MITNE-209

A MODULAR APPROACH TO FAULT TREE AND RELIABILITY ANALYSIS

by

Jaime Olmos
Lothar Wolf

August 1977

DEPARTMENT OF NUCLEAR ENGINEERING
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Cambridge, Massachusetts 02139

MITNE-209

A MODULAR APPROACH TO FAULT TREE AND RELIABILITY ANALYSIS

by

Jaime Olmos

Lothar Wolf

August 1977

Department of Nuclear Engineering

Massachusetts Institute of Technology

ABSTRACT

An analytical method to describe fault tree diagrams in terms of their modular composition is developed. Fault tree structures are characterized by recursively relating the top tree event to all its basic component inputs through a set of equations defining each of the modules for the fault tree. It is shown that such a modular description is an extremely valuable tool for making a quantitative analysis of fault trees.

The modularization methodology has been implemented into the PL-MOD computer code, written in PL/1 language, which is capable of modularizing fault trees containing replicated components and replicated modular gates. PL-MOD in addition can handle mutually exclusive inputs and explicit higher order symmetric (k-out of - n) gates.

The step-by-step modularization of fault trees performed by PL-MOD is demonstrated and it is shown how this procedure is only made possible through an extensive use of the list processing tools available in PL/1.

A number of nuclear reactor safety system fault trees were analyzed. PL-MOD performed the modularization and evaluation of the modular occurrence probabilities and Vesely-Fussell importance measures for these systems very efficiently. In particular its execution time for the modularization of a PWR High Pressure Injection System reduced fault tree was 25 times faster than that necessary to generate its equivalent minimal cut-set description using MOCUS, a code considered to be fast by present standards.

Inquiries about this research and for the computer program should be directed to the second author at MIT.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

ACKNOWLEDGEMENTS

The work summarized in this report was partly performed under the auspices of the U.S. Nuclear Regulatory Commission.

We are especially indebted to Dr. William E. Vesely, Special Assistant for Methodolgy at NRC, for providing the financial support.

In addition, we express our sincere appreciation to Professor Norman C. Rasmussen for his useful criticism, suggestions and for his guidance and interest in this research.

The authors wish to thank Rachel Morton who, with her limitless patience, helped to unravel the intricacies of the PL/1 programming and the compiler at MIT-IPC.

INTRODUCTION

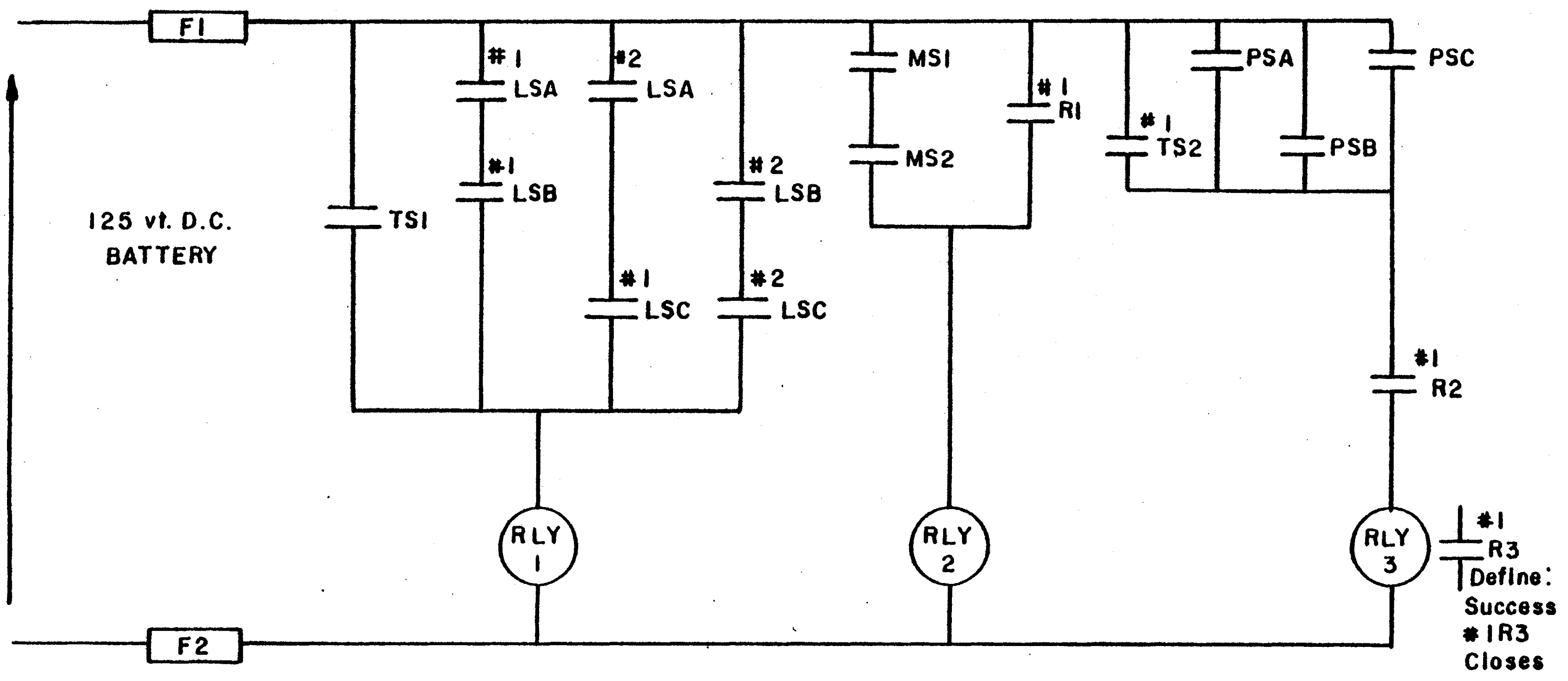
The objective of this research has been to develop and implement the modularization technique for the analysis of operating systems modeled by means of fault trees, and to apply this methodology to safety systems commonly found in nuclear reactors.

In the past the usual approach has been to describe the structure of a fault tree in terms of the minimal sets of basic event failures (cut-sets) causing overall system failure. However since for complex systems, a complete enumeration of its minimal cut-sets is not feasible, it is common practice to generate only the dominant contributor cut-sets, i.e., single, double and triple event fault cut-sets.

Figures 1.1 and 1.2 show the system and fault tree diagrams for a Standby Protective Circuit (SPC) found in reactor safety systems [18]. Inspection of the fault tree demonstrates that it is composed of 29 event inputs and 19 gates. In Table 1.1 a list is provided of the 100 minimal cut-sets associated with the SPC fault tree.

A closer scrutiny of the SPC fault tree diagram and minimal cut-set table indicates that certain classes of minimal cut-sets are closely associated to each other. Thus for example, if gate G8 is thought of as a super-component (i.e., a module) given by

$$G8 = \{C17, C18, C19, C20, C21, C22; U\}$$



F - Inline Fuse

TS - Test switches - used monthly test

LS - Level switch - tested yearly

MS - Manual switch - tested monthly

PS - Pressure switch - tested yearly

Figure 1.1 Standby Protective Circuit for Comparison Studies

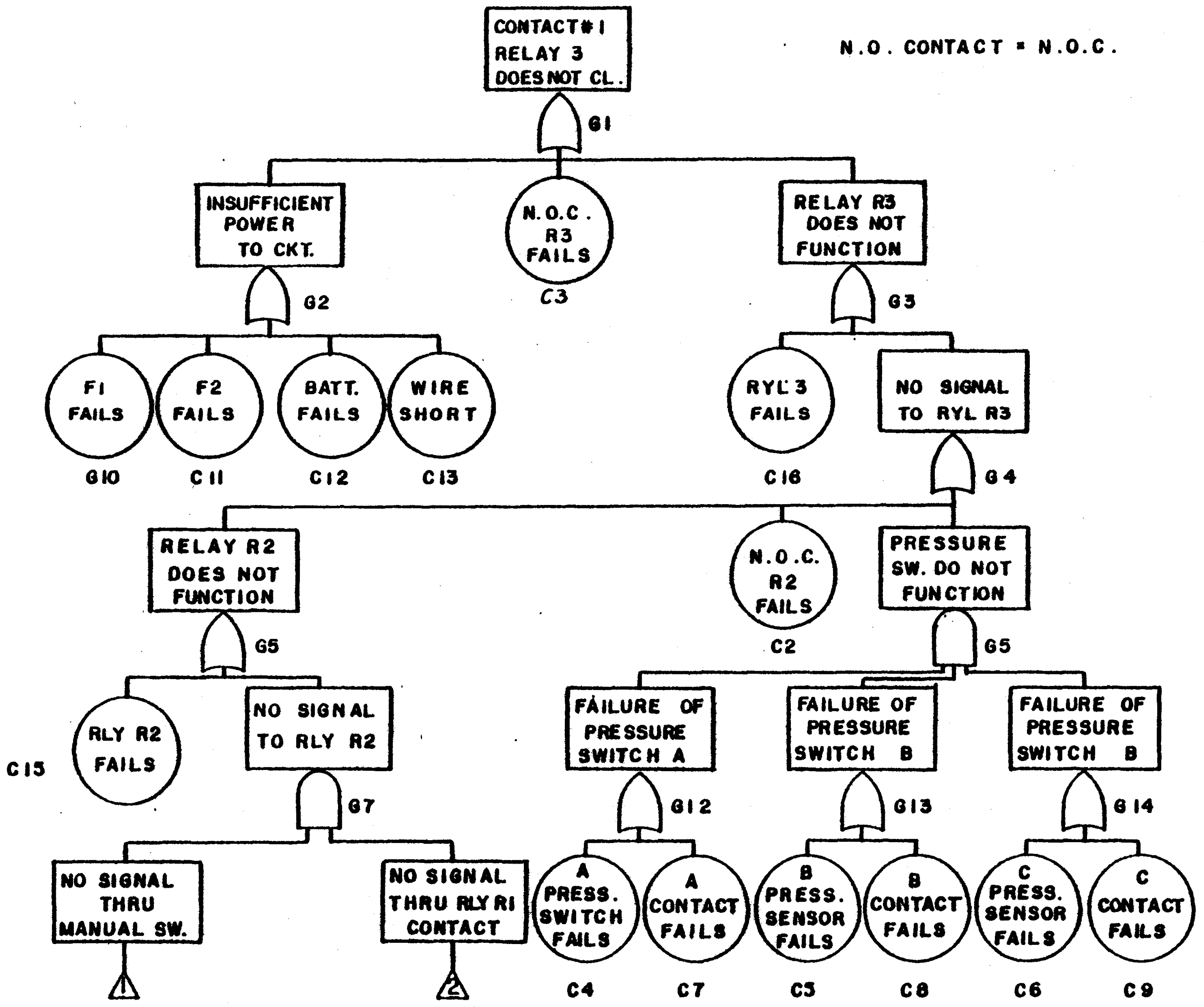


Figure 1.2 Fault Tree for Standby Protection Circuit

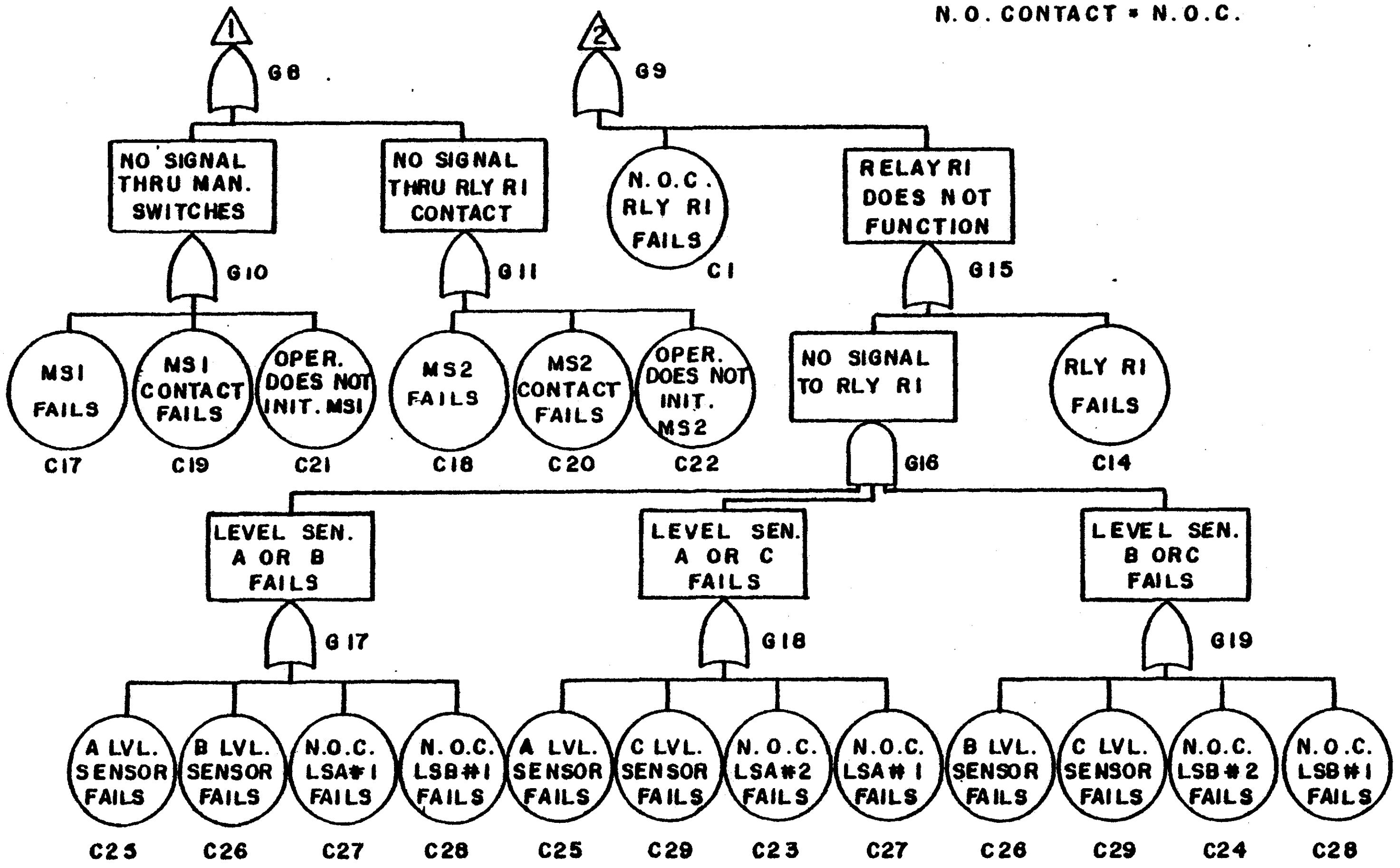


Figure 1.2 Continued
Fault Tree for Standby Protection Circuit

TABLE 1.1

MINIMAL CUT-SETS FOR THE S.P.C. FAULT TREE

SINGLE CUT-SETS

1) C10

2) C3

3) C11

4) C12

5) C13

6) C16

7) C2

8) C15

DOUBLE CUT-SETS

1) C17, C1

2) C17, C14

3) C18, C1

4) C18, C14

5) C19, C1

6) C19, C14

7) C21, C1

8) C21, C14

9) C20, C1

10) C20,C1

11) C22,C1

12) C22,C14

TABLE 1.1. CONTINUED

1) C4,C5,C6
2) C7,C5,C6
3) C4,C8,C6
4) C7,C8,C6
5) C17,C25,C24
6) C4,C5,C9
7) C18,C25,C24
8) C19,C25,C24
9) C7,C5,C9
10) C21,C25,C24
11) C20,C25,C24
12) C22,C25,C24
13) C17,C27,C24
14) C4,C8,C9
15) C18,C27,C24
16) C19,C27,C24
17) C21,C27,C24
18) C7,C8,C9
19) C20,C27,C24
20) C22,C27,C24
21) C17,C25,C26
22) C17,C25,C28
23) C17,C25,C29
24) C18,C25,C26
25) C18,C25,C28
26) C18,C25,C29
27) C19,C25,C26
28) C19,C25,C28
29) C19,C25,C29
30) C21,C25,C26
31) C21,C25,C28
32) C21,C25,C29
33) C20,C25,C26
34) C20,C25,C28
35) C20,C25,C29
36) C22,C25,C26
37) C22,C25,C28
38) C22,C25,C29
39) C17,C27,C26
40) C17,C27,C28
41) C17,C27,C29
42) C18,C27,C26
43) C18,C27,C28
44) C18,C27,C29
45) C19,C27,C26
46) C19,C27,C28
47) C19,C27,C29
48) C21,C27,C26
49) C21,C27,C28
50) C21,C27,C29
51) C20,C27,C26
52) C20,C27,C28
53) C20,C27,C29
54) C22,C27,C26
55) C22,C27,C28
56) C22,C27,C29
57) C17,C23,C26
58) C17,C23,C28
59) C18,C23,C26
60) C18,C23,C28
61) C19,C23,C26
62) C19,C23,C28
63) C21,C23,C26
64) C21,C23,C28
65) C20,C23,C26
66) C20,C23,C28
67) C22,C23,C26
68) C22,C23,C28
69) C17,C19,C26
70) C17,C29,C28
71) C18,C29,C26
72) C18,C29,C28
73) C19,C29,C26
74) C19,C29,C28
75) C21,C29,C26
76) C21,C29,C28
77) C20,C29,C26
78) C20,C29,C28
79) C22,C29,C26
80) C22,C29,C28

TABLE 1.2

MODULARIZED MINIMAL CUT-SETS

$G8 = \{C17, C18, C19, C20, C21; U\}$

	Cut-sets
1)	(G8,C1)
2)	(G8,C14)
3)	(G8,C25,C24)
4)	(G8,C25,C26)
5)	(G8,C27,C24)
6)	(G8,C25,C28)
7)	(G8,C25,C29)
8)	(G8,C27,C26)
9)	(G8,C27,C28)
10)	(G8,C27,C29)
11)	(G8,C23,C26)
12)	(G8,C23,C28)
13)	(G8,C29,C26)
14)	(G8,C29,C28)

it becomes clear that for every minimal cut-set containing component C17, five other similar cut-sets may be found with component C18,C19,C20,C21, or C22 replacing component C17, e.g. (C17,C1), (C18,C1), C19,C1), (C20,C1), (C21,C1), (C22,C1). In fact by modularizing gate G8, 14 groups of similar cut-sets will be found. Therefore, as shown in Table 1.2, the listing of 84 different minimal cut-sets would be unnecessary to describe the SPC fault tree structure by keeping track of the cut-sets affected by the modularization of gate G8.

It is clear then that there are advantages to be gained by using the modularization procedure to describe fault trees as illustrated by the above example. In this thesis, the formalism necessary to characterize fault trees in terms of their modular structures shall be presented. And the methodology adopted by the computer program PL-MOD in order to implement a modular approach to fault tree and reliability analysis will also be discussed.

The organization of the thesis is as follows:

Chapter One consists of a summary of the concepts used and of the methods devised for the safety and reliability analysis of operating systems by the fault tree technique. The structural relationship between a system and its components shall be defined in terms of a deterministic coherent structure function, while the reliability of a system will be determined as a function of the probabilistic reliabilities of its components.

Coherent structure function relationships will be shown to be describable by means of minimal cut-set and path-set representations and by Boolean algebra and truth-table methods.

Since the exact computation of the system reliability parameters is in general too difficult, appropriate bounds will be given which can be easily computed. Also, probabilistic importance measures will be introduced for the purpose of numerically ranking the various sets of fault events leading to the occurrence of the top event in order of their significance.

Chapter Two deals with the means by which the structural as well as the probabilistic analysis of fault trees may be accomplished in terms of a modular tree description.

A module is defined to be a set of components behaving as a super-component, i.e., the set affects the overall system performance only through the operational state of the super-component. Modules will be classified into "simple" (AND and OR) gate modules and higher order "prime" gate modules describable by a set of Boolean state vector equations. Exact expressions as well as bounds will be given for the probability of occurrence ("reliability") and importance value of a modular gate event, and it will be shown how these quantities of interest can be straightforwardly computed.

In Chapter Three the computer program PL-MOD written in PL-1 language will be described. It will be shown how to implement an algorithm for the modularization of fault trees directly from their diagram description. The procedure which is to accomplish this task was only made possible by an extensive use of a number of unique tools available in PL-1, among them are the options to use dynamical variables, based structures, pointers, bit-string variables, Boolean operations and functions, etc.

In Chapter IV, results are presented for the analysis performed by PL-MOD on a number of nuclear reactor safety system fault tree, namely: A Triga Scram Circuit, a Standby Protective Circuit and a PWR High Pressure Coolant Injection System. The performance of the PL-MOD code is assessed with these examples and the advantages of modularizing large fault trees instead of generating their minimal cut-set event description is demonstrated.

In Chapter V the modular approach developed throughout this thesis is summarized and a discussion is given of further possible extensions to the PL-MOD computer code.

CHAPTER ONE

FAULT TREE AND RELIABILITY ANALYSIS CONCEPTS AND METHODS

I.1. Introduction

Fault tree analysis is one of the principal methods to analyze safety systems. It is a valuable tool for identifying potential accidents in a system design, and for predicting the most likely causes of system failure in the event of system breakdown [3].

In this chapter the basic concepts necessary for the structural analysis and probabilistic evaluation of fault trees are presented. In addition a review is given of the current methods used to analyze the logical structure of a fault tree diagram and for making a quantitative assessment of the reliability characteristics of safety systems modeled by fault trees.

I.2. Fault Tree Analysis

Fault tree analysis is a systematic procedure used to identify and record the various combinations of component fault states and other events that can result in a predefined undesired state of a system [19]. Fault trees are schematically represented by a logic diagram in which the various component failures and fault events combine through a set of logical gate operators leading to the top tree event defined as an undesired state of the system.

The term event, denotes a dynamical change of state occurring to a system element or to a set of system elements [3].

The symbols shown in Figure 1.3 represent the different type of tree events and logical gate operators commonly found in fault trees. In addition to the usual AND and OR gate operators, the less often used NOT gate operator has been included. A fault tree example is given in Figure 1.4 which will be used throughout to illustrate some of the concepts and methods dealt with in this chapter. Notice that for the example I fault tree the basic fault events 3 and 7 are twice replicated in the fault tree.

The following definitions will be used to develop the subject of fault tree analysis [7].

Branch: when a fault event is further developed, the subtree which results is called a branch. Thus, for the fault tree example I, a branch corresponds to each intermediate gate event E2,E3,E4,E5,E6,E7,E8.

Gate Domain: The set of all basic events that logically interact to produce an intermediate gate event is defined to be the domain for the intermediate gate.

Independent Gate Branch: If the domain of an intermediate gate is disjoint from the rest of the branches found elsewhere in the tree, then it is called an independent gate branch. Thus, for fault tree example I only gate events E4 and E5 are independent branches since they include no basic event replicated elsewhere in the fault tree.

Module: Since an independent gate branch does not contain in its domain any basic events appearing elsewhere in the tree, then the effect that these basic events have on the

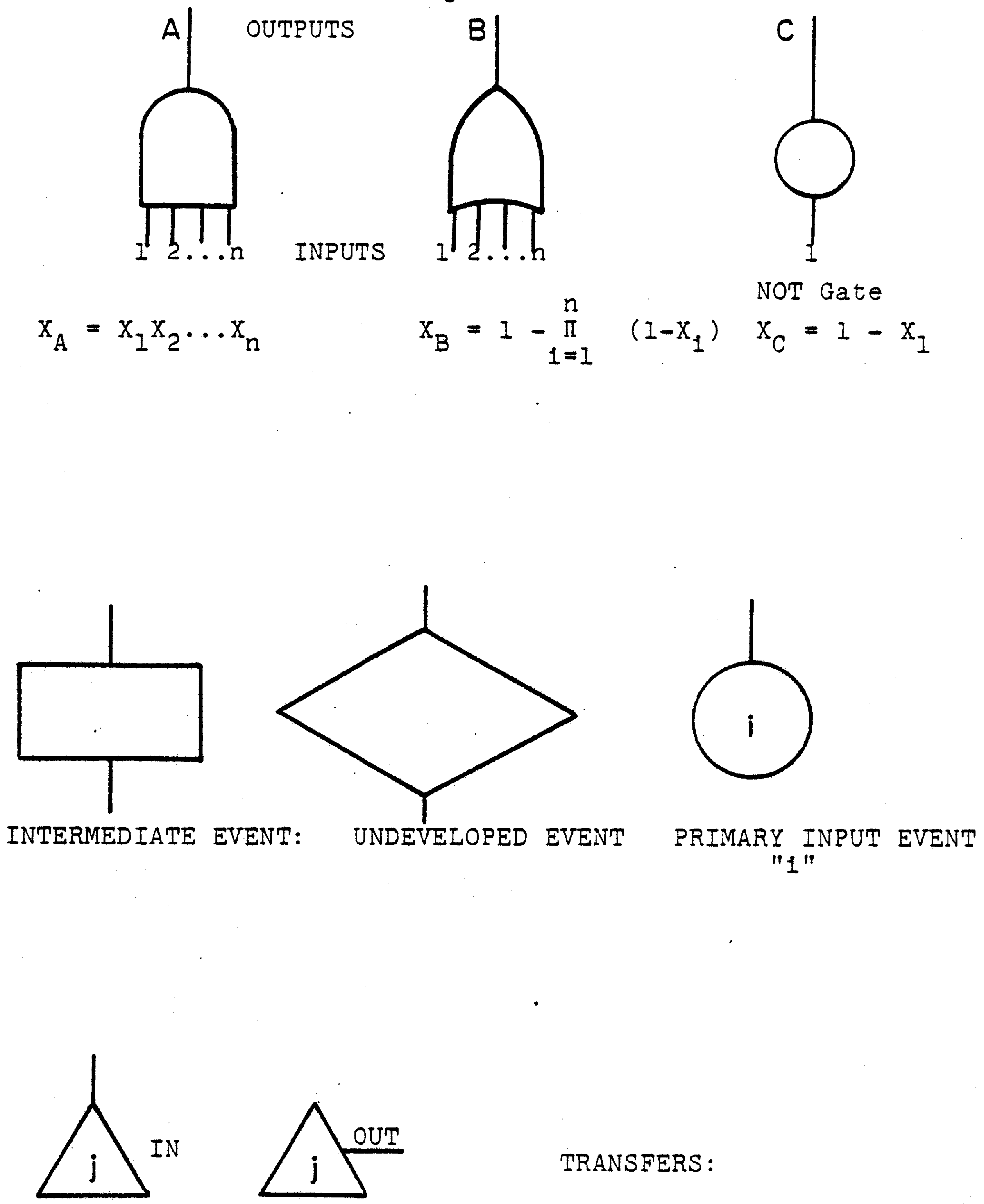


FIGURE 1.3 FAULT TREE SYMBOLS

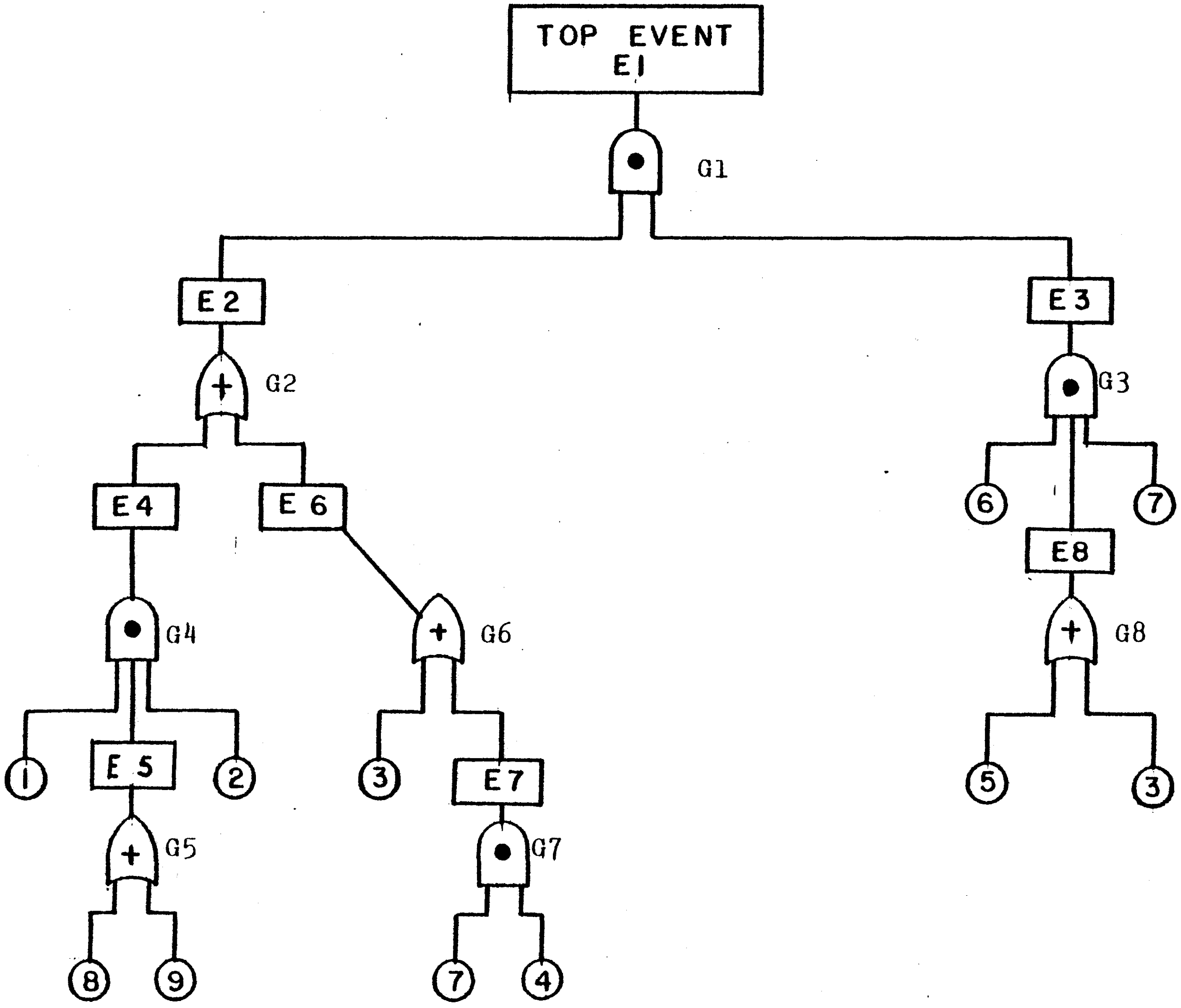


FIGURE 1.4   FAULT TREE EXAMPLE I

event is only through the functional state (failed or unfailed) of the gate event for the branch. Hence, it interacts with the rest of the tree as a super-component which in the context of coherent structure theory is equivalent to a module. Thus, for fault tree example I gates E4 and E5 corresponds to modules M4,M5 given by

$$M_5 = \{8,9,U\}$$
$$M_4 = \{1,2,M_5;\Omega\}$$

Where $U \equiv$ event union (OR) operator and

$\Omega \equiv$ event intersection (AND) operator.

It should be mentioned here that since both basic events and complete fault trees are fully characterized, as far as the tree logic is concerned, by being either in a failed or unfailed functional state, they therefore may also be considered to be modules.

I.3. Coherent Structure Theory

Let $N = (C_1,C_2...,C_n)$ be a set of basic events, and let

$$y_i = \begin{cases} 1 \text{ if basic event } i \text{ has occurred} & (1.1) \\ 0 \text{ otherwise} \end{cases}$$

Then $\underset{\sim}{Y}^N = (y_1,y_2,...,y_n)$ defines the vector of basic event outcomes, and the Boolean structure function [1] $\Phi(\underset{\sim}{Y}^N)$ determines the overall state of the system, i.e.

$$\Phi(\underset{\Rightarrow}{Y}^N) = \begin{cases} 1 \text{ if the TOP event occurs} \\ 0 \text{ otherwise} \end{cases} \qquad (1.2)$$

Consider the basic AND and OR logic gates operating on the set N of inputs. The structure function representing an AND gate is given by

$$\phi_{AND}(\underset{\sim}{Y}^N) = Y_1 , Y_2 , \ldots , Y_n \equiv \prod_{i-1}^{n} Y_i \qquad (1.3)$$

while an OR gate is represented by

$$\phi_{OR}(\underset{\sim}{Y}^N) = 1 - (1-y_1)(1 - Y_2) \ldots (1-y_n)$$

$$\equiv \coprod_{i=1}^{n} y_i \qquad (1.4)$$

In general a Boolean structure function will define a coherent system provided

(a) $\phi(\underset{\sim}{Y}^N)$ is an increasing function of each basic event Boolean indicator y_i, i.e.,

(1.5)

$$\phi(Y_1,Y_2,\ldots,Y_i = 0,\ldots,Y_n) \leq \phi(Y_1,Y_2,\ldots Y_i = 1,\ldots,Y_n)$$

(b) each basic event is relevant to the outcome, i.e., no basic event Boolean indicator y_i exists such that

$$\phi(y_1,y_2,\ldots,y_i = 0,\ldots,y_n) = \phi(y_1,y_2,\ldots,y_i=1,\ldots,y_n)$$

for all values of y_j $(j-1,2,\ldots,i-1,i+1,\ldots n)$

Using the following notational convention

$\phi(y_1,\ldots,y_i = 1,\ldots,y_n) = (1_i,Y),(Y_1,\ldots,y_i = 0,\ldots,y_n) = (0_i,\underset{\rightarrow}{Y})$ conditions (a) and (b) may be rewritten as

$$\text{(a)} \quad \phi(0_i,\underset{\rightarrow}{Y}) \leq \phi(1_i,Y) \text{ for all } (i,\underset{\rightarrow}{Y}) \qquad (1.7)$$

$$\text{and (b)} \quad \phi(0_i,\underset{\rightarrow}{Y}) \neq \phi(1_i,\underset{\rightarrow}{Y}) \text{ for some } (i,\underset{\rightarrow}{Y}) \qquad (1.8)$$

with $(i,\underset{\rightarrow}{Y})$ representing any of the 2^{n-1} vectors $(y_1,y_2,\ldots,y_i$ fixed, $y_{i+1},\ldots,y_n)$.

It should be pointed out that fault tree diagrams which include the NOT gate operator do not obey condition (a) and are therefore represented by a Boolean function which is not coherent. Thus, a single event Y_i operated by a NOT gate will be given by

$$\phi_{NOT}(Y_i) = 1-Y_i \qquad (1.9)$$

with $$\phi_{NOT}(0) = 1 > \phi_{NOT}(1) = 0 \qquad (1.10)$$

I.3.1. Dual Coherent Structures

A fault tree used for studying a safety system will have as its top event an overall system malfunction. However, for reliability considerations one may be interested in modeling the system with a diagram showing the occurrence of an unfailed functional state as its top event. Such a diagram may be easily obtained from the original fault tree by replacing its OR gates by AND gates and viceversa, and by replacing all basic event failures by the non-occurrence of such faults. The resulting diagram is called a dual fault tree.

In terms of coherent structures, the Boolean function describing a dual fault tree will be given by

$$\phi^D(\vec{Y}') = 1 - \phi(\vec{1} - \vec{Y}') \tag{1.11}$$

with ϕ associated with the original tree, $\vec{Y}'$ representing the Boolean vector of basic success events and $1 - \vec{Y}' = (1-Y_1', 1-Y_2', \ldots, 1-Y_n')$.

Thus, as expected, AND gate structure functions will be dual to OR gates and viceversa since

$$\phi_{AND}(\vec{Y}^N) = y_1, y_2, \ldots y_n \Rightarrow \phi^D_{AND} = 1-(1-y_1),,,(1-y_n)$$
$$= \phi_{OR} \tag{1.12}$$

and $$\phi_{OR}(\vec{y}^n) = 1-(1-y_1)\ldots(1-y_n) \Rightarrow \phi^D_{OR} = 1-(1-(1-1+y_1)$$
$$,\ldots, (1-1+y_n) = \phi_{AND} \tag{1.13}$$

I.3.2. <u>Minimal Cut-Set and Path-Set Representations of Coherent Structures</u>

A cut-set is a group of basic fault events whose occurrence will cause the top tree fault event to occur, while a path-set is a group of basic fault events whose non-occurrence will insure the non-occurrence of the top tree fault event. Furthermore a cut-set (or path-set) is minimal if it cannot be further reduced and still remains being a cut-set (or path-set).

As may be verified the minimal cut-sets corresponding to fault tree example 1 are

$$K_1 = (3,6,7)$$
$$K_2 = (4,5,6,7)$$
$$K_3 = (1,2,5,6,7,8)$$
$$K_4 = (1,2,5,6,7,9)$$

From this, the minimal path-sets may now be derived by taking minimal groups of elements P_i such that no minimal cut-set may be found which contains no element in the group P_i. Thus, for example element 7 by itself forms a minimal path-set since it is found in all universal cut-sets K_1, K_2, K_3, K_4. Hence $P_1 = (7)$, similarly, the remaining min. path-sets for the fault tree may be deduced to be

$$P_2 = (6)$$
$$P_3 = (3,5)$$
$$P_4 = (2,3,4)$$
$$P_5 = (1,3,4)$$
$$P_6 = (3,4,8,9)$$

Given the complete set of minimal cut-sets K_j ($j = 1,2,\ldots, t$) for a fault tree, its coherent structure may be expressed in terms of a set of minimal cut-set structure functions defined by

$$k_j = \prod_{i \varepsilon K_j} Y_i \qquad (1.14)$$

$$(j=1,2,\ldots,t)$$

as follows $$\phi(\underset{\rightarrow}{Y}^N) = 1 - \prod_{y=1}^{t} (1-k_j) = \coprod_{j=1}^{t} k_j \qquad (1.15)$$

should all elements in a min cut-set K_j fail (i.e., $y_i = 1$ for all $i_\varepsilon K_j$) then$\Rightarrow k_j = 1 \Rightarrow \phi = 1$.

In a similar way the coherent structure for a fault tree may be expressed in terms of its min path-set structure function defined by

$$P_j = 1 - \prod_{i\varepsilon P_j} (1- y_i) = \coprod_{i\varepsilon P_j} y_i \qquad (1.16)$$

$$(j = 1,2,\ldots,h)$$

as

$$\phi(\underset{\rightarrow}{Y}^N) = \prod_{j=1}^{h} P_j \qquad (1.17)$$

Should all elements in a min path-set not fail (i.e., y_i=0 for all $i_\varepsilon P_j$) $\Rightarrow P_j = 0 \Rightarrow \phi = 0$.

I.3.3. Simple and Higher Order Coherent Structure Gates

The minimal cut-set representation for an AND gate structure consists of a single cut-set

$$K = (C_1,C_2,\ldots,C_n) \qquad (1.18)$$

with C_i denoting the i-th event input to the AND gate, hence

$$\phi_{AND} = k = \prod_{i=1}^{n} y_i \qquad (1.19)$$

Similarly the minimal path-set representation for an OR gate structure consists of a single path-set

$$P = (C_1, C_2, \ldots, C_n) \qquad (1.20)$$

hence

$$\phi_{OR} = P = \coprod_{i=1}^{n} y_i \qquad (1.21)$$

Because of their simple cut-set and path-set representation, AND and OR gates are named 'simple' coherent structure gates. It is possible however to define other gates $\sigma(\underset{\rightarrow}{Y}^N)$ which operate on the set of Boolean indicator inputs $(y_1, y_2, \ldots, y_n)$ by characterizing them in terms of two or more minimal cut-sets or path-sets. Such gates are defined to be higher order gate structures. Thus for example given a set of three basic events (C_1, C_2, C_3), the following higher order gates may be defined (Figure 1.5)

$$\begin{array}{ll} \sigma\ 1; & (C_1) \\ & (C_2, C_3) \\ \sigma\ 2; & (C_1, C_2) \\ & (C_2, C_3) \end{array} \qquad (1.22)$$

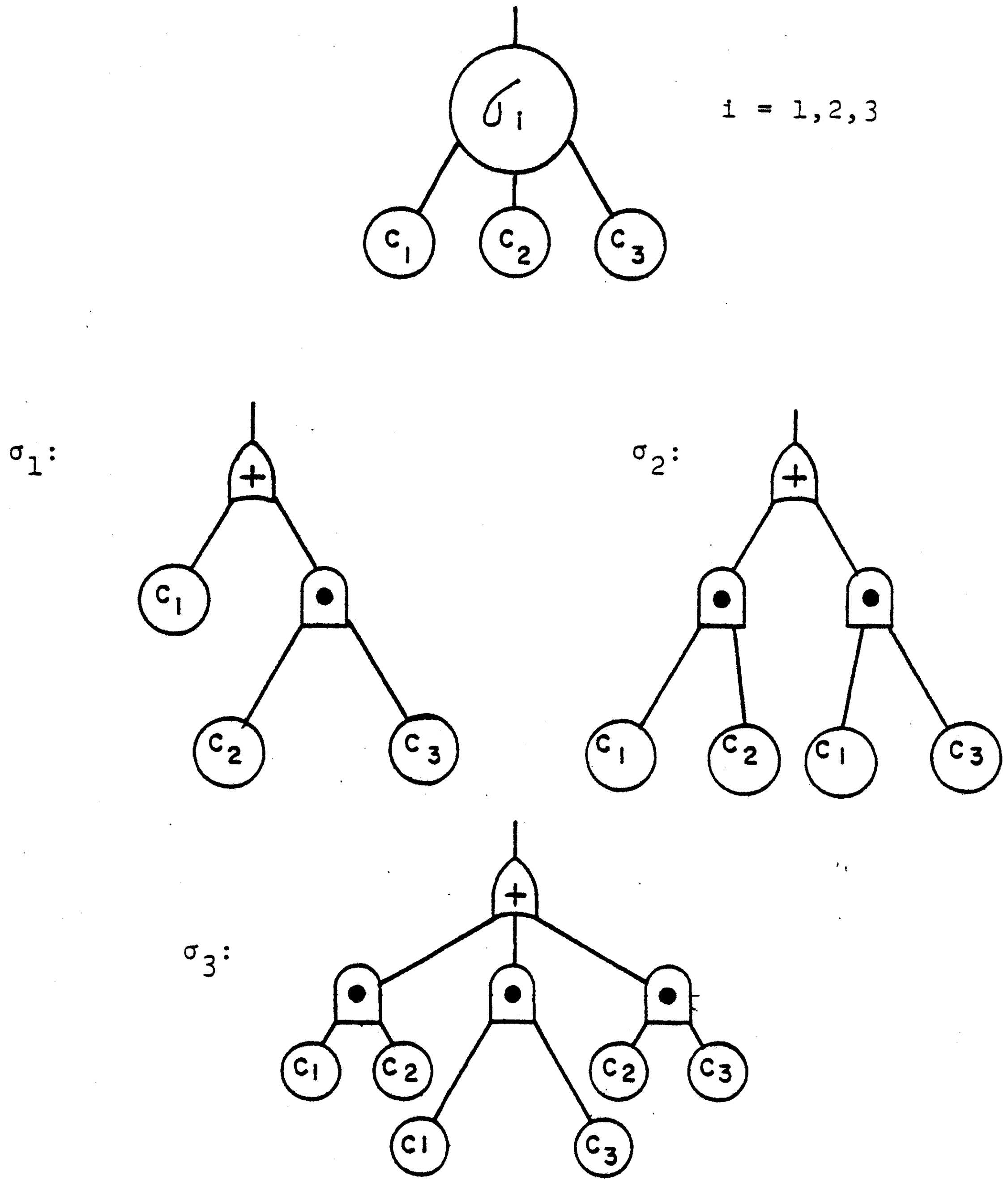


FIGURE 1.5

HIGHER ORDER STRUCTURES FOR A SET OF THREE INPUTS

σ 3: (C_1, C_2)
(C_1, C_3)
(C_2, C_3)

Each of the above gates exemplify the different characteristics that a higher order gate structure $\sigma(Y^N)$ operating on a set of event $(C_1, C_2 \ldots, C_n)$ may have. Thus, since for gate σ_1 its two cut-sets are disjoint, a fault tree diagram including no replicated events may be drawn which represents the gate. Furthermore σ_1 may be decomposed into two disjoint coherent structures σ_1, σ_2 as

$$\sigma_1 = 1 - (1 - \phi_1)(1-\phi_2) \text{ with } \phi_1 = y_1, \text{ and}$$

$$\phi_2 = y_2 y_3$$

In Chapter Two it will be shown that such a decomposition amounts the modularization of a fault tree.

Both gates, σ_2 and σ_3, do not contain any minimal cut-set which are disjoint to the others defining the gate structure. As a result such higher order structures will be called 'prime' gates since they do not allow for any further structural decomposition. If a higher order prime gate is represented by an equivalent diagram of AND and OR gates, then the gate at the top of the diagram is named the parent gate for the higher order structure.

Gate σ_3 is called symmetric since the order of its inputs does not alter its structure, i.e.

$$\sigma_3(y_1,y_2,y_3) = \sigma_3(y_1,y_3,y_2)=\sigma_3(y_3,y_2,y_1)$$
$$=\sigma_3(y_2,y_1,y_3) = \sigma_3(y_2,y_3,y_1) = \sigma_3(y_3,y_1,y_2) \qquad (1.20)$$

Symmetric gates are in fact completely defined by specifying the number k out of the n basic events necessary to cause the gate event to occur (k-out of-n). In contrast gate σ_2 is an asymmetric prime gate requiring its full min cut-set listing for its definition.

In terms of a higher order structure, fault tree example I is given by

$$\text{TOP}: \quad \begin{array}{l} (C_3,M_1) \\ (C_4,C_5,M_1) \\ (M_2,M_1) \end{array} \qquad (1.21)$$

with $\phi_{M2} = \phi_1 \cdot \phi_2$, $\phi_1 = y_1 \cdot y_2, \phi_2 = 1=(1-y_8)(1-y_9)$

and $\phi_{M1} = y_6 \cdot y_7$

I.4. Probabilistic Evaluation of Fault Trees

Given a coherent structure function $\phi(Y^N)$ which relates the occurrence of a top event to a set $(C_1,C_2,\ldots,C_n)$ of basic event occurrences each represented by a Boolean indicator variable $Y_i(i,1,2,\ldots,n)$ in the coherent structure expression it should be possible to find the probability of occurrence for the TOP event, P(TOP), as a function of the occurrence probabilities for each basic event P_i $(i=1,2,\ldots,n)$.

Formally, the occurrence probability for event C_i is obtained by applying the expectation value operator E to the Boolean variable Y_i, i.e.,

$$P_i = E_{Y_i} = P(Y_i = 1) \tag{1.22}$$

similarly, for the coherent structure $\phi(\underset{\sim}{Y}^N)$ the TOP event occurrence P(TOP) is given by

$$P(TOP) = E\phi(\underset{\sim}{Y}^N) = P(\phi(\underset{\sim}{Y}^N) = 1) \tag{1.23}$$

Assuming all basic event probabilities to be statistically independent it is possible to express P(TOP) as

$$P(TOP) = P(\phi(\underset{\sim}{Y}^N) = 1) = h(\underset{\sim}{P}) \quad \text{with } \underset{\sim}{P} = (P_{,}P_2,\ldots,P_n). \tag{1.24}$$

$h(\underset{\sim}{P})$ is commonly referred to as the reliability function by coherent structure theorists [1]. It must be realized however that when the coherent structure represents a fault tree, $h(\underset{\sim}{P})$ measures the unreliability of a system defined as the probability that the system is in a failed state.

In general the occurrence probability P_i for each basic fault event input will be a time dependent function, i.e., $P_i(t)$. For these cases one is interested in addition to find the unreliability of the system as a function of time, in evaluating the asymptotic system unavailability given by

$$U = \lim_{t\to\infty} h\,(\underset{\sim}{P}(t)) = h(\underset{\sim}{u}) \qquad (1.25)$$

with $\underset{\sim}{u} = (u_1,u_2,\ldots,u_n)$ measuring the unavailability for component i, i.e. $u_i = \lim\limits_{t\to\infty} P_i(t)$.

By using a minimal cut-set or path-set representation for the coherent structure function (equations 1.15 and 1.17) $h(\underset{\sim}{P})$ may be computed as

$$h(\underset{\sim}{P}) = E\Big(\coprod_{j=1}^{t} \prod_{i\,\varepsilon\,K_j} y_i\Big) = E\Big(\coprod_{j=1}^{h} \prod_{i\,\varepsilon\,P_j} y_i\Big) \qquad (1.26)$$

However since in general a basic event may appear in more than one min cut-set (or path-set) it follows that the probability of occurrence for a min cut-set (or path-set) event is not statistically independent of the other min cut-sets (or path-sets) defining the structure. Hence, the expectation value operator does not commute with the first (Pi) operator and (ip) operator indicated in Equation (1.26). To illustrate this, consider the coherent structure example σ_2 given in Equation (1.22).

$$\sigma_2 = \coprod_{j=1}^{2} \prod_{i\,\varepsilon\,K_j} y_i = \prod_{j=1}^{2} \coprod_{i\,\varepsilon\,P_j} y_i \qquad (1.27)$$

with $K_1 = (C_1,C_2)$, $K_2 = (C_2,C_3)$ and $P_1 = (C_2)$, $P_2 = (C_1,C_3)$.

$P_2(y_1,y_2,y_3)$ will be given by either of the following two expressions

$$\sigma_2 = 1 - (1-y_1y_2)(1-y_2y_3) \text{ (cut-sets)} \qquad (1.28)$$

or $$\sigma_2 = y_2(1-(1-y_1)(1-y_3)) \text{ (path-sets)} \qquad (1.29)$$

Since a Boolean variable y_i may only equal 0 or 1, then the idempotency rule applies, i.e., $y_i^2 = y_i$. Hence equations (1.28) and (1.29) further reduce to

$$\sigma_2 = 1 - (1-y_1y_2 - y_2y_3 + y_1y_2y_3)$$

and $$\sigma_2 = y_2 - y_2(1 + y_1y_3 - y_1 - y_3) \qquad (1.30)$$

therefore

$$\sigma_2 = y_1y_2 + y_2y_3 - y_1y_2y_3 \qquad (1.31)$$

and

$$E\sigma_2 = P_1P_2 + P_2P_3 - P_1P_2P_3$$

however

$$E\sigma_2 \neq \coprod_{j=1}^{2} E(\prod_{i \varepsilon K_j} y_i) = P_1P_2+P_2P_3 - P_1P_2^2P_3 \qquad (1.32)$$

Thus, in general,

$$h(\underset{\rightarrow}{P}) \neq \coprod_{j=1}^{t} \prod_{i \varepsilon K_j} P_i \text{ and } h(\underset{\rightarrow}{P}) \neq \prod_{j=1}^{h} \coprod_{i \varepsilon P_j} P_i. \qquad (1.33)$$

Esary and Proschan [8] have nevertheless proved that the above expressions give an upper and lower bound for h(P), i.e.

$$\prod_{j=1}^{h} \coprod_{i \varepsilon P_j} P_i \leq P(TOP) = h(\underset{\rightarrow}{P}) \leq \coprod_{j=1}^{t} \prod_{i \varepsilon K_j} P_i \qquad (1.34)$$

These bounds are known respectively as the minimal cut upper bound and minimal path lower bound.

The minimal cut upper bound may be further simplified by making a first order expansion of the full expression yielding

$$h(\underset{\sim}{P}) \leq \sum_{j=1}^{t} \prod_{i \varepsilon K_j} P_i \qquad (1.35)$$

which is the rare-event approximation to the minimal cut upper bound and neglects the simultaneous occurrence of minimal cut-sets. For values of $P_i<10^{-2}$ Equation (1.35) may be safely used.

I.5. Importance Measures for System Components and Fault Tree Events

Given a system made up by a network of components which performs a specific task or function, as a result of the system's structural arrangement only, some components will be more critical than others to the functioning of the system. Moreover a component's reliability will also be a factor in assessing its importance in determining the overall functional state of the system.

I.5.1. Structural Importance

The importance of a component purely by virtue of the role it plays in a system's structure characterized by the coherent structure $\phi(\underset{\sim}{Y})$ may be measured by

$$I_{\phi}^{S}(i) = \frac{1}{2^{n-1}} \sum_{\underset{\sim}{y}, y_i \text{ fixed}} [\phi(1_i,\underset{\sim}{Y}) - \phi(0_i,\underset{\sim}{Y})]$$

(1.36)

By fixing the value of Boolean variable y_i, 2^{n-1} possible state vectors $(y_1,y_2,\ldots,y_{i-1},y_i \text{ fixed}, y_{i+1},\ldots,y_n)$ may be found for each such vector the i-th event will be critical to the overall state of the system if

$$\phi(1_i,\underset{\sim}{Y}) = 1 \text{ and } \phi(0_i,\underset{\sim}{Y}) = 0, \text{ i.e.}$$
$$\phi(1_i,\underset{\sim}{Y}) - \phi(0_i,\underset{\sim}{Y}) = 1 \qquad (1.37)$$

Hence the structural importance $I_{\phi}^{S}(i)$ will rank each basic event i according to the number of critical state vectors that may be associated with the event.

I.5.2. Birnbaum's Importance

In terms of $\phi(1_i,\underset{\sim}{Y})$ and $\phi(0_i,\underset{\sim}{Y})$, the coherent structure function $\phi(\underset{\sim}{Y})$ is given by

$$\phi(\underset{\sim}{Y}) = Y_i\ \phi(1_i,\underset{\sim}{Y}) + (1-Y_i)\ \phi(0_i,\underset{\sim}{Y}) \qquad (1.38)$$

as may be verified since $\phi(0_i,\underset{\sim}{Y}) = (0)\ \phi(1_i,\underset{\sim}{Y}) + (1-0)\phi(0_i,\underset{\sim}{Y})$ and $\phi(1_i,\underset{\sim}{Y}) = (1)\phi\ (1_i,\underset{\sim}{Y}) + (1-1)\ \phi(0_i,\underset{\sim}{Y})$. Therefore by applying the expectation value operator E to equation (1.38) h(P) will be found to be given by

$$h(\underset{\sim}{P}) = E\ \phi(\underset{\sim}{Y}) = (EY_i)(E\phi(1_i,\underset{\sim}{Y})) + (1-EY_i)(E\phi(0_i,\underset{\sim}{Y}))$$
$$\Rightarrow h(\underset{\sim}{P}) = P_i h(1_i,\underset{\sim}{P}) + (1-P_i)h(0_i,\underset{\sim}{P}) \quad (i = 1,2,\ldots n)$$

(1.39)

Birnmaum's importance measure for event i is defined to be the partial derivative of h(P) with respect to P_i, i.e.,

$$I_i^B(\underset{\sim}{P}) = \frac{\partial h(\underset{\sim}{P})}{\partial P_i} = h(1_i, \underset{\sim}{P}) - h(0_i, \underset{\sim}{P}) \qquad (1.40)$$

It is seen from Equation (1.40) that the Birnbaum importance for event i is independent of its occurrence probability P_i.

I.5.3. Criticality Importance

The criticality importance for fault tree event i is defined as the probability that event i is in a failed state and at the same time is critical to the system's failure given that the system has failed, i.e.

$$I_i^{C_r} = \frac{P_i(h(1_i, \underset{\sim}{P}) - h\ (0_i, \underset{\sim}{P}))}{h(\underset{\sim}{P})} \qquad (1.41)$$

I.5.4. Vesely-Fussell Importance

The failure of a component c_i will contribute to system failure provided at least one min cut-set containing C_i has failed. Hence, the probability for the occurrence of the union event of all minimal cut-sets containing c_i will measure the contribution of the component to the system's failure, i.e.,

$$P(\underset{i \varepsilon K_j}{U}, K_j) = P(X_K^i\ (\underset{\sim}{Y}) = 1) \qquad (1.42)$$

where $X_K^i(\underset{\sim}{Y})$ is the Boolean indicator function for the union of all cut set functions containing Boolean variable y_i, thus

$$X_K^i(\underset{\rightarrow}{Y}) = \coprod_{j=1}^{N_k^i} \prod_{\substack{\ell \varepsilon K_j \\ i \varepsilon K_j}} y_\ell \qquad (1.43)$$

with N_k^i = total number of min cut-sets containing the ith component.

The Vesely-Fussell importance measure [10] is defined as the probability that component c_i contributes to system failure given that the system has failed, hence

$$I_i^{V.F.} = \frac{h_i(\underset{\rightarrow}{P})}{h(\underset{\rightarrow}{P})} \qquad (1.44)$$

with

$$h_i(\underset{\rightarrow}{P}) = EX_K^i(\underset{\rightarrow}{Y}) = P(X_K{}^i(\underset{\rightarrow}{Y}) = 1) \qquad (1.45)$$

The Vesely-Fussell and criticality importance measures differ from each other in that component c_i will contribute to a system's failure and still not be critical to the system if at least two minimal cut-sets have failed, one containing c_i and another one not containing c_i. Nevertheless, as shown below, if the minimal cut-upper bound is used in the rare event approximation form, to evaluate both $h_i(P)$ and $h(P)$, then the value obtained for both importance measures will coincide

$$h_i(\underset{\rightarrow}{P}) \simeq \sum_{j=1}^{N_k^i} \prod_{\substack{\ell \varepsilon K_j \\ i \varepsilon K_j}} P_\ell \qquad (1.46)$$

and

$$h(\underset{\rightarrow}{P}) \simeq \sum_{j=1}^{N} \prod_{\ell \varepsilon K_\ell} P_\ell \tag{1.47}$$

hence

$$I_i^{V.F.} = \frac{h_i(\underset{\rightarrow}{P})}{h(\underset{\rightarrow}{P})} \simeq \frac{\left(\sum_{j=1}^{N_K^i} \prod_{\substack{\ell \varepsilon K_j \\ i \varepsilon K_j}} P_\ell\right)}{\left(\sum_{j=1}^{N} \prod_{\ell \varepsilon K_\ell} P_\ell\right)} \tag{1.48}$$

at the same time

$$h(\underset{\rightarrow}{P}) \simeq \sum_{j=1}^{N-N_k^i} \prod_{\substack{i \notin K_j \\ \ell \varepsilon K_j}} P_\ell + \sum_{j=1}^{N_k^i} \prod_{\substack{i \varepsilon K_j \\ \ell \varepsilon K_j}} P_\ell \tag{1.49}$$

therefore

$$h(1_i,\underset{\rightarrow}{P}) \simeq \sum_{j=1}^{N-N_k^i} \prod_{\substack{i \varepsilon K_j \\ \ell \varepsilon K_j}} P_\ell + \sum_{j=1}^{N_k^i} (1) \prod_{\substack{i \varepsilon K_j \\ \ell \varepsilon K_j \\ i \neq \ell}} P_\ell$$

and

$$h(0_i,\underset{\rightarrow}{P}) \simeq \sum_{j=1}^{N-N_k{}^i} \prod_{\substack{i \varepsilon K_j \\ \ell \varepsilon K_j}} P_\ell + \underbrace{\sum_{j=1}^{N_k^i} (0) \prod_{\substack{i \varepsilon K_j \\ \ell \varepsilon K_j \\ i \neq \ell}} P_\ell}_{0}$$

Hence (1.50)

$$I_i^{C_r} = \frac{(h(1_i,\underset{\sim}{P}) - h(0_i,\underset{\sim}{P}))}{h(\underset{\sim}{P})} P_i \simeq \frac{\left(\sum_{j=1}^{N_k^i} \prod_{\substack{i \varepsilon K_j \\ \ell \varepsilon K_j}} P_\ell\right)}{\left(\sum_{j=1}^{N} \prod_{i \varepsilon K_\ell} P_\ell\right)}$$

Thus comparing Equations (1.48) and (1.50) it is found that

$$I_i^{C_r} = I_i^{V.F.} \qquad (1.51)$$

in the rare-event approximation.

I.6. Methods for the Generation of a Minimal Cut-Set or Path Set Fault Tree Description

For a large fault tree made up of hundreds of logical gates and basic events, its total number of min cut-sets can easily amount to thousands of cut-sets. Therefore a computer program will be needed even to generate the minimal cut-set which contribute the most to system failure [22], (i.e., single, double and triple fault cut-sets).

Computer programs MOCUS [9], TREEL and MICSUP [16] implement two different algorithms for the generation of a fault tree's minimal cut-sets. Both algorithms are based on the fact that AND gates increase the size of a cut-set while OR gate increase the number of cut-sets in a fault tree. Both MOCUS and TREEL & MICSUP were written in FORTRAN and are restricted to fault tree diagrams operated by AND and OR gates only. Thus NOT gates are not allowed by either of the two codes.

I.6.1. MOCUS

Computer program MOCUS [9] was written to replace PREP [23] as a minimal cut-set generator for computer programs KITT-1 and KITT-2 which evaluate time dependent fault trees in the framework of Kinetic Tree Theory [23]. As shown in Chapter IV for the particular case of a Standby Protective Circuit, it is a considerable improvement over PREP's deterministic minimal cut-set generation option COMBO. COMBO determines the minimal cut-sets for a fault tree by considering a combination of fault events at a time and testing if the fault tree logic implies that the combination considered causes the occurrence of the TOP tree event.

The algorithm used by MOCUS starts with the TOP event of the fault tree and proceeds, by successive substitution of gate equations, to move down the tree until only basic events remain in the list of possible TOP tree event occurrence causes.

For fault tree example I the process takes the following form

STEP 1	G1
STEP 2	G2, G3
STEP 3	G4, G3
	G6, G3
STEP 4	G4, 6, 7, G8
	G6, 6, 7, G8
STEP 5	1, 2, G5, 6, 7, G8
	3, 6, 7, G8
	G7, 6, 7, G8
STEP 6	1, 2, 8, 6, 7, G8
	1, 2, 9, 6, 7, G8
	3, 6, 7, G8
STEP 7	7, 4, 6, 7, G8
	1, 2, 8, 6, 7, 5
	1, 2, 8, 6, 7, 3
	1, 2, 9, 6, 7, 5
	1, 2, 9, 6, 7, 3
	3, 6, 7, 5
	3, 6, 7, 3
	4, 6, 7, 5
	4, 6, 7, 3

Thus, the idea of the algorithm is to replace each gate by its input gates and basic events until a list matrix is constructed, all of whose entries are basic events. Each time an OR gate is substituted, rows are added to the matrix, while a substituted AND gate results in the addition of elements to an existing row.

The cut-sets obtained this way are called Boolean Indicated Cut-Sets (BICS). For fault tree example I its list of BICS will be

	BICS	
(i)	1, 2, 5, 6, 7, 8	minimal
(ii)	1, 2, 3, 6, 7, 8	non-minimal
(iii)	1, 2, 5, 6, 7, 9	minimal
(iv)	1, 2, 3, 6, 7, 9	non-minimal
(v)	3, 5, 6, 7	non-minimal
(vi)	3, 6, 7	minimal
(vii)	4, 5, 6, 7	minimal
(viii)	3, 4, 6, 7	non-minimal

If a fault tree contains replicated events then its set of BICS will include certain cut-sets which are not minimal. The minimal cut-sets (MICS) are obtained by discarding those rows which are non-minimal since they are super-sets for another row in the list. For fault tree example I the second, fourth, fifth and eighth rows are supersets for the cut-set given in the sixth row (3,6,7). Hence they must be discarded in order to obtain a list of MICS for the fault tree

MICS

1, 2, 5, 6, 7, 8

1, 2, 5, 6, 7, 9

3, 6, 7

4, 5, 6, 7

The minimal path sets for a given fault tree may be easily obtained by applying the same algorithm to its dual fault tree. Thus, for fault tree example I, MOCUS will find its min path sets by applying the algorithm to the tree diagram shown in Figure 1.6 as follows

STEP 1	G1
STEP 2	G2
	G3
STEP 3	G4, G6
	6
	7
	G8
STEP 4	1 G6
	2 G6
	G5 G6
	6
	7
	5, 3
STEP 5	1, 3, G7
	2, 3, G7

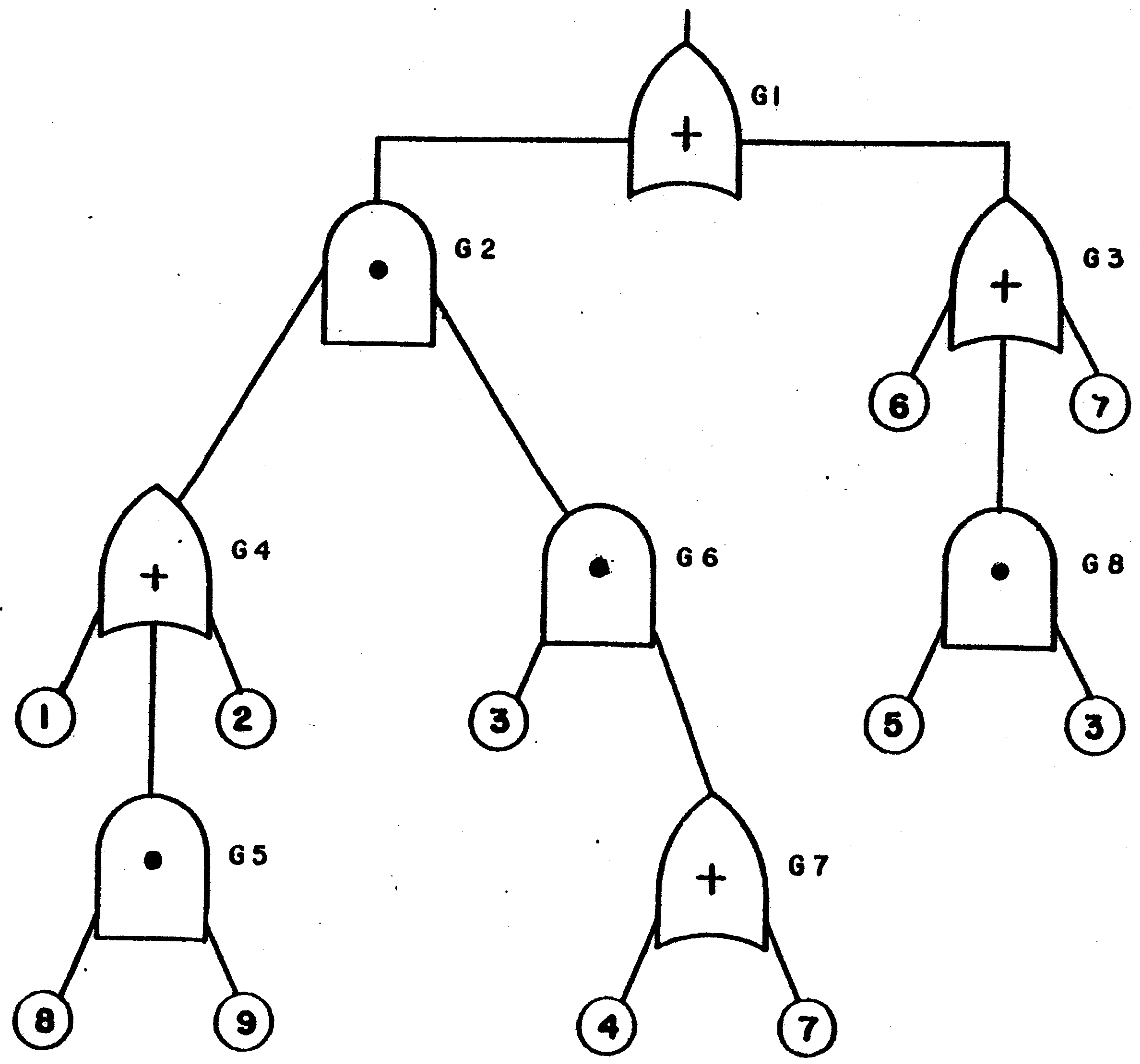


FIGURE 1.6 DUAL FAULT TREE FOR EXAMPLE 1

8, 9, 3, G7

6

7

5, 3

STEP 6 1, 3, 4

1, 3, 7

2, 3, 4

2, 3, 7

8, 9, 3, 4

8, 9, 3, 4

6

7

3, 5

Again here since the second, fourth and sixth rows are supersets to minimal path set (7), they must be discarded to obtain the set of minimal path-sets for the original fault tree

1, 3, 4

2, 3, 4

3, 4, 8, 9

3, 5

6

7

I.6.2. TREEL & MICSUP

The minimal cut-set upward algorithm [16] program obtains minimal cut-sets starting with the lowest level gate basic inputs and working upward to the TOP tree event. TREEL is a preprocessing program needed to execute MICSUP. TREEL transforms the tree into a form convenient for computer analy= sis, checks for possible errors in the tree construction and provides the number and maximum size for the Boolean Indicated Cut-sets and Path Sets. These numbers are useful since they provide an upper bound on the number and size of minimal cut-sets and path sets which characterize the fault tree, hence on that basis the user may decide to have MICSUP determine either a minimal cut-set or path-set description for the fault tree.

The algorithm used in MICSUP was given by Chatterjee [6]. As mentioned earlier it starts out with lowest level gates defined to be those gates which have basic event inputs only. The minimal cut-sets for these gates are found and are substituted as a representation for these gates. The procedure is repeated with those gates directly attached to the lowest level gates and so on, until the Boolean indicated cut-sets are found for the top event.

For fault tree example I the procedure takes the following form

```
          STEP 1      G5: 8
                          9
                      G7: 4, 7
                      G8: 3,
                      5
          STEP 2      G4: 1, 2, 8
                          1, 2, 9
                      G6: 3
```

		4, 7
	G3:	6, 7, 3
		6, 7, 5
STEP 3	G2:	1, 2, 8
		1, 2, 9
		3,
		4, 7
	G3:	6, 7, 3
		6, 7, 5
STEP 4	G1:	1, 2, 8, 6, 7, 3
		1, 2, 8, 6, 7, 5
		1, 2, 9, 6, 7, 3
		1, 2, 9, 6, 7, 5
		3, 6, 7
		3, 6, 7, 5
		4, 7, 6, 6, 3
		4, 7, 6, 5

therefore the BICS for the top event are

1, 2, 3, 6, 7, 8	non-minimal
1, 2, 5, 6, 7, 8	minimal
1, 2, 3, 6, 7, 9	non-minimal
1, 2, 5, 6, 7, 9	minimal
3, 5, 6, 7	non-minimal
3, 4, 6, 7	minimal
4, 5, 6, 7	minimal

yielding the expected TOP event MICS

1, 2, 5, 6, 7, 8

1, 2, 5, 6, 7, 9

3, 6, 7

4, 5, 6, 7

It should be noticed that in contrast to MOCUS, the MICSUP algorithm offers the advantage of generating the BICS for each gate in the tree. Therefore the minimal cut-set composition for each sub-tree in the system will be obtained by discarding at each level any non-minimal cut-sets that may appear. As a result for fault trees which include many event replications, a significant reduction in storage requirements will take place by discarding non-minimal BICS as soon as they appear for an intermediate gate in the tree. In Chapter III it will be shown that the computer program PL-MOD modularizes fault trees by an algorithm similar to that used in MICSUP in that it starts with the lowest level gates and proceeds upwards to the top event. Hence an analogous advantage to that cited for MICSUP will thereby apply for PL-MOD.

I.7. Methods for the Manipulation of Boolean Equations Describing a Fault Tree

In section I.3.2 coherent structure functions were expressed in terms of their minimal cut-set description as

$$\phi(\underset{\sim}{Y}^N) = \coprod_{j=1}^{t} k_j = \coprod_{j=1}^{t} \prod_{i \varepsilon K_J} y_i \qquad (1.52)$$

What this equation signifies is that the TOP event of a fault tree is given by the union of all its minimal cut-set event

K_i (i - 1,2,...,t), thus

$$TOP = K_1 U K_2 U \ldots U K_t \qquad (1,53)$$

with

$$K_i = (C_{i_1}, C_{i_2}, \ldots, C_{i_{n_i}}; \Omega) \qquad (1.54)$$

In section I.7.1. it will be discussed how the computer program SETS [21] generates the set of Equations (1.54) by a direct manipulation of the Boolean logic equations describing a fault tree. A feature particular to SETS is that in addition to the AND and OR gates commonly found in fault trees, it can also handle NOT gates, EXCLUSIVE OR gates and SPECIAL gates which are previously defined by the user in terms of a specific set of Boolean equations.

In section I.7.2. the BAM [18] (Boolean Arithmetic Model) computer program will be discussed which evaluates the TOP event occurrence probability

$$P(TOP) = P(K_1 U K_2 U, \ldots, K_t) \qquad (1.55)$$

by expanding the Boolean expression corresponding to the top event in a series of mutually exclusive events. As will be shown, such an expansion is only made possible by simultaneously considering the set of basic events $(c_1, \ldots, c_n)$ as well as their corresponding complement events $(\bar{c}_1, \bar{c}_2, \ldots, \bar{c}_n)$ obtained by applying the complement (upper bar) operation to the original basic events and defined by

$$c U \bar{c} = S \qquad (1.56)$$

where S = the universal set.

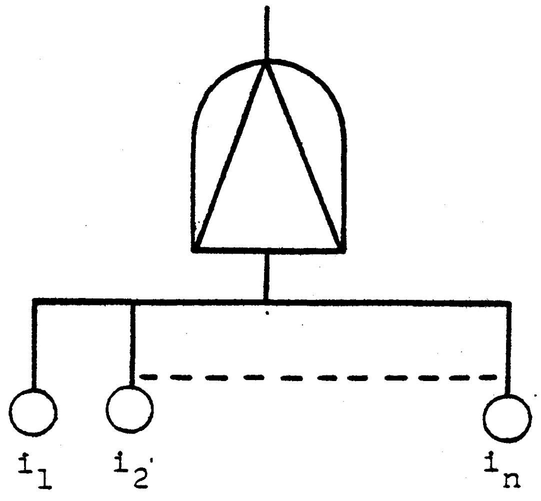
By including complement state events in its formalism, BAM succeeds to incorporate dependent as well as mutually exclusive events. As a result BAM is capable of computing the unavailabilities for systems undergoing test and maintenance procedures as well as for systems which are subject to common mode failures.

I.7.1. SETS

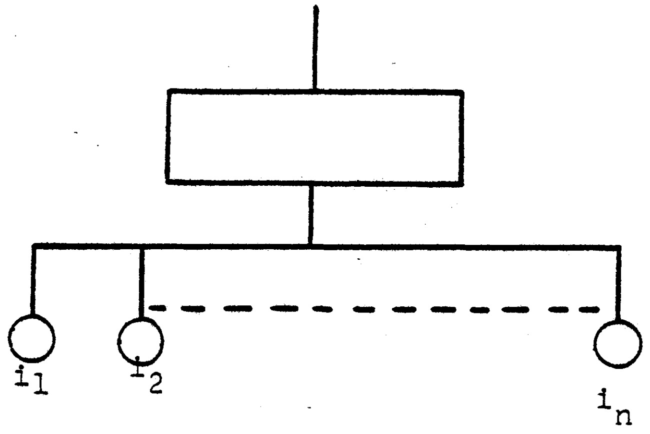
The Set Equation Transformation System [21] symbolically manipulates Boolean equations formed by a set of events operated on by a particular set of union, intersection and complement operators.

Given a fault tree, a Boolean equation is established to represent each intermediate event as a function of its input events. In addition to AND and OR gates, intermediate events may also be related by EXCLUSIVE OR gates and SPECIAL gates (Figure 1.7) to their inputs. For an EXCLUSIVE OR gate, its output event will occur only if exactly one of the input events occurs while the other inputs do not occur. Thus if the EXCLUSIVE OR gate operates on two events (c_1, c_2) then its output is given by

$$\text{EXCLUSIVE - OR } (c_1, c_2) = (c_1 \Omega \bar{c}_2) U (\bar{c}_1 \Omega c_2) \qquad (1.57)$$



EXCLUSIVE OR-GATE



SPECIAL GATE

FIGURE 1.7

EXCLUSIVE OR GATE AND SPECIAL GATES AVAILABLE IN SETS

Special gates are uniquely defined by a Boolean equation provided by the user. Thus, if for example a SPECIAL 2 - out of - 3 gate is wanted, then it must be defined by

$$\text{SPECIAL GATE}_1(c_1,c_2,c_3) = (c_1 \,\Omega\, c_2)\,U\,(c_1 \,\Omega\, c_3)\,U\,(c_2 \Omega c_3) \qquad (1.58)$$

The computer program SETS offers the user the option to develop the set of Boolean equations describing the fault tree in such a way as to directly derive the set of "prime implicants" [17] corresponding to any desired intermediate gate event.

Each prime implicant for an intermediate gate will correspond to one of its minimal cut-set events with the restriction that there be no simultaneous occurrence of a basic event (c) and its complement ($\bar{c}$) in the cut-set.

SETS derives the prime implicant description for an intermediate gate by using a set of substitutions and successively applying the distributive law

$$A\Omega(BUC) = (A\Omega B)U(A\Omega C) \qquad (1.59)$$

Suppose for example that SETS has been commanded to derive a representation for gate G2 of fault tree example I. The following procedure would take place

STEP 1 $\quad G2 = G4 \; U \; G6$

$G4 = C1 \,\Omega\, C2 \,\Omega\, G5, \quad G5 = C8 \; U \; C9$

$G6 = \overline{C3} \; U \; G7 \quad , \quad G7 = C7 \,\Omega\, C4$

STEP 2 $G6 = C3 \cup (C7 \Omega C4)$

$G4 = C1 \Omega C2 \Omega (C8 \cup C9)$

STEP 3 $G2 = (C1 \Omega C2 \; \Omega (C8 \cup C9) \cup (C3 \cup (C7 \Omega C4))$

STEP 4 Apply distributive law (equation 1.59)

$\Rightarrow G2 = (C1 \Omega (C2 \Omega C8) \cup (C2 \Omega C9) \cup$
$((C3) \cup (C7 \; \Omega C4)$

$\Rightarrow G2 = (C1 \Omega C2 \Omega C8) \cup (C1 \Omega C2 \Omega C9) \cup$
$(C3) \cup (C7 \Omega C4)$

Hence the prime implicants (minimal cut-sets) for G2 are

$K_1 = (C1, C2, C8)$

$K_2 = (C1, C2, C9)$

$K_3 = (C3)$

$K_4 = (C7, C4)$

The above procedure is generally used to derive the prime implicants for any fault tree, however the additional identities

$$C_i \; \Omega C_i = C_i, \; C_i \Omega \; \overline{C}_i = \phi \text{ (empty set)} \quad (1.60)$$

may sometimes be needed.

I.7.2 <u>BAM</u>

Computer program BAM [18] uses a Boolean algebra minimization technique to find intermediate and top event logic expressions from the input fault tree and calculates the point unavailabilities associated with these events.

By including basic events (on states) as well as their respective complements (OFF states) BAM is able to construct a truth table which describes each intermediate gate event in the fault tree as the union of mutually exclusive (ON and OFF) state events. Thus, for example consider an OR gate operating on components (C1, C2).

Its coherent structure description will be (Equation 1.4)

$$\phi_{OR} = 1 - (1 - y_1)(1 - y_2) \tag{1.61}$$

at the same time recall that (Equation 1.9)

$$\phi_{NOT}(y) = 1 - y \tag{1.62}$$

hence

$$\phi_{OR} = \phi_{NOT}(\phi_{NOT}(y_1) \cdot \phi_{NOT}(y_2)) \tag{1.63}$$

The above equation may now be reexpressed in set theoretical form by replacing AND, OR and NOT gates by union (U), intersection (Ω) and complement ($\bar{\ }$) operations, thus

$$C_1 \cup C_2 = \overline{(\bar{C}_1 \cap \bar{C}_2)} \tag{1.64}$$

Using now the identity

$$S = (C1 \cap C2) \cup (C1 \cap \bar{C}2) \cup (\bar{C}1 \cap C2) \cup (\bar{C}1 \cap \bar{C}2) \tag{1.65}$$

with $S = C \cup \bar{C} \equiv$ the universal set. It follows that

$$C_1 \cup C_2 = (C_1 \cap \bar{C}_2) \cup (\bar{C}_1 \cap \bar{C}_2) \cup (C_1 \cap C_2) \tag{1.66}$$

which is the desired expansion, since all events given in the right hand side of Equation (1.66) are mutually exclusive.

In Table 1.3 and 1.4 the truth tables [18] associated with the above logical expression (C_1 U C_2) as well as $C_1 U (C_2 \Omega \overline{C}_3)$ are given

I	II		II
p-terms	y_1	y_2	c_1 U c_2
$c_1 \Omega c_2$	1	1	1
$\overline{c}_1 \Omega c_2$	0	1	1
$c_1 \Omega \overline{c}_2$	1	0	1
$\overline{c}_1 \Omega \overline{c}_2$	0	0	0

Table 1.3 Canonical Expansion for C_1 U C_2

In general the truth table for an expression consisting of N distinct logical variables is expanded using 2^N P-terms. Columns I and II are equivalent representations for each P-term needed for a canonical expansion. Thus, Column II can be derived from Column I by assigning a 1 value to ON states and a 0 value to OFF states. The canonical expansion (Column III) for a particular logical expression is then obtained by performing for each row in the truth table a series of Boolean arithmetic operations equivalent to the set of operations indicated in the logical expression. Thus, C_1 U C_2 requires only that variables y_1 and y_2 be added at each row. While C_1 U $(C_2 \Omega \overline{C}_3)$ requires the set of operations

I	II				III
P-terms	y_1	y_2	y_3	$C_2 \Omega \overline{C}_3$	$C_1 U (C_2 \Omega \overline{C}_3)$
$C_1 \Omega C_2 \Omega C_3$	1	1	1	0	1
$C_1 \Omega C_2 \Omega C_3$	1	1	0	1	1
$C_1 \Omega C_2 \Omega C_3$	1	0	1	0	1
$C_1 \Omega C_2 \Omega C_3$	1	0	0	0	1
$C_1 \Omega C_2 \Omega C_3$	0	1	1	0	0
$C_1 \Omega C_2 \Omega C_3$	0	1	0	1	1
$C_1 \Omega C_2 \Omega C_3$	0	0	1	0	0
$C_1 \Omega C_2 \Omega C_3$	0	0	0	0	0

Table 1.4 Canonical Expansion for $C_1 U (C_2 \Omega \overline{C}_3)$

$$C_1 \text{ U } CC_2 \Omega\, \overline{C}_3) = y_1 + (y_2 \cdot \overline{y}_3) \qquad (1.67)$$

It should be recalled that the following identities apply for Boolean arithmetic variables

$$1 + 1 = 1 \qquad (1.68)$$
$$1 + 0 = 1$$
$$1 \cdot 0 = 0$$
$$1 \cdot 1 = 1$$
$$0 \cdot 0 = 1$$
$$\overline{0} = 1$$
$$\overline{1} = 0$$

Therefore the addition implied by C_1 U C_2 will result in

1st row $1 + 1 = 1$

2nd row $1 + 0 = 1$

3rd row $0 + 1 = 1$

4th row $0 + 0 = 0$

so as expected

$$C_1 \text{ U } C_2 = (C_1 \Omega\, C_2) \text{U} (C_1 \Omega\, \overline{C}_2) \text{ U } (\overline{C}_1 \Omega\, C_2)$$

$$\Rightarrow \quad P(C_1 \text{ U } C_2) = P(C_1\, \Omega\, C_2) + P(C_1 \Omega\, \overline{C}_2) + P(\overline{C}_1\, \Omega\, C_2)$$

$$\Rightarrow P(C_1 \text{ U } C_2) = p_1 p_2 + p_1(1 - p_2) + p_2(1 - p_1)$$

$$= P(C_1 \text{ U } C_2) = -p_1 p_2 + p_1 + p_2 \qquad (1.69)$$

Similarly for $C_1 U (C_2 \ \bar{C}_3)$ each row is applied the operation $y_1 + (y_2 \cdot \bar{y}_3)$.

Thus, it follows that

1st row	$1 + (1 \cdot \bar{1}\ ) = 1 + 0 = 1$
2nd row	$1 + (1 \cdot \bar{0}\ ) = 1 + 1 = 1$
etc.	

By inspection of Table 1.4 it is found that

$$P(C_1 \text{ U } CC_2 \,\Omega\, C_3)) = p_1p_2p_3 + p_1p_2(1 + p_3) +$$

$$+ p_1(1-p_2)\ p_3 + p_1(1 - p_2)\ (1 - p_2)(1 - p_3) + (1-p_1)$$

$$p_2(1 - p_3)$$

$$= P(C_1 \text{ U } CC_2 \,\Omega\, C_3)) = p_1 + p_2 - p_1p_2 - p_2p_3 + p_1p_2p_3 \tag{1.70}$$

The following examples illustrate how the BAM code is capable of handling fault trees which include mutually exclusive events and dependent failures.

Figure 1.8 depicts the fault tree for a system C, made up of two sub-systems A and B each of which may not be functioning due to either a hardward failure or because it is undergoing maintenance events MA and MB, should be mutually exclusive, hence the appearance of complement events $\overline{MA}$ and $\overline{MB}$ in the

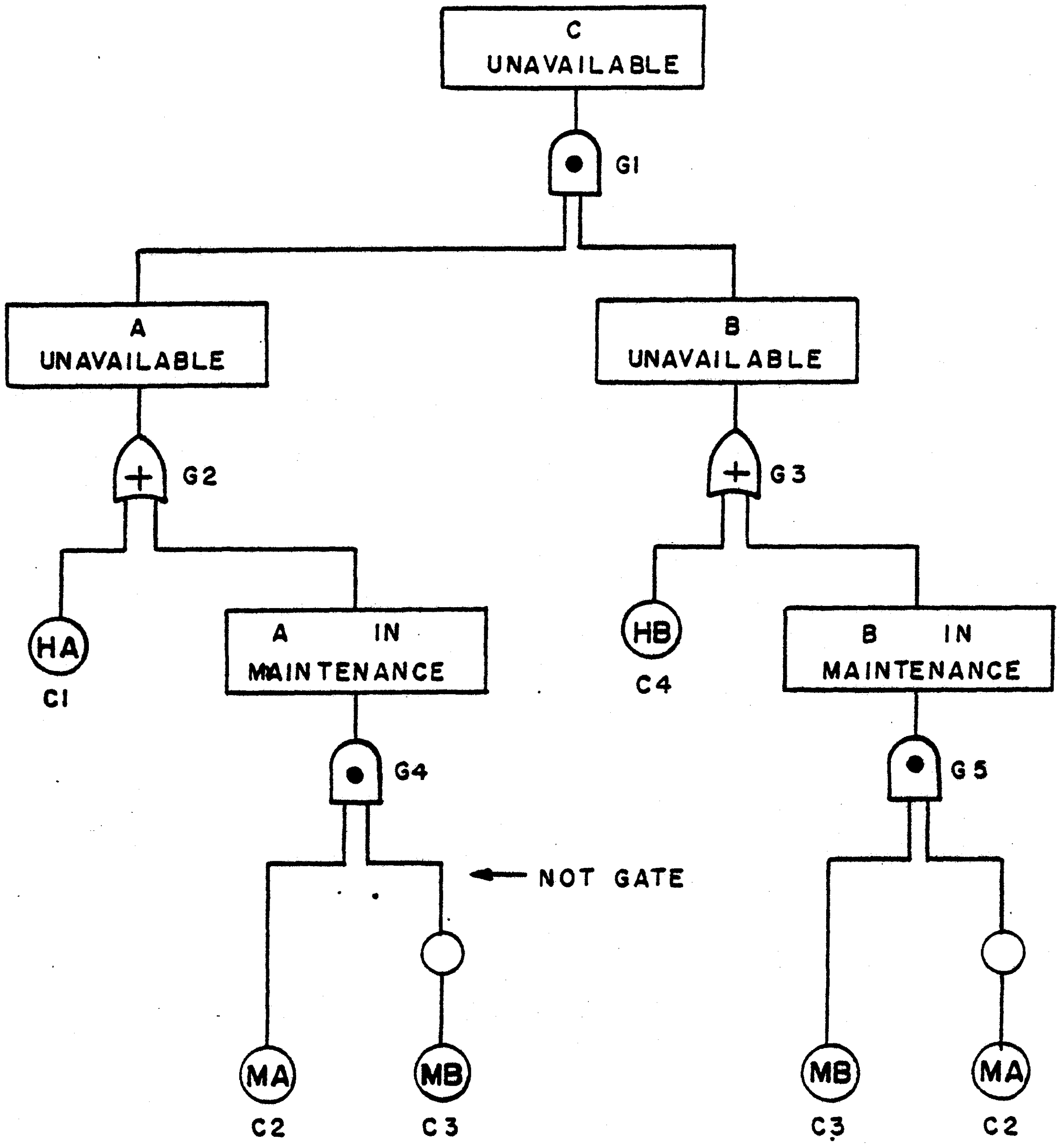


FIGURE 1.8 Fault Tree Including Mutually Exclusive Maintenance Events

TABLE 1.5

CANONICAL EXPRESSION FOR FAULT TREE WITH MAINTENANCE EVENTS

C_1 Y_1	C_2 Y_2	C_3 Y_3	C_4 Y_4	$G2=C_1 U(C_2 \Omega \overline{C}_3)$ $Z_1=Y_1+(Y_2 \cdot \overline{Y}_3)$	$G3=C_4 U(C_3 \Omega \overline{C}_2)$ $Z_2=Y_4+(Y_3 \cdot \overline{Y}_2)$	$G1=G_2 \Omega\ G_3$ $Z=Z_1 \cdot Z_2$
1	1	1	1	1	1	1
1	1	0	1	1	1	1
1	0	1	1	1	1	1
1	0	0	1	1	1	1
0	1	1	1	0	1	0
0	1	0	1	1	1	1
0	0	1	1	0	1	0
0	0	0	1	0	1	0
1	1	1	0	1	0	0
1	1	0	0	1	0	0
1	0	1	0	1	1	1
1	0	0	0	1	0	0
0	1	1	0	0	0	0
0	1	0	0	1	0	0
0	0	1	0	0	1	0
0	0	0	0	0	0	0

fault tree. Table 1.5 provides the truth table for the fault tree. Notice that even though

$$P(TOP) \neq P(G2) \cdot P(G3) \qquad (1.71)$$

since gates G2 and G3 are interdependent, it is however feasible to compute

$$P(TOP) = p_1p_4 + p_1p_3 + p_4p_2 \qquad (1.72)$$

using the cannonical expansion for G1 corresponding to

$$z = z_1 \cdot z_2 \ .$$

In figure 1.9 an event B <u>dependent</u> on the occurrence of event A is represented in terms of a tree logic diagram which includes the events

B/A = Event B given the occurrence of A

$B/\overline{A}$ ≡ Event B given the occurrence of $\overline{A}$

as

$$B = (A \; \Omega B/A) \; U \; (\overline{A} \; \Omega B/\overline{A}) \qquad (1.73)$$

This representation is quite convenient for performing quantitative evaluations of a fault tree which includes event B since

$$P(A \Omega B/A) = P(A) \cdot P(B/A)$$

and $$P(\overline{A} \; \Omega B/A) = P(\overline{A}) \cdot P(B/\overline{A}) \qquad (1.74)$$

The above representation for an event dependent on the occurrence of a single event has been generalized in BAM for the case of events dependent on a multiple number of basic events

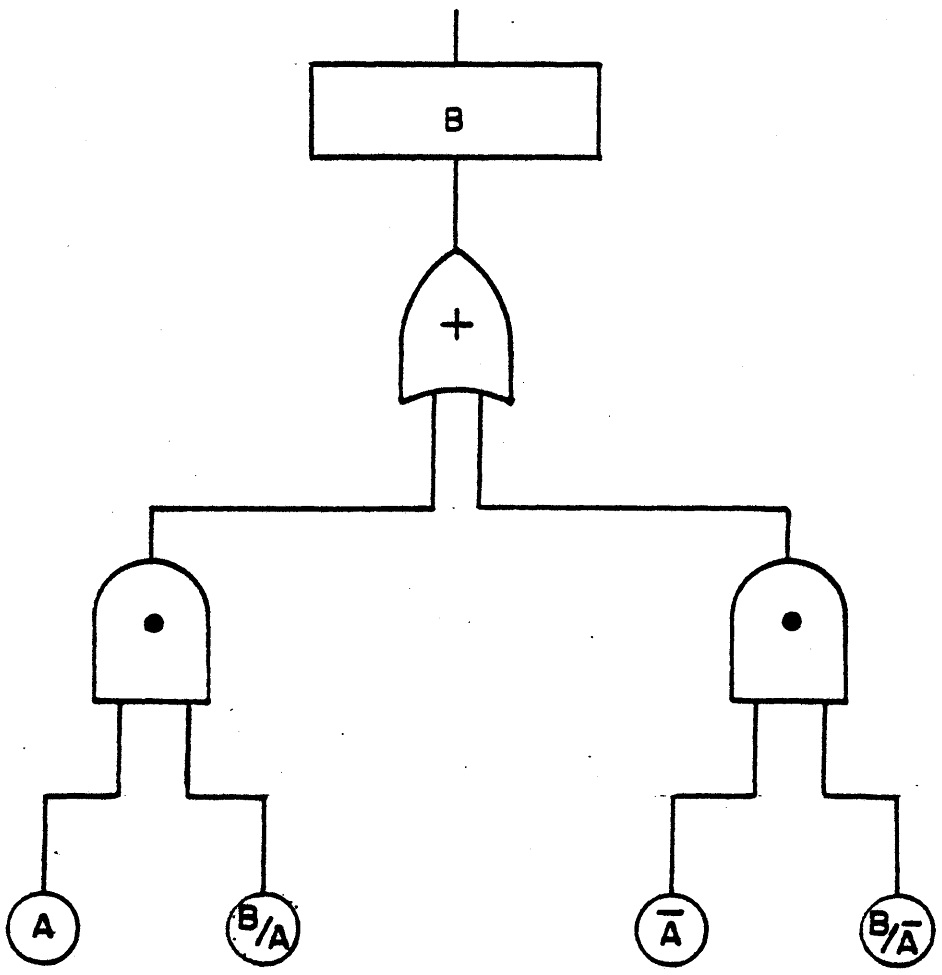


FIGURE 1.9 Representation of an Event B Dependent on the Occurrence of Event A

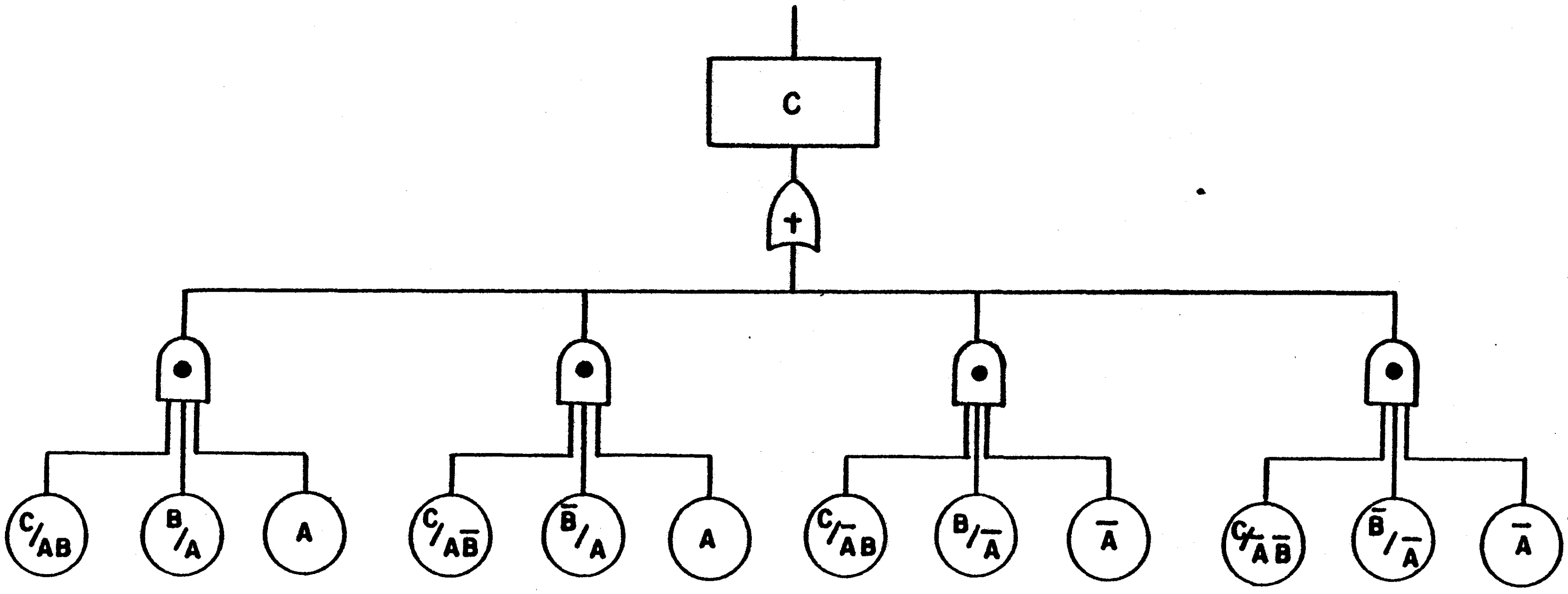


FIGURE 1.10

REPRESENTATION OF AN EVENT C DEPENDENT ON THE OCCURRENCE OF EVENTS B AND A

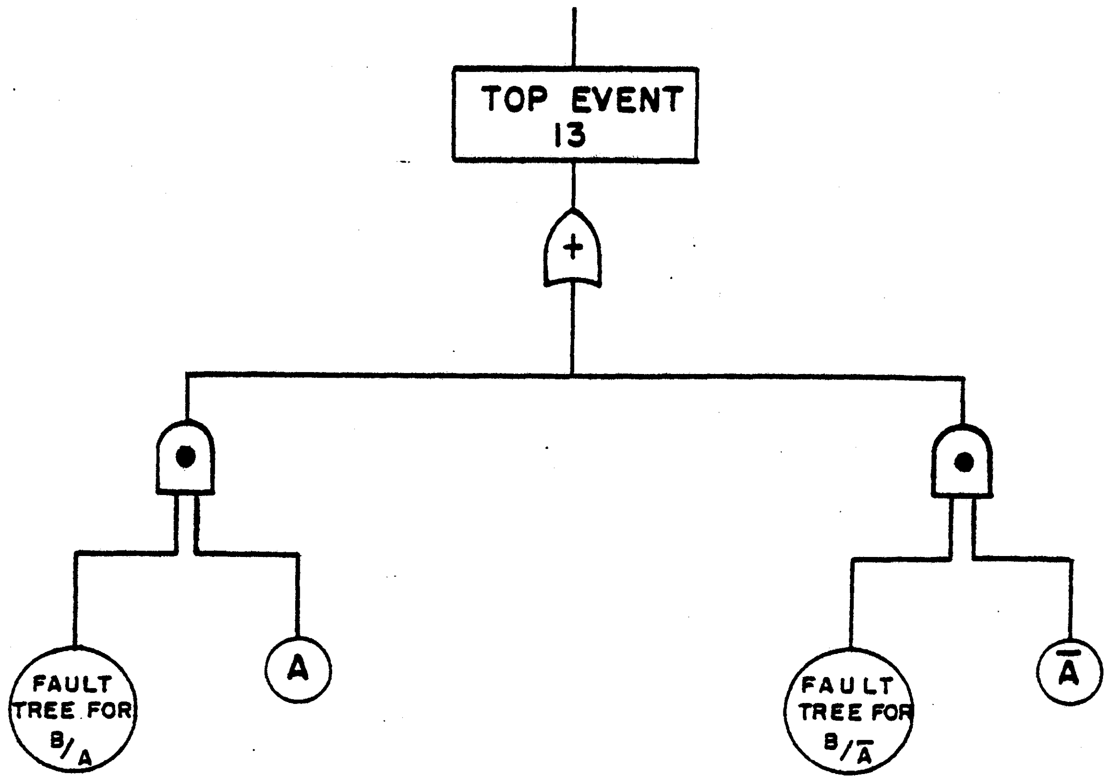


FIGURE 1.11

FAULT TREE INCLUDING COMMON MODE EVENT A

(Figure 1.10) as well as to common mode failures depicted as a multiple set of events whose occurrence probability is dependent on a common initiating event (Figure 1.11).

I.8. Reliability Calculations by a Pattern Recognition Method

The computer program PATREC [12] relies on the recognition of sub-tree patterns whose probability combination laws have been previously stored in the computer code's library. The sub-tree is then replaced by a supercomponent with an associated occurrence probability equal to that of the recognized sub-tree. By repeating this process the whole tree is eventually transformed into a single super-component whose occurrence probability corresponds to that of the top tree event.

The elementary pattern recogniztion methodology used by PATREC entails that large amounts of non-numerical data interrelated on a complicated way be handled. To this end the computer language PL-1 was chosen given its list processing capabilities.

The task of evaluating the TOP tree event occurrence probability is performed by PATREC through the following set of manipulations on the fault tree structure which is subject to the following restrictions:

(a) Pattern recognition is made possible by giving the fault tree diagram in a binary gate form (Fig. 1.12).

(b) Because of the binary gate form of the fault tree, to each gate there corresponds a left hand side and a right hand side sub-tree.

(c) Before proceeding on to identify sub-tree patterns at each step in the tree reduction, PATREC internally reorders the fault tree diagram in a way such that if to every AND gate one unit of weight is assigned and to every OR gate two units of weight are assigned, then for each gate its right hand sub-tree will be heavier than its left hand side. Figure 1.13 shows the fault tree example I1 reordered according to the above rule. The above tree reordering is done in order to avoid the storage of different patterns which correspond to the same logic structure (Figure 1.14)

(d) Using list processing methods the pattern library is stored in the computer memory in a tree-like form. As a result redundant information about similar sub-patterns isn't stored separately and moreover the largest pattern found in PATREC's library are guaranteed to be identified each time. In Figure 1.15 the tree representing the set of 12 basic patterns stored in PATREC is shown. Tree patterns are represented in reverse polish notation, thus

$$P_1 = A\ B\ \Omega = A\ \Omega B \qquad (1.75)$$

$\vdots$

$$P_5 = A\ B\ C\ \Omega U = A\ U\ (B\ \Omega\ C)$$

$\vdots$

$$P_8 = A\ B\ \Omega C\ D\ U = (A\ \Omega B)\ \Omega\ (C\ U\ D)$$

$\vdots$

etc.

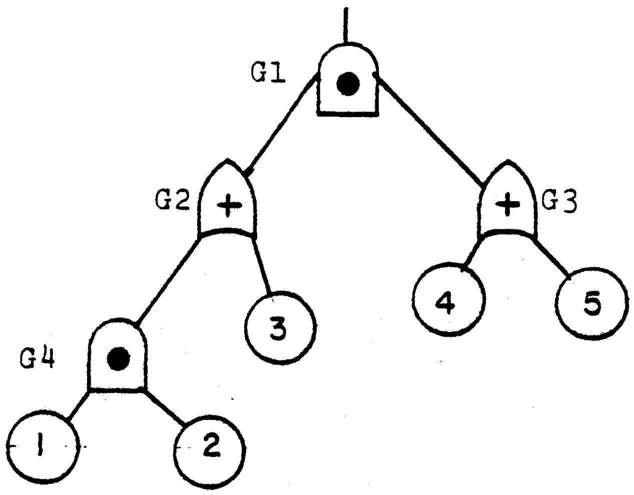


FIGURE 1.12 FAULT TREE EXAMPLE II IN BINARY GATE FORM

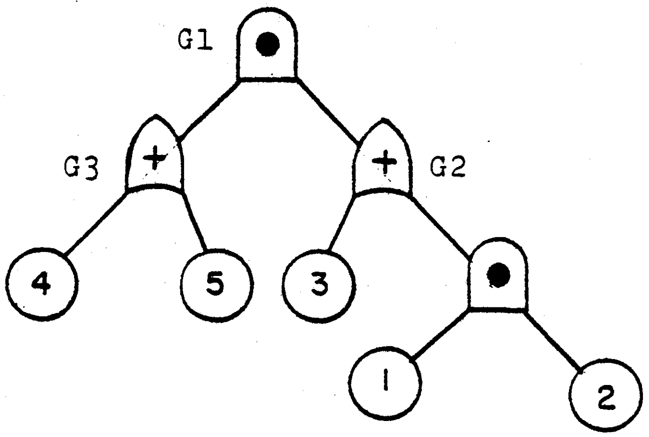
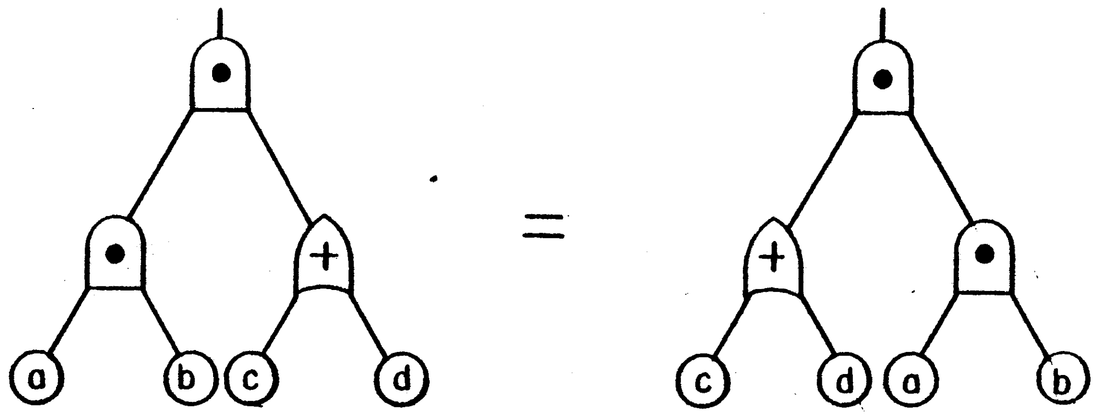


FIGURE 1.13 FAULT TREE EXAMPLE II IN ITS ORDERED FORM



Pattern found in PAT-REC's library

Pattern not found in PAT-REC's library

FIGURE 1.14

EQUIVALENT BINARY TREE PATTERNS

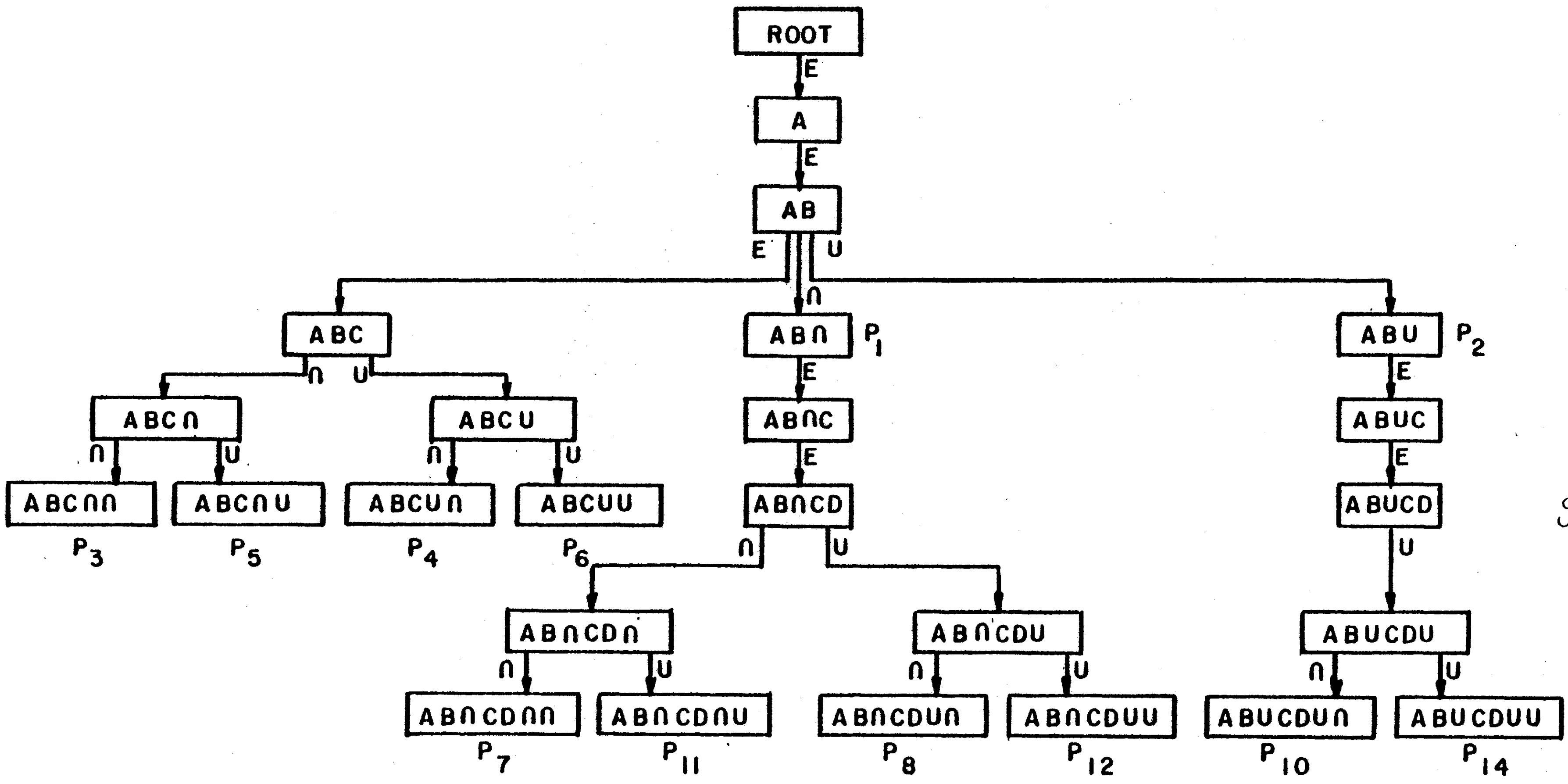


FIGURE 1.15

PAT-REC'S LIBRARY OF PATTERNS STORED IN A TREE-LIKE FORM

(e) Basic components are required not to be replicated in the fault tree. Consequently, each time a sub-tree is found to correspond to a particular pattern in PATREC's library, it will be possible to replace it by a supercomponent having the same occurrence probability as that of the sub-tree's top event. Thus, since gate G2 of fault tree example II is the top gate for a sub-tree with the same structure as that of pattern P_5, it will be replaced by a supercomponent having an occurrence probability

$$P_{G2} = P_3 + (P_2 \cdot P_1) - (P_3P_2P_1) \qquad (1.76)$$

Subsequently a new ordered representation for the fault tree will be found (Figure 1.16), which corresponds to pattern P_4 = ABC U , hence the TOP event occurrence probability is finally determined as

$$P(TOP) = P_{G2}\ (P_4 + P_5 - P_4P_5) \qquad (1.77)$$

As explained above the procedure used by PATREC is restricted to fault trees which does not include replicated events. For most real problems however a number of basic components will be replicated several times in the fault tree. Therefore it is necessary that the methodology be somehow generalized to handle these situations. Computer code PATREC-DE [4] was created for this purpose. Its procedure is based on expressing the structure of a fault tree which includes replicated events in terms of a number of fault trees having no replications in their structure. Thus, recall that the dependency of a coherent structure

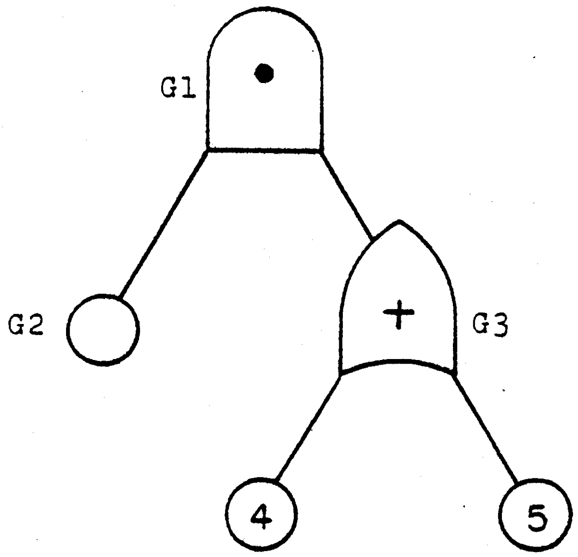


FIGURE 1.16

FINAL ORDERED FORM FOR FAULT TREE EXAMPLE II function $\phi(Y^N)$ on any of its basic inputs y_i may be explicitly indicated as

$$\phi(\underset{\sim}{Y}) = Y_i \; \phi(1_i,\underset{\sim}{Y}) + (1 - Y_i) \; \phi(0_i,\underset{\sim}{Y}) \quad (1.38)$$

$$= \; h(\underset{\sim}{P}) = P_i \; h(1_i,\underset{\sim}{P}) + (1 - P_i) \; h(0_i,\underset{\sim}{P}) \quad (1.39)$$

This expansion has the effect of wiping out the dependency on Y_i from the fault trees representing $\phi(1_i,\underset{\sim}{Y})$ and $\phi(0_i,\underset{\sim}{Y})$ (Figure 1.17). Therefore by repeatedly expanding in all variables Y_i ($i = 1,2,...,$) which correspond to replicated basic events, it is possible to relate the original fault tree to a number of fault trees which include no replicated events in their structure, i.e.,

$$\phi(\underset{\sim}{Y}^N) = \sum_{\underset{\sim}{Y}^R} \prod_{y=1}^{r} x_j^{Y_j} (1-x_j)^{1-y_j} \; \phi(\underset{\sim}{Y}^R,\underset{\sim}{Y}^{R^C}) \quad (.78)$$

where the sum is extended over all of the 2^r binary vectors $\underset{\sim}{Y}^R$ corresponding to a particular combination of ON and OFF states for the replicated events, $RUR^C = N$ and $0^o \equiv 1$.

The TOP event occurrence probability for the original fault tree will then be given by

$$P(TOP) = h(P) = \sum_{\underset{\sim}{Y}^R} \prod_{j=1}^{r} x_j^{Y_j} \; (1-X_j)^{1-Y_j} (\underset{\sim}{Y}^R,\underset{\sim}{P}^{R^C})$$

(1.79)

Notice, however that this procedure has the disadvantage of

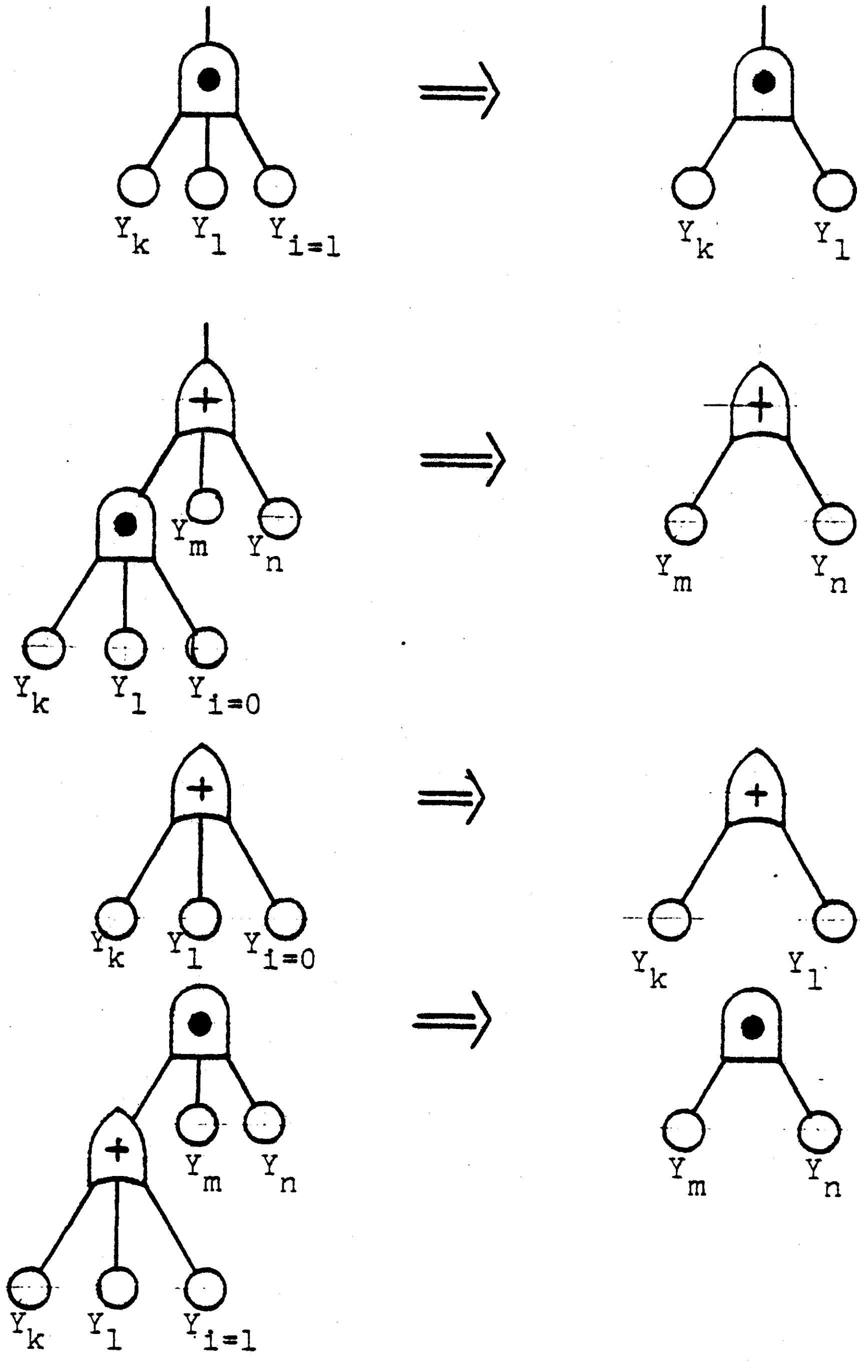


FIGURE 1.17

FAULT TREE DEPENDENCIES REDUCED OUT WHEN $Y_i = 0$ OR $Y_i = 1$

requiring that 2^r different fault tree TOP event occurrence probabilities be evaluated.

I.9. The IMPORTANCE Computer Program

IMPORTANCE [14] is a computer program which was developed to rank basic events and cut-sets according to various importance measures.

The IMPORTANCE computer code is capable of handling time-dependent fault trees under the assumption that each basic component be statistically independent and that its failure and repair distribution be exponential in time. Thus to each basic event there correspond a set of parameters $(\nu,\lambda)_i$ such that the failure occurrence probability $P_i(t)$ obeys the equations

$$q(t) = 1 - p(t) \tag{1.80}$$

$$\frac{dq(t)}{dt} + \lambda q(t) = \nu p(t)$$

$$\frac{dp(t)}{dt} + \nu p(t) = \lambda q(t)$$

$$q(0) = 1$$

Therefore p(t) will be given by

$$p(t) = \frac{\lambda}{\lambda+\nu} \left(1 - e^{-(\lambda+\nu)t} \right) \tag{1.81}$$

and $$U = \lim_{t\to\infty} P(t) = \frac{x}{\nu+\lambda} = \frac{\frac{1}{\mu}}{\frac{1}{\mu} + \frac{1}{\tau}} = \frac{\tau}{\tau+\mu}$$

TABLE 1.6

BASIC EVENT IMPORTANCE MEASURES COMPUTED BY THE IMPORTANCE CODE

Measure	Expression
1. Birnbaum	$\frac{\partial h(\underset{\rightarrow}{P}(t)}{\partial P_i(t)} = h(1_i,\underset{\rightarrow}{P}(t)) - h(0_i,\underset{\rightarrow}{P}(t))$
2. Criticality	$\frac{\partial h(\underset{\rightarrow}{P}(t))}{\partial P_i(t)} \cdot \frac{P_i(t)}{h(\underset{\rightarrow}{P}(t))}$
3. Upgrading Function	$\frac{\lambda_i}{h(\underset{\rightarrow}{P}(t))} \frac{\partial h(\underset{\rightarrow}{P}(t))}{\partial_{\lambda_i}}$
4. Vesely-Fussell	$\frac{h_i(\underset{\rightarrow}{P}(t))}{h(\underset{\rightarrow}{P}(t))}$
5. Barlow-Proschan	$\frac{\int_o^t [h(1_i,\underset{\rightarrow}{P}(t^1)) - h(0_i,\underset{\rightarrow}{P}(t^1)]dW_f,i(t^1)}{E[N_s(t)]}$

6. Steady State Barlow-Proschan (BP,SS)

$$\frac{[h(1_i,\underset{\rightarrow}{U}) - h(o_i,\underset{\rightarrow}{U})]/\mu_i + \tau_i}{\sum_{j=1}^{n} [h(1_j,\underset{\rightarrow}{U}) - h(0_j,\underset{\rightarrow}{U})]/\mu_i + \tau_j}$$

TABLE 1.6 (Continued)

7. Sequential Contributory

$$\sum_{\substack{j \\ i \neq j \\ i \& j \, \varepsilon K_\ell \\ \text{for some } \ell}} \int_0^t \frac{[h(1_i, 1_j, \underset{\sim}{P}(t^1)) - h(1_i, 0_j, \underset{\sim}{P}(t^1)]P_i(t^1)dW_{f,j}(t^1)}{E[N_j(t)]}$$

where $\mu \equiv$ component mean time to failure and $\tau \equiv$ component mean time to repair (for convenience the component index i has been omitted in the above equations).

Table 1.6 lists the seven measures of basic event importance computed by the IMPORTANCE code.

The first four basic event importance measures relate to the fault tree at a certain point in time t. The first, second and fourth measures were previously discussed in section 1.5. The Upgrading Function Importance measure proposed by Lambert [14] offers the advantage that λ_i as opposed to failure. probability $P_i(t)$ is a physically measurable parameter. Moreover Lambert has shown how the Upgrading Function may be used as a tool to decide on an optimal choice for system upgrade.

The fifth and seventh basic event importance measures are different in that they take into account the way components failed sequentially in time to cause system failure. Thus, the Barlow-Proschan importance [2] for component i measures the probability the system has failed by time t because a minimal cut-set critical to the system has failed with component i failing last.

The Barlow-Proschan measure is obtained by integrating over the component failure density W_f,i(t) and by dividing over the expected number of system failures $E[N_s(t)]$ by time t. W_f,i(t)dt is defined as the probability that event i will fail in the time interval (t,t+dt). Furthermore W_f,s(t) df is defined to be the probability that an overall system failure will occur in the interval (t,t +dt). Murchland [15] has shown that the system failure density $W_f,s(t)$ may be given in terms of $W_f,i(t)$ as

$$W_f,s(t) = \sum_{i=1}^{n} \frac{\partial h(\underset{\sim}{P}(t))}{\partial P_i(t)} W_f,i(t) \qquad (1.82)$$

From a knowledge of $W_f,s(t)$ the expected number of failures over the time interval [0,t] will be given by

$$E[N_s(t)] = \int_0^t W_f,s(t)dt \qquad (1.83)$$

The sequential contributory importance measure is useful to assess the role of the failure of a component i when any other component j is the cause of system failure. For this case the failure of i will contribute to system failure only if i and j are contained in at least one minimal cut-set associated with the fault tree.

Finally the Barlow-Proschan steady-state importance measure is concerned with the asymptotic behavior of each component in the fault tree. Asymptotically the probability that a component is down is given by its unavailability (Equation 1.81)

$$u_i = \frac{\tau_i}{\mu_i+\tau_i} \qquad (1.81)$$

hence the asymptotic value of its probability density $W_f,i^{(t)}$ will be

$$\lim_{t\to\infty} W_f,i(t) = \frac{\frac{\tau_i}{\mu_i+\tau_i}}{\tau_i} = \frac{1}{\mu_i+\tau_i} \qquad (1.84)$$

On the other hand the probability that component i causes system failure in the interval (t, t+dt) is given by

$$\frac{[h(1_i, \underset{\sim}{P}(t)) - h(0_i, \underset{\sim}{P}(t)] W_{f,i}(t)\, dt}{\sum_{j=1}^{n} [h(1_j, \underset{\sim}{P}(t)) - h(0_j, \underset{\sim}{P}(t))] W_{f,j}(t) dt} \tag{1.85}$$

therefore, the steady state probability that component i causes failure is

$$I_i^{BP,SS} = \frac{[h(1_i, \underset{\sim}{U}) - h(0_i, \underset{\sim}{U})] \frac{1}{\mu_i + \tau_i}}{\sum_{j=1}^{n} [h(1_j, \underset{\sim}{U}) - h(0_j, \underset{\sim}{U})] \frac{1}{\mu_j + \tau_j}} \tag{1.86}$$

CHAPTER TWO

MODULAR REPRESENTATION OF FAULT TREES

II.1. Introduction

Defined in terms of a reliability network diagram, a module is a group of components which behaves as a super-component. That means, it is completely sufficient to know the state of the super-component, and not the state of each component in the module, to determine the overall state of the system. In what follows, the properties associated with modularized fault trees and the computational advantages of analyzing fault trees by means of a modular decomposition will be presented.

II.2. Modular Decomposition of Coherent Systems

In the context of the theory of coherent structures, a module is formally defined as follows [1]:

Let $\Theta(\underset{\sim}{Y}^N)$ be the coherent structure function for a system having the vector $\underset{\sim}{Y}^N = (Y_1, Y_2, \ldots, Y_n)$ of basic input events. Then the subset M of basic events contained in N together with the coherent structure function $\sigma(\underset{\sim}{Y}^M)$ define a module provided

$$\Theta(\underset{\sim}{Y}^N) = \alpha(\, \sigma(\underset{\sim}{Y}^M),\ Y^{MC}) \qquad (2.1)$$

where α is a coherent structure function operation on the super-component state $\sigma(\underset{\sim}{Y}^M)$ and on the set of events $\underset{\sim}{Y}^{MC}$ with $N = M \cup M^C$.

Thus, a module $\sigma(\underset{\rightarrow}{Y}^M)$ for system $\Theta(\underset{\rightarrow}{Y}^N)$ is a coherent subsystem acting as a super-component. It follows then that in terms of a fault tree diagram, an intermediate gate event will be a module to the top event if the basic events contained in the domain of this gate do not appear elsewhere in the fault tree.

Hence the modularization of fault trees having no replicated events or gates can be easily accomplished, since every intermediate gate for such a fault tree will be the top event for a tree sub-module. Nevertheless, as soon as replicated events and gates occur in the fault tree, the modular decomposition becomes a more involved procedure.

II.3. <u>The Finest Modular Representation</u>

An algorithm to decompose a fault tree into its finest modular representation given its minimal cut-set structure composition, was originated by Chatterjee [7].

The finest modular representation for a coherent structure function $\Theta(\underset{\rightarrow}{Y}^N)$ is defined to be its mathematically equivalent fault tree diagram having the following properties:

1. All tree branches are independent, i.e., every intermediate gate event in the tree is modularizable;
2. The logic function associated with each gate is either "prime", or "simple" having no inputs from other "simple" gates of the same type.

AND and OR gates are defined as the "simple" gates, since they are characterized by a single cut-set and a single path-set, respectively. The second property requires that AND and OR gates present in the finest modular representation be of maximal size, i.e., if a simple gate has as inputs a number of simple gates of the same type, then all these gates must be collapsed together into one gate.

Higher order "prime" gates are defined to be Boolean logic functions which are not further modularizable. Prime logic functions are thus characterized by an irreducible set of Boolean cut-set vector equations.

Let $\sigma(\underset{\sim}{Y}^M)$ be the coherent structure function corresponding to a prime gate having inputs $\underset{\sim}{Y}^M = (Y_1, Y_2, \ldots Y_m)$, then each of its minimal cut-sets will be represented by a Boolean vector

$$S_j = (S_{1j}, S_{2j}, \ldots S_{nj}) \qquad (2.2)$$

($j = 1, \ldots, \ell$), with $S_{ij} = 1$ if the input i is contained in the cut-set j and $S_{ij} = 0$ if the input i is not contained in the cut-set j ($i = 1,2,\ldots,n$).

Thus, consider the sub-tree examples shown in Figures 2.1 and 2.3. Figure 2.1 represents a sub-tree having no replicated events, and its finest modular representation (Figure 2.2) is readily obtained by coalescing gates G1 and G2. Its modular structure is given by the following set of recursive equations.

$$M_1 = \{M_3, M_4, M_5; \Omega\} \qquad (2.3)$$

$$M_3 = \{a, b, c, ; \cup\} \qquad (2.4)$$

$$M_4 = \{d, e, f; \cup\}$$

$$M_5 = \{g, h, i; \cup\}$$

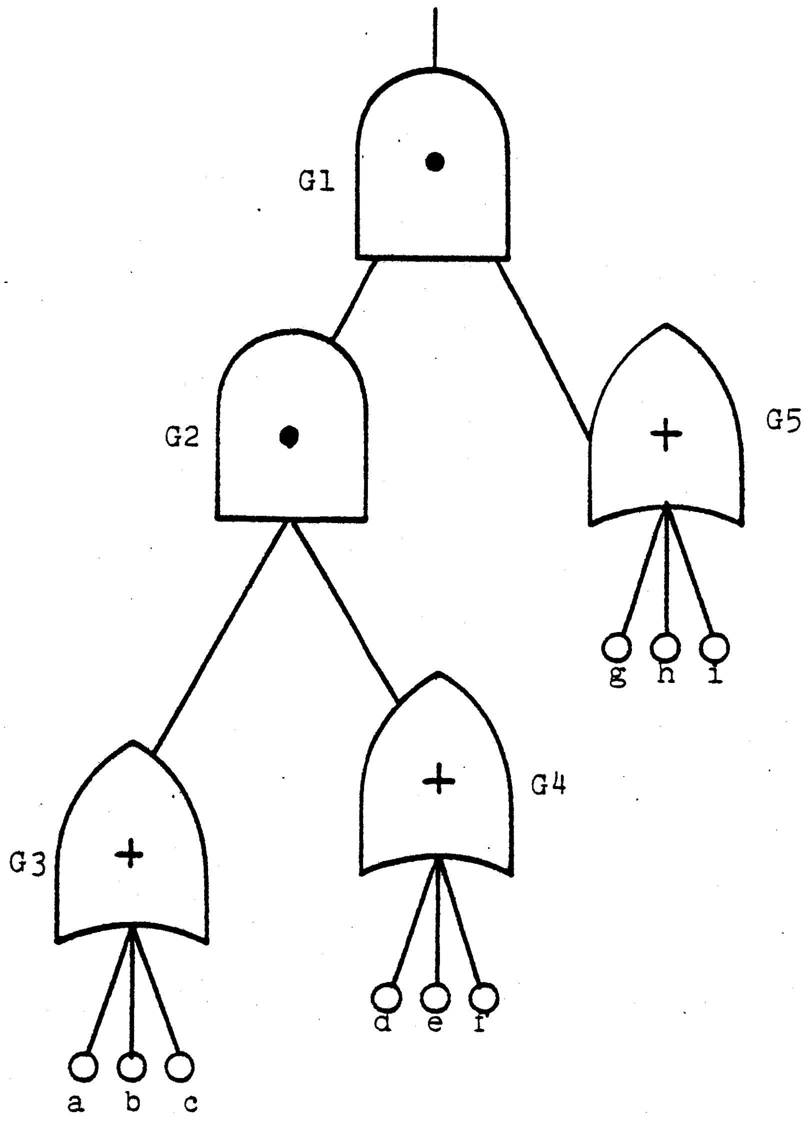


FIGURE 2.1

SAMPLE SUB-TREE I WITH NO REPLICATIONS

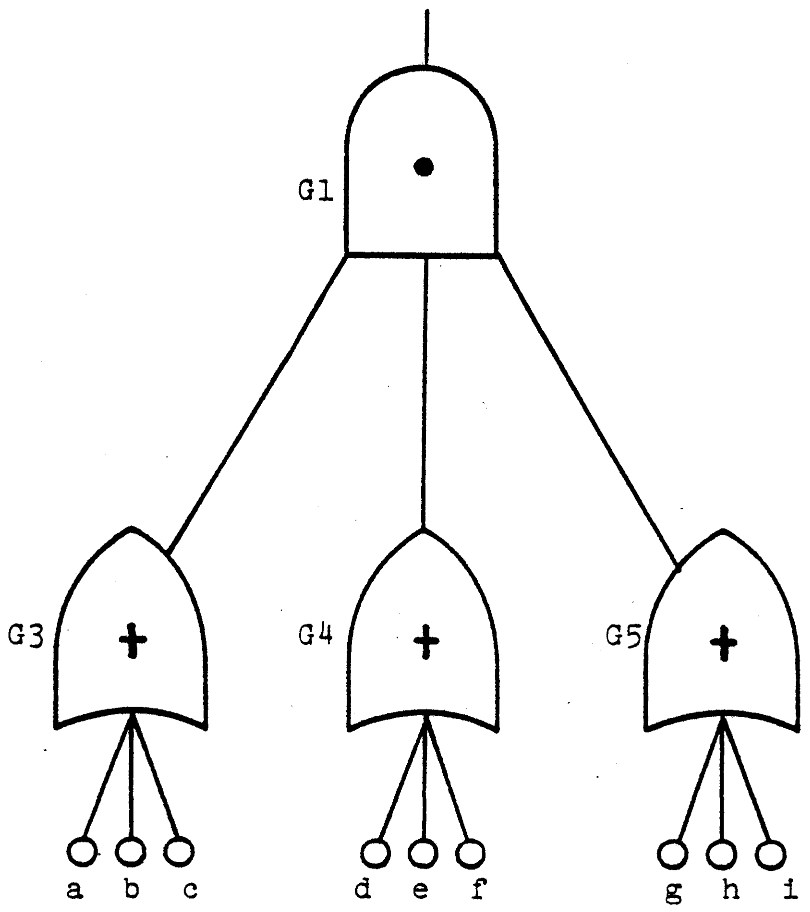


FIGURE 2.2

FINEST MODULAR REPRESENTATION OF SAMPLE SUB-TREE I

Alternately, the sub-tree structure could have been described by listing its 27 different minimal cut-sets (a,d,g), (b,d,g), (c,d,g), etc.

Figure 2.3 represents a sub-tree having replicated event r as an input to gates G3 and G5. To obtain its finest modular representation (Figure 2.4) one must first realize that events (a,b), (g,i) and (d,e,f) form modules associated with simple OR gates

$$M_3 = \{a,b;U\} \qquad (2.5)$$
$$M_4 = \{d,e,f;U\}$$
$$M_5 = \{g,i;U\}$$

Furthermore, these modules together with replicated event r will become the inputs to a higher order prime gate $\sigma(Y_r, Y_{M3}, Y_{M4}, Y_{M5})$ characterized by a set of MODULAR minimal cut-sets represented in Boolean vector form as:

$$\underset{\sim}{Y}^B = (Y_R, Y_{M3}, Y_{M4}, Y_{M5}) \qquad (2.6)$$

$$S_1 = (1,0,1,0) \qquad (2.7)$$
$$S_2 = (0,1,1,1)$$

It should be noted here how each of these modular minimal cut-sets is a compact representation for the usual basic event minimal cut-sets. Thus S_1 includes the 3 minimal cut-sets (r,d), (r,e), (r,f); while S_2 incorporates the other 12 remaining minimal cut-sets (a,d,g), (b,d,g), (a,d,i), etc. It must be stressed here that the algorithm given by Chatterjee was devised for deriving the modular composition of a fault tree given the minimal cut-set structural description

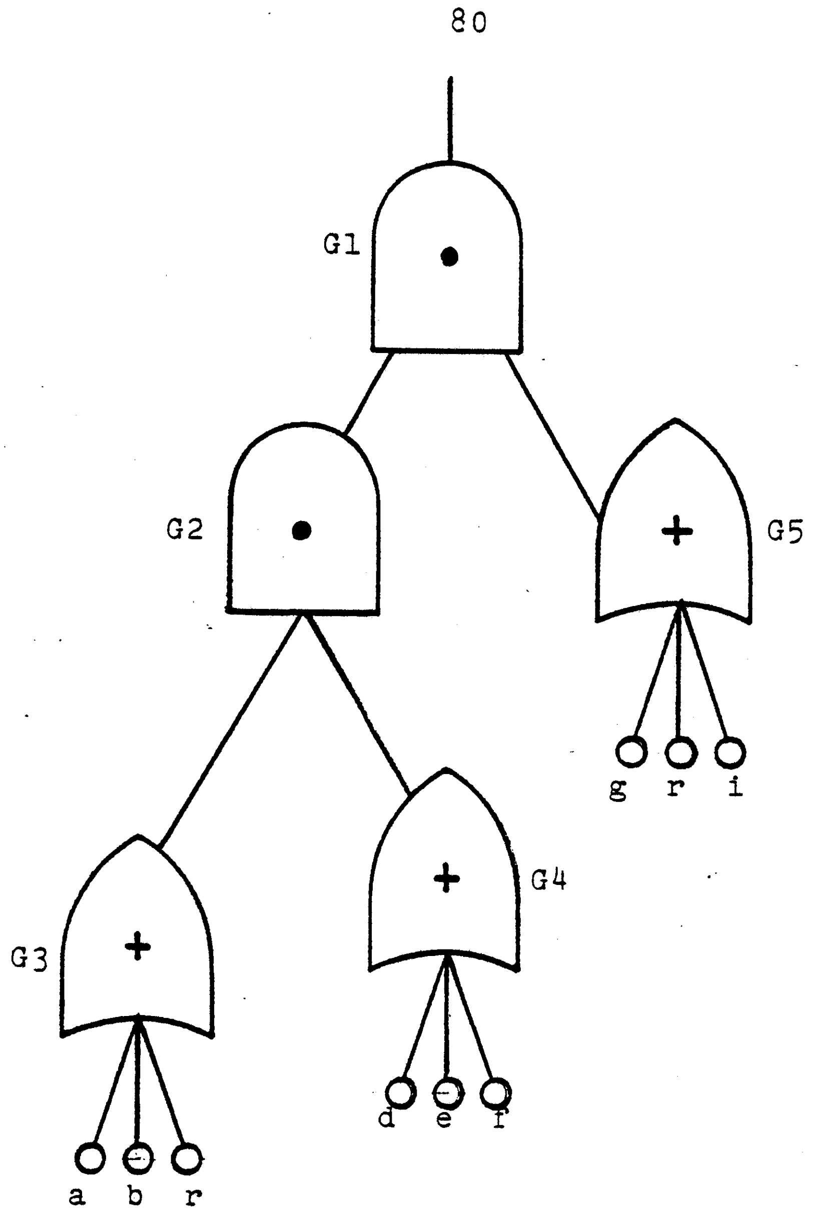


FIGURE 2.3

SAMPLE SUB-TREE II WITH REPLICATIONS

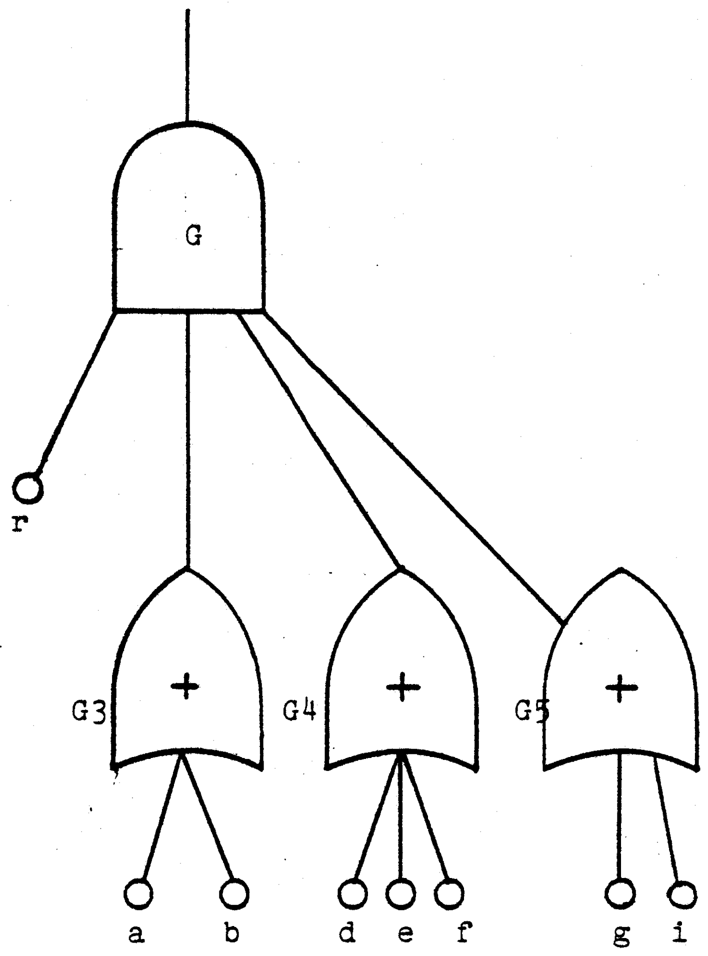


FIGURE 2.4

FINEST MODULAR REPRESENTATION OF SAMPLE SUB-TREE II of the fault tree. In complete contrast with this, the modularization algorithm given in Chapter III derives the modular composition of a fault tree directly from its diagram description.

II.4. Reliability Evaluation of Modularized Fault Trees

Once the modular structure of a fault tree has been derived, a quantitative evaluation of reliability and importance parameters of the fault tree may be efficiently performed. In particular, the probability of the occurrence of the top event, P(TOP), is obtained by means of a series of recursive calculations requiring the evaluation of the probability expectation value of each of the modules contained in the tree.

Thus, if a particular module M in the tree has a set $(M_1, M_2, \ldots, M_n)$ of modules as inputs, and is characterized by the coherent structure function σ_M

$$\sigma_M = \beta(\sigma_1, \sigma_2, \ldots, \sigma_n) \tag{2.8}$$

with $\sigma_i = \sigma_{M_i}$ $(i=1,\ldots,n)$, then its expectation value $h_\sigma(\underset{\rightarrow}{P})$ is given by

$$h_\sigma(\underset{\rightarrow}{P}) = h_\beta(h_{\sigma_1}(\underset{\rightarrow}{P}), h_{\sigma_2}(\underset{\rightarrow}{P}), \ldots, h_{\sigma_n}(\underset{\rightarrow}{P})) \tag{2.9}$$

For the case of simple AND and OR gate modules, the expression for h_β reduces to

$$M = \{M_1, M_2, \ldots, M_n; \Omega\}$$
$$\Rightarrow h_\beta = h_{\sigma_1} \cdot h_{\sigma_2} \ldots . h_{\sigma_n} = \prod_{i=1}^{n} h_{\sigma_i} \tag{2.10}$$

$$M = \{M_1, M_2, \ldots, M_n; U\}$$

$$\Rightarrow h_\beta = 1-(1-h_{\sigma_1})(1-h_{\sigma_2})\ldots(1-h_{\sigma_n}) = \coprod_{i=1}^{n} h_{\sigma_i} \qquad (2.11)$$

While for a higher order gate module $h_\beta(\underset{\rightarrow}{P})$ is given by

$$\sigma_M = \coprod_{j=1}^{N_k} \prod_{i \varepsilon K_j} \sigma_i$$

$$\Rightarrow h_\beta = \Sigma \left(\coprod_{j=1}^{N_k} \prod_{i \varepsilon K_j} \sigma_i \right) \qquad (2.12)$$

where $i\varepsilon K_j$ includes all modules contained in the minimal cut-set K_j, N_k is the total number of minimal cut-sets representing the module structure σ_M and E represents the probability expectation value operator which when applied on the structure function σ_i yields

$$E\ (\sigma_i) = h_{\sigma_i}(\underset{\rightarrow}{P}) \qquad (2.13)$$

An exact computation of h_β for a higher order gate may be done by performing the operations indicated on the right-hand side of equation (2.12) and using the idempotency property of σ_i i.e. $\sigma_i^2 = \sigma_i$. An expression for σ_M linearly dependent on σ_i for all i will be thus obtained. It is then possible to apply equation (2.13) yielding h_β as a function of h_{σ_i} $(i=1,\ldots,n)$.

For a higher-order module involving a large number of cut-sets, such an evaluation technique would be, however, too complex. So that for these cases it is preferred to use an approximation by applying the familiar minimal cut-set upper bound formula

$$h_\beta(\underset{\sim}{p}) \leq \coprod_{j=1}^{N_k} \ \underset{i\varepsilon K_j}{\pi} \ h_{\sigma_i}(\underset{\sim}{p}) \qquad (2.14)$$

which in its first order expansion reduces to the rare-event approximation

$$h_\beta(\underset{\sim}{p}) \leq \sum_{j=1}^{N_k} \ \underset{i\varepsilon kj}{\pi} \ h_{\sigma_i}(\underset{\sim}{p}) \qquad (2.15)$$

It may be seen now that the top event occurrence probability, P(TOP), can be derived by successivly using, where-ever necessary, the minimal cut upper bound approximation for the evaluation of modular reliabilities contained in the fault tree.

The following theorem states that such a series of approximations will yield an upper bound value closer to P(TOP) than that obtained by applying the minimal cut upper bound to the family of cut-sets characterizing the full fault tree. The proof of the theorem closely follows the line of arguments given by Barlow and Proschan [1] to show the analogous result for the minimal path lower bound approximation to P(TOP).

Theorem: Let $\Theta(\underset{\sim}{Y}^N)$ be a coherent structure of independent components with modular decomposition

$$\{(M_1,\sigma_1), (M_2,\sigma_2),\ldots.,(M_r,\sigma_r)\}$$

and organizing coherent structure function β i.e.

$$\Theta(\underset{\sim}{Y}^N) = \beta(\sigma_1,\sigma_2,\ldots,\sigma_r) \qquad (2.16)$$

with $M_i \Omega M_j$ = the empty set for $i \neq j$. Then

$$h_\Theta(\underset{\sim}{P}) \leq \upsilon_\beta(\upsilon_{\sigma_1}(\underset{\sim}{P}), \ldots, \upsilon_{\sigma_r}(\underset{\sim}{P}))$$
$$\leq \upsilon_\Theta(\underset{\sim}{P}) \tag{2.17}$$

Here υ_γ denotes the minimal cut upper bound for a coherent structure function $\gamma(y_1, \ldots, y_m)$ i.e.

$$\gamma(\underset{\sim}{Y}^M) = \coprod_{j=1}^{N_k} \prod_{i \varepsilon K_j} y_i \Rightarrow \upsilon_\gamma(\underset{\sim}{P}) = \coprod_{j=1}^{N_k} \prod_{i \varepsilon K_j} P_i \tag{2.18}$$

In order to prove the theorem (equation 2.17), it is necessary to first introduce the following Lemma:

<u>Lemma</u>: Let a coherent structure function γ consist of n modules connected in series, that is

$$\gamma(\underset{\sim}{Y}) = \prod_{i=1}^{n} \gamma_i(\underset{\sim}{Y}) \tag{2.19}$$

and consider all components to be statistically independent. Then

$$\prod_{i=1}^{n} \upsilon_{\gamma_i}(\underset{\sim}{P}) \leq \upsilon_\gamma(\underset{\sim}{P}) \tag{2.20}$$

<u>Proof of Lemma</u>: We may represent γ_i in terms of its minimal cut-set structure functions $\lambda_{i1}, \ldots, \lambda_{ik_i}$ as

$$\gamma_i = \coprod_{j=1}^{K_i} \lambda ij \ (\underset{\sim}{Y}) \qquad (i=1,\ldots,n) \tag{2.21}$$

it follows that

$$\upsilon_i(\underset{\sim}{P}) = \coprod_{j=1}^{k_i} P(\lambda_{ij}(\underset{\sim}{Y}) = 1) \tag{2.22}$$

and hence

$$\prod_{i=1}^{n} \upsilon_i(\underset{\sim}{P}) = \prod_{i=1}^{n} \coprod_{j=1}^{K_i} P(\lambda_{ij}(\underset{\sim}{Y})=1) \tag{2.23}$$

Now, if we replace replicated components in the minimal cut-set representation for $\gamma_i(\underset{\sim}{Y})$ by identical but mutually independent components, we will obtain a new coherent structure function γ^1 having the same upper bound as γ i.e.

$$\upsilon_{\gamma^1}(\underset{\sim}{P}) = \upsilon_\gamma(\underset{\sim}{P}) \tag{2.24}$$

But by the definition of γ^1

$$h_{\gamma^1}(\underset{\sim}{P}) = \prod_{i=1}^{n} \upsilon_{\gamma_i}(\underset{\sim}{P}) \tag{2.25}$$

therefore

$$\prod_{i=1}^{n} \upsilon_{\gamma_i}(\underset{\sim}{P}) \leq \upsilon_\gamma(\underset{\sim}{P}) \tag{2.26}$$

q.e.d.

<u>Proof of Theorem</u>: Let $\nu_1, \nu_2, \ldots, \nu_t$ denote the minimal cut-set structure functions of the organizing coherent structure function $\beta(\sigma_1, \ldots, \sigma_r)$; let $\beta_j(\underset{\sim}{Y}) = \nu_j[\sigma_1(\underset{\sim}{Y}), \ldots, \sigma_r(\underset{\sim}{Y})]$ be the minimal cut-set indicator function constituted

by a number of modules (M_{i1}, $M_{i2}, \ldots, M_{i\nu_j}$) which are necessarily connected in series. And let μ_{j1}, $\mu_{j2}, \ldots, \mu_{jtj}$ denote the minimal cut-set structure functions for β_j $(j=1,2,\ldots.t)$.

Then

$$\{\mu_{jk}\}\begin{matrix} k=1,\ldots,t_j \\ j=1,\ldots,t \end{matrix}$$

constitute the set of minimal cut-set structure function of $\Theta(\underset{\sim}{Y}^N)$ since (a) each μ_{jk} is distinct given that the modules in the structure $\beta(\sigma_1,\ldots,\sigma_r)$ are disjoint. (b) $\mu_{jk} = 1 \Rightarrow \nu_j = 1 \Rightarrow \beta = 1 \Rightarrow \Theta = 1$ therefore μ_{jk} is a cut-set structure function of β. Moreover the sets μ_{jk} are minimal.

It follows that

$$\upsilon_\Theta(\underset{\sim}{P}) = \coprod_{j=1}^{t} \coprod_{k=1}^{t_j} h_{\mu_{jk}}(\underset{\sim}{P}) \tag{2.27}$$

Furthermore since the modular components of ν_j are connected in series, one may apply the above Lemma to obtain

$$h_{\nu_j}(\upsilon_{\sigma 1}(\underset{\sim}{P}),\ldots.,\upsilon_{\sigma r}(\underset{\sim}{P})) \leq \upsilon_{\beta_j}(\underset{\sim}{P}) \tag{2.28}$$

Finally, using (2.27) and (2.28) it follows that

$$\upsilon_\beta(\upsilon_{\sigma 1}(\underset{\sim}{P})\ldots,\upsilon_{\sigma r}(\underset{\sim}{P})) = \coprod_{y=1}^{t} h_{\nu_j}(\upsilon_{\sigma_1}(\underset{\sim}{P}),\ldots,\upsilon_{\sigma r}(\underset{\sim}{P}))$$

$$\leq \overset{t}{\underset{j=1}{}} \upsilon_{\beta j}(\underset{\sim}{P}) = \coprod_{y=1}^{t} \coprod_{k=1}^{t_j} h\mu_{jk} = \upsilon_\phi(\underset{\sim}{P}) \tag{2.29}$$

q.e.d.

II.5. Reliability Importance of Modules

II.5.1 Summary of Reliability Importance Measures

It has been shown that for a modularized fault tree, the evaluation of the top event occurrence probability P(TOP) requires that the occurrence probabilities of all the intermediate gate events corresponding to a module in the fault tree be evaluated in advance. It is obvious, however, that because of the recursive nature of the modular equations, the execution of this task may be done very efficiently. Furthermore, it will be shown in this section that the additional information obtained in this process, i.e., the modular reliabilities, is needed to evaluate the reliability importance of each of the modules and basic events contained in the fault tree.

In Chapter I several measures of importance were introduced and defined in terms of $h(\underset{\rightarrow}{P})$ the top event occurrence probability given as a function of the occurrence probabilities of the basic events

$$P(TOP) = E(\Theta(\underset{\rightarrow}{Y}^N)) = \text{Prob}\ [\Theta(\underset{\rightarrow}{Y}) = 1] = h(\underset{\rightarrow}{P}) \qquad (2.30)$$

with $\underset{\rightarrow}{Y} = (y_1, y_2, \ldots, y_n)$ and $\underset{\rightarrow}{P} = (P_1, P_2, \ldots, P_n)$ defined as

$$E(y_i) = \text{Prob}\ (y_i = 1) = P_i \qquad (2.31)$$

Thus, Birnbaum's measure of importance for system's component i was defined as the rate of change of the overall system reliability as the reliability of component i is changed.

$$I_i^B = \frac{\partial h(\underset{\sim}{P})}{\partial P_i} = h(1_i, \underset{\sim}{P}) - h(0_i, \underset{\sim}{P}) \tag{2.32}$$

The criticality importance of component i was defined as the probability that the system is in a state in which component is both "critical" to the system and is in a failed state, given that the system has failed

$$I_i^{Cr} = \frac{\text{Prob (i critical)} \cdot P_i}{h(\underset{\sim}{P})} \tag{2.33}$$

where component i is defined to be critical to the system if the system fails provided i is in a failed state but does not fail if component i is not in a failed state, i.e., it is required that the state vector $\underset{\sim}{Y}$ be such that

$(1_i, \underset{\sim}{Y}) = 1$ and $(0_i, \underset{\sim}{Y}) = 0$

(Recall $(1_i, \underset{\sim}{Y}) \equiv \Theta(Y_1, Y_2, \ldots, Y_i = 1, \ldots, Y_n)$

Hence

$$\text{Prob(i critical)} = P(\{\Theta(1_i, \underset{\sim}{Y}) - \Theta(0_i, \underset{\sim}{Y})\} = 1)$$

$$= P(\Theta(1_i, \underset{\sim}{Y}) = 1) - P(\Theta(0_i, \underset{\sim}{Y}) = 1) \tag{2.34}$$

$$\Rightarrow P(\text{i critical}) = h(1_i, \underset{\sim}{P}) - h(0_i, \underset{\sim}{P}) \tag{2.35}$$

By substituting equation (2.35) into equation (2.33), the following equation is derived:

$$I_i^{Cr} = \frac{(h(1_i, \underset{\sim}{P}) - h(0_i, \underset{\sim}{P}))}{h(\underset{\sim}{P})} P_i \tag{2.36}$$

The Vesely-Fussell importance measure for component i was defined as the probability that component i will contribute to system failure, given that the system is in a failed state. As component i contributes to system failure only if a cut-set containing i has failed, it is convenient to define $\Theta_k^i(\underset{\sim}{Y})$ to be the Boolean operator function for the union of all cut-sets containing event i

$$\Theta_k^i\ (\underset{\sim}{Y}) = \coprod_{j=1}^{N_k^i} \prod_{\substack{\ell \varepsilon Kj \\ i \varepsilon Kj}} Y_\ell \tag{2.37}$$

with N_k^i = number of cut-sets containing basic event i, $\ell \varepsilon K_j$ and $i \varepsilon K_j$ implies index ℓ includes all basic events in cut-set Kj which necessarily contains event i. Then in terms of $\Theta_k^i\ (\underset{\sim}{Y})$ the Vesely-Fussell importance of component i is given by

$$I_i^{V.F.} = \frac{P(\Theta_k^i\ (\underset{\sim}{Y})=1)}{P(\Theta(\underset{\sim}{Y})=1)} = \frac{h_i\ (\underset{\sim}{P})}{h(\underset{\sim}{P})} \tag{2.38}$$

II.5.2 The Birnbaum and Criticality Measures of Importance for Modules

Since for a modularized fault tree each of its modules may be considered as a super-component independent of the rest of the tree, the above definitions may also correctly apply for modular importances. Thus, if $\sigma(\underset{\sim}{Y}^M)$ is the coherent structure function associated with module M for a fault tree characterized by coherent structure function $\Theta(\underset{\sim}{Y}^N)$, i.e.

$$\Theta(\underset{\sim}{Y}^N) = \alpha(\sigma(\underset{\sim}{Y}^M), \underset{\sim}{Y}^{MC}) \tag{2.39}$$

and

$$h_\Theta(\underset{\sim}{P}) = h_\alpha\,(h_\sigma(\underset{\sim}{P}^M), \underset{\sim}{P}^{MC}) \tag{2.40}$$

then Birnbaum's importance measure for module M will be

$$I^B_{\alpha,M} = \frac{\partial h_\alpha(h_\sigma(\underset{\sim}{P}^M), \underset{\sim}{P}^{MC})}{\partial h_\sigma(\underset{\sim}{P}^M)} \tag{2.41}$$

and since the set M of inputs is disjoint from the rest of the tree, we can use a partial derivative chain rule to obtain the Birnbaum importance of input i contained in module M [5]

$$I^B_{\Theta,i} = \frac{\partial h_\alpha(h_\sigma(\underset{\sim}{P}^M), \underset{\sim}{P}^{MC})}{\partial h_\sigma(\underset{\sim}{P}^M)} \cdot \frac{\partial h_\sigma(\underset{\sim}{P}^M)}{\partial P_i} \tag{2.42}$$

$(i \varepsilon M)$

$$\Rightarrow I^B_{\Theta,i} = I^B_{\alpha,M} \;\; I^B_{\sigma,i} \tag{2.43}$$

In words, the above chain-rule states that the Birnbaum importance of event i is given by the product of its Birnbaum importance with respect to the module to which it belongs and the Birnbaum importance of the module with respect to the top tree event.

The criticality importance measure for module M is given by

$$I_M^{Cr} = \frac{\partial h_\alpha\ (h_\sigma(\underset{\sim}{p}^M),\ \underset{\sim}{p}^{MC})}{\partial h_\sigma(\underset{\sim}{p}^M)} \cdot \frac{h_\sigma(\underset{\sim}{p}^M)}{h_\alpha(h_\sigma(\underset{\sim}{p}^M),\ \underset{\sim}{p}^{MC})} \tag{2.44}$$

so a reliability change in module M proportional to its expectation value

$$\Delta h_\sigma = C_M h_\sigma(\underset{\sim}{p}^M) \tag{2.45}$$

causes a system reliability fractional change given by

$$C_\alpha = \frac{\Delta h_\alpha}{h_\alpha} = C_M\ I_M^{Cr} \tag{2.46}$$

II.5.3 The Vesely-Fussell Importance Measure for Modules

The Vesely-Fussell importance measure for module M will be given by

$$I_M^{V.F.} = \frac{\text{Prob}\ (\alpha_K^M\ (\sigma(\underset{\sim}{Y}^M),\ \underset{\sim}{Y}^{MC}) = 1)}{\text{Prob}\ (\alpha(\sigma(\underset{\sim}{Y}^M),\ \underset{\sim}{Y}^{MC}) = 1)} \tag{2.47}$$

with $\alpha_K^M\ (\sigma(\underset{\sim}{Y}^M),\ \underset{\sim}{Y}^{MC})$ defined to be the Boolean operator function for the union of all cut-sets of $\alpha(\sigma(\underset{\sim}{Y}^M),\ \underset{\sim}{Y}^{MC})$ containing super-component event $\sigma(\underset{\sim}{Y}^M)$, i.e.

$$\alpha_K^M\ (\sigma(\underset{\sim}{Y}^M),\ \underset{\sim}{Y}^{MC}) = \coprod_{j=1}^{N_K^\sigma} (\sigma \prod_{\substack{\ell \varepsilon K_j \\ \sigma \varepsilon K_j}} Y_\ell) \tag{2.48}$$

with

$$Y_\ell \varepsilon \underset{\sim}{Y}^{MC}\ ,\ \sigma = \sigma(\underset{\sim}{Y}^M),\ \ N_k^\sigma = \text{number of cut-sets}$$

containing super-component σ and K_j a cut-set containing necessarily the super-component state σ.

Chatterjee [6] has shown that a chain-rule, analogous to the one given for the Birnbaum importance of component i in module M (equation 2.43), holds for the Vesely-Fussell importance measure, namely

$$I^{V.F.}_{\theta,i} = I^{V.F.}_{\alpha,M} \quad I^{V.F.}_{\sigma,i} \tag{2.49}$$

with

$$\theta(\underset{\sim}{Y}) = \alpha(\sigma(\underset{\sim}{Y}^M), \underset{\sim}{Y}^{MC}) \text{ and } Y_i \in \underset{\sim}{Y}^M.$$

This relation has been proven by Chatterjee as follows: The family of minimal cut-sets of $\theta(\underset{\sim}{Y})$ containing events i($=K_\theta(i)$) may be generated by taking the family of minimal cut-sets of $\alpha(\sigma(\underset{\sim}{Y}^M), \underset{\sim}{Y}^{MC})$ which include module M ($=K\alpha(M)$) and then substituting superevent M by the family of minimal cut-sets of $\sigma(\underset{\sim}{Y}^M)$ which contain event i ($= K_\sigma(i)$), therefore

$$K_\theta(i) = K_\sigma(i) \times \{K_\alpha(M) - (M)\} \tag{2.50}$$

By defining the following events

A = at least one of the minimal cut-sets of module M which contains i fails, i.e., $K\sigma(i)$ fails.

B = at least one of the minimal cut-sets of module M fails, i.e. K_σ fails (notice $A \subset B$).

C = at least one of the elements of $K\alpha(M)-(M)$ fails (notice event C is disjoint with any event within the module).

It follows that $C\Omega B$ is the event = module causes system failure. And $A\Omega B\Omega C$ is the event = module causes system failure with event i failing.

Also, one has

$$P(A \Omega B \Omega C) = P(B) \cdot P(A \Omega B | B) \cdot P(C) \qquad (2.51)$$

since event C is independent of A and B, and $P(A \Omega B | B)$ is the conditional probability that event $A \Omega B$ occurs, given that event B has occurred.

Furthermore, since $A \subset B$ then $A \Omega B = A$ and since C and B are independent events $P(C)P(B) = P(C \Omega B)$, hence

$$P(A \Omega B \Omega C) = P(A|B) \cdot P(C \Omega B) \qquad (2.52)$$

It is now only necessary to realize that the following relations hold

$$I^{V.F.}_{\Theta,i} = \frac{P(\text{i has failed with at least one of its minimal cut-sets})}{P(\text{the system has failed})}$$

$$\Rightarrow \quad I^{V.F.}_{\Theta,i} = \frac{P(A|B) \cdot P(C \Omega B)}{h_{\Theta}(\underset{\sim}{P})} \qquad (2.53)$$

also

$$I^{V.F.}_{\sigma,i} = P(A|B) \qquad (2.54)$$

$$I^{V.F.}_{\alpha,M} = \frac{P(C \Omega B)}{h_{\Theta}(\underset{\sim}{P})} \qquad (2.55)$$

Hence

$$I^{V.F.}_{\Theta,i} = I^{V.F.}_{\alpha,M} \quad I^{V.F.}_{\sigma,i} \qquad (2.56)$$

q.e.d.

II.5.4 Evaluation of the Vesely-Fussell Importance Measures for a Modularized Fault Tree

In what follows it will be shown how the Vesely-Fussell importance for modules and basic events can be easily computed from a knowledge of the modular structure of a fault tree by a successive use of the recursive modular equations

$$\sigma_M = \beta(\sigma_1,\sigma_2,\ldots,\sigma_n) \tag{2.57}$$

and by using the Vesely-Fussell modular importance chain-rule

$$I^{V.F.}_{\theta,i} = I^{V.F.}_{\alpha,M} \; I^{V.F.}_{\sigma,i} \tag{2.58}$$

Indeed, for the case of the super-module σM composed of modules $(\sigma_1,\sigma_2,\ldots,\sigma_n)$, the Vesely-Fussell importance of each of these modules is given by

$$I^{V.F.}_{\theta,\sigma_j} = I^{V.F.}_{\alpha,M} \; I^{V.F.}_{\beta,\sigma_j} \tag{2.59}$$

$(j=1, 2,\ldots n)$

with

$$\theta(\underset{\sim}{Y}) = \alpha(\sigma_M(\underset{\sim}{Y}^M), \underset{\sim}{Y}^{MC}) \tag{2.60}$$

Equation (2.59) giving the V.F. importance of modules $(\sigma_1,\ldots,\sigma_n)$ contained in σ_M with respect to the TOP tree event, acquires a very simple form for the case of "simple" AND and OR gates. Thus, for an AND gate (Figure 2.5) super-module the following equation results

$$\sigma_M = \sigma_{AND} = \prod_{j=1}^{n} \sigma_j \qquad (2.61)$$

Therefore, a failure of the super-module implies necessarily that all of its modules have failed, i.e., the probability that module σ_j (j=1, 2,...n) contributes to failure of σ_M given that σ_M has failed equals one

$$I^{V.F.}_{\beta,\sigma_j} = 1. \qquad (2.62)$$

$$I_{\theta,\sigma_j} = I^{V.F.}_{\alpha,M} \qquad (2.63)$$

In other words, a module σ_j which is an input to an AND gate super-module σ_M will have the same V.F. importance with respect to the TOP tree event as the super-module σ_M.

For the case of an OR gate super-module (Figure 2.6), the structure function will be given by

$$\sigma_M = \sigma_{OR} = \coprod_{j=1}^{n} \sigma_j \qquad (2.64)$$

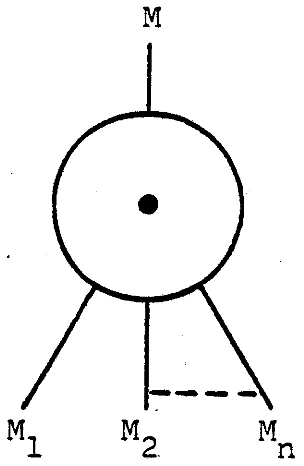
Here, module σ_j contributes to the failure of σ_M only through the single event cut-set (M_j). Therefore the probability that it contributes to the failure of σ_M given that σ_M has failed is

$$I^{V.F.}_{\beta,\sigma_j} = \frac{h_{\sigma_j}(\underset{\sim}{p}^{M_j})}{h_{\sigma_M}(\underset{\sim}{p}^{M})} \qquad (2.65)$$

$$\Longrightarrow \quad I^{V.F.}_{\theta,\sigma_j} = I^{V.F.}_{\alpha,M} \; \frac{h_{\sigma_j}}{h_{\sigma}} \qquad (2.66)$$

Cut-sets

$K = (M_1, M_2, \ldots, M_n)$



$$I^{V.F.}_{M_i} = I^{V.F.}_{M}$$

$i = 1, 2, \ldots, n$

FIGURE 2.5 AND GATE SUPER-MODULE

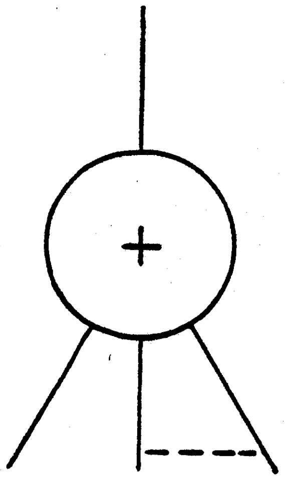
Cut-sets

$K_1 = (M_1)$

$K_2 = (M_2)$

$\vdots$

$K_n = (M_n)$



$$I^{V.F.}_{M_i} = I^{V.F.}_{M} \ \frac{h_{M_i}}{h_M}$$

$i = 1, 2, \ldots, n$

FIGURE 2.6 OR GATE SUPER-MODULE

It should be noticed here that $h\sigma_j$ and $h\sigma$ are the modular reliabilities which were needed to be evaluated in advance to fine the TOP tree event occurrence probability P(TOP).

Finally, the evaluation of the Vesely-Fussell importance of modules σ_j which are inputs to a higher order prime module σ_M (Figure 2.7) have to be considered:

$$\sigma_M = \beta(\sigma_1, \ldots, \sigma_n) = \coprod_{\ell=1}^{N_k^j} \prod_{i \varepsilon K\ell} \sigma_i \qquad (2.67)$$

$(i = 1, 2, \ldots, n)$

The probability that module σ_i will contribute to the failure of its parent module σ_M, given that the parent module has failed is given by

$$I^{V.F.}_{\beta,\sigma_j} = \frac{P(\beta_K^j\ (\sigma_1, \ldots, \sigma_n) = 1)}{P(\beta(\sigma_1, \ldots, \sigma_n) = 1)} \qquad (2.68)$$

now

$$P(\beta(\sigma_1, \ldots, \sigma_n) = h_{\sigma_M} \qquad (2.69)$$

and equation (2.67) implies that β_K^j is given by

$$\beta_K^j = \coprod_{\ell=1}^{N_K^j} \prod_{\substack{\ell \varepsilon K_\ell \\ j \varepsilon K_\ell}} \sigma\ell \qquad (2.70)$$

Thus, the V.F. importance for module $_j$ with respect to the TOP event will be

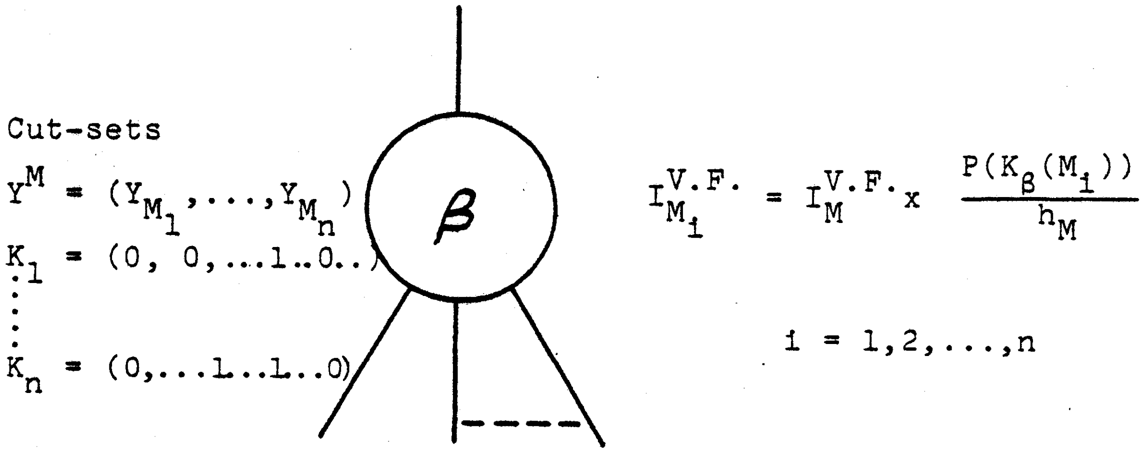

Cut-sets
$Y^M = (Y_{M_1}, \ldots, Y_{M_n})$
$K_1 = (0, 0, \ldots 1 \ldots 0 \ldots)$
$K_n = (0, \ldots 1 \ldots 1 \ldots 0)$
β
$I_{M_i}^{V.F.} = I_M^{V.F.} \times \frac{P(K_\beta(M_i))}{h_M}$
$i = 1, 2, \ldots, n$

FIGURE 2.7 HIGHER ORDER PRIME GATE SUPER-MODULE

$$I^{V.F.}_{\theta,\sigma_j} = I^{V.F.}_{\alpha,M} \frac{P\left(\left(\coprod_{\ell=1}^{N_K^j} \prod_{\substack{\ell \varepsilon K\ell \\ j \varepsilon K\ell}} \sigma\ell\right) = 1\right)}{h\sigma_M} \qquad (2.71)$$

CHAPTER THREE

PL-MOD: A FAULT TREE MODULARIZATION COMPUTER PROGRAM WRITTEN IN PL-1

III.1 Introduction

As pointed out in Chapter II, it is possible to find for any fault tree diagram an equivalent tree representation such that all of its intermediate gates correspond to a modular super-event independent from the rest of the tree. Furthermore, these modular gates are associated with Boolean logic functions which are either "prime", i.e., they are represented by an irreducible set of minimal cut-sets, or are "simple" of maximal size, i.e., they are AND or OR gates having no inputs from other gates of the same type.

A number of computational advantages result by using this modular representation to analyze fault trees:

(a) Probabilities of occurrence for the TOP and intermediate gate events may be efficiently computed, by evaluating these modular events in the same order that they are generated;

(b) Modular and component importance measures are easily computed by starting at the TOP tree event and successively using a modular importance chain-rule;

(c) For complex fault trees necessitating the use of minimal cut-set upper bounds for their quantification, sharper bounds will result by using the minimal cut-set upper bound at the level of modular gates.

In this chapter, an algorithm will be given for arriving at the modular decomposition of fault trees. The implementation of the algorithm by the computer code PL-MOD will be discussed and its operation shall be illustrated by means of the familiar Pressure Tank Rupture fault tree example [1]. Finally, it will be shown how PL-MOD proceeds to use the modular information for the evaluation of modular event occurrence probabilities and of modular and component Vesely-Fussell importance measures.

III.2. Algorithm for the Modular Decomposition of Fault Trees

In Figure 3.1 a flow-chart is given for the algorithm used by PL-MOD to modularly decompose fault trees.

The tree modularization is achieved by performing a series of manipulations on its nodes as outlined by the following steps:

(a) Each NODE in the fault tree is defined as a gate operator (AND , OR, K-out-of-N) together with a set of attached input gates and basic event components (Figure 3.2).

(b) A NODE's output will be an input to another NODE defined to be its NODE ROOT (Figure 3.3).

(c) NODES having common replicated inputs are interconnected (Figure 3.4). These interconnections then identify sets of nodes which are not immediately modularizable in the original form of the fault tree.

(d) The tree modular decomposition is simultaneously started at all bottom branch gate nodes (Figure 3.5) defined to be those having no gate inputs (GATELESS NODES).

(e) Simple (AND,OR) gateless nodes having as NODE ROOT another gate of the same type (Figure 3.6), are coalesced with their NODE ROOT by transferring all their inputs to the NODE ROOT and thus reducing the number of gate inputs to the NODE ROOT.

(f) Simple gateless nodes having a gate of a different type as NODE ROOT are modularized (Figure 3.7). Those gateless nodes having replicated components or "nested sub-modules as inputs are temporarily transformed into "nested" modules (Figure 3.8), unless it is found that the set of replicated events within the gate is complete (Figure 3.9) in which case a modular minimal cut-set representation for its composition will be performed. The minimal cut-sets will then be constituted by replicated events and proper modules arising from each of the nested modules (Figure 3.10).

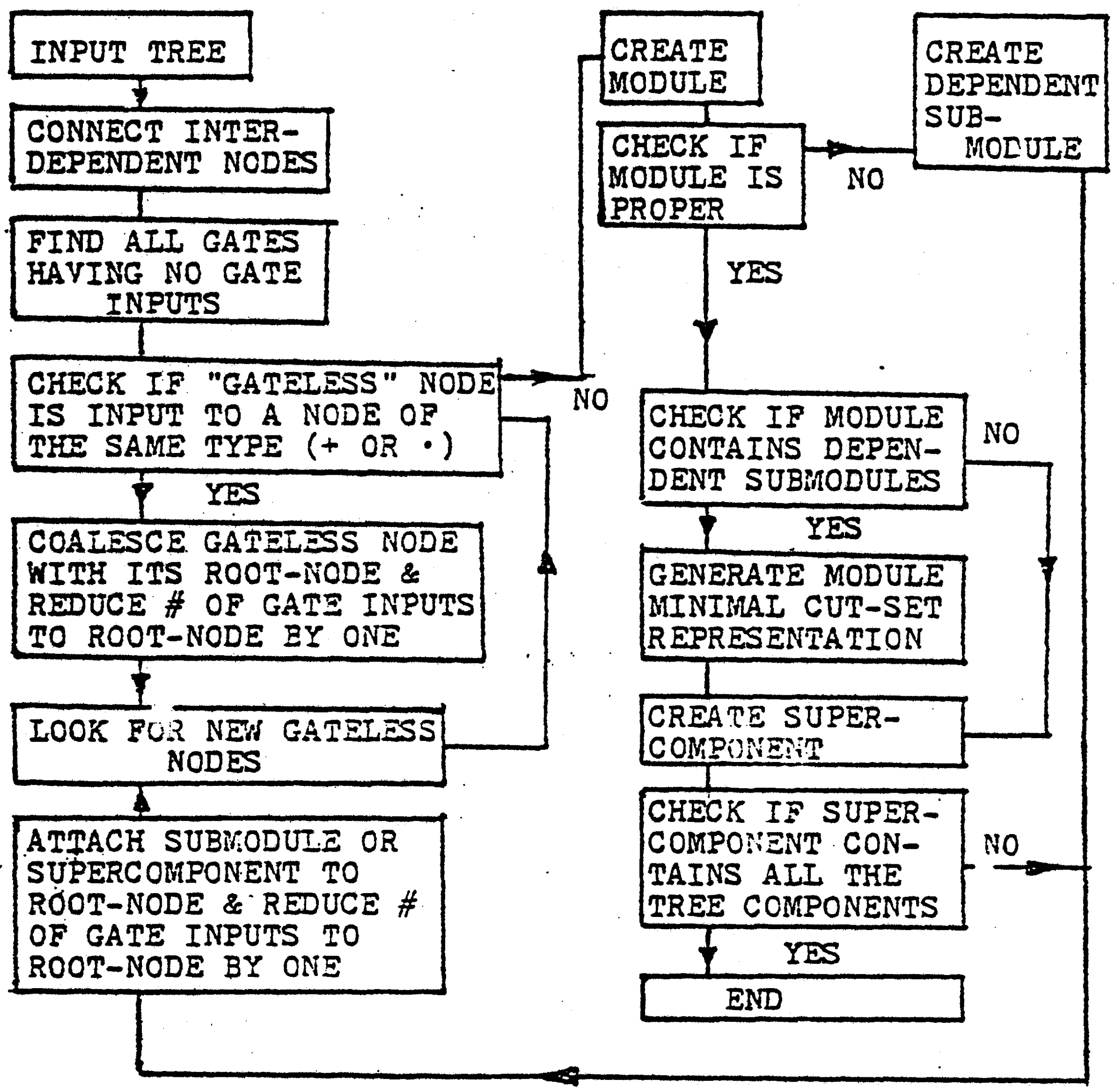
(g) Symmetric (K-out of-n) gate NODES are immediately modularized and given their Boolean representation (Figure 3.11).

(h) Nodes which have been transformed into proper modules or temporary nested sub-modules are attached to their NODE ROOT gate as additional component-like inputs thereby reducing the number of gate inputs to their NODE ROOT gate (Figure 3.12).

(i) As steps (e), (f), (g) and (h) reduce the number of gate inputs to each of the NODE ROOT gates attached to a gateless node, a new set of gateless nodes will necessarily be

FIGURE 3.1

FAULT TREE MODULARIZATION ALGORITHM


INPUT TREE
CONNECT INTER-DEPENDENT NODES
FIND ALL GATES HAVING NO GATE INPUTS
CHECK IF "GATELESS" NODE IS INPUT TO A NODE OF THE SAME TYPE (+ OR ·)
YES
NO
COALESCE GATELESS NODE WITH ITS ROOT-NODE & REDUCE # OF GATE INPUTS TO ROOT-NODE BY ONE
LOOK FOR NEW GATELESS NODES
ATTACH SUBMODULE OR SUPERCOMPONENT TO ROOT-NODE & REDUCE # OF GATE INPUTS TO ROOT-NODE BY ONE
CREATE MODULE
CHECK IF MODULE IS PROPER
NO
CREATE DEPENDENT SUB-MODULE
YES
CHECK IF MODULE CONTAINS DEPENDENT SUBMODULES
NO
YES
GENERATE MODULE MINIMAL CUT-SET REPRESENTATION
CREATE SUPER-COMPONENT
CHECK IF SUPER-COMPONENT CONTAINS ALL THE TREE COMPONENTS
NO
YES
END

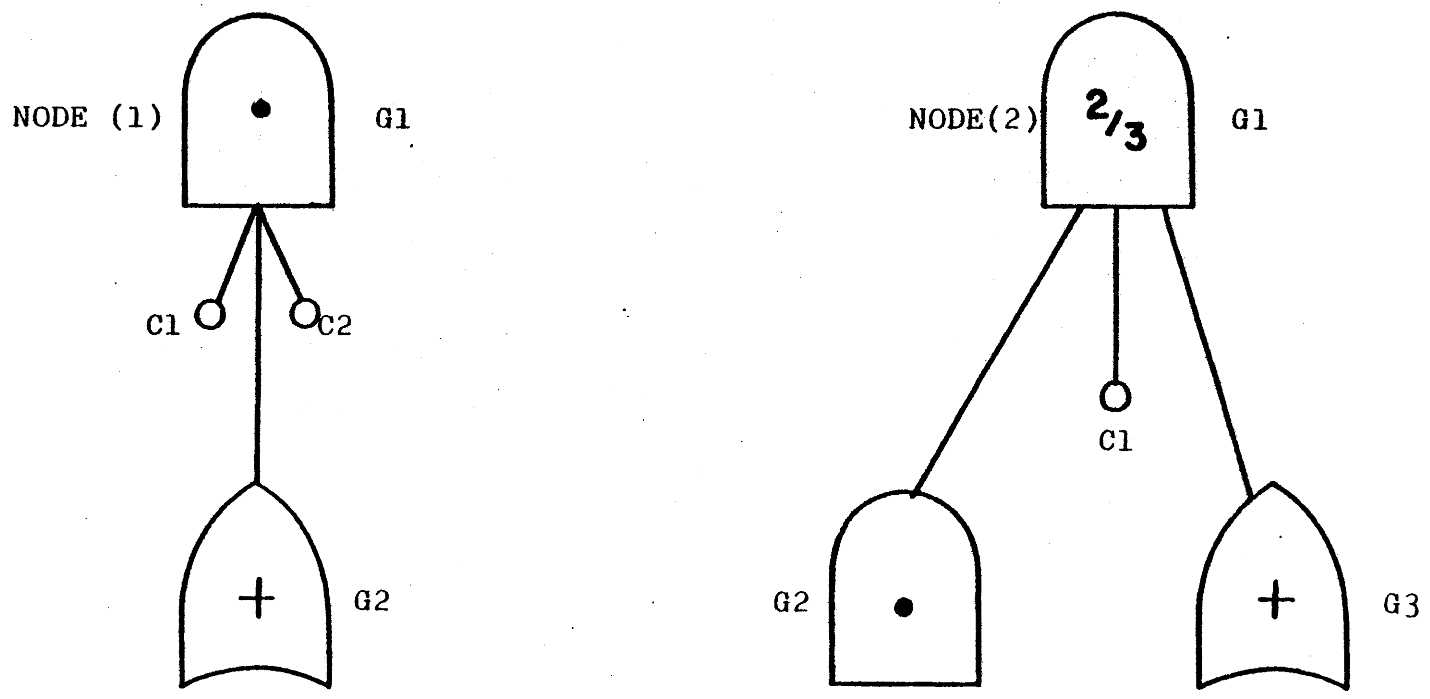


FIGURE 3.2

FAULT TREE NODES

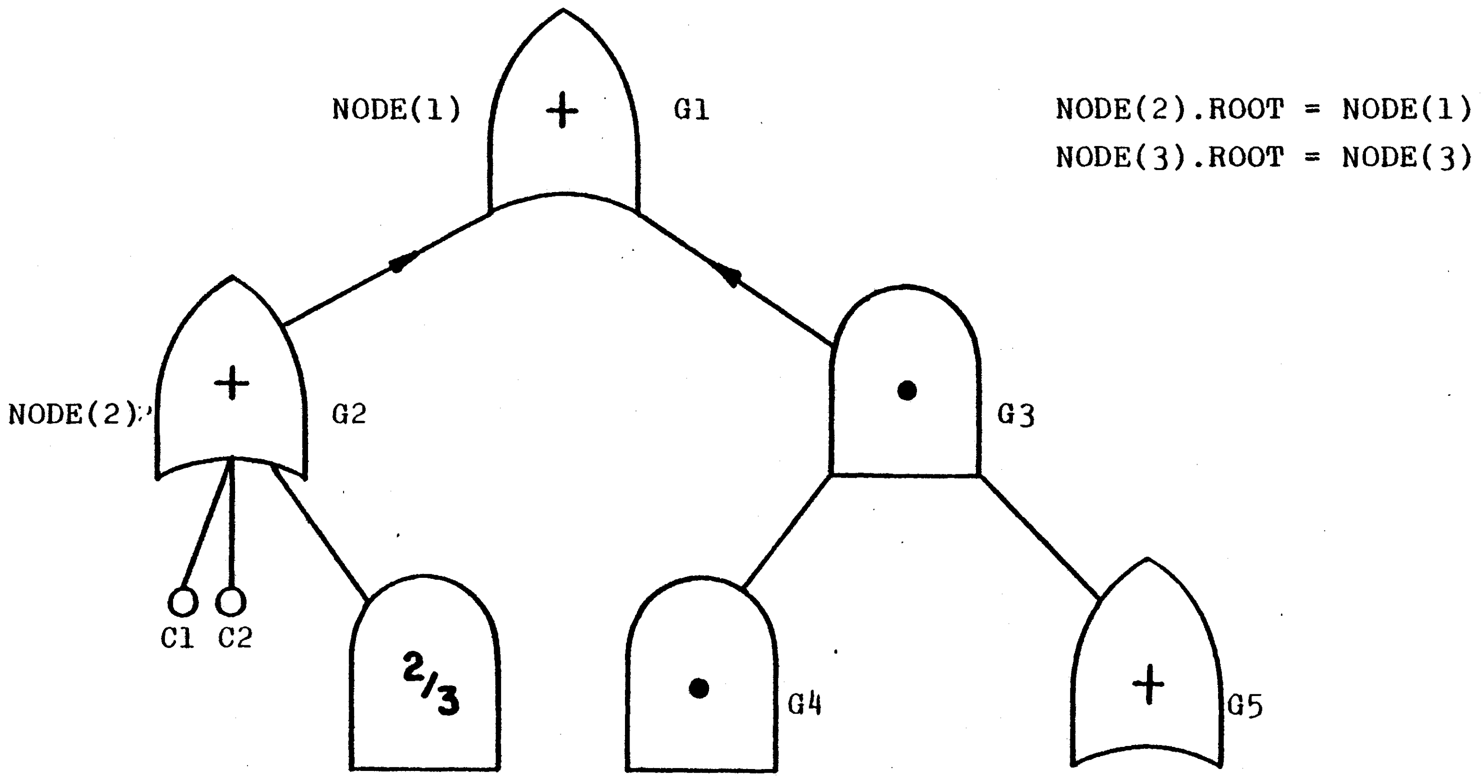


FIGURE 3.3

FAULT TREE NODE.ROOTS

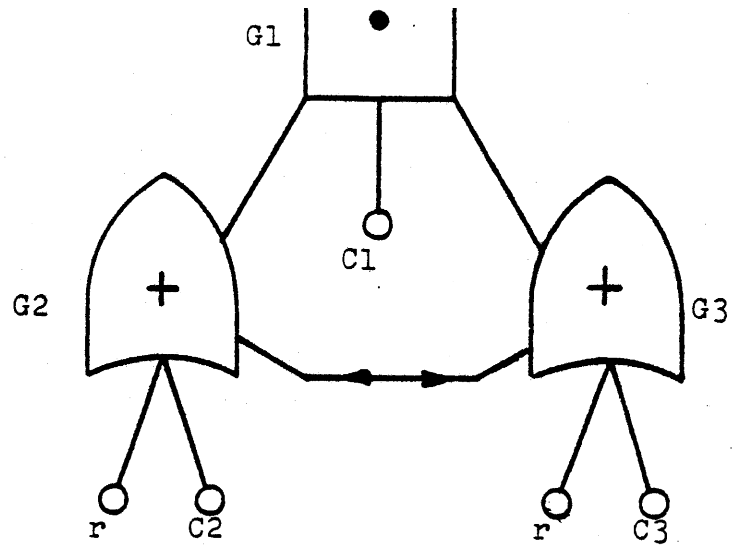
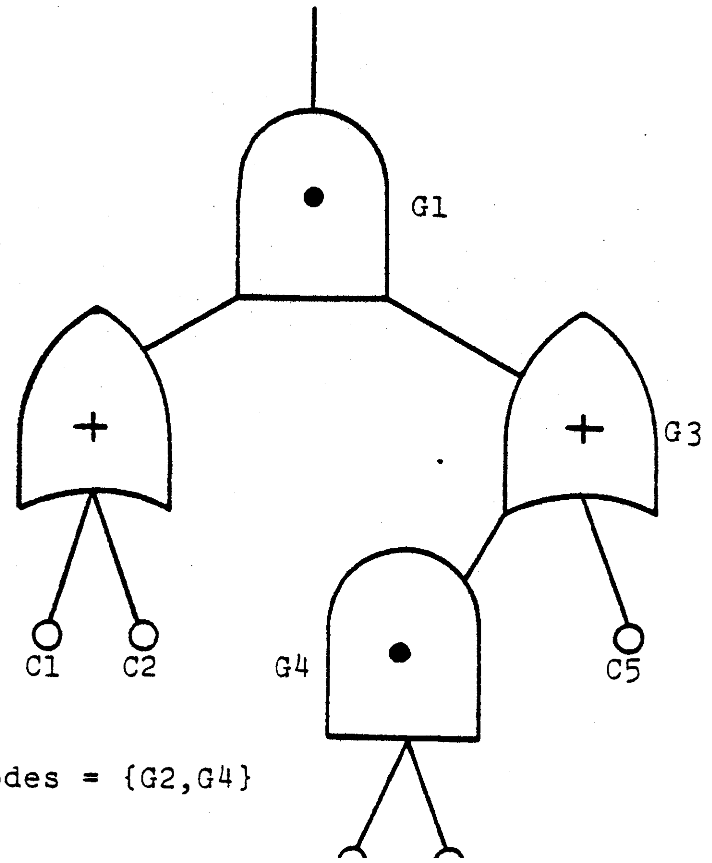


FIGURE 3.4   FAULT TREE NODE INTERCONNECTIONS



Bottom Nodes = {G2,G4}

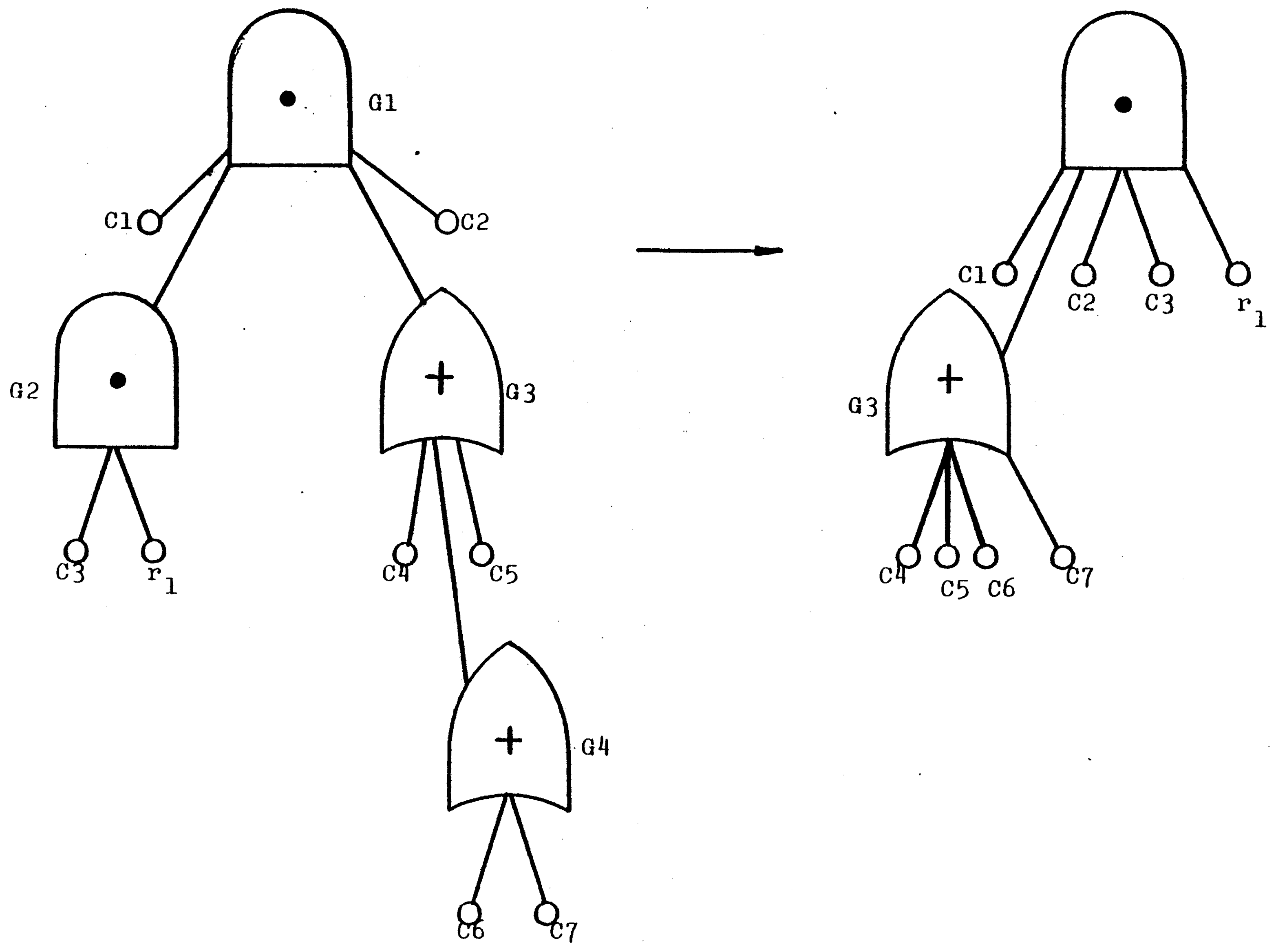
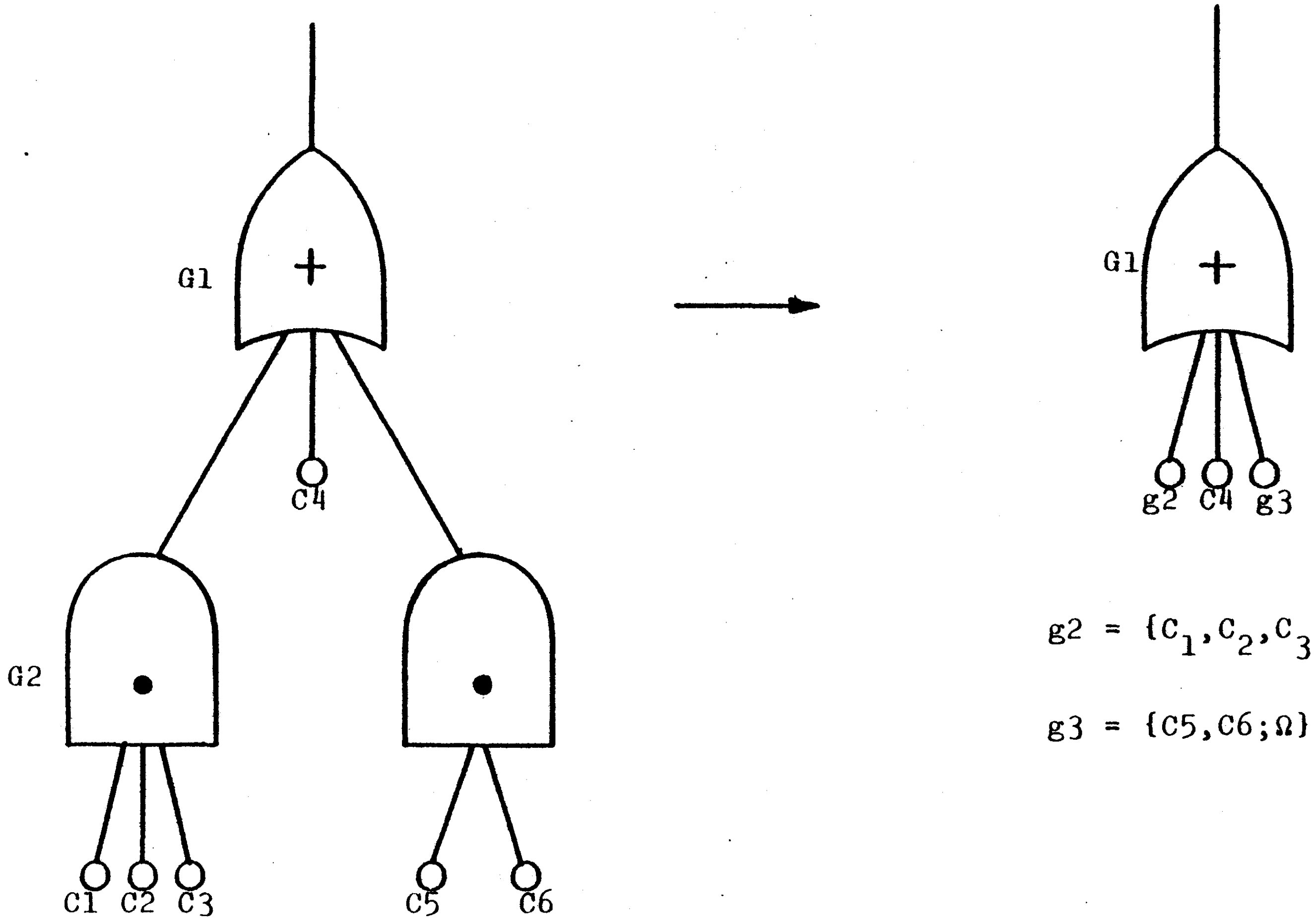


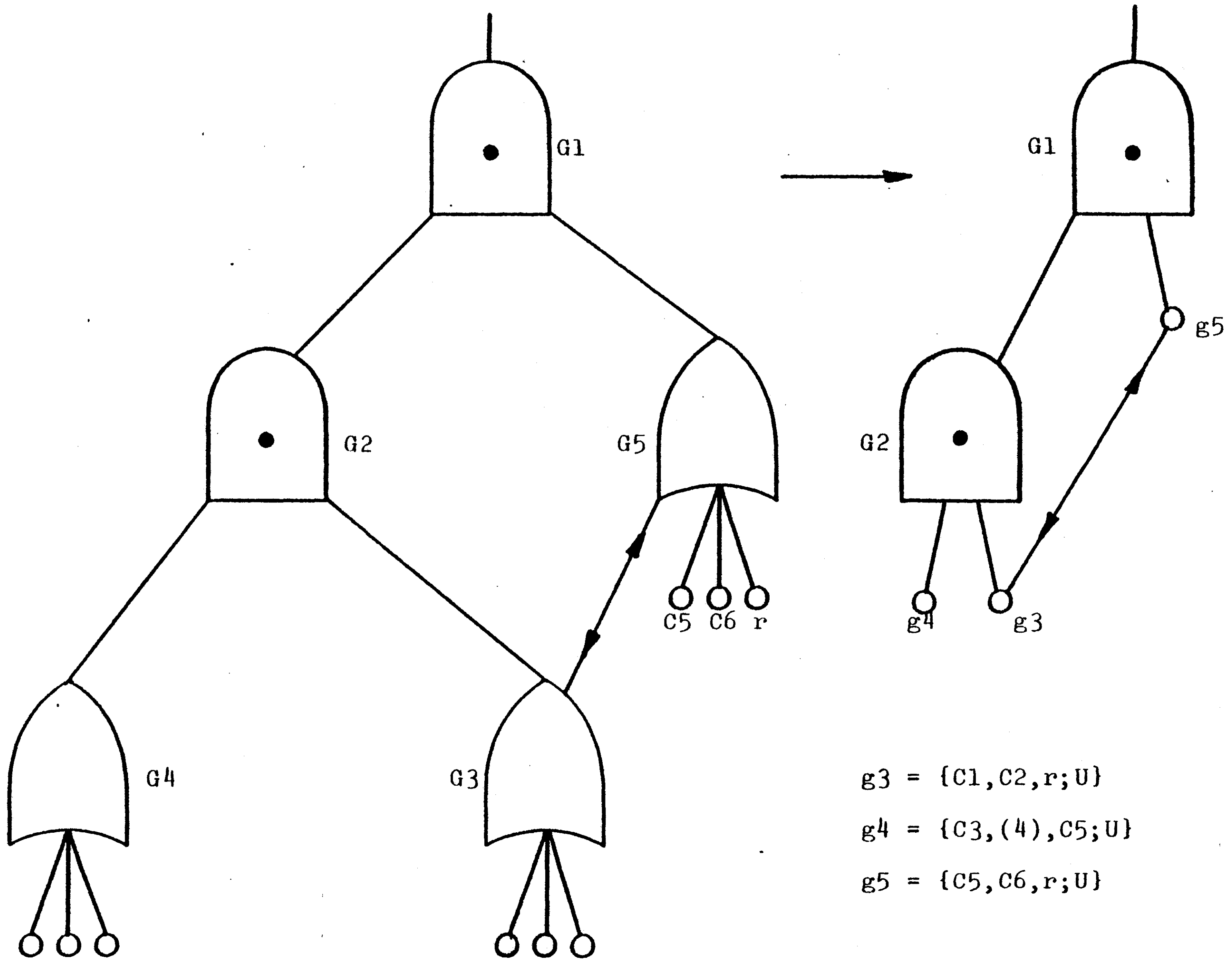
FIGURE 3.6 COALESCED GATELESS NODES



$g2 = \{C_1, C_2, C_3; \Omega\}$

$g3 = \{C5, C6; \Omega\}$

FIGURE 3.7 MODULARIZED GATELESS NODES



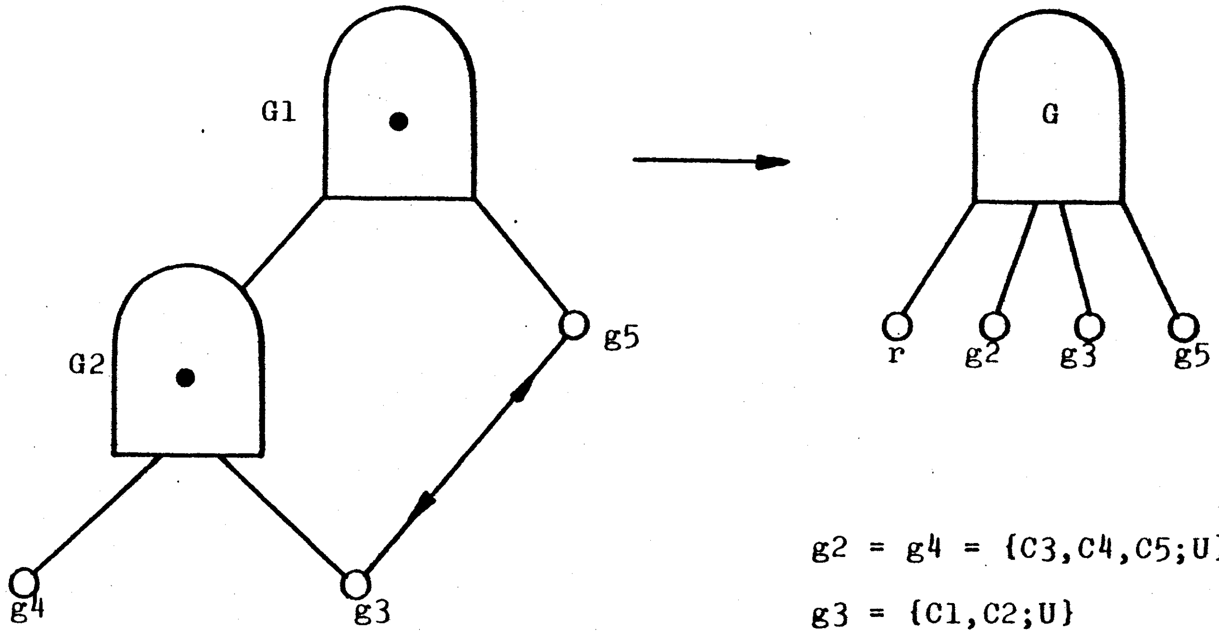
g3 = {C1,C2,r;U}

g4 = {C3,(4),C5;U}

g5 = {C5,C6,r;U}

FIGURE 3.8

INTERDEPENDENT NODES IN TEMPORARY NESTED MODULES g3, g5



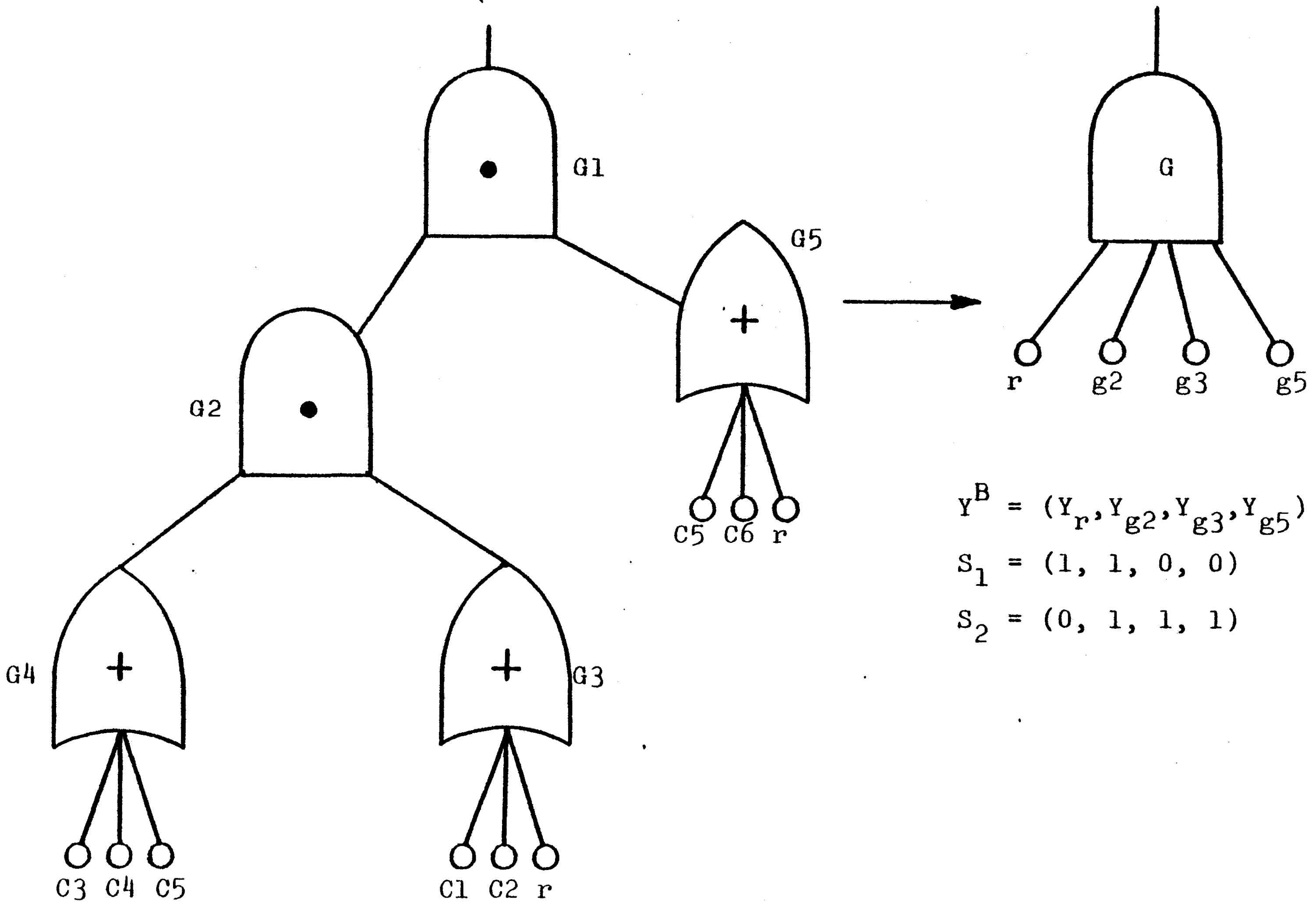
g2 = g4 = {C3,C4,C5;U}

g3 = {C1,C2;U}

g5 = {C5,C6;U}

FIGURE 3.9

COMPLETE SET OF NESTED SUB-MODULES



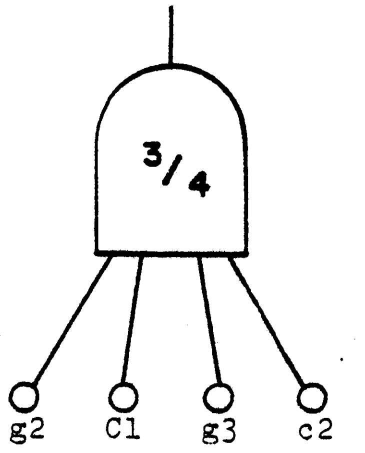
$Y^B = (Y_r, Y_{g2}, Y_{g3}, Y_{g5})$

$S_1 = (1, 1, 0, 0)$

$S_2 = (0, 1, 1, 1)$

FIGURE 3.10

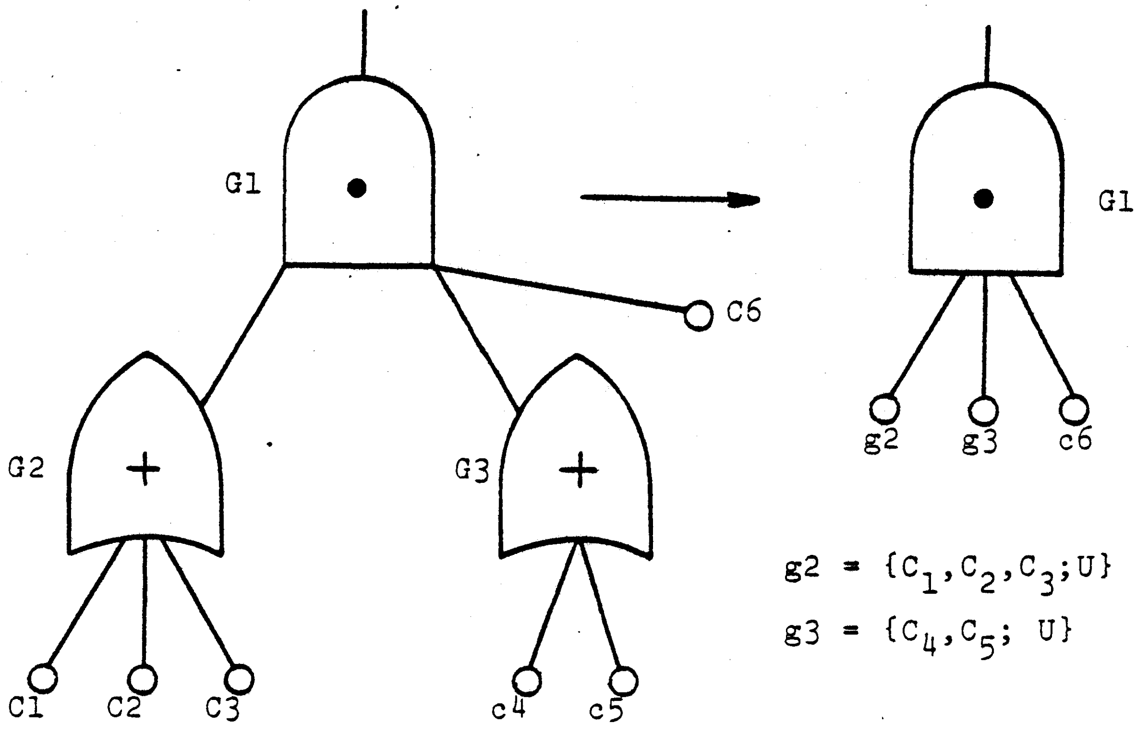
MODULAR MINIMAL CUT-SET REPRESENTATION



$$\underset{\sim}{Y}^B = (Y_{C1}, Y_{C2}, Y_{g2}, Y_{g3})$$
$$S_1 = (1, 1, 1, 0)$$
$$S_2 = (1, 0, 1, 1)$$
$$S_3 = (1, 1, 0, 1)$$
$$S_4 = (0, 1, 1, 1)$$

FIGURE 3.11

SYMMETRIC MODULARIZED GATE



$$g2 = \{C_1, C_2, C_3; U\}$$
$$g3 = \{C_4, C_5; U\}$$

FIGURE 3.12

MODULARIZED GATES AS PSEUDO-COMPONENTS

obtained. Therefore steps (e) through (h) will be successively applied to newly obtained sets of gateless nodes until the TOP tree event is reached, thus leading to a modularization of the whole tree.

Careful examination of the kinds of fault tree structural modifications needed to modularly decompose a fault tree, will lead to the conclusion that a quite involved logical procedure must be followed to accomplish this task. Therefore, in order to implement the modularization of fault trees by the computer program PL-MOD, it has been necessary to turn to a programming language capable of dynamically following the step-by-step structural changes effected by the modularization algorithm. In the following sections of the chapter, programming language PL-1, shall be shown to be particularly suited for this objective. Consequently the logical manipulations required to modularize fault trees will be illustrated throughout by the PL-1 statements contained in the PL-MOD code.

III.3. PL-1 Language Features Used for the Representation and Modularization of Fault Trees

III.3.1. Introduction

In Chapter I, it was discussed how the computer code PATREC [12] utilized a number of PL-1 language [11] tools for the analysis of non-replicated event fault trees by means of a pattern recognition technique. It was pointed out that its procedure relies on the recognition of sub-tree patterns within the fault tree which conform to known tree patterns stored in the the computer code library. Each recognized sub-tree portion is then replaced by a super-component with an occurrence probability which has been computed by PATREC. New sub-tree patterns are then recognized which include these super-components until ultimately the tree reduces to a single super-component with an occurrence probability equal to the overall system reliability.

The approach taken by PL-MOD is quite different in that its purpose is to obtain the full structural information for the fault tree. This information is needed to allow for a much more extensive analysis of the fault tree, rather than the sole evaluation of the overall system reliability.

III.3.2. Structure Variables

A structure in PL-1 is a hierarchical collection of related data items of different types.

In the computer code PL-MOD, a node is represented by a structure containing relevant information such as its NAME (chosen to be a number), its VALUE (a number which equals 1 for AND gates and 2 for OR gates), the number of gate inputs it contains = GIN, the number of non-replicated inputs it contains (called free leaves) = LIL, the number of replicated inputs it contains (called replicated leaves) = DIR, etc. Thus, the NODE structure has a declaration statement of the form

```
DECLARE  1  NODE
         2  NAME FIXED,
         2  VALUE FIXED,
         2  GIN FIXED
         2  LIL FIXED,
         2  DIR FIXED,
         2  etc.
```

III.3.3. <u>Pointers, Based and Controlled Variables</u>

PL/1 provides several facilities normally found only in assembler or in list-processing languages. The essence of list processing is the ability to dynamically allocate blocks of core storage, to link those blocks together into a structure, and to store and to retrieve data from the blocks. List processing for complicated data structures, such as those required by PL-MOD, are very difficult or impossible to achieve through manipulations of simple arrays.

Each individual block of list-processing storage is called a BASED VARIABLE and is usually defined as a data structure. Since several based variables with identical structures will in general exist at a time, a POINTER VARIABLE is required to point at a specific one.

Thus, in order to handle sets of similar NODE structures, it is necessary that they be declared as BASED variables

```
DECLARE 1  NODE BASED (NT),
        2  NAME FIXED,
        2  VALUE FIXED,
        2  GIN FIXED
        2  LIL FIXED,
        2  DIR FIXED,
        2  etc.
```

Each time a NODE structure needs to be created, an ALLOCATE statement is used (ALLOCATE NODE) with pointer variable NT automatically acquiring a different value for each NODE structure. This set of different NT pointer values may be then kept in an array of pointers SPINE (I) (I = 1,2, ...,GUM = total number of gates) for identification of each of the nodes in the tree.

The following statements allocate and identify a NODE associated with Gate I

```
ALLOCATE NODE;
SPINE (I) = NT;
```

After the node has been allocated, it will be possible to specifically refer to it through the qualified expression

```
SPINE (I)→NODE
```

Finally, whenever the NODE associated with Gate I is no longer needed, its storage space may be released by the statements

```
NT = SPINE (I);
FREE NODE;
```

Another type of variable used throughout PL-MOD is the

CONTROLLED variable. These variables are similar to BASED variables in that they can be dynamically allocated and released at any time by means of the ALLOCATE and FREE statements. Nevertheless, two or more CONTROLLED variables having the same name cannot coexist, since they are only identified by their name and no pointer exists which locates them in the computer memory.

III.3.4. The REFER Option for Based Variables

In Chapter I, it was mentioned that the computer code PATREC requires that fault trees be represented in binary gate form (Figure 3.13). As a result each NODE structure in PATREC requires the same amount of storage. In the approach taken by PL-MOD no restriction exists on the number of gates and component inputs that a NODE may have, and thus it is necessary that the NODE structures in PL-MOD be made of input arrays having a variable number of dimensions.

The REFER option for based structure variable can fulfill such a task as illustrated by the NODE example of Figure 3.14: AND Gate 7 consists of two gate inputs (8,9), three leaf inputs (3,5,7) and one replicated leaf input (r-leaf) (20001). Therefore, NODE.NAME = 7, NODE.VALUE = 1, NODE.GIN = 2, NODE.LIL = 3 and NODE.DIR = 1. Gate 7 is connected to its input gates by means of an array variable NODE.SPIT which stores the pointers corresponding to NODES 8 and 9 (i.e., SPINE (8) and SPINE (9)). NODE.SPIT is then a variably dimensioned array of pointers. Its dimension will be given by a variable (GINO) outside the NODE structure and its value shall be assigned to a

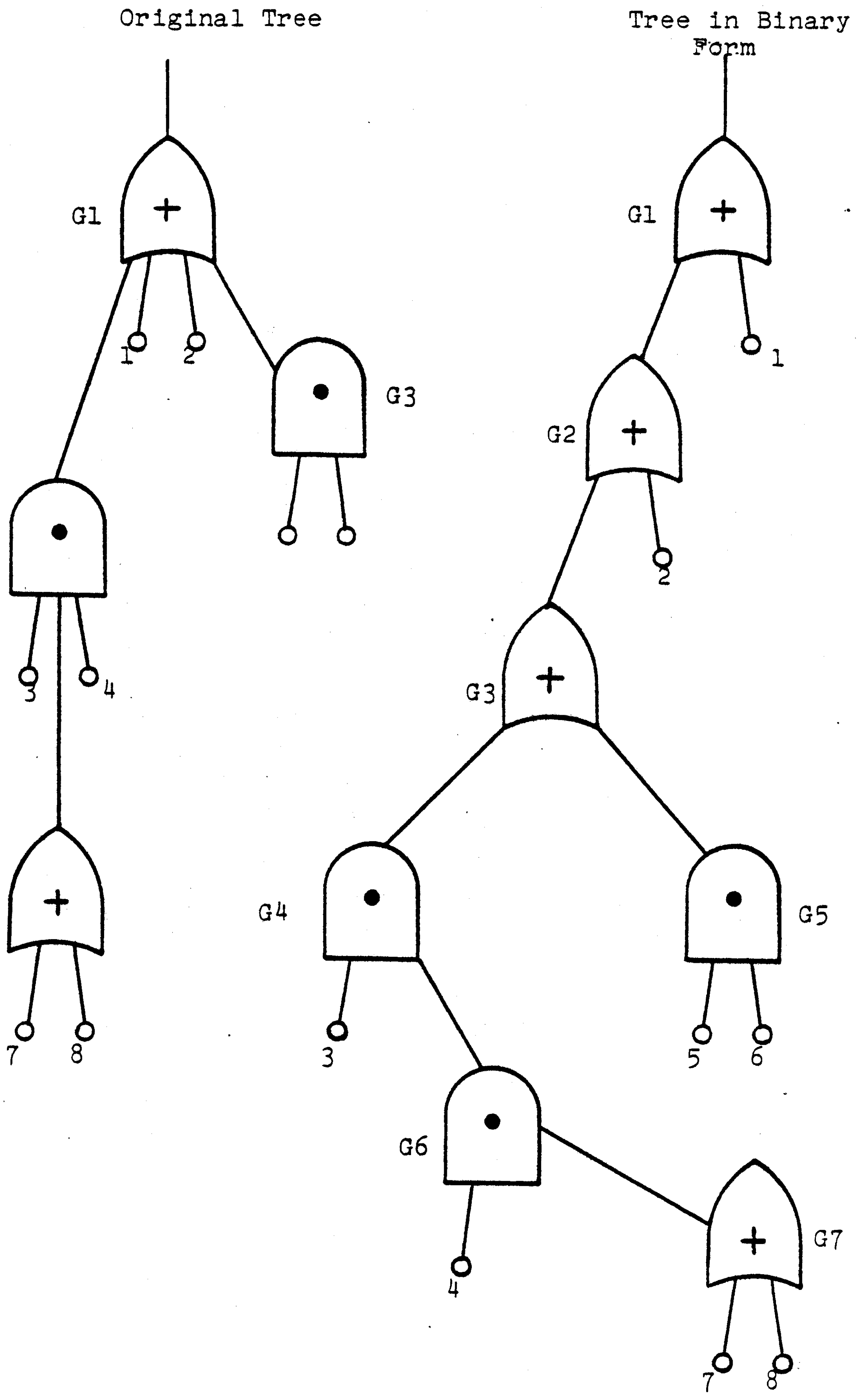


FIGURE 3.13 FAULT TREE IN BINARY GATE FORM

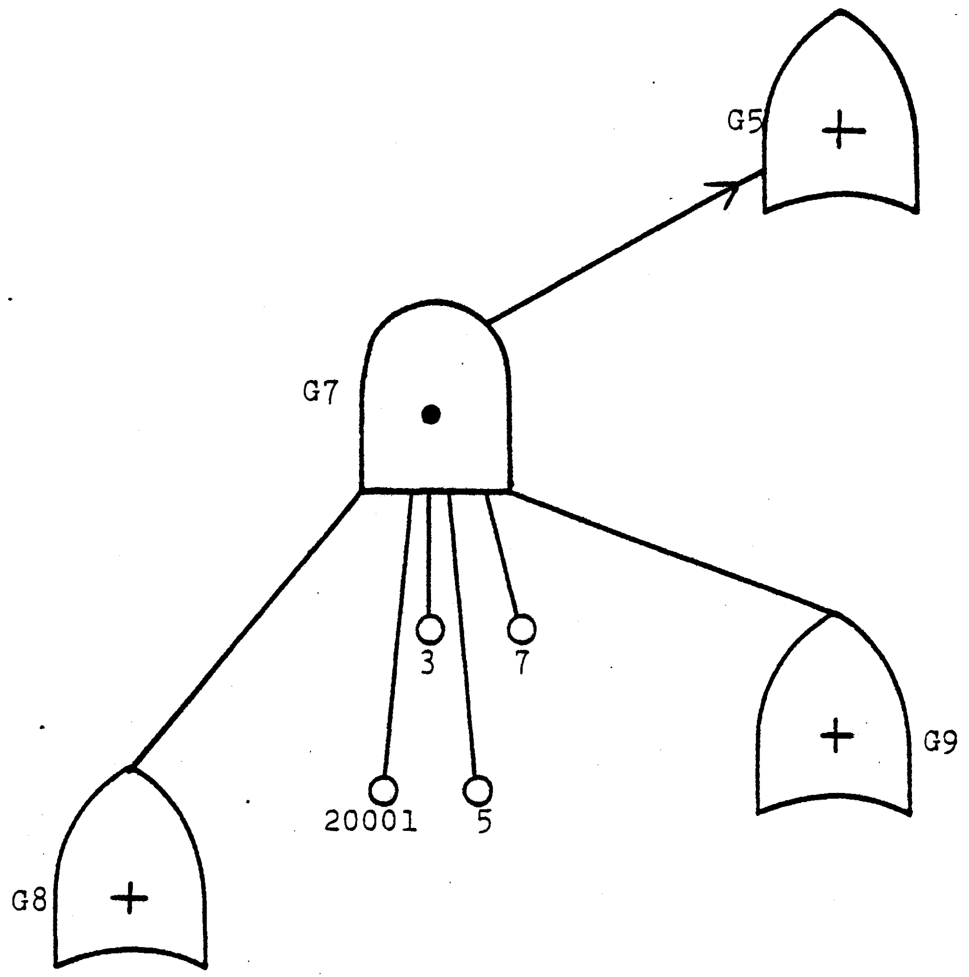


FIGURE 3.14

SAMPLE GATE NODE

NODE structure variable (NODE.GIN) as required by the PL/1 REFER option:

```
DECLARE 1  NODE BASED (NT),
        2  NAME FIXED,
        :
        2  GIN FIXED  BINARY,
        :
        2  SPIT (GINO REFER(NODE.GIN))POINTER,
        :
        etc.
(GINO = NODE.GIN)
```

In a similar way, the set of numerical values identifying the free leaf and r-leaf inputs of the NODE will be assigned to NODE.TIL(LILO REFER(NODE.LIL)) and NODE.TIR(LILO REFER (NODE.LIR)) respectively.

In addition, the pointer value locating the NODE for gate 5 will be assigned to structure variable NODE.ROOT.

The following statements allocate the required space and assign the desired set of inputs and output connection for NODE 7:

```
DECLARE 1  NODE BASED (NT),
        2  NAME FIXED,
        2  VALUE FIXED,
        2  GIN FIXED  BINARY,
        2  LIL FIXED  BINARY,
        2  DIR FIXED  BINARY,
        2  SPIT (GINO REFER (NODE.GIN))POINTER,
2 TIR (LIRO REFER (NODE.DIR))FIXED,
2  TIL (LILO REFER (NODE.LIL))FIXED:
:
:
GINO = 2;
LIRO = 1;
LILO = 3;
ALLOCATE NODE;
SPINE (7) = NT;
:
:
NT = SPINE (7);
NODE.TIL (1) = 3;
NODE.TIL (2) = 5;
NODE.TIL (3) = 7;
NODE.TIR (1) = 20001;
NODE.SPIT (1) = SPINE (8);
NODE.SPIT (2) = SPINE (9);
NODE.ROOT = SPINE (5);
```

III.3.5. Bit String Variables

In Chapter II, it was shown how prime modular gates may be represented by a set of Boolean state vectors each representing a cut-set member of the family of minimal cut-sets characterizing the module structure function.

Boolean vectors can be conveniently depicted in PL/1 by means of a string of BIT variables. A bit-string is simply a group of binary digits (0 or 1) enclosed in single quotes and followed by a B character (e.g., '01011'B).

A number of built-in functions and operations are provided in PL/1 for the effective handling and manipulation of bit-strings, as required by PL-MOD to generate a Boolean vector represenation for higher order modular gates. Thus, consider for example the following set of controlled bit variables

```
DECLARE TOD BIT(LARG) CONTROLLED;
DECLARE DOTT BIT (WEST) CONTROLLED;
DECLARE KOF BIT (JUST) CONTROLLED;
DECLARE KOD BIT (JUST) CONTROLLED;
DECLARE TOG BIT (JUST) CONTROLLED;
```

After these variables have been allocated with dimensions WEST = 3, LARG = 6 and JUST = LARG + WEST = 9, the following operations and funtions existing in PL/1 may be applied to them

```
Repeat function:
     KOD = REPEAT ('0'B, JUST) = KOD = '000000000'B
Substring pseudo-function
SUBSTR (KOD,LARG + 1,1) = '1'B = KOD '000000100'B
SUBSTR (KOF, NUB + 2,1) = '1'B = KOF = '000010000'B
Substring function:
DOTT = SUBSTR (KOD,LARG + 1, WEST) = DOTT = '100'B
INTERSECTION (&), Union (/) and complement (¬) oper-
   ations:
TOG = KOF & KOD =            TOG = '000000000'B
TOG = KOF /KOD =             TOG = '000010100'B
TOG = ¬ KOF =                TOG = '111101111'B
```

III.4. Definition and Organization of the Procedures Used in PL-MOD for the Modularization of Fault Trees

PL-MOD accomplishes the modularization of a fault tree by calling a number of procedures in the following order

```
CALL INITIAL;
CALL TREE-IN;
FLAG = 1;
DO WHILE (FLAG ¬= 0);
CALL COALESCE;
CALL MODULA;
END;
```

Internal procedures TRAVEL and TRAPEL are called by procedures COALESCE and MODULA, while internal procedure BOOLEAN is only called by MODULA.

The task performed by each of these procedures is defined below.

INITIAL: This procedure allocates the necessary storage space for each of the nodes in the fault tree (including NODE space for replicated module sub-trees).

TREE-IN: Attaches to each NODE its corresponding set of gate and component inputs, interconnects interdependent gates having common replicated inputs and assigns to each NODE its output gate defined to be its NODE.ROOT.

COALESCE: Collapses simple gateless NODES with their NODE.ROOT gates if they are of the same type.

MODULA: (a) Transforms simple gateless NODES having no replicated inputs into modular super-components and attaches them as inputs to their NODE.ROOT gate.

(b) Transforms simple gateless NODES having replicated inputs into temporary NESTED modules, unless the gate is the top event for a complete set of replicated events (i.e., a parent gate) in which case by calling BOOLEAN it modularizes the full set of NESTED modules into a higher order module whose inputs are the set of replicated events and a new set of proper modules in place of the temporary NESTED module set.

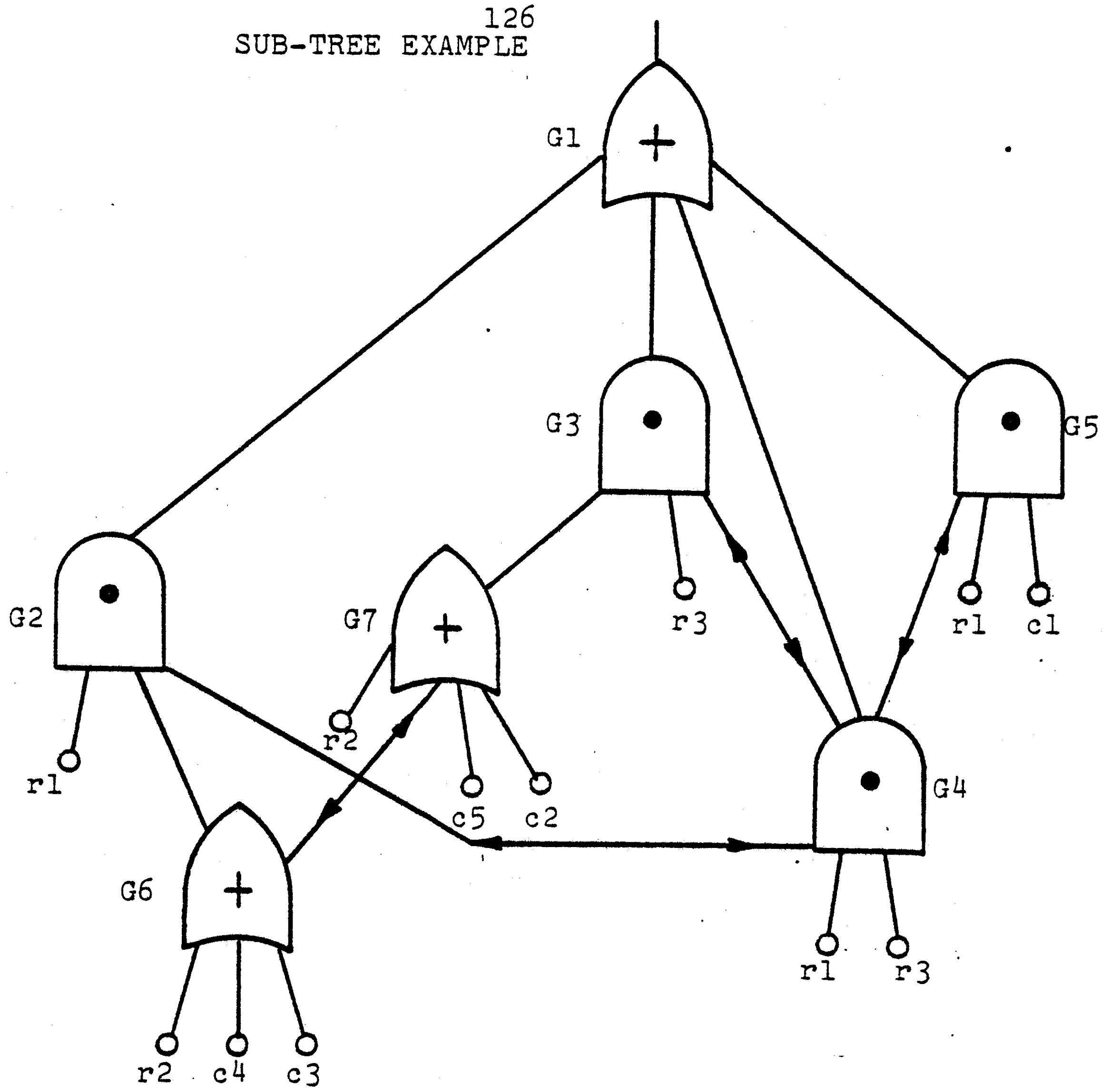
(c) Modularizes symmetric K-out of-n gates explicitly included in the fault tree.

Procedures COALESCE and MODULA are sequentially called one after the other until the TOP tree event is reached, at which time the complete fault tree will have been modularized.

TRAVEL and TRAPEL: As mentioned before, interdependent gate NODES are interconnected to insure that only proper modules are generated (Figure 3.15). Each interdependent gate will in general have two interconnections leading to other interdependent gates (e.g., $NAIL_{G4}$ and $WHIP_{G4}$ due to replicated component r_1) for each replicated input it contains (these interconnections are given the names NODE.WHIP and NODE. NAIL).

Particular care must be taken that these interconnections be kept each time the fault tree structure undergoes a transformation enacted by the COALESCE and MODULA procedures. Thus, whenever COALESCE collapses a simple gate containing replicated inputs with its NODE.ROOT gate, its WHIP and NAIL

SUB-TREE EXAMPLE



Connections

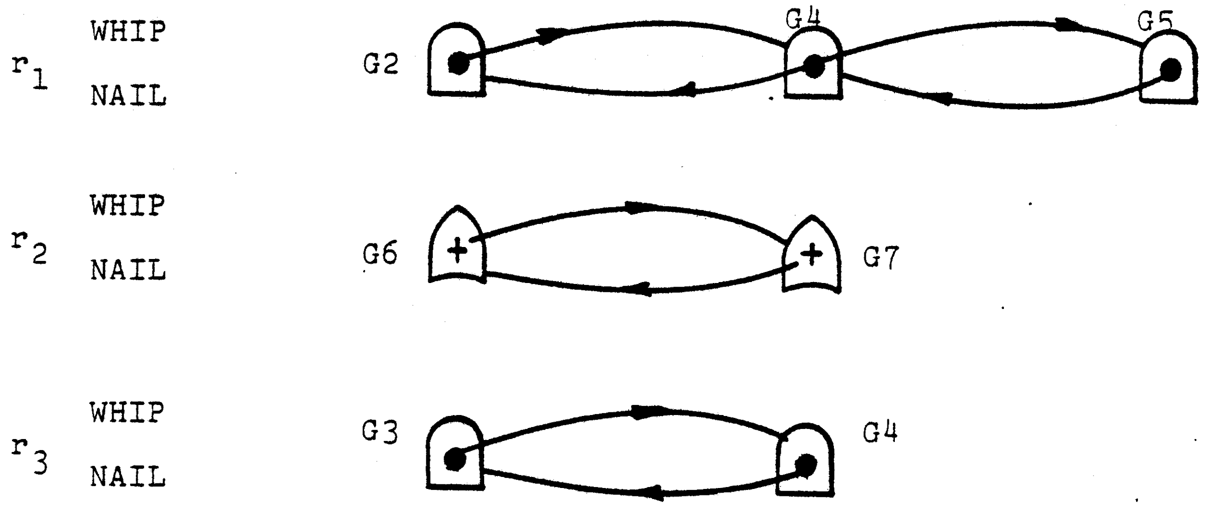
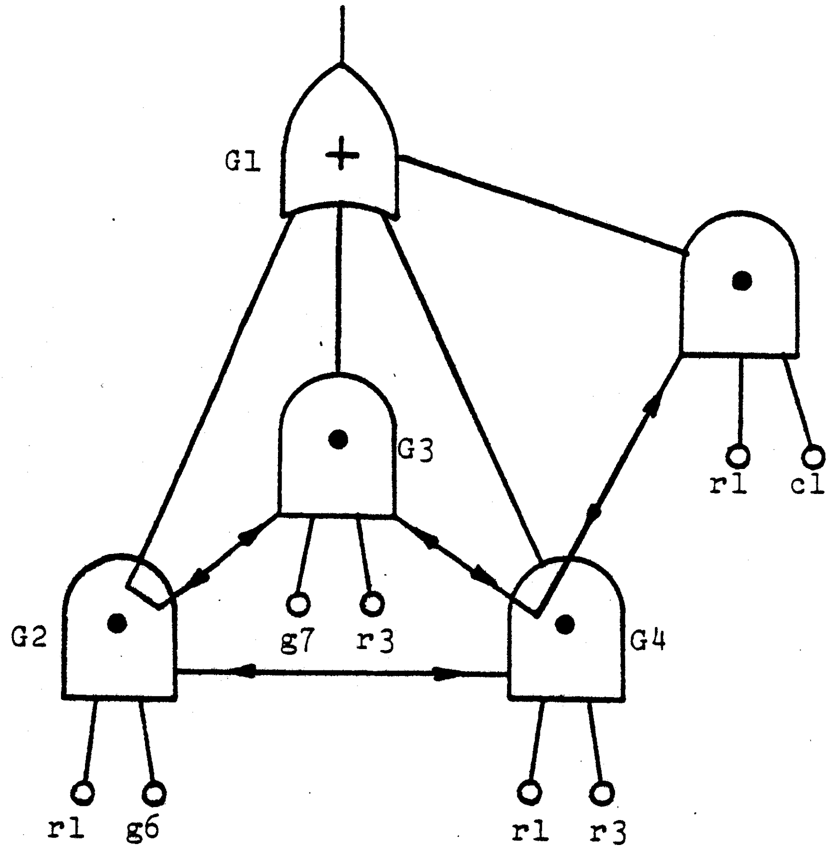


FIGURE 3.15

INTERDEPENDENT GATE INTERCONNECTIONS

SUB-TREE EXAMPLE



Nested modules $g6 = \{r2,c3,c4;U\}$

$g7 = \{r2,c2,c5;U\}$

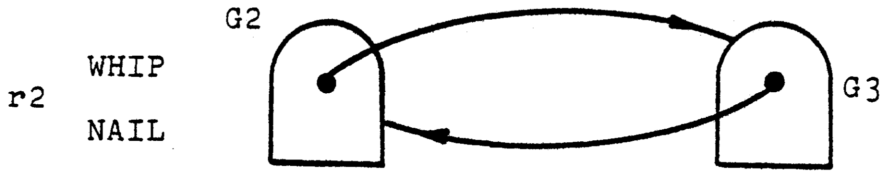
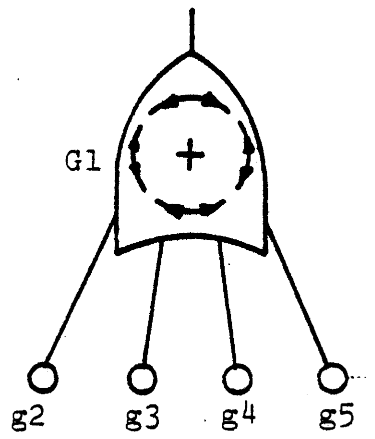


FIGURE 3.16

TRANSFER OF GATE INTERCONNECTIONS

SUB-TREE EXAMPLE



nested modules

$g2 = \{r1,\ g6;\Omega\}$

$g3 = \{r3, g7;\Omega\}$

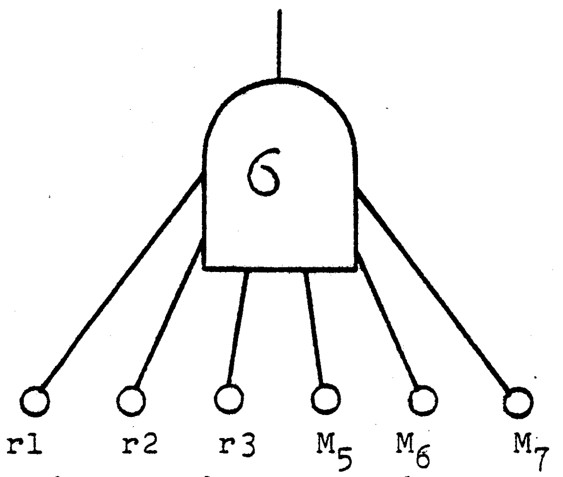
$g4 = \{r2, r3;\Omega\}$

$g5 = \{r1, c1;\Omega\}$

FIGURE 3.17

INTERNAL GATE INTERCONNECTIONS

SUB-TREE EXAMPLE



$M_5 = \{C1\}$, $M_6 = \{C_3, C_4; U\}$, $M_7 = \{C_2, C_5 ; U\}$

$\underset{\sim}{Y}^B = (Y_{r1}, Y_{r2}, Y_{r3}, Y_{M5}, Y_{M6}, Y_{M7})$

$S_1 = (1, 1, 0, 0, 0, 0)$

$S_2 = (1, 0, 0, 0, 1, 0)$

$S_3 = (1, 0, 1, 0, 0, 0)$

$S_4 = (1, 0, 0, 1, 0, 0)$

$S_5 = (0, 1, 1, 0, 0, 0)$

$S_6 = (0, 0, 1, 0, 0, 1)$

FIGURE 3.18

BOOLEAN VECTOR REPRESENTATION

interconnections must be transferred to the NODE.ROOT gate. Similarly when a gate with replicated inputs is temporarily transformed into a nested module input attached to its NODE. ROOT gate, its WHIP and NAIL connections must also be transferred (Figure 3.16).

Procedures TRAVEL and TRAPEL help perform this task. TRAVEL insures that NODES attached by means of a NAIL interconnection to another NODE which is to be absorbed by its NODE.ROOT gate in a COALESCE or MODULA step, are interconnected by a NAIL interconnection to the NODE.ROOT gate. Similarly, TRAPEL provides for the transfer of WHIP interconnections of NODES attached to a NODE which is collapsed or modularized by a COALESCE or MODULA step.

Notice that a set of nested modules will be complete, and thus representable by a higher order module, when a gate has been reached such that all its NAIL and WHIP interconnections are internal to the gate (Figure 3.17).

BOOLEAN: Yields a minimal cut-set representation in Boolean vector form for higher order modules.

Each state component in the Boolean vector corresponds to either a replicated event in the domain of the set of nested modules or a proper module derived out of one of the nested modules (Figure 3.18).

III.5. <u>The Pressure Tank Rupture Fault Tree Example</u>

The operation of each of the procedures in PL-MOD will be discussed in detail in the following sections of this chapter.

In order to clarify the discussion, at each step reference is made to a slightly modified version of the familiar pressure tank example due to Haasl [1]. The diagram of the system is given in Figure 3.19.

A hazard associated with the operation of the pressure tank system is the occurrence of a rupture of the pressure tank. Figure 3.20 is a fault tree showing the series of events leading to a pressure tank rupture.

The system is designed such that gas will start to be pumped into the pressure tank if the push-button switch S1 is actuated. This causes a flow of current in the control circuit of the system and thus activates relay coil K2. Relay contacts K2 will then close causing the pump motor to start. After about 20 seconds, the pressure switch contacts will open given an excess pressure has been detected by a 2-out of-3 pressure switch device. Contacts K2 will then open, shutting off the motor as soon as the K2 coils have been de-energized due to a lack of current in the control circuit. For additional safety, in case of a pressure switch malfunction, a timer relay is set to open the circuit after 60 seconds thus shutting off the pump motor.

In the fault tree shown, a common cause failure event among the control circuit devices has been assumed to be the main contribution to the secondary failure of each of the control circuit components, i.e., K1, K2 and T. Table 3.1 is a list of all the basic fault event inputs and of their occurrence probability.

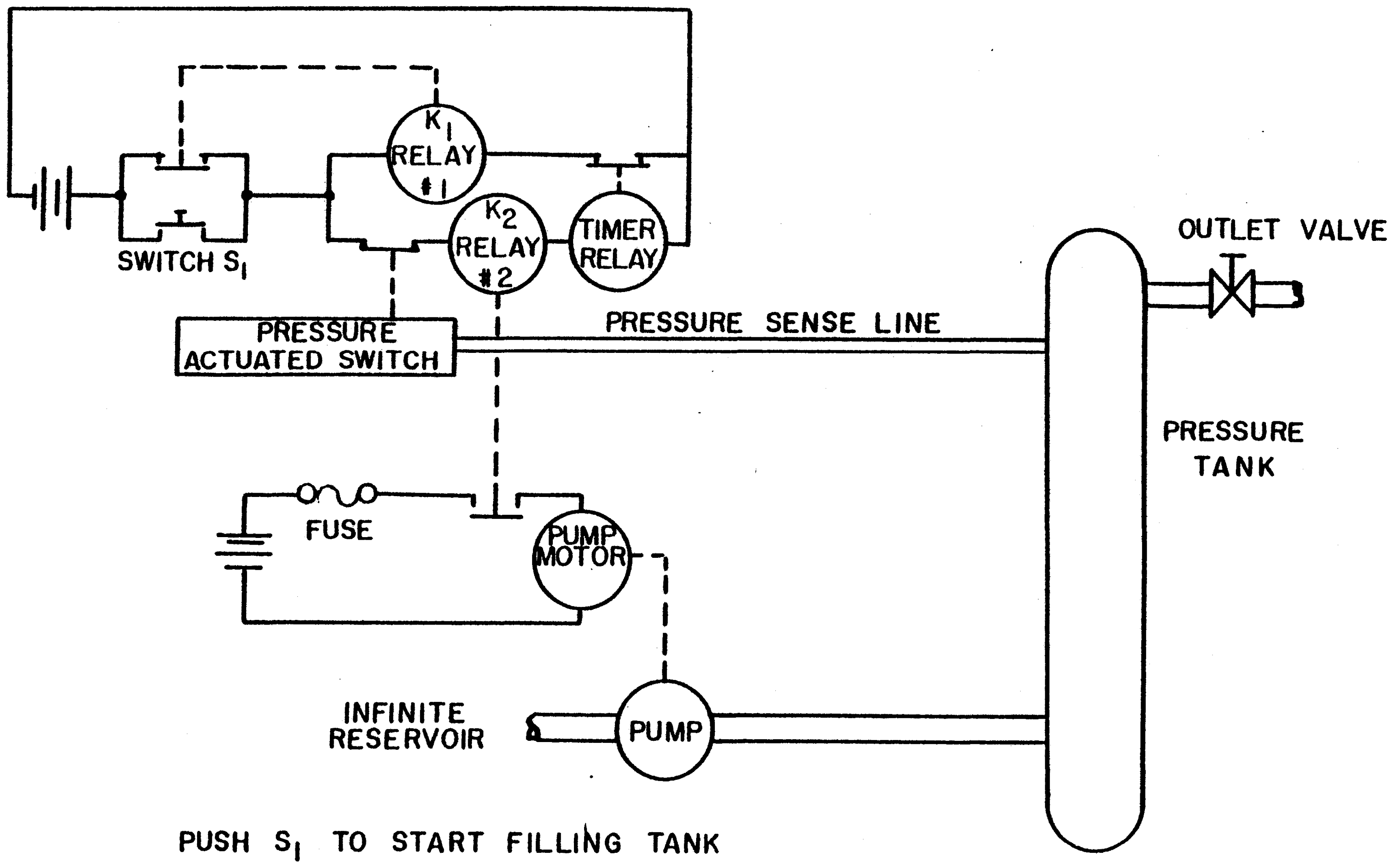


FIGURE 3.19 PRESSURE TANK EXAMPLE

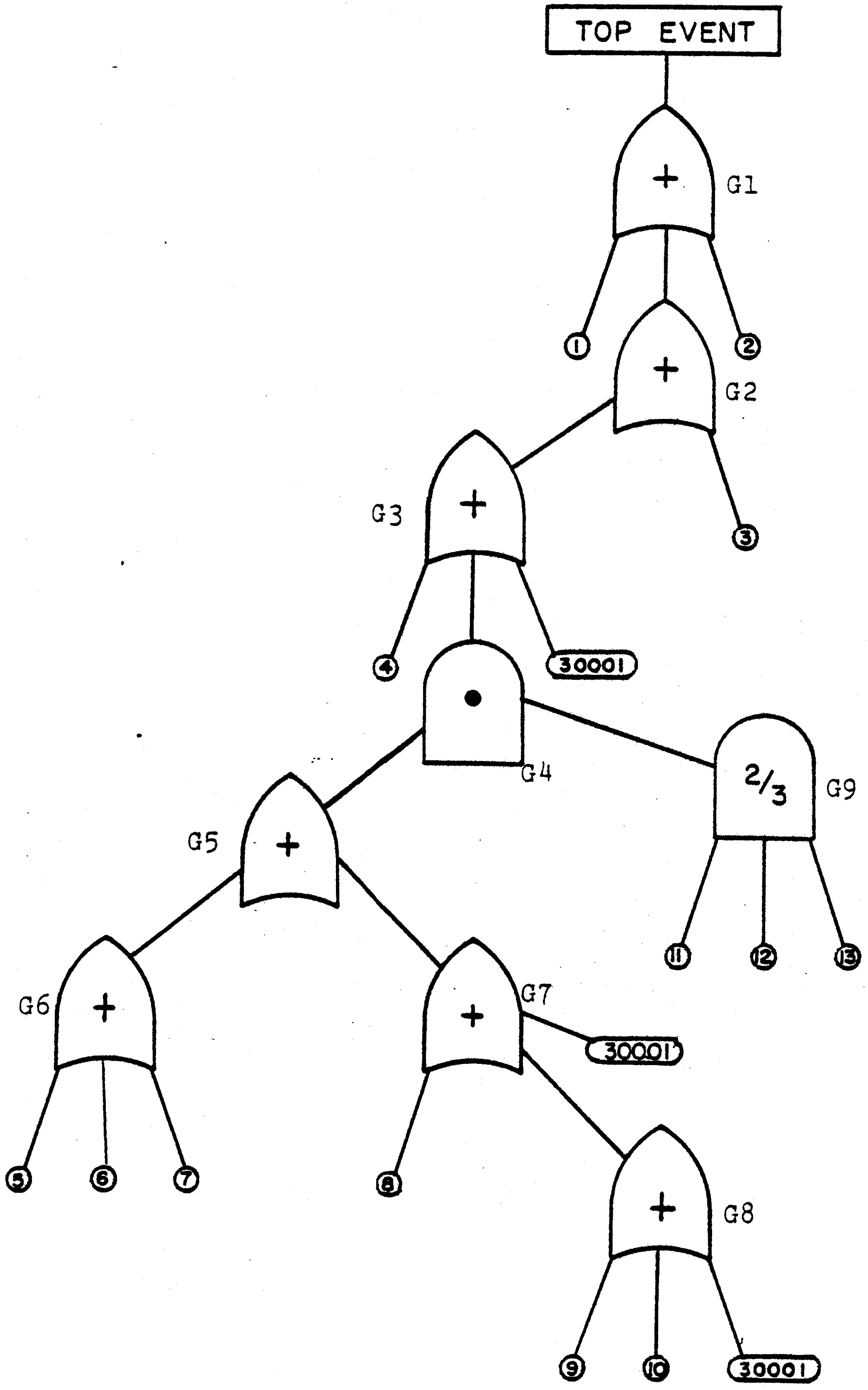


FIGURE 3.20   PRESSURE TANK RUPTURE FAULT TREE

TABLE 3.1

PRESSURE TANK RUPTURE FAULT TREE FAILURE PROBABILITY DATA

Basic Event i	Event Description	Failure Rate (Per Loading Cycle)
1	Pressure Tank Faulure	10-8
2	Secondary failure of Pressure Tank Due to Improper Selection	10-5
3	Secondary failure of Pressure Tank Due to out-of-tolerance conditions	10-5
4	K2 relay contacts fail to open	10-5
5	S1 switch secondary failure	10-5
6	S1 switch contacts fail to open	10-5
7	External reset actuation force remains on switch S1	10-5
8	K1 relay contacts fail to open	10-5
9	Timer does not "time-off" due to improper setting	10-5
10	Timer relay contacts fail to open	10-5
11	Pressure switch not actuated by sensor 1	10-5
12	Pressure switch not actuated by sensor 2	10-5
13	Pressure switch not actuated by sensor 3	10-5

Replicated Event i	Event Description	Failure Rate (Per Loading Cycle)
(3000)1	Common Cause failure among relays K_1,K_2 and timer T	10-5

III.6. INITIAL and TREE-IN

INITIAL: The INITIAL procedure allocates the necessary storage for each of the NODES making up the fault tree. The value of GUM = total number of gates in the fault tree, is read in and arrays

```
SPINE(GUM) POINTER CONTROLLED;
AGIN(GUM) FIXED CONTROLLED;
ALIL(GUM) FIXED CONTROLLED;
ALIR(GUM) FIXED CONTROLLED;
BOST(GUM) POINTER CONTROLLED;
```

are allocated.

Array SPINE is used to store the pointer values (NT) locating each NODE based structure. This allows that each of the different NODE structures allocated be assigned the set of input data corresponding to the gate they represent.

Arrays AGIN, ALIL and ALIR are used to store the number of gate, free leaf and replicated leaf inputs each node contains. Thus for the pressure tank example (Figure 3.20).

```
AGIN(1) = 1, ALIL(1) = 2, ALIR (1) = 0,
AGIN(2) = 1, ALIL (2) = 1, ALIR(2) = 0,
AGIN(3) = 1, ALIL(3) = 1, ALIL(3) = 1,
```

etc.

Finally, array BOST(GUM) will store the pointers locating each of the proper modules to be created by PL-MOD (clearly the number of modules to be found in a fault tree will be less than the number of gates (GUM) in the tree)

A DO loop group follows

```
DO I = 1 to GUM;
GET LIST (I, AGIN(I), ALIL (I), ALIR (I));
:
:
ZEN: ALLOCATE NODE;
:
:
SPINE (I) = NT,
END;
```

which allocates the space needed by each node given the number of gate, leaf and r-leaf inputs it contains. In addition each array variable is initialized to be zero or NULL depending on whether the variable is a number (FIXED) or a pointer and the pointer NT_1 associated with the NODE representing gate I (I = 1,2,...GUM), is assigned to SPINE (I) for later reference.

The value of NOR = the number of dependent components is read in and arrays

```
SPRING (NOR) POINTER CONTROLLED;
F (NOR) FIXED CONTROLLED;
```

are allocated. SPRING(K) (K = 1,2,...,NOR) will later be used in TREE-IN to attach the NODE.WHIP and NODE.NAIL interconnections among interdependent gates having common replicated component K as input. The numerical variable F(K) is initialized to be zero and is later increased by one in TREE-IN, each time replicated component K is read in as an input to some gate in the fault tree.

TREE-IN: Once each NODE has been allocated by INITIAL, TREE-IN proceeds to assign initial values to each NODE varible as inferred from the node input data NODE IN which is read in. In addition, TREE-IN finds the initial set of "gateless" nodes which are to be processed by the set of procedures COALESCE and MODULA.

The full NODE structure is composed of the following variables

```
1  NODE BASED (NT),
2  TIPO FIXED
2  NAME FIXED,
2  VALUE FIXED,
2  GINT FIXED,
2  LILT FIXED,
2  LIRT FIXED,
2  LIMD FIXED,
2  LIMT FIXED,
2  NEST FIXED,
2  WHIZ FIXED,
2  ROOT POINTER,
2  LIP POINTER,
2  LID POINTER,
2  GIN FIXED BINARY,
2  LIL FIXED BINARY,
2  DIR FIXED BINARY,
2  NAIL(LIRO REFER (NODE.DIR)) POINTER
2  WHIP (LIRO REFER (NODE.DIR)) POINTER
2  TIR (LIRO REFER (NODE.DIR)) FIXED,
2  SPIT (GINO REFER (NODE.GIN)) POINTER
2  TIL (LILO REFER (NODE.LIL)) FIXED:
```

In Section III.3.4., variables NAME, VALUE, ROOT, GIN, LIL, DIR,TIR,SPIT and TIL have already been defined. As explained in section III.4., variables NAIL and WHIP are the arrays of pointers used for interconnecting NODES having common replicated events.

The methodology employed by PL-MOD to modularize a complete fault tree consists of piecewise collapsing and modularizing portions of the tree. As a consequence, at the intermediate stages of the modularization procedure some nodes are taken away from the tree while others undergo changes in the type and number of inputs they have. For this purpose, a number of variables need to be added to the NODE structure. Thus NODE.LIP is a pointer variable used to add on to the node a set of free leaf and r-leaf inputs which have been collapsed into the node. These additions to the NODE are done by means of based structure variables STIP.

NODE.LID is a pointer variable used to add on to the node free and nested module structures. . These additions are done through based structure variables STID.

NODE.GINT equals the total number of gate inputs to the node. Initially NODE.GINT = NODE.GIN, however, as each of the gate inputs is either collapsed or modularized to the node, NODE.GINT is reduced by one until it eventually equals zero (i.e., the node has become gateless).

NODE.LILT equals the total number of free leaf inputs to the node (initially NODE.LILT = NODE.LIL).

NODE.LIRT equals the total number of replicated inputs to the node (initially NODE.LIRT = NODE.LIR).

NODE.LIMD measures the number of nested modules directly attached as modular inputs to the node.

NODE.NEST measures the total number of nested modules in the domain of the node gate, these nested modules are therefore directly or indirectly connected to the node.

NODE.LIMT measures the total number of free modules attached as inputs to the node.

NODE.WHIZ is an index used by TREE-IN to keep track of the WHIP interconnections that are being attached to the node as the NODE IN data for each of the gates in the trée is read in.

NODE.TIPO equals 1 for every node in the tree. Its purpose is to distinguish NODE structures from other structures which are involved in the TRAVEL and TRAPEL procedures (thus STIP.TIPO - 2, STID.TIPE = 3, MOD.TIPO = 4, AP,TIPO = 0).

The set of statements making up TREE-IN are

```
                   /*       TRFE_IN       */
 1  0             TREE_IN: PROC;
 2  0                ALLOCATE ELM(GUM);
 2  0                J=1;
 2  0                DO I=1 TO GUM;
 2  1                GINO=AGIN(I);
 2  1                LIRO=ALIR(I);
 2  1                LILO=ALIL(I);
 2  1                ALLOCATE NODEIN;
 2  1                GET LIST(NODEIN);
```

```
2  1      PUT EDIT ('NODE=',NODEIN.NAME) (SKIP(2),A(5),F(5))
          ('VALUE=',NODEIN.VALUE) (X(2),A(6),F(5))
           ('GATE INPUTS=') (X(2),A(12));
2  1      PUT LIST( NODEIN.PIT);
2  1      PUT EDIT('FREE LEAF INPUTS=') (X(2),A(17));
2  1      PUT LIST (NODEIN.QTIL);
2  1      PUT EDIT ('DEP LEAF INPUTS=') (X(2),A(16));
2  1            PUT LIST(NODEIN.QTIR);
2  1            NT=SPINE(NODEIN.NAME);
2  1            NODE.NAME=NODEIN.NAME;
2  1             NODE.VALUE=NODEIN.VALUE;
2  1             NODE.TIL=NODEIN.QTIL;
2  1            NODE.LILT=NODEIN.LILI;
2  1             NODE.TIP=NODEIN.QTIR;
2  1            NODE.LIRT=NODEIN.LIRI;
2  1            IF(NODE.LIRT=0) THEN GO TO LOCA;
2  1            DO LA=1 TO LIRO;
2  2            MA=NODE.TIR(LA);
2  2            DA=-CEIL(-MA/10000);
2  2            JA=-CEIL(-MA/1000);
2  2            JAK=JA-10*DA;
2  2            NA=MA-(1000)*JA;
2  2            F(NA)=F(NA)+1;
2  2            IF (F(NA)¬=1) THEN GO TO LOCE;
2  2            ELSE NODE.NAIL(LA)=NT;
2  2            SPRING(NA)=NT;
2  2            GO TO LOCO;
2  2      LOCE: NODE.NAIL(LA)=SPRING(NA);
2  2            ARI=NT;
2  2            IF(F(NA)¬=DA) THEN GO TO AMP;
2  2            IF(JAK¬=9) THEN GO TO LUXE;
2  2            DO IX=1  TO RMOP;
2  3            IF(TRIM(IX)=MA) THEN GO TO LUCE;
2  3            END;
2  2      LUCE:      ALLOCATE AP;
2  2            PRIN(IX)=APT;
2  2            AP.SPIT=PRIM(IX);
2  2            PRIM(IX)->NODE.ROOT=APT;
2  2             GO TO LUCI;
2  2      LUXE:    ALLOCATE AP;
2  2            AP.SPIT=NULL;
2  2       LUCI:       ZA=NODE.WHIZ+1;
2  2            NODE.WHIP(ZA)=APT;
2  2            NODE.WHIZ=ZA;
2  2             IF(JAK=1|JAK=2) THEN AP.REP=-DA;
2  2            ELSE AP.REP=DA;
2  2            AP.TIPO=0;
2  2            AP.VALUE=0;
2  2            AP.NAP=MA;
```

```
PL/I OPTIMIZING COMPILER                /*    MODULE     PROGRAM      */

     STMT LEV NT

      224    2  2       PUT EDIT('DEP COMP=', AP.NAP, 'APPEARANCES=', AP.REP)
                       (SKIP(2),X(2),A(9),F(5),X(2),A(12),F(5));
      225    2  2      AMP: NT=SPRING(NA);
      226    2  2           ZA=NODE.WHIZ+1;
      227    2  2            NODE.WHIP(ZA)=ARI;
      228    2  2           NODE.WHIZ=ZA;
      229    2  2           SPRING(NA)=ARI;
      230    2  2           NT=ARI;
      231    2  2      LOCO: END;
      232    2  1      LOCA: NODE.GINT=NODEIN.GID;
      233    2  1            IF(NODE.GINT=0) THEN GO TO BOTTOM;
      234    2  1           DO L=1 TO GINO;
      235    2  2           NODE.SPIT(L)=SPINE(NODEIN.PIT(L));
      236    2  2           AT=NODE.SPIT(L);
      237    2  2           AT->NODE.ROOT=NT;
      238    2  2           END;
      239    2  1           GO TO BOTE;
      240    2  1      BOTTOM: ELM(J)=NT;
      241    2  1           J=J+1;
      242    2  1      BOTE: FREE NODEIN;
      243    2  1           END;
      244    2  0           BUM=J-1;
      245    2  0           ALLOCATE OLM(BUM);
      246    2  0           DO K=1 TO BUM;
      247    2  1           OLM(K)=ELM(K);
      248    2  1           END;
      249    2  0           FREE ELM;
      250    2  0           FREE AGIN;
      251    2  0           FREE ALIL;
      252    2  0           FREE ALIR;
      253    2  0           FREE SPINE;
      254    2  0           FREE SPRING;
      255    2  0           RETURN;
      256    2  0            END  TREE_IN;
```

In anticipation of the set of initial gateless nodes to be found by TREE-IN, controlled pointer array variable ELM(GUM) is allocated (clearly the number of initial gateless nodes in the tree BUM is less than GUM) to store the locations of each gateless node.

The set of values associated with each node are read in by means of the controlled structure variable NODEIN.

```
1 NODEIN CONTROLLED,
2 NAME FIXED,
2 VALUE FIXED,
2 GID FIXED,
2 PIT (GINO)FIXED
2 LILI FIXED,
2 QTIL (LILO) FIXED,
2 LIRI FIXED,
2 QTIR (LIRO) FIXED;
```

Thus, for our pressure tank example, the first NODEIN values read from the input are

```
1 NODEIN,
2 NAME = 1,
2 VALUE = 2,
2 GID = 1              (GID = NODE.GIN)
2 PIT(1) = 2
2 LILI = 2             (LILI = NODE.LIL
2 QTIL(1) = 1, QTIL(2) = 2,
2 LIRI = 1             (LIRI = NODE.LIR)
2 QTIR(LIRO) = 0;
```

and they are passed on to the node whose pointer NT satisfies NT = SPINE (NODEIN NAME). Thus a correspondence exists between

NT_1 = SPINE(1) and NODEIN.NAME = 1

NT_2 = SPINE(2) and NODEIN.NAME = 2

etc.

Those nodes having replicated events (i.e., NODE.LIRT $\neq$ 0) are processed by an internal loop (DO LA = 1 to LIRO;) which sets up the interconnections among interdependent nodes.

Replicated components are identified by means of a five digit number (Table 3.2). The three lower digits are reserved for numbering (this convention allows for a total of 999 replicated events. The next digit will be zero unless the event represents a replicated module (in which case it equals nine) or if the replicated component is operated by a NOT gate somewhere in the tree (ON and OFF states are then distinguished by a 1 or 2 value for the fourth digit.*

Finally, the last digit denotes the total number of times the replicated component appears in the tree.

		NOMENCLATURE
SIMPLE REPLICATED COMPONENT		AOBCD
REPLICATED MODULE		A9BCD
DUAL REPLICATED COMPONENT	ON	A1BCD
	OFF	A2BCD

(A = Total number of appearances)

Table 3.2 Replicated Event Nomenclature

* Replicated modules and dual state replicated components are discussed in Sections III.11 and III.12.

Each time a replicated component is found in a new $NODE_a$ it is connected to the previous $NODE_b$, containing the same replicated component by a NAIL pointer (i.e., $NODE_a$ - NAIL=NT_b), while the previous $NODE_b$ is connected to the new $NODE_a$ with a WHIP pointer (i.e., $NODE_b$.WHIP = NT_a). At the same time, variable F(K) is increased by one each time replicated component K is found in a NODE (K = 1,2,...,NOR). When F(K) equals the total number of appearances for r - leaf K, a structure variable AP is allocated

```
1 AP BASED (APT),
2 TIPO FIXED,
2 NAP FIXED,
2 VALUE FIXED,
2 REP FIXED,
2 SPIT POINTER
```

and is interconnected by means of a WHIP pointer to the last node including replicated event K.

The variables making up the AP structure have the following definitions: AP.TIPO = 0 and AP.VALUE = 0 for every AP stru-structure, AP.NAP = replicated input name, AP.REP = number of appearances in the fault tree for the replicated input, AP.SPIT = NULL for all AP structures except those associated with a replicated module input (See Section III.11).

For the pressure tank example the following NAIL and WHIP interconnections exist (Figure 3.20).

```
1 NODE BASED (NT = SPINE(3)),
2 TIPO = 1,
2 NAME = 3,
2 VALUE = 2,
.
.
2 DIR = 1,
2 NAIL(1) = SPINE(3),
2 WHIP(1) = SPINE(7),
.
.
```

```
1 NODE BASED (NT = SPINE(7))
2 TIPO = 1
2 NAME = 7,
2 VALUE = 2,
.
.
2 DIR = 1,
2 NAIL(1) = SPINE(3)
2 WHIP(1) = SPINE(8)
.
.
```

```
1 NODE BASED (NT = SPINE(8)),
2 TIPO = 1,
2 NAME = 8
2 VALUE = 2,
.
.
2 DIR = 1,
2 NAIL(1) = SPINE(7),
2 WHIP(1) = APT_1,
.
.
```

```
1 AP BASED (APT_1)
2 TIPO = 0
2 NAP = 30001,
2 VALUE = 0
2 REP = 3,
2 SPIT = NULL;
```

Notice that the node with the first r-leaf appearance is "self-nailed" and that the node with the last r-leaf appearance has a whip interconnection to the AP structure corresponding to the particular replicated leaf. This last interconnection is needed later by BOOLEAN in order to set up a Boolean vector representation which includes the required r-leaf inputs.

Following the loop for the node interconnections, TREE-IN proceeds to attach gate inputs and root connections to each node with the statements

```
IF (NODE.GINT = 0) THEN GO TO BOTTOM;
DO L = 1 TO GINO;
NODE.SPIT(L) = SPINE(NODEIN.PIT(L)0;
AT = NODE.SPIT(L);
AT→NODE.ROOT = NT;
END;
(AT is a pointer variable)
```

Thus, for the pressure tank example, the following connections would be established:

```
1 NODE BASED (NT = SPINE (1)),
2 TIPO = 1,
2 NAME = 1,
:
2 ROOT = NULL,
2 GIN = 1,
:
2 SPIT(1) = SPINE(2),
:
-----------------------------------------
1 NODE BASED (NT = SPINE (2)),
2 TIPO = 1,
2 NAME = 2,
2 ROOT = SPINE (1),
2 GIN = 1,
:
2 SPIT(1) = SPINE(3),

-----------------------------------------
1 NODE BASED (NT = SPINE (3)),
2 TIPO = 1,
2 NAME = 3,
:
2 ROOT = SPINE(2),
:
2 GIN = 1,
:
2 SPIT(1) = SPINE(4),
:
-----------------------------------------
1 NODE BASED (NT = SPINE (4)),
2 TIPO = 1,
2 NAME = 4
```

```
2 VALUE = 1,
:
:
2 ROOT = SPINE(3)
:
:
2 GIN = 2,
:
:
2 SPIT(1) = SPINE(5), SPIT(2) = SPINE(9),
:
:
```

etc.

At the same time the pointers locating all gateless nodes (i.e., NODE.GINT = 0) are singled out for storage in array ELM

```
BOTTOM:  ELM(J) = NT;
            J   = J + 1;
  BOTE:  FREE NODEIN;
   END;
```

And at the end of TREE-IN's main external loop (DO I = 1 TO GUM), all these pointers are transferred to pointer array OLM(BUM).

For the pressure tank example 3 gateless nodes are initially found, i.e.,

```
BUM = 3;
OLM(1)  = SPINE(6);
OLM(2)  = SPINE(8);
OLM(3)  = SPINE(9);
```

Finally those controlled variables no longer needed for the rest of the program are released

```
FREE ELM;
FREE AGIN;
FREE ALIL;
FREE ALIR;
FREE SPINE;
FREE SPRING:
```

This storage saving capability of PL/1 is used throughout the procedures of PL-MOD.

III.7 COALESCE

Inspection of the pressure tank fault tree example indicates that gates (G6, G7, G8) can be collapsed together with gate G5.

The COALESCE procedure, given by the following statements, will be shown to perform this task by successively allocating STIP structures and connecting them to the node corresponding to gate G5.

```
                /*     COALESCE       */
          COALESCE:PROC;
                BUD=BUM;
                ALLOCATE OLD(BUD);
                DO K=1 TO BUD;
                OLD(K)=OLM(K);
                END;
                FREE OLM;
                 M=1;
                ALLOCATE GOLM(GUM);
          LOOP_1:     JO=1;
                ALLOCATE ELD(BUD);
           LOOP_2:    DO  I=1 TO BUD;
                CAT=OLD(I);
                DOG=CAT->NODE.ROOT;
                IF (DOG=NULL) THEN GO TO SKIP;
                IF (DOG->NODE.VALUE¬=CAT->NODE.VALUE) THEN GO TO SKIP;
                SEARCH=DOG->NODE.LIP;
                IF (SEARCH=NULL) THEN AJAX=1;
                ELSE AJAX=0;
                DO WHILE (SEARCH¬=NULL);
                SEAL=SEARCH;
                SEARCH=SEARCH->STIP.LIP;
                END;
                NT=CAT;
                LENO=NODE.LILT;
                RENO=NODE.LIRT;
                  DILO=NODE.LIL;
                DIRO=NODE.DIR;
                MENO=NODE.LIMT;
                MEDO=NODE.LIMD;
                 MEZO=NODE.NEST;
                ALLOCATE STIP;
                 STIP.TIPO=2;
                QUEEN=ST;
                IF( AJAX=1) THEN DOG->NODE.LIP=ST;
                ELSE SEAL->STIP.LIP=ST;
                STIP.TIL=NODE.TIL;
```

```
PL/I OPTIMIZING COMPILER                  /*    MODULE     PROGRAM     */

     STMT LEV NT

      367   2  1          STIP.TIR=NODE.TIR;
      368   2  1          IF (NODE.TIR(1)=0) THEN GO TO STACK;
      369   2  1          DO NAL=1 TO DIRO;
      370   2  2          LAD=CAT->NODE.WHIP(NAL);
      371   2  2          IF (LAD=CAT) THEN GO TO HAWK;
      372   2  2          CALL TRAVEL(LAD, QUEEN, CAT);
      373   2  2      HAWK: LAD=CAT->NODE.NAIL(NAL);
      374   2  2          IF (LAD=CAT) THEN GO TO SNACK;
      375   2  2          CALL TRAPEL (LAD, QUEEN, CAT);
      376   2  2      SNACK:  END;
      377   2  1          ST=QUEEN;
      378   2  1          NT=CAT;
      379   2  1      STACK: DO K=1 TO DIRO;
      380   2  2          IF( NODE.WHIP(K)=CAT) THEN STIP.WHIP(K)=ST;
      381   2  2          ELSE STIP.WHIP(K)=NODE.WHIP(K);
      382   2  2          IF( NODE.NAIL(K)=CAT) THEN STIP.NAIL(K)=ST;
      383   2  2          ELSE STIP.NAIL(K)=NODE.NAIL(K);
      384   2  2          END;
      385   2  1          SEARCH=DOG->NODE.LID;
      386   2  1          IF (SEARCH=NULL) THEN AJAX=1;
      387   2  1          ELSE AJAX=0;
      388   2  1           DO WHILE(SEARCH¬=NULL);
      389   2  2          SEAL=SEARCH;
      390   2  2          SEARCH=SEARCH->STID.LID;
      391   2  2          END;
      392   2  1          IF AJAX=1  THEN DOG->NODE.LID=CAT->NODE.LID;
      393   2  1          ELSE   SEAL->STID.LID=CAT->NODE.LID;
      394   2  1          STIP.LIP=CAT->NODE.LIP;
      395   2  1           A=DOG->NODE.GIN;
      396   2  1          B=CAT;
      397   2  1          NT=CAT;
      398   2  1          FREE NODE;
      399   2  1          NT=DOG;
      400   2  1          DO J=1 TO A;
      401   2  2          IF (NODE.SPIT(J)=B) THEN GO TO REDD;
      402   2  2          END;
      403   2  1      REDD:  NODE.SPIT(J)=NULL;
      404   2  1          NODE.LILT=NODE.LILT+LENO;
      405   2  1          NODE.LIRT=NODE.LIRT+RENO;
      406   2  1          NODE.LIMT=NODE.LIMT+MENO;
      407   2  1          NODE.LIMD=NODE.LIMD+MEDO;
      408   2  1          NODE.NEST=NODE.NEST+MEZO;
      409   2  1          NODE.GINT=NODE.GINT-1;
      410   2  1          IF(NODE.GINT¬=0) THEN GO TO LEAP;
      411   2  1          ELD(JO)=DOG;
      412   2  1          JO=JO+1;
      413   2  1          GO TO LEAP;
      414   2  1      SKIP:GOLM(M)=CAT;
      415   2  1          M=M+1;
```

```
PL/I OPTIMIZING COMPILER                    /*    MODULE     PROGRAM      */

    STMT LEV NT

     416    2  1      LEAP:   END;
     417    2  0           FREE OLD;
     418    2  0           BUD=JO-1;
     419    2  0            IF(BUD=0) THEN GO TO ALE;
     420    2  0           ALLOCATE OLD(BUD);
     421    2  0           DO K=1 TO BUD;
     422    2  1           OLD(K)=ELD(K);
     423    2  1           END;
     424    2  0           FREE ELD;
     425    2  0           GO TO LOOP_1;
     426    2  0      ALE:  BUG=M-1;
     427    2  0           ALLOCATE GOLD(BUG);
     428    2  0           DO M=1 TO BUG;
     429    2  1           GOLD(M)=GOLM(M);
     430    2  1           END;
     431    2  0           FREE GOLM;
     432    2  0           RETURN;
     433    2  0           END COALESCE;
```

The array of initial gateless node pointers OLM(K) (K = 1,2,...,BUD) is freed after its values have been passed on to array OLD. And in anticipation of the set of NODES to be modularized array GOLM is allocated.

For the pressure tank example it may be seen that once G8 has been collapsed with G7, G7 can immediately be collapsed with G5. Two nested loops (LOOP-1 and LOOP-2) are needed by COALESCE to be able to deal with this type of situations. Thus, in LOOP-2 every time a coalescing of a NODE pointed at by OLD(I) (for some I) unfold, a new gateless node, array ELD(JO) (JO = 1,2,...BUD) will store the pointer location for the new gateless node pointers OLD(I) (I = 1,2,...,new BUD value). And this new set is in turn processed by LOOP-2, and so on until no gate can be found which may be coalesced (i.e., until BUD = 0). At this point a set of NODES located by GOLD has to be modularized by MODULA before any further collapsing of gates is possible.

For the pressure tank example, initially array OLD consists of

```
OLD(1) = SPINE(6);
OLD(2) = SPINE(8);
OLD(3) = SPINE(9);
```

The first set of iterations for LOOP-2 will find which nodes are to be coalesced and which must be collapsed. Thus for

```
I = 1:      CAT = SPINE(6), DOG = SPINE(5)
  =>         CAT → NODE.VALUE = DOG → NODE.VALUE = 2

I = 2:      CAT = SPINE(8), DOG = SPINE(7)
  =>        CAT → NODE.VALUE = DOG → NODE.VALUE = 2

I = 3:      CAT = SPINE(9), DOG = SPINE(4)
  =>        CAT → NODE.VALUE = 203 ≠ DOG → NODE.VALUE
```

Therefore SPINE(9)→ NODE must be modularized, while SPINE(6) →NODE and SPINE(8) →NODE should be freed and their inputs transferred to SPINE(5)→ NODE and SPINE(7)→ NODE respectively, by means of two STIP structures. STIP structures have the following composition

```
1 STIP BASED(ST)
2 TIPO FIXED,
2 LIP POINTER,
2 DIL FIXED BINARY,
2 DIR FIXED BINARY,
2 NAIL(DIRO REFER(STIP.DIR)) POINTER,
2 WHIP(DIRO REFER(STIP.DIR)) POINTER,
2 TIR(DIRO REFER(STIP.DIR)FIXED,
2 TIL(DILO REFER(STIP.DIL)) FIXED;
```

Variables DIL and TIL are needed for the storage of free leaf inputs, while DIR, TIR, NAIL and WHIP handle the information associated with r-leaf inputs including their interconnections with other structures in the tree.

Procedures TRAVEL and TRAPEL are called by COALESCE in order to reassign to the new STIP structure the NAIL and WHIP interconnections other structures originally had with the node which is replaced by the STIP structure.

For the pressure tank example the first two STIP structures created are

1 STIP BASED(ST_1)

```
2 TIPO = 2,
2 LIP = NULL,
2 DIL = 3,
2 DIR = 1,
2 NAIL(1) = NULL,
2 WHIP(1) = NULL,
2 TIR(1) = 0
2 TIL(1) = 5, TIL(2) = 6, TIL(3) = 7;
```

```
1 STIP BASED(ST_2)
2 TIPO = 2,
2 LIP = NULL
2 DIL = 2,
2 DIR = 1,
2 NAIL(1) = SPINE(7)
2 WHIP(1) = APT_1
2 TIR(1) = 30001,
2 TIL(1) = 9, TILL(2) = 10;
```

At the same time TRAPEL transfers the WHIP interconnection of SPINE(7) → NODE

```
1 NODE BASED (NT = SPINE(7)),
2 TIPO = 1,
2 NAME = 7,
2 VALUE = 2,
:
2 DIR = 1,
2 NAIL(1) = SPINE(3)
2 WHIP(1) = ST_2,
:
```

The two structures ST_1→ STIP and ST_2→ STIP, are attached to SPINE(5) → NODE and SPINE(7) → NODE respectively by the statements

```
SEARCH = DOG  NODE.LIP;
IF(SEARCH = NULL, THEN AJAX = 1);
:
IF (AJAX = 1) THEN DOG → NODE.LIP = ST;
```

(Recall NODE.LIP was initialized to be NULL in INITIAL. Similarly NODE.LID, STIP.LIP and STID.LID are also initialized to be NULL).

Hence SPINE(5) → NODE.LIP = ST_1 and SPINE(7) → NODE.LIP = ST_2.

The STIP.LIP pointer is necessary since more than one node may coalesce with the same NODE.ROOT. In fact, after a second iteration through LOOP-1 gates (G5, G6, G7, G8) will be collapsed together for the pressure tank rupture fault tree. The set of gates will then be represented by

```
1 NODE BASED (NT = SPINE(5)),
2 TIPO = 1,
2 NAME = 5,
2 VALUE = 2,
2 GINT = 0,
2 LILT = 6,
2 LIRT = 2,
2 LIMD = 0,
2 LIMT = 0,
2 NEST = 0,
2 WHIZ = 0,
2 ROOT = SPINE(4),
2 LIP = ST1
```

```
2 LID = NULL,
2 GIN = 2,
2 LIL = 1,
2 DIR = 1,
2 NAIL(1) = NULL,
2 WHIP(1) = NULL,
2 TIR(1) = 0,
2 SPIT(1) = NULL, SPIT(2) = NULL,
2 TIL(1) = 0;
```

---

1 STIP BASED(ST_1)

2 TIPO = 2,

2 LIP = ST_3

2 DIL = 3,

2 DIR = 1,

2 NAIL(1) = NULL,

2 WHIP(1) = NULL,

2 TIR(1) = 0,

2 TIL(1) = 5, TIL(2) = 6, TIL(3) = 7;

---

1 STIP BASED (ST_3)

2 TIPO = 2,

2 LIP = ST_2

2 DIL = 1,

2 DIR = 1,

2 NAIL(1) = SPINE(3),

2 WHIP(1) = ST_2,

```
2 TIR(1) = 30001,
2 TIL(1) = 8;
```

```
1 STIP BASED (ST2)
2 TIPO = 2,
2 LIP = NULL,
2 DIL = 2,
2 DIR = 1,
2 NAIL(1) = ST2
2 WHIP(1) = APT1,
2 TIR(1) = 30001,
2 TIL(1) = 9, TIL(2) = 10;
```

At this point gates G5 and G9 are ready to be processed by MODULA and no more gateless nodes can be found which may be coalesced, i.e.,

```
BUD = 0;
BUG = 2;
GOLD(1) = SPINE(5);
GOLD(2) = SPINE(9);
```

III.8. <u>MODULA</u>

The objective of procedure MODULA, is to modularize all those gateless nodes which cannot be further coalesced with their root-node.

Recall that a gateless node will have WHIP and NAIL interconnections with other parts of the tree if the set of replicated events within its domain is not complete. To allow for this possibility, MODULA temporarily allocates a MOD structure to represent a modularized node. A MOD structure, say MOD_a, will then be transformed into a proper module (represented by a PROP structure) only if it shows no interconnections with other nodes in the tree. Otherwise procedures COALESCE and MODULA will need to further transform the tree

```
DO WHILE (FLAG ¬= 0),
CALL COALESCE;
CALL MODULA;
END;
```

until a MOD structure is found connected to a set of MOD structures (nested modules) including MOD_a and containing in its domain a complete set of replicated inputs.

This set of nested modules will then be given a higher order modular representation by procedure BOOLEAN. In general a tree will contain several complete sets of nested modules, and each time such a set is found BOOLEAN will be called by MODULA.

Structures MOD and PROP have the following composition

```
1 MOD BASED(MT)
2 TIPO FIXED,
2 NAME FIXED,
2 VALUE FIXED,
2 NEST FIXED,
2 LIM FIXED BINARY,
2 RIM FIXED BINARY,
2 RIMO FIXED BINARY,
2 MIM FIXED BINARY,
2 MID FIXED BINARY,
2 NAIL (LIRO REFER(RIMO)) POINTER
2 WHIP (LIRO REFER(RIMO)) POINTER,
2 TIR (LIRE REFER(RIM)) POINTER,
2 TID (LIDE REFER(MID)) POINTER
2 PIM (LIME REFER(MOD.MIM)) POINTER,
2 TIM (LIME REFER(MOD.LIM)) FIXED;
```

```
1 PROP BASED (PT),
2 TIPO FIXED,
2 ROOT POINTER,
2 REZ FIXED BINARY,
2 NAME FIXED,
2 VALUE FIXED,
2 LIM FIXED BINARY,
2 MIM FIXED BINARY,
2 HOST POINTER,
2 REL (DEL REFER (PROP.REZ)) FLOAT,
2 TIL (LILE REFER (PROP.LIM)) FIXED,
2 PIM (LIME REFER(PROP.MIM)) POINTER;
```

Before proceeding on to define each of the variables contained in structures PROP and MOD, it is necessary to explain how STID structures are used to represent MOD and PROP structures while their root node has not been modularized.

Structure STID has the following composition

```
1 STID BASED (SD),
2 TIPO FIXED,
2 LID POINTER,
2 STIM FIXED,
2 LTIM POINTER,
2 DIR FIXED BINARY,
2 NAIL (DIRO REFER (STID.DIR)) POINTER,
2 WHIP (DIRO REFER(STID.DIR)) POINTER;
```

(STID.TIPO = 3 for all STIDs)

For every newly created PROP or MOD structure a STID structure is allocated and attached in its place as an input to the root node which corresponds to the MOD or PROP structure. Variables LTIM and STIM identify the structure represented by STID i.e.,

$$\text{STID.LTIM} = \begin{cases} \text{MT for MOD structures} \\ \text{PT for PROP structures} \end{cases}$$

$$\text{STID.STIM} = \begin{cases} \text{MT} \quad \text{MOD.NAME} \\ \text{PT} \quad \text{PROP.NAME} \end{cases}$$

If STID represents a nested module (i.e., a MOD structure) then necessarily a set of WHIP and NAIL interconnections exists between the nested module and other gates in the tree, these interconnections are therefore passed on from MOD to its STID representation, i.e.,

$$\text{STID.NAIL} = \begin{cases} \text{MOD.NAIL for nested modules} \\ \text{NULL for PROP modules} \end{cases}$$

$$\text{STID.WHIP} = \begin{cases} \text{MOD.WHIP for nested modules} \\ \text{NULL for PROP modules} \end{cases}$$

Finally, STID.LID is necessary in case more than one MOD or PROP structures are attached as inputs to a node. In general a set of LID connections will exist of the form

```
1 NODE BASED (NT)
2 TIPO = 1,
:
2 LIP,
2 LID = SD1,
:
________________________
1 STID BASED (SD1)
2 TIPO = 3,
2 LID = SD2
:
________________________
:
:
:
________________________
1 STID BASED (SDn),
2 TIPO = 3,
2 LID = NULL,
:
________________________
```

A description of the variables contained in structure MOD follows:

MOD.TIPO = 4 for every MOD structure. It is needed to distinguish MOD from the other type of structures (STIP, STID, NODE, AP) handles together by TRAVEL and TRAPEL.

MOD.NAME is a number identifying the gate associated with the MOD structure (MOD.NAME = NODE.NAME).

MOD.VALUE identifies the type of gate operator associated with the MOD structure (MOD.VALUE = NODE.VALUE).

MOD.NEST measures the total number of nested modules (MOD structures) within the domain of the gate associate with the MOD structure (MOD.NEST = NODE.NEST).

MOD.LIM dimensions the array of free leaf inputs attached to MOD.

MOD.RIM dimensions the array of replicated leaf inputs attached to MOD.

MOD.RIMO dimensions the array of WHIP and NAIL interconnections attached to MOD (notice MOD.RIM ≠ MOD.RIMO).

MOD.MIM dimensions the array of independent module (PROP structures) inputs attached to MOD.

MOD.MID dimensions the array of nested modules (MOD structures) inputs directly attached to MOD (Notice MOD.MID ≠ MOD.NEST).

MOD.NAIL and MOD.WHIP are the arrays of pointers interconnecting MOD with other parts of the tree which have replicated inputs in common to the full domain of MOD.

MOD.TIR is the array of replicated leaf inputs attached to MOD.

Thus MOD.PID(I) will be the pointer for the Ith nested module input to MOD (MOD.PID(1) = MT_I) and MOD.TID will be the name of the Ith nested module input (MOD.TID(I) = MT_1 MOD.NAME) Arrays MOD.PIM and MOD.TIM identify the free module inputs attached to MOD. Thus MOD.PIM(J) is the pointer for the Jth free module input to MOD(MOD.PIM(J) = PT_J) and MOD.TIM is the name of Jth free module input (MOD.TIM(J) = PT_J PROP. NAME). MOD.TIL is the array of free leaf inputs attached to MOD.

The procedure modula starts out by determining the storage space needed to allocate a MOD structure for gateless node M (M=1,2,...,BUG) and assigns the values to variables MOD.VALUE, MOD.NAME, MOD.NEST and MOD.TIPO with the following statements:

```
                 /*     MODULA    */
 1  0          MODULA: PROC;
 2  0                ALLOCATE MODUL;
 2  0                IT=IT+1;
 2  0                BUR(IT)=BUT;
 2  0                ALLOCATE FELD(BUG);
 2  0                MO=1;
 2  0                DO M=1 TO BUG;
 2  1                CAT=GOLD(M);
 2  1                NT=CAT;
 2  1                LILE=NODE.LILT;
 2  1                LIRE=NODE.LIRT;
 2  1                LIME=NODE.LIMT;
 2  1                 LIRO=NODE.LIRT;
 2  1                LIDE=NODE.LIMD;
 2  1                SEARCH=NODE.LID;
 2  1                DO WHILE (SEARCH¬=NULL);
 2  2                SEAL=SEARCH;
 2  2                DIRT=SEAL->STID.DIR;
 2  2                IF (DIRT=1 & SEAL->STID.NAIL(1)=NULL) THEN DIRT=0;
 2  2                 LIRO=LIRO+DIRT;
 2  2                SEARCH=SEAL->STID.LID;
 2  2                END;
 2  1                IF(LILE=0) THEN LILE=1;
 2  1                IF LIME=0 THEN LIME=1;
 2  1                IF LIDE=0 THEN LIDE=1;
 2  1               IF LIRE=0 THEN LIRE=1;
 2  1                IF LIRO=0 THEN ORO=1;
 2  1                 ELSE ORO=0;
 2  1                IF ORO=1 THEN LIRO=1;
 2  1                ALLOCATE MOD;
```

```
2   1          QUEEN=MT;
2   1          MOD.TIL=0;
2   1           MOD.TIR=0;
2   1          MOD.NAIL=NULL;
2   1          MOD.WHIP=NULL;
2   1          MOD.PIM=NULL;
2   1           MOD.TIM=0;
2   1          MOD.PID=NULL;
2   1          MOD.TID=0;
2   1          MODUL.DULL(M)=MT;
2   1          MOD.VALUE=NODE.VALUE;
2   1          MOD.NAME=NODE.NAME;
2   1          MOD.NEST=NODE.NEST;
2   1      MOD.TIPO=4;
```

Notice that structure MOD has a number of interconnections (WHIP (I) and NAIL(I), I = 1,2,...,LIRO) which is in general different from the number of replicated inputs (TIR(I) I = 1,2, ...,LIRE) it contains, i.e., LIRO $\neq$ LIRE. This reflects the fact that structure MOD absorbs only those inputs contained in the structure NODE and all its connected STIP structures. At the same time, however, MOD receives all interconnections attached to the NODE structure as well as its STIP and STID connected structures. This feature particular to MOD structures makes it possible to identify higher order modules contained in the tree. Indeed, a MOD structure will correspond to a higher order module only if all its interconnections are self-contained, i.e.,

$$\text{MOD.NAIL(I)} = \text{MT}$$

$$\text{and}$$

$$\text{MOD.WHIP(I)} = \begin{cases} \text{MT} \\ \text{APT}_J \end{cases}$$

for all I (I= 1,2,...,LIRO; J = 1,2,...,NUM; with NUM = total number of replicated components in the domain of the higher order module).

The next variables to be assigned values by MODULA. are MOD.TIL and MOD,TIR which get values from the NODE structure and the set of STIP structures connected to the NODE:

```
2  1         SEARS=NODE.LIP;
2  1         BIL=0;
2  1          DIR=0;
2  1         DO WHILE(SEARS¬=NULL);
2  2         ST=SEARS;
2  2         DIAL=STIP.DIL;
2  2         IF (DIAL=1 & STIP.TIL(1)=0)  THEN DIAL=0;
2  2         IF DIAL=0 THEN GO TO BACH;
2  2         DO I=1 TO DIAL;
2  3         MOD.TIL(BIL+I)=STIP.TIL(I);
2  3         END;
2  2    BACH:  DIAR=STIP.DIR;
2  2         IF (DIAR=1 & STIP.TIR(1)=0)  THEN DIAR=0;
2  2         IF DIAR=0  THEN GO TO MACH;
2  2         DO I=1 TO DIAR;
2  3         MOD.TIR(DIR+I)=STIP.TIR(I);
2  3         END;
2  2    MACH: BIL=BIL+DIAL;
2  2         DIR=DIR+DIAR;
2  2         SEARS=SEARS->STIP.LIP;
2  2          END;
2  1         DO I=BIL+1 TO LILE;
2  2         J=I-BIL;
2  2         MOD.TIL(I)=NODE.TIL(J);
2  2          END;
2  1         DO I=BIR+1 TO LIRE;
2  2         J=I-BIR;
2  2         MOD.TIR(I)=NODE.TIR(J);
2  2         END;
```

At this point once all WHIP and NAIL interconnections in structure NODE and the set of STIPS connected to the NODE are transferred to MOD, then all these structures may be freed.

```
2  1         NIR=NODE.DIR;
2  1         IF (NIR=1 & NODE.TIR(1)=0)  THEN NIR=0;
2  1         IF (NIR=0)  THEN GO TO BITE;
2  1         DO NAL=1 TO NIR;
2  2         LAD=CAT->NODE.WHIP(NAL);
2  2         IF (LAD=CAT)  THEN GO TO CITE;
```

```
2  2          CALL TRAVEL (LAD, QUEEN, CAT);
2  2      CITE: LAD=CAT->NODE.NAIL(NAL);
2  2           IF LAD=CAT THEN GO TO RITE;
2  2          CALL TRAPEL(LAD, QUEEN, CAT);
2  2       RITE: END;
2  1          NT=CAT;
2  1          DO K=1 TO NIR;
2  2          IF (NODE.WHIP(K)=CAT) THEN MOD.WHIP(K)=NT;
2  2          ELSE MOD.WHIP(K)=NODE.WHIP(K);
2  2          IF(NODE.NAIL(K)=CAT) THEN MOD.NAIL(K)=NT;
2  2          ELSE MOD.NAIL(K)=NODE.NAIL(K);
2  2          END;
2  1      BITE: SEARCH=NODE.LIP;
2  1          SEARS=NODE.LID;
2  1          SEAN=NODE.ROOT;
2  1          FREE NODE;
2  1          DO WHILE (SEARCH¬=NULL);
2  2          ST=SEARCH;
2  2          BAT=ST;
2  2           SIR=STIP.DIP;
2  2          IF (SIR=1 & STIP.TIR(1)=0) THEN SIR=0;
2  2          IF SIR=0 THEN GO TO BITS;
2  2          DO NAL=1 TO SIR;
2  3          LAD=BAT->STIP.WHIP(NAL);
2  3           IF (LAD=BAT) THEN GO TO CITS;
2  3          CALL TRAVEL(LAD, QUEEN, BAT);
2  3      CITS: LAD=BAT->STIP.NAIL(NAL);
2  3          IF(LAD=BAT) THEN GO TO RITS;
2  3          CALL TRAPEL (LAD, QUEEN, BAT);
2  3      RITS:   END;
2  2          ST=BAT;
2  2          DO K=1 TO SIR;
2  3          IF(STIP.WHIP(K)=ST) THEN MOD.WHIP(NIR+K)=NT;
2  3          ELSE MOD.WHIP(NIR+K)=STIP.WHIP(K);
2  3          IF (STIP.NAIL(K)=ST) THEN MOD.NAIL(NIR+K)=NT;
2  3          ELSE MOD.NAIL(NIR+K)=STIP.NAIL(K);
2  3           END;
2  2       BITS:  NIR=NIR+SIR;
2  2           SEARCH=SEARCH->STIP.LIP;
2  2          FREE STIP;
2  2           END;
```

It should be noted that before freeing structure NODE, its pointer variable NODE.LID was assigned to variable SEARS. Keeping this pointer will make it possible to transmit to MOD all the values it receives from the set of STID structures previously connected to the NODE.

A loop similar to the one used for transmitting to MOD values from the STIP structures (DO WHILE (SEARCH¬= NULL;) follows for the set of STID structures

```
2  1          LAU=0;
2  1          PAU=0;
2  1          DO WHILE(SEARS¬=NULL);
2  2          SD=SEARS;
2  2          BAT=SD;
2  2          SIR=STID.DIR;
2  2          IF (SIR=1 & STID.NAIL(1)=NULL) THEN SIR=0;
2  2          IF SIR=0 THEN GO TO BITA;


2  2          DO NAL=1 TO SIR;
2  3          LAD=BAT->STID.WHIP(NAL);
2  3          IF(LAD=BAT ) THEN GO TO CITA;
2  3          CALL TRAVEL (LAD, QUEEN, BAT);
2  3     CITA: LAD=BAT->STID.NAIL(NAL);
2  3          IF (LAD=BAT) THEN GO TO RITA;
2  3          CALL TRAPEL(LAD, QUEEN, BAT);
2  3     RITA: END;
2  2          SD=BAT;
2  2          DO K=1 TO SIR;
2  3          IF(STID.WHIP(K)=SD) THEN MOD.WHIP(NIR+K)=MT;
2  3          ELSE MOD.WHIP(NIR+K)=STID.WHIP(K);
2  3          IF(STID.NAIL(K)=SD) THEN MOD.NAIL(NIR+K)=MT;
2  3          ELSE MOD.NAIL(NIR+K)=STID.NAIL(K);
2  3          END;
2  2          NIR=NIR+SIR;
2  2           LAU=LAU+1;
2  2           MOD.TID(LAU)=STID.STIM;
2  2           MOD.PID(LAU)=STID.LTIM;
2  2          GO TO PITA;
2  2     BITA: PAU=PAU+1;
2  2          MOD.TIM(PAU)=STID.STIM;
2  2          MOD.PIM(PAU)=STID.LTIM;
2  2     PITA: SEARS=SEARS->STID.LID;
2  2          FREE STID;
2  2          END;
```

A STID structure will either transmit values to MOD.TIM and MOD.PIM if it represents a PROP structure (proper module) in which case STID includes no WHIP and NAIL interconnections, or it will transmit values to MOD.TID and MOD.PID as well as values to pointers MOD.WHIP and MOD.NAIL if it represents a MOD structure (nested module).

Each STID pertaining to the set is processed by the loop (SEARS = SEARS →STID.LID; ⇒SEARS points each time at a new STID in the set after which its storage is released (FREE STID;).

At this point all variables contained in the new MOD structure have been assigned their values, so MODULA can proceed now to check whether the MOD structure created represents a proper or a nested module.

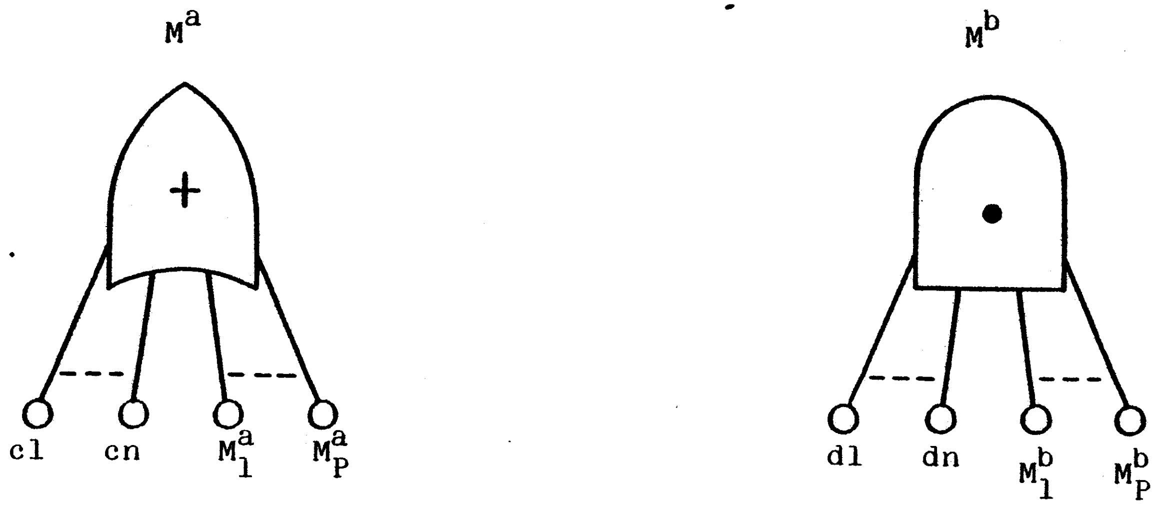
Before allocating a MOD structure, variable ORO was used to distinguish those gateless nodes having no replicated events in their domain (IF(LIRO = 0) THEN ORO = 1; ELSE ORO = 0;). The MOD structure for a gateless node having no replicated inputs may be immediately transformed into either a "simple" PROP structure (Figure 3.21) or into a set of PROP structures organized by a set of Boolean vectors characteristic of a symmetric (k-out of-n) gate (Figure 3.22).

Symmetric gates are allowed to appear explicitly in the fault tree, as long as each of their inputs is independent from the rest of the tree (i.e., each input to the gate is either a component or a super-component). Symmetric gate operators are represented by a three digit number (KON). The highest digit represents the minimum number of simultaneous failures necessary to cause a gate failure, the middle digit is always equal to zero, and the lowest digit represented the total number of inputs to the gate (Thus, a node having a 2-out of- 4 gate operator has a NODE.VALUE = 204).

In the next statements MODULA considers the two possibilities available for a non-replicated event MOD structure.

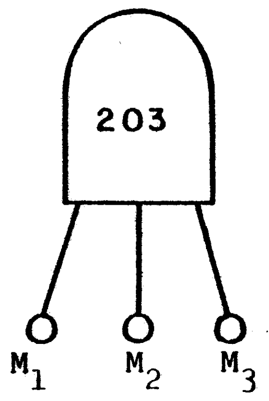
```
    IF (ORO = 1 & MOD.VALUE > 2) THEN GO TO RED;
    IF (ORO = 1 & MOD.VALUE <= 2) THEN GO TO HANA;
```

For the pressure tank example MODULA will allocate two MOD structures. The first one (GOLD(1)) associated with gate G5 does contain replicated events in its domain and will there-



(M^a and M^b are simple prop structures)

FIGURE 3.21 SIMPLE OR AND AND GATE PROP STRUCTURES

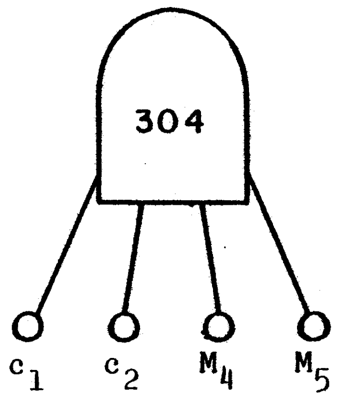


$Y^B = (Y_{M1}, Y_{M2}, Y_{M3})$

$S_1 = (0, 1, 1)$

$S_2 = (1, 0, 1)$

$S_3 = (1, 1, 0)$



$\underset{\to}{Y}^B = (Y_{c1}, Y_{c2}, Y_{M4}, Y_{M5})$

$S_1 = (0, 1, 1, 1)$

$S_2 = (1, 0, 1, 1)$

$S_3 = (1, 1, 0, 1)$

$S_4 = (1, 1, 1, 0)$

FIGURE 3.22 SYMMETRIC HIGHER ORDER MODULES

fore be later checked on whether it represents a nested module or the top event for a higher order module (i.e., the parent gate for a set of nested modules).

The second MOD structure associated with gate G9 (GOLD(2)) represents a symmetric gate module and will therefore be given its corresponding Boolean representation by procedure SYMM.

In the next section of this Chapter, the methods by which procedures BOOLEAN and SYMM derive a Boolean representation for higher order modules and for symmetric gate modules explicitly included in a fault tree, are discussed.

For the pressure tank example, the following MOD structures represent gates G5 and G9.

1 MOD BASED (MT_1),
2 TIPO = 4,
2 NAME = 5,
2 VALUE = 2,
2 NEST = 0,
2 LIM = 6,
2 RIM = 2,
2 RIMO = 2,
2 MIM = 1,
2 MID = 1,
2 NAIL(1) = SPINE(3), NAIL(2) = MT_1,
2 WHIP(1) = MT_1, WHIP(2) = APT_1,
2 TIR(1) = 30001, TIR(2) = 30001,
P
2 RID(1) = NULL,
2 TID(1) = 0,

```
2 PIM(1) = NULL,
2 TIM(1) = 0,
2 TIL(1) = 5, TIL(2) = 6, TIL(5) = 7, TIL(4)=8,

  TIL(5) = 9, TIL(6) = 10;
```

It may be seen that this MOD structure, associated with gate G5 represents a nested module since the requirement MOD.NAIL (I) = MT_1 is not satisfied for I = 1.

```
1 MOD BASED (MT2)
2 TIPO = 4,
2 NAME = 9,
2 VALUE = 203,
2 NEST = 0,
2 LIM = 3,
2 RIM = 1,
2 RIMO = 1,
2 MID = 1,
2 NAIL(1) = NULL,
2 WHIP(1) = NULL,
2 TIR(1) = 0,
2 PID(1) = NULL,
2 TID(1) = 0,
2 PIM(1) = NULL,
2 TIM(1) = 0
2 TIL(1) = 11, TIL(2) = 12, TIL(3) = 13;
```

procedure SYMM will automatically generate the Boolean representation for this MOD structure associated with gate G9

$$Y^B = (y_{c11}, y_{c12}, y_{c13})$$
$$S_1 = (1, 0, 1)$$
$$S_2 = (0, 1, 1)$$
$$S_3 = (1, 1, 0)$$

and these vectors will be attached to the PROP structure representing gate G9 (see section III.9.2).

The set of statements outlined below form the final part of the MODULA procedure. The tasks they perform include

(a) Testing if a MOD structure containing replicated components represents a nested or a higher order module.

(b) Calling procedures BOOLEAN and SYMM to generate minimal cut-set representations for higher order and explicitly symmetric modules.

(c) Allocating PROP structures for those MOD structures which include no replicated events.

(d) Allocating STID structures to represent PROP and MOD structures and attaching them to NODE structures in the fault tree.

```
 588    2  1           IF (ORO=1  & MOD.VALUE>2)  THEN GO TO RED;
 589    2  1          IF (ORO=1  & MOD.VALUE<=2)  THEN GO TO HANA;
 590    2  1          SUM=0;
 591    2  1          IR=1;
 592    2  1           ALLOCATE GUT;
 593    2  1          NOX=0;
 594    2  1          DO CAP=1 TO LIRO;
 595    2  2          VIC=MOD.NAIL(CAP);
 596    2  2          IF(VIC¬=MT) THEN GO TO DANA;
 597    2  2          VIT=MOD.WHIP(CAP);
 598    2  2          IF (VIT¬=MT & VIT->NODE.TIPO¬=0) THEN GO TO DANA;
 599    2  2          IF (VIT->NODE.TIPO¬=0) THEN GO TO SANA;
 600    2  2          REV=VIT->REP;
 601    2  2           IF (REV<0) THEN DO;
 602    2  3          NOX=1;
 603    2  3          SUM=SUM-REV;
 604    2  3          MA=VIT->NAP;
 605    2  3          DA=-CEIL(-MA/10000);
 606    2  3          JA=-CEIL(-MA/1000);
 607    2  3          NA=MA-(1000)*JA;
 608    2  3         GUT(IR)=10000*DA+1000+NA;
 609    2  3          GUT(IR+1)=GUT(IR)+1000;
 610    2  3         IR=IR+2;


 611    2  3          END;
 612    2  2          ELSE DO;
 613    2  3          SUM=SUM+REV;
 614    2  3          GUT(IR)=VIT->NAP;
 615    2  3           IR=IR+1;
 616    2  3          END;
 617    2  2     SANA:   END;
 618    2  1          PUT EDIT('TOTAL SUM REP=',SUM)
                     (SKIP(2),X(2),A(14),F(5));
 619    2  1          NUM=IR-1;
 620    2  1          ALLOCATE PUT;
 621    2  1          DO I=1 TO NUM;
 622    2  2           PUT(I)=GUT(I);
 623    2  2          END;
 624    2  1          FREE GUT;
 625    2  1          CALL BOOLEAN;
1111    2  1     CANA:        DIRO=1;
1112    2  1          ALLOCATE STID;
1113    2  1          SEARS=SD;
1114    2  1          STID.NAIL=NULL;
1115    2  1          STID.WHIP=NULL;
1116    2  1          STID.LID=NULL;
1117    2  1          STID.STIM=STORK->PROP.NAME;
1118    2  1          STID.LTIM=STORK;
1119    2  1           MT=MODUL.DULL(M);
1120    2  1          FREE MOD;
1121    2  1          IF (SEAN=NULL) THEN GO TO REAL;
1122    2  1         IF SEAN->NODE.TIPO=1  THEN GO TO CANX;
1123    2  1          APT=SEAN;
1124    2  1          AP.SPIT=STORK;
1125    2  1          STORK->PROP.ROOT=SEAN;
1126    2  1          GO TO REAP;
1127    2  1      CANX:  NT=SEAN;
1128    2  1          NODE.LINT=NODE.LINT+1;
1129    2  1           SIERRA=NODE.LID;
1130    2  1          IF (SIERRA=NULL) THEN NODE.LID=SEARS;
1131    2  1           ELSE GO TO ZEAL;
```

```
PL/I OPTIMIZING COMPILER                 /*     MODULE     PROGRAM      */

    STMT LEV NT

    1132   2  1          GO TO VEAL;
    1133   2  1    RED:       NUB=MOD.LIM;
    1134   2  1           IF(NUB=1 & MOD.TIL(1)=0) THEN NUM=0;
    1135   2  1          ELSE  NUM=NUB;
    1136   2  1          WEST=MOD.MIM;
    1137   2  1          IF(WEST=1 & MOD.PIM(1)=NULL) THEN NEZT=0;
    1138   2  1          ELSE NEZT=WEST;
    1139   2  1          ALLOCATE PER;
    1140   2  1          PER.TAR=MOD.TIL;
    1141   2  1          PER.KIM=MOD.PIM;
    1142   2  1          PER.JIM=MOD.TIM;
    1143   2  1          LOST=PR;
    1144   2  1          LILE=1;
    1145   2  1          LIME=1;
    1146   2  1          ALLOCATE  PROP;
    1147   2  1          PROP.TIPO=5;
    1148   2  1          IB=IB+1;
    1149   2  1           STORK=PT;
    1150   2  1          BOST(IB)=STORK;
    1151   2  1          DO L=1 TO WEST;
    1152   2  2          AT=PER.KIM(L);
    1153   2  2          IF (AT¬=NULL) THEN AT->PROP.ROOT=STORK;
    1154   2  2          END;
    1155   2  1          PROP.NAME=MOD.NAME;
    1156   2  1          PROP.VALUE=MOD.VALUE;
    1157   2  1          PROP.TIL=0;
    1158   2  1          PROP.TIM=0;
    1159   2  1          PROP.PIM=NULL;
    1160   2  1          PUT EDIT ('SYMM MODULE NAME=',PROP.NAME,'VALUE=',
                         PROP.VALUE) (SKIP(2),A(17),F(5),X(2),A(6),F(5));
    1161   2  1          PROP.HOST=LOST;
    1162   2  1          LARG=NUM+NEZT;
    1163   2  1          KAY=(PROP.VALUE-LARG)/100;
    1164   2  1          CALL SYMM;
    1165   2  1          LOST->HECTOR=QUEEN;
    1166   2  1           PUT EDIT ('DEP COMPS=') (SKIP(1),A(10));
    1167   2  1          PUT LIST(PER.TAR);
    1168   2  1          PUT EDIT('DEP MODS=') (SKIP(1),A(9));
    1169   2  1          PUT LIST (PER.JIM);
    1170   2  1          PUT EDIT ('MINIMAL CUT SETS') (SKIP(2),X(12),A(16));
    1171   2  1          VIT=PER.HECTOR;
    1172   2  1          DO WHILE (VIC¬=NULL);
    1173   2  2          VIC=VIT;
    1174   2  2          PUT EDIT (VIC->COMP) (SKIP(1),P);
    1175   2  2          VIT=VIC->FLOOR;
    1176   2  2          END;
    1177   2  1          GO TO CANA;
```

```
 2  1       HANA:   LILE=MOD.LIM;
 2  1                   LIMF=MOD.MIM;
 2  1                   ALLOCATE PROP;
 2  1               PROP.TIPO=5;
 2  1               STORK=PT;
 2  1                PROP.HOST=NULL;
 2  1               PROP.NAME=MOD.NAME;
 2  1               PROP.VALUE=MOD.VALUE;
 2  1               PROP.TIL=MOD.TIL;
 2  1               PROP.TIM=MOD.TIM;
 2  1               PROP.PIM=MOD.PIM;
 2  1               ARI=PT;
 2  1               DO L=1 TO LIME;
 2  2               AT=PROP.PIM(L);
 2  2               IF (AT¬=NULL) THEN AT->PROP.ROOT=ARI;
 2  2               END;
 2  1       PUT EDIT ('FREE   MODULE NAME=', PROP.NAME,'VALUE=',
             PROP.VALUE, 'NUM LEAF INP=', PROP.LIM, 'NUM MOD INP=', PROP.MIM)
             (SKIP(2),A(19),F(5),X(2),A(6),F(5),X(2),A(13),F(5),X(2),A(12),
                                                                        F(5));
 2  1       PUT  EDIT ('LEAF INS=') (SKIP(1), A(9));
 2  1       PUT LIST( PROP.TIL);
 2  1        PUT EDIT('MOD INS=') (SKIP(1),A(8));
 2  1        PUT LIST(PROP.TIM);
 2  1               IB=IB+1;
 2  1               BOST(IB)=PT;
 2  1               FREE MOD;
 2  1               DIRO=1;



 2  1               ALLOCATE STID;
 2  1                SEARS=SD;
 2  1               STID.NAII=NULL;
 2  1               STID.WHIP=NULL;
 2  1                STID.LID=NULL;
 2  1               STID.STIM=BOST(IB)->PROP.NAME;
 2  1               STID.LTIM=BOST(IB);
 2  1               IF (SEAN=NULL) THEN GO TO REAL;
 2  1               IF (SEAN->NODE.TIPO=1) THEN GO TO HANE;
 2  1                APT=SEAN;
 2  1               AP.SPIT=STORK;
 2  1                STORK->PROP.ROOT=SEAN;
 2  1               GO TO REAP;
 2  1       HANE:  NT=SEAN;
 2  1               NODE.LIMT=NODE.LIMT+1;
 2  1               SIERRA=NODE.LID;
 2  1               IF(SIERRA=NULL)   THEN NODE.LID=SEARS;
 2  1               ELSE GO TO ZEAL;
 2  1               GO TO VEAL;
 2  1       DANA:       DIRO=MOD.RIMO;
 2  1               ALLOCATE STID;
 2  1                STID.TIPO=3;
 2  1               SEARS=SD;
 2  1               STID.STIM=MOD.NAME;
 2  1               VIC=MODUL.DULL(M);
 2  1               NT=VIC;
 2  1               STID.LTTM=VIC;
 2  1       PUT EDIT('NESTID=',STID.STIM)
                    (SKIP(1),X(2),A(7),F(5));
```

```
2  1          STID.LID=NULL;
2  1           IF (SEAN=NULL) THEN GO TO REAL;
2  1          DO NAL=1 TO DIRO;
2  2          LAD=VIC->MOD.WHIP(NAL);
2  2           IF (LAD=VIC) THEN GO TO CITO;
2  2          CALL TRAVEL(LAD,SEARS,VIC);
2  2      CITO:       LAD=VIC->MOD.NAIL(NAL);
2  2          IF (LAD=VIC)  THEN GO TO RITO;
2  2          CALL TRAPEL (LAD,SEARS,VIC);
2  2     RITO:      END;
2  1           SD=SEARS;
2  1          DO K=1 TO DIRO;
2  2          IF (MOD.WHIP(K)=VIC)  THEN STID.WHIP(K)=SD;
2  2          ELSE   STID.WHIP(K)=MOD.WHIP(K);
2  2          IF MOD.NAIL(K)=VIC THEN STID.NAIL(K)=SD;
2  2          ELSE STID.NAIL(K)=MOD.NAIL(K);
2  2          END;
2  1           NT=SEAN;
2  1          NODE.LIMD=NODE.LIMD+1;
2  1          NODE.NEST=NODE.NEST+MOD.NEST+1;

2  1            SIERRA=NODE.LID;
2  1           IF (SIERRA=NULL) THEN NODE.LID=SEARS;
2  1           ELSE GO TO ZEAL;
2  1           GO TO VEAL;
2  1      ZEAL:      DO WHILE (SIERRA¬=NULL);
2  2           TIERRA=SIERRA;
2  2           SIERRA=SIERRA->STID.LID;
2  2           END;
2  1           TIERRA->STID.LID=SEARS;
2  1           GO TO VEAL;
2  1      VEAL:      A=NODE.GIN;
2  1           DO J=1 TO A;
2  2            IF(NODE.SPIT(J)=CAT)      THEN GO TO FRED;
2  2           END;
2  1      FRED:      NODE.SPIT(J)=NULL;
2  1            NODE.GINT=NODE.GINT-1;
2  1           IF(NODE.GINT¬=0)  THEN GO TO REAP;
2  1           FELD(MO)=SEAN;
2  1            MO=MO+1;
2  1            GO TO REAP;
2  1         REAL:     STORK->PROP.ROOT=NULL;
2  1           FLAG=0;
2  1      REAP:     END;
2  0           BUM=MO-1;
2  0           ALLOCATE OLM(BUM);
2  0           DO I=1    TO   BUM;
2  1           OLM(I)=FELD(I);
2  1           END;
2  0           FREE FELD;
2  0           RETURN;
2  0            END MODULA;
```

The set of statements following label HANA create PROP structures which represent simple gate modules. Variables PROP. NAME, PROP.VALUE, PROP.LIM, PROP.MIM, PROP.TIL, PROP.TIM and PROP.PIM have the same meaning and are therefore assigned the same values formerly associated with the MOD structure for the gate i.e.,

```
PROP.NAME = MOD.NAME
PROP.PIM(J) = MOD.PIM(J)(J = 1,2,...,MIM)
etc.
(PROP.TIPO = 5 for all PROP structures)
```

In the numerical evaluation to be performed later by PL-MOD, modular occurrence probabilities and Vesely-Fussell importances will be computed. These values shall be stored for each PROP structure in PROP.REL(L) and PROP.REL(2) (thus parameter DEL must be set equal to 2).

Pointer variable PROP.HOST is only needed to attach to a parent gate the Boolean vector representation for its higher order symmetric or asymmetric structure. Therefore, PROP.HOST= NULL for the case of simple gate modules.

Inspection of the DO loop (DO CAP = 1 TO LIRO;) used to test if a MOD structure represents a higher order module or a nested module, reveals that nested modules are handled by the set of statements following label DANA.MOD structures representing nested modules may not be immediately freed. Therefore for this case the STID structure created locates a MOD structure and it contains the WHIP and NAIL interconnections which were passed on by MOD to the STID structure.

Both higher order modules and explicitly symmetric modules are handled by the statements following label CANA. However this is done only after they were previously processed by BOOLEAN or SYMM respectively.

In all cases, whether the STID represents a PROP structure (simple gate module, or higher order parent module) or a MOD structure (nested module), it is attached as a pseudo-component to its node root (SEAN = CAT→NODE.ROOT). This therefore results in a decrease in the number of gates which are input to the nodes which are roots to the modularized gates (FRED: NODE.SPIT(J) = NULL; NODE.GINT = NODE.GINT-1;). Hence a number of new gateless nodes (OLM(BUM)) will be found to which procedures COALESCE and MODULA may be then applied.

III.9 BOOLEAN and SYMM

III.9.1. Description of Higher Order Modules by Means of PROP, PER and VECTOR Structures.

In its final form the modular structure for a fault tree will be given by a set of PROP structures each of them containing a set of basic events (free leaf and replicate leaf components) and proper modules (PROP structures) as inputs.

For the case of simple modular gates (Figure 3.23) each input holds the same structural relation to its gate operator. Therefore a listing of the inputs to the PROP structure together with the gate operator (AND,OR) coupling the inputs, will completely define the module. Thus, the PROP structure

```
1 PROP  BASED ($PT_{14}$),
2 TIPO = 5,
2 REZ = 2,
2 ROOT = $PT_{15}$,
2 NAME = 14,
2 VALUE = 2,
2 LIM = 2,
2 MIM = 3,
2 HOST = NULL,
2 REL(2) FLOAT,
2 TIM(1) = 10, TIL(2) = 11,
2 TIM(1) = 13, TIM(2) = 12, TIM(3) = 11,
2 PIM(1) = $PT_{13}$, PIM(2) = $PT_{12}$, PIM(3) = $PT_{11}$;
```

uniquely defines module $M_{14} = \{C_{10}, C_{11}, M_{11}, M_{12}, M_{13}; U\}$, with

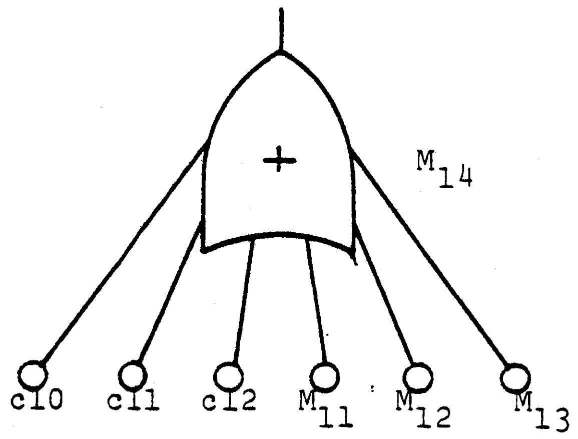


FIGURE 3.23 SIMPLE GATE MODULE

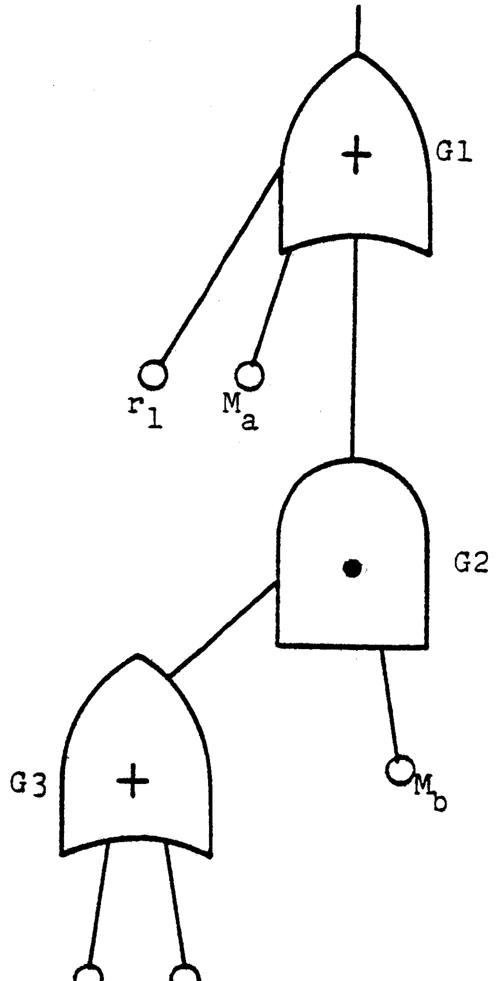


FIGURE 3.24 HIGHER ORDER MODULE

module M_{14} included as an input to module M_{15}.

However, for the case of a higher order modular gate, all its inputs do not hold the same relation with the parent gate operator. Thus, consider the higher order modulae shown in Figure 3.24 (the pressure tank fault tree example shall later be shown to have a structure similar to that of Figure 3.24). Because of the appearance of replicated input r_1 in gates G1 and G5, gates G 1, G4 and G5 do not correspond to simple gate modules representable by a PROP structure. Instead, each of these gates can be seen to be composed of a proper and an improper part

	Proper Part	Improper Part
Parent Gate G1	M_a	r_1, G_4
Nested Gate G4	M_b	G_5
Nested Gate G5	M_c	r_1

The higher order module representing this fault tree may now be constructed by taking the proper part for each gate in the structure, as well as the replicated events which provide for the interdependency among the gates, i.e.,

$$\theta G_1 = \sigma(r_1, M_1, M_4, M_5)$$

where M_i denotes the proper part for each of the gates in the higher order module. Hence $M_1 = M_a$, $M_4 = M_b$, $M_5 = M_c$.

The Boolean vector describing the minimal cut-set composition for the higher order module will then be

$\underset{\sim}{Y}^B = (y_{r_1}, y_{M_1}, y_{M_4}, y_{M_5})$ and as a result the minimal cut-sets will be represented by

$S_1 = (0, 1, 0, 0)$

$S_2 = (1, 0, 0, 0)$

$S_3 = (0, 0, 1, 1)$

From this it follows that a higher order module may be described by a set of PROP structures associated with the proper part of the parent and nested module gates, together with a set of replicated events and a series of Boolean vectors denoting each of the minimal cut-sets for the module.

The approach taken by the procedures BOOLEAN and SYMM is to attach this minimal cut-set information to the PROP structure associated with the parent gate(Pointer variable PROP.HOST is used for this purpose). Thus, for the example given in Figure 3.24, the parent gate G1 is represented by a $PROP_1$ structure containing information on its proper part M_1. In addition a structure PER will be attached to $PROP_1$ containing the information on the structural composition of the higher order module whose parent gate is G1, that is, $PROP_1.HOST = PR_1$, with PR locating a based structure PER.

Structure PER has the following composition

```
1 PER BASED (PR),
2 REZ FIXED BINARY,
2 HECTOR POINTER,
2 DEXTER POINTER,
2 RAM FIXED BINARY,
2 REL(DEL REFER (PER.REZ)) FLOAT,
2 TAR (NUM REFER(PER.RAM)) FIXED,
                    2 KIM (WEST REFER (PER.LEAL)) POINTER,
                    2 JIM (WEST REFER(PER.LEAL)) FIXED;
```

The variables contained on PER are defined as follows:

PER.REZ dimensions array PER.REL which is used to store the reliability and importance information for the higher order module (normally DEL = 2 ⇒ PER.REZ = 2).

PER.HECTOR is the pointer locating the list of VECTOR structures each defining a minimal cut-set for the higher order module.

VECTOR structures are defined by

```
                    1 VECTOR BASED (VT),
                    2 LORO FIXED BINARY,
                    2 FLOOR POINTER,
                    2 COMP BIT (LARG REFER (VECTOR.LORO));
```

The set of minimal cut-sets are then attached by PER.HECTOR = VT_1, $VT_1 \rightarrow$ VECTOR.FLOOR = $VT_2, \ldots, VT_n \rightarrow$ VECTOR.FLOOR = NULL. With VECTOR.COMP holding the Boolean bit-string representation for a minimal cut-set.

PER.DEXTER is a pointer locating a structure QER derived by procedure IMPORTANCE (see sections 3.15 and 3.16).

PER.RAM dimensions array PER.TAR which stores the number of variables identifying each of the replicated event inputs to the higher order module.

PER.LEAL dimensions arrays PER.KIM and PER.JIM, PER.LEAL equals the total number of nested modules in the domain of the parent gate.

PER.KIM contains the pointer locating the PROP structures associated with each nested module, while PER.JIM contains the number variable identifying the structure (i.e., PER.KIM(I)→ PROP.NAME = PER.JIM(I), I = k,2,...,PER.LEAL).

Thus, the PER and VECTOR structures describing the higher order modular structure of Figure 3.24 are

```
1 PER BASED (PR = PT$_1$),
2 REZ = 2,
2 HECTOR = VT$_1$,
2 DEXTER POINTER,
2 RAM = 1,
2 LEAL = 2,
2 REL(2) FLOAT,
2 TAR(1) = 20001,
2 KIM(4) = PT$_4$, KIM(2) = PT$_5$,
2 JIM(1) = 4, JIM(2) =  5;
1 VECTOR BASED (VT$_1$)
2 LORO = 4,
2 FLOOR = VT$_2$,
2 COMP = '10100'B;
1 VECTOR BASED (VT$_2$),
2 LORO = 4,
2 FLOOR = VT$_3$
2 COMP = '1000'B;
1 VECTOR BASED (UT$_3$),
2 LORO = 4,
2 FLOOR = NULL,
2 COMP = '0011'B;
```

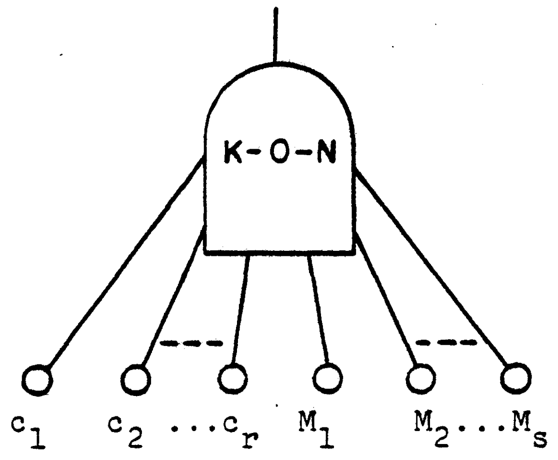
(With PT_1, PT_4 and PT_5 locating the PROP structures corresponding to gates G1, G4 and G5.)

III.9.2. Procedure SYMM

When a fault tree diagram explicitly includes a symmetric higher order module, procedure SYMM will be used to generate its Boolean vector representation. A restriction imposed by PL-MOD is that the inputs to the symmetric gate be either non-replicated basic events or modules (Figure 3.25).

Before procedure SYMM is called, the PROP and PER structure associated with the symmetric gate are created by a set of statements following label RED.

```
RED:      NUB=MOD.LIM;
            IF(NUB=1 & MOD.TIL(1)=0) THEN NUM=0;
           ELSE  NUM=NUB;
           WEST=MOD.MIM;
           IF(WEST=1 & MOD.PIM(1)=NULL) THEN NEZT=0;
           ELSE NEZT=WEST;
           ALLOCATE PER;
           PER.TAR=MOD.TIL;
           PER.KIM=MOD.PIM;
           PER.JIM=MOD.TIM;
           LOST=PP;
           LILE=1;
           LIME=1;
           ALLOCATE  PROP;
           PROP.TIPO=5;
           IB=IB+1;
            STORK=PT;
           BOST(IB)=STORK;
           DO L=1 TO WEST;
           AT=PER.KIM(L);
           IF (AT¬=NULL) THEN AT->PROP.ROOT=STORK;
           END;
           PROP.NAME=MOD.NAME;
           PROP.VALUE=MOD.VALUE;
           PROP.TIL=0;
           PROP.TIM=0;
           PROP.PIM=NULL;
           PUT EDIT ('SYMM MODULE NAME=',PROP.NAME,'VALUE=',
           PROP.VALUE) (SKIP(2),A(17),F(5),X(7),A(6),F(5));
           PROP.HOST=LOST;
           LARG=NUM+NEZT;
           KAY=(PROP.VALUE-LARG)/100;
           CALL SYMM;
```



$$\begin{array}{l}
\text{MOD.TIL(1)} \rightarrow C_1 \\
\quad\vdots \qquad\qquad\quad \vdots \\
\text{MOD.TIL(r)} \rightarrow C_r \\
\text{MOD.TIM(1)} \rightarrow M_1 \\
\quad\vdots \\
\text{MOD.TIM(s)} \rightarrow M_s
\end{array}$$

$(r + s = n)$

FIGURE 3.25

EXPLICITLY SYMMETRIC MODULAR GATE

```
2  1          LOST->HECTOR=QUEEN;
2  1           PUT EDIT ('DEP COMPS=') (SKIP(1),A(10));
2  1          PUT LIST(PER.TAR);
2  1          PUT EDIT('DEP MODS=') (SKIP(1),A(9));
2  1          PUT LIST (PER.JIM);
2  1          PUT EDIT ('MINIMAL CUT SETS') (SKIP(2),X(12),A(16));
2  1          VIT=PER.HECTOR;
2  1          DO WHILE (VIC¬=NULL);
2  2          VIC=VIT;
2  2          PUT EDIT (VIC->COMP) (SKIP(1),P);
2  2          VIT=VIC->FLOOR;
2  2          END;
2  1          GO TO CANA;
```

It should be noticed here that for a symmetric gate, the role played by its free leaf inputs corresponds to that of the replicated inputs for a higher order module since

PER.TAR(1) = MOD.TIL(1) I = 1,...,MOD.LIM

At the same time its modular inputs (MOD.TIM(J)) will play the role which corresponds to the nested gate PROP structures for a higher order module since

PER.KIM(J) = MOD.PIM(J)   J = 1,...,MOD.MIM
PER.PIM(J) = MOD.TIM(J)

As a result the PROP structure associated with a symmetric gate will have no direct inputs (PROP.TIL = 0, PROP.TIM = 0).

For the pressure tank fault tree example gate G9 is a 2-out of-3 symmetric gate. Its MOD structure was given in section III.8 as

1 MOD BASED (MT_2)

2 TIPO = 4,

2 NAME = 9

2 VALUE = 203,

```
2 NEST = 0,
2 LIM = 3,
2 RIM = 1,
2 RIMO = 0,
2 MID = 1,
2 NAIL(1) = NULL,
2 WHIP(1) = NULL,
2 TIR(1) = 0,
2 PID(1) = NULL,
2 TID(1) = 0,
2 PIM(1) = NULL,
2 TIM(1) = 0
2 TIL(1) = 11, TIL(12), TIL(13) = 13;
```

So for this particular example the Boolean state vector include no modular inputs (since MOD.TIM = 0) but only basic component events (MOD.TIL(1), I = 1,2,3).

The PROP and PER structure associated with gate G9 are

```
1 PROP BASED (PT_1),
2 TIPO = 5,
2 REZ = 2,
2 ROOT POINTER,
2 NAME = 9,
2 VALUE = 203,
2 LIM = 1,
2 MIM = 1,
2 HOST = PR_1
```

```
2 REL(2) FLOAT,
2 TIL(1) = 0,
2 TIM(1) = 0,
2 PIM(1) = NULL;
```

(PROP.ROOT will later be assigned the pointer locating the PROP structure for gate G4.)

```
1 PER BASED ($PR_1$)
2 REZ = 2,
2 HECTOR POINTER,
2 DEXTER POINTER,
2 RAM = 3,
2 LEAL = 1,
2 REL(2)  FLOAT,
2 TAR(1) = 11, TAR(2) = 12, TAR(3) = 13,
2 KIM(1) = NULL,
2 JIM(1) = 0;
```

Procedure SYMM, outlined by the statements given below, will generate the set of VECTOR structures for a symmetric gate given the values of LARG = NUM +NEZT and KAY = (PROP.VALUE - LARG)/100.

```
                     /*        SYMMETRIC     GATES      */
 2  1      SYMM:  PROC:
 3  1             ALLOCATE SOF;
 3  1             ALLOCATE   TOD;
 3  1             ALLOCATE  VECTOR;
 3  1             QUEEN=VT;
 3  1              SOF=REPEAT('0'B,LARG);
 3  1             SUBSTR(SOF,LARG,1)='1'B;
 3  1             VECTOR.COMP=SOF;
 3  1             LADY=VT;
 3  1             DO  I=1  TO LARG-3 ;
 3  2             ALLOCATE  VECTOR;
 3  2             LADY->FLOOR=VT;
 3  2             LADY=VT:
```

```
3  2          SOP=REPEAT('0'B,LARG);
3  2          SUBSTR(SOP,LARG-I,1)='1'B;
3  2          VECTOR.COMP=SOP;
3  2          END;
3  1          ALLOCATE  VECTOR;
3  1          LADY->FLOOR=VT;
3  1          VECTOR.FLOOR=NULL;
3  1          SOP=REPEAT('0'B,LARG);
3  1          SUBSTR(SOP,2,1)='1'B;
3  1          VECTOR.COMP=SOP;
```

Up to here, SYMM has created a set of LARG-1 vectors which contain a single '1' bit component. Consider for example a 3-out of-5 symmetric gate, then PROP.VALUE = 305, LARG = 5 => KAY = 3 and the vectors created are

1 VECTOR BASED (VT_1),

2 LORO = 5,

2 FLOOR = VT_2,

2 COMP = '00001'B;

(QUEEN = VT_1)

1 VECTOR BASED (VT_2),

2 LORO = 5,

2 FLOOR = VT_3,

2 COMP = '00010'B;

1 VECTOR BASED (VT_3),

2 LORO = 5,

2 FLOOR = VT_4

2 COMP = '00100'B;

1 VECTOR BASED (VT_4)

2 LORO = 5,

2 FLOOR = NULL,

2 COMP = '01000' B;

The minimal cut-sets for the 3- out of -5 gate are then found by adding '1' bits in any position to the left of the place where the first '1' bit is found, and by successively repeating this operation KAY-1 times requiring that each final vector include a total of KAY (=3) bits

Initial Vectors	'00001' B
	'00010' B
	'00100' B
	'01000' B
Vectors After 1st Iteration	'00011' B
	'00101' B
	'01001' B
	('10001' B) Cancelled out
	'00110' B
	'01010' B
	('10010' B) Cancelled out
	'01100' B
	('10100' B) Cancelled out
	('11000' B) Cancelled out

<u>Minimal cut-set vectors</u>
found after 2nd iteration

'00111' B
'01011' B
'10011' B
'01101' B
'10101' B

'11001' B

'01110' B

'10110' B

'11010' B

'11100' B

The following DO loop performs this operation (function INDEX (VECTOR.COMP, '1'B) yields the number location for the first element of the string matching substring '1'B, e.g., INDEX ('01101' B, '1'B) = 2).

```
3  1              DO I=2  TO  KAY:
3  2              LADY=QUEEN;
3  2              DO WHILE (LADY¬=NULL);
3  3        ST1:      VT=LADY;
3  3              J=INDEX(VECTOR.COMP,'1'B);
3  3              IF  J=1  THEN DO;
3  4              IF  LADY=QUEEN THEN DO;
3  5              QUEEN=LADY->FLOOR;
3  5              FREE VECTOR;
3  5              LADY=QUEEN;
3  5              END;
3  4              ELSE DO;
3  5              MOAN->FLOOR=LADY->FLOOR;
3  5              FREE VECTOR;
3  5              LADY=MOAN->FLOOR;
3  5              END;
3  4              END;
3  3              ELSE  DO;
3  4               TOD=VECTOR.COMP;
3  4              DO L=1 TO J-1;
3  5              ALLOCATE  VECTOR;
3  5              IF L=1  THEN KING=VT;
3  5              ELSE PAWN->FLOOR=VT;
3  5              SOF=REPEAT('0'B,LARG);
3  5              SUBSTR(SOF,L,1)='1'B;
3  5              VECTOR.COMP=SOF|TOD;
3  5              PAWN=VT;
3  5              PAWN->FLOOR=NULL;
3  5              END;
3  4              IF LADY=QUEEN THEN DO;
3  5              QUEEN =KING;
3  5              PAWN->FLOOR=LADY->FLOOR;
3  5               MOAN=PAWN;
3  5               LADY=PAWN->FLOOR;
3  5              END;
3  4              ELSE DO;
3  5              MOAN->FLOOR=KING;
3  5               PAWN->FLOOR=LADY->FLOOR;
3  5              MOAN=PAWN:
```

```
3   5          LADY=PAWN->FLOOR;
3   5          END;
3   4          END;
3   3          END;
3   2          END;
3   1          FREE SOF;
3   1          FREE TOD;
3   1          END  SYMM;
                 /*       END    OF    SYMMETRIC */
```

For the pressure tank fault tree, procedure SYMM will thus yield the following vectors associated with gate G9.

1 VECTOR (VT_1),
2 LORO = 3,
2 FLOOR = VT_2,
2 COMP = '011' B;
1 VECTOR (VT_2),
2 LORO = 3,
2 FLOOR = VT_3,
2 COMP = '101'B;
1 VECTOR (VT_3),
2 LORO = 3,
2 FLOOR = NULL,
2 COMP = '110' B;

with $PR_1 \rightarrow$PER.HECTOR = VT_1.

III.9.3. Procedure BOOLEAN

The generation of a Boolean vector representation for a higher order module, composed of a set of replicated events and nested modules, is a quite complicated task as compared with that of finding a Boolean representation for an explicitly symmetric gate. PL-MOD's capability of handling higher order symmetric gates (Figure 3.26) in an explicit fashion is therefore a very desirable feature, since considerable savings will result by using this option for the analysis of systems containing a large number of symmetric redundencies.

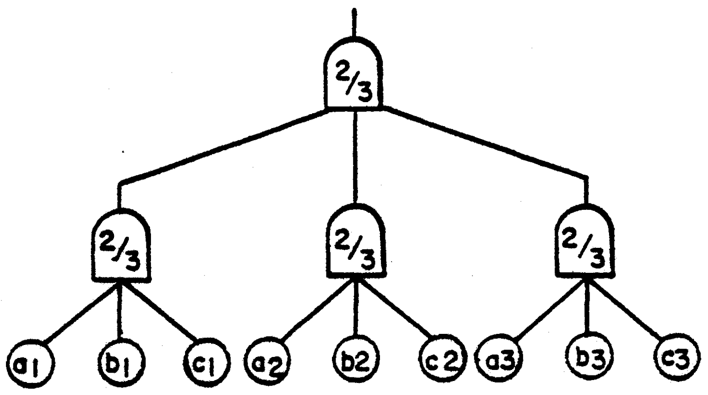
In general, however, fault trees will be composed of higher order modules whose structural composition needs to be found. For these cases it will be necessary to call upon BOOLEAN to generate a minimal cut-set representation for the higher order module.

Consider the pressure tank fault tree example. Up to this point it has been shown how PL-MOD internally represents gate G9 as a PROP structure ($PT_1 \rightarrow$ PROP) and gate G5 as a nested MOD structure ($MT_1 \rightarrow$ MOD). The following set of internal transformations still need to be performed by PL-MOD before the modularization for the full tree has been completed:

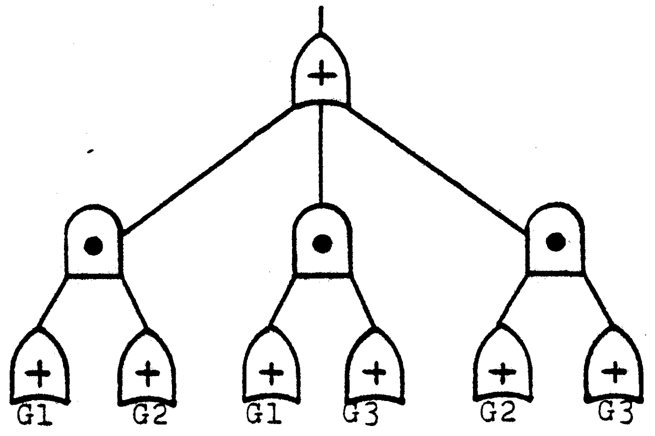
(a) G5 and G9 become nested module (MOD) and proper module (PROP) entries to a MOD structure associated with G4

```
1 MOD BASED (MT3),
2 TIPO = 4,
2 NAME = 4,
2 VALUE = 1,
2 NEST = 1,
2 LIM = 1,
2 RIM = 1,
2 RIMO = 2,
2 MIM = 1,
```

EXPLICIT FORM



IMPLICIT FORM



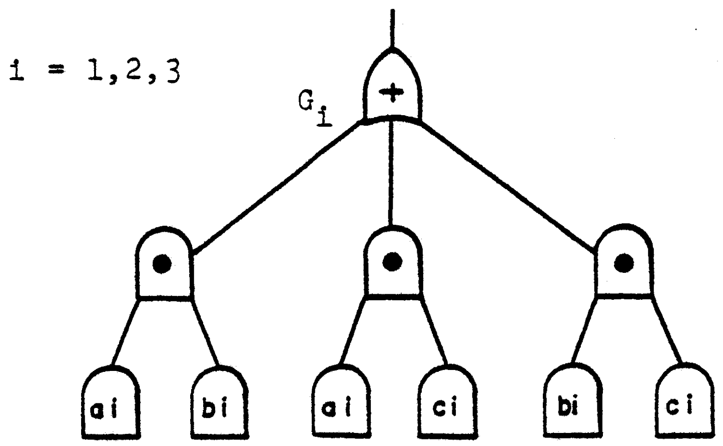


FIGURE 3.26

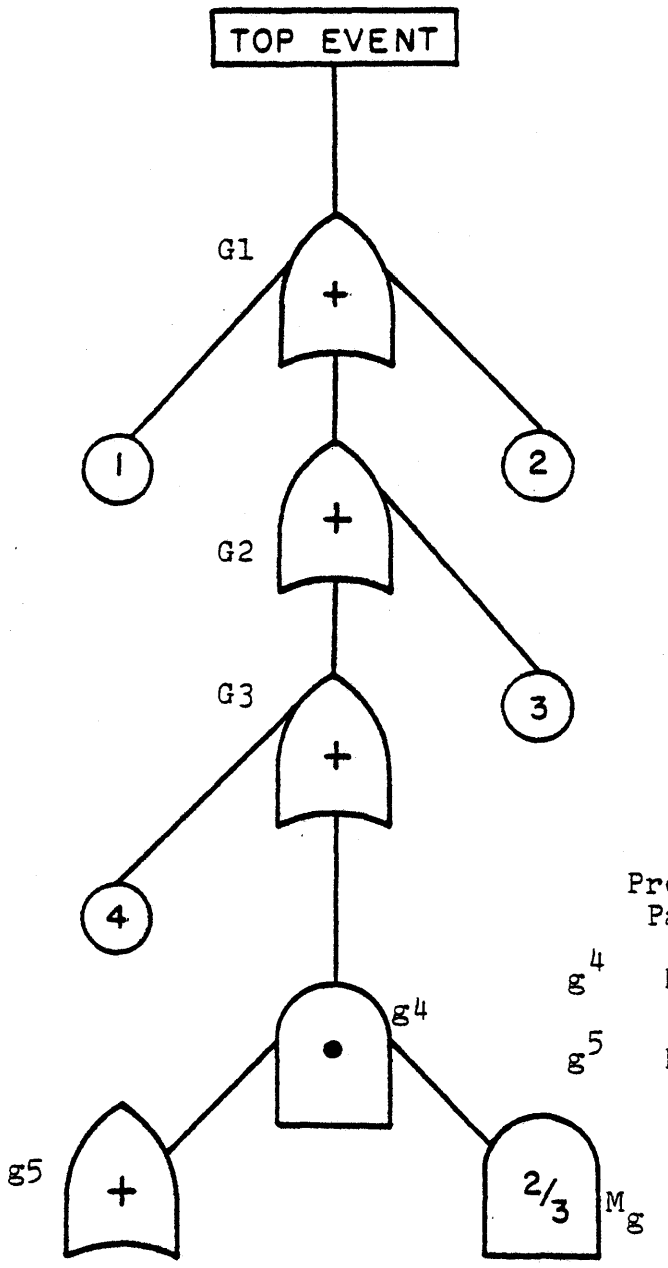
SYMMETRIC HIGHER ORDER MODULES

```
2 MID = 1,
2 NAIL(1) = SPINE(3),NAIL(2) = MT_3,
2 WHIP(1) = MT_3, WHIP(2) = APT_1,
2 TIR(1) = 0,
2 PID(1) = MT_1,
2 TID(1) = 5,
2 PIM(1) = PT_1,
2 TIM(1) = 9,
2 TIL(1) = 0;
```

Since MOD.NAIL(I) = MT_3 is not satisfied for I = 1, then gate G4 does not correspond to a higher order module, so structures $MT_1 \rightarrow$MOD (given in section III.8) and $MT_3 \rightarrow$MOD must be kept in the same form until the parent gate for the higher order module to which they belong is found (Figure 3.27).

(b) G3 will become a gateless node once G4 is attached to it as a STID structure. Furthermore, since gates G1, G2 and G3 are all of the same type, procedure COALESCE will collapse them together (Figure 3.28). The NODE structure representing G1 will then be given by

```
1 NODE BASED (VT = SPINE(1)),
2 TIPO = 1,
2 NAME = 1,
2 VALVE = 2,
2 GINT = 0,
2 LILT = 4,
2 LIRT = 1,
```



	Proper Part	Improper Part
g^4	M_4	g5
g^5	M_5	r_{30001}

FIGURE 3.27

PRESSURE TANK FAULT TREE WITH GATES G4,G5,G9 MODULARIZED

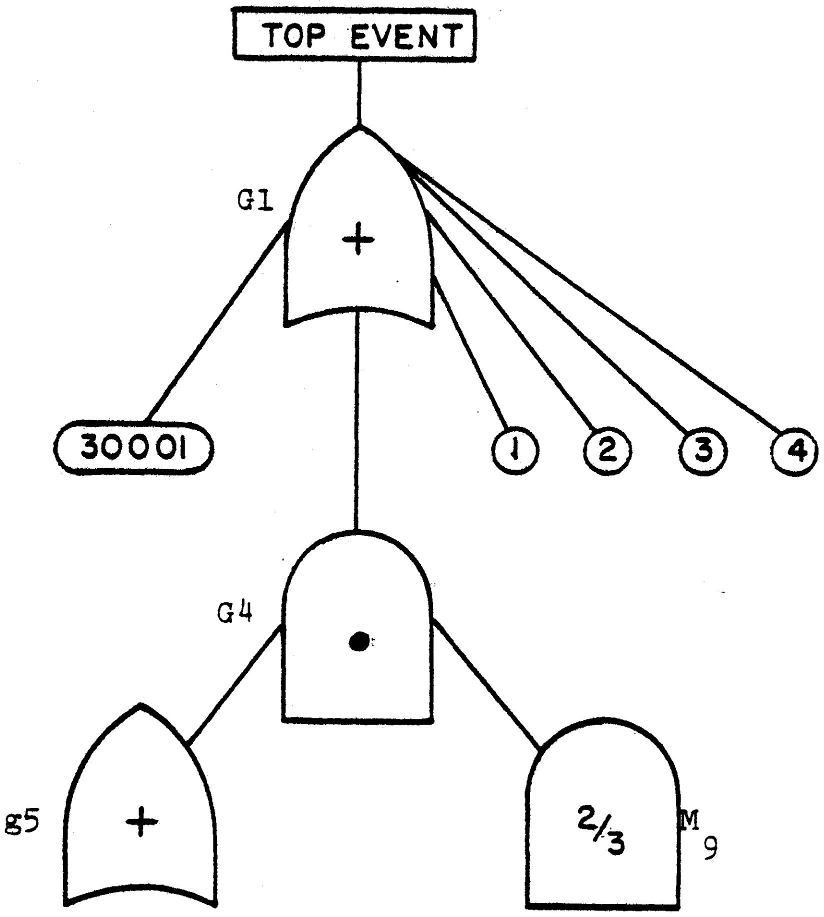


FIGURE 3.28

PRESSURE TANK FAULT TREE WITH GATES G4, G5 AND G9 MODULARIZED AND GATES G1, G2, G3 COALESCED

```
2 LIMD = 1,
2 LIMT = 0,
2 NEST = 2,
2 WHIZ = 1,
2 ROOT = NULL,
2 LIP = ST4,
2 LID = SD2
2 GIN = 1,
2 LIL = 2,
2 DIR = 1,
2 NAIL(1) = NULL,
2 WHIP(1) = NULL,
2 TIR(1) = 0,
2 SPIT(1) = NULL,
2 TIL(1) = 1, TIL(2) = 2;
```

And the set of STIP and STID structures attached to the NODE are

```
1 STIP BASED (ST4),        (Represents gate G2)
2 TIPO = 2,
2 LIP = ST5,
2 DIL = 1,
2 DIR = 1,
2 NAIL(1) = NULL,
2 WHIP(1) = NULL,
2 TIR(1) = 0,
2 TIL(1) = 3;
```

```
1 STIP BASED (ST_5),      (Represents Gate G3)
2 TIPO = 2,
2 LIP = NULL,
2 DIL = 1,
2 DIR = 1,
2 NAIL(1) = ST_5,
2 WHIP(1) = SD_2,
2 TIR(1) = 30001,
2 TIL(1) = 4;

1 STID BASED(SD_2),      (Represents Gate G4)
2 TIPO = 3,
2 LID = NULL,
2 STIM = 4,
2 LTIM = MT_3,
2 DIR = 2,
2 NAIL(1) = ST_5, NAIL(2) = SD_2,
2 WHIP(1) = SD_2, WHIP(2) = APT_1,
```

(c) Procedure MODULA will then create a MOD structure to represent SPINE(1) NODE including its attached STID and STIP structures

```
1 MOD BASED (MT_4),
2 TIPO = 4,
2 NAME = 1,
2 VALUE = 2,
2 NEST = 2,
2 LIM = 4,
2 RIM = 1,
2 RIMO = 3,
2 MIM = 1,
2 MID = 1,
2 NAIL(1) = MT4, NAIL(2) = MT4, NAIL(3) = MT4,
2 WHIP(1) = MT4, WHIP(2) = MT4, WHIP(3) = APT1,
2 TIR(1) = 30001,
2 PID(1) = MT3,
2 TID(1) = 4,
2 PIM(1) = NULL,
2 TIM(1) = 0,
2 TIL(1) = 1, TIL(2) = 2, TIL(3) = 3, TIL(4) =4,
```

Inspection of the MOD structure shows that the criterion

$$(I = 1,2,3) \qquad \begin{array}{l} \text{MOD.NAIL(I)} = MT_4 \\ \text{MOD.WHIP(I)} = MT_4 \text{ or } APT_1 \end{array}$$

is met. Therefore BOOLEAN must derive a representation for the higher order module associated with $MT_4 \rightarrow$MOD.

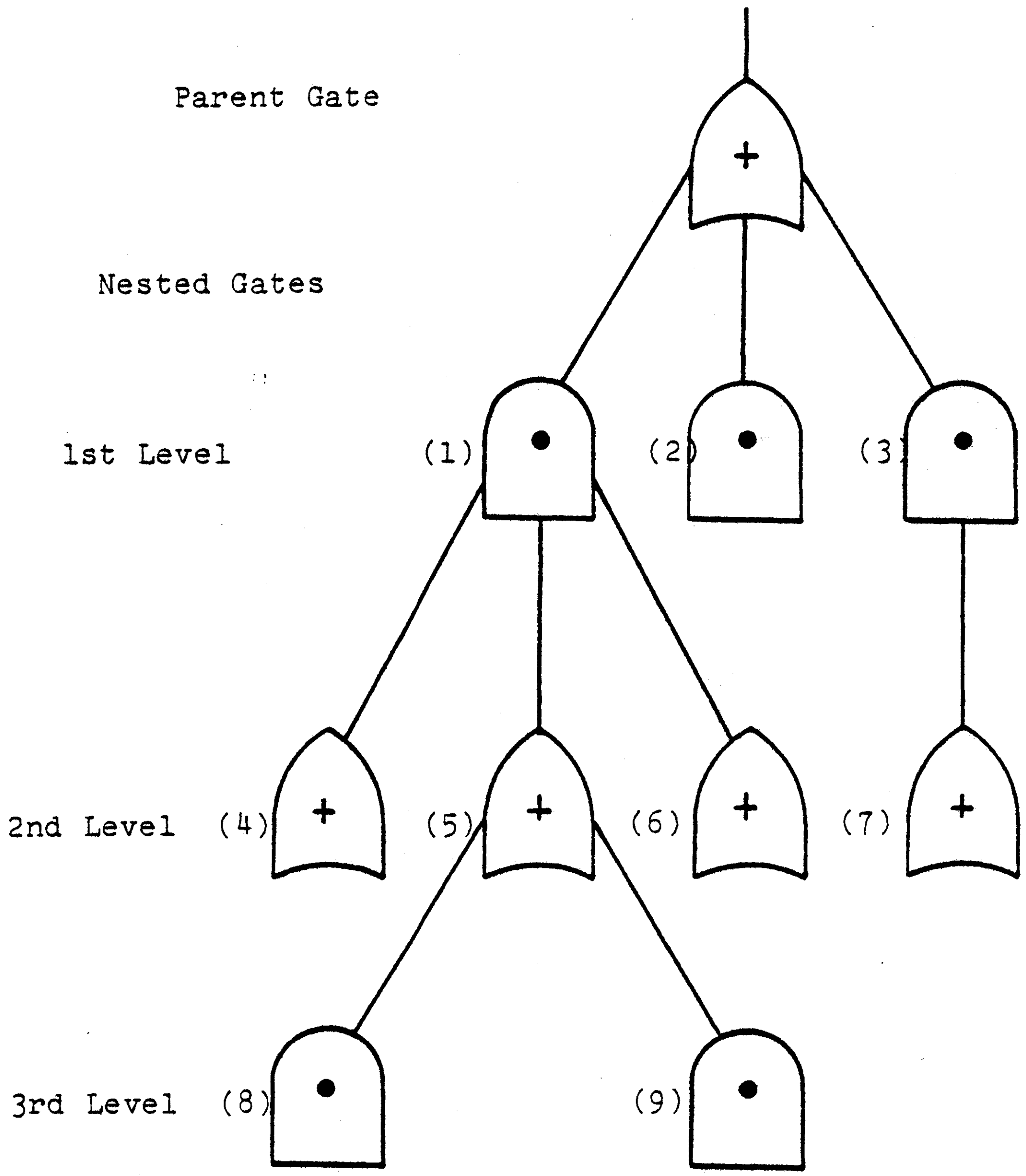
Procedure BOOLEAN starts off by creating the PROP structures associated with the parent gate and each nested gate, as well as the PER structure containing the structural information for the higher order module

```
          /*   BOOLEAN SUBROUTINE*/
(CHECK(WEST,LEG,EST,LOG,MEG,LARG,B1,FOX,B3,C1,C2,
 FOG,XOD,C1C,XOG,C1Z,C2M,C2Z,KOF,KOD,TOD,DOTT,
 MICS,SPU4)):
 BOOLEAN: PROC;
   PUT SKIP LIST('BOOLEAN HAS BEEN CALLED');
          MT=MODUL.DULL(M);
         WEST=MOD.WEST;
          JEST=WEST+1;
         NUB=NUM;
         ALLOCATE PER;
         PER.TAR=PUT;
         FREE PUT;
         LOST=PR;
          ALLOCATE PEN;
         FROST=PN;
                LILF=MOD.LIM;
                LIMF=MOD.MIM;
         ALLOCATE PROP;
         PROP.TIPO=5;
         IB=IB+1;
         STORK=PT;
         BOST(IB)=STORK;
         PROP.NAME=MOD.NAME;
         PROP.VALUE=MOD.VALUE;
         PROP.TIL=MOD.TIL;
         PROP.TIM=MOD.TIM;
         PROP.PIM=MOD.PIM;
 PUT EDIT ('PARENT MODULE NAME=', PROP.NAME,'VALUE=',
  PROP.VALUE, 'NUM LEAF INP=', PROP.LIM, 'NUM MOD INP=', PROP.MIM)
  (SKIP(2),A(19),F(5),X(2),A(6),F(5),X(2),A(13),F(5),X(2),A(12),
                                                               F(5));
 PUT  EDIT ('LEAF INS=') (SKIP(1), A(9));
 PUT LIST( PROP.TIL);
  PUT EDIT('MOD INS=') (SKIP(1),A(8));
  PUT LIST(PROP.TIM);
         PROP.HOST=LOST;
         FOG=MOD.MID;
         DO I=1 TO FOG;
         PEN.KIN(I)=MOD.PID(I);
         PEN.JIN(I)=MOD.TID(I);
          END;
         LEG=FOG;
         ALLOCATE DRUG;
         FROG=MOD.PID;
         ZEG=FOG;
         GREG=DR;
         GROG=1;
         EST=0;
          GREY=DR;
         DO WHILE (GROG¬=0);
         LOG=0;
         DO K=1 TO MEG;
         IF (FROG(K)->PID(1)=NULL) THEN PEG=0;
         ELSE PEG=1;
          LOG=LOG+PEG;
         END;
          IF(LOG=0) THEN GROG=0;
          WER=ZEG;
 3  2          DR=GREY;
 3  2          DO Q=1 TO MEG;
 3  3         MT=FROG(Q);
 3  3         LILE=MOD.LIM;
 3  3         LIME=MOD.MIM;
 3  3         ALLOCATE PROP;
 3  3         PROP.TIPO=5;
 3  3         ARI=PT;
 3  3         IB=IB+1;
 3  3         BOST(IB)=PT;
 3  3          EST=EST+1;
 3  3         PER.KIM(EST)=PT;
 3  3         PER.JIM(EST)=MOD.NAME;
 3  3         PROP.NAME=MOD.NAME;
 3  3         PROP.VALUE=MOD.VALUE;
 3  3         PROP.TIL=MOD.TIL;
 3  3         PROP.TIM=MOD.TIM;
 3  3         PROP.PIM=MOD.PIM;
 3  3         PROP.ROOT=STORK;
 3  3         DO L=1 TO LIME;
 3  4         AT=PROP.PIM(L);
 3  4         IF (AT¬=NULL) THEN AT->PROP.ROOT=ARI;
 3  4              END;

 3  3     PUT EDIT ('NESTED MODULE NAME=', PROP.NAME,'VALUE=',
           PROP.VALUE, 'NUM LEAF INP=', PROP.LIM, 'NUM MOD INP=', PROP.MIM)
          (SKIP(2),A(19),F(5),X(2),A(6),F(5),X(2),A(13),F(5),X(2),A(12),
                                                                   F(5));

 3  3     PUT  EDIT ('LEAF INS=') (SKIP(1), A(9));
 3  3     PUT LIST( PROP.TIL);
 3  3      PUT EDIT('MOD INS=') (SKIP(1),A(8));
 3  3      PUT LIST(PROP.TIM);
 3  3           PROP.HOST=NULL;
 3  3         FOG=MOD.MID;
 3  3         IF( FOG=1 & MOD.PID(1)=NULL) THEN GO TO UNO;
 3  3          DO I=1 TO FOG;
 3  4          PEN.KIN(ZEG+I)=MOD.PID(I);
 3  4         PEN.JIN(ZEG+I)=MOD.TID(I);
 3  4           END;
 3  3         ZEG=ZEG+FOG;
 3  3     UNO:    END;
 3  2         FREE DRUG;
 3  2         LEG=ZEG-WER;
 3  2         ALLOCATE DRUG;
 3  2         GREY=DR;
 3  2         DO ID=1 TO LEG;
 3  3         DRUG.FROG(ID)=PEN.KIN(WER+ID);
 3  3           END;
 3  2         END;
```

PROP structures are allocated starting at the top with the parent gate and then proceeding to successively deeper levels of nested gate modules in the higher order structure. Figure 3.29 shows an example of a higher order module consisting of 3 levels of nested gates. In the diagram only the nested gates of the structure are portrayed and all other input details to the higher order module have not been included



Allocation order is given by (i), i=1,2,...,9

FIGURE 3.29

ORDERING OF PROP STRUCTURE ALLOCATIONS FOR A HIGHER ORDER MODULE (i.e., replicated inputs and proper modular inputs to each gate).

BOOLEAN succeeds to allocate the PROP structures in the desired order with the help of a set of DRUG structures which contain the pointer locations for each of the MOD structures at a given nested gate level. Structure DRUG is defined by

```
1   DRUG BASED (DR)
2   MEG FIXED BINARY,
2   FROG (LEF REFER(MEG)) POINTER;
```

Thus, for the example given in Figure (3.29), three DRUG structures would be needed by BOOLEAN

1 DRUG BASED (DR_1),

2 MEG = 3,

2 FROG(1) = MT_1, FROG(2) = MT_2. FROG(3) = MT_3;

---

1 DRUG BASED (DR_2),

2 MEG = 4,

2 FROG(1) = MT_4, FROG(2) = MT_5,
FROG(3) = MT_6, FROG(4) = MT_7;

---

1 DRUG BASED (DR_3)

2 MEG = 2,

2 FROG(1) = MT_8, FROG(2) = MT_9;

Where this notation means that MT_i locates the MOD structure associated with the (i-th) nested gate.

While the name and pointer location for each nested gate PROP structure are stored in PER.JIM(I) and PER.KIM(I)

(I = 1,2,...,WEST), the name and pointer location for the MOD structure associated with each nested gate are stored in the structure PEN defined by

```
1 PEN BASED (PN)
2 LEAL FIXED BINARY,
2 KIM (WEST REFER(PEN.LEAL)) POINTER,
2 JIN (WEST REFER (PEN.LEAL)) FIXED;
```

The higher order modular structure composition for the pressure tank fault tree example is quite simple, since only two nested gate levels exist each consisting of a single gate (Figure 3.30). Its PROP, PER and PEN structures are given by

1 PROP BASED (PT_2),

2 TIPO = 5,

2 REZ = 2,

2 ROOT = NULL,

2 NAME = 1,

2 VALUE = 2,

2 LIM = 4,

2 MIM = 1,

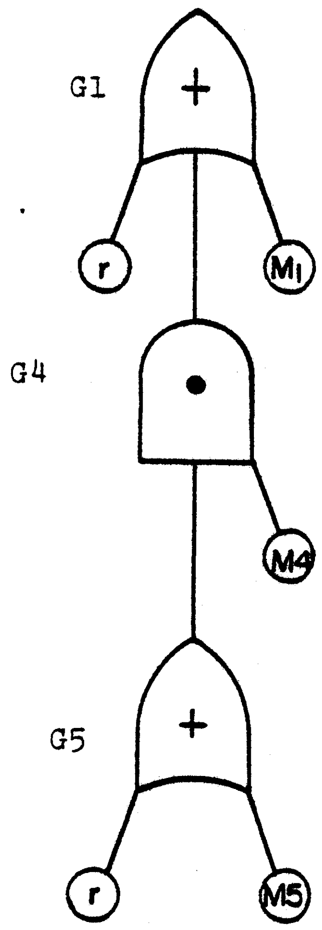
2 HOST = PR_2,

2 REL(2) FLOAT,

2 TIL(1) = 1, TIL(2) = 2, TIL(3)=3,TIL(4)=4,

2 TIM(1) = 0,

2 PIM(1) = NULL;



Proper Part	Improper Part
M_1 = {C1, C2,C3,C4;U}	r_{30001}, G4
M4 = {M_9}	G5
M_5 = {C_5,C_6,C_7,C_8,C_9,C_{10};U}	r_{30001}

(M_9 = {C_{11},C_{12},C_{13}; 2-out of-3 operator})

FIGURE 3.30

HIGHER ORDER MODULAR COMPOSITION FOR THE PRESSURE TANK FAULT TREE

```
1 PROP BASED (PT3),
2 TIPO = 5,
2 REZ = 2,
2 ROOT = PT2,
2 NAME = 4,
2 VALUE = 1,
2 LIM = 1,
2 MIM = 1,
2 HOST = NULL,
2 REL(2)  FLOAT,
2 TIL(1) = 0,
2 TIM(1) = 9,
2 PIM(1) = PT1;

1 PROP BASED (PT4),
2 TIPO = 5,
2 REZ = 2,
2 ROOT = PT2
2 NAME = 5,
2 VALUE = 2,
2 LIM = 6,
2 MIM = 1,
2 HOST = NULL,
2 REL(2)  FLOAT,
2 TIL(1)=5,TIL(2)=6,TIL(3)=7,TIL(4) = 8,
  TIL(5)=9, TIL(6)=10,
2 TIM(1) = 0,
2 PIM(1) = NULL;

1 PER BASED (PR$_2$)
2 REZ = 2,
2 HECTOR POINTER,
2 DEXTER POINTER,
2 RAM = 1,
2 LEAL = 2,
2 REL(2) FLOAT,
2 TAR(1) = 30001,
2 KIM(1) = PT$_3$, KIM(2) = PT$_4$,
2 JIM(1) = 4, JIM(2) = 5;

1 PEN BASED (PN$_1$)
2 LEGAL = 2,
2 KIN(1) = MT$_3$, KIN(2) = MT$_1$,
2 JIN(1) = 4, JIN(2) = 5;
```

Once BOOLEAN has mapped out the structural composition for the higher order module, it is then ready to proceed to generate the set of VECTOR structures representing the modular minimal cut-sets for the higher order structure.

The process by which each minimal cut-set VECTOR is found, is a recursive one. By starting with a Boolean representation for the parent gate given in terms of its improper modular inputs (MOD structures), each of the nested gates are explicitly incorporated by making a set of substitutions consistent with the structural relationship each nested gate holds with the parent gate. Ultimately each minimal cut-set is given by a VECTOR structure of dimension LARG = NUB + 1 + WEST, where NUB = total number of replicated event inputs to the higher order module and WEST = total number of nested gates contained by the higher order module. That is

$$Y^B = (y_1, y_2, \ldots, y_\ell) \quad (\ell = \text{LARG})$$

the order in which each of the inputs to the higher order module is entered is given by

$$Y^B = (y_{r_1}, y_{r_2}, \ldots, y_{r_n}, y_{m_o}, y_{m_1}, \ldots, y_{m_n})$$

with r_i = replicated input i, n = NUB, m_o = parent gate PROP input, m_i = ith nested gate PROP input, w = WEST, $n + 1 + w = \ell$.

However, as discussed earlier BOOLEAN derives this set of VECTORS by a series of substitutions of improper modules (MOD structures) by their replicated input (r-leaf) and proper input (PROP) parts. Therefore in order to make this feasible BOOLEAN needs to perform a set of manipulations with a set of SECTOR based structures defined by

```
1 SECTOR BASED (SR),
2 LORO FIXED BINARY,
2 DOOR POINTER,
2 COD BIT (JUST REFER(SECTOR.LORO));
```

with JUST = LARG + WEST.

Every replicated input, PROP and MOD structure in the higher order module will be represented by a Boolean variable within each SECTOR structure in the following order

$$\underset{\rightarrow}{Z}^B = (y_{r_1},\ldots,y_{r_n},\ y_{m_o},\ y_{m_1},\ldots,y_{m_w},y_{d_1},\ldots,y_{d_w})$$

$$\underset{\rightarrow}{Z}^B = (\underset{\rightarrow}{y}^B,\ \underset{\rightarrow}{x}^B)$$

with $\underset{\rightarrow}{Y}^B$ containing the same inputs as a VECTOR bit-string and $\underset{\rightarrow}{X}^B = (y_{d_1},\ldots,y_{d_w})$ representing the nested MOD structures in the higher order module, i.e., d_i = ith nested gate MOD structure.

The minimal cut-set generation procedure is begun by finding the set of VECTOR and SECTOR structures which initially represent the parent gate. Figures 3.31 and 3.32 illustrate the two possible instances of higher order modules with an OR-operator or an AND-operator parent gate. For the OR-parent gate, example I, the full modular structure consists of five nested gates and two replicated events. Its VECTOR and SECTOR bit-strings will therefore have the form

$$\underset{\rightarrow}{Y}^B = (Y_{r_1},Y_{r_2},Y_{m_o},Y_{m_1},Y_{m_2},Y_{m_3},Y_{m_4},Y_{m_5})$$

$$\underset{\rightarrow}{Z}^B = (\underset{\rightarrow}{Y}^B,Y_{d_1},Y_{d_2},Y_{a_3},Y_{d_3},Y_{d_4},Y_{d_s}) = (\underset{\rightarrow}{Y}^B,\underset{\rightarrow}{X}^B)$$

and the parent gate shall be initially represented by

M_o=>Y_{mo} = 1

```
1 VECTOR BASED(VT1),
2 LORO = 8
2 FLOOR = NULL,
2 COMP = '00100000'B;
```

G_1 =>$Y_{d1=1}$

```
1 SECTOR BASED (SR1),
2 LORO = 13,
```

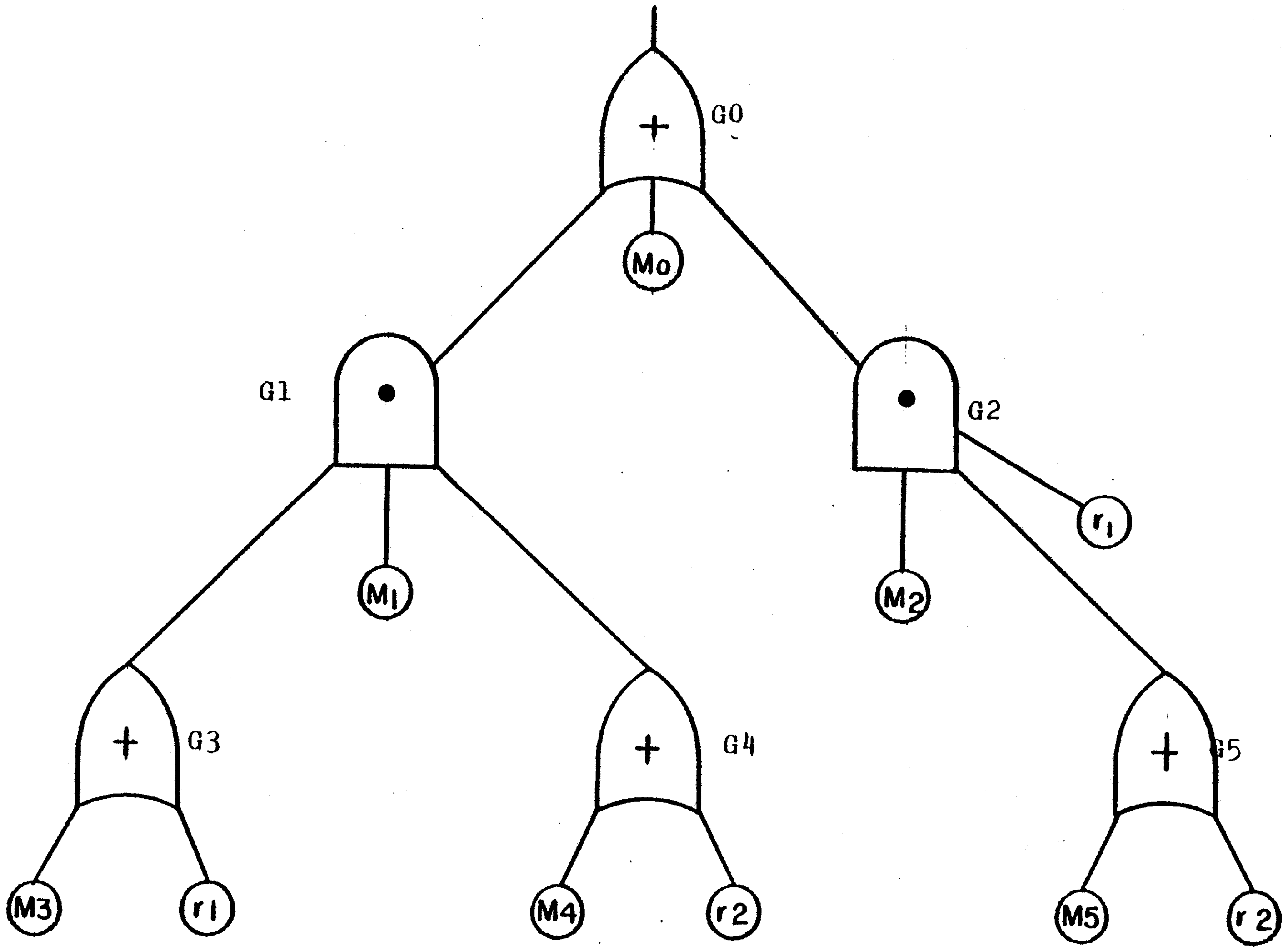


FIGURE 3.31 <u>OR</u>-PARENT GATE HIGHER ORDER MODULE EXAMPLE I

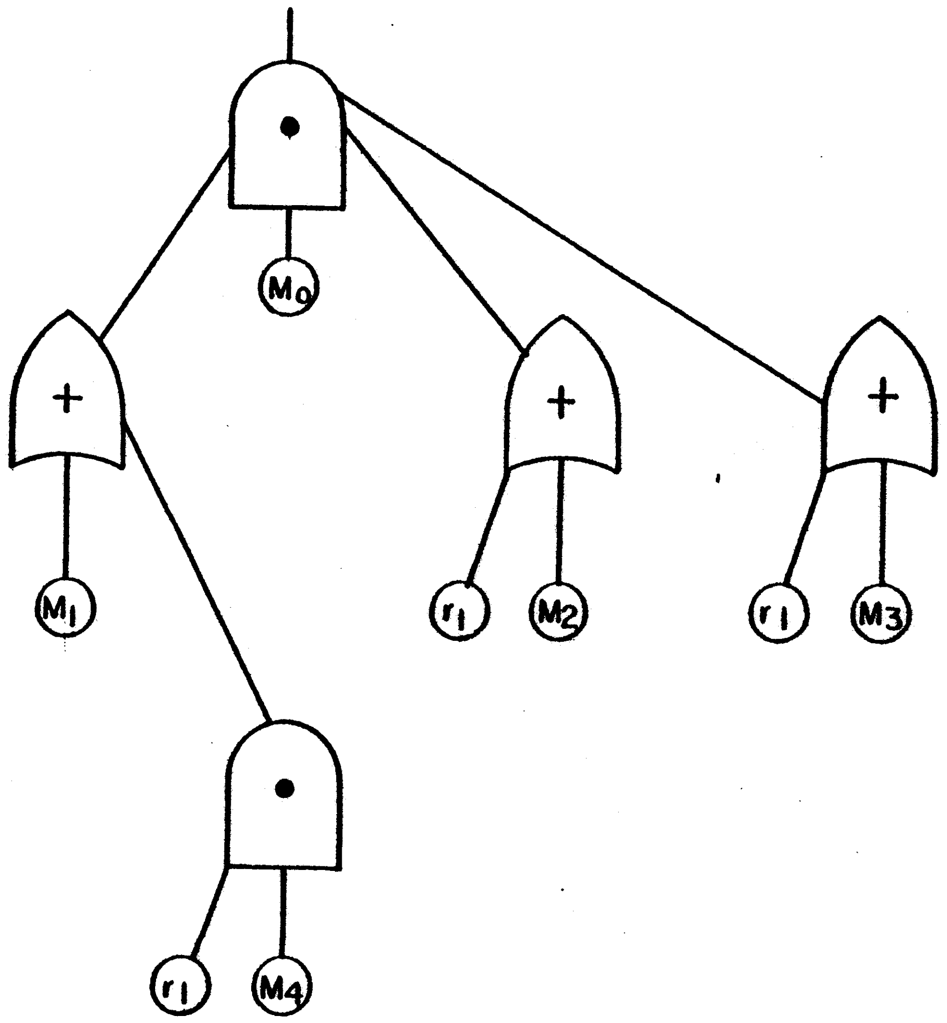


FIGURE 3.32 AND - PARENT GATE HIGHER ORDER EXAMPLE II

2 DOOR = SR_2

2 COMP = '$\underbrace{00000000}_{LARG}\overbrace{10000}^{WEST}$'B'

$G_2 \Rightarrow Y_{d2} = 1$ 1 SECTOR BASED (SR_2)

2 LORO = 13,

2 DOOR = NULL,

2 COMP = '$\underbrace{00000000}\overbrace{01000}$' B;

For the AND-parent gate, example II, a single SECTOR shall initially represent it. Since the full modular structure for example II consists of one replicated event and four nested gates then

$$Y^B = (Y_{r_1}, Y_{m_o}, Y_{m_1}, Y_{m_2}, Y_{m_3}, Y_{m_4})$$

$$Z^B = (\underset{\rightarrow}{Y}^B, Y_{d_1}, Y_{d_2}, Y_{d_3}, Y_{d_4}) = (\underset{\rightarrow}{Y}^B, \underset{\rightarrow}{X}^B)$$

so the initial representation for the parent gate shall be

1 SECTOR BASED (SR_1),

2 LORO = 10,

2 DOOR = NULL,

2 COMP = '$\underbrace{010000}_{LARG}1110$' B

$(Y_{m_o} = Y_{d_1} = Y_{d_2} = Y_{d_3} = 1)$

The following statements outline the method used by BOOLEAN to derive the initial parent gate Boolean representation for a higher order module. For the OR-parent gate case (MOD.VALUE=

OP=2) the statements following label B2 apply, while for AND-parent gates the statement following label B1 apply.

```
3   1          MT=MODUL.DULL(M);
3   1          LARG=NUB+WEST+1;
3   1          JUST=LARG+WEST;
3   1           ALLOCATE KOF;
3   1           ALLOCATE KOD;
3   1           ALLOCATE XOD;
3   1     ALLOCATE  TOD;
3   1          ALLOCATE  DOTT;
3   1          ALLOCATE  TOG;
3   1          ALLOCATE XOG;
3   1           OP=MOD.VALUE;
3   1          LADY=NULL;
3   1           LORD=NULL;
3   1           IF(OP=1) THEN GO TO B1;
3   1          IF (OP=2) THEN GO TO B2;
3   1        B1:  ALLOCATE SECTOR;
3   1           KING=SR;
3   1          SECTOR.DOOR=NULL;
3   1          SECTOR.COD=REPEAT('0'B,JUST);
3   1           TOG=REPEAT('0'B,JUST);
3   1          SUBSTR(TOG,NUB+1,1)='1'B;
3   1          SECTOR.COD=TOG;
3   1            FOX=MOD.RIM;
3   1          IF (FOX=1 & MOD.TIR(1)=0) THEN GO TO B1A;

3   1          DO Q=1 TO FOX;
3   2          TEST=MOD.TIR(Q);
3   2           DO R=1 TO NUB;
3   3          IF (TEST=PER.TAR(R)) THEN GO TO B1B;
3   3          END;
3   2     B1B:  TOG=REPEAT('0'B,JUST);
3   2          SUBSTR(TOG,R,1)='1'B;
3   2           SECTOR.COD=SECTOR.COD|TOG;
3   2          END;
3   1       B1A:     FOG=MOD.MID;
3   1          DO Q=1 TO FOG;
3   2          TOG=REPEAT('0'B,JUST);
3   2          SUBSTR(TOG,LARG+Q,1)='1'B;
3   2          SECTOR.COD=SECTOR.COD|TOG;
3   2          END;
3   1           ESTO=FOG;
3   1          GO TO B3;
3   1      B2:      ALLOCATE VECTOR;
3   1          QUEEN=VT;
3   1     TOD=REPEAT('0'B,LARG);
3   1          SUBSTR(TOD,LARG-WEST,1)='1'B;
3   1      VECTOR.COMP=TOD;
3   1           VECTOR.FLOOR=NULL;
3   1          LADY=QUEEN;
3   1          FOX=MOD.RIM;
3   1          IF (FOX=1 & MOD.TIR(1)=0) THEN GO TO B2A;
3   1          DO Q=1 TO FOX;
3   2          TEST=MOD.TIR(Q);
3   2           DO R=1 TO NUB;
3   3          IF (TEST=PER.TAR(R)) THEN GO TO B2B;
3   3          END;
```

```
3  2        B2B:         ALLOCATE VECTOR;
3  2            IF( Q=FOX) THEN VECTOR.FLOOR=NULL;
3  2            LADY->FLOOR=VT;
3  2            LADY=VT;
3  2       TOD=REPEAT('0'B,LARG);
3  2        SUBSTR(TOD,R,1)='1'B;
3  2        VECTOR.COMP=TOD;
3  2            END;
3  1        B2A:        FOG=MOD.MID;
3  1            ESTO=FOG;
3  1            DO Q=1 TO FOG;
3  2             ALLOCATE SECTOR;
3  2            IF (Q=FOG) THEN SECTOR.DOOR=NULL;
3  2             IF (LORD¬=NULL) THEN GO TO B2C;
3  2              KING=SR;
3  2            LORD=SR;
3  2            GO TO B2D;
3  2        B2C:       LORD->DOOR=SR;

3  2            LORD=SR;
3  2        B2D:    SECTOR.COD=REPEAT('0'B,JUST);
3  2            TOG=REPEAT('0'B,JUST);
3  2             SUBSTR(TOG,LARG+Q,1)='1'B;
3  2            SECTOR.COD=TOG;
3  2             END;
```

It should be noticed here that the SECTOR.COD bit strings associated with the parent gate imply a dependence on all nested gates contained within the higher order module. This dependence shows up through the non-zero entries in the X^B portion of Z^B the SECTOR.COD bit string (X^B = SUBSTR(SECTOR. COD, LARG + 1, WEST). The objective of BOOLEAN will now be to substitute for each improper modular entry in SECTOR.COD an equivalent set of replicated leaf, proper module and improper modular entries.

Thus, for the two examples given above their dependence on nested gate G1 may be eliminated (i.e., Y_{d_1} may be set to zero) as follows:

Example I: $G_1 = \{M_1 G_3 G_4; \Omega\}=$

$$\Rightarrow (Y_{d_1}) \rightarrow (Y_{m_1} = 1)\ \Omega (Y_{d_3} = 1) \Omega (Y_{d_4} = 1)$$

Hence $SR_1 \rightarrow$SECTOR.COD = '0000000010000'B is replaced by $SR_1 \rightarrow$SECTOR.COD = $'\underbrace{00010000}\overbrace{00110}'B$

Example II: $G_1 = \{M_1, G_4; U\}$

$$\Rightarrow (Y_{d_1} = 1) \rightarrow (Y_{m_1} = 1) U (Y_{d_4} = 1)$$

Hence $SR_1 \rightarrow$ SECTOR is replaced by the two new sectors with .

$$SR_1 \rightarrow \text{SECTOR.COD} = '\underbrace{011000}\overbrace{0110}'B$$

$$SR_2 \rightarrow \text{SECTOR.COD} = '\underbrace{010000}\overbrace{0111}'B$$

By continuing this process all nested gate improper dependencies that a SECTOR might have will eventually be eliminated. That is, ultimately all SECTORS generated will contain a null substring $\underset{\sim}{X}^B = \underset{\sim}{0}$, and therefore will have been transformed into Boolean Indicated cut-set VECTORS (BICS)[16].

An outline of the statements in Boolean which provide for the deduction of Boolean indicated cut-set VECTORS follows

```
 3  1      B3:     DO IL=1 TO WEST;
 3  2           MT=PEN.KIN(IL);
 3  2           OP=MOD.VALUE;
 3  2           PAWN=KING;
 3  2           XOD=REPEAT('0'B,JUST);
 3  2           XOG=REPEAT('0'B,JUST);
 3  2           SUBSTR(XOD,LARG+IL,1)='1'B;
 3  2           SUBSTR(XOG,NUB+IL+1,1)='1'B;
 3  2           KOF=REPEAT('0'B,JUST);
 3  2           KOD=REPEAT('0'B,JUST);
 3  2           IF(OP=1) THEN GO TO C1;
 3  2            IF (OP=2) THEN GO TO C2;
 3  2      C1:        FOX=MOD.RIM;
 3  2           IF (FOX=1 & MOD.TIR(1)=0) THEN GO TO C1A;
 3  2           DO Q=1 TO FOX;
 3  3           TEST=MOD.TIR(Q);
 3  3           DO R=1 TO NUB;
 3  4            IF (TEST=PER.TAR(R)) THEN GO TO C1B;
```

```
3   4              END;
3   3    C1B:    TOG=REPEAT('0'B,JUST);
3   3            SUBSTR(TOG,R,1)='1'B;
3   3            KOF=KOF|TOG;
3   3              END;
3   2     C1A:     FOG=MOD.MID;
3   2              IF (FOG=1 & MOD.TID(1)=0)  THEN GO TO C1C;
3   2            DO Q=1 TO FOG;
3   3            TOG=REPEAT('0'B,JUST);
3   3            SUBSTR(TOG,LARG+Q+ESTO,1)='1'B;
3   3            KOD=KOD|TOG;
3   3            END;
3   2       ESTO=ESTO+FOG;
3   2    C1C:    DO WHILE(PAWN¬=NULL);
3   3            SR=PAWN;
3   3            TOG=SECTOR.COD&XOD;
3   3            IF (TOG)  THEN GO TO C1K;
3   3            ELSE GO TO C1Y;
3   3    C1K:     SECTOR.COD=SECTOR.COD&(¬XOD);
3   3            SECTOR.COD=SECTOR.COD|KOD;
3   3            SECTOR.COD=SECTOR.COD|XOG;
3   3            SECTOR.COD=SECTOR.COD|KOF;
3   3            DOTT=REPEAT('0'B,WEST);
3   3            DOTT=SUBSTR(SECTOR.COD,LARG+1,WEST);
3   3            IF (DOTT¬='0'B) THEN GO TO C1Y;
3   3            ALLOCATE VECTOR;
3   3            IF (LADY=NULL) THEN QUEEN=VT;
3   3            ELSE LADY->FLOOR=VT;
3   3            LADY=VT;
3   3            VECTOR.FLOOR=NULL;
3   3            VECTOR.COMP=SUBSTR(SECTOR.COD,1,LARG);
3   3            IF(SR=KING) THEN KING=SECTOR.DOOR;
3   3            ELSE GO TO D1;
3   3            PAWN=KING;
3   3            FREE SECTOR;
3   3            IF (PAWN=NULL) THEN GO TO MICS;
3   3            GO TO C1Z;
3   3     D1:          PAWN=SECTOR.DOOR;
3   3              FREE SECTOR;
3   3              MOAN->DOOR=PAWN;
3   3            GO TO C1Z;
3   3     C1Y:     MOAN=PAWN;
3   3            PAWN=SECTOR.DOOR;
3   3     C1Z: END;
3   2            GO TO C2Z;
3   2    C2:       ALLOCATE SECTOR;
3   2            SECTOR.DOOR=NULL;
3   2            KONG=SR;
3   2            LERD=SR;
3   2            SECTOR.COD=XOG;
3   2            FOX=MOD.RIM;
3   2            IF (FOX=1 & MOD.TIR(1)=0)  THEN GO TO C2A;
3   2             DO Q=1 TO FOX;
3   3            TEST=MOD.TIR(Q);
3   3            DO R=1 TO NUB;
3   4             IF(TEST=PER.TAR(R))  THEN GO TO C2B;
3   4            END;
3   3    C2B:        ALLOCATE SECTOR;
3   3            SECTOR.DOOR=NULL;
3   3     C2C:       LERD->DOOR=SR;
3   3            LERD=SR;
3   3    C2D:   TOG=REPEAT('0'B,JUST);
3   3            SUBSTR(TOG,R,1)='1'B;
3   3            SECTOR.COD=TOG;
```

```
3   3           END;
3   2      C2A:      FOG=MOD.MID;
3   2          IF(FOG=1 & MOD.TID(1)=0)   THEN GO TO C2R;
3   2           DO O=1 TO FOG;
3   3           ALLOCATE SECTOR;
3   3            SECTOR.DOOR=NULL;
3   3       C2F:     LERD->DOOR=SR;
3   3           LERD=SR;
3   3      C2G:    SECTOR.COD=REPEAT('0'B,JUST);
3   3           TOG=REPEAT('0'B,JUST);
3   3           SUBSTR(TOG,LARG+O+ESTO,1)='1'B;
3   3           SECTOR.COD=TOG;
3   3           END;
3   2      C2E:      ESTO=ESTO+FOG;
3   2         C2U:   MOAN=NULL;
3   2           DO WHILE(PAWN¬=NULL);
3   3            SR=PAWN;
3   3            KOF=REPEAT('0'B,JUST);
3   3           TOD=REPEAT('0'B,LARG);
3   3           TOD=SUBSTR(SECTOR.COD,1,LARG);
3   3           SUBSTR(KOF,1,LARG)=TOD;
3   3           KOD=REPEAT('0'B,JUST);
3   3           DOTT=REPEAT('0'B,WEST);
3   3           DOTT=SUBSTR(SECTOR.COD,LARG+1,WEST);
3   3           SUBSTR(KOD,LARG+1,WEST)=DOTT;
3   3           TOG=KOD&XOD;
3   3           IF (TOG)  THEN GO TO C2K;
3   3           ELSE GO TO C2L;
3   3     C2K:   PEON=KONG;
3   3            LUTE=NULL;
3   3           KOD=KOD&(¬XOD);
3   3           DO WHILE (PEON¬=NULL);
3   4           ALLOCATE SECTOR;
3   4           SECTOR.DOOR=NULL;
3   4           SECTOR.COD=PEON->SECTOR.COD|KOF;
3   4           SECTOR.COD=SECTOR.COD|KOD;
3   4           DOTT=REPEAT('0'B,WEST);
3   4           DOTT=SUBSTR(SECTOR.COD,LARG+1,WEST);
3   4           IF (DOTT¬='0'B) THEN GO TO C2X;
3   4             ALLOCATE VECTOR;
3   4           IF (LADY=NULL) THEN QUEEN=VT;
3   4           ELSE LADY->FLOOR=VT;
3   4           LADY=VT;
3   4            VECTOR.FLOOR=NULL;
3   4           VECTOR.COMP=SUBSTR(SECTOR.COD,1,LARG);
3   4           FREE SECTOR;
3   4           GO TO C2Y;
3   4      C2X:       IF (LUTE=NULL) THEN KUNG=SR;
3   4            ELSE   LUTE->DOOR=SR;
3   4           LUTE=SR;
3   4       C2Y:    MOON=PEON;
3   4           PEON=MOON->SECTOR.DOOR;
3   4           END;
3   3           SR=PAWN;
3   3           IF  (LUTE=NULL & MOAN=NULL) THEN GO TO C2Q;
3   3           ELSE GO TO C2R;
3   3      C2Q:   IF (SECTOR.DOOR¬=NULL) THEN GO TO C2W;
3   3           FREE SECTOR;
```

```
PL/I OPTIMIZING COMPILER                    /*    MODULE     PROGRAM     */

   STMT LEV NT

    942   3  3          GO TO MICS;
    943   3  3      C2W:   PAWN=SECTOR.DOOR;
    944   3  3          KING=PAWN;
    945   3  3          FREE SECTOR;
    946   3  3          GO TO C2M;
    947   3  3     C2R:    IF (MOAN¬=NULL & LUTE¬=NULL) THEN GO TO C3A;
    948   3  3          ELSE GO TO C3B;
    949   3  3     C3A:       MOAN->DOOR=KUNG;
    950   3  3          LUTE->DOOR=SECTOR.DOOR;
    951   3  3          FREE SECTOR;
    952   3  3          MOAN=LUTE;
    953   3  3          PAWN=LUTE->DOOR;
    954   3  3           GO TO C2M;
    955   3  3      C3B:      IF (LUTE=NULL) THEN GO TO C3C;
    956   3  3          ELSE GO TO C3D;
    957   3  3     C3C:        PAWN=SECTOR.DOOR;
    958   3  3          FREE SECTOR;
    959   3  3          MOAN->DOOR=PAWN;
    960   3  3          GO TO C2M;
    961   3  3     C3D:       KING=KUNG;
    962   3  3          LUTE->DOOR=SECTOR.DOOR;
    963   3  3          FREE SECTOR;
    964   3  3           MOAN=LUTE;
    965   3  3          PAWN=LUTE->DOOR;
    966   3  3          GO TO C2M;
    967   3  3      C2L:      MOAN=PAWN;
    968   3  3          PAWN=SECTOR.DOOR;
    969   3  3      C2M:      END;
    970   3  2     C2Z:   END;
```

The step-by-step process by which BOOLEAN derives the VECTOR BICS for the pressure tank fault tree example, is as follows

Replicated inputs: r_{30001} =>NUB = 1

Parent gate G1, nested gates (G_4, G_5) =>

WEST = 2,   LARG = NUM + 1 + WEST = 4   JUST = 6

$\underset{\sim}{Y}^B = (Y_r, Y_{m1}, Y_{m4}, Y_{m5})$

$\underset{\sim}{Z}^B = (\underset{\sim}{Y}^B, \underset{\sim}{X}^B), \underset{\sim}{X}^B = (Y_{d4}, Y_{d5})$

Step 1) Parent gate Boolean representation

$(Y_{m1} = 1) U (Y_r = 1) U (Y_{d_4} = 1)$

1 VECTOR BASED (VT_1),

2 LORO = 4,

2 FLOOR = VT_2,

2 COMP = '0100'B;

1 VECTOR BASED (VT_2),

2 LORO = 4,

2 FLOOR = NULL,

2 COMP = '1000'B;

1 SECTOR BASED (SR_1),

2 LORO = 6,

2 DOOR = NULL,

2 COD = '000010'B;

Step 2) Eliminate second nested gate (G4) by the substitution $Y_{d4} = 1 \quad (Y_{m4}=1) \Omega (Y_{d5}=1)$

=> 1 SECTOR BASED(SR_1),

2 LORO = 6,

2 DOOR = NULL,

2 COD = '$\underline{0010}$01' B;

Step 3) Eliminate second nested gate (G5) by the substitution $Y_{d5=1} \rightarrow (Y_{m5=1}) U (Y_r=1)$

=> 1 SECTOR BASED (SR_1),

```
2 LORO = 6,
2 DOOR = SR_2,
2 COD = '001100'B;
1 SECTOR BASED(SR_2),
2 LORO = 6,
2 DOOR = NULL,
2 COD = '101000'B;
```

Since $\underset{\sim}{X}^B = \underset{\sim}{0}$ for both $SR_1 \rightarrow$ SECTOR.COD and $SR_2 \rightarrow$ SECTOR.COD, they may be replaced by two new vectors

```
1 VECTOR BASED (VT_3),
2 LORO = 4,
2 FLOOR = VT_4,
2 COMP = '0011'B;
```

(with $VT_2 \rightarrow$VECTOR.FLOOR = VT_3)

```
1 VECTOR BASED (VT_4),
2 LORO = 4,
2 FLOOR + NULL,
2 COMP = '1010'B;
```

Hence, the set of BICS for the pressure tank fault tree is

$$Y_1^B = (0,1,0,0)$$
$$Y_2^B = (1,0,0,0)$$
$$Y_3^B = (0,0,1,1)$$
$$Y_4^B = (1,0,1,0)$$

To obtain now the set of minimal cut-sets (MICS), it is only necessary to eliminate those BICS vectors containing a sub-set of non-zero elements which also form a BICS vector. For the pressure tank fault tree $\underset{\to}{Y}_2^B$ is contained in $\underset{\to}{Y}_4^B$, therefore the set of MICS for the pressure tank fault tree consists only of $\underset{\to}{Y}_1^B$, $\underset{\to}{Y}_2^B$, and $\underset{\to}{Y}_3^B$.

The following BOOLEAN statements derive the set of MICS by eliminating the non-minimal cut-set vector included in the set of BICS.

```
                        /*        MICS      */
  3  1        MICS:     LADY=QUEEN;
  3  1          PUT SKIP LIST('BICS');
  3  1         DO WHILE(LADY¬=NULL);
  3  2         VT=LADY;
  3  2        PUT LIST('COMP=',VECTOR.COMP);
  3  2         LADY=LADY->FLOOR;
  3  2         END;
  3  1         LADY=QUEEN;
  3  1             ALLOCATE SOF;
  3  1             DO WHILE  (LADY¬=NULL);
  3  2             TOD=LADY->COMP;
  3  2             MOON=QUEEN;
  3  2             DO WHILE(MOON¬=NULL);
  3  3             IF (MOON=LADY) THEN GO TO MSZ;
  3  3             VT=MOON;
  3  3             IF(TOD=VECTOR.COMP) THEN GO TO MSA;
  3  3             SOF=(TOD&VECTOR.COMP);
  3  3             IF (SOF=TOD) THEN GO TO MSA;
  3  3             IF (SOF=VECTOR.COMP) THEN GO TO MSB;

  3  3             GO TO MSZ;
  3  3        MSA:        IF(MOON=QUEEN) THEN QUEEN=MOON->FLOOR;
  3  3          ELSE GO TO MSQ;
  3  3          FREE VECTOR;
  3  3          MOON=QUEEN;
  3  3           GO TO MSY;
  3  3      MSQ: MOON->FLOOR=MOON->FLOOR;
  3  3             FREE VECTOR;
  3  3             GO TO MSY;
  3  3       MSB:      VT=LADY;
  3  3             IF(LADY=QUEEN) THEN QUEEN=LADY->FLOOR;
  3  3         ELSE GO TO MSR;
  3  3         FREE VECTOR;
  3  3          MOAN=QUEEN;
  3  3          GO TO MSX;
  3  3       MSR:  MOAN->FLOOR=LADY->FLOOR;
  3  3             FREE VECTOR;
  3  3             GO TO MSX;
  3  3       MSZ:      MOON=MOON;
  3  3        MSY:     MOON=MOON->FLOOR;
  3  3             END;
  3  2             MOAN=LADY;
  3  2      MSX:         LADY=MOAN->FLOOR;
  3  2             END;
```

III.10 TRAVEL and TRAPEL

Gates having replicated event inputs in common are interconnected by means of WHIP and NAIL pointer variables. However, since PL-MOD arrives at the final modular decomposition through a series of different intermediate structural representations for the fault tree, at each step interdependent gate interconnections are attached to a different set of NODE,STIP, STID and MOD structures.

Procedures TRAVEL and TRAPEL are called by COALESCE and MODULAR to transfer NAIL and WHIP interconnections whenever a structural transformation is effected which involves interconnected structures.

Thus, given a set of structures $A_i (i = 1,2,...,n)$ attached by NAIL pointers to a structure B (i.e., A_i.NAIL j_i = pointer locating B for some j_i) which is to be replaced by a new structure C. Then TRAVEL will replace the old NAIL pointers connecting the set of structures A_i to B by a new set connecting them to C (i.e., A_i.NAIL j_i = pointer locating C for $i = 1,2,...,n$). Similarly TRAPEL will replace all WHIP connections to structure B by a new set of connections to structure C (i.e., if originally D_i· WHIP j_i = pointer locating B, then TRAPEL will change this to D_i.WHIP j_i = pointer locating C $i = 1,...,m$).

For example, in Section III.9 the NODE, STIP and STID structures representing the top gate for the pressure tank fault tree were given. In particular, structures $ST_5 \rightarrow$STIP and $SD_2 \rightarrow$STID were interconnected by

```
PRIM(1) = SPINE(4)
```

The values of TRIM (IX) and TRIN(IX) (IX = 1,2,...,RMOD) are read in and the values corresponding to PRIM(IX) are assigned in procedure INITIAL with the following statements

```
DO IX = 1 to RMOD;
GET LIST (TRIM)(IX),TRIN(IX));
ICH = TRIN (IX);
PRIM (IX) = SPINE(ICH);
END;
```

In Section III.6 it was pointed out that for every replicated input a structure AP is allocated by procedure TREE-IN. Structure AP is connected to the tree by a WHIP pointer corresponding to a structure containing the particular replicated event. AP has the following composition

```
1 AP BASED (APT),
2 TIPO = 0,
2 NAP = replicated event name,
2 REP = total number of appearances
        of the event in the fault tree,
2 SPIT POINTER,
```

(With A.WHIP$_j$ = APT for some structure A)

Pointer AP.SPIT is in general NULL except when the replicated event represents a module. In that case TREE-IN will use AP.SPIT to store the pointer locating the top gate for the modular sub-tree (i.e. AP.SPIT = PRIM(IX) for some IX).

$$ST_5 \rightarrow STIP\ .\ NAIL\ (1) = ST_5$$
$$ST_5 \rightarrow STIP\ .\ WHIP(1) = SD_2$$

$$SD_2 \rightarrow STID\ .\ NAIL(1) = ST_5$$
$$SD_2 \rightarrow STID\ .\ WHIP(1) = SD_2$$

$$SD_2 \rightarrow STID\ .\ NAIL(2) = SD_2$$
$$SD_2 \rightarrow STID\ .\ WHIP(2) = APT_1$$

However, in the next stage of the tree modularization procedure, gate B_1 was represented by the single structure MT_4 MOD. Hence TRAVEL and TRAPEL were needed to transfer all NAIL and WHIP interconnection to MT_4. Thus,

$$MT_4 = MOD.NAIL(1) = MOD.NAIL(2) = MOD.NAIL(3)$$

and

$$MT_4 = MOD.WHIP(1) = MOD.WHIP(2)$$

The statements corresponding to the TRAVEL AND TRAPEL procedures are given below.

```
1  0      TRAVEL: PROC(GRIS,KING,MOON):
2  0      DECLARE (GRIS, KING, MOON) POINTER;
2  0           GAL=GRIS->NODE.TIPO;
2  0           IF (GAL=0) THEN GO TO CIME;
2  0           ELSE IF (GAL=1) THEN GO TO CINE;
2  0           ELSE IF (GAL=2) THEN GO TO CIPE;
2  0            ELSE IF (GAL=3) THEN GO TO CIDE;
2  0       ELSE IF (GAL=4) THEN GO TO CIXE;
2  0      CINE:  NT=GRIS;
2  0           FAL=NODE.DIR;
2  0           DO MAL=1 TO FAL;
2  1           IF (NODE.NAIL(MAL)=MOON) THEN GO TO LANE;
2  1           END;
2  0      LANE:      NODE.NAIL(MAL)=KING;
```

```
2  0         · RETURN;
2  0       CIPE:   ST=GRIS;
2  0             FAL=STIP.DIR;
2  0             DO MAL=1 TO FAL;
2  1             IF (STIP.NAIL(MAL)=MOON) THEN GO TO LAPE;
2  1             END;
2  0       LAPE:      STIP.NAIL(MAL)=KING;
2  0             RETURN;
2  0         CIDE:    SD=GRIS;
2  0             FAL=STID.DIR;
2  0             DO MAL=1 TO FAL;
2  1             IF (STID.NAIL(MAL)=MOON) THEN GO TO LADE;
2  1             END;
2  0       LADE:     STID.NAIL(MAL)=KING;
2  0         RETURN;
2  0       CIXE:  MT=GRIS;
2  0        FAL=MOD.RIMO;
2  0       DO MAL=1 TO FAL;
2  1       IF (MOD.NAIL(MAL)=MOON) THEN GO TO LAXE;
2  1        END;
2  0       LAXE:     MOD.NAIL(MAL)=KING;
2  0        CIME:  RETURN;
2  0         END TRAVEL;
                /*   TRAPEL    */
1  0       TRAPEL: PROC (GRIS, KING, MOON);
2  0       DECLARE (GRIS, KING, MOON) POINTER;
2  0             GAL=GRIS->NODE.TIPO;
2  0             IF GAL=1  THEN GO TO CORN;
2  0             IF (GAL=2) THEN GO TO CORP;
2  0              IF (GAL=3) THEN GO TO CORD;
2  0       IF GAL=4  THEN GO TO CORX;
2  0       CORN: NT=GRIS;
2  0             FAL=NODE.DIR;
2  0             DO MAL=1 TO FAL;
2  1             IF (NODE.WHIP(MAL)=MOON) THEN GO TO LINE;
2  1             END;
2  0          LINE: NODE.WHIP(MAL)=KING;
2  0             RETURN;
2  0       CORP:   ST=GRIS;
2  0             FAL=STIP.DIR;
2  0             DO MAL=1 TO FAL;
2  1             IF (STIP.WHIP(MAL)=MOON) THEN GO TO LIPE;
2  1             END;
2  0       LIPE: STIP.WHIP(MAL)=KING;
2  0             RETURN;
2  0        CORD: SD=GRIS;
2  0              FAL=STID.DIR;
2  0            DO MAL=1 TO FAL;
2  1            IF (STID.WHIP(MAL)=MOON) THEN GO TO LYDE;
2  1            END;
2  0        LYDE: STID.WHIP(MAL)=KING;
2  0            RETURN;
2  0       CORX: MT=GRIS;
2  0       FAL=MOD.RIMO;
2  0       DO MAL=1 TO FAL;
2  1       IF(MOD.WHIP(MAL)=MOON) THEN GO TO LIXE;
2  1       END;
2  0       LIXE:   MOD.WHIP(MAL)=KING;
2  0        RETURN;
2  0        END TRAPEL;
```

III.11. Replicated Modules

An option exists in PL-MOD which provides for the analysis of fault trees containing smaller independent replicated sub-trees (i.e., replicated modules).

PL-MOD handles replicated modules by analyzing their sub-tree representation separately and by associating to each replicated module a replicated leaf input (Figure 3.33).

The total number of replicated modules RMOD in the tree is read in by procedure INITIAL which allocated the following four arrays

```
          GET LIST (RMOD);
          IF (RMOD = 0) THEN GO TO XEN;
          ALLOCATE TRIM (RMOD);
          ALLOCATE TRIN (RMOD);
          ALLOCATE PRIM (RMOD);
          ALLOCATE PRIN (RMOD)
     XEN:
```

Variables TRIM and TRIN are number arrays storing the replicated leaf and gate names associated with the top event of each replicated module. Thus, for the example given in Figure 3.33

```
RMOD = 1
TRIM(1) = 29001
TRIN(1) = 4
```

Variable PRIM is a pointer array which stores the locations of the node structures associated with the replicated module TOP gates. Thus, for the above example PRIM(1) = SPINE(4) = NT_4

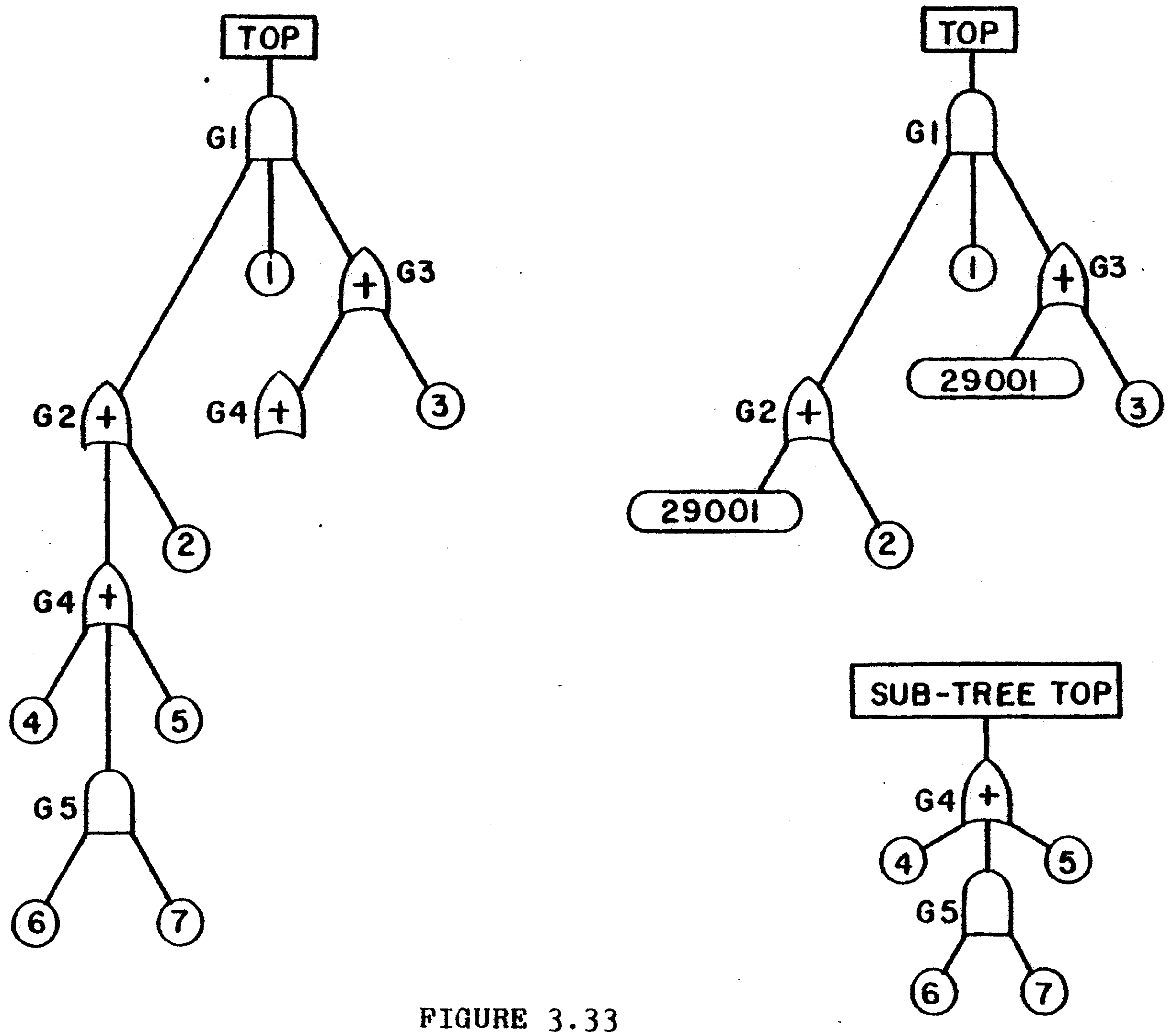


FIGURE 3.33

REPLICATED LEAF ASSOCIATED WITH A MODULE

Moreover the top modular gate NODE.ROOT will point to AP (PRIM(IX)→ NODE.ROOT = APT) and the set of pointers APT associated with replicated modules will be stored by array PRIN(IX).

III.12. Dual State Replicated Components

In Chapter I the NOT gate operator was shown to be a useful tool for handling common mode failure event dependencies and mutually exclusive events normally found in systems undergoing tests and maintenance [18]. PL-MOD contains an option that allows the handling of dual component states which arise by the application of the NOT gate operator (Figure 3.34). Applying the NOT operator to basic event b results in an event b = NOT(b). Since events b and b are mutually exclusive, the gates to which these dual states are attached become interdependent. Hence dual state components necessarily belong to the same higher order module (Figure 3.35).

As explained in Section III.6 dual states are identified by the nomenclature A1BCD, A2BCD (1 = ON state, 2 = OFF state). Notice that since the three lower digits are the same for both the ON and OFF states of a dual component, procedure TREE-IN will attach WHIP and NAIL interconnections among mutually exclusive gates as desired. Therefore, if a higher order modular structure contains an ON dual state, then it will also contain its corresponding OFF state.

In the following statements included in BOOLEAN, the cancellation of all modular minimal cut-sets which require the

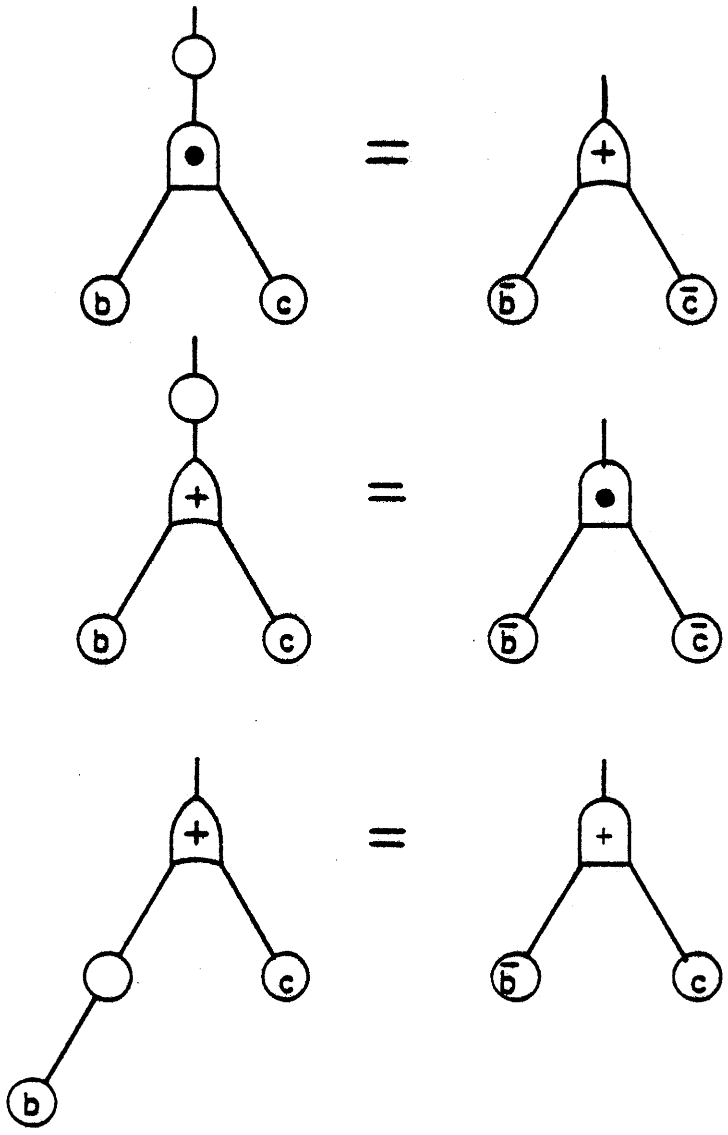


FIGURE 3.34

DUAL COMPONENT STATES

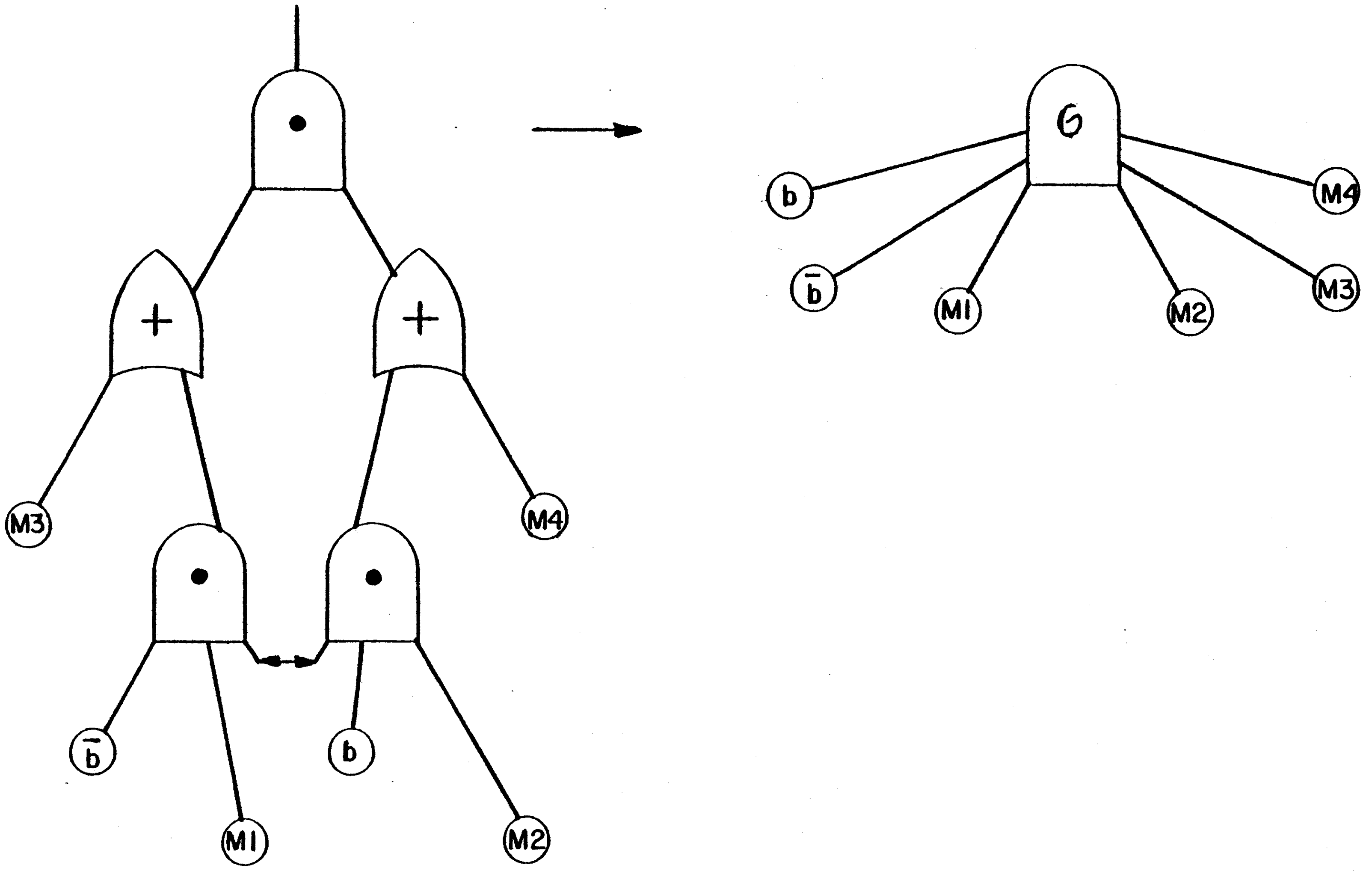


FIGURE 3.35

INTERDEPENDENT GATES DUE TO MUTUALLY EXCLUSIVE DUAL COMPONENT STATES

simultaneous occurrence of mutually exclusive events will be
ie
acheived.

```
                  /*    (A&¬A) STATE CANCELLATION   */
3  1                    IF (NOX=1) THEN DO;
3  2                    PR=LOST;
3  2                    NUM=PER.RAM;
3  2                    ALLOCATE  ZOTO;
3  2                    ALLOCATE  ZOCO;
3  2                    ZOTO=REPEAT('0'B,NUM);
3  2                    DO KIX=1 TO NUM;
3  3                    MA=PER.TAB(KIX);
3  3                    DA=-CEIL(-MA/10000);
3  3                     JA=-CEIL(-MA/1000);
3  3                    IF((JA-10*DA)=1) THEN DO;
3  4                    SUBSTR(ZOTO,KIX,2)='11'B;
3  4                    KIX=KIX+1;
3  4                    END;
3  3                    END;
3  2                    VIT=QUEEN;
3  2                     DO WHILE(VIT¬=NULL);
3  3                    VT=VIT;
3  3                    ZOCO=SUBSTR(VECTOR.COMP,1,NUM);
3  3                    ZOCO=ZOCO&ZOTO;
3  3                    IF(INDEX(ZOCO,'11'B)¬=0) THEN DO;
3  4                    IF VIT=QUEEN THEN QUEEN=VECTOR.FLOOR;
3  4                    ELSE GO TO SNU1;
3  4                    FREE VECTOR;
3  4                     VIT=QUEEN;
3  4                   GO TO SNU2;
3  4               SNU1: LAD->FLOOR=VIT->FLOOR;
3  4                    FREE VECTOR;
3  4                    END;
3  3                    ELSE LAD=VIT;
3  3                    VIT=LAD->FLOOR;
3  3             SNU2:    END;
3  2                    FREE ZOTO;
3  2                    FREE ZOCO;
3  2                    END;
3  1                    LOST->HECTOR=QUEEN;
```

III. 13. NUMERO

III.13.1. PL-MOD's Quantitative Analysis of Modularized Fault Trees

Up to now this Chapter has dealt with the methodology used by PL-MOD to obtain the modular decomposition for a fault tree. Once the modularization task has been accomplished, PL-MOD proceeds to evaluate modular event occurrence probabilities as well as Vesely-Fussell importance values for modular and basic component events. The set of procedures used by PL-MOD for this purpose are all contained within procedure NUMERO. Therefore PL-MOD commands a quantitative analysis for a fault tree by the statement

CALL NUMERO;

It should be stressed here that the modular structure information derived by PL-MOD is internally arranged in a manner which allows for an efficient numerical evaluation of the fault tree. Thus, storage space has been provided in structures PROP and PER for assigning reliability parameters to the simple and higher order modules represented by the structures

```
(Simple Module)     1  PROP BASED (PT)
                    2  TIPO FIXED,
                    2  ROOT POINTER,
                    2  REZ FIXED BINARY,
                    :
                    2  REL(DEL REFER (PROP. REZ)) FLOAT,
                    :
-------------------------------------------------------
(Higher Order
  Module)           1  PER BASED (PR)
                    2  REZ FIXED BINARY,
                    :
                    2  REL (DEL REFER (PR. REZ))  FLOAT,
-------------------------------------------------------
```

In the present PL-MOD version REZ = 2 since only a set of occurrence probabilities and Vesely-Fussell importance point values are evaluated. It should be noticed here that the pointer location for each module is stored both as an input to another module (PROP.TIM(I) or PER.TAR(J) and as the root to other modules (PROP.ROOT).

Procedure NUMERO internally calls the following procedures

```
CALL STAT-IN;
CALL EXPECT ;
CALL IMPORTANCE;
```

Procedure STAT-IN is used for reading in a list of input values for the basic event occurrence probabilities, such as those given in Table 3.1 for the pressure tank rupture fault tree. Having this information procedures EXPECT and IMPORTANCE then perform the evaluation of modular event occurrence probabilities and modular and basic component Vesely-Fussell importance measures respectively.

III.13.2 STAT-IN

Procedure STAT-IN is given by the following statements

```
1 0      STAT_IN:  PROC;
2 0           P=DEL;
2 0           GET LIST(FUN);
2 0           PUT EDIT('NUM FREE EVENT INPUTS=',FUN)  (SKIP(2),A(22),F(5));
2 0           GET LIST (DUN);
2 0      PUT EDIT('NUM REPLICATED EVENT INPUTS=',DUN) (SKIP(2),A(28),F(5));
2 0           ALLOCATE STATE;
2 0            ALLOCATE STATD;
2 0      PUT EDIT('FREE INPUT','RELIABILITY')
               (SKIP(2),X(2),A(10),X(1),A(11));
2 0           DO I=1 TO FUN;
2 1           GET LIST(I,STATE(1,I));
2 1           PUT EDIT(I,STATE(1,I))  (SKIP(2),F(12),E(18,6));
2 1           END;
2 0      PUT EDIT('DEP INPUT','RELIABILITY')
                (SKIP(2),X(3),A(9),X(1),A(11));
2 0           DO I=1 TO DUN;
2 1           GET LIST(I,STATD(1,I));
2 1           PUT EDIT(I,STATD(1,I))  (SKIP(2),F(12),E(18,6));
2 1           END;
2 0           END STAT_IN;
```

The number of free event (FUN) and replicated event (DON) inputs is read in. And arrays STATE (P.FUN) and STATD(P.DON) are allocated with P = 2. The free and replicated basic event probability values are read in and stored in STATE (1,I) and STATD (L,I). Later on the Vesely-Fussell importance corresponding to each free and replicated basic event will be stored in STATE (2,I) and STATD (2,J) respectively.

III.14 DOT, PLUS and MINUP

Proceudres DOT, PLUS and MINUP are internally called by EXPECT to evaluate the occurrence probability for a simple AND, simple OR and higher order prime module, given the set of occurrence probability values for all the inputs to the module. Moreover procedure MINUP is also called by IMPORTANCE to evaluate the Vesely-Fussell importance value for events which are inputs to a higher order module.

Given the occurrence probabilities for the set of inputs to a simple gate PROP structure (Figure 3.36), the probability of occurrence for the modular gate event will be given by

OR gate: $P(M) = \text{PLUS}(C_1, C_2, \ldots, C_n M_1, \ldots M_P)$

AND gate: $PCM = \text{DOT}(C_1, C_2, \ldots, C_n, M_1, \ldots, M_P)$

In its present form procedure PLUS uses the rare-event approximation to evaluate OR gate modular event probabilities. Thus

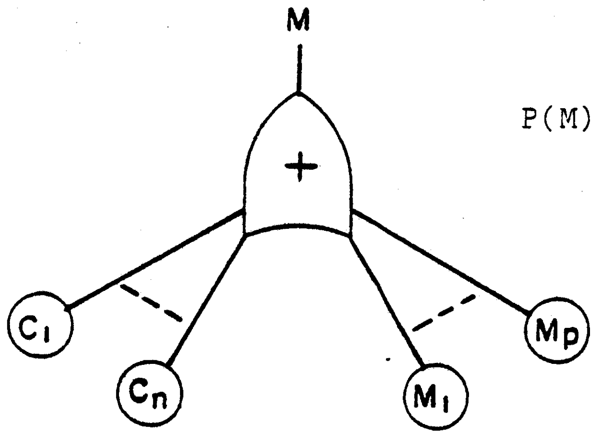
$$\text{PLUS}\ (C_1, C_2, \ldots, C_n, M_1, \ldots, M_P) = \sum_{i=1}^{n} P_i + \sum_{i=1}^{P} P_{M_i}$$

while

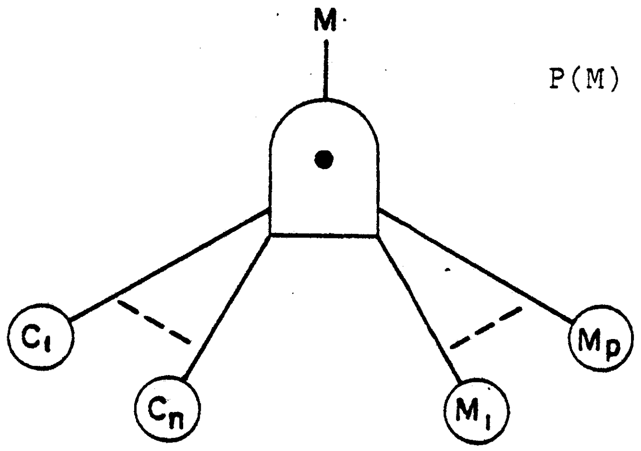
$$\text{DOT}\ (C_1, C_2, \ldots, C_n, M_1, M_2, \ldots M_P = \left(\prod_{i=1}^{n} P_i\right)\left(\prod_{i=1}^{P} P_{M_i}\right)$$

Procedures PLUS and DOT are given by the following statements.

```
1  0     PLUS: PROC(BAT,EXA);
2  0     DECLARE BAT POINTER;
2  0       DECLARE EXA LABEL;
2  0           PT=BAT;
2  0           REX=0;
2  0           IF (PROP.LIM=1 & PROP.TIL(1)=0) THEN GO TO PLUA;
2  0           DO J=1 TO PROP.LIM;
2  1           REX=REX+STATE(1,PROP.TIL(J));
2  1           END;
2  0     PLUA:  IF (PROP.MIM=1 & PROP.PIM(1)=NULL) THEN GO TO PLUB;
2  0            DO J=1 TO PROP.MIM;
2  1           IF (PROP.PIM(J)->PROP.HOST¬=NULL) THEN DO;
2  2            PP=PROP.PIM(J)->PROP.HOST;
2  2            REX=PEX+PER.REL(1);
2  2           END;
2  1          ELSE RFX=RFX+PROP.PIM(J)->PROP.REL(1);
2  1           END;
2  0     PLUB:  PROP.REL(1)=RFX;
2  0           GO TO EXA;
2  0      END PLUS;
1  0     DOT: PROC(BAT,EXA);
2  0           DECLARE  BAT POINTER;
2  0       DECLARE EXA LABEL;
2  0            PT=BAT;
```



$P(M) = PLUS(c_1,\ldots,c_n,M_1,\ldots,M_p)$



$P(M) = DOT(c_1,\ldots,c_n,M_1,\ldots,M_p)$

FIGURE 3.36

SIMPLE GATE MODULAR OCCURRENCE PROBABILITIES

```
    2  0        REX=1;
    2  0        IF (PROP.LIM=1 & PROP.TIL(1)=0) THEN GO TO DOTA;
    2  0        DO J=1 TO PROP.LIM;
    2  1        REX=REX*STATE(1,PROP.TIL(J));
    2  1        END;
    2  0  DOTA:  IF (PROP.MIM=1  & PROP.PIM(1)=NULL) THEN GO TO DOTB;
    2  0        DO J=1 TO PROP.MIM;
    2  1        IF (PROP.PIM(J)->PROP.HOST¬=NULL) THEN DO;
    2  2        PR=PROP.PIM(J)->PROP.HOST;
    2  2        REX=REX*PER.REL(1);
    2  2         END;
    2  1        ELSE REX=REX*PROP.PIM(J)->PROP.REL(1);
    2  1        END;
    2  0  DOTB:    PROP.REL(1)=REX;

    -  -  -

    2  0        GO TO EXA;
    2  0        END DOT;
```

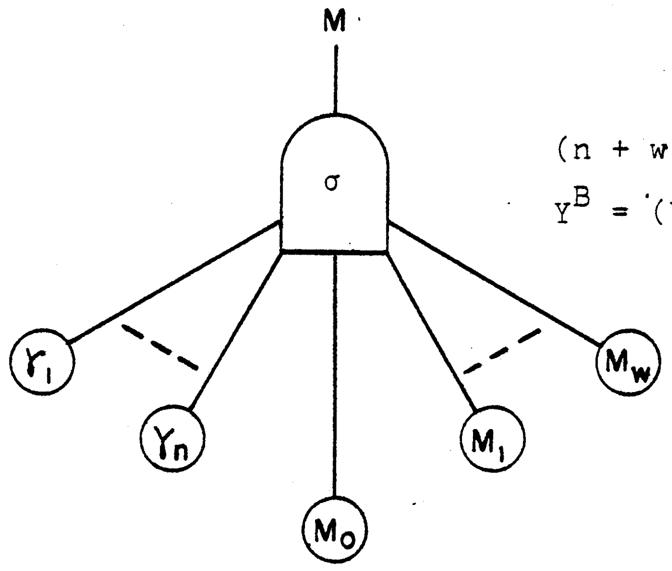
Since higher order modular structures (Figure 3.37) are characterized by a set of modular minimal cut-sets, their occurrence probability may be evaluated using the minimal cut upper bound in its rare-event approximation form (Equation 2.15) i.e.,

$$P(M_o) \leq \sum_{j=1}^{N_k} \prod_{i \varepsilon K_j} P_i = \text{MINUP}(r_1,\ldots,r_n,M_o,M_1\ldots,M_w)$$

with N_k = total number of cut-sets associated with the prime gate. Given the occurrence probabilities for each input to the prime gate, procedure EXPECT will store these values in a structure QER defined by

```
1 QER BASED (AT),
2 QEL FIXED BINARY,
2 QU (LARG REFER (QER.QEL)) FLOAT;
```

with PER.DEXTER = AT for the PER structure associated with a particular prime moduel, Procedure MINUP will then use the QER.QU(1) (I = 1,2,...,LARG) values coupled with the set of MICS VECTORS for the prime module to evaluate its occurrence probability as follows:



$(n + w + 1 = L)$

$Y^B = (Y_{r_1}, \ldots Y_{r_n}, Y_{M_o}, Y_{M_1}, \ldots, Y_{M_w})$

$K_1 = \ldots, 1, \ldots, 0, \ldots, 1, \ldots)$

$\vdots$

$K_t = (\ldots .1, \ldots 0, \ldots 1)$

$(t = N_k)$

$$P(M) = \text{MINUP}(r_1, r_2, \ldots, M_o, M_1, \ldots, M_w)$$

FIGURE 3.37

PRIME GATE MODULAR OCCURRENCE PROBABILITY

```
               /*     RELIABILITY CALCULATION    */
                 /*       MINUP      */
1  0      MINUP: PROC(EX);
2  0      DECLARE  EX  FIXED;
2  0        PR=TIERRA;
2  0           VIT=PER.HECTOR;
2  0           LARG=VIT->VECTOR.LORO;
2  0            REY=0;
2  0            DO WHILE (VIT¬=NULL);
2  1            REX=1;
2  1             VT=VIT;
2  1            DO EL=1 TO LARG;
2  2           POW=SUBSTR(VECTOR.COMP,EL,1);
2  2           IF EL=EX THEN DO;
2  3           IF POW='0'B THEN REX=0;
2  3           ELSE GO TO NUB;
2  3            GO TO NUP;
2  3            END;
2  2      NUB:   IF (POW='1'B) THEN NOW=1;
2  2           ELSE  NOW=0;
2  2           REM=QER.QU(EL);
2  2           IF(REM=0 & NOW=0) THEN REM=1;
2  2           REX=REX*(REM**NOW);
2  2           END;
2  1      NUP:      REY=REY+REX;
2  1            VIT=VIT->FLOOR;
2  1            END;
2  0            END MINUP;
```

As shown in Sections III.15 and III.16, each time procedure MINUP is called by EXPECT, variable EX equals zero. However whenever MINUP is called by the IMPORTANCE procedure, to evaluate nested gate and replicated event Vesely-Fussell importances, the value of EX is always different from zero.

Procedure MINUP essentially consists of a DO loop in which pointer VT successively locate a different MICS VECTOR for the prime gate module. The contribution of each vector to the minimal cut upper bound is found by multiplying the occurrence probabilities (QER.QU(EL)) corresponding to non-zero bits in the vector (i.e. POW=SUBSTR(VECTOR.COMP,EL,1) ≠ '0'B). Finally all the vector contributions are added together (REY=REY + REX) to obtain the rare-event approximation to the minimal cut-set upper bound. Notice however that that when EX is different from zero, only those contributions coming from a vector which has a '1' bit in the EX-th location are added together (IF POW = '0'B THEN REX = 0;).

III.15. EXPECT

Modular occurrence probabilities are easily computed by procedure EXPECT following the same order in which the modules were originally created by procedure MODULAR. Each time a PROP structure was crested in MODULA, its pointer location was stored in array BOST(IB) and variable IB was increased by one. Hence the set of modular occurrence probabilities are computed in the desired order by means of the DO LOOP

```
DO  I = 1 to IB;
CAT = BOST(I);
PT = CAT;
:
:
:
:
:
ESTA
END;
```

For the case of simple AND and OR gate modules, their occurrence probabilities are easily evaluated using the statements

```
CALL DOT(CAT,ESTA);
```

and

```
CALL PLUS(CAT,ESTA);
```

where the values for the modular input occurrence probabilities are guaranteed to have been previously evaluated by EXPECT because of its recursive computational ordering (DO I = 1 to IB;).

Particular care must however be taken for the case of higher order modular structures (Figure 3.37). For this case BOOLEAN first allocated the PROP structure associated with the parent gate (M_0) and later on allocated the set of PROP structures associated with each of the nested gates ($M_1, M_2, \ldots, M_n$) included in the higher order module.

As explained in Section III.14, EXPECT calls procedure MINUP(EX) (EX = 0) to compute the higher order gate occurrence probability (PER.REL(1)). However to make this evaluation possible, it is necessary that EXPECT previously (a) compute the set of occurrence probabilities corresponding to each nested simple gate PROP structure (WEST = total number of nested gates) by calling procedures DOT and PLUS, and that (b) QER.QU(J) (J = 1,2,...,LARG; LARG = NUM + WEST + 1) be assigned the set of values associated with each replicated event and nested gate module contained in the prime gate module.

This set of tasks are performed by EXPECT through the following statements:

```
111    1   0       EXPECT:  PROC;
112    2   0           DO I=1 TO IB;
113    2   1           CAT=BOST(I);
114    2   1           PT=CAT;
115    2   1            IF (PROP.HOST¬=NULL) THEN GO TO CUTS;
116    2   1           IF PROP.VALUE=1 THEN CALL DOT(CAT,ESTA);
117    2   1            IF PROP.VALUE=2 THEN CALL PLUS(CAT,ESTA);
118    2   1       CUTS:   IF (PROP.VALUE<=2) THEN EYE=1;
119    2   1           ELSE EYE=0;
120    2   1           PR=PROP.HOST;
121    2   1            TIERRA=PR;
122    2   1           NUB=PER.RAM;
123    2   1            IF (NUB=1 & PER.TAR(1)=0) THEN NUM=0;
124    2   1           ELSE NUM=NUB;
125    2   1           WEST=PER.LEAL;
126    2   1           IF (WEST=1 & PER.JIM(1)=0) THEN NEZT=0;
127    2   1            ELSE NEZT=WEST;
128    2   1           IF EYE=0 THEN LARG=NUM+NEZT;
129    2   1           ELSE LARG=NUM+NEZT+1;
130    2   1           ALLOCATE QER;
131    2   1           PER.DEXTER=QT;
132    2   1           IF EYE=0 THEN GO TO CUTA;
                       /*    ASYMMERTIC CASE */
133    2   1            DO J=1 TO NUM;
134    2   2           MA=PER.TAR(J);
135    2   2           DA=-CEIL(-MA/10000);
136    2   2           JA=-CEIL(-MA/1000);
137    2   2           JAK=JA-10*DA;
138    2   2           NA=MA-(1000)*JA;
139    2   2            IF(JAK=0 |JAK=1)  THEN DO;
140    2   3           QER.OU(J)=STATD(1,NA);
141    2   3           END;
142    2   2           IF (JAK=2)  THEN DO;
143    2   3           QER.OU(J)=1-STATD(1,NA);
144    2   3            END;
145    2   2           IF (JAK=9)  THEN DO;
146    2   3           DO IX=1 TO RMOD;
147    2   4           IF(TRIM(IX)=MA) THEN GO TO XUTA;
148    2   4           END;
149    2   3       XUTA:   APT=PRIN(IX);
150    2   3          IF (AP.SPIT->PROP.HOST¬=NULL) THEN DO;
151    2   4           PR=AP.SPIT->PROP.HOST;
152    2   4           STATD(1,NA)=PER.REL(1);
153    2   4           PR=TIERRA;
154    2   4           END;
155    2   3           ELSE STATD(1,NA)=AP.SPIT->PROP.REL(1);
156    2   3           QER.OU(J)=STATD(1,NA);
```

```
PL/I OPTIMIZING COMPILER            NUMERO: PROCEDURE;

    STMT LEV NT

     157   2  3          PUT EDIT('REP MODULE=',NA,'REL=',STATD(1,NA))
                            (SKIP(1),A(11),F(5),X(2),A(4),E(18,6));
     158   2  3          END;
     159   2  2          END;
     160   2  1          IF PROP.VALUE=1 THEN CALL  DOT(CAT,ELSA);
     161   2  1          IF PROP.VALUE=2  THEN CALL PLUS(CAT,ELSA);
     162   2  1     ELSA: PUT SKIP LIST('PATRIARCH SUBMODULE');
     163   2  1        PUT EDIT('MODULE NAME=',PROP.NAME,'REL=',PROP.REL(1))
                             (SKIP(1),A(12),F(5),X(2),A(4),E(18,6));
     164   2  1          QER.QU(NUM+1)=PROP.REL(1);
     165   2  1          BAT=PT;
     166   2  1          DO IN=I+1 TO I+NEZT;
     167   2  2           LAD=BOST(IN);
     168   2  2          PT=LAD;
     169   2  2          IF PROP.VALUE=1 THEN CALL DOT(LAD,ELNA);
     170   2  2          IF PROP.VALUE=2 THEN CALL PLUS(LAD,ELNA);
     171   2  2     ELNA: PUT SKIP LIST('NESTED MODULE') ;
     172   2  2        PUT EDIT('MODULE NAME=',PROP.NAME,'REL=',PROP.REL(1))
                             (SKIP(1),A(12),F(5),X(2),A(4),E(18,6));
     173   2  2          QER.QU(NUM+1+IN-I)=PROP.REL(1);
     174   2  2          END;
     175   2  1          EX=0;
     176   2  1          CALL MINUP(EX);
     177   2  1          PER.REL(1)=REY;
     178   2  1         PUT SKIP LIST('PATRIARCH MODULE');
     179   2  1           I=I+NEZT;
     180   2  1          PT=BAT;
     181   2  1          PUT EDIT('MODULE NAME=',PROP.NAME,'REL=',PER.REL(1))
                         (SKIP(1),A(12),F(5),X(2),A(4),E(18,6));
     182   2  1          GO TO EZTA;
                        /*      SYMMETRIC     CASE    */
     183   2  1     CUTA:       PROP.REL(1)=0;
     184   2  1          BAT=PT;
     185   2  1          IF NUM=0 THEN GO TO CUTB;
     186   2  1          DO J=1  TO NUM;
     187   2  2          QER.QU(J)=STATE(1,PER.TAR(J));
     188   2  2          END;
     189   2  1     CUTB: IF NEZT=0 THEN GO TO CUTC;
     190   2  1          DO IX=NUM+1 TO NUM+NEZT;
     191   2  2           PT=PER.KIM(IX-NUM);
     192   2  2           IF (PROP.HOST=NULL)   THEN QER.QU(IX)=PROP.REL(1);
     193   2  2          ELSE   QER.QU(IX)=PROP.HOST->PER.REL(1);
     194   2  2          END;
     195   2  1      CUTC:  EX=0;
     196   2  1          CALL MINUP(EX);
     197   2  1          PER.REL(1)=REY;
     198   2  1          PT=BAT;
     199   2  1          PROP.REL(1)=REY;
     200   2  1          PUT SKIP LIST('SYMM SUPERMODULE');
```

```
PL/I OPTIMIZING COMPILER              NUMERO: PROCEDURE;

   STMT LEV NT

    201   2  1          PUT EDIT('MODULE NAME=',PROP.NAME,'REL=',PER.REL(1))
                        (SKIP(1),A(12),F(5),X(2),A(4),E(18,6));
    202   2  1          GO TO EZTA;
    203   2  1      ESTA: PUT SKIP LIST('FREE MODULE');
    204   2  1         PUT EDIT('MODULE NAME=',PROP.NAME,'REL=',PROP.REL(1))
                            (SKIP(1),A(12),F(5),X(2),A(4),E(18,6));
    205   2  1      EZTA:   END;
    206   2  0         END   EXPECT;
```

For the pressure tank fault tree example procedure EXPECT computes the modular and top event occurrence probabilities in the following steps

STEP 1 Symmetric higher order module M_9

$$\underset{\sim}{Y}^B = (Y_{11}, Y_{12}, Y_{13})$$

$$K_1 = (0,1,1)$$

$$K_2 = (1,0,1)$$

$$K_3 = (1,1,0)$$

$$P_1 = P_2 = P_3 = 10^{-5}$$

$$\Rightarrow P_{M_9} = 3 \times 10^{-10}$$

STEP 2 (a) Parent gate sub-module M_1

$$M_1 = \{1,2,3,4;\ U\}$$

$$P_1 = 10^{-8}\ ,\ P_2 = P_3 = P_4 = 10^{-5}$$

$$\Rightarrow P_{M_1} = 3.001 \times 10^{-5}$$

(b) Nested gate module M_4

$M_4 = \{M_g\}$

$\Rightarrow P_{M_4} = 3 \times 10^{-10}$

(c) Nested gate module M_5

$M_5 = \{5,6,7,8,9,10;\ U\}$

$P_5 = P_6 = P_7 = P_8 = P_9 = P_{10} = 10^{-5}$

$\Rightarrow P_{M_5} = 6 \times 10^{-5}$

STEP 3 Top tree event higher order module M

$\underset{\sim}{Y}^B = (Y_r, Y_{M_1}, Y_{M_4}, Y_{M_5})$

$K_1 = (0, 1, 0, 0)$

$K_2 = (1, 0, 0, 0)$

$K_3 = (0, 0, 1, 1)$

$(r = 30001)\ P_r = 10^{-5}$

$\Rightarrow P(TOP) = 4.001 \times 10^{-5}$

III.16 IMPORTANCE

Procedure IMPORTANCE evaluates the Vesely-Fussell importance ($I^{V.F.}$) for every modular event and every basic component in the fault tree. IMPORTANCE performs this evaluation by starting at the top tree gate event ($I^{V.F.}_{TOP} = 1$) and proceeding down to the bottom branch modules of the tree by means of the modular importance chain-rule (See Section II.5.4.)

For the case of simple AND and OR gate modules, the modular importance chain rule takes the forms

AND gate: $I^{V.F.}_{C_i} = I^{V.F.}_{M} \; (i = 1,2,\ldots,n)$

$$I^{V.F.}_{M_i} = I^{V.F.}_{M} (i = 1,2,\ldots,n)$$

OR gate: $I^{V.F.}_{C_i} = I^{V.F.}_{M} \left(\frac{P_i}{P_M}\right) \; (i = 1,2,\ldots,n)$

$$I^{V.F.}_{M_i} = I^{V.F.}_{M} \left(\frac{P_{M_i}}{P_M}\right) (i = 1,3,\ldots,P)$$

For an AND gate module, all its inputs have the same importance as the module since the probability that any input has failed given that the AND gate module has failed equals one. However for an OR gate, the probability that a given input is in a failed state given that the OR gate has failed is equal to

$$\frac{P(\text{input has failed})}{P_M}$$

Notice that the required modular occurrence probabilities (P_M and P_{M_i}) were previously computed by EXPECT. For the case of higher order modular gates (Figure 3.37) the modular importance chain rule in the rare-event approximation takes the form

$$I_{r_i}^{V.F.} = I_M^{V.F.} \left(\frac{\sum_{j, r_i \varepsilon K_j} P(K_j)}{P(M)} \right) (i = 1,\ldots,n)$$

$$I_{M_i}^{V.F.} = I_M^{V.F.} \left(\frac{\sum_{j, M_i \varepsilon K_j} P(K_j)}{P(M)} \right) \quad (i = 0,1,\ldots,u)$$

$$\text{with } P(M) = \sum_{j=1}^{t} P(K_j)$$

It should be recalled that the occurrence probability for a higher order module P(M) was computed in EXPECT by calling procedure MINUP(EX) with EX = 0. Nevertheless the expression appearing in the numerator

$$\sum_{j, x \varepsilon K_j} P(K_j) \quad (x = r_i \text{ or } M_i)$$

is yet to be evaluated by IMPORTANCE. To this end procedure MINUP(EX) will be called with variable EX locating the position in the VECTOR.COMP bit-string which corresponds to input x (See Section III.14).

Procedure IMPORTANCE starts out by assigning importance values to all modular and component inputs to the top gate event (First generation), and at the same time stores in array OLM(BUM) all the pointer locations for the modular gate inputs to the top gate module. This task is performed for simple and prime gate top event modules by the following statements

```
                   /*      IMPORTANCE   (VESELY-  FUSSELL)     */
 1  0          IMPORTANCE:   PROC;
 2  0            BUG=1;
 2  0            PT=STORK;
 2  0            IF PROP.HOST¬=NULL THEN GO TO IMA;
 2  0             BUM=PROP.MIM;
 2  0            ALLOCATE OLM(BUM);
 2  0            OLM=PROP.PIM;
 2  0            PROP.REL(2)=1;
 2  0          PUT EDIT('MODULE=',PROP.NAME,'IMP=',PROP.REL(2))
                 (SKIP(1),A(7),F(5),A(4),E(18,6));
 2  0            IF PROP.VALUE=1 THEN DO;
 2  1            IF (PROP.LIM=1 & PROP.TIL(1)=0) THEN GO TO IME;
 2  1            DO I=1 TO PROP.LIM;
 2  2            STATE(2,PROP.TIL(I))=1;
 2  2            END;
 2  1            END;
 2  0            IF PROP.VALUF=2  THEN DO;
 2  1            IF (PROP.LIM=1 & PROP.TTL(1)=0) THEN GO TO IME;
 2  1             DO I=1 TO PROP.LIM;
 2  2            STATE(2,PROP.TIL(I))=STATE(1,PROP.TIL(I))/PROP.REL(1);
 2  2            END;
 2  1            END;
 2  0        GO TO IME;
               /*    CUT SET  CASE    */
 2  0         IMA:    PR=PROP.HOST;
 2  0            IF (PROP.MIM=1 & PROP.PIM(1)=NULL) THEN DO;
 2  1            BUM=0;
 2  1            BUN=0;
 2  1            END;
 2  0            ELSE  DO;
 2  1            BUM=PROP.MIM;
 2  1            BUN=1;
 2  1            END;
 2  0            BUM=BUM+PER.LEAL;
 2  0             BUZ=BUM;
 2  0             DO IK=1  TO PER.RAM;
 2  1             MA=PER.TAR(IK);
 2  1            DA=-CEIL(-MA/10000);
 2  1            JA=-CEIL(-MA/1000);
 2  1             JAK=JA-10*DA;
```

```
PL/I OPTIMIZING COMPILER             NUMERO: PROCEDURE;

    STMT LEV NT

     245    2  1             IF  JAK=9  THEN  DO;
     246    2  2            BUM=BUM+1;
     247    2  2            END;
     248    2  1            END;
     249    2  0            BUZ=BUM-BUZ;
     250    2  0            ALLOCATE OLM(BUM);
     251    2  0            IF BUN=0 THEN DO;
     252    2  1            I=0;
     253    2  1       GO TO IMA0;
     254    2  1            END;
     255    2  0            DO I=1 TO PROP.MIM;
     256    2  1            OLM(I)=PROP.PIM(I);
     257    2  1            END;
     258    2  0      IMA0:   DO IL=I+1  TO BUM-BUZ;
     259    2  1            OLM(IL)=PER.KIM(IL-I);
     260    2  1            END;
     261    2  0            IF (BUZ¬=0)  THEN DO;
     262    2  1            DO IK=1  TO PER.RAM;
     263    2  2            MA=PER.TAR(IK);
     264    2  2             DA=-CEIL(-MA/10000);
     265    2  2             JA=-CEIL(-MA/1000);
     266    2  2            JAK=JA-10*DA;
     267    2  2             IF (JAK=9)    THEN DO;
     268    2  3             DO  IX=1  TO RMOD;
     269    2  4             IF (TRIM(IX)=MA) THEN GO TO IMA4;
     270    2  4             END;
     271    2  3      IMA4:   OLM(IL)=PRIN(IX)->AP.SPIT;
     272    2  3       PUT EDIT('INDEX=',IL,'PROP=',OLM(IL)->PROP.NAME)
                                    (SKIP(1),A(6),F(5),A(5),F(5));
     273    2  3            IL=IL+1;
     274    2  3            END;
     275    2  2            END;
     276    2  1            END;
     277    2  0            PER.REL(2)=1;
     278    2  0          PUT EDIT('PATR=',PROP.NAME,'IMP=',PER.REL(2))
                            (SKIP(1),A(5),F(5),A(4),E(18,6));
     279    2  0            IF PROP.VALUE>2 THEN GO TO IMA2;
     280    2  0            IF PROP.VALUE=1 THEN DO;
     281    2  1            IF (PROP.LIM=1 & PROP.TIL(1)=0) THEN DO ;
     282    2  2            PROP.REL(2)=0;
     283    2  2            GO TO IMA1;
     284    2  2            END;
     285    2  1            PROP.REL(2)=1;
     286    2  1            DO I=1 TO PROP.LIM;
     287    2  2            STATE(2,PROP.TIL(I))=1;
     288    2  2            END;
     289    2  1             END;
     290    2  0             IF PROP.VALUE=2 THEN DO;
     291    2  1            IF (PROP.LIM=1 & PROP.TIL(1)=0) THEN DO;
```

```
PL/1 OPTIMIZING COMPILER              NUMERO: PROCEDURE;

    STMT LEV NT

     292   2  2            PROP.REL(2)=0;
     293   2  2            GO TO IMA1;
     294   2  2            END;
     295   2  1             PROP.REL(2)=PROP.REL(1)/PER.REL(1);
     296   2  1            DO I=1 TO PROP.LIM;
     297   2  2            STATE(2,PROP.TIL(I))=STATE(1,PROP.TIL(I))/PER.REL(1);
     298   2  2            END;
     299   2  1            END;
     300   2  0       IMA1:   DO I=1  TO PER.RAM;
     301   2  1            EX=I;
     302   2  1            TIERRA=PR;
     303   2  1            QT=PER.DEXTER;
     304   2  1            CALL MINUP(EX);
     305   2  1            PUT EDIT('I=',I,'PER.TAR=',PER.TAR(I),'REY=',REY)
                           (SKIP(2),A(2),F(5),A(8),F(5),A(4),E(18,6));
     306   2  1            MA=PER.TAR(I);
     307   2  1            DA=-CEIL(-MA/10000);
     308   2  1            JA=-CEIL(-MA/1000);
     309   2  1             JAK=JA-10*DA;
     310   2  1            NA=MA-(1000)*JA;
     311   2  1            IF (JAK¬=2)  THEN STATD(2,NA)=REY/PER.REL(1);
     312   2  1            IF (JAK=2) THEN DO;
     313   2  2            SNOT=REY/PER.REL(1);
     314   2  2            PUT EDIT('NOTSTATE=',MA,'IMP=',SNOT)
                              (SKIP(2),A(9),F(5),X(2),A(4),E(18,6));
     315   2  2            END;
     316   2  1            END;
     317   2  0            GO TO IME;
                            /*   SYMMETRIC   CASE    */
     318   2  0        IMA2:  PROP.REL(2)=0;
     319   2  0            IF (PER.RAM=1 & PER.TAR(1)=0) THEN GO TO IME;
     320   2  0            ELSE DO I=1  TO PER.RAM;
     321   2  1            EX=I;
     322   2  1            TIERRA=PR;
     323   2  1            QT=PER.DEXTER;
     324   2  1            CALL MINUP(EX);
     325   2  1            PUT EDIT('I=',I,'PER.TAR=',PER.TAR(I),'REY=',REY)
                           (SKIP(2),A(2),F(5),A(8),F(5),A(4),E(18,6));
     326   2  1             STATE(2,PER.TAR(I))=REY;
     327   2  1            END;
     328   2  0             GO TO IME;
```

At this point IMPORTANCE is ready to assign importance values to the second generation of fault tree inputs, and at the same time storing the pointers locating the second generation modules. This process will then be continued on until a generation (last generation) is found which contains no modular inputs (i.e., no-gates). IMPORTANCE performs this task by means of a DO LOOP which stops when the last generation is found (⇒ BUG = 0).

Each generation of modules GOLD(BUG) is created by passing on the old values of array OLM(BUM) found in the previous sweep. Moreover, a new generation of module pointers is created and assigned to OLM(BUM) with the following statements

```
                          /*    LOOP STARTS HERE    */
 2  0         IME:   DO WHILE(BUG¬=0);
 2  1                   BUG=BUM;
 2  1                   PUT LIST('BUG=',BUG);
 2  1                   IF (BUG=0) THEN GO TO IME;
 2  1                   ALLOCATE GOLD(BUG);
 2  1                   DO I=1 TO BUG;
 2  2                   GOLD(I)=OLM(I);
 2  2               PUT EDIT('GOLD=',I,'PROP=',GOLD(I)->PROP.NAME)
                        (SKIP(1),A(5),F(5),A(5),F(5));
 2  2                   END;
 2  1                   FREE OLM;
 2  1                   BUM=0;
 2  1                   DO I=1  TO BUG;
 2  2                   PT=GOLD(I);
 2  2                   IF PROP.HOST=NULL THEN DO;
 2  3                   IF (PROP.MIM=1 & PROP.PIM(1)=NULL)  THEN GO TO IME3;
 2  3                   ELSE BUM=BUM+PROP.MIM;
 2  3                   GO  TO  IME3;
 2  3                   END;
 2  2                   ELSE PR=PROP.HOST;
 2  2                   IF (PROP.MIM=1 & PROP.PIM(1)=NULL) THEN GO TO IME2;
 2  2                   ELSE BUM=BUM+PROP.MIM;
 2  2         IME2: IF (PER.LEAL=1 & PER.KIM(1)=NULL) THEN GO TO IME1;
 2  2                    ELSE BUM=BUM+PER.LEAL;
 2  2          IME1:    DO IX=1 TO PER.RAM;
 2  3                    MA=PER.TAR(IX);
 2  3                   DA=-CEIL(-MA/10000);
 2  3                   JA=-CEIL(-MA/1000);
 2  3                   JAK=JA-10*DA;
 2  3                   IF JAK=9  THEN DO;
```

```
2  4              BUM=BUM+1;
2  4              END;
2  3              END;
2  2         IME3:          END;
2  1              IF BUM=0 THEN GO TO IMI3;
2  1               ALLOCATE OLM(BUM);
2  1               IL=0;
2  1              DO I=1    TO BUG;
2  2              PT=GOLD(I);
2  2              IF PROP.HOST=NULL THEN DO;
2  3               IF (PROP.MIM=1  &  PROP.PIM(1)=NULL)  THEN GO TO IMI4;
2  3              DO IT=1 TO PROP.MIM;
2  4              IL=IL+1;
2  4               OLM(IL)=PROP.PIM(IT);
2  4              END;
2  3              GO TO IMI4;
2  3              END;
2  2              ELSE PR=PROP.HOST;
2  2             PUT EDIT('HOST','PROP=',PROP.NAME)
                  (SKIP(1),A(4),A(5),F(5));
2  2               IF (PROP.MIM=1 & PROP.PIM(1)=NULL) THEN GO TO IMI2;
2  2              DO IT=1 TO PROP.MIM;
2  3              IL=IL+1;
2  3              OLM(IL)=PROP.PIM(IT);
2  3              END;
2  2         IMI2:  IF (PER.LEAL=1 & PER.KIM(1)=NULL) THEN GO TO IMI1;
2  2              DO IT=1 TO PER.LEAL;
2  3              IL=IL+1;
2  3              OLM(IL)=PER.KIM(IT);
2  3               END;
2  2          IMI1:   DO IK=1  TO PER.RAM;
2  3              MA=PER.TAR(IK);
2  3              DA=-CEIL(-MA/10000);
2  3              JA=-CEIL(-MA/1000);
2  3              JAK=JA-10*DA;
2  3              IF JAK=9  THEN DO;
2  4              DO IX=1  TO RMOD;
2  5              IF (TRIM(IX)=MA)  THEN GO TO IMK1;
2  5              END;
2  4         IMK1:  OLM(IL+1)=PRIM(IX)->AP.SPIT;
2  4              IL=IL+1;
2  4              END;
2  3              END;
2  2         IMI4:        END;
```

In addition, the set of basic component and modular gate inputs to the older generation of modules pointed at by GOLD(I) are assigned importance values with the following statements

```
                /*  ASSIGN IMPORTANCES OF OLDER GENERATION  */
2  1         IMI3:  DO I=1 TO BUG;
2  2              PT=GOLD(I);
2  2              CAT=PROP.ROOT;
2  2              IF (CAT->PROP.TIPO=0)  THEN DO;
2  3                  APT=CAT;
2  3              MA=AP.NAP;
2  3              JA=-CEIL(-MA/1000);
2  3              NA=MA-(1000)*JA;
2  3              IF(PROP.HOST¬=NULL) THEN DO ;
2  4              PR=PROP.HOST;
2  4                TIERRA=PR;
2  4              QT=PER.DEXTER;
2  4              PER.REL(2)=STATD(2,NA);
2  4              GO TO IMK3;
2  4              END;
2  3              ELSE PROP.REL(2)=STATD(2,NA);
```

```
2  3            GO TO IMK2;
2  3            END;
2  2          IF CAT->PROP.HOST¬=NULL THEN GO TO EMA2;
2  2          IF PROP.HOST¬=NULL THEN GO TO EMA1;
2  2   IMK4:    IF CAT->PROP.VALUE=1 THEN PROP.REL(2)=CAT->PROP.REL(2);
2  2        ELSE PROP.REL(2)=PROP.REL(1)*CAT->PROP.REL(2)/CAT->PROP.REL(1);
2  2     IMK2:    IF PROP.VALUE=1  THEN DO;
2  3          IF (PROP.LIM=1 & PROP.TIL(1)=0) THEN GO TO AME;
2  3          DO IT=1 TO PROP.LIM;
2  4          STATE(2,PROP.TIL(IT))=PROP.REL(2);
2  4          END;
2  3          GO TO AME;
2  3          END;
2  2          ELSE DO;

2  3            IF (PROP.LIM=1 & PROP.TIL(1)=0)  THEN GO TO AME;
2  3          DO IT=1 TO PROP.LIM;
2  4       STATE(2,PROP.TIL(IT))=STATE(1,PROP.TIL(IT))*PROP.REL(2)/
                PROP.REL(1);
2  4          END;
2  3          GO TO AME;
2  3          END;
2  2   EMA1:      PR=PROP.HOST;
2  2          TIERRA=PR;
2  2            QT=PER.DEXTER;
2  2       IF CAT->PROP.VALUE=1 THEN PER.REL(2)=CAT->PROP.REL(2);
2  2      ELSE PER.REL(2)=PER.REL(1)*CAT->PROP.REL(2)/CAT->PROP.REL(1);
2  2    IMK3:      IF PROP.VALUE=1  THEN DO;
2  3          IF (PROP.LIM=1 & PROP.TIL(1)=0)  THEN DO;
2  4            PROP.REL(2)=0;
2  4          GO TO EME1;
2  4            END;
2  3          ELSE PROP.REL(2)=PER.REL(2);
2  3          DO IT=1 TO PROP.LIM;
2  4          STATE(2,PROP.TIL(IT))=PROP.REL(2);
2  4          END;
2  3          GO TO EME1;
2  3          END;
2  2          IF PROP.VALUE=2 THEN DO;
2  3          IF (PROP.LIM=1 & PROP.TIL(1)=0)  THEN DO;
2  4          PROP.REL(2)=0;
2  4          GO TO EME1;
2  4          END;
2  3          ELSE PROP.REL(2)=PER.REL(2)*PROP.REL(1)/PER.REL(1);
2  3          DO IT=1 TO PROP.LIM;
2  4    STATE(2,PROP.TIL(IT))=STATE(1,PROP.TIL(IT))*PROP.REL(2)/
              PROP.REL(1);
2  4          END;
2  3          GO TO EME1;
2  3          END;
2  2          IF PROP.VALUE>2 THEN DO;
2  3          PROP.REL(2)=0;
2  3          IF (PER.RAM=1 & PER.TAR(1)=0) THEN GO TO AME;
2  3          DO IT=1 TO PER.RAM;
2  4          EX=IT;
2  4          CALL MINUP(EX);
2  4       STATE(2,PER.TAR(IT))=REY*PER.REL(2)/PER.REL(1);
2  4          END;
2  3          GO TO AME;
2  3            END;
2  2      EME1:      DO IT=1 TO PER.RAM;
2  3          EX=IT;
2  3          CALL MINUP(EX);
2  3          MA=PER.TAR(IT);
```

```
 2  3          DA=-CEIL(-MA/10000);
 2  3          JA=-CEIL(-MA/1000);
 2  3          JAK=JA-10*DA;
 2  3          NA=MA-1000*JA;
 2  3          IF (JAK¬=2)  THEN STATD(2,NA)=REY*PER.REL(2)/PER.REL(1);
 2  3          IF JAK=2  THEN DO;
 2  4          SNOT=REY*PER.REL(2)/PER.REL(1);
 2  4          PUT EDIT('NOTSTATE=',MA,'IMP=',SNOT)
                 (SKIP(2),A(9),F(5),X(2),A(4),E(18,6));
 2  4           END;
 2  3           END;
 2  2          GO TO AME;
                /*    NESTED  CASE  */
 2  2     EMA2:   PR=CAT->PROP.HOST;
 2  2          TIERRA=PR;
 2  2          QT=PER.DEXTER;
 2  2           DO IT=1 TO PER.LEAL;
 2  3          IF PER.KIM(IT)=GOLD(I) THEN GO TO EMA3;
 2  3          END;
 2  2          GO TO IMK4;
 2  2      EMA3:   IF CAT->PROP.VALUE<=2 THEN EX=IT+1+PER.RAM;
 2  2          ELSE IF (PER.RAM=1 & PER.TAR(1)=0)  THEN EX=IT;
 2  2          ELSE EX=IT+PER.RAM;
 2  2          CALL MINUP(EX);
 2  2          IF (PROP.HOST¬=NULL) THEN DO;
 2  3          PROP.HOST->PER.REL(2)=REY*PER.REL(2)/PER.REL(1);
 2  3          PR=PROP.HOST;
 2  3          TIERRA=PR;
 2  3           QT=PER.DEXTER;
 2  3          GO TO IMK3;
 2  3          END;
 2  2          ELSE PROP.REL(2)=REY*PER.REL(2)/PER.REL(1);
 2  2           IF PROP.VALUE=1 THEN DO;
 2  3           IF(PROP.LIM=1 & PROP.TIL(1)=0) THEN GO TO AME;
 2  3          ELSE DO IT=1 TO PROP.LIM;
 2  4          STATE(2,PROP.TIL(IT))=PROP.REL(2);
 2  4          END;
 2  3          END;
 2  2           IF PROP.VALUE=2 THEN DO;
 2  3          IF (PROP.LIM=1 & PROP.TIL(1)=0)  THEN GO TO AME;
 2  3           ELSE DO IT=1 TO PROP.LIM;
 2  4          STATE(2,PROP.TIL(IT))=STATE(1,PROP.TIL(IT))*PROP.REL(2)/
                 PROP.REL(1);
 2  4           END;
 2  3           END;
 2  2        AME:    END;
 2  1          FREE GOLD;
 2  1           END;
 2  0          PUT SKIP(2) LIST('VESELY-FUSSELL IMPORTANCES');
 2  0           PUT SKIP(2) LIST('FREE EVENTS');
 2  0          DO I=1 TO FUN;
 2  1          PUT SKIP DATA(I,STATE(2,I));
 2  1     END;
 2  0           PUT SKIP(2) LIST('REPLICATED EVENTS');
 2  0          DO I=1 TO DUN;
 2  1          PUT SKIP DATA(I,STATD(2,I));
 2  1           END;
 2  0           PUT SKIP(2) LIST('MODULES');
 2  0          DO I=1 TO ID;
 2  1          PT=BOST(I);
 2  1         PUT EDIT('MODULE NAME=',PROP.NAME,'IMP=',PROP.REL(2))
                     (SKIP(1),A(12),F(5),X(2),A(4),E(18,6));
 2  1           IF (PROP.HOST¬=NULL) THEN DO;
 2  2          PUT EDIT('IMP=',PROP.HOST->PER.REL(2))
                            (X(6),A(4),E(18,6));
 2  2            END;
 2  1          END;
 2  0          END IMPORTANCE;
 1  0          END NÜMERO;
```

For the pressure tank fault tree example, procedure IMPORTANCE assigns the modular and basic event Vesely-Fussell importance values in the following steps

STEP 1 $I_{TOP}^{V.F.} = 1$

$$I_r^{V.F.} = \frac{P_r}{P(TOP)} = 2.49937 \times 10^{-1}$$

$$I_{M_1}^{V.F.} = \frac{P_{M_1}}{P(TOP)} = 7.500625 \times 10^{-1}$$

$$I_{M_4}^{V.F.} = I_{M_5}^{V.F.} = \frac{P_{M_4} P_{M_5}}{P(TOP)} = 4.49887 \times 10^{-1}$$

$$I_1^{V.F.} = I_{M_1}^{V.F.} \frac{P_1}{P_{M_1}} = 2.49937 \times 10^{-4}$$

$$I_2^{V.F.} = I_3^{V.F.} = I_4^{V.F.} = I_{M_1}^{V.F.} \frac{10^{-5}}{P_{M_1}} = 2.49937 \times 10^{-1}$$

STEP 2

$$I_{M_9}^{V.F.} = I_{M_4}^{V.F.} = 4.49887 \times 10^{-10}$$

$$I_5^{V.F.} = I_6^{V.F.} = I_7^{V.F.} = I_8^{V.F.} = I_9^{V.F.} = I_{10}^{V.F.} =$$

$$= I_{M_5}^{V.F.} \frac{10^{-5}}{P_{M_5}} = 7.49812 \times 10^{-11}$$

STEP 3

$$I_{11}^{V.F.} = I_{12}^{V.F.} = I_{13}^{V.F.} = I_{M_9}^{V.F.} = \frac{2x(10^{-5})^2}{P_{M_9}} = 2.99924 \times 10^{-10}$$

CHAPTER FOUR

NUCLEAR REACTOR SAFETY SYSTEM FAULT TREE EXAMPLES

IV.1. Introduction

The PL-MOD code was used to analyze a number of nuclear reactor safety system fault trees, and its performance and results were compared to those obtained using the minimal cut-set generation codes PREP and MOCUS.

The safety systems analyzed included:

(a) a Triga Scram Circuit [14] fault tree composed of 22 simple AND and OR gates, a 3-out of - 4 symmetric gate, 20 non-replicated basic events and 2 replicated events.

(b) A Standby Protective Circuit [18] fault tree composed of 19 gates, 24 non-replicated basic events and 5 replicated basic events.

PL-MOD executed the modularization of the SPC fault tree in a time comparable to that taken by MOCUS (.034 min.) to list the set of 100 minimal cut-sets associated with the fault tree. However, the execution time taken by PREP's deterministic routine COMBO was about 6 times longer (2 min.).

(c) A PWR High Pressure Coolent Injection System [20] reduced fault tree composed of 59 non-replicated gates, 4 replicated modular gates, 142 non-replicated basic components and 9 replicated basic components.

The execution time taken by PL-MOD to modularize this larger tree was about 25 times smaller (.081 min.) than that taken by MOCUS (2.015 min.) to generate the set of 2724 single, double, and triple fault cut-sets associated with the fault tree.

IV.2. Triga Scram Circuit

A simplified diagram of the TRIGA Scram Circuit [14] is shown in Figure 4.1, while Figure 4.2 shows the fault tree describing the possible combination of events causing a failure of the reactor to scram as required when the steady state reactor power exceeds a one megawatt level.

The triga circuit is turned on when an operator pushes the "power-on" switch. An operator key switch is placed in the reset position to momentarily energize relays R19 and R20, which in turn energize relays R7 to R12. The lower "B" contacts of each of the relays receive voltage from one of the corresponding instrument channels, thus maintaining the coils energized. The upper "A" contacts will maintain relay K1 energized and thus

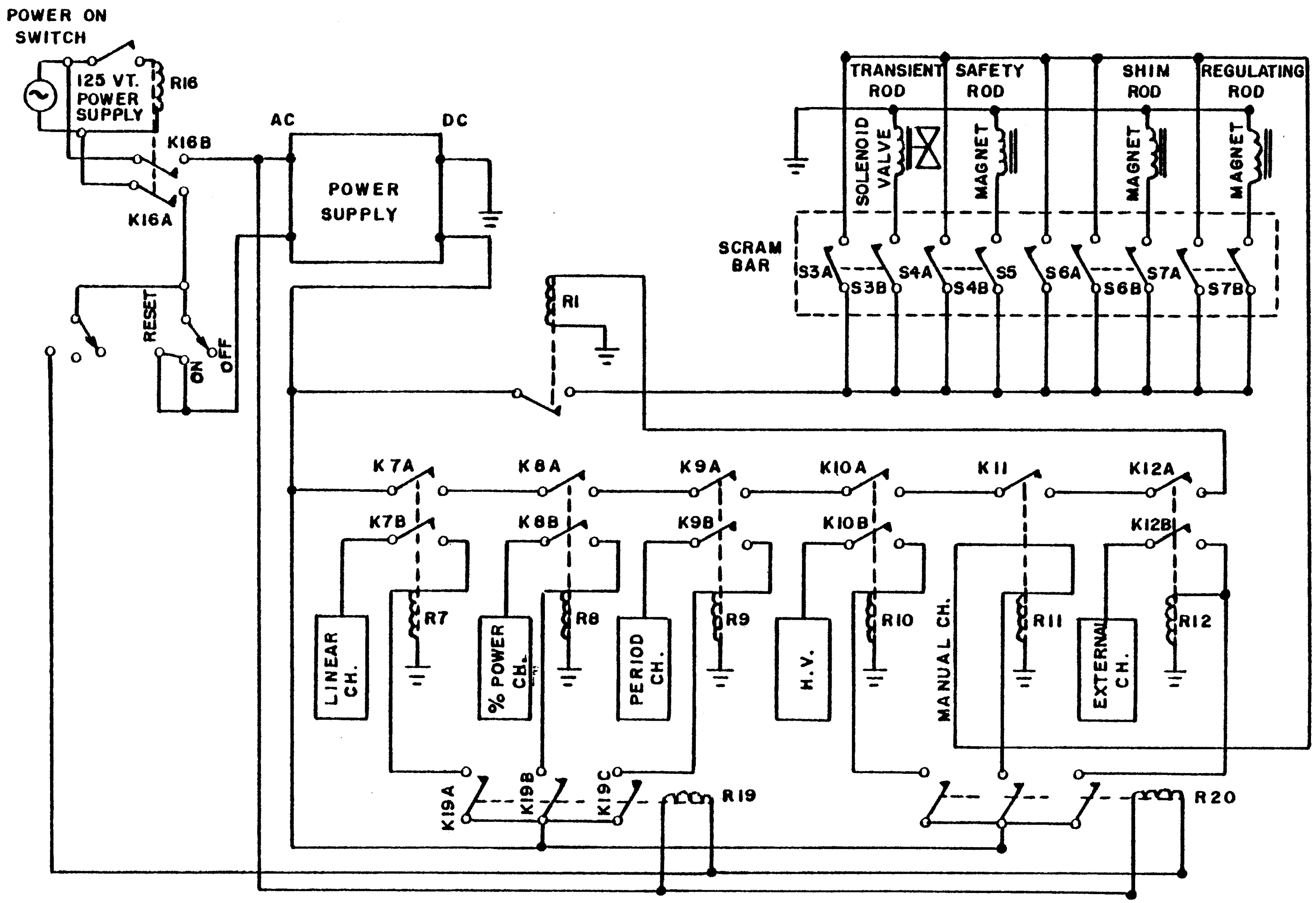


FIGURE 4.1 TRIGA Scram Circuit

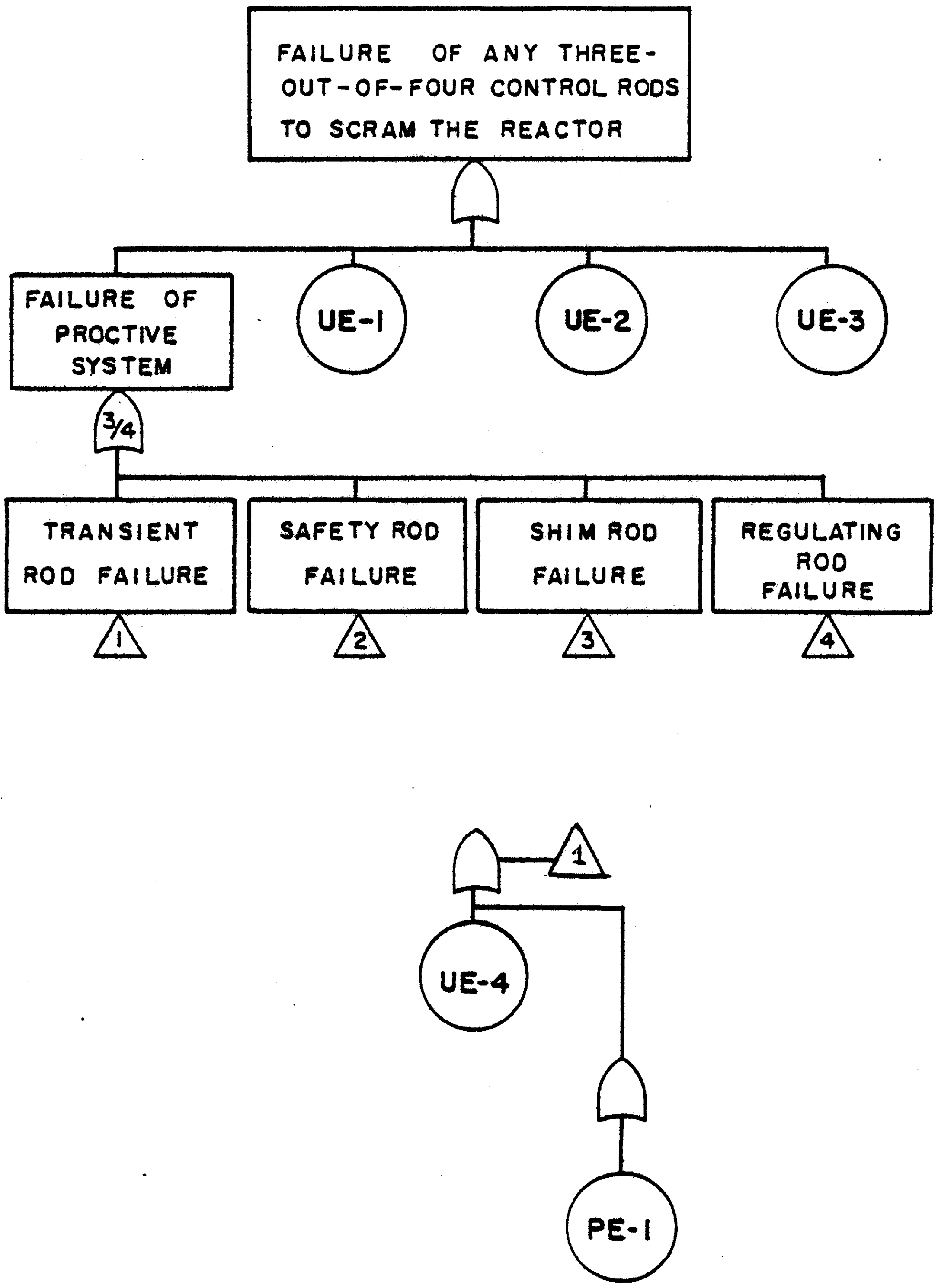


FIGURE 4.2 TRIGA Scram Fault Tree

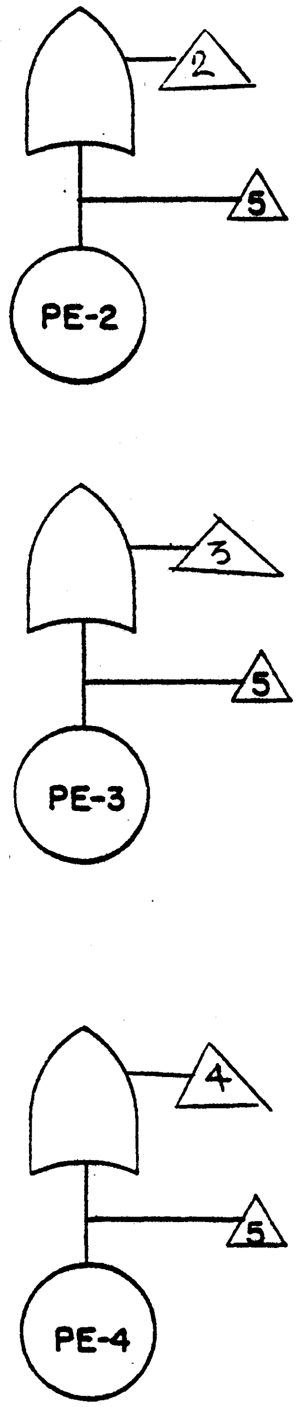


FIGURE 4.2 Continued

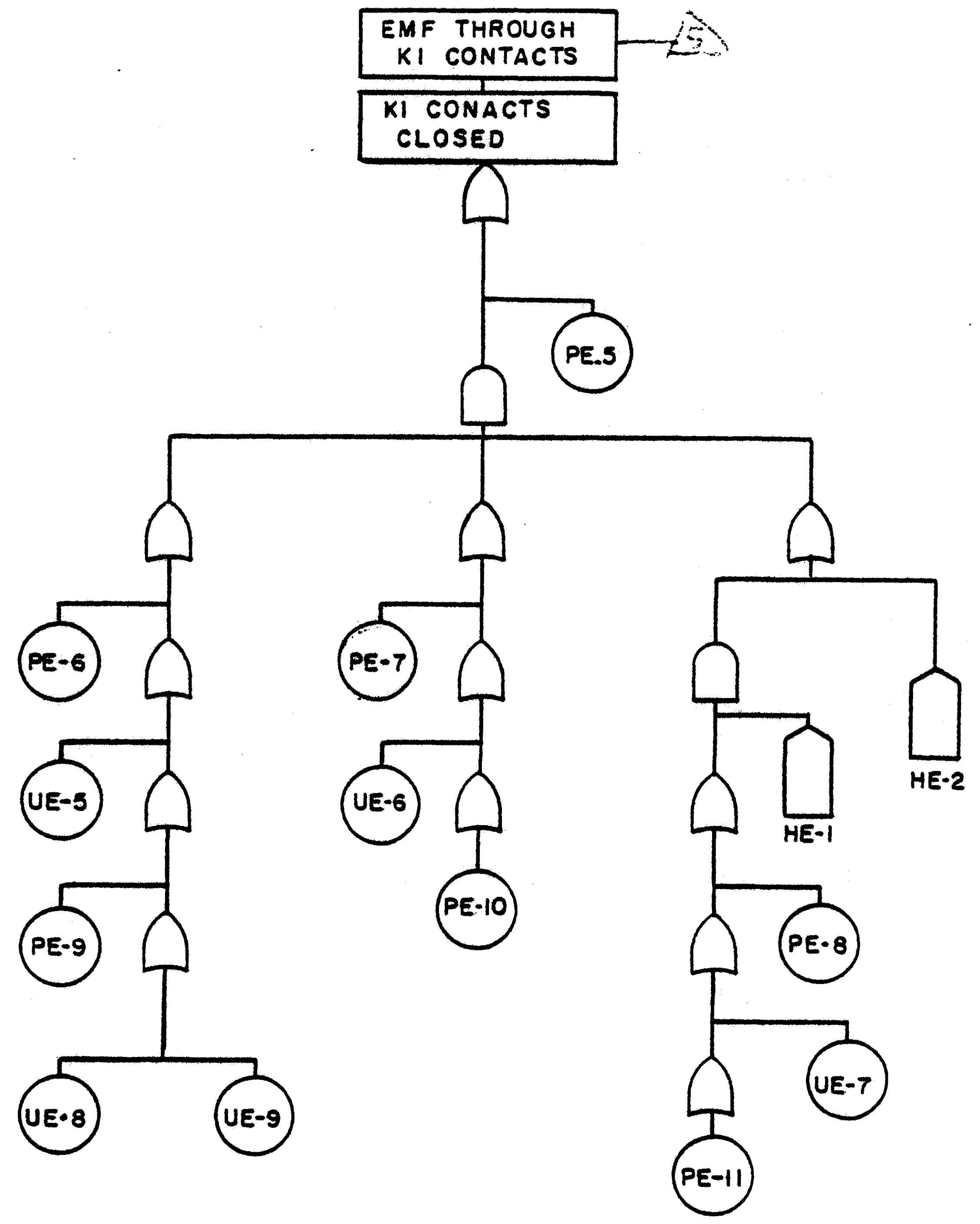


FIGURE 4.2 Continued

provide power to the magnets and solenoid valve. However, when any instrumentation channel interrupts its voltage supply to the corresponding relay, a scram control rod drop should occur due to a de-energized scram magnet or solenoid valve.

For a successful TRIGA reactor shut-down, at least 2 out of the 4 control rods must be inserted in the reactor. Hence G2 is a 3-out of-4 symmetric gate, since it is necessary that 3 out of the 4 control rod drop mechanisms fail to cause a TRIGA scram system failure. Notice that since relay K1 is common to each of the four rod-drop mechanisms, gate G8 may be taken as a direct input to gate G1.

In Table 4.1 the nomenclature identifying each basic event as well as its description and failure rate are given. The failure data are expressed in failures per cycle (there are 300 cycles per year assumed).

The modular structure determined by PL-MOD for the Triga scram fault tree is as follows:

G2: Symmetric 3-out of-4 module

$Y^B = (Y_{G3}, Y_{G5}, Y_{G6}, Y_7)$

$K_1 = (1, 1, 0, 1)$

$K_2 = (1, 0, 1, 1)$

$K_3 = (0, 1, 1, 1)$

$K_4 = (1, 1, 1, 0)$

$G_3 = \{1, 15; U\}$ $G_5 = \{2\}$, $G_6 = \{3\}$, $G_7 = \{4\}$

TABLE 4.1

TRIGA SCRAM CIRCUIT BASIC EVENT DATA

PL-MOD Identifier	Alphanumeric Identifier	Event Description	Failure Rate (Per Cycle)
1	PE-1	Solenoid Valve Fails to open	10^{-4}
2	PE-2	Electromagnet Safety rod shorts to ground	10^{-5}
3	PE-3	Electromagnet of Shim rod shorts to ground	10^{-5}
4	PE-4	Electromagnet of Regulat-ing rod shorts to ground	10^{-5}
5	PE-5	K1 Contacts fail to open	10^{-5}
6	PE-6	K7A Contacts fail to open	10^{-5}
7	PE-7	K8A contacts fail to open	10^{-5}
8	PE-8	K9A Contacts fail to open	10^{-5}
9	PE-9	K19A Contacts fail to open	10^{-5}
10	PE-10	K19B Contacts fail to open	10^{-5}
11	PE-11	K19C Contacts fail to open	10^{-5}
12	VE-1	Mechanical jamming of control rods	10^{-6}
13	VE-2	Gross movement of core	10^{-6}
14	VE-3	Control rods are of insuf-ficient worth	10^{-6}
15	VE-4	Air Tube to Piston Cham-ber clogged	10^{-5}
16	VE-5	Linear Channel remains energized when $P>1M_W$	10^{-4}
17	VE-6	% Power Channel remains energized when $P>1M_W$	10^{-4}

PL-MOD Identifier	Alphanumeric Identifier	Event Description	Failure Rate (Per Cycle)
18	VE-7	Period Channel fails to de-energize when $T<3$ sec.	10^{-4}
19	HE-1	$T<3$ sec when $P>1$ M_W	0.5
20	HE-2	T 3 sec when P 1 M_W	0.5
30001	VE-8	Reset Switch sticks in reset position	10^{-5}
30002	VE-9	External Force preventing switch from opening	10^{-5}

G9 = Higher Order Module

$(r_1 = 30001, r_2 = 30002)$

$Y^B = (Y_{r_1}, Y_{r_2}, Y_{m_9}, Y_{G10}, Y_{G13}, Y_{G17}, Y_{G18}, Y_{G19})$

$K_1 = (0, 0, 1, 1, 1, 1, 0, 0)$

$K_2 = (1, 0, 1, 0, 0, 1, 0, 0)$

$K_3 = (0, 1, 1, 0, 0, 1, 0, 0)$

$K_4 = (0, 0, 1, 1, 1, 0, 1, 1)$

$K_5 = (1, 0, 1, 0, 0, 0, 1, 0)$

$K_6 = (0, 1, 1, 0, 0, 0, 1, 0)$

G1: TOP gate event

G1 = 5, 12, 13, 14, G2, G9; U

Hence basic events 5, 12, 13 and 14 correspond to single event minimal cut-sets.

A list of all modular and single event minimal cut-set event occurrence probabilities (P) and Vesely-Fussell importance measures ($I^{V.F.}$) computed by PL-MOD for the fault tree after one cycle period is given in Table 4.2.

IV.3. Standby Protective Circuit

Figures 1.1 and 1.2 given in the thesis Introduction illustrate a standby Protective Circuit System's diagram and fault tree [18]. This system is similar to reactor protective circuits and is normally found in a standby mode. The purpose of the system is to recognize an abnormal pressure or level condition and then close a relay which initiates other action.

TABLE 4.2

OCCURRENCE PROBABILITIES AND VESELY-FUSSELL IMPORTANCE VALUES FOR THE TRIGA SCRAM FAULT TREE

Module	P	$I^{V.F.}$
G1	3.3007×10^{-5}	1
G9	2.0072×10^{-5}	6.0614×10^{-1}
G10	1.2×10^{-4}	2.1816×10^{-4}
G13	1.2×10^{-4}	2.1816×10^{-4}
G17	0.5	3.0318×10^{-1}
G18	0.5	3.0296×10^{-1}
G19	1.2×10^{-4}	2.6176×10^{-8}
G2	3.4×10^{-14}	1.0301×10^{-9}
G3	1.1×10^{-4}	10^{-9}
G5	10^{-5}	6.97×10^{-10}
G6	10^{-5}	6.97×10^{-10}
G7	10^{-5}	6.97×10^{-10}

Single Event Cut-Set	P	$I^{V.F.}$
5	10^{-5}	3.0296×10^{-1}
12	10^{-6}	3.0296×10^{-2}
13	10^{-6}	3.0296×10^{-2}
14	10^{-6}	3.0296×10^{-2}

The fault tree's top event corresponds to a failure of relay R3 contact #1 to close. Normally relays R1, R2 and R3 are deenergized. Relay R1 receives power if one of the branches of contacts in line with it permit current to flow (such as contacts LSA #1 and LSB #1). To be energized relay R2 requires that either contact R1 #1 or both manual switch MS1 and MS2 be closed. Relay R3 becomes energized if one pressure switch (PSA, PSB, or PSC) and the contact associated with relay R2 are closed (test switches TS1 and TS2 are not included in the fault tree). The nomenclature and unavailability data for each basic event are given in Table 4.3.

The minimal cut-set description for the SPC fault tree was given in Table 1.1 in the Introduction, while its modular structure determined by PL-MOD is as follows:

$G_{12} = \{4,7;U\}$ $G_{13} = \{5,8;U\}$ $G_{14} = \{6,9;U\}$

$G_6 = \{G_{12},G_{13},G_{14};\Omega\}$ Triple cut-sets)

$G_8 = \{17,18,19,20,21,22;\ U\}$

G_{16} = Higher Order Module

$Y^B = (Y_{r_1},\ Y_{r_2},\ Y_{r_3}, Y_{r_4},\ Y_{r_5},\ Y_{m_{16}},\ Y_{G_{17}},\ Y_{G_{18}},\ Y_{G_{19}})$

$K_1 = (1,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 1)$

$K_2 = (1,\ 0,\ 1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0)$

$K_3 = (1,\ 0,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 0)$

$K_4 = (1,\ 0,\ 0,\ 0,\ 1,\ 1,\ 0,\ 0,\ 0)$

$K_5 = (0,\ 0,\ 1,\ 0,\ 0,\ 1,\ 0,\ 1,\ 0)$

$K_6 = (0,\ 1,\ 1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0)$

TABLE 4.3

STANDBY PROTECTIVE CIRCUIT BASIC EVENT DATA

PL-MOD Identifier	Alphanumeric Identifier	Event Description	Unavailibility Per Demand
1	N.O.R1	Normally open contacts fail open	1.1×10^{-4}
2	N.O.R2		
3	N.O.R3		
4	APS	Pressure sensor fails	10^{-4}
5	BPS		
6	CPS		
7	N.O.AP	Normally Open Pressure sensor contacts fail open	4.3×10^{-4}
8	N.O.BP		
9	N.O.CP		
10	F1	Fuse Fails Open	3×10^{-4}
11	F2		
12	BAT	Battery Fails	1.1×10^{-3}
13	WSC	Wires short in circuit	1.1×10^{-4}
14	R1	Relay Fails on Demand	10^{-4}
15	R2		
16	R3		
17	MS1	Manual switch fails to function on demand	10^{-5}
18	MS2		
19	N.O.MS1	Manual Switch fails to close	3.6×10^{-5}
20	N.O.MS2		
21	OP.MS1	Operator does not initiate manual switch	10^{-3}
22	OP.MS2		
23	NO. LSA#2	Normally Open Level Sensor Contact fails Open	4.3×10^{-4}
24	NO. LSB#2		
20003	NO. LSA#1		
20004	NO. LSB#1		
20001	ALS	Level sensor fails	10^{-4}
20002	BLS		
20005	CLS		

K_7 = (0, 0, 1, 0, 1, 1, 0, 0, 0)

K_8 = (0, 1, 0, 0, 0, 1, 0, 0, 1)

K_9 = (0, 1, 0, 1, 0, 1, 0, 0, 0)

K_{10} = (0, 1, 0, 0, 1, 1, 0, 0, 0)

K_{11} = (0, 0, 0, 1, 0, 1, 0, 1, 0)

K_{12} = (0, 0, 0, 1, 1, 1, 0, 0, 0)

with r_1 = 20001, r_2 = 20003, r_3 20002, r_4 = 20004, r_5 = 20005. and

M_{16} = {empty set}

G_{17} ={empty set}

G_{18} = {23}

G_{19} = {24}

G_9 ={1, 4, G16; U}

G_7 ={G_8, G_9;Ω} (Double and Triple cut-sets)

TOP EVENT: G_1 = {2, 3, 10, 11, 12, 13, 15, 16, G6, G7;U}

Hence (2, 3, 10, 11, 12, 13, 15, 16) are single event cut-sets.

In Table 4.4 a list is provided of all modular and single event minimal cut-set unavailabilities (U) and Vesely-Fussell importances values ($I^{V.F.}$) computed by PL-MOD for the SPC fault tree.

TABLE 4.4

UNAVAILABILITIES AND VESELY-FUSSELL IMPORTANCE MEASURES FOR THE STANDBY PROTECTIVE CIRCUIT FAULT TREE

Module	U	$I^{V.F.}$
G1	3.2204×10^{-3}	1
G6	1.489×10^{-10}	4.623×10^{-8}
G7	4.4115×10^{-7}	1.37×10^{-4}
G8	2.092×10^{-3}	1.37×10^{-4}
G9	2.1087×10^{-4}	1.37×10^{-4}
G12	5.3×10^{-4}	4.623×10^{-8}
G13	5.3×10^{-4}	4.623×10^{-8}
G14	5.3×10^{-4}	4.623×10^{-8}
G16	8.748×10^{-7}	5.6827×10^{-7}
G18	1.1×10^{-4}	3.858×10^{-8}
G19	1.1×10^{-4}	3.858×10^{-8}

Single Event Cut-Set	U	$I^{V.F.}$
2	1.1×10^{-4}	3.416×10^{-2}
3	1.1×10^{-4}	3.416×10^{-2}
10	3×10^{-4}	9.315×10^{-2}
11	3×10^{-4}	9.315×10^{-2}
12	1.1×10^{-3}	3.416×10^{-1}
15	10^{-4}	3.105×10^{-2}
16	10^{-4}	3.105×10^{-2}

IV.4. High Pressure Injection System for a Pressurized Water Reactor

The PWR High Pressure Injection System (HPIS) is a part of the emergency coolant injection system (ECIS) which provides a high pressure source of emergency cooling water to the reactor coolant system (RCS) [20]. The HPIS is mainly used for small loss of coolant accident (LOCA) or secondary (steam) ruptures such that the RCS pressure is not low enough for use of the low pressure injection system (LPIS) or accumulator injection.

Figure 4.3 shows a simplified system diagram for the HPIS. The high pressure charging pumps are used to draw water from the refueling water storage tank (RWST) and injects the water at normal RCS pressure into the cold legs. Another function of the HPIS is to push the 12 weight percent boric acid solution in the 900 gallon boron injection tank (BIT) into the RCS to provide for a reactivity suppresion when a steam rupture occurs. The required flow for successful injection is 150 gpm, which corresponds to at least one charging pump function.

During normal operation, one operating charging pump draws water from the volume control tank (VCT) and discharges to the RCS through the open valves 1289A and 1289B. However, when the safety injection control system (SICS) is activated the following changes take place in the HPIS system configuration:

(1) The supply valves 1115B and 1115D are opened to allow the RWST to provide water for the HPIS pump suction.

(2) The standby charging pumps are started.

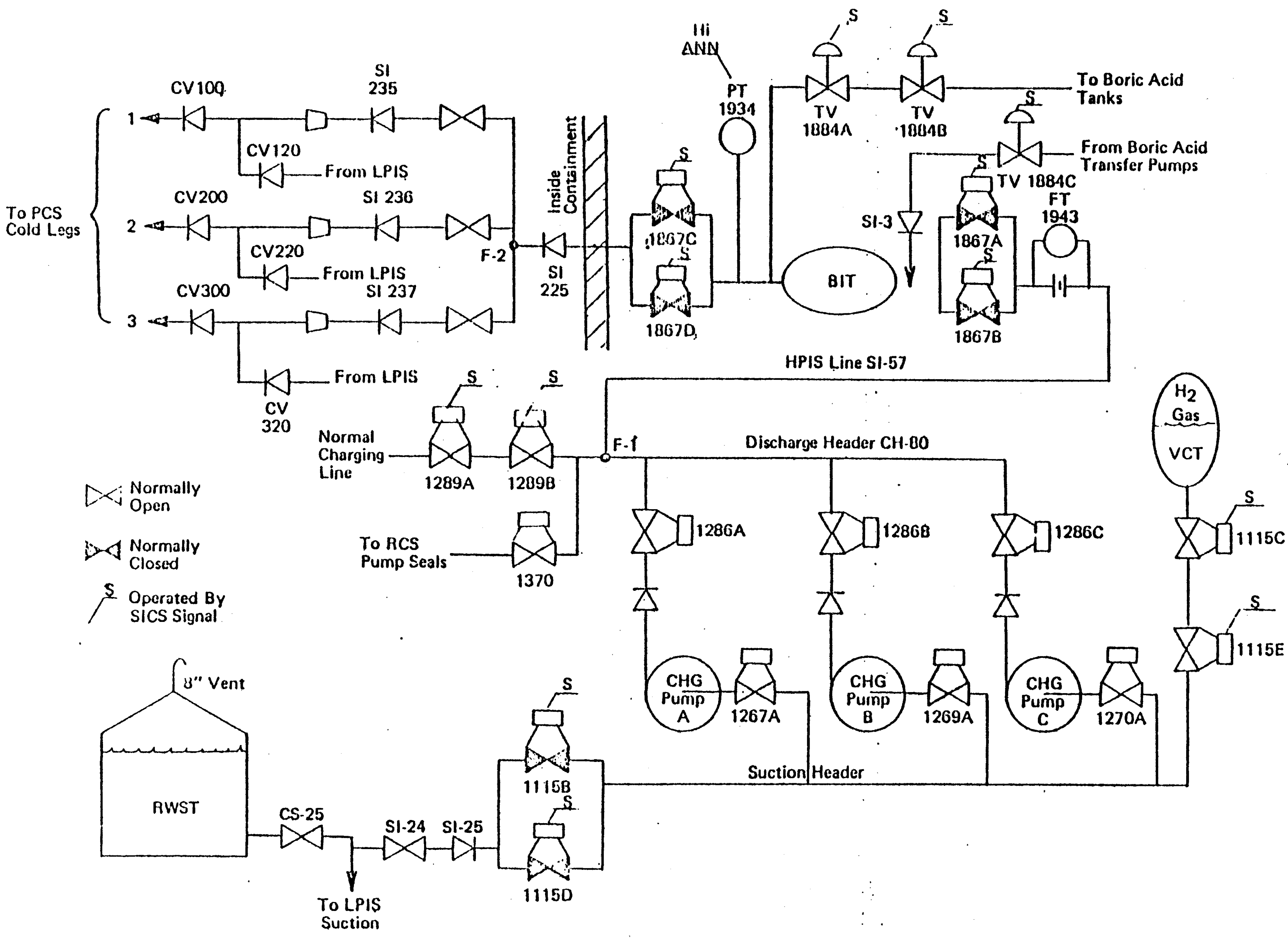


FIGURE 4.3

Simplified System Diagram

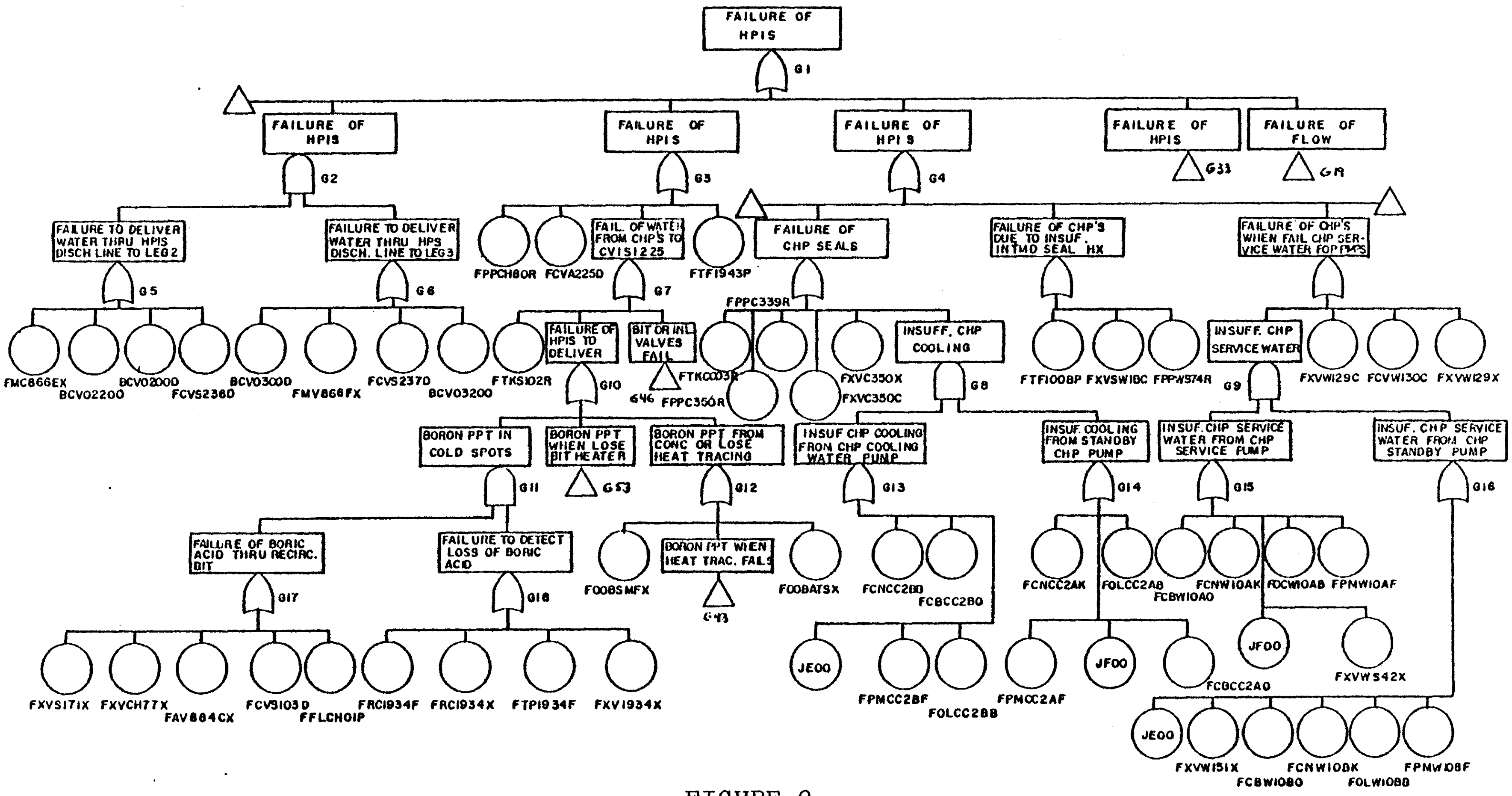


FIGURE 9

REDUCED FAULT TREE OF THE HPIS

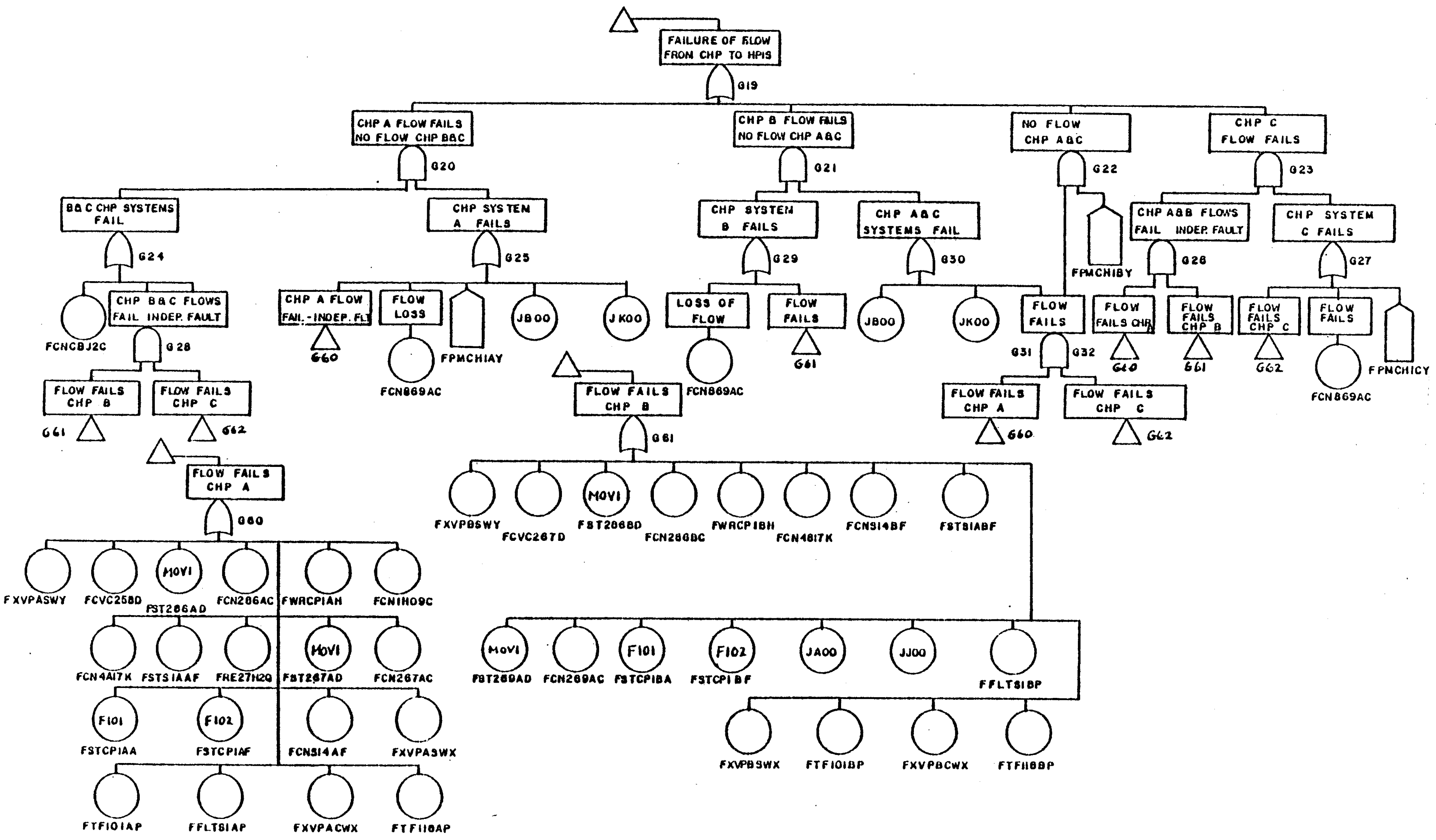


Figure 9 continued

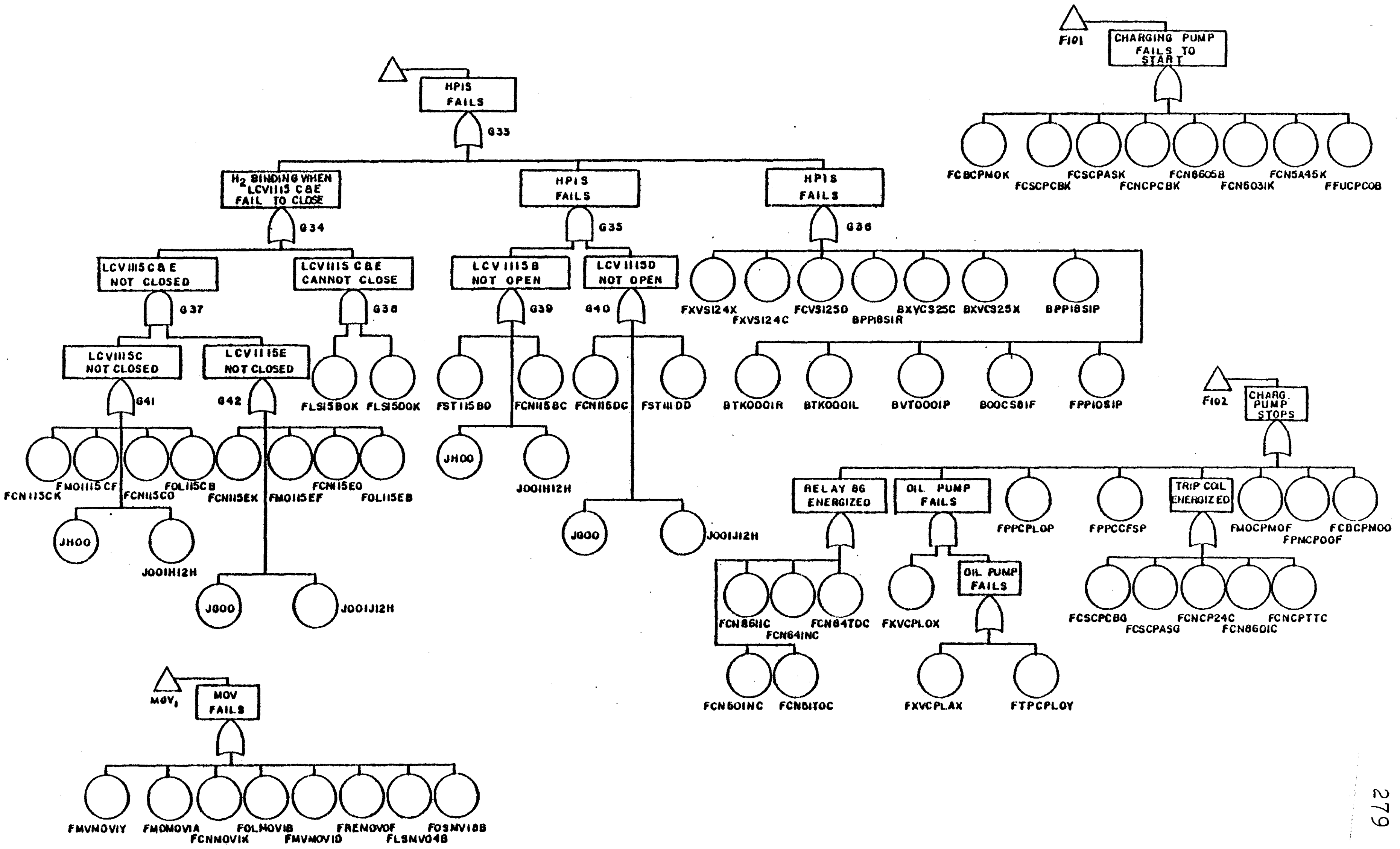


Figure 9 continued

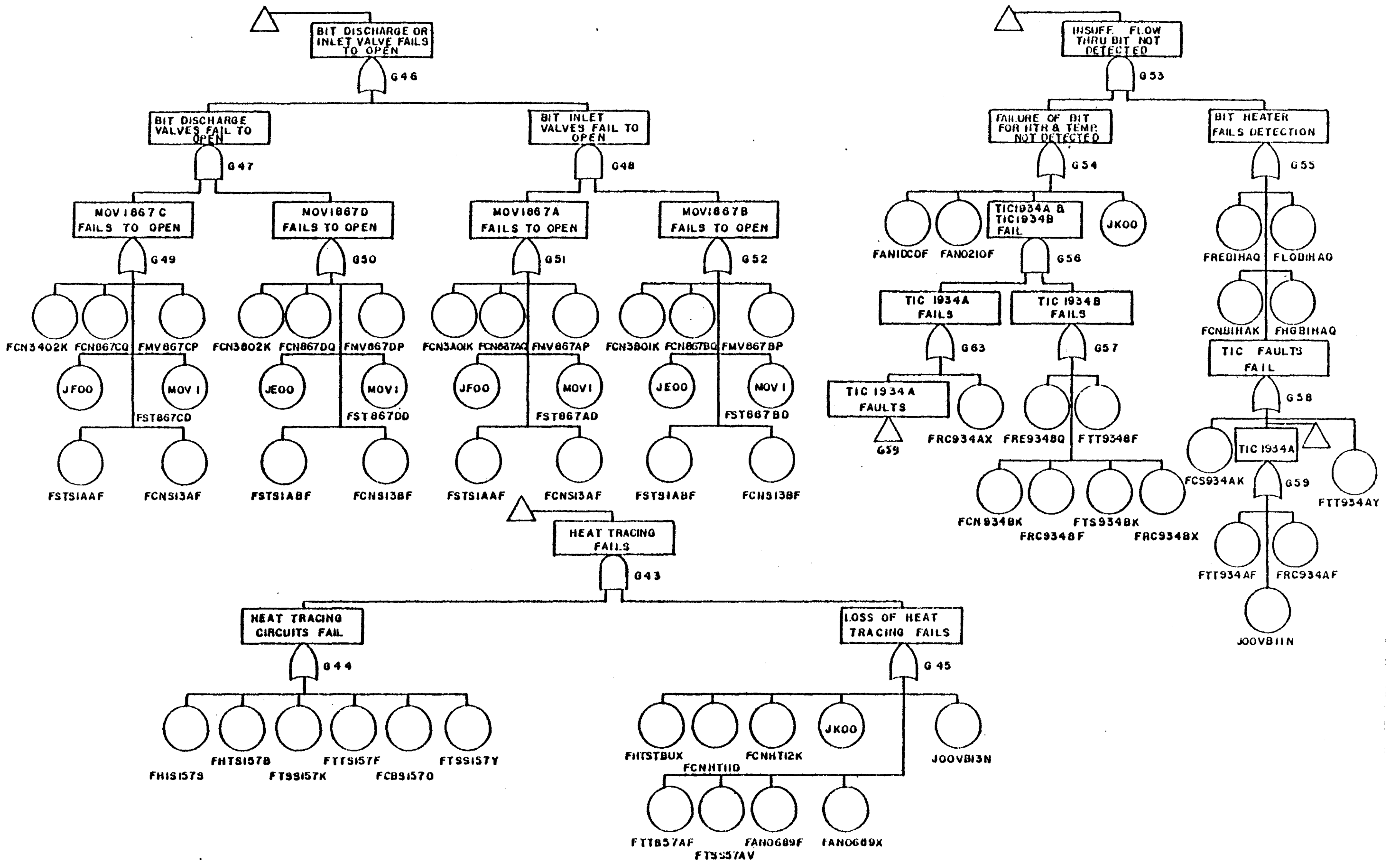


Figure 9 continued

(3) Isolation valves 1115C and 1115E are closed to prevent draining of the VCT.

(4) The normal charging line isolation valves 1289A and 1289B are closed.

(5) The isolation valves 1867A and 1967B at the BIT tank inlet are opened as well as the isolation valves 1967C and 1967D at the BIT outlet.

(6) The boric acid recirculation line trip valves are closed terminating recirculation between the Boric Acid Tanks (BAT) and the Boron Injection Tank (BIT).

(7) Charging System mini-flow valves are closed so that all operable charging pumps will pump water from the RWST to discharge header CH-80 through HPIS line S1-57, through the BIT, and to the RCS cold legs.

In the Reactor Safety Study, the HPIS unavailability estimates obtained were

$$U \text{ med} = 8.6 \times 10^{-3}$$

$$U \text{ lower} = 4.4 \times 10^{-3}$$

$$U \text{ upper} = 2.7 \times 10^{-2}$$

with the lower and upper bound evaluated by a Monte-Carlo simulation. The point estimates obtained were

$$U \text{ total} = 3.8 \times 10^{-3}$$

$$U \text{ singles} = 1.1 \times 10^{-3}$$

$$U \text{ doubles} = 2.5 \times 10^{-3}$$

$$U \text{ charging pump} = 7.0 \times 10^{-6}$$

U test and maintenance = $\varepsilon \approx 0$

The reduced fault tree given in the Reactor Safety Study for the HPIS system is shown in Figure 4.4. Each basic input event in the fault tree is labeled by an eight character code name [ ]. The coding scheme specifies the system, component type, identifier and failure mode for each basic event as follows:

TABLE 4.5

PWR SYSTEM IDENTIFICATION CODE

CODE	SYSTEM NAME
A	Accumulator (ACC)
G	Containment Leakage (CL)
N	Consequence Limiting Control System (CLCS)
K	Containment Heat Removal System (CHRS)
C	Containment Spray Injection System (CSIS)
D	Containment Spray Recirculation System (CSRS)
J	Electrical Power (EPS)
F	High Pressure Injection System (HPCIS)
H	High Pressure Recirculation System (HPCRS)
B	Low Pressure Injection System (LPIS)
E	Low Pressure Recirculation System (LPRS)
L	Sodium Hydroxide Addition System (SHAS)
I	Reactor Protection System (RPS)
M	Safety Injection Control System (SICS)
P	Auxiliary Feedwater (AF)

TABLE 4.6

COMPONENT CODE

Mechanical Components

Accumulator	AC	Sluice Gate	SL
Blower	BL	Sump	SP
Control Rod Drive Unit	CD	Subtree	ST
Cover Plate	FA	Tank	TK
Damper	DM	Tubing	TG
Diesel	DL	Turbine	TB
Expansion Joint	XJ	Valve, Check	CV
Filter or Strainer	FL	Valve, Explosive Operated	EV
Gas Bottle	GB	Valve, Hydraulic Operated	HV
Gasket	GK	Valve, Manual	XV
Heat Exchanger	HE	Valve, Motor Operated	MV
Nozzle	NZ	Valve, Pneumatic Operated	AV
Orifice	OR	Valve, Relief	RV
Pipe	PP	Valve, Safety	SV
Pipe Cap	CP	Valve, Solenoid Operated	KV
Pressure Vessel	PV	Valve, Stop Check	DV
Pump	PM	Valve, Vacuum Relief	VV
Reactor Control Rod	ED	Vent	VT
Refrigeration Unit	RF	Well	WL

TABLE 4.6 (Continued)

Electrical Components

Amplifier	AM	Ground Switch	GS
Annunciator	AN	Relay	RE
Battery	BY	Relay or Switch Contact	CN
Battery Charger	BC	Reset Switch	RS
Bus	BS	Resistor, Temp. Divice	RT
Cable	CA	Signal Comparator	AD
Circuit Breaker	CB	Switch, Pressure	PS
Clutch	CL	Switch, Torque	QS
Control Switch	CS	Switch, Temperature	TS
Coil	CO	Terminal Board	TM
Detector	DI	Diode or Rectifier	DE
DC Power Supply	DC	Fuse	FU
Flow Switch	FS	Generator	GE
Heating Element	HG	Heat Tracing	HT
Input Module	IM	Test Pushbutton	SB
Inverter (solid state)	IV	Thermal Overload	OL
Level Switch	ES	Timer	TI
Light	LT	Transformer, Current	CT
Limit Swtich	LS	Transformer, Potential (or control)	OT
Manual Switch	SW	Transformer, Power	TR
Motor	MO	Transmitter, Flow	TF
Motor Starter	MS	Transmitter, Level	TL
Neutron Detector	ND	Transmitter, Pressure	TP
Potentiometer	PT	Transmitter, Temperature	TT
Recorder	RC	Wire	WR
Lightning Arrester	LA	Event (where no component involved)	OO

TABLE 4.7

FAILURE MODE CODE

Failure Mode	
Closed	C
Disengaged	G
Does Not Close	K
Does Not Open	D
Does Not Start	A
Engaged	E
Exceeds Limit	M
Leakage	L
Loss of Function	F
Maintenance Fault	Y
No Input	N
Open	O
Open Circuit	B
Operational Fault	X
Overload	H
Plugged	P
Rupture	R
Short Circuit	Q
Short to Ground	S
Fault Transfer	T

Thus, for example, basic event FMV866FX refers to a High Pressure Injection System Motor Operated Valve tailoring due to an Operators error.

A large number of basic events shown in the reduced fault tree do not contribute to the system's failure since their unavailabilities were found to be negligible ($\varepsilon \to 0$) by the Reactor Safety Study. Table 4.8 is a list of those basic events which were included in the analysis performed by PL-MOD and MOCUS. The number identifying each event input along with its unavailability and alphanumeric identifier are given in the Table. A total of 142 non-replicated basic events, 9 replicated events adn 4 replicated modular gates were included in the reduced fault tree. PL-MOD computed a point unavailability

$$U = 4.71 \times 10^{-3}$$

for the HPIS reduced fault tree. The reduced fault tree was found to be representable by a 50 component Boolean vector higher order structure, i.e.

$$Y^B = (Y_{r_1}, \ldots, Y_{r_{13}}, Y_{m_0}, Y_{m_1}, \ldots, Y_{m_{36}})$$

Table 4.9 is the PL-MOD output giving the order in which each replicated event and nested module is listed in the Boolean vector, as well as the modular minimal cut-set matrix K representing the higher order gate.

Thus it may be seen by inspecting Table (4.9) that $r_1 = 20006$, $r_2 = 20005, \ldots\ldots, r_{13} = 29010$,

M_0 = G1 sub-module, M_1 = G8, M_2 = G9,...,M_{35} = G56, M_{36} = G63.

and

$$K = \begin{bmatrix} K_1 \\ K_2 \\ \vdots \\ K_{63} \end{bmatrix}$$

Notice that each modular cut-set may include single, double and triple basic event cut-sets. Thus for example K_1 consists of a single modular event $K_1 = (M_0)$ corresponding to the proper port attached to top gate G1. And as seen in Table (4.10)

M_0 ={48, 49, 50, 51, 52, 53, 54, 55, 1, 2, 3, 12, 13,, G2, G38, G11; U}

with G2 = {G5, G6;Ω}

G5 ={4, 5, 6, 7; U}   G6 ={8, 9, 10, 11; U}

G38 = {56, 57; Ω}

G11 = {G17, G18;Ω}

G17 = {30, 31, 32, 33, 34; U}   G18 ={36, 37, 38, 39; U}

Hence, K1 includes single as well as double basic event minimal cut-sets.

The modular gate event occurrence probabilities (unavailabilities) computed by PL-MOD for the reduced fault tree are given in Table 4.11. Thus for example gates G1, G5 and TOP have the unavailabilities

$$P(G1) = 1.126\text{x}10^{-3},\ P(G5) = 2.7 \times 10^{-3},$$
$$P(TOP) = 4.7118 \times 10^{-3}$$

It should be mentioned that "empty" nested AND gates appearing in a higher order structure are given a unit probability of occurrence (Figure 4.5). Thus, the fault tree shown in Figure 4.5 has the following cut-set description

(M_2 = empty AND gate)

$$K_1 = (0, 1, 0, 0, 0)$$
$$K_2 = (1, 0, 0, 0, 0)$$
$$K_3 = (0, 0, 1, 1, 1)$$

However, since $P(M_2\ ) \equiv 1$, then $P(K_3) = P_{M_3}P_{M_4}$ as required.

The modular Vesely-Fussell importance values are listed in Table (4.12). Thus, for example

$$I^{V.F.}_{TOP} = 1,\ I^{V.F.}_{M_0} = 2.39 \times 10^{-1},\ I^{V.F.}_{G59} = 2.08 \times 10^{-1}$$

The evaluation of the Vesely-Fussell importances may be seen to be particularly useful for cutting off unimportant portions of the fault tree before proceeding on to make a Monte-Carlo simulation to find upper and lower bounds on the uncertainty in the overall system unavailability. Thus, if for the HPIS reduced fault tree one were to cut off modules having an importance smaller than 2×10^{-2}, then its Boolean state vector representation would be considerably simplified to

$$Y^B = (Y_{r_1},\ldots,Y_{r_{13}},Y_{M_0},Y_{M_1},\ldots,Y_{M_{13}})$$

with

M1	=	G35
M2	=	G47
M3	=	G48
M4	=	G43
M5	=	G53
M6	=	G39
M7	=	G40
M8	=	G49
M9	=	G50
M10	=	G51
M11	=	G52
M12	=	G45
M13	=	G56

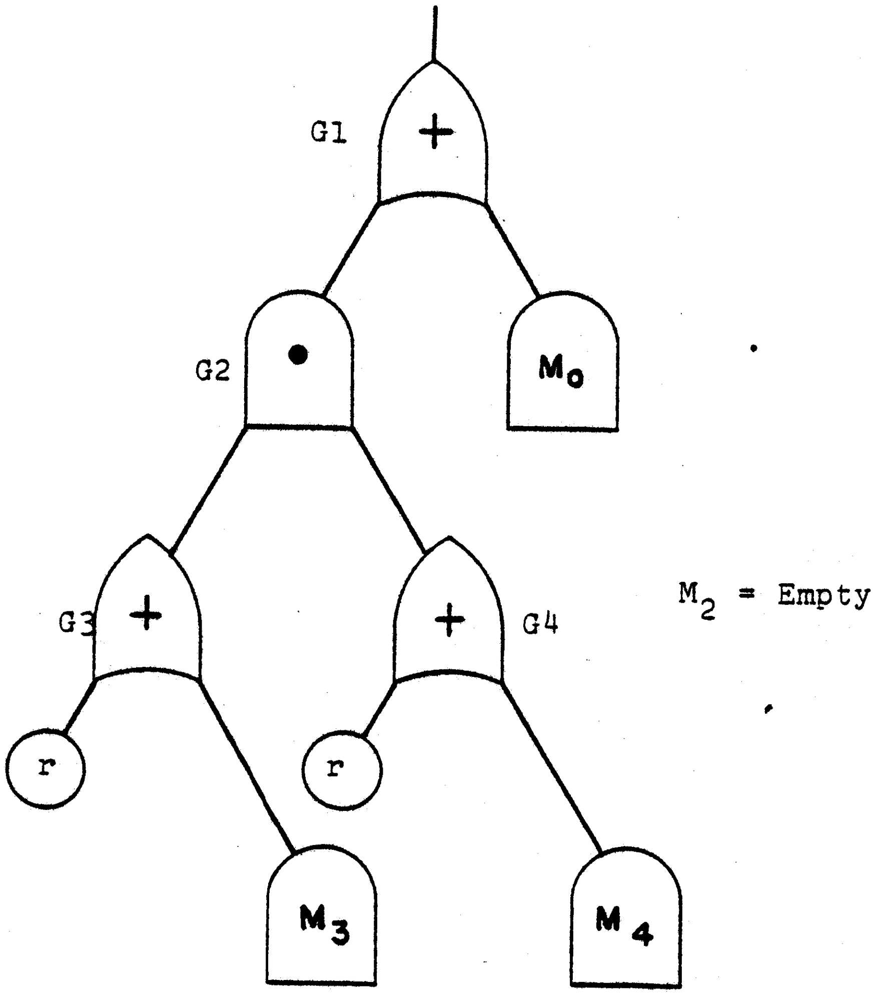


FIGURE 4.5

"EMPTY" NESTED AND GATE

TABLE 4.8

HPIS REDUCED FAULT TREE BASIC EVENT DATA

NUM FREE EVENT INPUTS= 142

NUM REPLICATED EVENT INPUTS= 13

FREE INPUT	RELIABILITY	ALPHANUMERIC
1	3.599999E-07	FPPCH80R
2	9.999999E-05	FCV3[illegible]D
3	3.599999E-07	FTF1913F
4	3.000000E-04	FMV866EX
5	1.300000E-03	FCV0[illegible]
6	9.999999E-05	FCV0220D
7	9.999999E-04	FCVS236D
8	9.999999E-04	FCV0300D
9	3.000000E-04	FMV866FX
10	9.999999E-04	FCVS237D
11	1.300000E-03	FCV03200
12	0.000000E+00	FTKSI02R

TABLE 4.8 (CONTINUED)

13	9.599999E-05	FOOBSMEX
14	4.299999E-05	FCNCC2BB
15	7.199999E-04	FCBCC2BO
16	1.400000E-03	FOLCC2BB
17	2.200000E-02	FPMCO2BF
18	5.500000E-04	FCNCC2AK
19	7.199999E-04	FOLCC2AB
20	9.999999E-04	FPMCC2AF
21	3.599999E-04	FCBCC2AO
22	7.199999E-04	FCBW1OAO
23	4.299999E-05	FCNW1OAK
24	1.400000E-03	FOLW1OAB
25	2.200000E-02	FPMWLOAF
26	3.599999E-04	FCBW1OBO
27	5.500000E-04	FCNWLOBK
28	7.199999E-04	FOLW1OBB
29	9.599999E-04	FPMW1OBF
30	3.999998E-04	FXVSI71X
31	1.500000E-03	FXVCH11X
32	4.999998E-04	FXVCH11C
33	3.000000E-04	FAVBF10C
34	9.599999E-05	FAVBF1CC
35	9.999999E-05	FCVSLO3D
36	2.699999E-05	FRC1951X
37	2.500000E-03	FRC1951F
38	2.500000E-03	FTP1951F
39	2.700000E-06	FXV1951X
40	9.999998E-03	FXVFALWY
41	9.999999E-05	FCVC25BD
42	1.799999E-05	FCN286AC

TABLE 4.8 (CONTINUED)

43	9.999999E-04	FWRCF1AH
44	1.799999E-05	FCN267AC
45	9.999999E-04	FXVPASWX
46	2.500000E-03	FST261AD
47	2.500000E-03	FST286AD
48	3.000000E-04	FXVSI24X
49	9.999999E-05	FXVSI24C
50	9.999999E-05	FCVSI25D
51	9.999999E-05	FXVCS25C
52	3.000000E-04	FXVCS25X
53	0.000000E+00	FPP16SIP
54	4.400000E-07	FVT0001P
55	4.400000E-07	FPP10SIP
56	3.000000E-04	FLS15BOK
57	3.000000E-04	FLS15DOK
58	2.200000E-04	FCN115BC
59	1.900000E-02	FST115BD
60	2.200000E-04	FCN115DC
61	1.900000E-02	FST115DD
62	7.799998E-03	FCN115CK
63	9.999999E-04	FMO115CF
64	8.799999E-05	FCN115CO
65	8.799999E-05	FOL115CB
66	7.799998E-03	FCN115EK
67	8.799999E-05	FCN115EO
68	8.799999E-05	FOL115EB
69	9.999999E-04	FMO115EF
70	3.000000E-04	FTSSI57Y
71	2.900000E-03	FHTSI57B
72	3.999998E-04	FTSSI57K

TABLE 4.8 (CONTINUED)

73	8.800000E-03	FTTSI57F
74	2.900000E-03	FCBSI57O
75	1.300000E-03	FCNHT11D
76	4.399999E-05	FCNHT12K
77	1.100000E-03	FTTS57AF
78	9.999999E-05	FTSS57AX
79	0.000000E+00	FANO68[illegible]F
80	3.000000E-02	FANO68[illegible]X
81	1.300000E-03	FCN3A02K
82	2.200000E-04	FCN867CQ
83	0.000000E+00	FMV867CP
84	1.900000E-02	FST867CD
85	1.300000E-03	FCN3C02K
86	2.200000E-04	FCN867DQ
87	0.000000E+00	FMV867DP
88	1.900000E-02	FST867DD
89	1.300000E-03	FCN3A01K
90	2.200000E-04	FCN867AQ
91	0.000000E+00	FMV867AP
92	1.900000E-02	FST867AD
93	1.300000E-03	FCN3B01K
94	2.200000E-04	FCN867BQ
95	0.000000E+00	FMV867BP
96	1.900000E-02	FST867BD
97	1.100000E-04	FAN1DC0F
98	9.999999E-04	FRC931BX
99	3.600000E-05	FKE931BX
100	1.100000E-02	FTT931BF
101	1.100000E-02	FRC931BF
102	1.100000E-04	FCN931BX

TABLE 4.8 (CONTINUED)

103	9.999999E-05	FTS934BK
104	7.200000E-05	FREBIHAQ
105	7.199999E-04	FOLBIHAQ
106	2.200000E-04	FCNBIHAQ
107	7.199999E-04	FHGBIHAQ
108	7.200000E-05	FC5934AK
109	9.999999E-04	FTT934AY
110	2.200000E-02	FTT934AF
111	2.200000E-02	FRC934AF
112	5.799998E-03	FSTS1AAF
113	1.330000E-03	FSTCP1AA
114	5.190000E-03	FSTCP1AF
115	9.999998E-03	FXVPBSWY
116	9.999999E-05	FCVC267D
117	1.799999E-05	FCN286BC
118	9.999999E-04	FWRCP1BH
119	1.799999E-05	FCN26[illegible]AC
120	9.999999E-04	FXVPBSWX
121	2.500000E-03	FST26[illegible]AD
122	2.500000E-03	FST286BD
123	5.799998E-03	FSTS1ABF
124	1.330000E-03	FSTCP1BA
125	5.190000E-03	FSTCP[illegible]
126	9.999998E-03	FXVPCSWY
127	9.999999E-05	FCVC270D
128	1.799999E-05	FCN286CC
129	9.999999E-04	FWRCP1CH
130	1.799999E-05	FCN270AC
131	9.999999E-04	FXVPCSWX
132	2.500000E-03	FST270AD

TABLE 4.8 (CONTINUED)

133	2.500000E-03	FST2BGCD
134	5.799998E-03	FSTS1ACF
135	1.330000E-03	FSTCPICA
136	5.190000E-03	FSTCPICB
137	1.900000E-02	FPMCHIAY
138	1.900000E-02	FPMCHIBY
139	1.900000E-02	FPMCHI[illegible]Y
140	9.999999E-05	FCNCBJ2C
141	4.099999E-05	JA00
142	1.099999E-06	JJ00
DEP INPUT RELIABILITY		
1	4.099999E-05	JE00
2	4.099999E-05	JF00
3	4.099999E-05	JB00
4	1.099999E-06	JK00
5	4.099999E-05	JG00
6	4.099999E-05	JH00
7	5.799998E-03	SIS1
8	5.799998E-03	SIS2
9	1.799999E-05	FCN[illegible]AC
10	0.000000E+00	F934
11	0.000000E+00	FCPA
12	0.000000E+00	FCPB
13	0.000000E+00	FCPC

TABLE 4.9

HPIS Reduced Falut Tree Minimal Cut-set Boolean Matrix

```
PARENT MODULE=    1  NUM DEP COMPONENTS=   13  NUM DEP MODULES=   36
DEP COMES=             20006          20005          49011          49013          30009
   39012               20003          40002          20007          49001          20008
   40004               29010
DEP MODS=                  8              9             35             37             22
      23                  20             21             47             48             43
      53                  13             14             15             16             39
      40                  41             42             27             25             24
      29                  30             49             50             51             52
      45                  55             54             28             31             56
      63

             MINIMAL CUT SETS
0000000000000100000000000000000000000000000000000
0011000000000000001000000000000000000000000000000
0000000000000010000000000011000000000000000000000
0000000100000010000000000010000000000000000000000
0000000001000010000000000001000000000000000000000
0000000101000010000000000000000000000000000000000
0000000000000001000000000000110000000000000000000
0000000001000001000000000000100000000000000000000
0000000100000001000000000000010000000000000000000
0000000101000001000000000000000000000000000000000
0000000000000000100000000000001100000000000000000
0100000000000000100000000000001000000000000000000
1000000000000000100000000000000100000000000000000
1100000000000000100000000000000000000000000000000
0000000000000000010000000000000011000000000000000
0100000000000000001000000000000010000000000000000
1000000000000000001000000000000000100000000000000
1100000000000000010000000000000000000000000000000
0010010000000001000000000000000000100000000000000
0011010000000000000100000000000000000000000000000
0010110000000000000100000000000000000000000000000
0000000000000000000001000000000000011000000000000
0000100000000000000001000000000000001000000000000
0010000000000000000010000000000000001000000000000
0000001000000000000010000000000000010000000000000
0000000000010000000010000000000000001000000000000
0000101000000000000001000000000000000000000000000
0000100000010000000001000000000000000000000000000
0000011000000000000000100000000000000000000000000
0000010000010000000010000000000000000000000000000
0000000000000000000000100000000000000001100000000
0000000001000000000000100000000000000001000000000
0000000000100000000000100000000000000000100000000
0000000100000000000000100000000000000000100000000

0000000101000000000000100000000000000000000000000
0000000100100000000000100000000000000000000000000
0000000010000000000000100000000000000000100000000
0000000011000000000000100000000000000000000000000
0000000010100000000000100000000000000000000000000
0000000000000000000000010000000000000000011000000
0000000001000000000000010000000000000000010000000
0000000000100000000000010000000000000000010000000
0000000100000000000000100000000000000000010000000
0000000101000000000000100000000000000000000000000
0000000100100000000000010000000000000000000000000
0000000001000000000000010000000000000000001000000
0000000011000000000000010000000000000000000000000
0000000010100000000000010000000000000000000000000
0000000000000000000000001000000000000000000100000
0000000000010000000000001000000000000000000000000
0000000000000000000000000100000000000000000110000
0000000000010000000000000100000000000000000100000
0000000000001000000000000100000000000000000010000
0000000000011000000000000100000000000000000000000
0001010000000000000010000000000000010000000001000
0001110000000000000010000000000000000000000001000
0011010000000000000010000000000000000000000001000
0001011000000000000010000000000000000000000001000
0001010000010000000010000000000000000000000001000
0011100000000000000001000000000000000000000000100
0011010000000000000001000000000000000000000000100
0000000000000000000000000100000000000000000100011
0000000000001000000000000100000000000000000000010
```

TABLE 4.10

HPIS Reduced Fault Tree Modular Components

```
FREE   MODULE NAME=    5  VALUE=    2  NUM LEAF INP=    4  NUM MOD INP=    1
LEAF INS=                        4                     5                      6                      7
MOD INS=                         0

FREE   MODULE NAME=    6  VALUE=    2  NUM LEAF INP=    4  NUM MOD INP=    1
LEAF INS=                        8                     9                     10                     11
MOD INS=                         0
  NESTID=    13
  NESTID=    14
  NESTID=    15
  NESTID=    16

FREE   MODULE NAME=   17  VALUE=    2  NUM LEAF INP=    6  NUM MOD INP=    1
LEAF INS=                       30                    31                     32                     33                     34
     35
MOD INS=                         0

FREE   MODULE NAME=   18  VALUE=    2  NUM LEAF INP=    4  NUM MOD INP=    1
LEAF INS=                       36                    37                     38                     39
MOD INS=                         0
  NESTID=    25
  NESTID=    27
  NESTID=    28
  NESTID=    29
  NESTID=    31

FREE   MODULE NAME=   38  VALUE=    1  NUM LEAF INP=    2  NUM MOD INP=    1
LEAF INS=                       56                    57
MOD INS=                         0
  NESTID=    39
  NESTID=    40
  NESTID=    41
  NESTID=    42

FREE   MODULE NAME=   44  VALUE=    2  NUM LEAF INP=    5  NUM MOD INP=    1
LEAF INS=                       70                    71                     72                     73                     74
MOD INS=                         0
  NESTID=    45
  NESTID=    49
  NESTID=    50
  NESTID=    51
  NESTID=    52
```

1 of 5

```
FREE   MODULE NAME=   57  VALUE=     2  NUM LEAF INP=     5  NUM MOD INP=     1
LEAF INS=                   99                   100                   101                   102                   103
MOD INS=                     0

FREE   MODULE NAME=   59  VALUE=     2  NUM LEAF INP=     2  NUM MOD INP=     1
LEAF INS=                  110                   111
MOD INS=                     0

FREE   MODULE NAME=   60  VALUE=     2  NUM LEAF INP=    11  NUM MOD INP=     1
LEAF INS=                   40                    41                    42                    43                    44
      45                    46                    47                   112                   113                   114
MOD INS=                     0

FREE   MODULE NAME=   61  VALUE=     2  NUM LEAF INP=    13  NUM MOD INP=     1
LEAF INS=                  115                   116                   117                   118                   119
     120                   121                   122                   123                   124                   125
     141                   142
MOD INS=                     0

FREE   MODULE NAME=   62  VALUE=     2  NUM LEAF INP=    11  NUM MOD INP=     1
LEAF INS=                  126                   127                   128                   129                   130
     131                   132                   133                   134                   135                   136
MOD INS=                     0
  NESTID=   63
  NESTID=   22
  NESTID=   55

FREE   MODULE NAME=    2  VALUE=     1  NUM LEAF INP=     1  NUM MOD INP=     2
LEAF INS=                    0
MOD INS=                     5                     6
  NESTID=    8
  NESTID=    9

FREE   MODULE NAME=   11  VALUE=     1  NUM LEAF INP=     1  NUM MOD INP=     2
LEAF INS=                    0
MOD INS=                    17                    18
  NESTID=   23
  NESTID=   24
  NESTID=   30
  NESTID=   35
  NESTID=   37
  NESTID=   43
  NESTID=   47
  NESTID=   48
  NESTID=   56
  NESTID=   20
  NESTID=   21
  NESTID=   54
  NESTID=   53

  TOTAL SUM REP=   38
BOCLEAN HAS BEEN CALLED

PARENT MODULE NAME=    1  VALUE=     2  NUM LEAF INP=    13  NUM MOD INP=     3
LEAF INS=                   48                    49                    50                    51                    52
      53                    54                    55                     1                     2                     3
      12                    13
MOD INS=                     2                    38                    11

NESTED MODULE NAME=    8  VALUE=     1  NUM LEAF INP=     1  NUM MOD INP=     1
```

```
LEAF INS=                 0
MOD INS=                  0

NESTED MODULE NAME=    9  VALUE=    1  NUM LEAF INP=    1  NUM MOD INP=    1
LEAF INS=                 0
MOD INS=                  0

NESTED MODULE NAME=   35  VALUE=    1  NUM LEAF INP=    1  NUM MOD INP=    1
LEAF INS=                 0
MOD INS=                  0

NESTED MODULE NAME=   37  VALUE=    1  NUM LEAF INP=    1  NUM MOD INP=    1
LEAF INS=                 0
MOD INS=                  0

NESTED MODULE NAME=   22  VALUE=    1  NUM LEAF INP=    1  NUM MOD INP=    1
LEAF INS=               138
MOD INS=                  0

NESTED MODULE NAME=   23  VALUE=    1  NUM LEAF INP=    1  NUM MOD INP=    1
LEAF INS=                 0
MOD INS=                  0

NESTED MODULE NAME=   20  VALUE=    1  NUM LEAF INP=    1  NUM MOD INP=    1
LEAF INS=                 0
MOD INS=                  0

NESTED MODULE NAME=   21  VALUE=    1  NUM LEAF INP=    1  NUM MOD INP=    1
LEAF INS=                 0
MOD INS=                  0

NESTED MODULE NAME=   47  VALUE=    1  NUM LEAF INP=    1  NUM MOD INP=    1
LEAF INS=                 0
MOD INS=                  0

NESTED MODULE NAME=   48  VALUE=    1  NUM LEAF INP=    1  NUM MOD INP=    1
LEAF INS=                 0
MOD INS=                  0

NESTED MODULE NAME=   43  VALUE=    1  NUM LEAF INP=    1  NUM MOD INP=    1
LEAF INS=                 0
MOD INS=                 44

NESTED MODULE NAME=   53  VALUE=    1  NUM LEAF INP=    1  NUM MOD INP=    1
LEAF INS=                 0
MOD INS=                  0

NESTED MODULE NAME=   13  VALUE=    2  NUM LEAF INP=    4  NUM MOD INP=    1
LEAF INS=                14                    15                    16                    17
MOD INS=                  0

NESTED MODULE NAME=   14  VALUE=    2  NUM LEAF INP=    4  NUM MOD INP=    1
LEAF INS=                18                    19                    20                    21
MOD INS=                  0

NESTED MODULE NAME=   15  VALUE=    2  NUM LEAF INP=    4  NUM MOD INP=    1
LEAF INS=                22                    23                    24                    25
MOD INS=                  0

NESTED MODULE NAME=   16  VALUE=    2  NUM LEAF INP=    4  NUM MOD INP=    1
```

```
LEAF INS=                 26                          27                         28                         29
MOD INS=                   0

NESTED MODULE NAME=   39  VALUE=    2  NUM LEAF INP=    2  NUM MOD INP=    1
LEAF INS=                 58                          59
MOD INS=                   0

NESTED MODULE NAME=   40  VALUE=    2  NUM LEAF INP=    2  NUM MOD INP=    1
LEAF INS=                 60                          61
MOD INS=                   0

NESTED MODULE NAME=   41  VALUE=    2  NUM LEAF INP=    4  NUM MOD INP=    1
LEAF INS=                 62                          63                         64                         65
MOD INS=                   0

NESTED MODULE NAME=   42  VALUE=    2  NUM LEAF INP=    4  NUM MOD INP=    1
LEAF INS=                 66                          67                         68                         69
MOD INS=                   0

NESTED MODULE NAME=   27  VALUE=    2  NUM LEAF INP=    1  NUM MOD INP=    1
LEAF INS=                139
MOD INS=                   0

NESTED MODULE NAME=   25  VALUE=    2  NUM LEAF INP=    1  NUM MOD INP=    1
LEAF INS=                137
MOD INS=                   0

NESTED MODULE NAME=   24  VALUE=    2  NUM LEAF INP=    1  NUM MOD INP=    1
LEAF INS=                140
MOD INS=                   0

NESTED MODULE NAME=   29  VALUE=    2  NUM LEAF INP=    1  NUM MOD INP=    1
LEAF INS=                  0
MOD INS=                   0

NESTED MODULE NAME=   30  VALUE=    2  NUM LEAF INP=    1  NUM MOD INP=    1
LEAF INS=                  0
MOD INS=                   0

NESTED MODULE NAME=   49  VALUE=    2  NUM LEAF INP=    4  NUM MOD INP=    1
LEAF INS=                 81                          82                         83                         84
MOD INS=                   0

NESTED MODULE NAME=   50  VALUE=    2  NUM LEAF INP=    4  NUM MOD INP=    1
LEAF INS=                 85                          86                         87                         88
MOD INS=                   0

NESTED MODULE NAME=   51  VALUE=    2  NUM LEAF INP=    4  NUM MOD INP=    1
LEAF INS=                 89                          90                         91                         92
MOD INS=                   0

NESTED MODULE NAME=   52  VALUE=    2  NUM LEAF INP=    4  NUM MOD INP=    1
LEAF INS=                 93                          94                         95                         96
MOD INS=                   0

NESTED MODULE NAME=   45  VALUE=    2  NUM LEAF INP=    6  NUM MOD INP=    1
LEAF INS=                 75                          76                         77                         78                         79
     80
MOD INS=                   0
```

4 of 5

TABLE 4.10 (CONTINUED)

```
NESTED MODULE NAME=    55   VALUE=     2   NUM LEAF INP=     6   NUM MOD INP=     1
LEAF INS=                  108                     109                    104                    105                    106
     107
MOD INS=                     0

NESTED MODULE NAME=    54   VALUE=     2   NUM LEAF INP=     1   NUM MOD INP=     1
LEAF INS=                   97
MOD INS=                     0

NESTED MODULE NAME=    28   VALUE=     1   NUM LEAF INP=     1   NUM MOD INP=     1
LEAF INS=                    0
MOD INS=                     0

NESTED MODULE NAME=    31   VALUE=     1   NUM LEAF INP=     1   NUM MOD INP=     1
LEAF INS=                    0
MOD INS=                     0

NESTED MODULE NAME=    56   VALUE=     1   NUM LEAF INP=     1   NUM MOD INP=     1
LEAF INS=                    0
MOD INS=                    57

NESTED MODULE NAME=    63   VALUE=     2   NUM LEAF INP=     1   NUM MOD INP=     1
LEAF INS=                   98
MOD INS=                     0
```

TABLE 4.11

HPIS REDUCED FAULT TREE MODULAR UNAVAILABILITIES

```
FREE MODULE
MODULE NAME=    5   REL=       2.700000E-03
FREE MODULE
MODULE NAME=    6   REL=       3.600000E-03
FREE MODULE
MODULE NAME=   17   REL=       2.899999E-03
FREE MODULE
MODULE NAME=   18   REL=       5.029697E-03
FREE MODULE
MODULE NAME=   38   REL=       8.999996E-08
FREE MODULE
MODULE NAME=   44   REL=       1.529999E-02
FREE MODULE


MODULE NAME=   57   REL=       2.224599E-02
FREE MODULE
MODULE NAME=   59   REL=       4.400000E-02
FREE MODULE
MODULE NAME=   60   REL=       2.945599E-02
FREE MODULE
MODULE NAME=   61   REL=       2.949807E-02
FREE MODULE
MODULE NAME=   62   REL=       2.945598E-02
FREE MODULE
MODULE NAME=    2   REL=       5.719997E-06
FREE MODULE
MODULE NAME=   11   REL=       1.458612E-05
REP MODULE=49011  REL=       2.945598E-02
REP MODULE=49013  REL=       2.945598E-02
REP MODULE=39012  REL=       2.949807E-02
REP MODULE=29010  REL=       4.400000E-02
PATRIARCH SUBMODULE
MODULE NAME=    1   REL=       1.125994E-03
NESTED MODULE
MODULE NAME=    8   REL=       1.000000E+00
NESTED MODULE
MODULE NAME=    9   REL=       1.000000E+00
NESTED MODULE
MODULE NAME=   35   REL=       1.000000E+00
NESTED MODULE
MODULE NAME=   37   REL=       1.000000E+00
NESTED MODULE
MODULE NAME=   22   REL=       1.900000E-02
NESTED MODULE
MODULE NAME=   23   REL=       1.000000E+00
NESTED MODULE
MODULE NAME=   20   REL=       1.000000E+00
NESTED MODULE
MODULE NAME=   21   REL=       1.000000E+00
NESTED MODULE
MODULE NAME=   47   REL=       1.000000E+00
NESTED MODULE
MODULE NAME=   48   REL=       1.000000E+00
NESTED MODULE
MODULE NAME=   43   REL=       1.529999E-02
NESTED MODULE
MODULE NAME=   53   REL=       1.000000E+00
NESTED MODULE
MODULE NAME=   13   REL=       2.416300E-02
NESTED MODULE
MODULE NAME=   14   REL=       2.630000E-03
NESTED MODULE
MODULE NAME=   15   REL=       2.416300E-02
NESTED MODULE
MODULE NAME=   16   REL=       2.630000E-03
NESTED MODULE
MODULE NAME=   39   REL=       1.921999E-02
NESTED MODULE
MODULE NAME=   40   REL=       1.921999E-02
NESTED MODULE
MODULE NAME=   41   REL=       8.975994E-03
NESTED MODULE
MODULE NAME=   42   REL=       8.975994E-03
NESTED MODULE
```

TABLE 4.11 (CONTINUED)

```
MODULE NAME=   27   REL=      1.900000E-02
NESTED MODULE
MODULE NAME=   25   REL=      1.900000E-02
NESTED MODULE
MODULE NAME=   24   REL=      9.999999E-05
NESTED MODULE
MODULE NAME=   29   REL=      0.000000E+00
NESTED MODULE
MODULE NAME=   30   REL=      0.000000E+00
NESTED MODULE
MODULE NAME=   49   REL=      2.051999E-02
NESTED MODULE
MODULE NAME=   50   REL=      2.051999E-02
NESTED MODULE
MODULE NAME=   51   REL=      2.051999E-02
NESTED MODULE
MODULE NAME=   52   REL=      2.051999E-02
NESTED MODULE
MODULE NAME=   45   REL=      3.254399E-02
NESTED MODULE
MODULE NAME=   55   REL=      2.803999E-03
NESTED MODULE
MODULE NAME=   54   REL=      1.100000E-04
NESTED MODULE
MODULE NAME=   28   REL=      1.000000E+00
NESTED MODULE
MODULE NAME=   31   REL=      1.000000E+00
NESTED MODULE
MODULE NAME=   56   REL=      2.224599E-02
NESTED MODULE
MODULE NAME=   63   REL=      9.999999E-04
PATRIARCH MODULE
MODULE NAME=    1   REL=      4.711870E-03
INDEX=
```

TABLE 4.12

HPIS REDUCED FAULT TREE VESELY-FUSSELL MODULAR IMPORTANCES

```
MODULES
MODULE NAME=    5   IMP=      2.062873E-03
MODULE NAME=    6   IMP=      2.062873E-03
MODULE NAME=   17   IMP=      3.095610E-03
MODULE NAME=   18   IMP=      3.095610E-03
MODULE NAME=   38   IMP=      1.910067E-05
MODULE NAME=   44   IMP=      1.056777E-01
MODULE NAME=   57   IMP=      2.077488E-01
MODULE NAME=   59   IMP=      2.087731E-01
MODULE NAME=   60   IMP=      2.392970E-02
MODULE NAME=   61   IMP=      2.358088E-02
MODULE NAME=   62   IMP=      2.331232E-02
MODULE NAME=    2   IMP=      2.062873E-03
MODULE NAME=   11   IMP=      3.095610E-03
MODULE NAME=    1   IMP=      2.389697E-01      IMP=      1.000000E+00
MODULE NAME=    8   IMP=      1.372042E-02
MODULE NAME=    9   IMP=      1.372042E-02
MODULE NAME=   35   IMP=      7.873422E-02
MODULE NAME=   37   IMP=      1.725560E-02
MODULE NAME=   22   IMP=      3.498702E-03
MODULE NAME=   23   IMP=      8.938856E-03
MODULE NAME=   20   IMP=      9.976272E-03
MODULE NAME=   21   IMP=      5.698875E-03
MODULE NAME=   47   IMP=      1.474788E-01
MODULE NAME=   48   IMP=      1.474788E-01
MODULE NAME=   43   IMP=      1.056777E-01
MODULE NAME=   53   IMP=      2.088525E-01
MODULE NAME=   13   IMP=      1.369718E-02
MODULE NAME=   14   IMP=      1.350981E-02
MODULE NAME=   15   IMP=      1.369718E-02
MODULE NAME=   16   IMP=      1.350981E-02
MODULE NAME=   39   IMP=      7.856667E-02
MODULE NAME=   40   IMP=      7.856667E-02
MODULE NAME=   41   IMP=      1.717714E-02
MODULE NAME=   42   IMP=      1.717714E-02
MODULE NAME=   27   IMP=      3.503702E-03
MODULE NAME=   25   IMP=      3.906935E-03


MODULE NAME=   24   IMP=      1.029656E-03
MODULE NAME=   29   IMP=      0.000000E+00
MODULE NAME=   30   IMP=      0.000000E+00
MODULE NAME=   49   IMP=      1.148009E-01
MODULE NAME=   50   IMP=      1.148009E-01
MODULE NAME=   51   IMP=      1.148009E-01
MODULE NAME=   52   IMP=      1.148009E-01
MODULE NAME=   45   IMP=      1.056741E-01
MODULE NAME=   55   IMP=      7.935319E-05
MODULE NAME=   54   IMP=      1.092653E-03
MODULE NAME=   28   IMP=      8.946620E-03
MODULE NAME=   31   IMP=      5.935154E-03
MODULE NAME=   56   IMP=      2.077488E-01
MODULE NAME=   63   IMP=      1.323841E-05

THE END
```

CHAPTER FIVE

CONCLUSIONS AND RECOMMENDATIONS

V.1. Summary and Conclusions

The methodology to analyze a fault tree in terms of its modular structure has been developed in this thesis. An algorithm to derive a fault tree's modular composition directly from its diagram was given. The procedure consists of piecewise collapsing and modularizing portions of the tree, until eventually the full tree structure is described as a set of modular equations recursively relating the top tree event to its basic component inputs.

The structural representation of fault trees containing replicated events was shown to necessitate the use of higher order gate modules. A Boolean vector representation was chosen to express the family of minimal cut-sets corresponding to a higher order gate.

Once the modular structure for a fault tree has been obtained, it was demonstrated how a quantitative evaluation of reliability and importance parameters may be efficiently performed. Thus, by following the same order in which the fault tree modules were originally found (i.e., starting with the bottom gate branches), each modular occurrence probability can can be easily computed as a function of the occurrence probabilities of its basic event and modular inputs. In contrast, basic event and modular Vesely-Fussell importance measures are best evaluated by starting at the top tree event and successively applying the modular importance chain rule.

The modular approach to fault tree analysis outlined above was implemented into the computer program PL-MOD. The code was written in PL/1 in order to take advantage of the list processing capabilities available in this computer language. In particular, extensive use was made of based structures, pointer variables and dynamical storage allocation. Moreover, the manipulation of Boolean state vectors, required to handle higher order modular structures, was conveniently performed using bit-string variables.

PL-MOD was used to analyze a number of nuclear reactor safety system fault trees, and its performance was tested against that of the minimal cut-set generation codes PREP and MOCUS. It was demonstrated that the code's execution time to modularize a larger sized fault tree will be significantly smaller than that taken to generate the thousands of minimal cut-sets required to characterize the fault tree. Thus, the execution time to modularize the High Pressure Injection System reduced fault tree, composed of 63 gates and 151 components, was 25 times faster than that taken by MOCUS to generate the 13 single event, 294 double event, and 2477 triple event minimal cut-sets associated with the fault tree. Furthermore, because of the structural organization of the modular information describing a fault tree, the evaluation of its reliability parameters is easier to perform using this information than from a mere listing of its minimal cut-sets.

V.2. Recommendations for Future Work

In its present form PL-MOD generates a complete Boolean vector representation for the modular minimal cut-sets of a fault tree. In practice, however, it is sufficient to generate those minimal cut-sets which significantly contribute to the occurrence of the top tree event. Thus, the incorporation in PL-MOD of a capability to generate only those modular minimal cut-sets which require the occurrence of less than N simultaneous modular events (with N = 2,3,4,etc.) would be highly desirable.

In the Reactor Safety Study reduced fault trees were derived by eliminating those basic events which contribute to the TOP tree event only through minimal cut-sets of high order, say quadruple or quintuple event cut-sets. This reduction procedure has however never been automated. PL-MOD would be particularly suited as a tool for deriving reduced fault trees, since the following two criteria for cutting off portions of a tree are available in the code:

(a) Modular events, rather than basic events, contributing to the top tree event only through minimal cut-sets of an order larger than N may be deleted as explained above.

(b) Once an upper limit N has been chosen, the Vesely-Fussell modular importances calculated by PL-MOD can be used to further reduce the tree by cutting off modules whose importances are smaller than a preselected cut-off value.

In order to handle more effectively fault trees which extensively include common mode failure events, it is recommended that the following two capabilities be incorporated into the PL-MOD code:

(a) In its present version, PL-MOD can only handle replicated modular gates, i.e., only replicated gates representing a supercomponent event independent from all other gates in the tree may be treated. In general, replicated gates may exist which do not represent a supercomponent event. Eliminating this restriction would significantly enhance the capabilities of the code.

(b) Similarly, PL-MOD allows the appearance of explicit symmetric (k-out of -n) gates, only if the inputs to these gates are non-replicated components or super-component events. It is proposed that symmetric gates be allowed to operate on input events which are replicated elsewhere in the fault tree.

Thus far, PL-MOD has been restricted to a deterministic evaluation of steady-state occurrence probabilities for a fault tree. Given the efficient recursive computational procedure used by the code, the inclusion of a time-dependent (kinetic) tree analysis capability as well as of a Monte-Carlo package enabling the code to perform a probabilistic distributional analysis would be justified.

REFERENCES

1. R.E. Barlow and F. Proschan; Statistical Theory of Reliabibility and Life Testing; Holt, Reinhart and Winston (1975).

2. R.E. Barlow and F. Proschan; Importance of System Components and Fault Tree Analysis; ORC-74-3 (1974).

3. R.E. Barlow and H.E. Lambert; Introduction to Fault Tree Analysis, Reliability and Fault Tree Analysis; SIAM (1975).

4. A. Blin et al; PATREC-DE Code: Evaluation of Common Mode Failures Impact on Reliability; Transactions on European Nuclear Society Conference (April, 1975).

5. Z.W. Birnbaum; On the Importance of Different Components in a Multicomponent System, Multivariate Analysis II, edited by P. Krisnaiah; Academic Press (1969).

6. P. Chatterjee; Fault Tree Analysis; Reliability Theory and Systems Safety Analysis; ORC 74-34(1974).

7. P. Chatterjee; Modularization of Fault Trees: A Method to Reduce the Cost of Analysis, Reliability and Fault Tree Analysis; SIAM (1975).

8. J.D. Esary and F. Proschan; Coherent Structures with Non-Identical Components; Technometrics 5 p. 191 (1963)

9. J.B. Fussell et al; MOCUS - A Computer Program to Obtain Minimal Sets from Fault Trees; Aerojet Nuclear Co. ANCR-1156 (August, 1974).

10. J.B. Fussell; Special Techniques for Fault Tree Analysis; Aerojet Nuclear No. (April, 1974).

11. I.B.M. Systems Reference Library; PL/1 Language Reference Manual and Programmer's Guide; C28-8201-2 and C28 -6594.

12. B.V. Koen and A. Carnino; Reliability Calculations with a List Processing Technique; IEEE Transactions on Reliability Vol. B-23 No. 1(April, 1974).

13. H.E. Lambert; Measures of Importance of Events and Cut-sets in Fault Trees, Reliability and Fault Tree Analysis; SIAM (1975).

14. H.E. Lambert; Fault Trees for Decision Making in Systems Analysis; UCRL-51829 (Oct., 1975).

15. J. Murchland; Fundamental Probability Relations for Repairable Items; NATO Advanced Study Institute on Generic Techniques in System Reliability Assessment, the University of Liverpool (July, 1973).

16. P.K. Pande et al; Computerized Fault Tree Analysis: TREEL and MICSUP; ORC 75-3 (1975).

17. W. Quine; The Problem of Simplifying Truth Functions, Am. Math. Monthly, 59(1952).

18. E.T. Rumble et al; Generalized Fault Tree Analysis for Reactor Safety; EPRI 217-2-2(1975).

19. Reactor Safety Study; Appendix II (Volume 1) Fault Tree Methodology; WASH-1400 Draft (August, 1974).

20. Reactor Safety Study; Appendix II (Volume 2) PWR Fault Trees; WASH-1400 Draft (August, 1974).

21. R.B. Worrell; Using the Set Equation Transformation System in Fault Tree Analysis, Reliability and Fault Tree Analysis; SIAM (1975).

22. R.B. Worrell and G.R. Burdick; Qualitative Analysis in Reliability and Safety Studies; IEEE Transactions on Reliability, Volume R-25, Number 3 (August, 1976).

23. W.E. Vesely and R.E. Narum; PREP and KITT: Computer Codes for the Automatic Evaluation of Fault Trees; Idaho Nuclear Co. (1970).

APPENDIX

PL-MOD'S INPUT AND OUTPUT DESCRIPTION

Data Input

No FORMAT restrictions exist as far as the listing of data items is concerned. Each data item is only required to be delimited by one or more blank spaces or a comma.

1st Item: 'TITLE' = a set of CHARACTERS enclosed by a pair of single quote marks.

2nd Item: DEL = number of reliability parameters to be computed (FIXED DECIMAL). (In the present PL-MOD version DEL = 1 or 2)

3rd Item: GUM = total number of fault tree gates (FIXED DECIMAL).

4th Item: RMOD = total number of replicated modules (FIXED DECIMAL).

5th Item: (I,AGIN(I), ALIL(I),ALIR(I))(FIXED DECIMAL)
I = gate number, AGIN(I) = number of gate inputs,
ALIL(I) = number of free leaf inputs,
ALIR(I) = number of replicated leaf inputs.
(1 = 1,2,...,GUM)

6th Item: (TRIM(IX), TRIN(IX))(FIXED DECIMAL)
TRIM(IX) = replicated leaf name associated with a module
TRIN(IX) = replicated gate number
(IX = 1,2,...,RMOD)

7th Item: NOR = total number of replicated leaf inputs (FIXED DECIMAL).

8th Item:

NODEIN(J): (NAME,VALUE,GIN,PIT(GIN),LIL,TIL(LIL),LIR,

TIR(LIR))(FIXED DECIMAL)

(J = 1,2,...,GUM)

NAME = gate number

$$\text{VALUE} = \begin{cases} 1 & \text{AND gate} \\ 2 & \text{OR gate} \\ \text{KON} & \text{K-out of-n gate} \end{cases}$$

GIN = number of gate inputs

PIT(I) = Ith gate input (I-1,2,...,GIN)

(If GIN = 0 then PIT = 0)

LIL = number of free leaf inputs

TIL(I) = Ith free leaf input (I = 1,2,...,LIL)

(If LIL = 0 then TIL = 0)

LIR = number of replicated leaf inputs

TIR(I) = Ith replicated leaf input (I=1,2,...,LIR)

(If LIR = 0 then TIR = 0)

(5th and 7th Items must be listed in the same order)

9th Item: FOX = 0 if no numerical evaluation is

desired, FOX = 1 otherwise

If FOX = 0 then delete items 10,11 and 12

10th Item: (FUN,DUN) (FIXED DECIMAL)

FUN = Total number of free leaf inputs

DUN = Total number of replicated leaf inputs

11th Item: (I,STATE(1,I)) (FIXED DECIMAL,FLOAT)

STATE(1,1) = probability associated with Ith free input

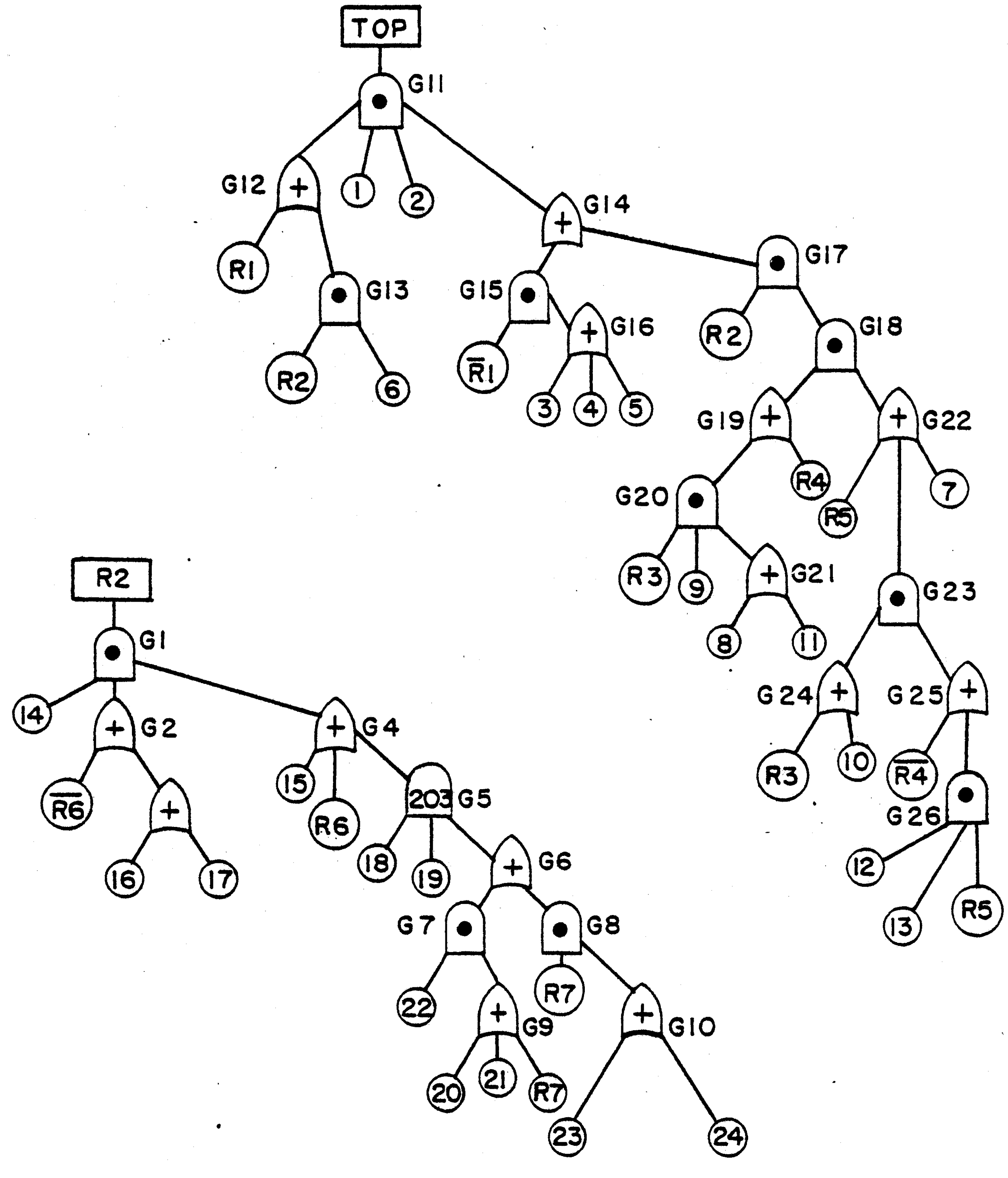
occurrence

(I = 1,2,...,FUN)

12th Item: (I, STATD (1,I)) (FIXED DECIMAL, FLOAT)

STATD (1,I) = probability associated with Ith replicated input (If Ith input is associated with a module then STAT D (1,I) = 0) (I = 1, ..., DUN)

An example of input data is given for the fault tree SAMPLE PROBLEM shown in Figure A-1. Table A-1 shows the input deck, whereas Table A-2 represents the output as given by PL-MOD.



SAMPLE FAULT TREE

```
'SAMPLE PROBLEM'          TABLE A-1          SAMPLE PROBLEM INPUT
2     26     1
1     2      1     0
2     1      0     1
3     0      2     0
4     1      1     1
5     1      2     0
6     2      0     0
7     1      1     0
8     1      0     1
9     0      2     1
10     0      2     0
11     2      2     0
12     1      0     1
13     0      1     1
14     2      0     0
 15     1      0      1
16     0      3     0
17     1      0     1
 18     2      0      0
19     1      0     1
20     1      1     1
21     0      2     0
22     1      1     1
23     2      0     0
24     0      1     1
25     1      0     1
26     0      2     1
29002     1
7
1. 1. 2 2 4. 1 14. 0 0.
2. 2. 1 3. 0 0. 1 22006.
3. 2. 0 0. 2 16 17. 0 0.
4. 2. 1 5. 1 15. 1 21006.
5. 203. 1 6. 2 18 19. 0 0.
6. 2. 2 7 8. 0 0. 0 0.
7. 1. 1 9. 1 22. 0 0.
8. 1. 1 10. 0 0. 1 20007.
9. 2. 0 0. 2 20 21.   1 20007.
10. 2. 0 0. 2 23 24. 0 0.
11. 1. 2 12 14. 2 1 2. 0 0.
12. 2. 1 13. 0 0. 1 21001.
13. 1. 0 0.     1 6. 1 29002.
14. 2. 2 15 17. 0 0. 0 0.
15. 1. 1 16. 0 0. 1 22001.
16. 2. 0 0. 3 3 4 5. 0 0.
17. 1 . 1 18. 0 0. 1 29002.
18. 1. 2 19 22. 0 0. 0 0.
19. 2. 1 20. 0 0. 1 21004.
20   . 1.  1 21. 1 9. 1 20003.
21. 2. 0 0. 2 8 11. 0 0.
22. 2. 1 23. 1 7. 1 20005.
23. 1. 2 24 25. 0 0. 0 0.
24. 2. 0 0. 1 10. 1 20003.
25. 2. 1 26. 0 0. 1 22004.
26. 1. 0 0. 2 12 13. 1 20005.
24     /
1      1.0E-01
2      1.0E-01
3      1.0E-02
4      1.0E-02
```

TABLE A-1 (CONTINUED)

5	1.0E-02		
6	1.0E-01		
7	1.0E-03	8	.5E-03
9	1.0E-03	10	.5E-03
11	1.0E-03	12	.5E-03
13	1.0E-03	14	.5E-03
15	1.0E-03	16	.5E-03
17	1.0E-03	18	.5E-03
19	1.0E-03	20	.5E-03
21	1.0E-03	22	.5E-03
23	1.0E-03	24	.5E-03
1	1.0E-01		
2	0		
3	1.0E-01		
4	1.0E-02		
5	1.0E-01		
6	.9		
7	1.0E-01		

TABLE A-2 SAMPLE PROBLEM OUTPUT

```
                                 TREE ANALYSIS BY MODULES

THE TIME IS             215558205

THE DATE IS             770620


SAMPLE  PROBLEM



OPTION=     2

NUM GATES=    26

NUM REPLICATED MODS=     1
        NODE     GATE INS FREE LEAVES  DEP LEAVES
           1            2           1           0
           2            1           0           1
           3            0           2           0
           4            1           1           1
           5            1           2           0
           6            2           0           0
           7            1           1           0
           8            1           0           1
           9            0           2           1
          10            0           2           0
          11            2           2           0
          12            1           0           1
          13            0           1           1
          14            2           0           0
          15            1           0           1
          16            0           3           0
          17            1           0           1
          18            2           0           0
          19          1          0          1

          20          1          1          1

          21          0          2          0

          22          1          1          1

          23          2          0          0

          24          0          1          1

          25          1          0          1

          26          0          2          1

RGATE=    1  LEAF=29002

NUMBER OF DEPENDENT COMPONENTS=     7

NODE=    1  VALUE=    1  GATE INPUTS=                 2                   4  FREE LEAF INPUTS=                     14   D
EP LEAF INPUTS=              0

NODE=    2  VALUE=    2  GATE INPUTS=                 3  FREE LEAF INPUTS=                    0  DEP LEAF INPUTS=
   22006

NODE=    3  VALUE=    2  GATE INPUTS=                 0  FREE LEAF INPUTS=                   16                      17   D
EP LEAF INPUTS=              0

NODE=    4  VALUE=    2  GATE INPUTS=                 5  FREE LEAF INPUTS=                   15  DEP LEAF INPUTS=
   21006

  DEP COMP=21006  APPEARANCES=   -2

NODE=    5  VALUE=  203  GATE INPUTS=                 6  FREE LEAF INPUTS=                   18                      19   D
EP LEAF INPUTS=              0

NODE=    6  VALUE=    2  GATE INPUTS=                 7                   8  FREE LEAF INPUTS=                      0   D
EP LEAF INPUTS=              0

NODE=    7  VALUE=    1  GATE INPUTS=                 9  FREE LEAF INPUTS=                   22  DEP LEAF INPUTS=
       0

NODE=    8  VALUE=    1  GATE INPUTS=                10  FREE LEAF INPUTS=                    0  DEP LEAF INPUTS=
   20007

NODE=    9  VALUE=    2  GATE INPUTS=                 0  FREE LEAF INPUTS=                   20                      21   D
EP LEAF INPUTS=          20007

  DEP COMP=20007  APPEARANCES=    2

NODE=   10  VALUE=    2  GATE INPUTS=                 0  FREE LEAF INPUTS=                   23                      24   D
EP LEAF INPUTS=              0

NODE=   11  VALUE=    1  GATE INPUTS=                12                  14  FREE LEAF INPUTS=                      1
       2   DEP LEAF INPUTS=                           0

NODE=   12  VALUE=    2  GATE INPUTS=                13  FREE LEAF INPUTS=                    0  DEP LEAF INPUTS=
   21001
```

```
NODE=   13  VALUE=    1  GATE INPUTS=                  0   FREE LEAF INPUTS=                    6   DEP LEAF INPUTS=
   29002

NODE=   14  VALUE=    2  GATE INPUTS=                 15                       17   FREE LEAF INPUTS=                    0  D
EP LEAF INPUTS=              0

NODE=   15  VALUE=    1  GATE INPUTS=                 16   FREE LEAF INPUTS=                    0   DEP LEAF INPUTS=
   22001

  DEP COMP=22001  APPEARANCES=   -2

NODE=   16  VALUE=    2  GATE INPUTS=                  0   FREE LEAF INPUTS=                    3                        4
      5   DEP LEAF INPUTS=                             0

NODE=   17  VALUE=    1  GATE INPUTS=                 18   FREE LEAF INPUTS=                    0   DEP LEAF INPUTS=
   29002

  DEP COMP=29002  APPEARANCES=    2

NODE=   18  VALUE=    1  GATE INPUTS=                 19                       22   FREE LEAF INPUTS=                    0  D
EP LEAF INPUTS=              0

NODE=   19  VALUE=    2  GATE INPUTS=                 20   FREE LEAF INPUTS=                    0   DEP LEAF INPUTS=
   21004

NODE=   20  VALUE=    1  GATE INPUTS=                 21   FREE LEAF INPUTS=                    9   DEP LEAF INPUTS=
   20003

NODE=   21  VALUE=    2  GATE INPUTS=                  0   FREE LEAF INPUTS=                    8                       11  D
EP LEAF INPUTS=              0

NODE=   22  VALUE=    2  GATE INPUTS=                 23   FREE LEAF INPUTS=                    7   DEP LEAF INPUTS=
   20005

NODE=   23  VALUE=    1  GATE INPUTS=                 24                       25   FREE LEAF INPUTS=                    0  D
EP LEAF INPUTS=              0

NODE=   24  VALUE=    2  GATE INPUTS=                  0   FREE LEAF INPUTS=                   10   DEP LEAF INPUTS=
   20003

  DEP COMP=20003  APPEARANCES=    2

NODE=   25  VALUE=    2  GATE INPUTS=                 26   FREE LEAF INPUTS=                    0   DEP LEAF INPUTS=
   22004

  DEP COMP=22004  APPEARANCES=   -2

NODE=   26  VALUE=    1  GATE INPUTS=                  0   FREE LEAF INPUTS=                   12                       13  D
EP LEAF INPUTS=          20005

  DEP COMP=20005  APPEARANCES=    2
  NESTID=    9

FREE   MODULE NAME=   10  VALUE=    2  NUM LEAF INP=    2  NUM MOD INP=    1
LEAF INS=                   23                        24
MOD INS=                     0
  NESTID=   13

FREE   MODULE NAME=   16  VALUE=    2  NUM LEAF INP=    3  NUM MOD INP=    1
LEAF INS=                    3                         4                        5
```

```
MOD INS=                          0

TREE   MODULE NAME=    21  VALUE=     2  NUM LEAF INP=     2  NUM MOD INP=     1
LEAF INS=                         8                      11
MOD INS=                          0
  NESTID=    24
  NESTID=    26
  NESTID=     2
  NESTID=     7
  NESTID=     8
  NESTID=    12
  NESTID=    15
  NESTID=    20
  NESTID=    25

  TOTAL SUM REP=     2
BOOLEAN HAS BEEN CALLED

PARENT MODULE NAME=     6  VALUE=     2  NUM LEAF INP=     1  NUM MOD INP=     1
LEAF INS=                         0
MOD INS=                          0

NESTED MODULE NAME=     7  VALUE=     1  NUM LEAF INP=     1  NUM MOD INP=     1
LEAF INS=                        22
MOD INS=                          0

NESTED MODULE NAME=     8  VALUE=     1  NUM LEAF INP=     1  NUM MOD INP=     1
LEAF INS=                         9
MOD INS=                         10

NESTED MODULE NAME=     9  VALUE=     2  NUM LEAF INP=     2  NUM MOD INP=     1
LEAF INS=                        20                      21
MOD INS=                          0
BICS                    COMP=                    '01000'B              COMP=                  '10010'B              COMP=
'00101'B                COMP=                    '10100'B

PARENT MODULE=     6  NUM DEP COMPONENTS=     1  NUM DEP MODULES=     1
DEP COMPS=                    20007
DEP MODS=                         7                       8                       9

            MINIMAL CUT SETS
10010
00101
10100

  NESTID=    19
  NESTID=    23

SYMM MODULE NAME=     5  VALUE=  203
DEP COMPS=                       18                      19
DEP MODS=                         6

            MINIMAL CUT SETS
101
011
110

  NESTID=    22
  NESTID=     4
  NESTID=    17
  TOTAL SUM REP=     2
BOOLEAN HAS BEEN CALLED

PARENT MODULE NAME=     1  VALUE=     1  NUM LEAF INP=     1  NUM MOD INP=     1
LEAF INS=                    14
MOD INS=                      0

NESTED MODULE NAME=     2  VALUE=     2  NUM LEAF INP=     2  NUM MOD INP=     1
LEAF INS=                    16                          17
MOD INS=                      0

NESTED MODULE NAME=     4  VALUE=     2  NUM LEAF INP=     1  NUM MOD INP=     1
LEAF INS=                    15
MOD INS=                      5
BICS                    COMP=                   '00111'B                COMP=                   '10110'B                COMP=
'01101'B                COMP=                   '11100'B

PARENT MODULE=     1  NUM DEP COMPONENTS=     2  NUM DEP MODULES=     2
DEP COMPS=                21006                   22004
DEP MODS=                     2                       4

            MINIMAL CUT SETS
00111
10110
01101

  NESTID=    14

  TOTAL SUM REP=    10
BOOLEAN HAS BEEN CALLED

PARENT MODULE NAME=    11  VALUE=     1  NUM LEAF INP=     2  NUM MOD INP=     1
LEAF INS=                     1                           2
MOD INS=                      0

NESTED MODULE NAME=    12  VALUE=     2  NUM LEAF INP=     1  NUM MOD INP=     1
LEAF INS=                     0
MOD INS=                      9

NESTED MODULE NAME=    14  VALUE=     2  NUM LEAF INP=     1  NUM MOD INP=     1
LEAF INS=                     0
MOD INS=                      0

NESTED MODULE NAME=    13  VALUE=     1  NUM LEAF INP=     1  NUM MOD INP=     1
LEAF INS=                     6
MOD INS=                      0

NESTED MODULE NAME=    15  VALUE=     1  NUM LEAF INP=     1  NUM MOD INP=     1
LEAF INS=                     0
MOD INS=                     16

NESTED MODULE NAME=    17  VALUE=     1  NUM LEAF INP=     1  NUM MOD INP=     1
LEAF INS=                     0
MOD INS=                      0

NESTED MODULE NAME=    19  VALUE=     2  NUM LEAF INP=     1  NUM MOD INP=     1
LEAF INS=                     0
MOD INS=                      0

NESTED MODULE NAME=    22  VALUE=     2  NUM LEAF INP=     1  NUM MOD INP=     1
```

```
LEAF INS=                      7
MOD INS=                       0

NESTED MODULE NAME=    20  VALUE=     1  NUM LEAF INP=     1  NUM MOD INP=     1
LEAF INS=                      9
MOD INS=                      21

NESTED MODULE NAME=    21  VALUE=     1  NUM LEAF INP=     1  NUM MOD INP=     1
LEAF INS=                      0
MOD INS=                       0

NESTED MODULE NAME=    24  VALUE=     2  NUM LEAF INP=     1  NUM MOD INP=     1
LEAF INS=                     10
MOD INS=                       0

NESTED MODULE NAME=    25  VALUE=     2  NUM LEAF INP=     1  NUM MOD INP=     1
LEAF INS=                      0
MOD INS=                       0

NESTED MODULE NAME=    26  VALUE=     1  NUM LEAF INP=     2  NUM MOD INP=     1
LEAF INS=                     12                  11
MOD INS=                       0
BICS                      COMP=       '00000001110000000000'B COMP=       '10000001010000000000'B COMP=
'00100000101100000000'B COMP=       '01000001100100000000'B COMP=       '11000001000100000000'B COMP=
'01100001001100000000'B COMP=       '00100001100011100000'B COMP=       '00100011100011000000'B COMP=
'00101001110001010000'B COMP=       '00101011100010000000'B COMP=       '10100001000011100000'B COMP=
'10100011100011000000'B COMP=       '10101001000010100000'B COMP=       '10101011000010000000'B COMP=
'00100001001011100000'B COMP=       '00100011001011000000'B COMP=       '00101001001010100000'B COMP=
'00101011001010000000'B COMP=       '00110001100010110000'B COMP=       '00110011100010010000'B COMP=
'10110001000010110000'B COMP=       '10110011000010010000'B COMP=       '00110001001010110000'B COMP=
'00110011001010010000'B COMP=       '00100001100011001110'B COMP=       '00100101100011001100'B COMP=
'00110001100011001010'B COMP=       '00110101100011001000'B COMP=       '00101001100010001110'B COMP=
'00101101100010001100'B COMP=       '00111001100010001010'B COMP=       '00111101100010001000'B COMP=
'00110001100010011110'B COMP=       '00110101100010011100'B COMP=       '00110001100010011010'B COMP=
'00110101100010011000'B COMP=       '10100001000011001110'B COMP=       '10100101000011001100'B COMP=
'10110001000011001010'B COMP=       '10110101000011001000'B COMP=       '10101001000010001110'B COMP=
'10101101000010001100'B COMP=       '10111001000010001010'B COMP=       '10111101000010001000'B COMP=
'10110001000010011110'B COMP=       '10110101000010011100'B COMP=       '10110001000010011010'B COMP=
'10110101000010011000'B COMP=       '00100001001011001110'B COMP=       '00100101001011001100'B COMP=
'00110001001011001010'B COMP=       '00110101001011001000'B COMP=       '00101001001010001110'B COMP=
'00101101001010001100'B COMP=       '00111001001010001010'B COMP=       '00111101001010001000'B COMP=
'00110001001010011110'B COMP=       '00110101001010011100'B COMP=       '00110001001010011010'B COMP=
'00110101001010011000'B COMP=       '00100011100011001101'B COMP=       '00110011100011001001'B COMP=
'00101011100010001101'B COMP=       '00111011100010001001'B COMP=       '00110011100010011101'B COMP=
'00110011100010011001'B COMP=       '10100011000011001101'B COMP=       '10110011000011001001'B COMP=
'10101011000010001101'B COMP=       '10111011000010001001'B COMP=       '10110011000010011101'B COMP=
'10110011000010011001'B COMP=       '00100011001011001101'B COMP=       '00110011001011001001'B COMP=
'00101011001010011101'B COMP=       '00111011001010001001'B COMP=       '00110011001010011101'B COMP=
'00110011001010011001'B

PARENT MODULE=    11  NUM DEP COMPONENTS=     7  NUM DEP MODULES=    12
DEP COMPS=                 21001                22001               29002               20003               21004
   22004                   20005
DEP MODS=                     12                   14                  13                  15                  17
      19                      22                   20                  23                  24                  25
      26

           MINIMAL CUT SETS
01100001001100000000
10101001000010100000
```

```
10101011000001000000
00101001001010100000
00101011001010000000
10110001000010110000
10110011000010011000
00110001001010110000
00110011001010010000
10110101000010011000
00110101001010011000

NUM FREE EVENT INPUTS=   24

NUM REPLICATED EVENT INPUTS=     7

  FREE INPUT RELIABILITY

           1       9.999996E-02

           2       9.999996E-02

           3       9.999998E-03

           4       9.999998E-03

           5       9.999998E-03

           6       9.999996E-02

           7       9.999999E-04

           8       4.999998E-04

           9       9.999999E-04

          10       4.999998E-04

          11       9.999999E-04

          12       4.999998E-04

          13       9.999999E-04

          14       4.999998E-04

          15       9.999999E-04

          16       4.999998E-04

          17       9.999999E-04

          18       4.999998E-04

          19       9.999999E-04

          20       4.999998E-04

          21       9.999999E-04

          22       4.999998E-04
          23      9.999999E-04

          24      9.999990E-04

   DEP INPUT RELIABILITY

           1      9.999996E-02

           2      0.000000E+00

           3      9.999996E-02

           4      9.999998E-03

           5      9.999996E-02

           6      9.000000E-01

           7      9.999996E-02
FREE MODULE
MODULE NAME=   10  REL=       1.500000E-03
FREE MODULE
MODULE NAME=   16  REL=       2.999999E-02
FREE MODULE
MODULE NAME=   21  REL=       1.500000E-03
PATRIARCH SUBMODULE
MODULE NAME=    6  REL=       0.000000E+00
NESTED MODULE
MODULE NAME=    7  REL=       4.999998E-04
NESTED MODULE
MODULE NAME=    8  REL=       1.500000E-03
NESTED MODULE
MODULE NAME=    9  REL=       1.500000E-03
PATRIARCH MODULE
MODULE NAME=    6  REL=       2.007499E-04
SYMM SUPERMODULE
MODULE NAME=    5  REL=       8.011244E-07
PATRIARCH SUBMODULE
MODULE NAME=    1  REL=       4.999998E-04
NESTED MODULE
MODULE NAME=    2  REL=       1.500000E-03
NESTED MODULE
MODULE NAME=    4  REL=       1.000801E-03
PATRIARCH MODULE
MODULE NAME=    1  REL=       7.257900E-07
REP MODULE=29002  REL=      7.257900E-07
PATRIARCH SUBMODULE
MODULE NAME=   11  REL=       9.999990E-03
NESTED MODULE
MODULE NAME=   12  REL=       0.000000E+00
NESTED MODULE
MODULE NAME=   14  REL=       0.000000E+00
NESTED MODULE
MODULE NAME=   13  REL=       9.999996E-02
NESTED MODULE
MODULE NAME=   15  REL=       2.999999E-02
NESTED MODULE
MODULE NAME=   17  REL=       1.000000E+00
NESTED MODULE
MODULE NAME=   19  REL=       0.000000E+00
```

```
NESTED MODULE
MODULE NAME=   22  REL=       9.999999E-04
NESTED MODULE
MODULE NAME=   20  REL=       1.500000E-06
NESTED MODULE
MODULE NAME=   23  REL=       1.000000E+00
NESTED MODULE
MODULE NAME=   24  REL=       4.999998E-04
NESTED MODULE
MODULE NAME=   25  REL=       0.000000E+00
NESTED MODULE
MODULE NAME=   26  REL=       4.999994E-07
PATRIARCH MODULE
MODULE NAME=   11  REL=       2.106253E-11
INDEX=    13PROP=     1
PATR=    11IMP=       1.000000E+00

I=     1PER.TAR=21001KEY=       7.331645E-13

I=     2PER.TAR=22001KEY=       1.959628E-11

NOTSTATE=22001   IMP=       9.303861E-01

I=     3PER.TAR=29002KEY=       2.106253E-11

I=     4PER.TAR=20003KEY=       2.375592E-16

I=     5PER.TAR=21004KEY=       1.466091E-12

I=     6PER.TAR=22004KEY=       2.155509E-16

NOTSTATE=22004   IMP=       1.021424E-05

I=     7PER.TAR=20005KEY=       1.451597E-12       BUG=                  1.30000E+01
GOLD=     1PROP=    12
GOLD=     2PROP=    14
GOLD=     3PROP=    13
GOLD=     4PROP=    15
GOLD=     5PROP=    17
GOLD=     6PROP=    19
GOLD=     7PROP=    22
GOLD=     8PROP=    20
GOLD=     9PROP=    23
GOLD=    10PROP=    24
GOLD=    11PROP=    25
GOLD=    12PROP=    26
GOLD=    13PROP=     1
HOSTPROP=     1

NOTSTATE=22006   IMP=       6.894559E-02       BUG=                  4.00000E+00
GOLD=     1PROP=    16
GOLD=     2PROP=    21
GOLD=     3PROP=     2
GOLD=     4PROP=     4       BUG=                  1.00000E+00
GOLD=     1PROP=     5
HOSTPROP=     5            BUG=                  1.00000E+00
GOLD=     1PROP=     6
HOSTPROP=     6            BUG=                  3.00000E+00
GOLD=     1PROP=     7
GOLD=     2PROP=     8
```

```
GOLD=     JPROP=     9    BUG=                        1.00000E+00
GOLD=     IPROP=    10    BUG=                        0.00000E+00

VESELY-FUSSELL IMPORTANCES

FREE EVENTS
I=          1             STATE(2,1)=  1.00000E+00;
I=          2             STATE(2,2)=  1.00000E+00;
I=          3             STATE(2,3)=  3.10128E-01;
I=          4             STATE(2,4)=  3.10128E-01;
I=          5             STATE(2,5)=  3.10128E-01;
I=          6             STATE(2,6)=  9.65193E-01;
I=          7             STATE(2,7)=  6.89184E-04;
I=          8             STATE(2,8)=  3.75444E-06;
I=          9             STATE(2,9)=  1.12783E-05;
I=         10             STATE(2,10)=  0.00000E+00;
I=         11             STATE(2,11)=  7.51888E-06;
I=         12             STATE(2,12)=  0.00000E+00;
I=         13             STATE(2,13)=  0.00000E+00;
I=         14             STATE(2,14)=  1.00000E+00;
I=         15             STATE(2,15)=  6.99237E-02;
I=         16             STATE(2,16)=  3.10351E-01;
I=         17             STATE(2,17)=  6.20702E-01;
I=         18             STATE(2,18)=  4.19804E-05;
I=         19             STATE(2,19)=  4.89970E-05;
I=         20             STATE(2,20)=  2.62213E-08;
I=         21             STATE(2,21)=  5.24426E-08;
I=         22             STATE(2,22)=  5.32293E-06;
I=         23             STATE(2,23)=  1.04885E-05;
I=         24             STATE(2,24)=  5.24426E-06;

REPLICATED EVENTS
I=          1             STATD(2,1)=  3.40039E-02;
I=          2             STATD(2,2)=  1.00000E+00;
I=          3             STATD(2,3)=  1.12783E-05;
I=          4             STATD(2,4)=  6.96066E-02;
I=          5             STATD(2,5)=  6.89184E-02;
I=          6             STATD(2,6)=  9.39020E-01;
I=          7             STATD(2,7)=  2.09770E-05;

MODULES
MODULE NAME=   10   IMP=      1.573200E-05
MODULE NAME=   16   IMP=      9.303861E-01
MODULE NAME=   21   IMP=      1.127833E-05
MODULE NAME=    6   IMP=      0.000000E+00        IMP=      2.105575E-05
MODULE NAME=    7   IMP=      5.322932E-06
MODULE NAME=    8   IMP=      1.573200E-05
MODULE NAME=    9   IMP=      7.866402E-08
MODULE NAME=    5   IMP=      0.000000E+00        IMP=      5.601764E-05
MODULE NAME=    1   IMP=      1.000000E+00        IMP=      1.000000E+00
MODULE NAME=    2   IMP=      9.310544E-01
MODULE NAME=    4   IMP=      6.997979E-02
MODULE NAME=   11   IMP=      1.000000E+00        IMP=      1.000000E+00
MODULE NAME=   12   IMP=      0.000000E+00
MODULE NAME=   14   IMP=      0.000000E+00
MODULE NAME=   13   IMP=      9.651930E-01
MODULE NAME=   15   IMP=      9.303861E-01
MODULE NAME=   17   IMP=      6.961781E-02
MODULE NAME=   19   IMP=      0.000000E+00
MODULE NAME=   22   IMP=      6.891850E-04
MODULE NAME=   20  IMP=       1.127831E-05
MODULE NAME=   21  IMP=       1.021424E-05
MODULE NAME=   24  IMP=       0.000000E+00
MODULE NAME=   25  IMP=       0.000000E+00
MODULE NAME=   26  IMP=       0.000000E+00

THE END
```