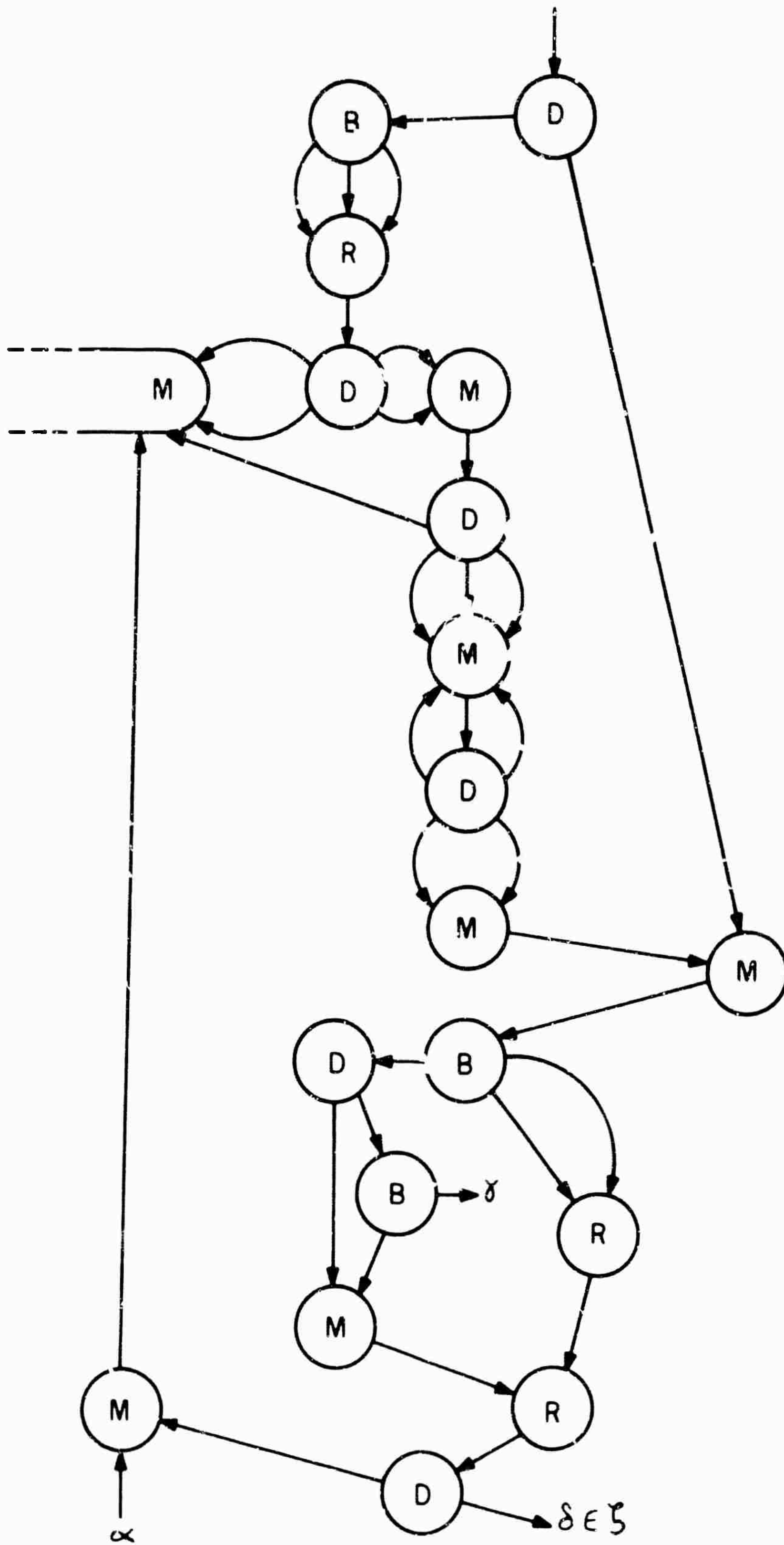


FIGURE 7.1.7 FIGURE 7.1.6 RE-DRAWN WITH SEPARABLE SUBGRAPHS REPLACED BY ARCS (CONTINUED ON FOLLOWING PAGE)

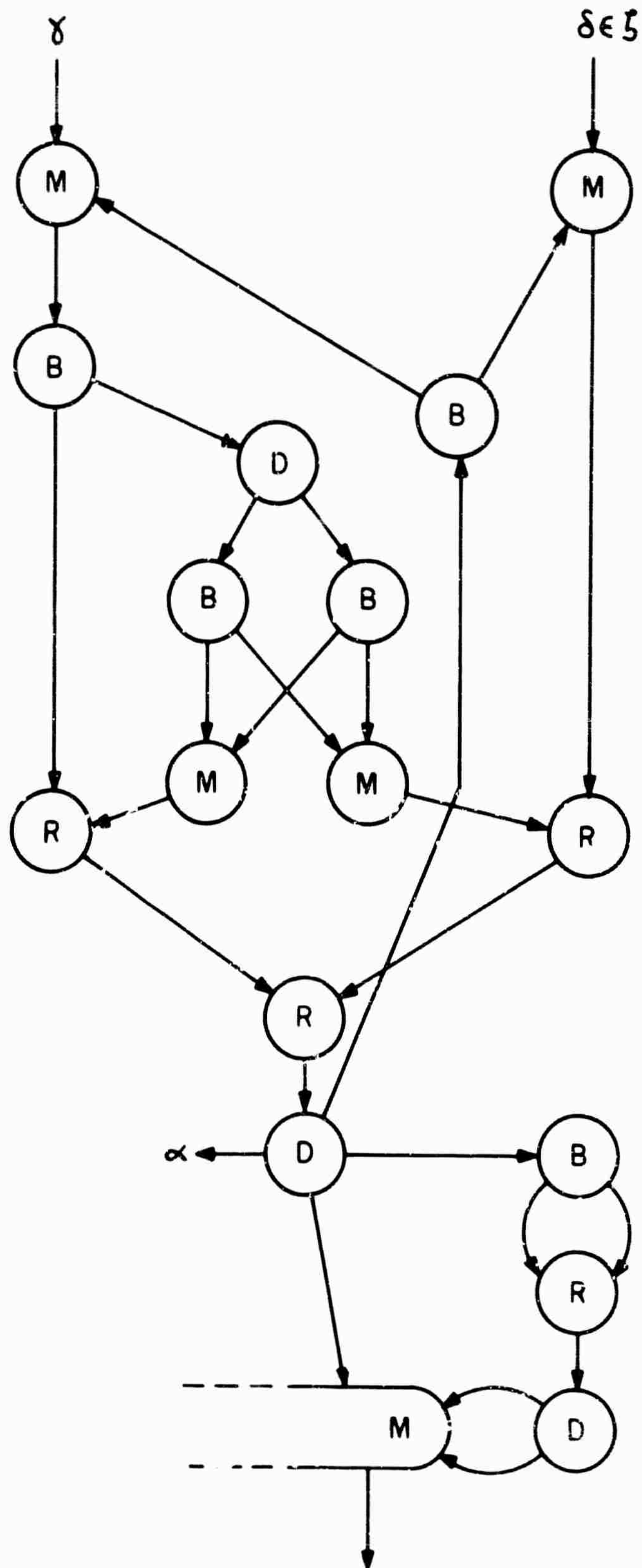


FIGURE 7.1.7 (CONTINUED)

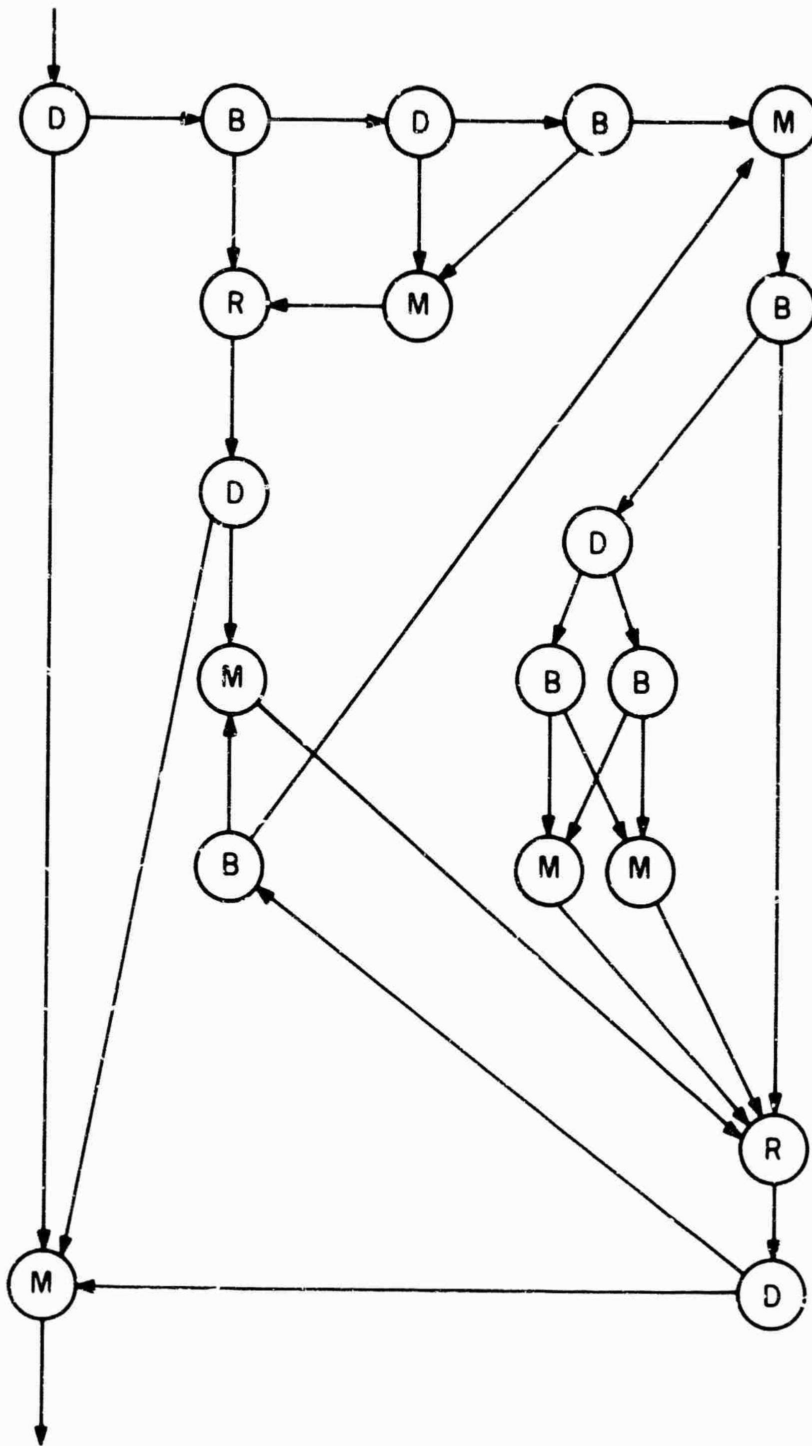


FIGURE 7.1.8 RESULT OF REPEATED APPLICATIONS OF THE REDUCTION ALGORITHM TO FIGURE 7.1.7

8. REFERENCES

1. Mathis, Wiley, and Spandorfer, editors, *Proceedings of a Symposium on Microelectronics and Large Systems*, Spartan Books, 1965.
2. von Neumann, John, *Collected Works*, New York: Macmillan, 1961-1963.
3. Slotnick, et al., *The SOLOMON Computer*, Proceedings of the Fall Joint Computer Conference, 1963, *774-781*.
4. Holland, J.H., *A Universal Computer Capable of Executing An Arbitrary Number of Sub-Programs Simultaneously*, Proceedings of the Eastern Joint Computer Conference, 1959, *108-113*.
5. Murtha, John C., *Highly Parallel Information Processing Systems*, Advances in Computers, Vol. 7, Academic Press, 1966, *1-116*.
6. Block, Richard M., *Mark I Calculator*, Proccedings of a Symposium on Large-Scale Digital Calculating Machinery; Annals of the Computation Laboratory of Harvard University, Vol. 16, *23-30*, 1948.
7. Tabor, Lewis P., *Brief Description and Operating Characteristics of the ENIAC*, Proceedings of a Symposium on Large-Scale Digital Calculating Machinery; Annals of the Computation Laboratory of Harvard Laboratory, Vol. 16, *31-39*, 1948.
8. Curtin, William A., *Multiple Computer Systems*, Advances in Computers, Vol. 4, *245-303*, 1963.
9. Estrin, Gerald, *Organization of Computer Systems—The Fixed Plus Variable Structure Computer*, Proceedings of the Western Joint Computer Conference, 1960, *33-401*.
10. Clark, Stucki, and Ornstein, *A Macromodular Approach to Computer Design*, Washington University Computer Research Laboratory, Technical Report No. 1, February 1966.
11. Rothkopf, Michael H., *Scheduling Independent Tasks on Parallel Processors*, Management Science, Vol. 12, No. 5, *437-447*, January, 1966.
12. Schwartz, Eugene S., *An Automatic Sequencing Procedure with Application to Parallel Programming*, Journal of the Association for Computing Machinery, Vol. 8, *513-537*, October 1961.
13. Hu, T. C., *Parallel Sequencing and Assembly Line Problems*, Journal of Operations Research, Vol. 9, *841-848*, November—December 1961.
14. Dorn, Hu, and Rivlin, *Some Mathematical Aspects of Parallel Computation*, International Business Machines Research Report, RC-647, January 1962.
15. Shedler and Lehman, *Parallel Computation and the Solution of Polynomial Equations*, International Business Machines Research Report, RC-1550, February 1966.
16. Nievergelt, J., *Parallel Methods for Integrating Ordinary Differential Equations*, Communications of the Association for Computing Machinery, Vol. 7, No. 12, *731-733*, December 1964.
17. Dorn, W. S., *Generalization of Horner's Rule for Polynomial Evaluation*, International Business Machines Journal, *239-245*, April 1962.
18. Allard, Wolf, and Zemlin, *Some Effects of the 6600 Computer on Language Structures*, Communications of the Association for Computing Machinery, Vol. 7, No. 2, *112-119*, February 1964.
19. Stone, Harold S., *One-Pass Compilation of Arithmetic Expressions for a Parallel Processor*, Communications of the Association for Computing Machinery, Vol. 10, No. 4, *220-223*, April 1967.
20. Hellerman, H., *Parallel Processing of Algebraic Expressions*, Institute of Electrical and Electronics Engineers, Transactions, Vol. EC-15, No. 1, *82-91*, February 1966.
21. Roder and Rosene, *Memory Protection In Multiprocessing Systems*, Institute of Electrical and Electronics Engineers, Transactions, Vol. EC-16, No. 3, *320-326*.

22. Gountanis and Viss, *A Method of Processor Selection for Interrupt Handling in a Multiprocessor System*, Proceedings of the Institute of Electrical and Electronics Engineers, Vol. 54, No. 12, *1812-1819*, December 1966.

23. Tomasulo, R. M., *An Efficient Algorithm for Exploiting Multiple Arithmetic Units*, International Business Machines Journal, Vol. 11, No. 1, *25-33*, January 1967.

24. Wann, Donald F., *Error Analysis in Parallel Processing*, Washington University Computer Systems Laboratory, Technical Memorandum No. 13, December 1966.

25. Ellis, Robert A., *Applications of Graph Theory to the Analysis of Computer Structures*, Washington University Computer Systems Laboratory, Technical Memorandum No. 11, 1966.

26. Wann, Ellis, Stucki, and Keller, *Problems Encountered in Control Networks in Highly Restructurable Digital Systems*, Institute of Electrical and Electronic Engineers, First Annual Computer Conference, September 1967.

27. Karp and Miller, *Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing*, Society of Industrial and Applied Methematics Review Vol. 14, No. 6, *1390-1411*, November 1966.

28. Reiter, Raymond, *A Study of a Model for Parallel Computation*, University of Michigan Systems Engineering Laboratory, Technical Report ISL-65-4, 1965.

29. Reiter, Raymond, *Initiation Timing in a Model for Parallel Computation*, University of Michigan Systems Engineering Laboratory, Technical Report 0690-6-T.

30. Marimont, R. B., *A New Method of Checking Consistency of Precedence Matrices*, Journal of the Association for Computing Machinery, Vol. 6, *164-171*, 1959.

31. Clark, W. A., *Macromodular Computer Systems*, Proceedings of the Spring Joint Computer Conference, 1967, *335-336*.

32. Clark, Ornstein, and Stucki, *A Functional Description of Macromodules*, Proceedings of the Spring Joint Computer Conference, 1967, *337-355*.

33. Conway, Melvin E., *A Multiprocessor System Design*, Proceedings of the Fall Joint Computer Conference, 1963, *139-146*.

34. Richards, P., *Parallel Programming*, Technical Operations Incorporated, Report No. TO-B 69-27, 1960.

35. Opler, Ascher, *Procedure-Oriented Language Statements to Facilitate Parallel Processing*, Communications of the Association for Computing Machinery, Vol. 8, No. 5, *306-307*, May 1965.

36. Anderson, James P., *Program Structures for Parallel Processing*, Communications of the Association for Computing Machinery, Vol. 8, No. 5 *786-788*, December 1965.

37. Leipold and Rekowski, *A Method for the Simultaneous Processing of Several Programs*, Proceedings of the International Federation of Information Processing Congress, Vol. 2, *320-321*, 1965.

38. Gosden, J. A., *Explicit Parallel Processing Description and Control in Programs for Multi- and Uni-Processor Computers*, Proceedings of the Fall Joint Computer Conference, 1966, *651-660*.

39. Dennis and Van Horn, *Programming Semantics for Multiprogrammed Computations*, Communications of the Association for Computing Machinery, Vol. 9, No. 3, March 1966.

40. Wirth, Niklaus, *A Note on Program Structures for Parallel Processing*, Communications of the Association for Computing Machinery, Vol. 9, No. 5, March 1966.

41. International Business Machines Corporation Program Language – One, Language Specifications.

42. Porter, R. E., *The RW-400 – A New Polymorphic Data System*, Datamation, *9-14*, January-February 1960.

43. Schwartz, J., *Large Parallel Computers*, Journal of the Association for Computing Machinery, Vol. 13, No. 1, *25-31*, January 1966.

44. Aschenbrenner, Flynn, and Robinson, *Intrinsic Multiprocessing*, Proceedings of the Spring Joint Computer Conference, 1967, *81-86*.

45. Berge, C., *Theory of Graphs and Its Applications*, Wiley, 1962.

46. Ore, O., *Theory of Graphs*, American Mathematical Society, 1964.

47. Harary, Norman, and Cartwright, *Structural Models: An Introduction to the Theory of Directed Graphs*, Wiley, 1965.

48. Prosser, R. T., *Applications of Boolean Matrices to the Analysis of Flow Diagrams*, Proceedings of the Eastern Joint Computer Conference, 1957, *133-138*.

49. Karp, R. M., *A Note on the Application of Graph Theory to Digital Computer Programming*, Information and Control, Vol. 3, 179-190, 1960.

50. Schurmann, A., *The Application of Graphs to the Analysis of Distribution of Loops in a Program*, Information and Control, Vol. 7, *275-282*, 1964.

51. Hain, G., *Automatic Flow Chart Design*, Proceedings of the Twentieth Association for Computing Machinery National Conference, 1965, *513-523*.

52. Hamburger, Paul E., *On an Automated Method of Symbolically Analyzing Times of Computer Programs*, Proceeds of the Association for Computing Machinery National Meeting, 1966, *321-330*.

53. Marimont, Rosalind B., *Applications of Graphs and Boolean Matrices to Computer Programming*, Society of Industrial and Applied Mathematics Review, Vol. 2, No. 4, *259-268*, October 1960.

54. Martin ahd Estrin, *Experiments on Models of Computations and Systems*, Institute of Electrical and Electronics Engineers, Transactions, Vol. EC-16, No. 1, *60-69*, February 1967.

55. Martin and Estrin, *Models of Computations and Systems––Evaluation of Vertex Probabilities in Graph Models of Computations*, Journal of the Association for Computing Machinery, Vol. 12, No. 7, *281-299*, April 1967.

56. Martin and Estrin, *Models of Computational Systems–Cyclic to Acyclic Graph Transformations*, Institute of Electrical and Electronics Engineres, Transactions, Vol. EC-16, No. 1, *70-79*, February 1967.

57 Ramamoorthy, C. V., *Analysis of Graphs by Connectivity Considerations*, Journal of the Association for Computing Machinery, Vol. 13, No. 7, *211-222*, April 1966.

58. Ramamoorthy, C. V., *The Analytic Design of a Dynamic Look Ahead and Program Segmenting System for Multiprogrammed Computers*, Proceedings of the Association for Computing Machinery National Meeting, *229-239*, 1966.

59. Martin, D. F., *The Automatic Assignment and Sequencing of Computations on Parallel Processor Systems*, University of California, Los Angeles, Department of Engineering, Report No. 66-4, January 1966.

60. Warshall, Stephen, *A Theorem on Boolean Matrices*, Journal of the Association for Computing Machinery, Vol. 9, *11-12*.

61. McCluskey, E. J., *An Introduction to the Theory of Switching Circuits*, McGraw-Hill, 1966.

62. Littlefield, Warren M., *Interlocken*, Washington University Computer Systems Laboratory, Technical Memorandum No. 26, June, 1967.

63. Iverson, Kenneth E., *A Programming Language*, Wiley, 1962.

64. Wann, D. F., *Macromodular Implementation of a Floating Point Arithmetic Unit*, Washington University Computer Systems Laboratory, Technical Memorandum No. 5, August 1966.

65. Ellis, Robert A., Personal communication, Washington University Computer Systems Laboratory, July 1967.

66. Stucki, Mishell J., Personal communication, Washington University Computer Systems Laboratory, July 1967.

9. BIBLIOGRAPHY

Akushsky, I. Y., "Methods of Speeding Up the Operation of Digital Computers", *Proceedings of the International Conference on Information Processing,* Unesco, Paris, June 1959.

Allard, R.W., K.A. Wolf, and R.A. Zemlin, "Some Effects of the 6600 Computer on Language Structures", *Communications Association for Computing Machinery,* Vol. 7, No. 2, February 1964, *112-119.*

Amdahl, Gene M., "New Concepts in Computing System Design, *Proceedings of the Institute of Radio Engineers,* May 1962, *1073-1077.*

Amdahl, Gene M., "Multi-Computers Applied to On-Line Systems Symposium on On-Line Computing Systems", UCLA 1965, *American Data Processing Incorporated,* 38-42.

Anderson, James P., "Program Structures for Parallel Processing", *Communications of the Association for Computing Machinery,* Vol. 8, No. 12, December 1965, *786-788.*

Anderson, Hoffman, Shifman, and Williams, "D825 – A Multiple-Computer System for Command and Control", *Proceedings of the Fall Joint Computer Conference,* 1962, *86-96.*

Anderson, Sparacio, and Tomasulo, "System/360 Model 91 Machine Philosophy and Instruction Handling", *IBM Journal of Research and Development,* Vol. 11, No. 1, January 1967, *8-24.*

Anderson, Goldschmidt, Earle, and Powers, "System/360 Model 91 Floating-Point Execution Unit", *IBM Journal of Research and Development,* Vol. 11, No. 1, January 1967, *34-53.*

Aoki and Estrin, "The Fixed-Plus-Variable Computer System in Dynamic Formulation of Control System Optimization Problems", UCLA Department of Engineering, No. 60-66, 1961.

Aoki, Estrin and Mandell, "Analysis of Computing-Load Assignment in a Multi-Processor Computer", *Proceedings of the Fall Joint Computer Conference,* 1963, *147-160.*

Aoki, Estrin, and Tang, "Parallelism in Computer Organization, Random-Number Generation in the Fixed-Plus-Variable Computer System", *Proceedings of the Western Joint Computer Conference,* 1961, *157-172.*

Aschenbrenner, Richard A., Michael J. Flynn, and George A. Robinson, "Intrinsic Multiprocessing", *Proceedings of the Spring Joint Computer Conference,* 1967, *81-86.*

Aschenbrenner, R.A., and R. Mueller, *Intrinsic Multiprocessing – Analysis of Execution Efficiency,* Argonne National Laboratory Applied Mathematics Division Technical Memorandum No. 135, May 1967.

Auerbach Corporation, Editors, Standard EDP Reports, BNA Incorporated.

Baldwin, Gibson, and Poland, "A Multiprocessing Approach to a Large Computer System", *IBM Systems Journal,* Vol. 1, September 1962, *64.*

Ball, J.R., R.C. Bollinger, T.A. Jeeves, R.C. McReynolds, and D.H. Shaffer, "On the Use of the Solomon Parallel-Processing Computer", *Proceedings of the Fall Joint Computer Conference,* Vol. 22, 1962, *137-146.*

Bauer, Walter F., "Why Multi-Computers?", *Datamation,* September 1962, *51-55.*

Bekisev, G.A., "On the Disparallelization of Computer Algorithms", Vychisl Systemy, No. 5, 1962, Russian, *22-30.*

Bernstein, A.J., "Analysis of Programs for Parallel Programming", *Institute of Electrical and Electronics Engineers,* Vol. EC-15, No. 5, October 1966, *757-763.*

Bingham, H., D. Fisher, and W. Semon, *Detection of Implicit Computational Parallelism from Inptu-Output Sets,* U. S. Army Electronics Command Technical Report, ECOM-02463-1, ASTIA AD645120.

Bingham, H., D. Fisher, and W. Semon, *Detection of Essential Ordering in Compiler Language Programs,* U. S. Army Electronics Command Technical Report, ECOM-02463-2.

Blaauw, G. A., "The Structure of System/360 - Multisystem Organization, *IBM Systems Journal,* Vol. 3, No. 2, 1964, *181.*

Blaauw, Gerrit A., "IBM System/360 Multisystem Organization", *Institute of Electrical and Electronics Engineers,* 1965 International Convention Record, Part 3, *226-235.*

Bloch, Erich, "The Engineering Design of the Stretch Computer", *Proceedings of the Eastern Joint Computer Conference of IRE., 48-58.*

Blosk, R. T., "The Instruction Unit of the Stretch Computer", *Proceedings of the Eastern Joint Computer Conference of IRE,* 1960, *299-324.*

Boland, Granito, Marcotte, Messina, and Smith, "System/360 Model 91 Storage System", *International Business Machines Corp. Journal of Research and Development,* Vol. 11, No. 1, January 1967, *54-68.*

Bosset, J., "Sur Certains Aspects de la Conception Logique du Gamma 60", *Proceedings of the International Conference on Information Processing,* Unesco, Paris, 1959, *348-353.*

Boyell, R.L., "Analysis of Time-Sharing in Digital Computers", *Journal of the Society of Industrial and Applied Mathematics,* March 1960, Vol. 8, *102-124.*

Bright, Herbert S., "A Philco Multiprocessing System", *Proceedings of the Spring Joint Computer Conference,* 1964, 97-*141.*

Brown, G. W., *A New Concept in Programming Management and the Computer of the Figure,* M. Greenberger. Editor, Wiley 1962.

Buchholz, W., Editor, *Planning a Computer System,* McGraw-Hill, 1962.

Bussell, B., *Properties of a Variable Structure Computer System in the Solution of Parabolic Partial Differential Equations,* PhD Dissertation, UCLA, August 1962.

Bussell and Estrin, "An Evaluation of the Effectiveness of Parallel Processing", *Proceedings of the Pacific Conference,* May 1963, *201-220.*

Cantor, Estrin, and Turn, "Logarithmic and Exponential Function Evaluation in a Variable Structure Computer", *Institute of Radio Engineers,* April 1962, *155-164.*

Cantor, Estrin, Fraenkel, and Turn, "A Very High Speed Digital Number Sieve Mathematics of Computation", April 1962, Vol. 16, *141-154.*

Carroll, A.B. and R.T. Wetherald, "Application of Parallel Processing to Numerical Weather Prediction", *Journal of the Association for Computing Machinery,* Vol. 14, No. 13, July 1967, *591;614.*

Carroll, Gregory, Leonard, and Slotnick, "The Solomon II Computing System", *Proceedings of the International Federation of Information Processing,* Congress 65, Vol. 2.

Casale, C.T., "Planning the 3600", *Proceedings of the Eastern Joint Computer Conference, 1962, Vol.* 22, *73-85.*

Cheatham, T.E. Jr. and G.F. Leonard, "An Introduction to the CL-II Programming System", *Computer Associates Incorporated,* CA-6311-0111, November 1963.

Chen, Tien Chi, "The Overlap Design of the IBM System/360 Model 92 Central Processing Unit", *Proceedings of the Spring Joint Computer Conference,* 1964, *73-80.*

Clark, Wesley A., "The Lincoln TX-2 Computer Development" *Proceedings of the Western Joint Computer Conference,* 1957, *143-145.*

Clark, Stucki, and Ornstein, *A Macromodular Approach to Computer Design,* Washington University Computer Research Laboratory Technical Report, No. 1, February 1966.

Codd, E.F., "Multiprogram Scheduling", *Communications of the Association for Computing Machinery,* Vol.3, 1960.

Codd, E.F., "Multiprogramming", *Advances in Computers.* Vol. 3, Academic Press, 1962, *78-155.*

Coffman, E.G., *Stochastic Models of Multiple and Time-Shared Computer Operations*, UCLA Department of Engineering Report No. 66-38, June 1966.

Comfort, W.T., "A Modified Holland Machine", *Proceedings of the Fall Joint Computer Conference*, Vol. 24, 1963, 4[illegible]-493.

Comfort, W.T., "Highly Parallel Machines", *Proceedings of the 1962 Workshop on Computer Organization*, Spartan, 1963, *126-155*.

Comfort, W.T., "A Computing System Design for User Service", *Proceedings of the Fall Joint Computer Conference*, 1965, *619-626*.

Conway, Melvin E., "A Multiprocessor System Design", *Proceedings of the Fall Joint Computer Conference*, Vol. 24, 1963, *139-146*.

Corbato, F.J. and V.A. Vyssotsky, "Introduction and Overview of the Multics System", *Proceedings of the Fall Joint Computer Conference*, 1965, *185-196*.

Cotton, L.W., "Circuit Implementation of High-Speed Pipeline Systems", *Proceedings of the Fall Joint Computer Conference*, 1965, *489*.

Crane, B.A., 'Economics of the DDLM, A Batch-Fabricated Parallel Processor, *Proceedings of the Institute of Electrical and Electronics Engineers*, Symposium on Batch Fabrication, 1965, *144-149*.

Critchlow, A.J., "Generalized Multiprocessing and Multiprogramming Systems", *Proceedings of the Fall Joint Computer Conference*, Vol. 23, *107-126*.

Curtin, William A., "Multiple Computer Systems", *Advances in Computers*, Vol. 4, 1963, *245-303*.

Dahm, D.M., F.H.Gerbstadt, and M.M.Pacelli, "A System Organization for Resource Allocation", *Communications of the Association for Computing Machinery*, Vol. 10, No. 12, December 1967, *772-779*.

DeBruijn, N.G., "Additional Comments on a Problem in Concurrent Programming Control". *Communications of the Association for Computing Machinery*, Vol. 10, No. 3, March 1967, *137-138*.

Dennis, Jack B., "Segmentation and the Design of Multiprogrammed Computer Systems", *Institute of Electrical and Electronics Engineers*, 1965 International Convention Record, Part 3, *214-225*.

Dennis, Jack B. and Earl C. Vanhorn, "Programming Semantics for Multiprogrammed Computations", *Communications of the Association for Computing Machinery*, Vol. 9, No. 3, March 1966, *143-155*.

Dennis and Glaser, "The Structure of On-Line Information Processing Systems", *Second Congress on Information Systems Sciences*, Spartan, 1965, *5-14*.

Dijkstra, E.W., "Solution of a Problem in Concurrent Programming Control". *Communications of the Association for Computing Machinery*, Vol. 8, September 1965, *569*.

Dijkstra, E.W., *Cooperating Sequential Processes*, Mathematical Dept., Technological University, Eindhoven, Netherlands, September 1965.

Dobbie, J. and D. Zatyko, "A Mass Memory System Designed for the Multi-Program Multi-Processors Users", *Proceedings of the Association for Computing Machinery*, 20th National Conference, 1965, *487-500*.

Dorn, W.S., "Generalization of Horner's Rule for Polynomial Evaluation", *International Business Machines Corp. Journal*, April 1962, *239-245*.

Dorn, Hsu, and Rivlin, *Some Mathematical Aspects of Parallel Computation*, International Business Machines Research Report, RC-647.

Dreyfus, P., "Programming on a Concurrent Digital Computer", *Frontier Research on Digital Computers*, Vol. 1, University of North Carolina Summer Institute 1959.

Dreyfus, P., "France's Gamma 60", *Datamation*, Vol. 4, May-June 1958, *34-35*.

Dreyfus, P., "System Design of the Gamma 60", *Proceedings of the Western Joint Computer Conference of the Institute of Radio Engineers*, 1958, *130-133.*

Eckert, J.P., J.C. Chu, A.B. Tonik, and W.F. Smith, "Design of Univac-Larc System 1", *Proceedings of the Eastern Joint Computer Conference of the Institute of Radio Engineers.*

Epstein, Samuel D., *A General Approach to Parallel Operation in a Multiprocessor Environment,* Rome Air Development Corporation Technical Report No. RADC-TR-67-83, March 1967, ASTIA AS812274.

Estrin,Gerald, "Organization of Computer Systems — The Fixed Plus Variable Structure Computer", *Proceedings of the Western Joint Computer Conference of the Institute of Radio Engineers*, 1960, *33-40.*

Estrin,Gerald, *Microelements in Processor Networks Microelectronics and Large Systems,* Spartan,1965, *157-170.*

Estrin, Bussell, Turn, and Bibb, "Parallel Processing in a Restructurable Computer System", *Institute of Electrical and Electronics Engineers Transactions on Electronic Computers,* December 1963, *747-755.*

Estrin and Turn, "Automatic Assignment of Computations in a Variable Structure Computer System", *Institute of Electrical and Electronics engineers,* Vol. EC-12, December 1963, *755-773.*

Estrin and Viswantathan, "Organization of a Fixed-Plus-Variable Structure Computer for Computation of Eigenvalues and Eigenvectors of Real Symmetric Matrices", *Journal of the Association for Computing Machinery,* Vol. 9, January 1962, *41-60.*

Fisher, David A., *Program Analysis for Multiprocessing,* Burroughs Corporation, May 1967.

Flores, I., "Derivation of a Waiting-Time Factor for a Multiple Bank Memory", *Journal of the Association for Computing Machinery,* Vol. 11, July 1964, *265-282.*

Flynn, Michael J., "Very High-Speed Computing Systems", *Proceedings of the Institute of Electrical and Electronics Engineers,* Vol. 54, No. 2,December 1966, *1901-1909.*

Flynn, M.J., and G.M. Amdahl, "Engineering Aspects of Large High Speed Computer Design", *Microelectronics and Large Systems,* Spartan 1965, *77-96.*

Forgie, James W., "The Lincoln TX-2 Input-Output System", *Proceedings of the Western Joint Computer Conference,* 1957, *156-160.*

Frankovich, J.M. and J.H. Petersen, "A Functional Description of the Lincoln TX-2 Computer", *Proceedings of the Western Joint Computer Conference,* 1957, *146-155.*

Fuller, R.H. and R.M. Bird, "An Associative Parallel Processor with Application to Picture Processing", *Proceedings of the Fall Joint Computer Conference,* 1965, *105-116.*

Fuller, R.H., "Associative Parallel Processing", *Proceedings of the Spring Joint Computer Conference,* Vol. 30, 1967, *471-475.*

Garner, H.L., *A Study of Iterative Circuit Computers,* University of Michigan Information Systems Laboratory Report No. TDR-64-24, 1964.

Gill, S., "Parallel Programming", *Computer Journal,* Vol. 1, April 1958, *2-10.*

Gill, S., "Introduction to Time-Sharing", *Introduction to System Programming,* Wegner, Ed., Academic Press, 1964, *214-226.*

Goodman, Edith H., Ed., "Computer Yearbook and Directory", *American Data Processing Incorporated.*

Gosden, John A., "The Operations Control Center Multi-Computer Operating System", *Proceedings of the Association for Computing Machinery,* National Conference.

Gosden, J.A., "Explicit Parallel Processing Description and Control in Programs for Multi- and Uni-Processor Computers", *Proceedings of the Fall Joint Computer Conference,* Vol. 29, 1966, *651-660.*

Gotlieb, C.C., "Programming a Duplex Computer System", *Communications of the Association for Computing Machinery,* Vol. 4, November 1961, *507-513.*

Gountanis, R.J. and N.L. Viss, "A Method of Processor Selection for Interrupt Handling in a Multiprocessor System", *Proceedings of the Institute of Electrical and Electronics Engineers*, Vol. 54n12, December 1966, *1812-1819*.

Grasellí, A., "Control Units for Sequencing Complex Asynchronous Operations", *Institute of Radio Engineers*, Vol. EC-11, No. 4, 1962.

Gregory, J. and R. McReynolds, "The Solomon Computer", *IEEE Transactions on Electronic Computers*, December 1963, *774-781*.

Harper, S.D., "Automatic Parallel Processing", *Proceedings of the Computing and Data Processing Society of Canada*, Second Conference,June 1960, *321-331*.

Hawkins, J.K. and C.J. Munsey, "A Parallel Computer Organization and Mechanizations", *IEEE Transactions on Electronic Computers*, June 1963, *251-262*.

Hawkins, Joseph K., *A Highly Parallel Computing System*, IEEE Computer Group Repository R-67-122.

Heller, J., "Sequencing Aspects of Multiprogramming", *Journal of the Association for Computing Machinery*, Vol. 8, No. 3, July 1961.

Hellerman, H., *On the Organization of a Multiprogramming-Multiprocessing System*, IBM Research Report, RC-522, September 1961.

Hellerman, H., *Parallel Processing of Algebraic Expressions*, IEEE, Vol. EC-15, No. 1, February 1966.

Hobbs,L.C., "Effects of Large Arrays on Machine Organization and Hardware/Software Tradeoffs", *Proceedings of the Fall Joint Computer Conference*, 1966, *89-96*.

Holland, John, "A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously", *Proceedings of the Eastern Joint Computer Conference*, 1959, *108-113*.

Hollander, Gerhard L., "Architecture for Large Computer Systems", *Proceedings of the Spring Joint Computer Conference*, Vol. 30, 1967, [illegible]3-466.

Hoover, W.R., A. Arcand, and T.B. Miller, "A Real Time Multi-Computer System for Lunar and Planetary Space Flight Data Processing", *Proceedings of the Spring Joint Computer Conference*, 1963, *127-140*.

Hosaka,Mamoru and Takahiko Tani, "A Real Time Multicomputer System for Train Seat Reservation", *Proceedings of the International Federation of Information Processing*, Congress 65.

Hu, T.C., "Parallel Sequencing and Assembly Line Problems", *Journal of Operations Research*, Vol. 9, November 1961, *841-848*.

International Businsss Machines Corporation, *PL/I Language Specifications*, IBM Form C28-6571.

International Business Machines Corporation, *Introduction to General Purpose Systems Simulator III*, IBM Form B20-0001.

Karp, Richard M. and Raymond E. Miller, "Properties of a Model for Parallel Computations - Determinacy, Terminations, and Queueing", *Society of Industrial and Applied Mathematics*, Vol. 14, No 6, November 1966, *1391-1411*.

Katz, Jesse H., "Simulation of a Multiprocessor Computer System", *Proceedings of the Spring Joint Computer Conference*, 1966, *127-139*.

Keit, H.A., "Polymorphic Principle in Data Processing", *Institute of Radio Engineers*, Wescon Convention Record, 1960, Pt. 4, *24-28*.

Knapp,Morris A., *Parallel Processing Computer Systems*, Rome Air Development Center, Report RADC-TR-66-567, November 1966, ASTIA AD803485.

Knuth, Donald E., "Additional Comments on a Problem in Concurrent Programming Control", *Communications of the Association for Computing Machinery*, Vol. 9, No. 5, May 1966, *321-322*.

Knuth and McNeley, "SOL - A Symbolic Language for General Purpose Systems Simulation", *Institute of Electrical and Electronics Engineers*, Vol. EC-13, August 1964, *401-408.*

Knuth and McNeley, "Formal Definition of SOL", *Institute of Electrical and Electronics Engineers*, Vol.EC-13, August 1964, *409-414.*

Kolsky, H.G., "Computer Organization - A Survey of Current Trends and Problems", *Proceedings of the IEEE*, 6th Region Annual Convention 1966, *420-428.*

Lampson, B.W., W.W. Lichtenberger and M.W. Pirtle, "A User Machine in a Time-Sharing System", *Proceedings of the IEEE*, Vol. 54, No. 12, December 1966, *1766-1774.*

Lawless, W.J., *Developments in Computer Logicial Organization Advances in Electronics and Electron Physics*, Vol. 100, 1959.

Leeds and Weinberg, *Computer Programming Fundamentals*, McGraw-Hill 1966.

Lehman, M., "Serial Mode Operation and High-Speed Parallel Processing", *Proceedings of the International Federation of Information Processing*, 1965, Pt. 2, *631-633.*

Lehman, M., "A Survey of Problems and Preliminary Results Concerning Parallel Processing and Parallel Processors", *Proceedings of the IEEE*, Vol. 54, No. 12, December 1966, *1889-1901.*

Leiner, A.L., W.A. Notz, J.L. Smith, and A. Weinberger, "Organizing a Network of Computers to Meet Deadlines", *Proceedings of the Eastern Joint Computer Conference*, December 1957, *115-128.*

Leiner, A.K., J.L. Smith, W.A. Notz, and A. Weinberger, "Pilot, The NBS Mluticomputer System", *Proceedings of the Eastern Joint Computer Conference*, 1958, *71-75.*

Leiner, Notz, Smith, and Marimont, "Concurrently Operating Computer Systems", *Information Processing*, 1959, Unesco, Paris, *353-361.*

Leiner, Notz, Smith, and Weinberger, "PILOT - A New Multiple Computer System", *Journal of the Association for Computing Machinery*, Vol. 6, July 1959, *313-335.*

Leipold, K. and W. Rekowski, "A Method for the Simultaneous Processing of Several Programs", *Proceedings of the International Federation of Information Processing*, Congress 65.

Leonard, G.F. and J.R. Goodroe, "An Environment for an Operating System", *Proceedings of the Association for Computing Machinery*, 1964.

Levy, W., *Congestion in a Multi-Modular Computer System*, Pennsylvania Research Associates, April 1965.

Lewis and Mellen, "Stretching LARC's Capability by [illegible] - A New Multiprocessor System", *Symposium on Microelectronics and Large Systems*, Spartan 1964.

Littlefield, Warren M., *Interlocken*, Washington University Computer Systems Laboratory Technical Memorandum Number 26, June 1967.

Lock, Kenneth, "Structuring Programs for Multiprogram Time-Sharing On-Line Applications", *Proceedings of the Fall Joint Computer Conference*, 1965, *457-472.*

Lombardi, L.A., "Mult. Access and Multi-Computer Systems", *Data Systems Design*, Vol. 1, No. 8, 1964, *16-24.*

Lourie, N.,H.Schrimpf, R.Reach, and W.Kahn, "Arithmetic and Control Techniques in a Multiprogram Computer", *Proceedings of the Eastern Joint Computer Conference*, 1959, *75-81.*

Mayer, R.J., "Problems of Storage Allocation in a Multiprocessor Multi-Programmed System", *Communications of the Association for Computing Machinery*, Vol. 4, October 1961.

Markowitz, Hauser, and Karr, *Simscript - A Simulation Programming Language*, Prentice-Hall 1963.

Martin and Estrin, "Experiments on Models of Computations and Systems", *Institute of Electrical and Electronics Engineers*, Vol. EC-16, No. 1, February 1967, 59-69.

Martin and Estrin, "Models of Computations and Systems – Evaluation of Vertex Probabilities in Graph Models of Computrtions", *Journal of the Association for Computing Machhnery*, Vol. 14, No. 2, April 1967, *281-299*.

Martin and Estrin, "Models of Computational Systems – Cyclic to Acyclic Graph Transformations". *Institute of Electrical and Electronics Engineers*, Vol. EC-16, No. 1, February 1967, *70-79*.

Martin, D.F., *The Automatic Assignment and Sequencing of Computations on Parallel Processor Systems*, UCLA Dept. of Engineering, Report 66-4, January 1966.

Mathis, Samuel J. Jr., and Richard E.Wiley, Eds., *Symposium on Microelectronics and Large Systems*, Spartan, 1965.

Maynard, B.A. Ed., *Manual of Computer Systems*, George and Company Ltd.

Metz, Joachim, *Dimensionierung von Rechenanlagen*, Technischen Universitat Dresden.

Miller, W.F., and R. Aschenbrenner, "The GUS Multicomputer System", *IEEE Transactions on Electronic Computers*, December 1963, *671-676*.

Mills, M.R., "Operational Experience of Time Sharing and Parallel Processing", *Journal Computer*.

Miranker and Liniger, "Parallel Methods for the Numerical Integration of Ordinary Differential Equations", Math., *Journal Computer*.

Murtha, John C., "Highly Parallel Information Processing Systems", *Advances in Computers*, Vol. 7, 1966.

McCullough, Spierman, and Zurcher, "A Design for a Multiple User Multiprocessing System", *Proceedings of the Fall Joint Computer Conference*, 1965, *611-617*.

McKenney, James Larrimore, *Simultaneous Multiprogramming of Electronic Computers*, University of California, Los Angeles, Management Sciences Research Project, Research Report No. 69, February 1961.

Nekora, M.R., "Comment on a Paper on Parallel Processing", *Communications of the Association for Computing Machinery*, Vol. 4, No. 2, February 1961.

Nelson, J.C., "On the Limitation of Processor-Memory Recycle Rate in a Multiprocessing System", *Abstract only*, *IEEE* 1965, *International Convention Record*, Pt. 3, *281*.

Nievergelt, J., "Parallel Methods for Integrating Ordinary Differential Equations", *Communications of the Association for Computing Machinery*, Vol. 7, No. 12, December 1964, *731-733*.

Opler, Ascher, "Procedure-Oriented Language Statements to Facilitate Parallel Processing", *Communications of the Association for Comptuing Machinery*, Vol. 8, May 1965, *306-307*.

Opler, Ascher, "Requirements for Real-Time Languages", *Communications of the Association for Computing Machinery*, Vol. 9, No. 3, March 1966, *196-199*.

Ornstein, Severo M., Mishell J. Stucki, and Wesley A. Clark, "A Functional Description of Macromodules", *Proceedings of the Spring Joint Computer Conference*, Vol. 30, 1967, *337-355*.

Ossana, Mikus, and Dursten, "Communication and Input-Output Switching in a Multiple Computing System", *Proceedings of the Fall Joint Computer Conference*, 1965, *231-241*.

Pariser, J.J., "Multiprocessing with Floating Executive Control", *IEEE International Convention Record*, Pt. 3, 1965, *266-275*.

Parkhill, D.F., *The Challenge of the Computer Utility*, Addison-Wesley 1966.

Parnas, D.L., *Sequential Equivalents of Parallel Processes* Carnegie Institute of Technology, February 1967.

Parnas, David L., "On Facilitating Parallel and Multiprocessing in ALGOL", *Communications of the Association for Computing Machinery*, Vol. 9, No. 4, April 1966, *257*.

Pease, Marshall C., "Matrix Inversion Using Parallel Processing". *Journal of the Association for Computing Machinery*, Vol. 14, No. 4, October 1967, *757-764*.

Perkins, R., and W. McGee, "Programmed Control of Multi-Computer Systems", *Proceedings of the International Federation of Information Processing*, Congress 62.

Pickering, Mutschler, and Erickson, "Multicomputer Programming for a Large Scale Real-Time Data Processing System", *Proceedings of the Spring Joint Computer Conference*, 1964, *445-461*.

Pomeren, J.H., "An Approach to Parallel Processing", *Proceedings of the International Federation of Information Processing*, Congress 65.

Porter, R.E., "The RW-400 – A New Polymorphic Data System", *Datamation*, January-February 1960, *8-14*.

Porter, R.E., "Programming in a Polymorphic System", *Conference on Parallel Programming*, May 1960.

Proctor, *Automatic Detection of Computational Parallelism*, Burroughs Corporation, May 1964.

Pyke, Thomas N., "Time-Shared Computer Systems", *Advances in Computers*, Vol. 8,1967, Academic Press, *1-45*.

Ramamoorthy, C.V., "The Analytic Design of a Dynamic Look Ahead and Program Segmenting System for Multiprogrammed Computers, *Proceedings of the ACM National meeting*, 1966, *229-239*.

Reiter, Raymond, *A Study of a Model for Parallel Computation*, University of Michigan, Systems Engineering Laboratory, Technical Report ISL-65-4, July 1965.

Reiter, Raymond, *Initiation Timing in a Model for Parallel Computation*, University of Michigan, Systems Engineering Laboratory, Technical Report SEL-66-3, March 1966.

Reiter, Raymond, *A Study of a Model for Parallel Computations*, University of Michigan, Systems Engineering Laboratory, Technical Report SEL-67-15, June 1967.

Richards, P., *Parallel Programming*, Technical Operations Incorporated, Report No. TO-B 60-27, 1960.

Roder and Rosene, "Memory Protection in Multiprocessing Systems", *IEEE Transactions*, Vol. EC-16, No. 3, *320-326*.

Rodriguez, *Analysis and Transformation of Computational Processes*, MIT MSGM 22, MAC-M-301, March 1966.

Rosene, A.F., "Memory Allocation for Multiprocessors", Transactions of the *IEEE*, Vol. EC-16, No. 5, October 1967, *659-665*.

Rothkopf, Michael H., "Scheduling Independent Tasks on Parallel Processors". *Management Science*, Vol. 12, No. 5, January 1966, *437-447*.

Russell, E.C., *Automatic Assignment of Computational Tasks in a Variable Structure Computer*, UCLA Dept. of Engineering 63-45, 1963.

Ryle, B.L., "Multiple Programming Data Processing", *Communications of the ACM*, Vol. 3, June 1960.

Sable, et al., *Proposal for Investigation of Implicit Computational Parallelism*, AUERBACH Corp. P-6104-066, April 1966.

Savidge, David V., "An Example of Multi-Processor Organization Symposium on On-Line Computing Systems", *American Data Processing Inc.*, UCLA 1965.

Schmitt, W.T. and A.B. Tonik, "Sympathetically Programmed Computers", *Proceedings of the International Conference on Information Processing*, Unesco, Paris, 1959.

Schwartz, Eugene,S., "An Automatic Sequencing Procedure with Application to Parallel Processing", *Journal of the ACM*, Vol. 8, October 1961, *513-537*.

Schwartz, J., "Large Parallel Computers", *Journal of the ACM*, Vol. 13, No. 1, January 1966, *25-32*.

Schwartz, "A Heuristic Procedure for Parallel Sequencing with Choice of Machines", *Management Science*, Vol. 10, No. 4, July 1964.

Seeber, R.R. and A B. Lindquist, "Associative Logic for Highly Parallel Systems", *Proceedings of the Fall Joint Computer Conference*, Vol. 24, 1963, *489-493*.

Senzig, D.N. and R.V. Smith, "Computer Organization for Array Processing", *Proceedings of the Fall Joint Computer Conference*, 1965, *117-128*.

Shedler, G.S. and M.M. Lehmen, *Parallel Computation and the Solution of Polynomial Equations*", IBM Research Report RC-1550, February 1966.

Shooman, William, "Parallel Computing with Vertical Data", *Proceedings of the Eastern Joint Computer Conference*, 1960, *111-115*.

Slotnick, Borch, and McReynolds, "The Solomon Computer – A Preliminary Report", *1962 Workshop on Computer Organization*, Spartan 1963.

Slotnick, Daniel L., W. Carl Borch, and Robert C. McReynolds, "The SOLOMON Computer", *Proceedings of the Fall Joint Computer Conference*, Vol. 22, 1962, *97-107*.

Slotnick, Daniel L., "Unconventional Systems", *Proceedings of the Spring Joint Computer Conference*, 1967, Vol. 30, *477-481*.

Smith, Arthur Anshel, *Input/Output in Time-Shared, Segmented, Multiprocessor System*, MIT Project MAC, MAC-TR-28, June 1966, ASTIA AD637215.

Squire and Polais, "Programming and Design Considerations of a Highly Parallel Computer", *Proceedings of the Spring Joint Computer Conference*, 1963, *395-400*.

Stanga, D.C., "Univac 1108 Multiprocessor System", *Proceedings of the Spring Joint Computer Conference*, 1967, *67-74*.

Stone, Harold S., "One-Pass Compilation of Arithmetic Expressions for a Parallel Processor", *Communications of the ACM*, Vol. 10, No. 4, April 1967, *220-223*.

Strachey, C., "Time Sharing in Large Fast Computers", *Computers and Automation*, August 1959, *14*.

Stucki, Mishell J., Severo M. Ornstein, and Wesley A. Clark "Logical Design of Macromodules", *Proceedings of the Spring Joint Computer Conference*, 1967, Vol. 30, *357-363*.

Tasini and Winograd, "Multiple Input-Output Links in Computer Systems", *IBM Journal of Research and Development*, Vol. 6, No. 3, July 1962.

Thompson, Rankin N. and John A. Wilkinson, "The D825 Automatic Operating and Scheduling Program", *Proceedings of the Spring Joint Computer Conference*, 1963, *41-49*.

Thorlin, J.F., "Code Generation for PIE Parallel Instruction Execution Computers" *Proceedings of the Spring Joint Computer Conference*, Vol. 30, 1967, *641;643*.

Thornton, James E., "Parallel Operation in the Control Data 6600", *Proceedings of the Spring Joint Computer Conference*, 1964,*63-40*.

Tomasulo, R.M., "An Efficient Algorithm for Exploiting Multiple Arithmetic Units", *IBM Journal of Research and Development*, Vol. 11, No. 1, January 1967, *25-33*.

Turn, R., *Assignment of Inventory of a Variable Structure Computer*, UCLA Department of Engineering Report 63-5, January 1963.

Unger, S.H., "A Computer Oriented Toward Spatial Problems", *Proceedings of the Institute for Radio Engineers*, October 1958, *17-44*.

Van Horn, Earl C. Jr., *Computer Design for Asynchronously Reproducible Multiprocessing*, MIT Project MAC, PAC-TR-34, ASTIA AD650407.

Vyssotsky, Corbato, and Graham, "Structure of the Multics Supervisor", *Proceedings of the Fall Joint Computer Conference*, 1965, *203-212*.

Wald, Bruce, "Utilization of a Multiprocessor in Command and Control", *Proceedings of the Institute of Electrical and Electronics Engineers*, Vol. 54, No. 12, December 1966, *1885-1888*.

Wallace, Fife, and Rosin, *A Study of Information Flow in Multiple Computer and Multiple Console Data Processing Systems*, Rome Air Development Center, Report RADC-TDR-64-427, December 1964.

Wann, D.F., *Error Analysis in Parallel Processing*, Washington University, Computer Research Laboratory, Technical Memorandum No. 15, December 1966.

Wann, D.F., *Macromodular Implementation of a Floating Point Arithm— Unit*, Washington University, Computer Research Laboratory, Technical Memorandum No. 5, August 1966.

Wann, Ellis, Stucki, and Keller, "Problems Encountered with Control Networks in Highly Restructurable Digital Systems", *Institute of Electrical and Electronics Engineers*, First Annual Computer Conference, Sept.1967.

Ward, James A., "The Need for Faster Computers", *Proceedings of the Pacific Computer Conference, 1-4.*

Warshall, Stephen, *Some Remarks on the Design of Multi-Proeessor Computer Systems*, Computer Associates, Inc. CA-6304-0111, April 1963.

Welch, P.D., "On the Reliability of Polymorphic Systems", *IBM Systems Journal*, Vol. 4, No. 1, 1965, *43-52.*

West, G.P., "Advantages of a System Utilizing Selectable Banks of Memory Over Multiplexing Computer Accesses to a Single Large Memory", *Institute of Electrical and Electronics Engineers*, 1965 International Convention Record, Pt. 3, *276-280.*

West, George P., "The Best Approach to a Large Computing Capability", *Proceedings of the Spring Joint Computer Conference*, Vol. 30, 1967, *467-469.*

Wirth, Niklaus, "A Note on Program Structures for Parallel Processing", *Communications of the Association for Computing Machinery*, Vol. 9, No. 5, May 1966, *320-321.*

Witt, B.I., "The Functional Structure of OS/360, Part II Job and Task Management", *IBM Systems Journal*, Vol. 5, No. 1, 1966, *12-29.*

Yarbrough, Lynn D., "Some Thoughts on Parallel Processing", *Communications of the Association for Computing Machinery*, Vol. 3, No. 10, October 1960, *539.*

Yevreinov, E.Z. and Y.G. Kosarev, "High Efficiency Computing Systems", *Engineering Cybernetics*, Vol. 1, No. 4, 1963.

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Computer Systems Laboratory Washington University St. Louis, Missouri	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

Analysis of Implementation Errors in Digital Computing Systems

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Interim

5. AUTHOR(S) (First name, middle initial, last name)

Robert M. Keller and Donald F. Wann

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
March 1968	89	66

8a. CONTRACT OR GRANT NO. (1) DOD(ARPA) Contract SD-302 (2) NIH(DRFR) Grant No. FR-00218 b. PROJECT NO. (1) ARPA Project Code No. 5880 Order No. 655 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) Technical Report No. 6 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

Distribution of this document is unlimited

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY ARPA – Information Processing Techniques, Washington, D.C. N.I.H., Div. of Research

13. ABSTRACT

This report discusses problems encountered with control networks in highly restructurable digital systems. In particular the treatment of implementation errors is covered with emphasis on concurrent processing. The implementation of concurrent processing networks may result in errors which will be quite complex to detect and systematic methods are warranted. A model representing a particular type of computing system is presented, and methods for introducing concurrent control into the model discussed. The automatic detection of a certain class of errors caused by improper design of these systems is investigated. Graph theoretic representation is employed in demonstrating several error detection techniques. The properties of these techniques are compared and it is concluded that one technique, of those investigated, is of sufficient generality, thoroughness, and simplicity in implementation to be used for automatic error analysis.

DD FORM 1 NOV 65 1473 REPLACES DD FORM 1473, 1 JAN 64, WHICH IS OBSOLETE FOR ARMY USE.

Unclassified
Security Classification

Unclassified
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Concurrent						
Parallelism						
Errors in concurrent processes						
Parallel implementation errors						

Security Classification