[illegible]

7/25/68

SYNTHETIC TREE MODEL

A FORMAL METHODOLOGY FOR FAULT TREE CONSTRUCTION

A THESIS

Presented to

The Faculty of the Division of Graduate

Studies and Research

By

Jerry Bernard Fussell

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

in the School of Nuclear Engineering

Georgia Institute of Technology

December, 1972

SYNTHETIC TREE MODEL

A FORMAL METHODOLOGY FOR FAULT TREE CONSTRUCTION

FOREWORD

I have enjoyed preparing this dissertation. The technical aspects were exciting but more important, the people involved, each in a special way, provided me an atmosphere in which to work that allowed me to use all my abilities as effectively as I could. Dr. C. J. Roberts began making this thesis a reality when he arranged for my being able to retain my financial aid while studying under Dr. W. E. Vesely at the Aerojet Nuclear Company in Idaho during the summer of 1971. Dr. Roberts did much to provide an environment that gave me, and other students, the freedom to learn. If I could repeat my dissertation experience, I would again choose Dr. J. D. Clement as my advisor. Dr. Clement always found time to help me and showed genuine concern. His guidance and patience were most valuable.

Dr. W. E. Vesely receives special thanks for the time and effort he gave so generously. Dr. Vesely introduced me to the dissertation topic and his continued enthusiasm kept my spirits high. His technical assistance was invaluable. I would also like to thank Dr. J. R. Penland of the Babcock and Wilcox Company for helping me get fault tree analysis in proper perspective and for his willingness to lend a hand when I needed help.

Every faculty member of the School of Nuclear Engineering here at the Georgia Institute of Technology has my thanks. Each gave me a great deal to appreciate. Additional gratitude is extended to members of my reading committee, Dr. G. G. Eichholz, Dr. W. W. Graham, III, and Dr. W. W. Hines, for their most valuable professional support.

To list my fellow students to whom I am indebted would be a much too ambitious task. Special consideration is given to the individuals in the 1972 summer quarter NE 707-F class, Reliability Analysis of Nuclear Reactor Systems. These students questioned and finally accepted Synthetic Tree Model as a valid, formal means of fault tree construction. They had a large influence concerning the manner in which the model is presented in this dissertation.

Recognition is given to Mrs. Lydia Geeslin, a truly professional typist, for producing a handsome thesis from a very rough looking rough draft.

My love for Nancy and our son, Tye, has leveled the rough spots I experienced during this effort. It seems useless to try to select words to express my sincere appreciation of Nancy's being with me.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Concluded)

LIST OF TABLES

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Concluded)

SUMMARY

Fault tree analysis is a recently developed method of reliability analysis and is generally applicable to complex, dynamic systems which include nuclear reactor systems. Fault tree analysis offers a tool by which nuclear reactor systems may be optimized in design to achieve, within the limits of engineering capabilities, the dual requirements for maximum safety and plant availability and minimum cost and complexity. The influence of the application of formal reliability analysis to all nuclear plant systems will result in higher probability of the systems functioning properly when they are called upon to operate. The greatest need today, however, is in the area of nuclear safety systems.

This dissertation provides a formal methodology, Synthetic Tree Model, for constructing fault trees for electrical systems to the point where identifiable primary component failures will directly produce the required fault events. Existing fault tree terminology is used in Synthetic Tree Model. The resultant fault trees are in a conventional format and are, consequently, immediately compatible with presently used fault tree solution techniques. Actually, they differ from a conventionally constructed fault tree in few ways. A difference is that, should any number of analysts construct fault trees independently for a given system and main failure event, using Synthetic Tree Model, they will all obtain identical fault trees. This is not a characteristic of conventional fault tree construction. This dissertation offers a model of considerable

importance since it puts forth a model that affords the opportunity to reduce the cost of and time required for a fault tree analysis as well as provides potential for a standard by which fault trees can be constructed or checked.

Synthetic Tree Model is a synthesis technique for piecing together, with proper editing, a fault tree from small segments called component failure transfer functions. The component failure transfer functions are obtained from a system-independent failure mode analysis of individual components. This piecing together is an uncomplicated process but does involve "bookkeeping" such that the appropriate editing of the component failure transfer functions can be carried out. The component failure transfer functions are a limiting factor on the resolution of the fault trees resulting from Synthetic Tree Model.

While automation of fault tree construction is possible in the framework of Synthetic Tree Model, a computer program, DRAFT, has been written to accomplish this for certain electrical systems, this automation formulates yet another distinct type of analysis. The automated construction has potential as an overall, summary-type analysis that can be routinely done in a relatively small amount of time. Automation of Synthetic Tree Model provides the fault tree analyst a valuable tool to complement his present skills while Synthetic Tree Model itself is immediately applicable to manual fault tree construction with the advantages of this manual analysis.

While Synthetic Tree Model is developed herein only for electrical systems, its implications extend to all fault tree constructions. The model is purposely left "open ended" to allow for its extension.

Synthetic Tree Model shows potential for becoming a standard for fault tree construction as it is a formal approach to fault tree construction. The technique is of a general enough nature to allow fault tree construction for systems both in the nuclear industry and elsewhere.

CHAPTER I

INTRODUCTION

1.1 Background

Reliability analysis is a relatively new subject, continuously developing and expanding. Consequently its extent as a subject is not clearly defined. On one hand, it might be thought of as simply analysis to obtain statistical estimations of numerical reliability. On the other extreme, it might be thought of as analysis encompassing the whole development program. In reality it is neither of these extremes, but rather is a set of analytical techniques generated by an attitude of anticipation of unreliability and an appreciation of the necessity of pre-planned elimination of the associated problems.[1]

A commonly accepted definition of reliability is the following: "The reliability of a system is the probability that it will perform a required function under specified conditions, without failure, for a specified period of time."[2] As this definition implies, reliability prediction is based on detailed knowledge of system configuration, knowledge of the conditions of system use, and the failure characteristics of its components.

Concepts and methods of reliability prediction have been continually developed and refined over the past decade, and now reliability prediction is an important condition in the design of many systems such as aircraft, ships and their electronic systems, missiles, and spacecraft.[3] These systems are characterized by requirements for safety, predictable mission success and minimum maintenance per operating hour--three attributes that apply strongly to nuclear reactor systems.

Reliability, like several other important reactor parameters, for example the Departure from Nucleate Boiling ratio, is not a directly measurable property of the system; it can be estimated only from other measurable parameters. Reliability analysis methodology offers a tool by which nuclear reactor systems may be optimized in design to achieve, within the limits of engineering capabilities, the dual requirements for maximum safety and plant availability and minimum cost and complexity. The influence of the application of formal reliability analysis to all nuclear plant systems will result in higher probability of the systems functioning properly when they are called upon to operate. The greatest need today, however, is in the area of nuclear safety systems.

Formal reliability methods do not evaluate a system's capability to meet the functional requirements for which it was designed. Rather, reliability prediction methods establish the relative probability of the system performing adequately for the period intended under the operating conditions specified. The capability of the system to adequately meet the design function is not a part of the reliability analysis, but rather is the design adequacy. For example, the ability of a pump to deliver a given flow rate is a measure of its design adequacy. The probability of the pump functioning at some future time is its reliability.

1.1.1 System Structure Models

In the development of relevant system structure models, the concern is not with failure rates or distribution functions; rather, it should be focused on an adequate logical description of all events that must occur to cause system failure.[4] An adequate logical description can be derived only if the functional design, physical layout, and method of operation of a system are known.

Approximations to reality can be achieved with probability models derived from reliability block diagrams.[5] Basically, block diagram models are probabilistic statements of component and part combinations necessary to achieve satisfactory operation. The sophistication or realism in block diagram models can vary greatly from simple part-count models modified to reflect redundancy to computerized programs that consider dependency, redundancy, and time sequencing in system operation by defining a system in terms of functions and components essential to the functions.[6] Block diagram models allow consideration of redundancy, are well suited to available data and system descriptions, and provide some capability to handle dependency.

The complexity of the more sophisticated block diagram models, suggests a more logical approach to the development of reactor system probabilistic models. In effect what is required is a definition of all event sequences that give rise to the failure event or events of interest. In this approach a system logic model is developed that is addressed solely to the failure of interest; for example, the maximum credible accident and how this failure might develop. Such an approach does not require assumptions about independence in redundancy. It is solely based on the physical design and functional description of a system. Fault tree analysis is the method used to develop system failure logic. In this approach an undesired event is defined. This event must be real and measurable. Subsequently the subevents necessary to cause the undesired event are developed in a graphical display by using various logic gates; for example, AND or OR gates. The appeal of fault tree analysis is that it simulates the critical aspects of system failure behavior as closely as possible without construction of the real system. Other advantages of system logic models are that they account for redundancy, repair, interdependence, and second order failures. Second order failures are failures induced by interaction of the component with the results of other component's failures. For example, if a relay fails in a valve actuation system, it is possible to observe the direct effect of the failure, as well as the effect of failures that may be induced in other parts of the system such as improper valve sequencing and false system status information. Also an unusual failure mode in one component may be examined for its effect, if it occurs, on the other similar components in the system.

1.1.2 Fault Tree Analysis

Fault tree analysis provides an all inclusive, versatile mathematical tool for analyzing complex systems.[7] Its application can include a complete plant as well as any of the systems and subsystems. Fault tree analysis provides an objective basis for analyzing system design, performing trade-off studies, analyzing common mode failures, demonstrating compliance with Atomic Energy Commission requirements, and justifying system changes or additions.

The logic of the approach makes it a visibility tool for both engineering and management. Conventional reliability analysis techniques are inductive in nature and are primarily concerned with assuring that hardware will reliably accomplish its assigned functions. The fault tree method is concerned with assuring that all critical activities are identified and eliminated or controlled.

In 1961 the concept of fault tree analysis was originated by Bell Telephone Laboratories as a technique with which to perform a safety evaluation of the Minuteman Launch Control System.[8] At the 1965 Safety Symposium, sponsored by the University of Washington and the Boeing Company, several papers were presented that expounded the virtues of fault tree analysis.[9] The presentation of these papers marked the beginning of a widespread interest in the possibility of using fault tree analysis as a reliability tool in the nuclear reactor industry. In the early 1970's great strides were made in the solution of fault trees to obtain complete reliability information about relatively complex systems.[10,11,12,13,14] The collection and evaluation of failure data is still of the utmost importance.[7,15,16,17]

Main benefits of fault tree analysis include:

1. Directing the analyst to ferret out failures in a deductive way.

2. Pointing out the aspects of the system important in respect to the failure of interest.

3. Providing a graphical aid giving system management visibility to those removed from the system design changes.

4. Providing options for qualitative or quantitative system reliability analysis.

5. Allowing the analyst to concentrate on one particular system at a time.

6. Providing the analyst with genuine insight into system behavior.

Fault tree models do have disadvantages. Probably the most outstanding is the cost of development in first time application to a system.[3] As in the development of engineering drawings for a nuclear reactor system, the cost is somewhat offset by future application of the models in accident prevention and system modifications. Another possible disadvantage is that the validity of the model is controlled by the skill and thoroughness of the analyst. This is true of all safety analysis work.

Fault tree analysis is a sophisticated form of reliability analysis and is consequently relatively expensive. The additional expense is justified by detail of the qualitative or quantitative analysis resulting from fault tree analysis. Another aspect of fault tree analysis that limits its application at this time is the relatively small number of people skilled in the techniques of fault tree analysis.[18] Even skilled personnel might develop a fault tree for a given system in different ways.[19]

The worst pitfalls that can confront one unskilled in performing fault tree analysis is over-sight and omission.[19] Significant omissions sometimes occur if the analyst jumps ahead two or more logical levels in his development of a deductive chain of factors and causes. For example, he may skip from initiation of a command to its acceptance, and neglect transmission. The tendency for this to happen is minimized if one follows the rule of listing very direct, immediate causes of any factor considered before going on to consider the next lower level of causes.

While certain single failures that can result in several component failures simultaneously, common mode failures, can be pointed out by a detailed fault tree analysis, the analyst must be alert to include other common mode failures properly in the fault tree. At any rate, the analyst should be aware that fault tree analysis does not inherently ferret out common mode failures.

1.2 Objectives

The objective of this thesis is to present a formal methodology for fault tree construction. A method formal enough to allow automated hardware-oriented fault tree construction for certain electrical systems as examples is sought, with its implications extending to fault tree construction in general, neglecting secondary failures. The methodology, called Synthetic Tree Model (STM), is to be "open ended" to allow for its extension to various types of systems and to allow increased resolution of the resultant fault trees.

The fault trees resulting from Synthetic Tree Model are to be in conventional format, use conventional symbols, and are to be constructed beginning with the main fault event of interest and proceeding to the individual component failure as is done in conventional fault tree construction. Actually, they should differ from a conventionally constructed fault tree in few ways. A main difference should be that should any number of analysts construct fault trees independently for a given system and main failure event using Synthetic Tree Model, they will all obtain identical fault trees. This is not a characteristic of conventional fault tree construction.

Being a formal methodology, Synthetic Tree Model is to offer potential as a standard for fault tree construction. The technique is to be of a general enough nature to allow fault tree construction for systems both in the nuclear industry and elsewhere.

1.3 Importance of the Study

Fault tree analysis has become of considerable importance as a tool of safety and reliability analysis in the nuclear industry. Much has been published dealing with developing techniques to quantify existing fault trees during the past decade while little has been published dealing with the construction of the fault tree itself. There is no published formal model for fault tree construction other than Synthetic Tree Model as presented herein. Other techniques for fault tree construction depend on the analyst ferreting out system logic, a technique that has the advantage of insuring that the analyst obtains a detailed knowledge of the system.[1,7,8,19] A major disadvantage of fault tree analysis has been the large amount of time required to develop the fault tree itself. This thesis offers a method of considerable importance since it presents a model that affords the opportunity to reduce the cost of and time required for a fault tree analysis as well as provides potential for a standard by which fault trees can be evaluated and checked.

1.4 Limitations

While all the objectives were obtained for Synthetic Tree Model, there are certain limitations. The method does not account for secondary failures--that is, failure related feedback between components is ignored. This is not a limitation of fault tree analysis but only of Synthetic Tree Model. The fault trees are constructed to the point where identifiable primary component failures will directly produce the fault event in question.

Synthetic Tree Model provides the basis for totally automated reliability prediction. Automated analysis should be thought of as a distinct type of analysis that could never replace conventional fault tree analysis. This automated tool could stop the system analyst from thinking. A value of the fault tree technique is that the analyst is forced to truly understand the system. Many weaknesses are typically corrected while constructing the fault tree. A value of the technique is the construction process, as well as the tree itself and resulting probability numbers. The automated analysis presented herein is a hardware oriented approach that does not include environmental and human effects that can cause failures and, therefore, is apart from an in-depth fault tree analysis.

Some systems may not lend themselves to analysis using Synthetic Tree Model since it may not be possible to determine certain necessary parameters for these systems. Indeed, there is no guarantee that a sufficient set of these parameters can be determined for systems other than the types presented in this thesis.

1.5 Method of Approach

Synthetic Tree Model is a synthesis method for constructing fault trees from small segments called component failure transfer functions. The component failure transfer functions are obtained from a system-independent analysis of every component appearing in the system for which the fault tree is to be constructed. Once the component failure transfer functions are obtained, they may be used repeatedly, without modification, for any other system in which the component appears.

The system is defined by its associated schematic diagram and by system boundary conditions. The system boundary conditions give the main failure of interest, the one for which the fault tree is to be drawn, and also define the configurations of the components that have more than one operating state in the "non-failed" system. These boundary conditions along with other boundary conditions generated during the fault tree construction itself provide a basis for editing the failure transfer functions as they are connected into the fault trees.

The component failure transfer functions, inter-correlation between boundary condition and fault events, and several other parameters are catalogued as library data and are thereby available to the analyst or computer.

Basic concepts and definitions of Synthetic Tree Model are presented in Chapter II. Conventional fault tree terminology is presented in Chapter III. Details about basic parameters of Synthetic Tree Model are provided in Chapter IV, while the synthesis and editing processes are described in Chapter V. Chapter VI provides an example demonstrating Synthetic Tree Model. A complete, automated reliability prediction is then given in Chapter VII demonstrating the role of Synthetic Tree Model. Appendix C presents a computer constructed fault tree for a reactor scram system using Synthetic Tree Model.

CHAPTER II

CONCEPTS OF SYNTHETIC TREE MODEL

Synthetic Tree Model is a formal methodology for constructing fault trees for electrical systems to the point where identifiable primary component failures will directly produce the required fault events. Synthetic Tree Model (STM) is unique in that it is formal enough to have permitted automated fault tree construction for certain electrical systems. While STM is developed herein only for electrical systems, its implications, extend to all fault tree construction.

STM is a synthesis technique for piecing together, with proper editing, a fault tree from small segments called component failure transfer functions. These component failure transfer functions are obtained from a system-independent failure mode analysis of individual components. Failure mode analysis is identifying all possible means by which a component can fail to perform its required functions. In some cases failure mode analysis has included not only the systematic identification of all the mechanisms of each mode of failure, but also assessing the probability of occurrence of these mechanisms. For STM the probability assessment can be neglected or at least deferred until a quantitative analysis is appropriate. This piecing together is an uncomplicated process but does involve "bookkeeping" such that the appropriate editing of the component failure transfer functions can be carried out. The component failure transfer functions are a limiting factor on the quality of the fault trees resulting from STM.

2.1 Basic Definitions

Primary failures are basic component failures that require no further dissection since probability data for these failures are available. These probabilistic data are inputs to the quantitative analysis using the fault tree.

A fault event is a failure situation resulting from one of the logical interactions of more than one primary failure. The most undesired fault event is at the top of the fault tree and is called the TOP event. The TOP event is the starting point of fault tree construction. There is only one TOP event in any given fault tree.

A system component is a basic system constituent for which failures are considered primary failures during fault tree construction. Consequently, the components of a given system can change depending on the TOP event being studied or the detail the analyst wishes to include in the fault tree analysis. Some components have several operating states, none of which are necessarily failed states. Relay contacts can be open or closed for example. The description of these states is called the component configuration.

Fault tree construction is the logical development of the TOP event. As the construction proceeds each fault event is also developed until primary failures are reached. The development of any event results in a branch of the fault tree. The event being developed is called the base event of the branch. The branch is complete only when all events in the branch are developed to the level of primary failures. Every event in a branch is in the domain of the base event. In addition, if the base event is an input to an AND gate, every event in the branch is in the domain of every input to that AND gate.

A fault tree gate is composed of two parts, (1) the Boolean logic symbol that relates the inputs of the gate to its output event and (2) the output event description. However, a gate is equivalent to another gate if, and only if, the logic symbol, the output event description, and another parameter, the "effective boundary conditions" associated with the output event, are identical. These effective boundary conditions will be considered in detail later.

There are two parts to the event description, (1) the incident identification and (2) the entity identification. The incident identification defines, as briefly as possible, the fault without indicating any hardware involved. The entity identification specifies the component or sub-system involved. These two parts are both required to describe the fault event.

2.2 An Introduction to the Failure Transfer Functions

The key to STM is associating a complete set of failure transfer functions with each system component. A component failure transfer function describes one mode of failure for a component and is a fundamental property of the component. The failure transfer functions are then independent of the system being analyzed.

It is convenient at this time to define a device. A device is a piece of hardware whose modes of failure are somewhat different from the modes of failure of all other devices. Systems are composed of devices.

All components are devices. All components that are of identical design are the same device. Components that are not of identical design may, however, be the same device. If the complete set of failure transfer functions for a component is identical to the complete set of transfer functions of another component, they are then the same device. The number of devices in a given system is then always less than or equal to the number of components. Usually there are many more components than devices. Failure transfer functions can be thought of as a minute sub-fault tree. However, their appearance in the final system fault tree may be altered considerably.

Once a failure transfer function for a device has been determined it may be catalogued as library data. This library data can then be updated to reflect as much detail as desired. Otherwise, it is a constant property of the device.

A failure transfer function may consist of as many as six parts, (1) an output event, (2) an output logic gate, (3) internal events, (4) internal logic gates, (5) input events, and (6) a discriminator. All of these parts can be determined from the fundamental workings of the component isolated from any system environment.

The <u>output event</u> is the mode of failure being considered. For a particular component, there is only one failure transfer function for a given output event. The output event is different from a fault event in that it requires no entity identification.

The <u>output gate</u> designates the logic with which the failure transfer function is coupled into the fault tree with other appropriate failure transfer functions having the same output event. There is one output gate for each transfer function.

<u>Internal events</u> are fault events requiring further logical development within the failure transfer function. There is always enough information available from the component isolated from any system environment to allow further development of these events. <u>Internal gates</u> designate the logical development of the internal events as required by the output and input events.

<u>Input events</u> can be either primary events or undeveloped fault events. Input events represent the furthest development of the output event possible by considering the isolated component.

The <u>discriminator</u> is a flag designating which failure transfer functions may coexist in the final fault tree. The discriminator can be determined from the component since it indicates which output events can actually coexist within the same component. There is no more than one discriminator assigned to each failure transfer function.

A concept of the failure transfer function is illustrated in Figure 1. In a conventional sense, only the internal events and gates would be considered a transfer function; however, for the purposes of STM the conglomerate of all the parameters shown in Figure 1 is designated as the failure transfer function.

An example of a failure transfer function for electrical contacts causing no current in a circuit is shown in Figure 2. An equally valid representation of this failure transfer function is shown as a Boolean logic diagram in Figure 3. The implication of the discriminator is that when developing the input event, system input to the contacts causes the

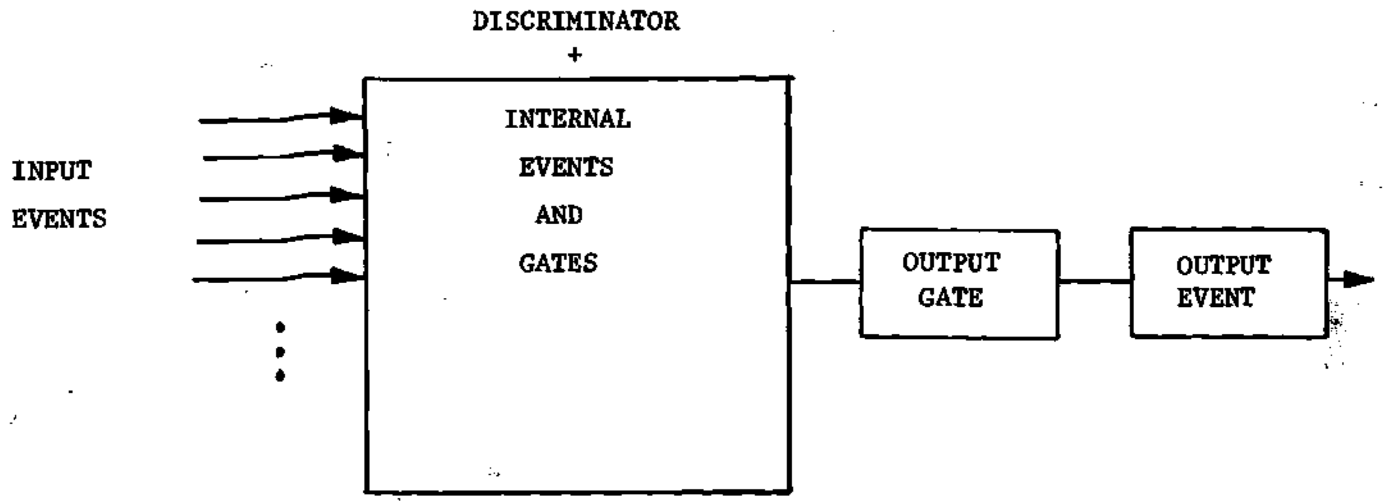
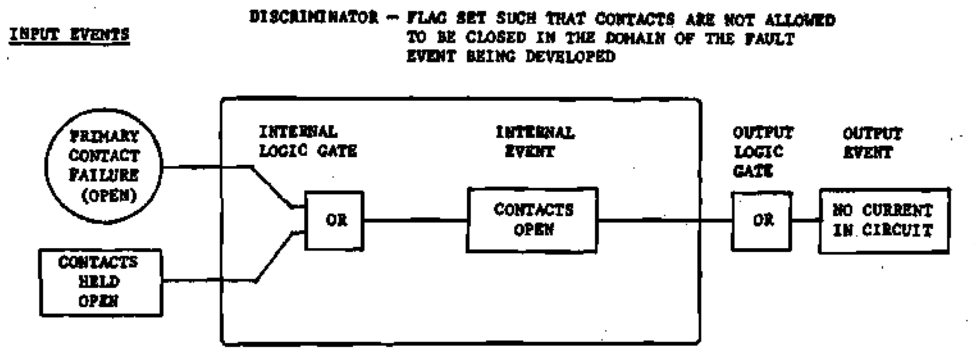
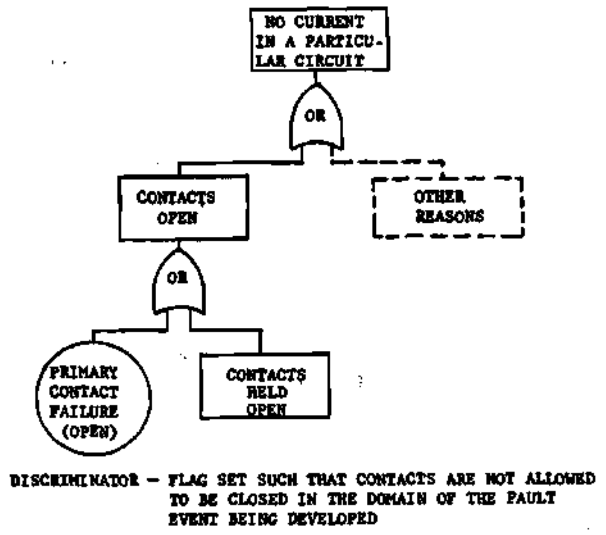


Figure 1. Concept of the Failure Transfer Function of Synthetic Tree Model



INPUT EVENTS
DISCRIMINATOR — FLAG SET SUCH THAT CONTACTS ARE NOT ALLOWED TO BE CLOSED IN THE DOMAIN OF THE FAULT EVENT BEING DEVELOPED
PRIMARY CONTACT FAILURE (OPEN)
CONTACTS HELD OPEN
INTERNAL LOGIC GATE
INTERNAL EVENT
OR
CONTACTS OPEN
OUTPUT LOGIC GATE
OUTPUT EVENT
OR
NO CURRENT IN CIRCUIT

Figure 2. Example of a Failure Transfer Function for Electrical Contacts



NO CURRENT IN A PARTICULAR CIRCUIT
OR
CONTACTS OPEN
OTHER REASONS
OR
PRIMARY CONTACT FAILURE (OPEN)
CONTACTS HELD OPEN
DISCRIMINATOR — FLAG SET SUCH THAT CONTACTS ARE NOT ALLOWED TO BE CLOSED IN THE DOMAIN OF THE FAULT EVENT BEING DEVELOPED

Figure 3. Boolean Logic Representation of the Failure Transfer Functions Shown in Figure 2

contacts to be open. Any event that is excluded by the output event, no current in the circuit, is considered to be not-allowed. In short, this means in this case that the contacts cannot be open and closed at the same time.

In reflection, important characteristics of failure transfer functions are:

1. The component failure transfer functions are independent of the system being analyzed.

2. A complete set of failure transfer functions for a device may be used to represent many system components.

3. Failure transfer functions may be catalogued and used as library data.

4. There are as many failure transfer functions for a component as there are modes of failure for that component.

5. When a failure transfer function is used to develop an event, the use of certain other failure transfer functions can be excluded from the domain of that event.

2.3 Ordered Fault Events

Fault events that are used only as TOP events are First Order Fault Events. This development may be catalogued for frequently used First Order Fault Events or provided as input to STM for each individual fault tree constructed.

Fault events that state a condition of the system that extends beyond any single component are Second Order Fault Events. The entity identification then refers to a particular sub-system, or, more specifically, to a variable that will later be defined as the "component coalition." Component failure transfer functions are always used as the first step in developing the Second Order Fault Events. Examples of Second Order Fault Events are "current too long in a particular circuit" and "no current in a particular circuit."

Fault events that cause a component to "behave failed" because part of the system itself, not simply another individual component, is causing that component to behave failed are Third Order Fault Events. An example of a Third Order Fault Event is "no current to a particular light bulb." Second Order Fault Events are always used as the first step in developing Third Order Fault Events.

Fault events that result in component A behaving failed because another component has direct input to component A, are Fourth Order Fault Events. An example of a Fourth Order Fault Event is "relay contacts held open." Component failure transfer functions are always used to develop Fourth Order Fault Events.

2.4 Boundary Conditions

2.4.1 System Boundary Conditions

Without System Boundary Conditions it would not be possible to construct a fault tree. The boundary conditions in conjunction with the system schematic define the situation for which the fault tree is to be constructed. The System Boundary Conditions must be determined before any fault tree construction begins.

A most important system boundary condition is the TOP event. For any given system, a multitude of possibilities for TOP events exist. The selection of the "correct" TOP event is sometimes a difficult task. There are, however, no limitations on the event chosen as the TOP event.

The system initial configuration is described by additional System Boundary Conditions. This configuration must represent the system in the unfailed state. Consequently these System Boundary Conditions depend on the TOP event. Initial Conditions are then System Boundary Conditions that define the operating condition of the system, i.e. all component configurations, for which the TOP event is applicable.

System Boundary Conditions also include any fault event declared to exist or to be not-allowed for the duration of the fault tree construction. These events are called Existing System Boundary Conditions or Not-allowed System Boundary Conditions. An Existing System Boundary Condition is treated as certain to occur while a Not-allowed System Boundary Condition is treated as an event with no possibility of occurring. Neither Existing nor Not-allowed System Boundary Conditions ever appear as events in the final system fault tree.

2.4.2 Event Boundary Conditions

Event Boundary Conditions are boundary conditions associated with fault events in a fault tree and are implied by System Boundary Conditions or fundamental principles of set theory. A fault event is defined only when both the event description and the corresponding Event Boundary Conditions are known. A most important corollary is that a fault event is equivalent to another fault event if, and only if, their event descriptions and their associated Event Boundary Conditions are, in effect, identical.

Event Boundary Conditions are fault events or failure transfer functions that are considered as not-allowed or existing. All Not-allowed or Existing System Boundary Conditions are Event Boundary Conditions for every fault event in the fault tree. All other Event Boundary Conditions are generated by fault events as they appear in the fault tree. Once an Event Boundary Condition has been generated it is a boundary condition for every fault event that is in the domain of the fault event that generated the boundary condition. Fault events outside the domain of this base event are in no way affected by the boundary condition.

2.4.3 Effective Boundary Conditions

Effective Boundary Conditions are Event Boundary Conditions of a gate that actually affect the development of the gate. An event with an arbitrary number of Event Boundary Conditions may have no Effective Boundary Conditions. In practice, Effective Boundary Conditions are the only boundary conditions of any significance. Unfortunately, it is not possible to predict which Event Boundary Conditions are Effective Boundary Conditions; an event must be developed before its Effective Boundary Conditions are known. However, an observation that proves to be helpful in STM is that if the Event Boundary Conditions of two gates are identical the Effective Boundary Conditions must also be identical.

2.5 Class of Third Order Fault Events

Recall that Third Order Fault Events require development using Second Order Fault Events. Several Second Order Fault Events may be required as input to a single gate whose event is a Third Order Fault Event.

The logic symbol used for this gate is dependent on the Third Order Fault Event being developed and is independent of the system being analyzed. Each incident identification for Third Order Fault Events is assigned to a Class. The entity identification does not affect the Class of the Third Order Fault Event. Class I indicates a Third Order Fault Event that requires an OR gate while Class II requires an AND gate.

2.6 Category of Second Order Fault Events

Recall that Second Order Fault Events are developed using failure transfer functions of the components. However, it is possible for a Second Order Fault Event to appear during fault tree construction for which no transfer functions are available as input. This occurs because no appropriate component can fail in a manner so as to cause (or transmit) the Second Order Fault Event or Event Boundary Conditions can censor all, otherwise appropriate, failure transfer functions.

A Second Order Fault Event is in Category I if it is considered not-allowed if no failure transfer functions are available for its development. Category II indicates a Second Order Fault Event that is considered existing if no failure transfer functions are available.

CHAPTER III

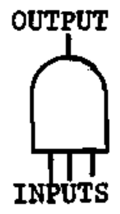
FAULT TREE TERMINOLOGY

3.1 Fault Tree Symbols

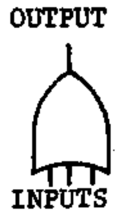
Fault tree symbols fall, basically, into two categories: logic symbols and event symbols. Logic symbols are shown in Figure 4 while event symbols are shown in Figure 5.[7,8,20]

The logic symbols, or logic gates, are used to interconnect the events that could cause the specified main event, or TOP event. The logic gates that are most frequently used to develop fault trees are the basic AND and OR Boolean expressions. The AND gate provides an output event only if all input events are presented simultaneously. The OR gate provides an output event if one or more of the input events are present. The Boolean algebra associated with these two logic gates is presented in greater detail in the next section.

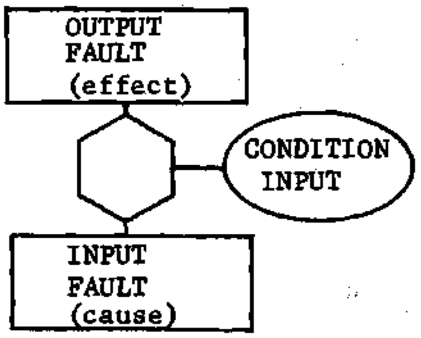
The more frequently used event symbols are the rectangle, circle and diamond. The rectangle represents a fault event resulting from the combination of more basic faults acting through logic gates. They may indeed be thought of as part of their associated logic gate. The circle designates a basic system component failure or fault input that is mutually independent from all other events designated by circles and diamonds. The diamond symbol describes fault inputs that are considered basic in a given fault tree. However, the event described is not basic in the



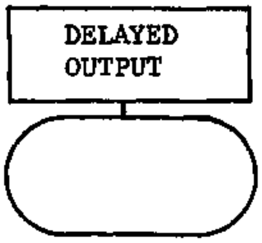
AND Gate
Coexistence of all inputs required to produce output.



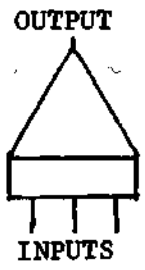
OR Gate
Output will exist if at least one input is present.



INHIBIT Gate
Input produces output directly when conditional input is satisfied.



DELAY Gate
Output occurs after specified delay time has elapsed.



MATRIX Gate
Output is related to one or more unspecified combinations of undeveloped inputs.

Figure 4. Fault Tree Logic Symbols

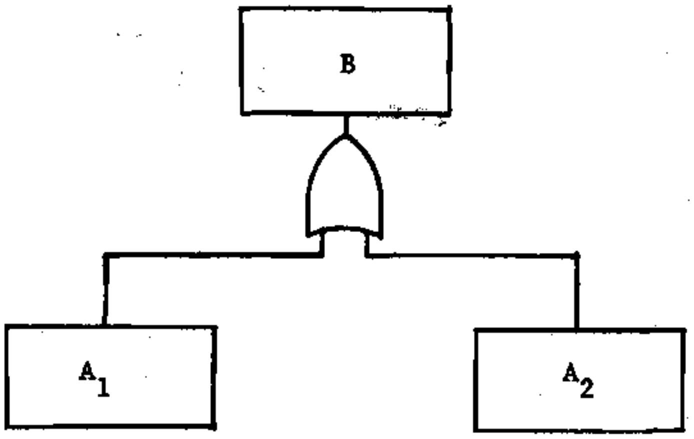
sense that laboratory data is applicable. Rather, the fault tree is simply not developed further, either because the event is of insufficient consequence or the necessary information is unavailable. Nevertheless, in order to obtain a solution for a fault tree, both circles and diamonds must represent events for which reliability information is input to the fault tree. For the study presented herein, events that appear as circles or diamonds are referred to as primary events.

The triangles shown in Figure 5 are not strictly event symbols although they have traditionally been classified as such. The triangle indicates a transfer from one part of the fault tree to another. A line from the side of the triangle (transfer out triangle) denotes an event transfer out from the associated logic gate. A line from the apex of the triangle denotes an event transfer into the associated logic gate from the transfer out triangle with the same identification number.

The other logic gates and events symbols are shown in Figure 4 and Figure 5 and are explained in those figures.

<u>3.2 The OR Gate</u>[21]

The fault tree symbol is an OR gate and represents the union of the events attached to the gate. Any one or more of the events input to the gate must occur in order for the event above the gate to occur. The OR gate is equivalent to the Boolean symbol $\bigcup$. For example, the OR gate with two input events, as shown below, is equivalent to the Boolean expression, $B = A_1 \bigcup A_2$.

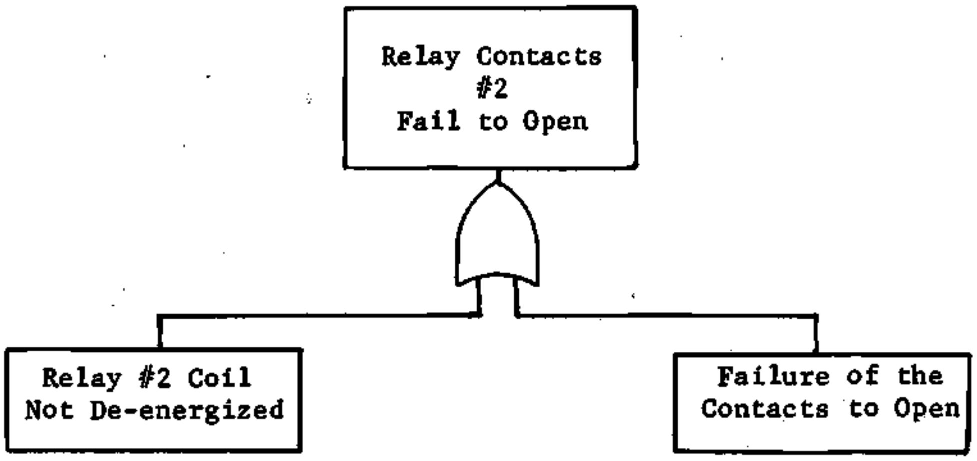


$$\updownarrow$$

$$B = A_1 \bigcup A_2$$

In the above illustration, the symbol " $\updownarrow$ " is to be interpreted as "in equivalent to." Either of the events A_1 or A_2, or both, must occur in order for B to occur.

Shown below is a realistic example of an OR gate for a fault condition of a set of normally open contacts.


Relay Contacts #2 Fail to Open
Relay #2 Coil Not De-energized
Failure of the Contacts to Open

This OR gate is equivalent to the Boolean expression

$$\begin{array}{ccccc} \boxed{\begin{array}{c}\text{Relay Contacts}\\ \text{\#2}\\ \text{Fail to Open}\end{array}} & \equiv & \boxed{\begin{array}{c}\text{Relay \#2 Coil}\\ \text{Not De-energized}\end{array}} & \cup & \boxed{\begin{array}{c}\text{Failure of the}\\ \text{Contacts to Open}\end{array}} \\ \text{Event } B & & \text{Event } A_1 & & \text{Event } A_2 \ . \end{array}$$

An "OR" gate is merely a re-expression of the event above the gate (B) in terms of the more elementary input events (A_1,A_2). The event above the gate encompasses all of these more elementary events; if any one or more of these elementary events occurs, then B occurs. This "re-expression" interpretation is quite important since it characterizes an OR gate and differentiates it from an AND gate. Whenever an event can be broken into more elementary events, then an OR gate is immediately drawn. The input events to an OR gate do not cause the event above the gate, they simply are the event above the gate "separated" into more detail.

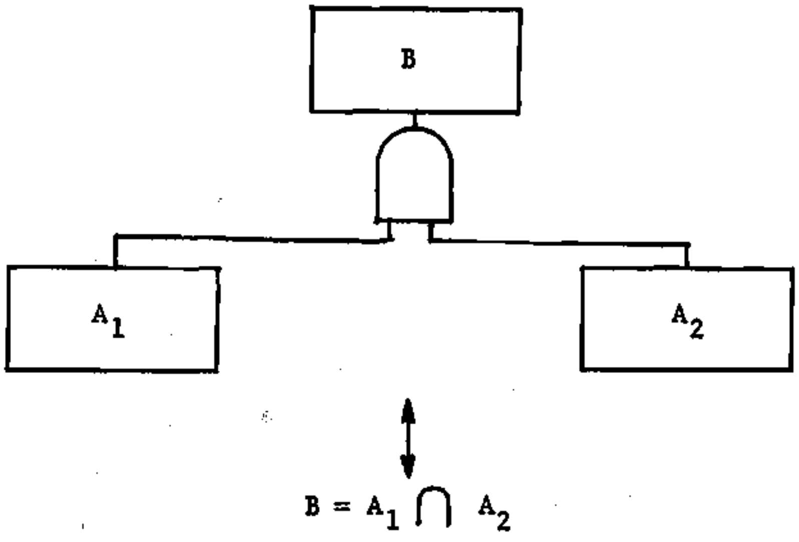
If any one or more of the more particular events $A_1 \ldots A_n$, assuming a case where n events are attached to B by an OR gate, occurs, then the more general event B occurs.

$$A_i \rightarrow B; \ i = 1, \ldots n,$$

where the symbol "$\rightarrow$" is to be interpreted as "implies."

3.3 The AND Gate[21]

The fault tree symbol ⌓ is an AND gate and represents the intersection of the events attached to the gate. The AND gate is equivalent to the Boolean symbol $\cap$. All of the events input to the AND gate must occur in order for the event above the gate to occur. For two events attached to the AND gate, the equivalent Boolean expression is $B = A_1 \cap A_2$.



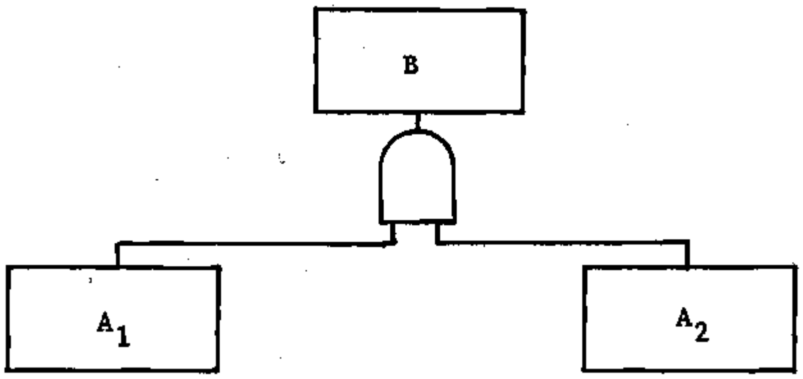
The event is only caused, or happens, if every one of the input events occurs. The cause relationship is what differentiates an AND gate from an OR gate. If the event above the gate occurs when any one of the input events occurs, then the gate is an OR gate and the event is merely a restatement of the input events. If the event above the gate occurs only when combinations of more elementary events occur, then the gate is an AND gate and each input is a cause of the event above the gate. (In set theory terminology, for an OR gate, each input event is a subset of the event above the gate while for an AND gate each input is not a subset of the event above the gate.)

For n events attached to the AND gate, the equivalent Boolean expression is

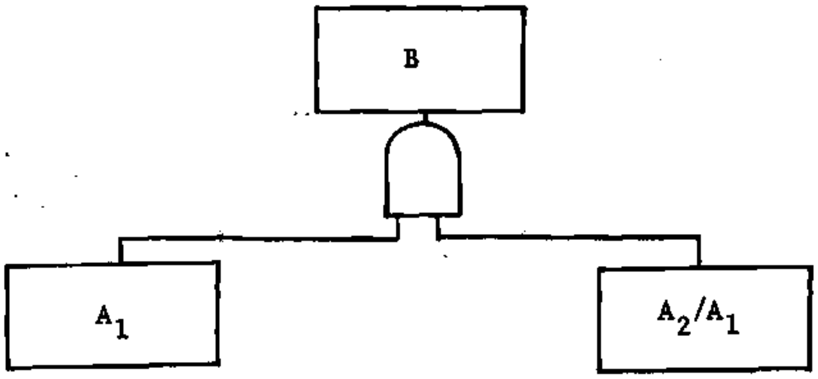
$$B = A_1 \cap A_2 \ldots \cap A_n$$

The event B is caused by A_1 and A_2 and A_3...and A_n all occurring simultaneously.

In general, the events attached to the AND gate are not interpreted to be independent, but instead are interpreted as occurring when the events to its left have already occurred. For example, in the two event illustration of the AND gate,

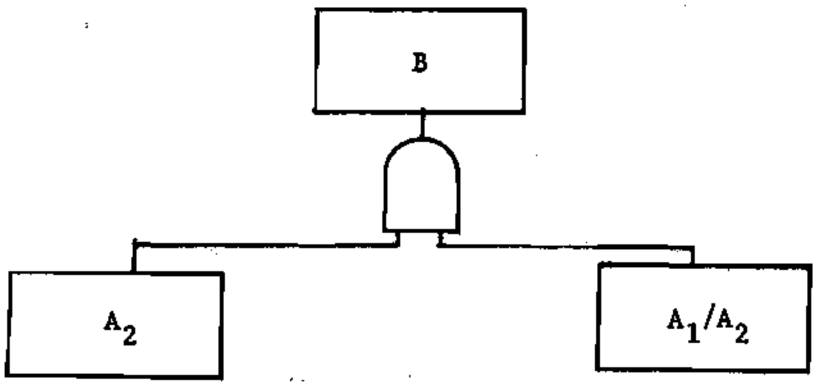


means



where A_2/A_1 is the event A_2 given that A_1 has already occurred. A_1 is a failure occurring with no other failures already existing in the system; it is the "first" failure. A_2/A_1 is the failure A_2 occurring with the failure A_1 already existing in the system; A_2 is thus the "second" failure. If A_1 is traced to more basic failures or causes, then the system will be examined with no previous failure already having occurred. If A_2 is traced to more basic failures or causes the system already having the failure A_1 will be examined for more basic events. The system examined for A_1 is thus of a different nature from the system examined for A_2. For example, the failure A_1 may have caused the system to undergo a different operation. The failure A_2 will then be traced to the more primary failures with the system in this different operation.

The AND gate may also be represented with A_2 as the "first" event, i.e., the event leftmost in the gate.

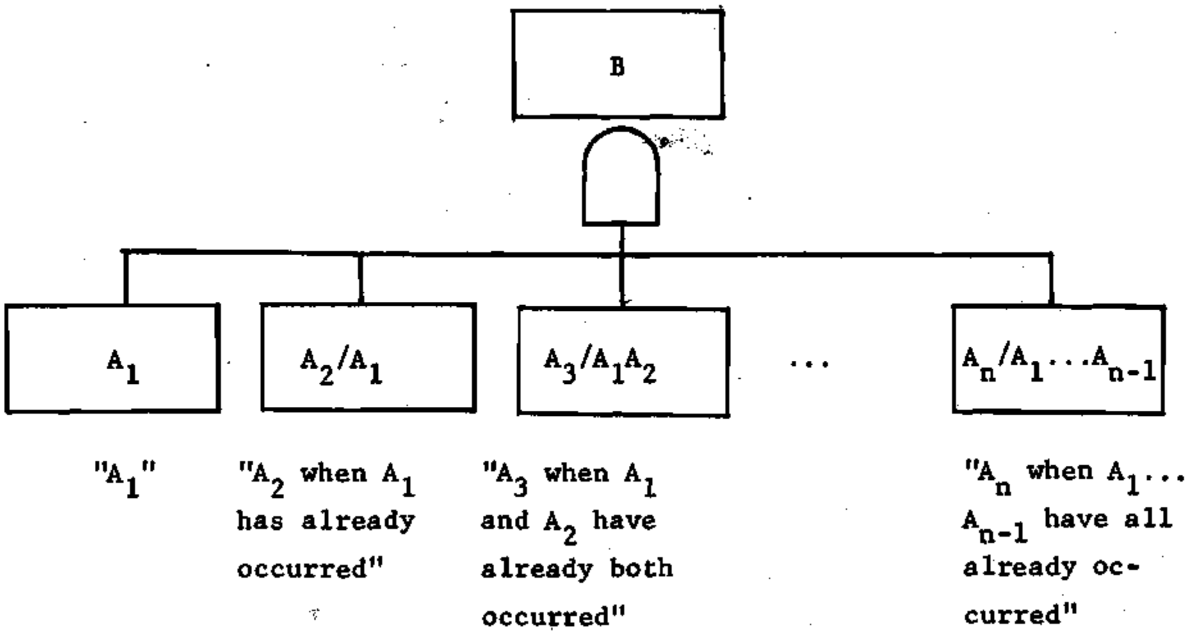


For those instances in which order of cause is not significant (which applies to most situations), this representation is entirely equivalent with the preceding one (where A_1 was the "first" event). For A_2 represented first, as shown above, A_1 will now be traced with the failure A_2 already existing. Where order of cause is not important, which failure is represented first is completely arbitrary and the particular sequence chosen is that which most simplifies the analysis.

When the order of occurrence of the failures is pertinent, the first failure having to occur is represented leftmost on the gate and then the second failure necessary is the fight failure on the gate. The second failure will still be analyzed with the first failure already existing, as before.

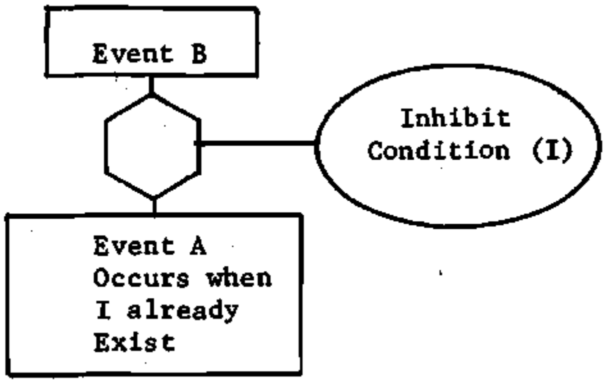
One may say then that when order is unimportant, an ordering of the causes for further analysis is arbitrarily picked. When order is important the particular ordering necessary is dictated from the failure being investigated. In either case, the second failure is investigated with the first failure already existing.

For the case of n events $A_1, \ldots A_n$ attached to the AND gate refer to the figure on the next page. The event B is caused by all of the events A_1, $A_2, \ldots A_n$ having to occur. For A_2/A_1 we are examining the occurrence of A_2 under the condition that A_1 has occurred. For $A_n/A_1 \ldots A_{n-1}$ we are examining the occurrence of A_n under the condition that $A_1, \ldots A_{n-1}$ have already all previously occurred. In general, we are examining the possible causes or re-expressions of the event A_k with the system in the state such that $A_1, A_2, \ldots A_{k-1}$ are already existing. We are thus looking at the system with succeedingly more failures already existing.


B
A_1
A_2/A_1
A_3/A_1A_2
...
$A_n/A_1 \ldots A_{n-1}$
"A_1"
"A_2 when A_1 has already occurred"
"A_3 when A_1 and A_2 have already both occurred"
"A_n when $A_1 \ldots A_{n-1}$ have all already occurred"

Only when the event (A_k) is independent of the events to its left ($A_{k-1}, \ldots A_1$) can we neglect these left events as having already occurred. That is, if $A_1, A_2, \ldots A_{k-1}$ have not changed in any way the nature of the system, then we can neglect all of these failures already existing and consider A_k as occurring in a system free of other failures.

The INHIBIT condition representation is shown below.



In Boolean representation

$$B = I \cap (A/I) = I \cap A$$

An INHIBIT gate therefore is equivalent to an AND gate. The INHIBIT condition represents an event, a condition, or an environment that must exist along with A in order to cause B. The INHIBIT condition is only different from an event in that the causes of the inhibit condition are of no concern and are not further traced on the tree. The event A is traced to its causes with the condition that I is not existing.

3.4 Cut Sets[22]

The events of a fault tree can be Boolean manipulated in order to obtain the minimal cut sets of a fault tree. A minimal cut set is the smallest set of primary events which must all occur in order for the TOP event to occur. A primary event is a circle or diamond on the tree of is an INHIBIT condition. A primary event is thus a component failure, environmental effect, administrative error, etc. The primary events represent the resolution of the fault tree. The minimal cut sets represent the modes by which the TOP event can occur. For example, the minimal cut set A_1A_2 means that both the primary events A_1 and A_2 must occur in order for the TOP event to occur. A_1 and A_2 is a mode by which the TOP event occurs. If either A_1 or A_2 does not occur, then the TOP event does not occur by this mode. The set of events A_1A_2C, where C is another primary event, is not a minimal cut set since C is redundant and is not necessary for the occurrence of the TOP event; C can either occur or not occur and as long as A_1 and A_2 both occur, then the TOP event will occur.

The minimal cut sets are significant since they depict which failures must first be corrected in order for the TOP failure to be corrected.* The minimal cut sets often give the "weakest links" in the system. The primary failures (i.e., primary events) in the one event minimal cut sets usually should first be corrected. With these single failures corrected, the failures in the two event critical paths should then usually be corrected, and so forth. A single failure analysis is an investigation, or fault tree drawn, in order to obtain only the one primary event minimal cut set, (single failures) of the TOP event. For a single failure analysis, the fault tree ends whenever an AND gate is reached, that does not have deeper common causes (which effectively transform an AND gate to an OR gate).

The basic Boolean operations are summarized below.[23]

1. Distributive Laws

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$
$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

2. DeMorgan's Theorems

$$(A \cup B)' = A' \cap B'$$
$$(A \cap B)' = A' \cup B'$$

3. Laws of Absorption

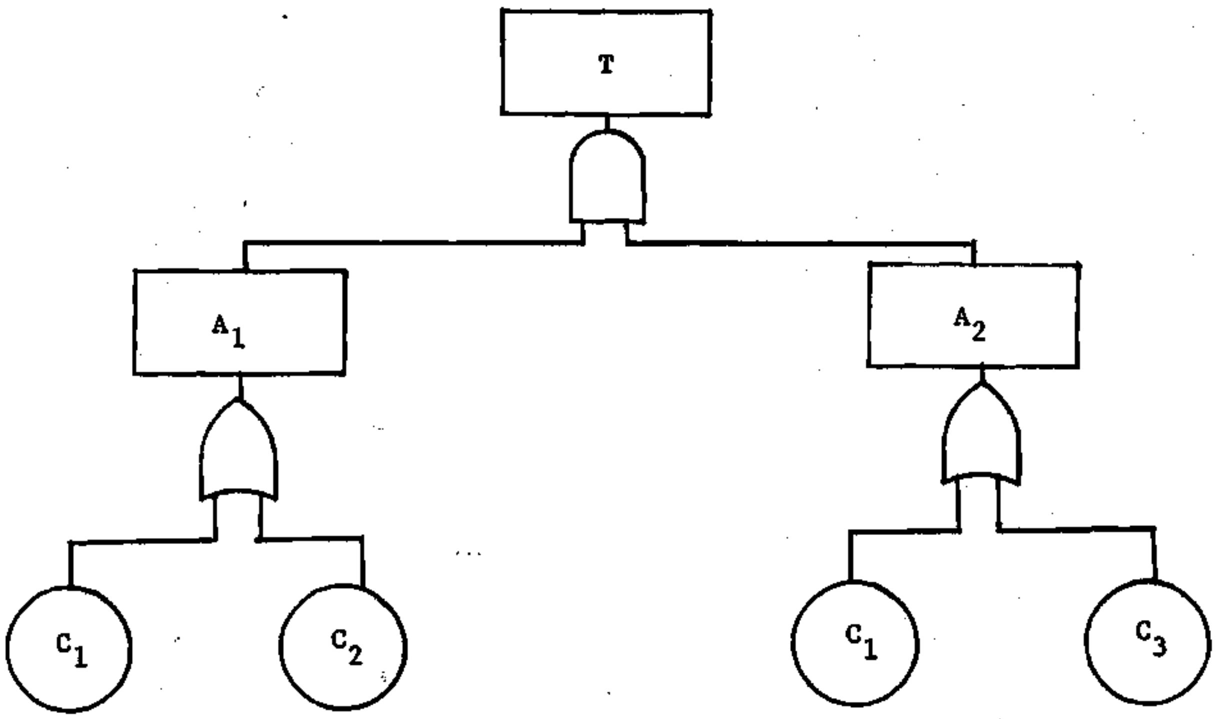
$$A \cup B = A \; ; \; B \subset A$$
$$A \cap B = B \; ; \; B \subset A$$

The symbol " ' " denotes the complement of an event; A' thus means "the event A not occurring." The symbol "$\subset$" denotes "is a subset of";

*A failure being "corrected" means its removal, a lowering of its probability, or a coupling of the failure with another failure (adding a redundancy).[21]

B $\subset$ A denotes that B is a subset of A. B is a subset of A if and only if the occurrence of B implies the occurrence of A; if B occurs then A "automatically" occurs. For the manipulation of events on a fault tree we will be principally concerned with the Distributive Laws and the Laws of Absorption.

Consider the following simple fault tree.



T is the TOP event and A_1 and A_2 are certain fault events and C_1, C_2, C_1, and C_3 are the primary failures. As stated previously, for Boolean manipulation a unique symbol is assigned to each unique event. Thus T corresponds to the word description of the TOP failure, etc. In terms of Boolean algebra;

$T = A_1 \bigcap A_2$

$A_1 = C_1 \bigcup C_2$

$A_2 = C_1 \bigcup C_{\cancel{2}3}$

By inspection of the fault tree shown above the sets of occurrence of the events C_1C_1, C_1C_3, C_2C_1, and C_2C_3 will cause the TOP event to occur. These sets are called the Boolean Indicated Cuts Sets of the fault tree.[24]

In Boolean algebra, these three equations are equivalent to the fault tree. For a Boolean representation of a fault tree, every gate on the fault tree must have its equivalent Boolean equation. Here a gate corresponds to the event (rectangle) immediately above the gate. Thus, for the above fault tree, there is a Boolean equation for T, for A_1, and for A_2.

Substituting the expressions for A_1 and A_2 into the expression for T,

$$T = (C_1 \bigcup C_2) \bigcap (C_1 \bigcup C_3) .$$

Using the distributive law,

$$(C_1 \bigcup C_2) \bigcap (C_1 \bigcup C_3) = C_1 \bigcup (C_2 \bigcap C_3) .$$

The minimal cut sets of T are then the one primary event minimal cut set C_1 and the two event minimal cut set C_2C_3. T occurs if C_1 occurs or if both C_2 and C_3 occur.

The object of Boolean manipulation is to obtain the TOP event T in the form

$$T = M_1 \bigcup M_2 \bigcup M_3 \ldots \bigcup M_n$$

where the M_i's are events consisting of intersections of primary events (circles or diamonds),

$$M_i = A_{i1} \bigcap A_{i2} \bigcap \ldots A_{im}$$

where A_{i1}, etc. are primary events and where M_i is not a subset of another M_j(i.e., the primary events of a certain M_j are not all contained in another M_j). If this form for T is obtained, then the M_i's are the minimal cut set of the fault tree.

CHAPTER IV

CHARACTERISTIC FACTORS OF SYNTHETIC TREE MODEL

There are five characteristic, discrete factors that must be determined before Synthetic Tree Model can be implemented. It is felt that these five factors limit universal application of Synthetic Tree Model. A sufficient number of values are determined for each of these variables to allow Synthetic Tree Model to be applicable to certain electrical systems.

These characteristic, discrete factors are the:

(1) Component failure transfer functions.

(2) Component Coalition scheme.

(3) Class of the Third Order Fault Events.

(4) Category of the Second Order Fault Events.

(5) Inter-correlation between the fault events and the boundary conditions.

With the exception of the Component Coalition scheme each of these factors can be catalogued as library data and corrected and updated as necessary, since they are of an "open ended" nature. It is possible to ascertain the Component Coalition scheme in complete, exact, closed form for electrical systems.

4.1 Particulars of the Failure Transfer Functions

The quality of the fault tree constructed by Synthetic Tree Model depends on the quality of the component failure transfer functions.

Recall that it is not necessary to catalogue a failure transfer function for each component, only for each device. Each component that is in effect the same device, use the same failure transfer function. It is envisioned, that as Synthetic Tree Model becomes more sophisticated the component failure transfer functions will become more detailed.

A consideration during the development of all failure transfer functions is that only failure modes are to be considered. Often a Second Order Fault Event can be "allowed to happen" by a component that can never fail in a manner to cause that Second Order Fault Event. Such possibilities are never included as failure transfer functions. For example, a fuse may well allow the fault "current too long in its circuit" to happen but a fuse never fails such that it causes "current too long in its circuit."

For the failure transfer functions given as examples herein, it is assumed the system was constructed perfectly with no components installed that do not meet specifications. This is not a limitation of Synthetic Tree Model but rather just a convention adopted here for convenience.

Several examples of failure transfer functions are given in Appendix A in the Boolean logic notation. Some care must be used when drawing conclusions from the failure transfer functions presented here since they are of a very simple nature. The failure transfer functions for some devices can be very complex and somewhat more difficult to determine. Even in conventional fault tree construction, however, the failure transfer functions must, in effect, be determined.

4.1.1 How to Obtain the Failure Transfer Functions

The majority of a failure transfer function for a device is determined by conventional failure mode analysis. Recall that failure mode analysis has been defined as a method of identifying all possible means by which a device can fail to perform its required function. This failure mode analysis then immediately provides the failure transfer function output event--that is the Second Order and Fourth Order Fault Events that the component failure transfer functions can be used to develop.

The output logic gate is determined by recognizing the logical relation the device failure has to the Second or Fourth Order Fault Event that the device failure transfer function can be used to develop. That is to say, the output logic gate depicts the way the event being developed is transferred through the component. If the component failure alone can cause the fault event being developed then the output logic gate is OR. If, however, the component failure is required in addition to the fault event being developed, then the output logic gate is AND.

Internal events give further information about the failure mode and should be used liberally. Their appearance in a final fault tree gives local insight about the transfer function input events. The internal gates depict the logical relationship between the internal events and the input events. The input events are primary failures or Third Order Fault Events or Fourth Order Fault Events.

After this failure mode analysis information has been supplied to the set of failure transfer functions, a discriminator is set. The discriminator is a flag set to indicate which failure modes (failure transfer functions) can coexist in the component and is generally determinable from the output event description. Failure transfer functions used to develop Fourth Order Fault Events do not need a discriminator since Fourth Order Fault Events arise only from failure transfer functions that do have discriminators.

4.2 Examples of Determining Failure Transfer Functions

Examples of determining the failure transfer functions reflect the same general procedure as a Failure Mode Analysis of a system component. The examples of the fuse and contacts provide general examples of this determination.

4.2.1 The Failure Transfer Function for a Fuse

To determine the failure transfer function for a fuse, consider the ways a fuse can fail by considering its design. A fuse is an over-current protective device, with a circuit-opening fusible member directly heated and destroyed by the passage of over current. A fuse, by not performing as intended, can fail by transmitting an overload. Also since the fusible member of a fuse transmits current under normal operation, the fuse can randomly fail so as to cause "no current." There are then two failure transfer functions for a fuse, one with the output event "overload" and another with the output event "no current."

The failure transfer function for the output event "overload" will be determined first. Since the fuse alone can not cause an overload--it can only transmit an existing overload--the output gate is AND. There is only one input event to the transfer function, primary failure of the fuse to open circuit when subjected to an overload. There are no internal events or gates.

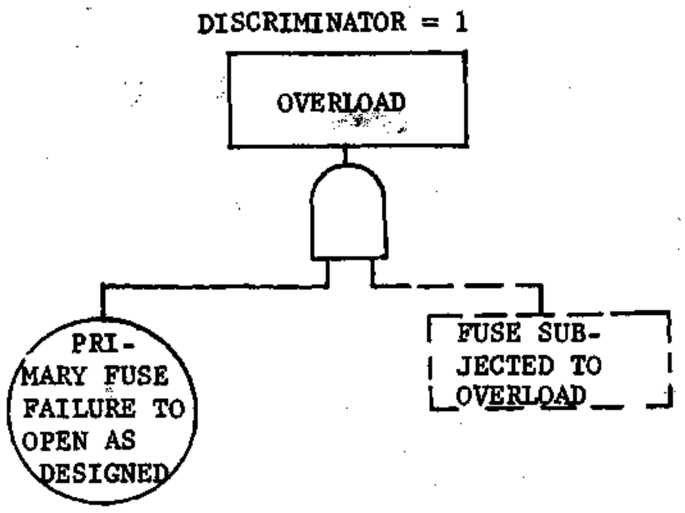
For the output event "no current," the output gate is OR since the fuse alone can cause "no current." Again there is only one input event--primary failure of the fuse (fuse opens).

The failure transfer functions for a fuse are given in Figure 6 in the Boolean logic tree notation. The discriminators are set different so as to denote that the two failure transfer function output events are not allowed to coexist, that is a fuse cannot be failed open and closed at the same time. Note that other input events could have been provided such as "oversize fuse installed," but recall that all components are assumed to meet specifications. This does, however, demonstrate that the failure transfer functions are not complete, only sufficient.

4.2.2 Failure Transfer Functions for Contacts

Contacts are a device that represent several different type components--switches, relay contacts, circuit breaker contacts, etc. Since contacts can be designed to be normally open or closed in a system, either condition must be considered as a possible failed state.

By their designed intent contact failures can result in "no-current," "current," "no current too long," or "current too long." Since contacts are a low resistance device, their shorting to cause an overload is not credible. The Second Order Fault Event "current too long" and "no current too long" included because, during later analysis, a timer relay coil will be considered. The event "no current" can be caused by the contacts alone; therefore, its associated output logic gate is OR. On the other hand the output logic gates for the other two fault events are AND. Each


DISCRIMINATOR = 1
OVERLOAD
PRIMARY FUSE FAILURE TO OPEN AS DESIGNED
FUSE SUBJECTED TO OVERLOAD

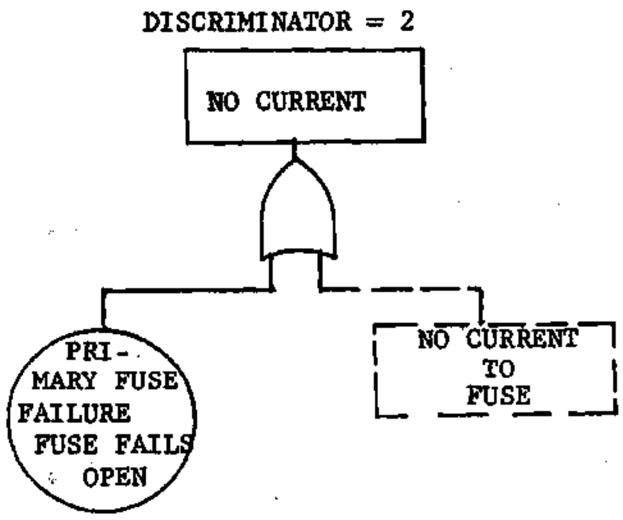


Figure 6. Failure Transfer Functions for Fuse in Boolean Logic Notation

has an interval event, "contacts open," "contacts closed," "contacts closed too long," and "contacts open too long," respectively. The input events to the four failure transfer functions include both primary events and Fourth Order Fault Events as shown in Figure 7.

4.3 Particulars of the Component Coalition Scheme

While the Component Coalition Scheme is determined for electrical systems in exact, closed form, it perhaps affords the most interesting challenge encountered in an effort to extend Synthetic Tree Model to include other types of systems. This results from its being a scheme rather than determinable on the basis of fault events.

The appropriate scheme is, however, certainly not difficult to apply to electrical systems. The component coalition is determinable from the system schematic diagram alone and is independent of the particular components involved with the exception of the power supply.

4.3.1 How to Determine the Component Coalition

Components in an electrical system receive system "feedback" through electrical wiring. This connection forms the premise of determining the component coalition. One electrical flow path from the power supply through a component constitutes a means that component can receive (or not receive) power.

If the system components are separated by a minimum number of nodes and no node appears more than once in such an electrical flow path, that electrical flow path is a series circuit path and indicates a component coalition. One component can appear in several series circuit paths. If a component receives no current, it must receive no current from each and

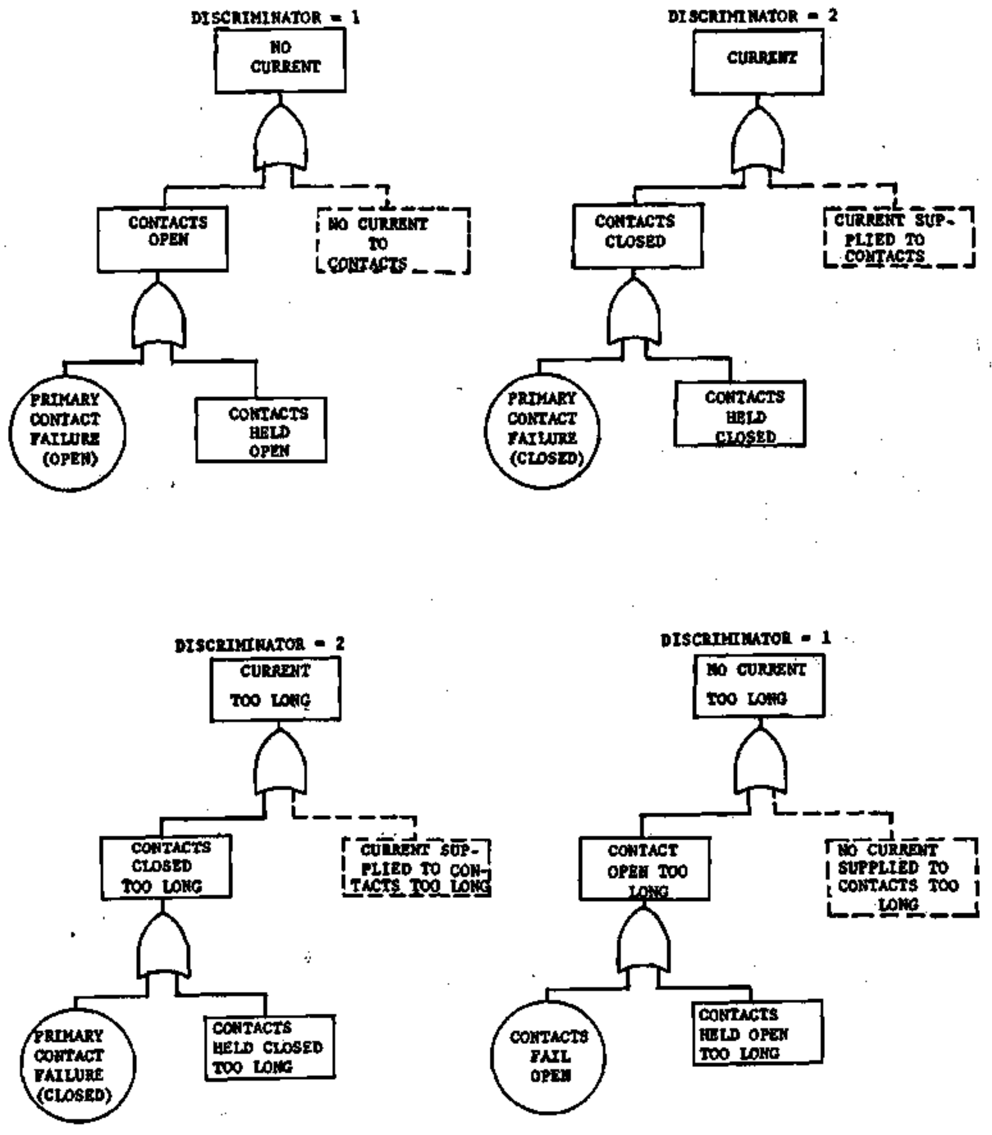


Figure 7. Failure Transfer Functions for Contacts in Boolean Logic Notation

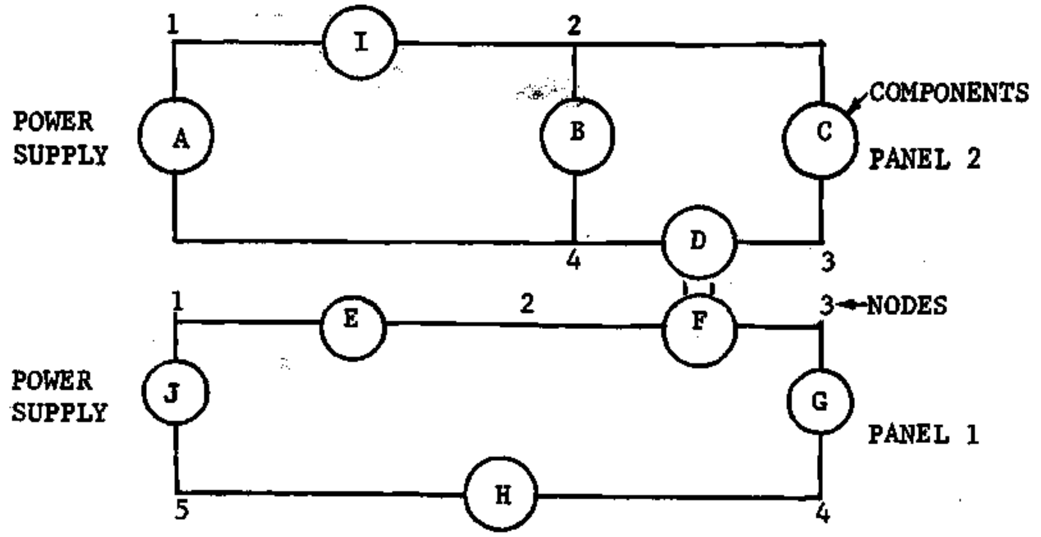
every series circuit path that contains that component. On the other hand, a component can be supplied current by any one of the series circuit paths containing that component.

A component coalition for an electrical system is a collection of components in a series circuit path. If wiring is to be included in the component coalition, it must be designated as a component. There are as many component coalitions as there are series circuit paths.

In many systems, there may be several sub-systems each with its own power supply. In Synthetic Tree Model, each such sub-system is called a panel. No panel can have wiring common to another panel; however, more than one power supply is allowed per panel. Each component coalition can contain components from only one panel. There can be interfacing between panels by mechanical coupling between one or more components. For example, a relay coil may provide mechanical input to relay contacts in another panel. There may also be such mechanical interplay within a given panel. This mechanical interplay in no way affects the component coalition. Panels with no electrical wiring then may contain only one component. The component coalition for such a panel is simply the one component.

4.3.2 Example of Determining the Component Coalition

As an example of determining the component coalition, consider the following sample system.



Component D is the relay coil for contacts F.

To determine the component coalition, determine as many current flow paths through the power supply as possible for each panel such that no node occurs in the flow path more than once.

For panel 1, there is one component coalition, components J, E, F, G, and H. For panel 2 there are two component coalitions, components A, I, and B and components A, I, C, and D.

4.4 Particulars of the Category of the Second Order Fault Event

Before determining the Category of the Second Order Fault Event it is necessary to determine the incident identification of all the Second Order Fault Events themselves. Recall that all output events of transfer functions are either the incident identification of a Second Order Fault Event or the incident identification of a Fourth Order Fault Event. All Fourth Order Fault Event incident identifications indicate mechanical input to one or more components while the Second Order Fault Event incident identifications indicate a condition of a component coalition. The

Second Order Fault Event incident identifications are then determinable from the output events of the failure transfer functions.

4.4.1 How to Determine the Category of Second Order Fault Events

All Second Order Fault Events with identical incident identifications are of the same Category. That is to say the Category is independent of the component coalition involved. Recall Second Order Fault Events of Category I are considered "not-allowed" if there are no failure transfer functions available to develop that Second Order Fault Event from the associated component coalition, while Category II indicates a Second Order Fault Event is "existing" if no failure transfer functions are available. A basic premise of electrical system design is that conductors are used to transmit a specified amount of current. Therefore, any fault event that denotes an event contrary to this premise and there is nothing in the component coalition to cause that fault event is not-allowed or Category I. If, however, the fault event indicates a condition compatible to this premise and there is nothing in the component coalition to cause that fault event the fault event is existing or Category II. All Second Order Fault Events with the same incident identification are of the same Category.

4.4.2 Examples of Categories of Second Order Fault Events

In accordance with the above stipulations all Second Order Fault Events with the incident identification, "no current," "no current too long," and "overload" are of Category I while "current" and "current too long" are of Category II.

4.5 Particulars of the Class of Third Order Fault Events

Third Order Fault Event identification can be determined directly from the Second Order Fault Event incident identification. Recall that while the entity identification of a Third Order Fault Event indicates a particular component coalition, the Third Order Fault Event has the same incident identification as a Second Order Fault Event. There is then a one-to-one correspondence between Third Order Fault Events and Second Order Fault Event incident identifications. Also recall that Second Order Fault Events are used exclusively to develop Third Order Fault Events.

4.5.1 How to Determine the Class of the Third Order Fault Event

The Class of the Third Order Fault Events are determined in a straightforward, logical manner. If failure, in a manner indicated by the Third Order Fault Event incident identification, of every component coalition involving a given component is required to produce the Third Order Fault Event in that given component, the Third Order Fault Event is of Class I. If, on the other hand, failure in the manner indicated by the Third Order Fault Event incident identification, of any of the component coalitions involving a given component will produce the Third Order Fault Event in that given component, the Third Order Fault Event is of Class II. Every Third Order Fault Event with the same incident identification is of the same Class.

4.5.2 Examples of the Class of Third Order Fault Events

Third Order Fault Events with the incident identification "no current" are of Class I while Third Order Fault Events with the incident identification "current," and "overload" are of Class II.

4.6 Particulars of the Inter-Correlation Between Fault Events and Boundary Conditions

Only First and Second Order Fault Events are involved in the Inter-correlation between Fault Events and Boundary Conditions. Recall that higher order Fault Events always involve a particular component. The purpose of the Boundary Condition is to edit the Component Transfer Functions. The effect of higher order Fault Event Boundary Condition generation must be taken into account during the development of the failure transfer functions themselves.

4.6.1 How to Determine the Inter-Correlation Between First Order Fault Events and Boundary Conditions

A First Order Fault Event always generates Not-allowed Event Boundary Conditions. Since the First Order Fault Event is the base event for the entire fault tree, the failure transfer functions for the component indicated by the entity identification of the First Order Fault Event are never allowed to appear in the fault trees and hence are Not-allowed Boundary Conditions.

4.6.2 How to Determine the Inter-Correlation Between Second Order Fault Events and Boundary Conditions

Second Order Fault Events can generate Not-allowed Event Boundary Conditions or Existing Event Boundary Conditions. Not-allowed Event Boundary Conditions are generated because of events being excluded by the Second Order Fault Event, while Existing Event Boundary Conditions result from implications of the system initial conditions.

<u>4.6.2.1 Type 1 Second Order Fault Event Boundary Conditions</u>. The occurrence of a Second Order Fault Event, A, generates other Second Order

Fault Events that are excluded by the Second Order Fault Event, A, as Not-allowed Event Boundary Conditions.

<u>4.6.2.2 Type 2 Second Order Fault Event Boundary Conditions</u>. The occurrence of Second Order Fault Event can imply that certain component failures are not-allowed during the development of that Second Order Fault Event because the Second Order Fault Events exclude the component failure.

<u>4.6.2.3 Type 3 Second Order Fault Event Boundary Conditions</u>. Once a Transfer Function, A, is used to develop a Second Order Fault Event, Transfer Functions with discriminators different from A are Not-allowed Event Boundary Conditions.

<u>4.6.2.4 Type 4 Second Order Fault Event Boundary Conditions</u>. Second Order Fault Events can also generate Existing Event Boundary Conditions. These Event Boundary Conditions are always implied by the system initial conditions. The system unfailed state is defined by the initial conditions; therefore, if a fault event indicates a component, when functioning as designed, is in this "unfailed," initial state, this "unfailed" state is an Existing Event Boundary Condition. That is to say, if the system, as indicated by a fault event, "forces" a component into a state corresponding to its "initial," "unfailed" state, then this "unfailed" state does, indeed, subsequently exist and is not considered a "fault" event at all.

The need for this type of Event Boundary Condition arises because a given component configuration in one system may represent a failed state, while in another system this same configuration may indicate the unfailed state. It is then not surprising that under certain conditions in certain systems, certain events that generally can be fault events are not and, indeed, are considered existing.

Recall that Second Order Fault Events "state a condition of the system." If this system state does, indeed, generate a component configuration identical to the component configuration in the "unfailed" system then this component configuration is subsequently an existing event boundary condition. Second Order Fault Events are the only fault events capable of generating Existing Event Boundary Conditions.

4.6.3 Examples of the Inter-Correlation Between Fault Events and Boundary Conditions

First Order Fault Event Boundary Condition--If the TOP Event is "a certain light bulb failing to produce light," later in the fault tree the bulb is not-allowed to fail again by short circuiting or open circuiting.

Type 1 Second Order Fault Event Boundary Condition--If the fault event "no current in a given series circuit path" is being developed, the Second Order Fault Events indicating current in that same series current path are not-allowed.

Type 2 Second Order Fault Event Boundary Condition--If the Second Order Fault Event "current in a given series circuit path" is being developed, none of the components in the coalition indicated by that series circuit path are allowed to fail so as to cause no current in any series current path.

Type 3 Second Order Fault Event Boundary Condition--If during the development of a base event, a Second Order Fault Event calls for the fuse failure transfer function indicating that fuse is causing an overload, then during the continued development of that same base event that fuse is not-allowed to fail to open.

Type 4 Second Order Fault Event Boundary Condition--If a Second Order Fault Event indicates current is being supplied to a relay coil so that its contacts would be closed and the initial condition includes those contacts being closed, then those contacts being closed is an Existing Event Boundary Condition for the Second Order Fault Event.

4.6.4 Comments

The fully developed Inter-correlation between Second Order Fault Events and Boundary Conditions is given for a number of Second Order Fault Events in Appendix A. The inter-correlation provided there is felt to be sufficient for all system Fault Trees that use only those Second Order Fault Events. The inter-correlation is of an "open-ended" structure and, consequently, additional inter-correlation information can be added as Synthetic Tree Model is extended.

CHAPTER V

FAULT TREE DEVELOPMENT STRATEGY OF SYNTHETIC TREE MODEL

Synthetic Tree Model is similar to conventional fault tree construction techniques in that it starts with the TOP event and the development then proceeds through intermediate gates to the primary failures of the components. The construction is then complete when the terminal events of every branch are primary fault events.

If the TOP event is a fault event of a higher order than a First Order Fault Event, then defining the TOP event is sufficient to trigger the mechanism of STM to complete the fault tree. If, however, the TOP event is a First Order Fault Event, the development of that First Order Fault Event must be supplied such that all input events to that First Order Fault Event are higher order events or primary events. That is to say, the analyst must provide enough of the fault tree such that STM can get started.

5.1 Catalogued First Order Fault Events

Certain First Order Fault Events are often required repeatedly during the application of fault tree analysis. The required development of such First Order Fault Events can be conveniently catalogued in a library in a manner similar to the failure transfer functions. In fact, an identical format can be used with the exception that the discriminator is not required.

There is then a one to one correspondence between:

(1) the First Order Fault Event and the failure transfer function output event,

(2) the first logic gate under the First Order Fault Event and the output logic gate,

(3) gates used to develop the First Order Fault Event and the failure transfer function internal events,

(4) the required final level events--higher order than the First Order Fault Event--and the failure transfer function input events.

An example of the development of such a First Order Fault Event is given in Appendix A.

5.2 How to Use Boundary Conditions

During the construction of the fault tree by STM, before any event is placed in the fault tree or any failure transfer function is used, it is checked to see if it is a boundary condition. The procedure used to deal with events that are boundary conditions depends on what kind of boundary condition it is and on the logic gate to which the event is attached.

5.2.1 What to Do if a Fault Event Is a Not-allowed Boundary Condition

If a fault event or failure transfer function about to appear in the fault tree is a Not-allowed Boundary Condition for the gate to which the fault event or failure transfer function is about to be attached and that gate is an OR gate, the fault event is simply not used in the fault tree. This simple removal is possible since the gate from which the fault event or failure transfer function is removed can still provide an output event trigger since it is an OR gate.

If, however, the fault event or failure transfer function is a Not-allowed Boundary Condition and it is about to be attached to an AND gate, the entire AND gate is removed from the fault tree as are all the immediately preceding AND gates up to the next OR gate. This is because, if one of the inputs to an AND gate does not occur, there can be no occurrence of the output event, hence no failure through that AND gate. The AND gate is then unnecessary for the fault tree and is removed since it too is not-allowed. The same argument can be extended to all immediately preceding AND gates.

5.2.2 What to Do if a Fault Event Is an Existing Boundary Condition

If a fault event or failure transfer function about to appear in the fault tree is an Existing Boundary Condition for the gate to which the fault event or failure transfer function is about to be attached and that gate is an AND gate, the fault event is simply not used in the fault tree. This simple removal is possible because an input to an AND gate being "true" makes no contribution to the fault tree.

If, however, the fault event or failure transfer function about to appear is an Existing Boundary Condition and it is attached to an OR gate, the entire OR gate is removed from the fault tree as are all the immediately preceding OR gates up to the next AND gate. An existing event then triggers through OR gates.

5.3 How to Develop a Second Order Fault Event

A Second Order Fault Event is always developed using failure transfer functions. Only failure transfer functions of the components in the component coalition indicated by the entity identification of the Second

Order Fault Event are considered. The failure transfer functions with AND output logic gates, if any, are then added, in any order, to the fault tree to develop the Second Order Fault Event. Finally, the failure transfer functions with OR output logic gates, if any are added, in any order, to the development of the Second Order Fault Event.

The output event of the failure transfer function does not appear in the fault tree but rather is only a flag to indicate which failure transfer functions to use to develop a given Second Order Fault Event. If there are several failure transfer functions with AND output logic gates, each of these failure transfer functions is connected to only one AND gate in the fault tree. If there are, in addition, failure transfer functions with OR output logic gates, one OR gate is used as an input to the previous AND gate and these failure transfer functions are then connected to this OR gate.

If there are, however, no failure transfer functions available with AND output logic gates, but there are failure transfer functions with OR output logic gates, then the Second Order Fault Event being developed has an OR logic gate only.

An example of the development of Second Order Fault Events appears in Chapter VI.

5.4 How to Develop Third Order Fault Events

Third Order Fault Events are developed using Second Order Fault Events. Every component coalition containing the component indicated by the entity identification of the Third Order Fault Event is determined. There is a one to one correspondence between those component coalitions and the Second Order Fault Events used to develop the Third Order Fault Event. The input events to the Third Order Fault Event are Second Order Fault Events with the same incident identification as the Third Order Fault Event and with their entity identification representing one of these component coalitions.

The logic gate used to connect the Second Order Fault Events to the Third Order Fault Events is determined directly by the Class of the Third Order Fault Event.

Examples of development of Third Order Fault Events are given in Chapter VI.

5.5 How to Develop Fourth Order Fault Events

Fourth Order Fault Events are developed using failure transfer functions. Recall that STM allows for direct interplay between components. If a component, A, receives input from one or more components, this interplay correlation must be provided as input to STM. From this correlation and incident identification of the Fourth Order Fault Event, the exact failure transfer functions used to develop the Fourth Order Fault Event, are determined.

The failure transfer function output event is a flag used to correlate the failure transfer function to the particular Fourth Order Fault Event incident identification and does not appear in the fault tree. The output gate of the failure transfer function indicates the logical relationship of the transfer functions, if more than one, used to develop the Fourth Order Fault Event. If no interplay correlation is provided for the component indicated by the entity identification of the Fourth Order

Fault Event, that Fourth Order Fault Event is not developed and appears in the fault tree as a diamond symbol.

5.6 Final Editing Concerns

The use of STM as presented thus far results in fault trees that may require further editing. While the fault tree is perhaps "academically correct" without this editing, the editing puts the fault tree in conventional format that is convenient for the analyst.

5.6.1 Transfers

Transfers within a fault tree should be approached with extreme caution. A transfer within a fault tree cannot be used simply because two events have identical incident and entity identifications. The Effective Boundary Conditions must also be the same. There are two ways to be sure this criterion is met. If the sets of Event Boundary Conditions for each otherwise identical fault event are also identical, a transfer can be made. If these Event Boundary Conditions of the events in question are not identical, each event must be developed to completion. If, on the other hand, the branches of these base events are identical, the desired transfer can be made by leaving one such branch in the fault tree while other identical branches are removed with the appropriate transfer indicated. If this latter approach is used, the transfer will abbreviate the fault tree itself but not its construction time.

5.6.2 Loops

A loop exists in a fault tree constructed by STM if event A occurs and in the domain of A an event occurs that has the same incident identification, entity identification, and Effective Boundary Conditions as A.

There are two approaches for dealing with this situation:

(1) Indicate the appropriate transfer back to the first occurrence of A from the latter occurrence of A and do not show the development of the latter occurrence of A in the fault tree.

(2) Eliminate the loop situation from the fault tree.

The first approach perhaps provides a fault tree that gives the greatest system management visibility. It is not possible, however, to solve the fault tree containing a loop with a computer if the fault tree contains such a transfer due to limitations of all present methods of locating the minimal cut sets.

The second approach is easily implemented within the framework of STM since, if a loop occurs, the second occurrence of the event is simply treated as a Not-allowed Boundary Condition. (See sections 5.2.1 and 5.2.2.)

5.6.3 Only One Input to a Gate

If there is finally only one input to any gate, the logic gate type (AND or OR) is immaterial and consequently is not stated.

CHAPTER VI

MANUAL FAULT TREE CONSTRUCTION USING SYNTHETIC TREE MODEL

The system chosen for manual fault tree construction using STM is the classical fault tree example system that was first presented by Haasl[8] in 1965. While this example does not demonstrate every facet of STM, it is doubtful any one relatively simply system could, and it does present the basic synthesis procedure very well.

This system is represented by the schematic shown in Figure 8. When the switch is closed, power is applied to the timer coil. This closes the timer contacts and applies power to the relay coil, which in turn closes the relay contacts. Power is then supplied through the fuse to the motor. When the switch is opened, the reverse procedure applies.

The fuse and the timer are safeguards; if the motor fails shorted while the relay contacts are closed, then the fuse opens and shuts off the power, and if the switch fails to open again after some time (which is preset) then the timer will open its contacts and remove power from the motor.

The overheating of the wire is an undesirable event in this circuit and it can be prevented if the safeguards operate.

It is now possible to list the system boundary conditions as follows:

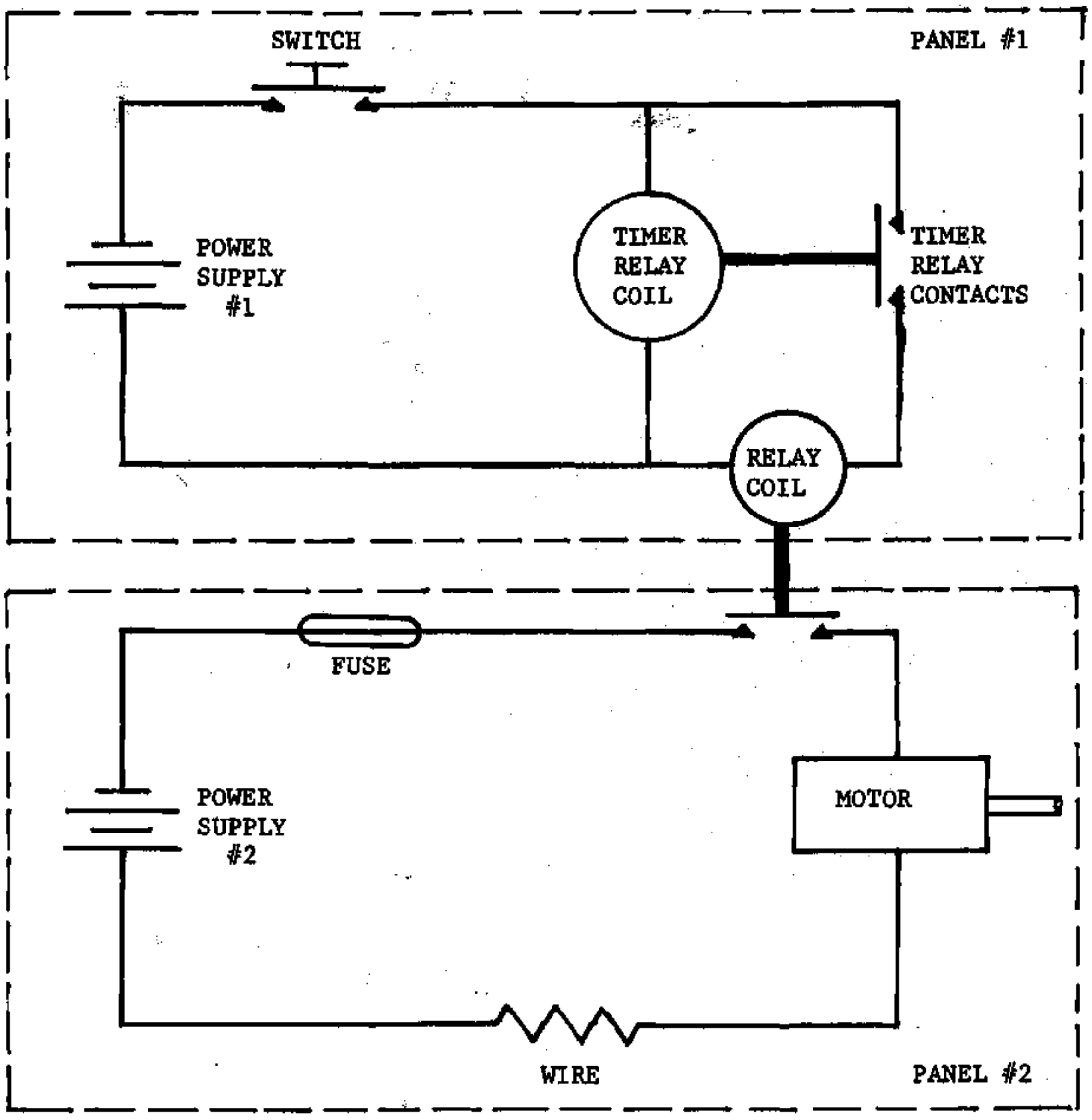


Figure 8. Schematic for Manual Fault Tree Construction Using Synthetic Tree Model

TOP Event	Overheated wire
Initial Conditions	Relay contacts closed Timer contacts closed Switch closed
Existing System Boundary Conditions	None
Not-allowed System Boundary Conditions	None

The TOP event is a First Order Fault Event and therefore must be developed to the level of higher order fault events and/or primary events. This development is shown in Figure 9 and also appears in Appendix A as a catalogued First Order Fault Event.

It is necessary now to determine the component coalitions. There are two series circuit paths in Panel 1, hence two component coalitions, while in Panel 2 there is only one component coalition.

Component Coalition	Components
1	Switch Timer Relay Coil Power Supply #1
2	Switch Power Supply #1 Timer Contacts Relay Coils
3	Power Supply #2 Fuse Wire Motor Relay Contacts

The failure transfer functions for these components are given in Appendix A and will be used as found there. The First Order Fault Event generates the following Not-allowed Event Boundary Conditions for gate #1 (see Section 4.1.1).

(1) No Current Because of Wire

(2) Overload Because of Wire

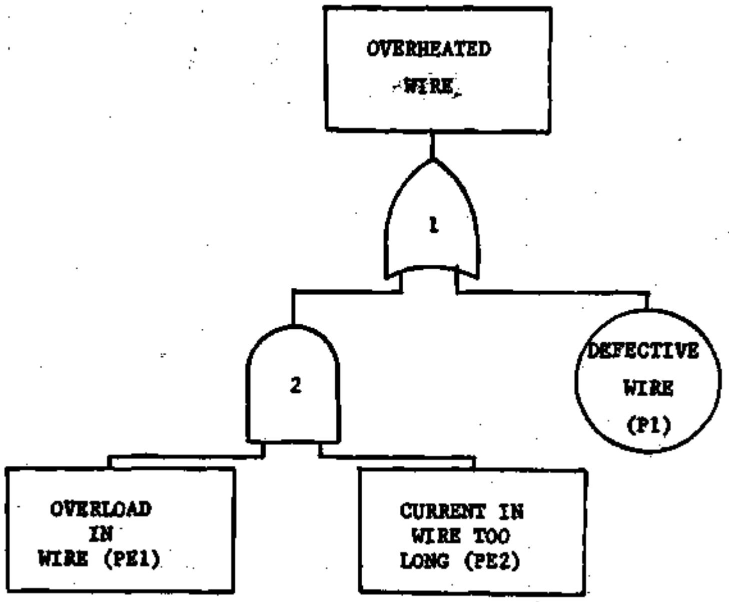


Figure 9. Sufficient Development of the First Order Fault Event

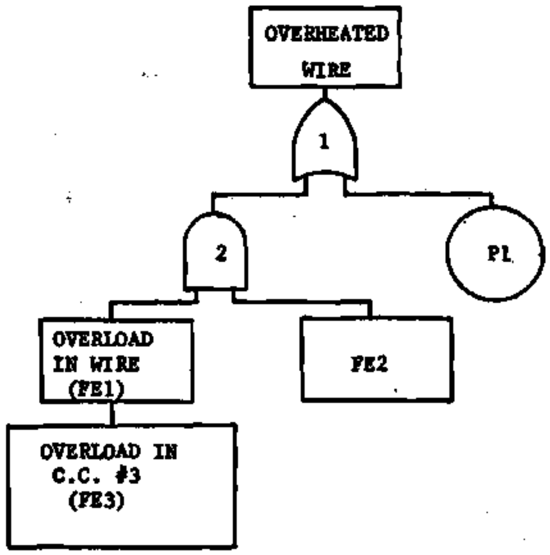


Figure 10. Development Stage Number 1 of Sample Fault Tree Construction

These boundary conditions will then be Not-allowed Event Boundary Conditions for every gate in the fault tree since every gate is in the domain of the TOP event (see Section 3.4.2).

It is now possible to proceed with the construction of the fault tree. A particular event must be chosen to be developed.

6.1 Development of the Third Order Fault Event, Overload in Wire

The only component coalition #3 contains the wire; therefore, the class of the Third Order Fault Event is immaterial (see Section 5.6.3). The fault tree development at this stage then is shown in Figure 10.

6.2 Development of the Second Order Fault Event, Overload in Component Coalition #3

This Second Order Fault Event can be developed using the component failure transfer function of the components in component coalition #3 (see Section 5.3). The failure transfer function for the fuse must be coupled into the tree first since its output logic gate is AND (see Figure 25). The failure transfer functions of the power supply, motor, and wire can be inserted into the fault tree next since both of these components can fail to cause the overload (see Figures 2, 27, and 32). This development is shown in Figure 11.

However, recall that "overload because of wire" is a Not-allowed Event Boundary Condition; therefore, this failure transfer function must be dealt with as described in Section 5.2.1. The result is shown in Figure 12.

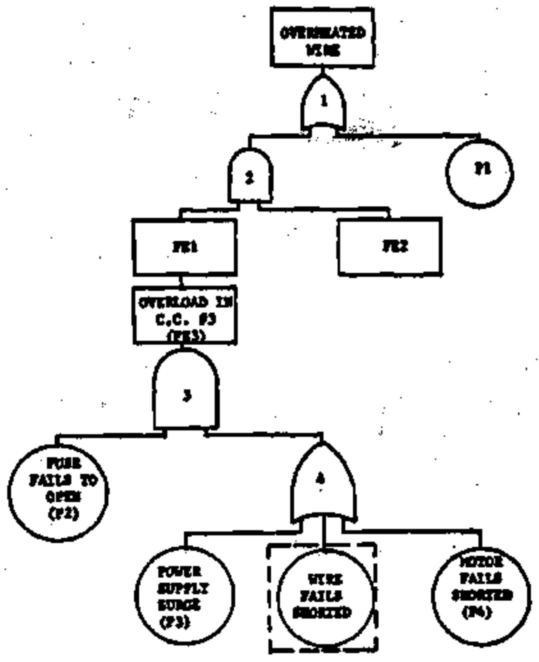


Figure 11. Development Stage Number 2 of Sample Fault Tree Construction

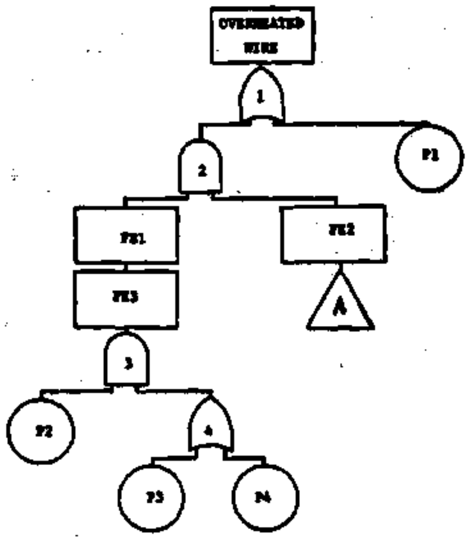


Figure 12. Development Stage Number 3 of Sample Fault Tree Construction

The additional Event Boundary Conditions generated by the Second Order Fault Event are the Not-allowed Event Boundary Conditions:

(1) No current in component coalition #3 (see Section 4.6.2.1)

(2) No current because of relay contacts (see Section 4.6.2.2)

(3) No current because of fuse (see Section 4.6.2.2 or Section 4.6.2.3)

(4) No current because of power supply (see Section 4.6.2.2 or Section 4.6.2.3)

(5) No current because of motor (see Section 4.6.2.2 or Section 4.6.2.3).

These boundary conditions are not Effective Boundary Conditions, however, since the domain of the Second Order Fault Event generating the boundary conditions contains only primary fault events. This then completes this branch of the fault tree.

6.3 Development of the Third Order Fault Event, Current in Wire Too Long

Again only component coalition #3 contains the wire; therefore, the Third Order Fault Event development is as shown in Figure 13. If the wire had appeared in more than one component coalition, each would be connected into the tree with an OR since the class of the Third Order Fault Event is II.

6.4 Development of the Second Order Fault Event, Current in Component Coalition #3 Too Long

The only component in component coalition #3 with a failure transfer function indicating an output event of current too long is the relay contacts. This failure transfer function is coupled into the fault tree as shown in Figure 14. The Second Order Fault Event generates new boundary conditions for its domain. These boundary conditions are coincidentally identical to those for the Second Order Fault Event in Section 6.2.

6.5 Development of the Fourth Order Fault Event, Relay Contacts Held Closed Too Long

From the schematic it is seen that the relay coil supplies direct input to the relay contacts; therefore, the failure transfer function of this relay coil indicating holding the contacts closed is used to develop the Fourth Order Fault Event (see Section 5.5). The results are shown in Figure 15.

6.6 Development of the Third Order Fault Event, Current to Relay Coil Too Long

The relay coil appears in the component coalition #2 only. The development is shown in Figure 16 to be a single Second Order Fault Event, current in the component coalition #2 too long (see Section 5.4).

6.7 Development of the Second Order Fault Event, Current in Component Coalition #2 Too Long

The appropriate failure transfer functions to develop this Second Order Fault Event are one from each of the component's switch and timer contacts. Since both output gates for these failure transfer functions are AND, order of consideration is not important (see Section 5.3). The results are shown in Figure 16.

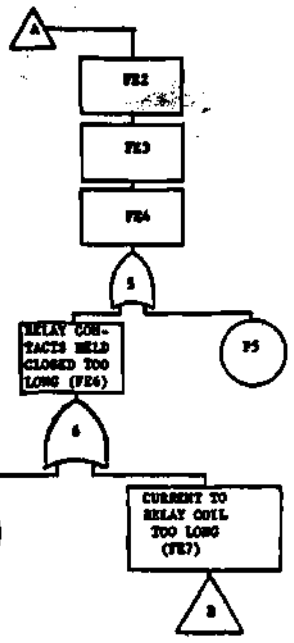

A
FE2
FE3
FE4
5
RELAY CON-TACTS HELD CLOSED TOO LONG (FE6)
F5
6
RELAY COIL FAILS CLOSED (F6)
CURRENT TO RELAY COIL TOO LONG (FE7)
B

Figure 15. Development Stage Number 6 of Sample Fault Tree Construction

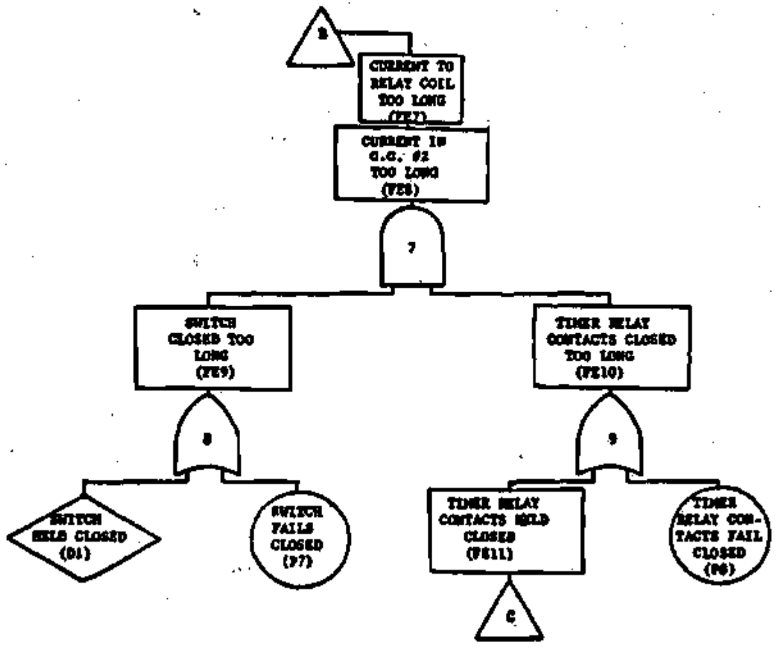

B
CURRENT TO RELAY COIL TOO LONG (FE7)
CURRENT IN C.C. #2 TOO LONG (FE8)
7
SWITCH CLOSED TOO LONG (FE9)
TIMER RELAY CONTACTS CLOSED TOO LONG (FE10)
8
9
SWITCH HELD CLOSED (D1)
SWITCH FAILS CLOSED (F7)
TIMER RELAY CONTACTS HELD CLOSED (FE11)
TIMER RELAY CON-TACTS FAIL CLOSED (F8)
C

Figure 16. Development Stage Number 7 of Sample Fault Tree Construction

Note that the "ordinarily" Fourth Order Fault Event, input to switch causes switch to be closed too long, appears in the diamond symbol in the tree. This happens because there is no input to the switch in the schematic, hence further development is not possible.

The new boundary conditions generated by this Second Order Fault Event are:

(1) No current in component coalition #2

(2) No current because of the relay coil

(3) No current because of the switch

(4) No current because of the timer contacts

(5) No current because of the power supply.

6.8 Development of the Fourth Order Fault Event, Input to Timer Relay Coil Contacts Causes Contacts to be Closed Too Long

From the schematic, it is seen that the timer relay coil supplies direct input to the relay contacts. The failure transfer function of the timer relay coil is, therefore, used to develop this Fourth Order Fault Event (see Section 5.5). The results are in Figure 17.

This event appears only in the component coalition #1 and, therefore, results from current applied too long in component coalition #1 as shown in Figure 18.

6.9 Development of the Second Order Fault Event, Current Applied Too Long in Component Coalition #1

The switch is the only component with a failure transfer function

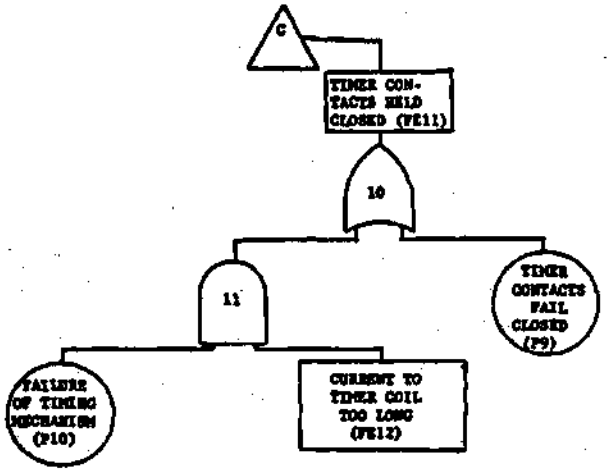

C
TIMER CONTACTS HELD CLOSED (FE11)
10
11
TIMER CONTACTS FAIL CLOSED (F9)
FAILURE OF TIMING MECHANISM (F10)
CURRENT TO TIMER COIL TOO LONG (FE12)

Figure 17. Development Stage Number 8 of Sample Fault Tree Construction

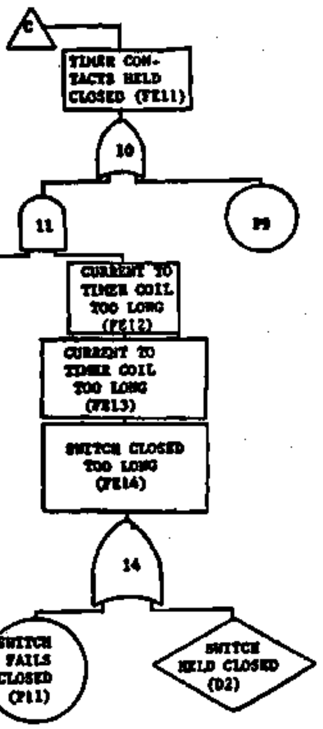

C
TIMER CONTACTS HELD CLOSED (FE11)
10
11
F9
F10
CURRENT TO TIMER COIL TOO LONG (FE12)
CURRENT TO TIMER COIL TOO LONG (FE13)
SWITCH CLOSED TOO LONG (FE14)
14
SWITCH FAILS CLOSED (F11)
SWITCH HELD CLOSED (D2)

Figure 18. Development Stage Number 9 of Sample Fault Tree Construction

able to develop this Second Order Fault Event. The results are shown in Figure 18. This Second Order Fault Event generates the new Not-allowed Event Condition no current in component coalition #1. This boundary condition will never be used, however, since the fault tree is complete.

Figures 12, 15, 16, and 18 then represent the final system fault trees for this example system. This fault tree is shown complete in Figure 19.

Note that a transfer symbol is used in Figure 19. This transfer was used only after both affected tree branches were developed to insure the effective boundary conditions were the same (see Section 5.6.1).

The fault tree shown in Figure 19 differs from the one presented by Haasl for this system. This is not to claim Haasl's fault tree was wrong. It does, however, demonstrate how variations in fault trees can occur. It is felt the fault tree constructed using STM does present more detail than was presented by Haasl.

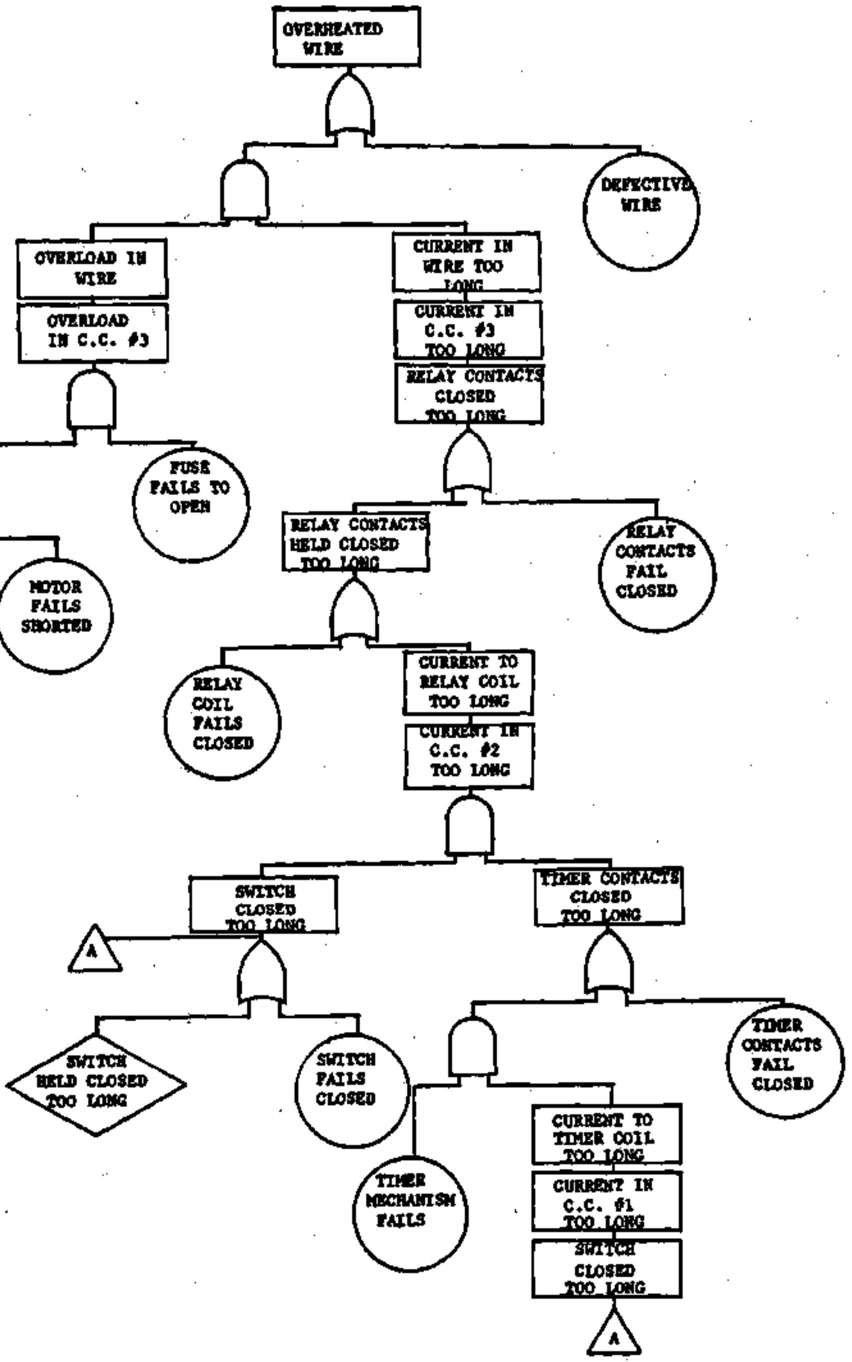

OVERHEATED WIRE
DEFECTIVE WIRE
OVERLOAD IN WIRE
OVERLOAD IN C.C. #3
CURRENT IN WIRE TOO LONG
CURRENT IN C.C. #3 TOO LONG
RELAY CONTACTS CLOSED TOO LONG
FUSE FAILS TO OPEN
POWER SUPPLY SURGE
MOTOR FAILS SHORTED
RELAY CONTACTS HELD CLOSED TOO LONG
RELAY CONTACTS FAIL CLOSED
RELAY COIL FAILS CLOSED
CURRENT TO RELAY COIL TOO LONG
CURRENT IN C.C. #2 TOO LONG
SWITCH CLOSED TOO LONG
TIMER CONTACTS CLOSED TOO LONG
A
SWITCH HELD CLOSED TOO LONG
SWITCH FAILS CLOSED
TIMER CONTACTS FAIL CLOSED
CURRENT TO TIMER COIL TOO LONG
CURRENT IN C.C. #1 TOO LONG
SWITCH CLOSED TOO LONG
TIMER MECHANISM FAILS
A

Figure 19. Complete Fault Tree for Sample System

CHAPTER VII

A COMPLETE, AUTOMATED PROBABILISTIC RELIABILITY PREDICTION USING SYNTHETIC TREE MODEL

In order to put forth the position of Synthetic Tree Model, in a complete quantitative analysis, such an analysis is presented here for a simple, but illustrative, example of a pressure tank system. This is not to imply that quantification of fault tree analysis is a necessary objective. A qualitative analysis often plays the most important role, especially in providing feedback to those involved in the system design. In addition, there is often considerable uncertainty in the data--that is, probabilistic information about the primary event--especially in the nuclear industry at this time. A quantification of fault tree analysis is, however, often desirable to determine the relative, if not absolute, reliability of a system.

7.1 Fault Tree Evaluation

Since the introduction of fault tree analysis, the area receiving the most research and development effort has been the evaluation of fault trees.[11,14,27-30] The evaluation of a fault tree is obtaining reliability information about the TOP event and perhaps the minimal cut-sets from the data supplied for the failure of the basic components. There have been basically three methods for solutions to fault trees presented to date: the direct simulation approach,[27] Monte Carlo methods,[31] and direct analytical solutions.[13]

The direct simulation approach basically uses Boolean logic hardware similar to that used in digital computers in a one-to-one correspondence with the fault tree Boolean logic to form an analog circuit. Immediately this method was seen to be prohibitively expensive. An effort was then made to obtain information from the fault tree by a hybrid method wherein parts of the solution were obtained using the analog technique and parts from a digital calculation in an effort to obtain a costwise competitive technique of solution. Because of the expense involved, this method has received a relatively small amount of attention.

Monte Carlo methods are perhaps the most simple in principle but in practice become outstandingly complex, as is the case with most uses of Monte Carlo. Until recently Monte Carlo was, for all practical purposes, the only computational method used for solving complex fault trees. Since Monte Carlo is not practical without the use of a digital computer, it will be discussed in that framework.

The most easily understood Monte Carlo technique is called "direct simulation."* Probability data are provided as input and the simulation program represents the fault tree on a computer to provide quantitative results. In this manner, thousands or millions of trial years of performance can be simulated. A typical simulation program involves the following steps:[7]

1. Assign failure data to input fault events within the tree, and if desired, repair data.

*The term "simulation" is used in conjunction with Monte Carlo methods frequently because Monte Carlo is, indeed, a form of mathematical simulation. This should not, however, be confused with the direct analog simulation as discussed above.

2. Represent the fault tree on a computer to provide quantitative results for the overall system performance, subsystem performance, and the basic input event performance.

3. List the failures that lead to the undesired event and identify minimal cutsets contributing event results.

4. Compute and rank basic input failure and availability performance results.

In accomplishing these steps, the computer program simulates the fault tree and, using the input data, randomly selects the various parameter data from assigned statistical distribution parameters, and then tests whether or not the specified final event occurred within the specified time period. Each test is a trial, and a sufficient number of trials is run until the desired quantitative resolution is obtained. Each time the final event occurs, the contributing effects of input events and the logical gates causing the specified final event are stored and listed as computer output. The resultant output provides a detailed perspective of the system under simulated operating conditions and provides a quantitative basis to support objective decisions.

The third method of solution is direct analytical solution. To illustrate how this might be done for a simple fault tree for static conditions, consider the following example. Consider the fault tree shown in Figure 20 that contains independent, primary events A, B, C, and D with constant probabilities of failure 0.1, 0.2, 0.3, and 0.4, respectively. This assumption of constant failure probabilities distinguishes this example from a realistic fault tree evaluation. The fault tree is, however, not in convenient form as shown in Figure 20, because events X1 and X2 are not independent since they both are functions of primary event B. By Boolean manipulation the fault tree shown in Figure 21 is equivalent to the one shown in Figure 20. The fault tree shown in Figure 21 is in convenient form for calculating the probability of the TOP event.

At this time it is necessary to introduce two basic laws of probability that are used in a fault tree evaluation:[22]

$$P(A1 \bigcup A2) = P(A1) + P(A2) - P(A1 \bigcap A2)$$

$$P(A1 \bigcap A2) = P(A1)P(A2/A1)$$

The first law simply states that the probability of a union $A1 \bigcup A2$ is the sum of the probabilities of the individual events minus the probability of their intersection. In terms of the fault tree, the probability of a two event OR gate is the sum of probabilities of the two events attached to the gate minus the probability of the two events both occurring. The second law states that the probability of an intersection of events $P(A1 \bigcap A2)$ is equal to the probability of one, P(A1), times the probability of the other, given the occurrence of the first, P(A1/A2). In terms of the fault tree, the probability of a two event AND gate is the product of the probabilities of the two attached events, since primary events of a fault tree are independent.

Since all events are independent in the fault tree shown in Figure 21, unlike the events of the tree shown in Figure 20, the event probabilities are as follows:

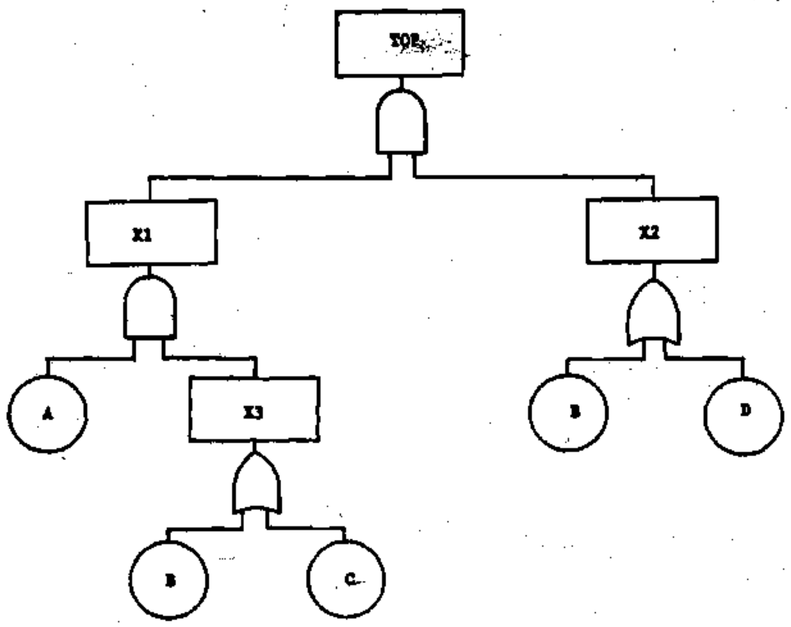


Figure 20. Sample Fault Tree for Probability Evaluation

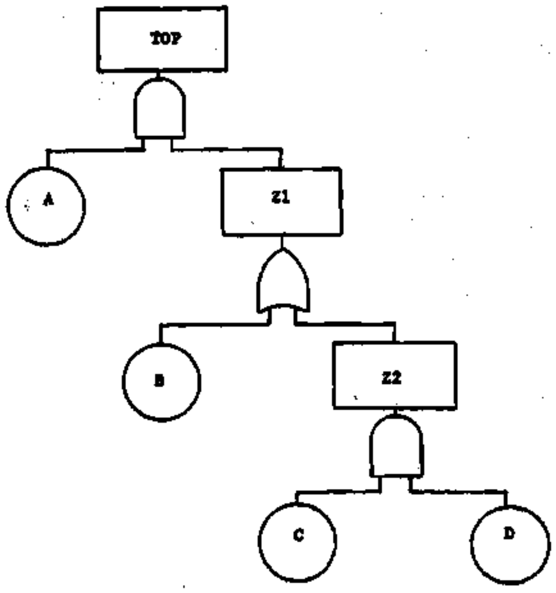


Figure 21. Boolean Equivalent of Sample Fault Tree Shown in Figure 3

$$P(Z2) = P(C)P(D)$$

$$P(Z1) = P(B) + P(Z2) - P(B)P(Z2)$$

$$P(TOP) = P(Z1)P(A)$$

Upon substitution

$$P(TOP) = P(A)P(B) + P(A)P(C)P(D) - P(A)P(B)P(C)P(D)$$

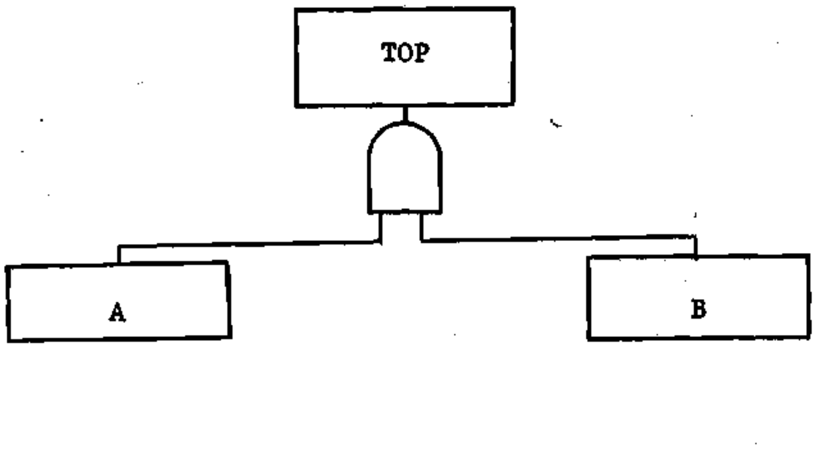
$$P(TOP) = 0.0236$$

This gives the probability of the system being in the failed state constant with respect to time and being 0.0236 for the given primary event failure probabilities. Also it is visible from the fault tree that the component most crucial to the system is A. This fault tree has two critical paths, AB and ACD. Primary event A appears in both critical paths. If the probability of event A can be reduced to one half of its original value, i.e., from 0.1 to 0.05, the system failure probability is reduced to 0.0118, or one half its original value given above.

In spite of the seeming simplicity of the above example, until very recently a practical method for solving complex fault trees analytically was not known for trees containing primary failures demonstrating failure probabilities as complex functions of time and repair possibilities.

With the advent of Kinetic Tree Theory[13] in 1970, such analytical solutions were possible for complex trees using relatively small amounts of computer time. Monte Carlo methods are sometimes used to obtain the critical paths of the fault tree as a prelude to Kinetic Tree Theory.[31] The solution of the fault tree itself is accomplished through a blend of probability theory and differential calculus.[32] Fault trees of any structure and of any complexity are handled. The use of AND, OR, and INHIBIT gates is allowed. General failure and repair distributions are handled; there is no limitation to these distributions as in other methodologies. Complete probabilistic information is first obtained for each primary failure of the fault tree, then for each minimal cutset and finally for the TOP failure itself. The information is obtained as a function of time, and, hence, with regard to reliability complete kinetic behavior is obtained. The expressions developed are in a simple form, and application to yield numerical results is both efficient and straightforward, with an average computer time on the order of one minute required for a 500 primary failure fault tree (on the IBM 360/75 computer).[13]

As an elementary example of a fault tree solution with failure and repair probabilities as functions of time, consider the case of two identical, independent system units, A and B, operating such that the simultaneous failure of both is required to cause system failures as shown in the fault tree below. There is then one minimal cut set, AB.



Let F(t) represent the time to failure distribution function. A repair facility is used such that the time to repair distribution function is represented by G(t).

$$F(t) = 1 - e^{-\lambda t}$$

$$G(t) = 1 - e^{-\mu t}$$

The quantity λ is termed the failure rate for a primary failure while μ is termed the repair rate. Both are assumed constant for this example. Let q(t) be the probability of the primary failure existing at time t. It has been shown that[33]

$$q(t) = \frac{\lambda}{\lambda + \mu} - \frac{\lambda}{\lambda + \mu} e^{-(\lambda + \mu)t} .$$

Now let Q(t) be defined as the probability that the TOP failure exists at time t. Since the TOP failure exists at time t if and only if all the primary failures exist at time t,

$$Q(t) = \prod_{j=1}^{2} q_j(t)$$

$$= [q(t)]^2$$

$$= \frac{\lambda^2 - 2\lambda^2 e^{-(\lambda+\mu)t} + \lambda^2 e^{-2(\lambda+\mu)t}}{(\lambda + \mu)^2}$$

The availability of the system A(t) then is given by

$$A(t) = 1 - Q(t)$$

$$= \frac{\mu^2 + 2\lambda\mu}{(\lambda + \mu)^2} - \frac{\lambda^2 e^{-2(\lambda+\mu)t}}{(\lambda + \mu)^2} + \frac{2\lambda^2 e^{-(\lambda+\mu)t}}{(\lambda + \mu)^2}$$

It is interesting to note that these are precisely the results obtained in reference 33 for a parallel redundant system configuration using the theory of Markov processes.

7.2 Pressure Tank System Example

The pressure tank system is shown in Figure 22. The system as designed has sufficient controls and interlocks such that the pump pressurizes the tank until a preset pressure has been reached or until a certain time has lapsed. To repeat the pressurization procedure, the reset switch must be momentarily closed. There is concern that the pump motor might run too long such that the tank becomes over-pressurized and ruptures. The pump motor operating too long is then the TOP event for this analysis. The timer relay is set such that, when operating properly, its contacts open if a preset amount of time lapses--less time than that required for the tank to become over-pressurized. The contacts will also open if the current is removed from the timer relay coil.

The pressure switch is designed to open its contacts when a predetermined pressure has been reached in the tank. The pump motor will then stop.

The availability of the system A(t) then is given by

$$A(t) = 1 - Q(t)$$

$$= \frac{\mu^2 + 2\lambda\mu}{(\lambda + \mu)^2} - \frac{\lambda^2 e^{-2(\lambda+\mu)t}}{(\lambda + \mu)^2} + \frac{2\lambda^2 e^{-(\lambda+\mu)t}}{(\lambda + \mu)^2}$$

It is interesting to note that these are precisely the results obtained in reference 33 for a parallel redundant system configuration using the theory of Markov processes.

7.2 Pressure Tank System Example

The pressure tank system is shown in Figure 22. The system as designed has sufficient controls and interlocks such that the pump pressurizes the tank until a preset pressure has been reached or until a certain time has lapsed. To repeat the pressurization procedure, the reset switch must be momentarily closed. There is concern that the pump motor might run too long such that the tank becomes over-pressurized and ruptures. The pump motor operating too long is then the TOP event for this analysis. The timer relay is set such that, when operating properly, its contacts open if a preset amount of time lapses--less time than that required for the tank to become over-pressurized. The contacts will also open if the current is removed from the timer relay coil.

The pressure switch is designed to open its contacts when a predetermined pressure has been reached in the tank. The pump motor will then stop.

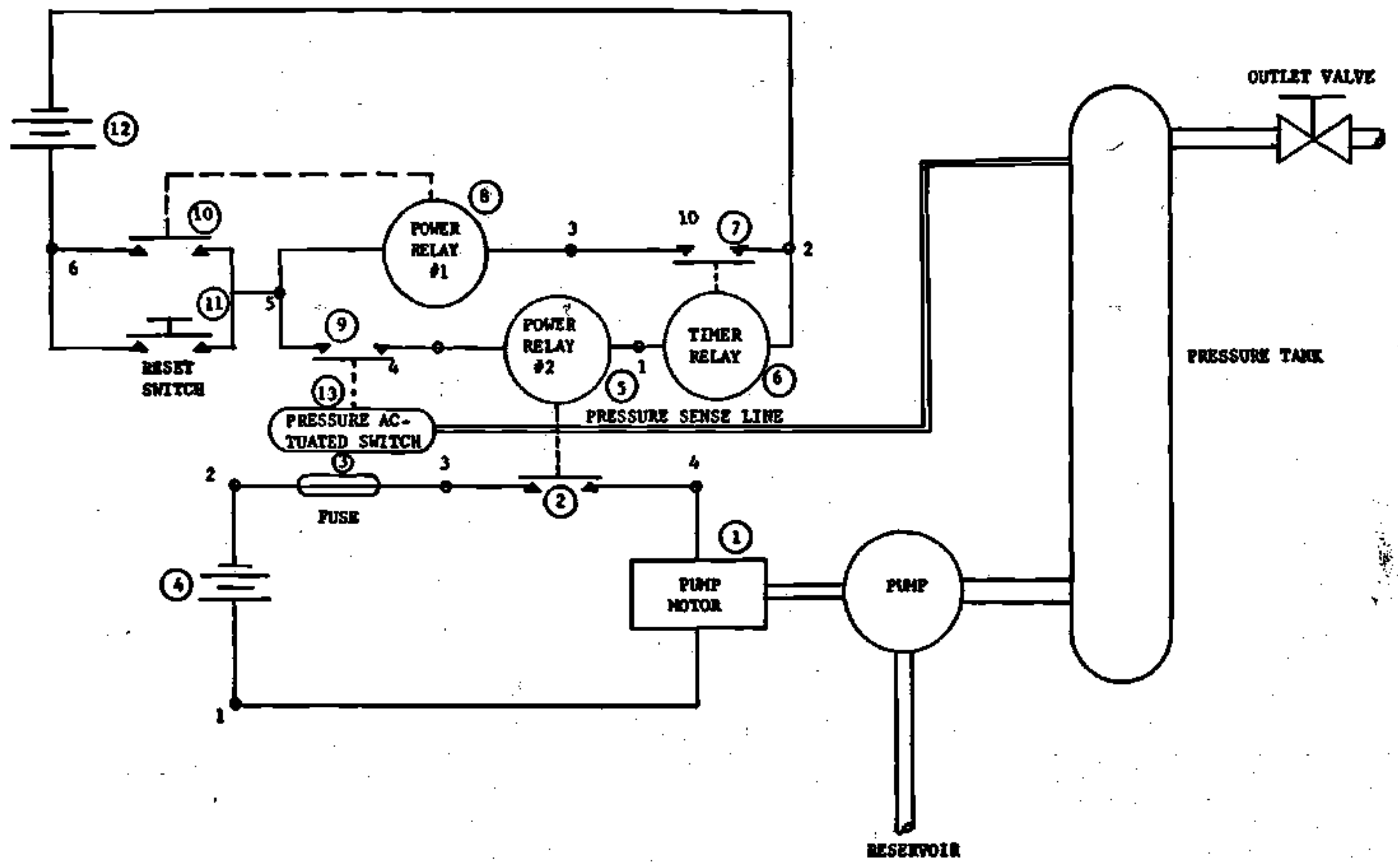


Figure 22. Schematic of Pressure Tank System

This analysis has three separate stages.

(1) The construction of the fault tree.

(2) The determination of the minimal cut sets for the fault tree.

(3) The quantitative analysis using these minimal cut sets and component probabilistic data.

Each stage is automated for the analyses given here. The first stage uses the computer program DRAFT as presented herein. The second uses MOCUS, a program to determine the minimal cut sets as described in reference 24. The final stage uses KITT-1, a program exercising Kinetic Tree Theory (see Section 2.5), as described in reference 32, to determine the quantitative aspects of the analysis.

<u>Step 1--The Use of DRAFT (see Appendix B)</u>

To prepare the input to DRAFT the schematic is first divided into panels as indicated in Figure 22. Components and nodes are numbered as shown. The components are assigned the device number corresponding to the appropriate library data (see Appendix A). The initial conditions are noted to be as follows:

Incident Identification	Entity Identification
contacts closed	2
contacts closed	9
contacts closed	10
contacts closed	7
contacts open	11

No events are declared Existing or Not-allowed System Boundary Conditions. The contact flag is not used. An input edit of these data is given in Table 1. A decoding sheet is given in Table 12.

The component coalitions are output from DRAFT and are given in Table 2 and the fault tree printout is given in Table 3.

Table 1. DRAFT Input Edit for Pressure Tank System

NUMBER OF PANELS 3

Panel Number	Component Number	Component Type	Node One	Node Two	Input Flag	Contact Flag
1	4	40	1	2	0	0
1	1	30	4	1	0	0
1	2	50	4	3	5	0
1	3	10	2	3	0	0
2	12	40	6	2	0	0
2	5	70	1	4	0	0
2	6	80	1	2	0	0
2	7	50	2	3	6	0
2	8	70	3	5	0	0
2	9	50	4	5	13	0
2	10	50	5	6	8	0
2	11	50	5	6	0	0
3	13	100	1	2	0	0

Boundary Conditions

TOP Event

5002	1

Initial Conditions

2003	2
2003	7
2003	9
2003	10
2001	11

Table 2. Component Coalitions for Pressure Tank System

	Component Coalitions					
Panel Number	Component Coalition Number	Component Number				
1	101	4	3	2	1	
2	102	12	6	5	9	10
2	103	12	6	5	9	11
2	104	12	7	8	10	
2	105	12	7	8	11	
3	106	13				

Table 3. Fault Tree Output from DRAFT for Pressure Tank System

Incident ID*	Entity ID*	Gate Type*	Gate Input Component Number*	Gate Input Incident ID*	Fault Tree Symbol*
Gate 1 (5002	1)		Gate 2		
Gate 2 (1004	1)		Gate 3		
Gate 3 (1004	101)		Gate 4		
Gate 4 (2004	2)	OR	Component (2	109)	Circle
			Gate 5		
Gate 5 (3004	2)	OR	Component (5	109)	Circle
			Gate 6		
Gate 66 (1004	5)	OR	Gate 7		
			Gate 8		
Gate 7 (1004	102)	AND	Gate 9		
			Gate 10		
Gate 8 (1004	103)	AND	Gate 9		
			Gate 11		
Gate 9 (2004	9)	OR	Component (9	109)	Circle
			Gate 12		
Gate 10 (2004	10)	OR	Component (10	109)	
			Gate 13		
Gate 11 (2004	11)	OR	Component (11	109)	Circle
			Component (11	3004)	Diamond
Gate 12 (3004	9)	OR	Component (13	109)	Circle
			Component (13	3005)	Diamond
Gate 13 (3004	10)	OR	Component (8	109)	Circle
			Gate 14		
Gate 14 (1004	8)	OR	Gate 15		
			Gate 16		
Gate 15 (1004	104)		Gate 17		
Gate 16 (1004	105)	AND	Gate 17		
			Gate 11		

Table 3. Concluded

Incident ID***	Entity ID****	Gate Type**	Gate Input Component Number***	Gate Input Incident ID****	Fault Tree Symbol**
Gate 17 (2004	7)	OR	Component (7	109)	Circle 10
			Gate 18		
Gate 18 (3004	7)	OR	Component (6	109)	Circle 11
			Gate 19		
Gate 19 (2006	6)	AND	Component (6	110)	Circle 12
			Gate 20		
Gate 20 (1004	6)	OR	Gate 21		
			Gate 22		
Gate 21 (1004	102)		Gate 9		
Gate 22 (1004	103)	AND	Gate 11		

This fault tree is drawn in Figure 23 directly from Table 4. The event descriptions are in coded form. An entity identification of less than 100 indicates a component number while one over 100 indicates a component coalition. Other decoding information is available in Table 12. The execution time for DRAFT on the UNIVAC 1108 computer was less than two seconds.

It is convenient at this time to point out some effects of the boundary condition on the fault tree in Figure 23. During the development of gate 15, current too long in component coalition number 104, one might expect that contacts number 10 being closed too long would be a fault event. However, since these contacts being closed is an initial condition and current too long in component coalition number 104 indicates current to the relay coil associated with contacts number 10, relay contacts number 10 being closed too long is an Existing Event Boundary Condition (see Section 4.6.2.4 and Table 10). Since the event was to be connected to an AND gate, the event is simply removed from the tree (see Section 5.2.2). Also, since the event descriptions of gates 7 and 22 are the same, one might expect a transfer is in order. Upon developing gate 22 it is found, however, that the Effective Boundary Conditions are not the same so the transfer cannot be made. Relay contacts number 10 being closed too long is an Effective Event Boundary Condition for gate 22 and not gate 7.

The output from DRAFT shown in Table 3 is modified somewhat to provide input to MOCUS. The minimal cut sets are output from MOCUS. These are shown in Table 4 and totally represent the fault tree to KITT-1. Execution time for MOCUS to locate the 23 minimal cut sets was less than

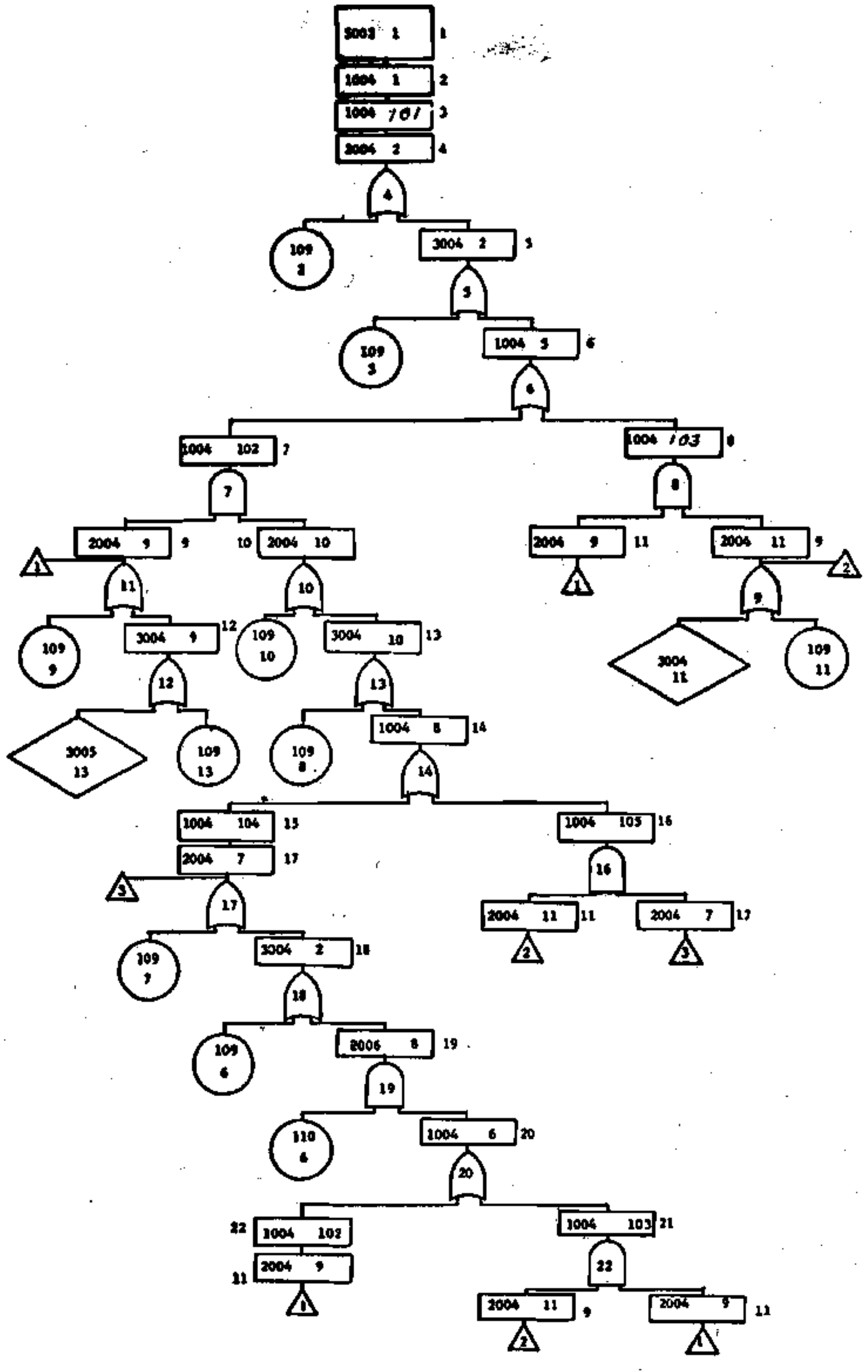


Figure 23. Fault Tree for Pressure Tank System

Table 4. Minimal Cut Sets for Pressure Tank System

Cutset Number	Entity ID	Incident ID
1	2 Power Relay #2 Contacts	Fail Closed
2	5 Power Relay #2 Coil	Fails Closed
3	9 Pressure Switch Contacts	Fail Closed
	10 Power Relay #1 Contacts	Fail Closed
4	9 Pressure Switch Contacts	Fail Closed
	11 Reset Switch	Fail Closed
5	13 Pressure Switch Mechanism	Fail Closed
	10 Power Relay #1 Contacts	Fail Closed
6	13 Pressure Switch Sensor Line	Fails Plugged
	10 Power Relay #1 Contact	Fail Closed
7	9 Pressure Switch Contacts	Fail Closed
	7 Timer Relay Contacts	Fail Closed
8	9 Pressure Switch Contacts	Fail Closed
	8 Power Relay #1 Coil	Fails Closed
9	13 Pressure Switch Mechanism	Fails Closed
	11 Reset Switch	Fails Closed
10	13 Pressure Switch Sensor Line	Fails Plugged
	11 Reset Switch	Fails Closed
11	9 Pressure Switch Contacts	Fails Closed
	11 Reset Switch	Held Closed
12	13 Pressure Switch Mechanism	Fails Closed
	7 Timer Relay Contacts	Fails Closed
13	13 Pressure Switch Mechanism	Fails Closed
	8 Power Relay Hot Coil	Fails Closed
14	13 Pressure Switch Sensor Line	Fails Plugged
	7 Timer Relay Contacts	Fails Closed
15	13 Pressure Switch Sensor Line	Fails Plugged
	8 Power Relay #1 Coil	Fails Closed
16	9 Pressure Switch Contacts	Fails Closed
	6 Timer Relay Coil	Fails Closed
17	13 Pressure Switch Mechanism	Fails Closed
	11 Reset Switch	Held Closed

Table 4. Concluded

Cutset Number	Entity ID	Incident ID
18	13 Pressure Switch Sensor Line	Fails Plugged
	11 Reset Switch	Held Closed
19	13 Pressure Switch Mechanism	Fails Closed
	6 Timer Relay Coil	Fails Closed
20	13 Pressure Switch Sensor Line	Fails Plugged
	6 Timer Relay Coil	Fails Closed
21	9 Pressure Switch Contacts	Fails Closed
	6 Timing Mechanism of Timer Relay	Fails Slow
22	13 Pressure Switch Mechanism	Fails Closed
	6 Timing Mechanism of Timer Relay	Fails Slow
23	13 Pressure Switch Sensor Line	Fails Closed
	6 Timing Mechanism of Timer Relay	Fails Slow

1.2 seconds.

The KITT-1 code obtains numerical probabilities by means of Kinetic Tree Theory,[13] a methodology by which exact, time-dependent probabilistic information is obtained. In the example presented here, only nonrepairability is considered; however, KITT-1 can handle constant repair times and exponential repair distributions as well.

Table 5 displays the primary event failure intensities, also commonly called failure rates, input to KITT-1. These constant failure intensities result in exponential failure distributions.

Figure 24 gives the probability the system is in in the failed state as a function of time which is output from KITT-1. A description of other output from KITT-1 is available in reference 32. Run time for KITT-1 was less than 12 seconds. The analysis is thereby complete.

Table 5. Failure Intensities for Components of Pressure Tank System

Primary Failure	Lambda (failures/hour)
Power Relay #2 Contacts Fail Closed	4.5×10^{-6}
Power Relay #2 Coil Fails Closed	0.5×10^{-6}
Pressure Switch Contacts Fail Closed	4.5×10^{-6}
Pressure Switch Mechanism Fails Closed	10.5×10^{-6}
Pressure Switch Sensor Line Fails Plugged	0.1×10^{-6}
Power Relay #1 Contacts Fail Closed	4.5×10^{-6}
Power Relay #1 Coil Fails Closed	0.5×10^{-6}
Reset Switch Fails Closed	1.0×10^{-6}
Timer Relay Contacts Fail Closed	4.5×10^{-6}
Timer Relay Coil Fails Closed	0.5×10^{-6}
Timer Relay Timing Mechanism Fails Slow	10.0×10^{-6}

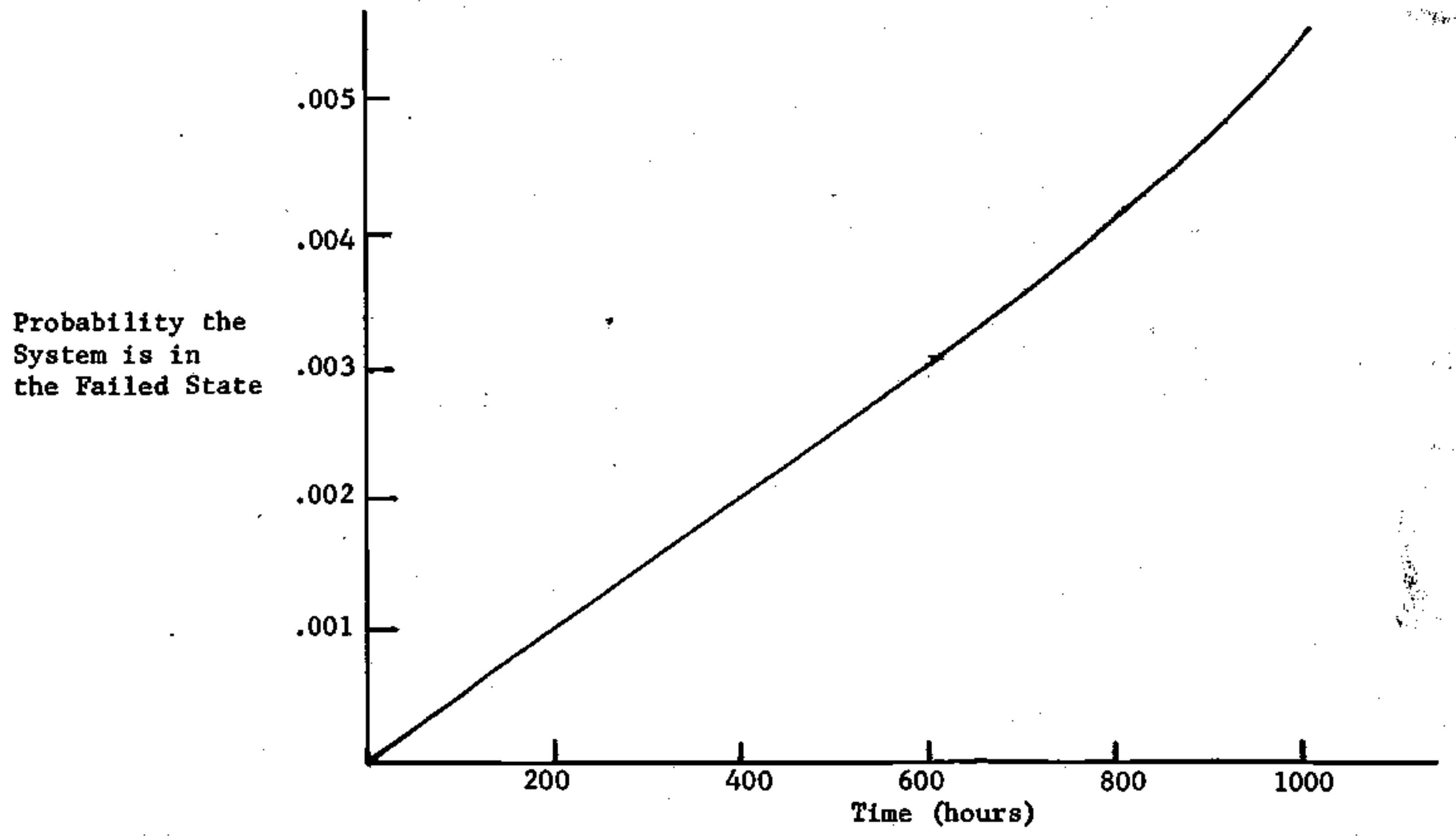


Figure 24. System Failed Probability vs. Time for the Pressure Tank System

CHAPTER VIII

CONCLUSIONS AND RECOMMENDATIONS

STM is a formal methodology for fault tree construction that has been developed to the point sufficient to allow automated fault tree construction to the level of primary failures for certain electrical systems. Extension of STM to other types of systems requires the determination of a sufficient number of values for each of five discrete characteristic factors. Some systems may, however, not lend themselves to STM since it may not be possible to define events of higher order than the First Order Fault Event. Recall that development of a First Order Fault Event to a level of higher order fault events is necessary to trigger STM into action. If such higher order fault events are not defined, the entire fault tree must be developed manually without the aid of STM. Indeed, there is no guarantee that the characteristic factors for any systems, other than those covered herein, can be determined.

While all the objectives were attained for STM, certain extensions and improvements of STM are recommended. STM should be extended to account for failure related feedback between components, commonly called secondary failures. This requires the use of the inhibit gate as described in Chapter II. This extension of STM will involve modifications of the transfer functions and possible call for additional ordered fault events. Since STM is implemented at this time only for electrical systems, another extension is the determination of the necessary characteristic factors to allow STM to be applied to various types of systems such as hydraulic and mechanical systems. Increasing the resolution of the fault trees constructed by STM is accomplished by adding detail to the failure transfer functions. This detailing and general broadening of the library data is an improvement that is needed if STM is to be applied to the complex, dynamic systems encountered in practical reliability studies in the nuclear industry and elsewhere.

Several improvements to the DRAFT program are needed. At present, the system given in Appendix C is representative of the largest that DRAFT can analyze. This results from only "in-core" computer storage being used. The implementation of rapid excess storage into DRAFT is required if the program is to be used for industrial systems. Also, the program is not programmed to execute in the most efficient manner. An application of sophisticated programming techniques is needed to hasten the execution.

Additional improvements to DRAFT that are recommended are:

(1) implementing all extensions to STM as they are developed;

(2) adding error messages to inform the user of mistakes in the input data;

(3) adding the option to allow System Boundary Conditions that are effective only after specified events have occurred, that is, conditional system boundary conditions;

(4) implementing a method to allow for changes in input data without re-execution of the program, that is, "change cases";

(5) writing a manual setting forth detailed information on the use of DRAFT; and, finally,

(6) a thorough checking of DRAFT on industrial systems is required.

The fault trees resulting from STM are in conventional format, use conventional symbols, and are constructed beginning with the main fault event of interest and proceeding to the individual component failure as is done in conventional fault tree construction. Actually, they differ from conventionally constructed fault trees in few ways. They do tend to be lengthy as STM uses no "short cuts" in its fault tree construction. A main difference is that, should any number of analysts construct fault trees independently for a given system and main failure event using STM, they will all obtain identical fault trees. This is not a characteristic of conventional fault tree construction.

STM provides an approach for totally automated reliability prediction as shown in Chapter VII. Automated prediction should be thought of as a distinct type of approach that could never replace conventional fault tree analysis. This automated tool could stop the system analyst from thinking. A value of the fault tree technique is that the analyst is forced to truly understand the system. Many system weaknesses are corrected while constructing the fault tree. A value of the technique is thus the construction process as well as the tree itself and resulting probability numbers. This automated analysis is a hardware oriented approach that does not include environmental and human effects that can cause failures and, therefore, is apart from a true in-depth fault tree analysis.

This is not to say automated analysis is undesirable; to the contrary, when verified on adequately complex systems, automated analysis could well become a routine type analysis. It could also provide an excellent start for a more in-depth fault tree analysis that includes environmental effects, common mode failure, and human errors. The automated analysis is of course extremely fast and frees the analyst from the routine hardware oriented fault tree construction as well as eliminating logic errors and errors of oversight in this part of the analysis. Automated analysis then affords the analyst a powerful tool to allow his prime efforts to be devoted to unearthing more subtle aspects of the modes of failure of the system.

While automation of fault tree construction has been accomplished using STM, application to manual fault tree construction could provide an immediate impact. The technique of STM can be easily and quickly learned. In fact, during the 1972 summer quarter, nuclear engineering students at the Georgia Institute of Technology who were new to the field of reliability analysis demonstrated that, once STM has been observed, it is awkward to construct fault trees any other way.

APPENDICES

APPENDIX A

EXAMPLE OF LIBRARY DATA OF SYNTHETIC TREE MODEL

The library data of STM include

(1) the component failure transfer functions

(2) the class of the Third Order Fault Event

(3) the category of the Second Order Fault Event

(4) the Inter-Correlation between the Second Order Fault Events and the Boundary Conditions

(5) Library of First Order Fault Events developed to higher order fault events.

The library need not contain all possible values of these parameters but only a sufficient number to construct fault trees for systems in question. A listing of such sufficient library data will be provided here. The listing is, in fact, sufficient for construction of all fault trees appearing in this dissertation.

Components Failure Transfer Functions

Figures 25 through 33 display the failure transfer function library data for the following components.

Components	Device No.
fuse	10
electric motor	30
power supply	40
contacts	50
circuit breaker coil	70

Components	Device No.
relay coil	60
timer relay coil	80
wiring	90
pressure switch	100

The "D" above Boolean logic diagrams indicates the discriminator value.

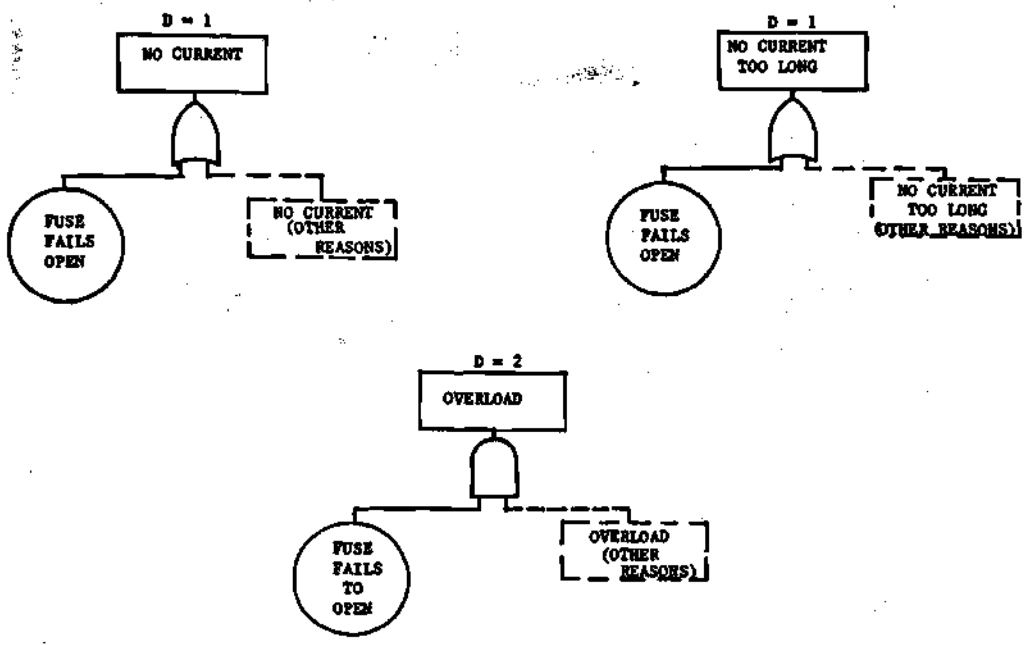


Figure 25. Failure Transfer Functions for a Fuse

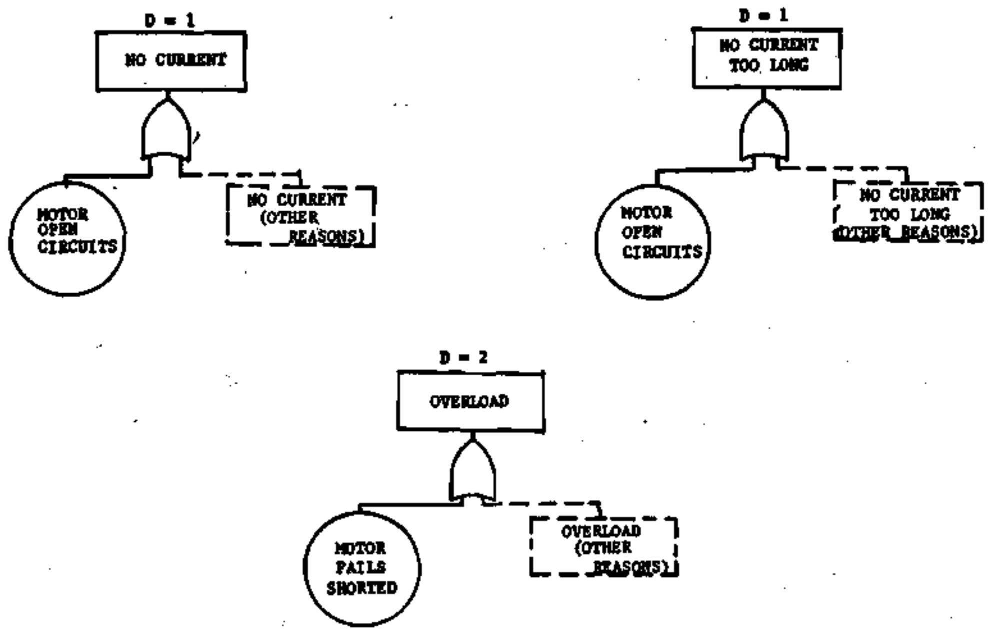


Figure 26. Failure Transfer Functions for an Electric Motor

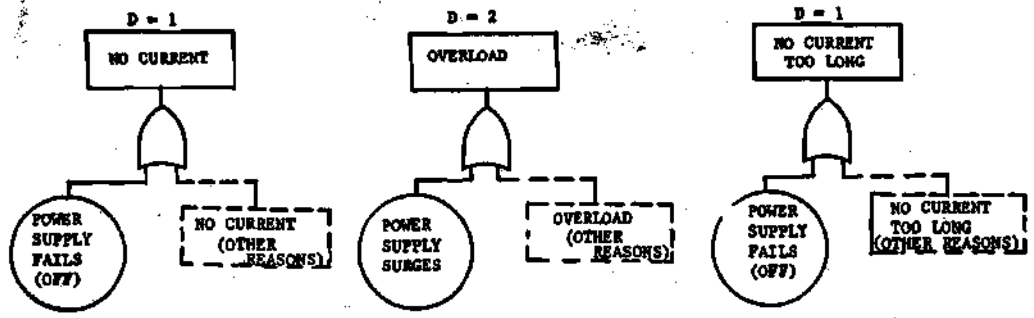


Figure 27. Failure Transfer Functions for a Power Supply

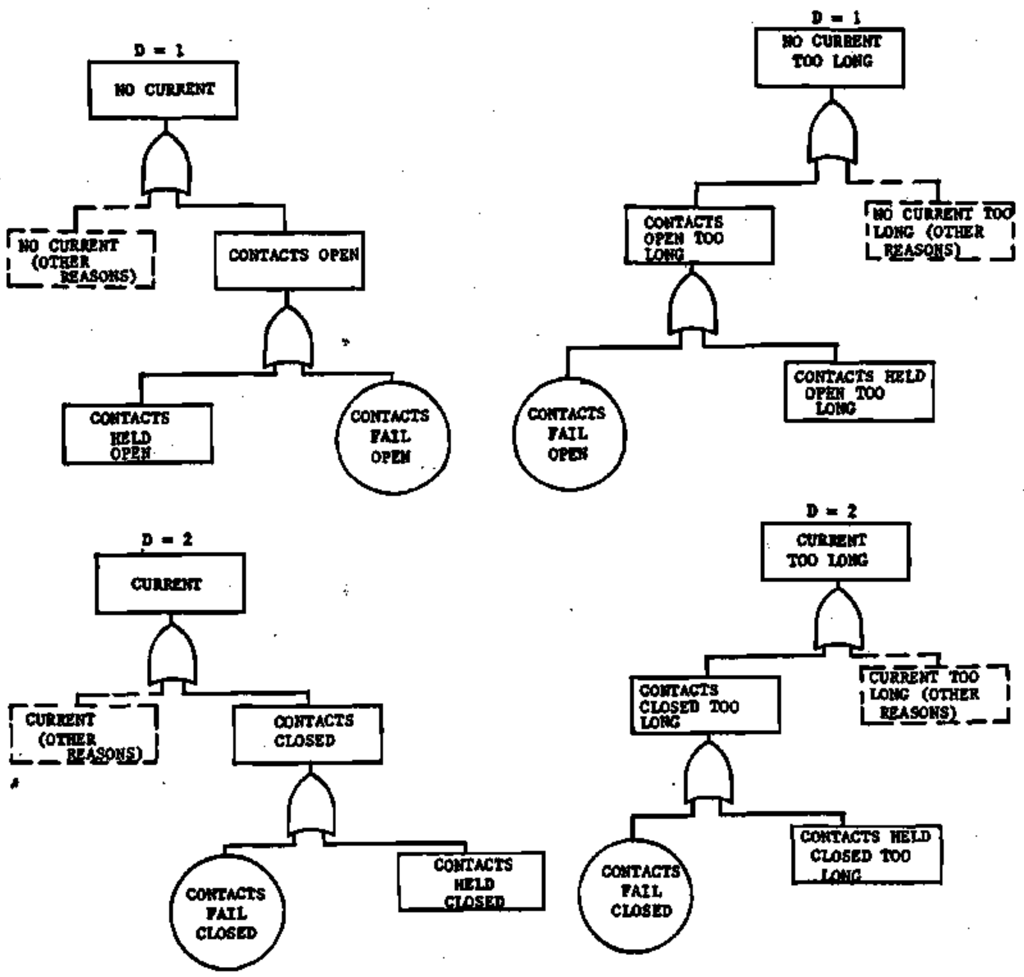


Figure 28. Failure Transfer Functions for Contacts

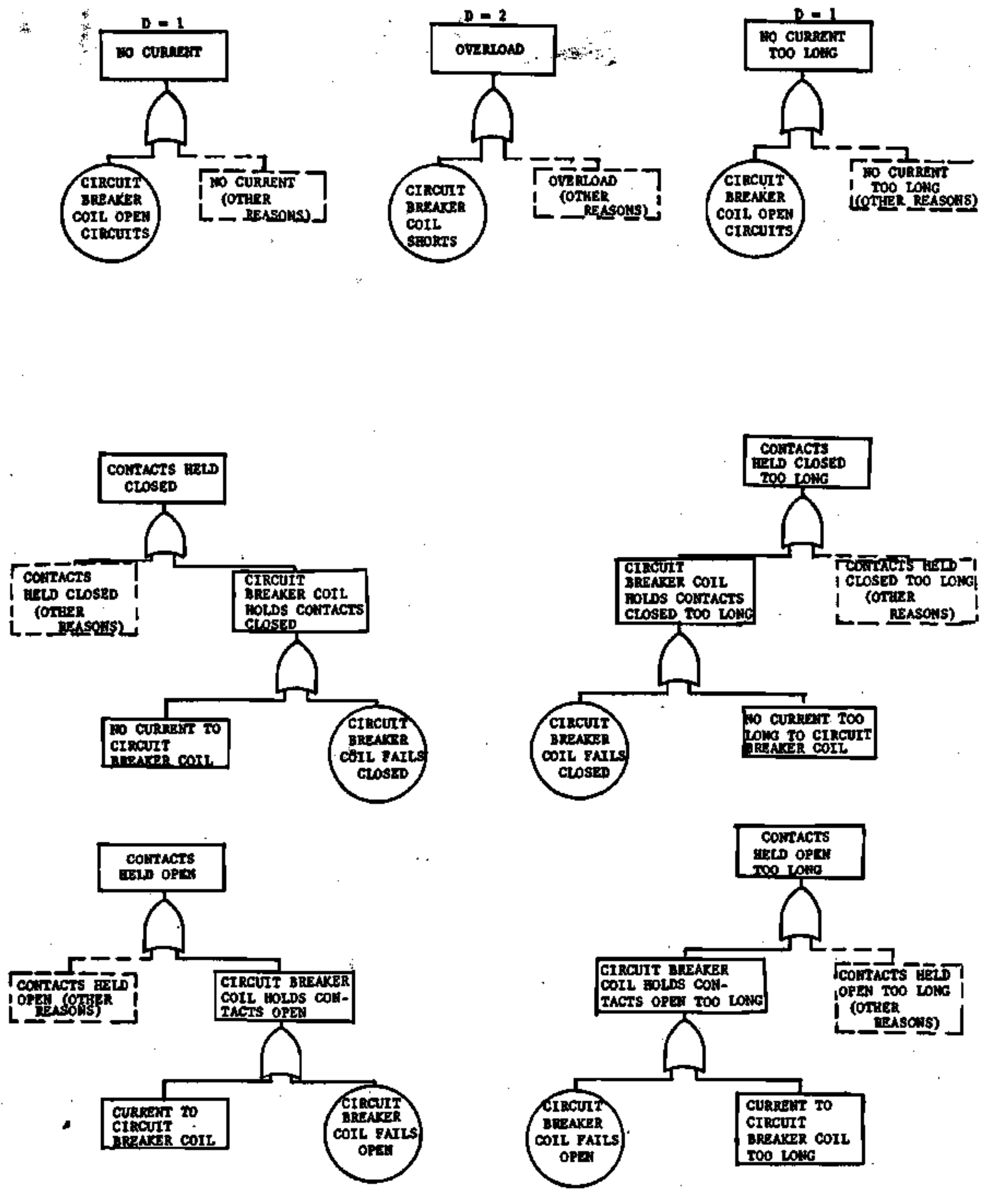
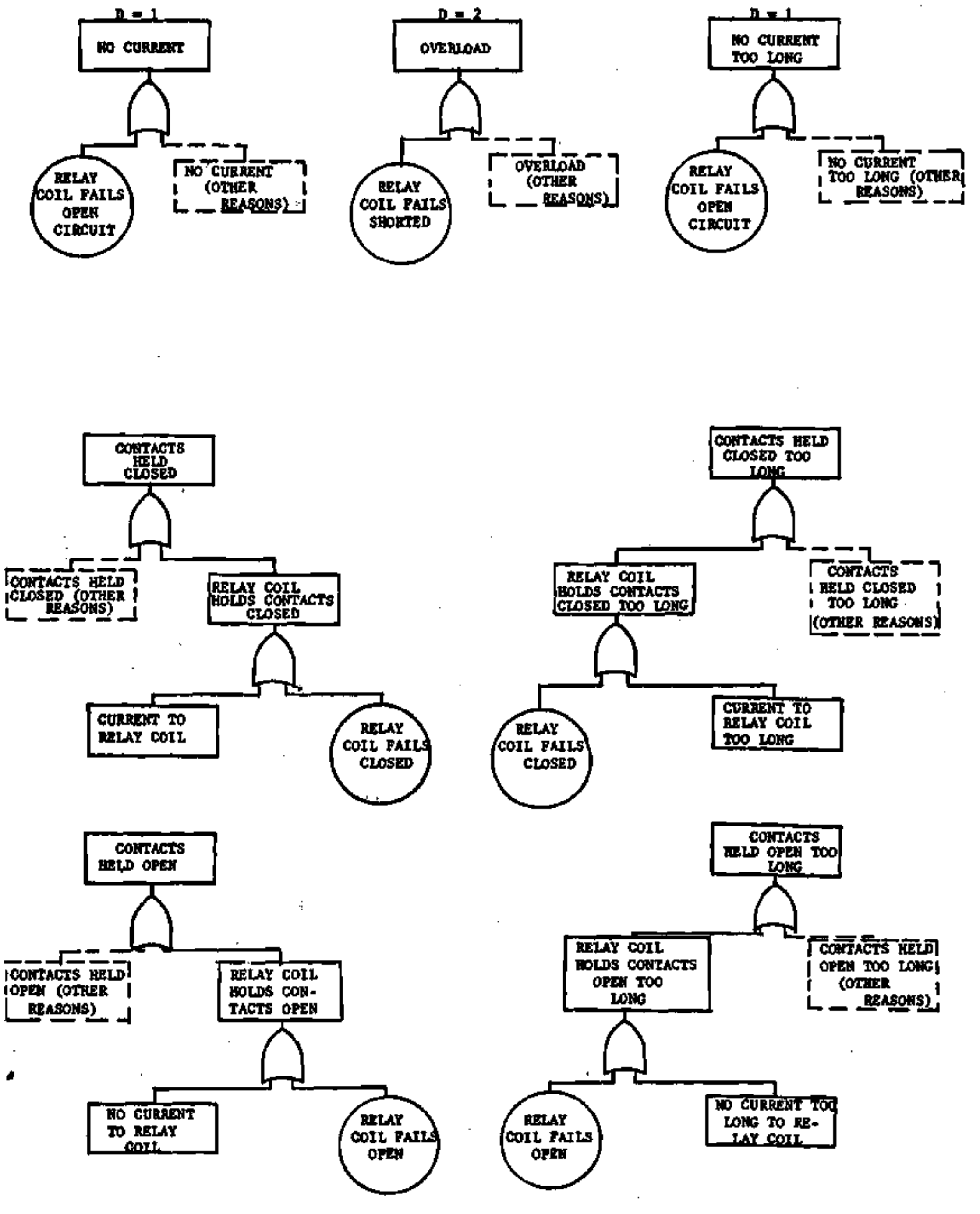

D = 1
NO CURRENT
CIRCUIT BREAKER COIL OPEN CIRCUITS
NO CURRENT (OTHER REASONS)
D = 2
OVERLOAD
CIRCUIT BREAKER COIL SHORTS
OVERLOAD (OTHER REASONS)
D = 1
NO CURRENT TOO LONG
CIRCUIT BREAKER COIL OPEN CIRCUITS
NO CURRENT TOO LONG (OTHER REASONS)
CONTACTS HELD CLOSED
CONTACTS HELD CLOSED (OTHER REASONS)
CIRCUIT BREAKER COIL HOLDS CONTACTS CLOSED
NO CURRENT TO CIRCUIT BREAKER COIL
CIRCUIT BREAKER COIL FAILS CLOSED
CONTACTS HELD CLOSED TOO LONG
CIRCUIT BREAKER COIL HOLDS CONTACTS CLOSED TOO LONG
CONTACTS HELD CLOSED TOO LONG (OTHER REASONS)
CIRCUIT BREAKER COIL FAILS CLOSED
NO CURRENT TOO LONG TO CIRCUIT BREAKER COIL
CONTACTS HELD OPEN
CONTACTS HELD OPEN (OTHER REASONS)
CIRCUIT BREAKER COIL HOLDS CONTACTS OPEN
CURRENT TO CIRCUIT BREAKER COIL
CIRCUIT BREAKER COIL FAILS OPEN
CONTACTS HELD OPEN TOO LONG
CIRCUIT BREAKER COIL HOLDS CONTACTS OPEN TOO LONG
CONTACTS HELD OPEN TOO LONG (OTHER REASONS)
CIRCUIT BREAKER COIL FAILS OPEN
CURRENT TO CIRCUIT BREAKER COIL TOO LONG

Figure 29. Failure Transfer Functions for a Circuit Breaker Coil



D = 1
NO CURRENT
RELAY COIL FAILS OPEN CIRCUIT
NO CURRENT (OTHER REASONS)
D = 2
OVERLOAD
RELAY COIL FAILS SHORTED
OVERLOAD (OTHER REASONS)
D = 1
NO CURRENT TOO LONG
RELAY COIL FAILS OPEN CIRCUIT
NO CURRENT TOO LONG (OTHER REASONS)
CONTACTS HELD CLOSED
CONTACTS HELD CLOSED (OTHER REASONS)
RELAY COIL HOLDS CONTACTS CLOSED
CURRENT TO RELAY COIL
RELAY COIL FAILS CLOSED
CONTACTS HELD CLOSED TOO LONG
RELAY COIL HOLDS CONTACTS CLOSED TOO LONG
CONTACTS HELD CLOSED TOO LONG (OTHER REASONS)
RELAY COIL FAILS CLOSED
CURRENT TO RELAY COIL TOO LONG
CONTACTS HELD OPEN
CONTACTS HELD OPEN (OTHER REASONS)
RELAY COIL HOLDS CONTACTS OPEN
NO CURRENT TO RELAY COIL
RELAY COIL FAILS OPEN
CONTACTS HELD OPEN TOO LONG
RELAY COIL HOLDS CONTACTS OPEN TOO LONG
CONTACTS HELD OPEN TOO LONG (OTHER REASONS)
RELAY COIL FAILS OPEN
NO CURRENT TOO LONG TO RELAY COIL

Figure 30. Failure Transfer Functions for a Relay Coil

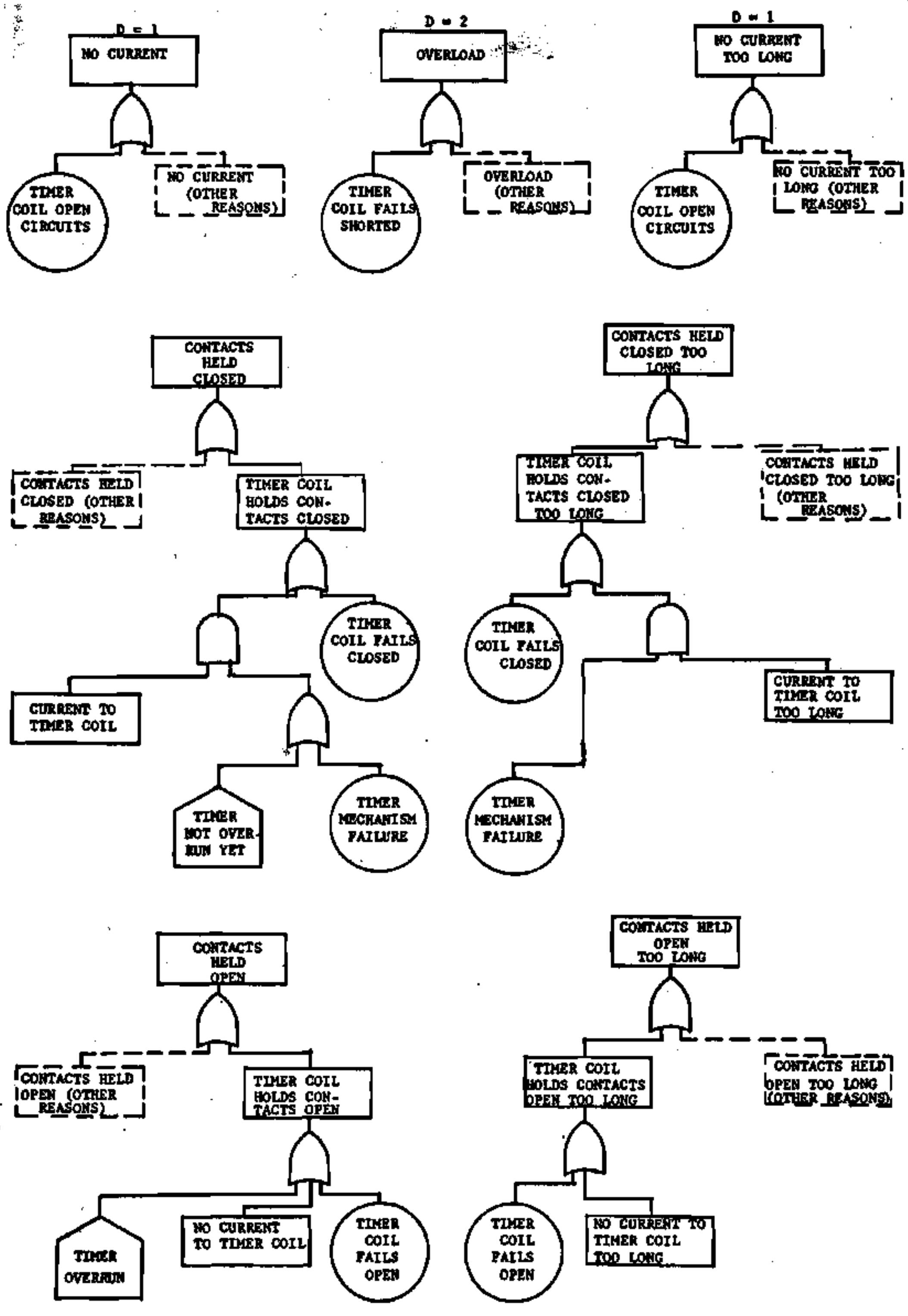

D = 1
NO CURRENT
TIMER COIL OPEN CIRCUITS
NO CURRENT (OTHER REASONS)
D = 2
OVERLOAD
TIMER COIL FAILS SHORTED
OVERLOAD (OTHER REASONS)
D = 1
NO CURRENT TOO LONG
TIMER COIL OPEN CIRCUITS
NO CURRENT TOO LONG (OTHER REASONS)
CONTACTS HELD CLOSED
CONTACTS HELD CLOSED (OTHER REASONS)
TIMER COIL HOLDS CONTACTS CLOSED
TIMER COIL FAILS CLOSED
CURRENT TO TIMER COIL
TIMER NOT OVERRUN YET
TIMER MECHANISM FAILURE
CONTACTS HELD CLOSED TOO LONG
TIMER COIL HOLDS CONTACTS CLOSED TOO LONG
CONTACTS HELD CLOSED TOO LONG (OTHER REASONS)
TIMER COIL FAILS CLOSED
CURRENT TO TIMER COIL TOO LONG
TIMER MECHANISM FAILURE
CONTACTS HELD OPEN
CONTACTS HELD OPEN (OTHER REASONS)
TIMER COIL HOLDS CONTACTS OPEN
TIMER OVERRUN
NO CURRENT TO TIMER COIL
TIMER COIL FAILS OPEN
CONTACTS HELD OPEN TOO LONG
TIMER COIL HOLDS CONTACTS OPEN TOO LONG
CONTACTS HELD OPEN TOO LONG (OTHER REASONS)
TIMER COIL FAILS OPEN
NO CURRENT TO TIMER COIL TOO LONG

Figure 31. Failure Transfer Functions for a Timer Relay Coil

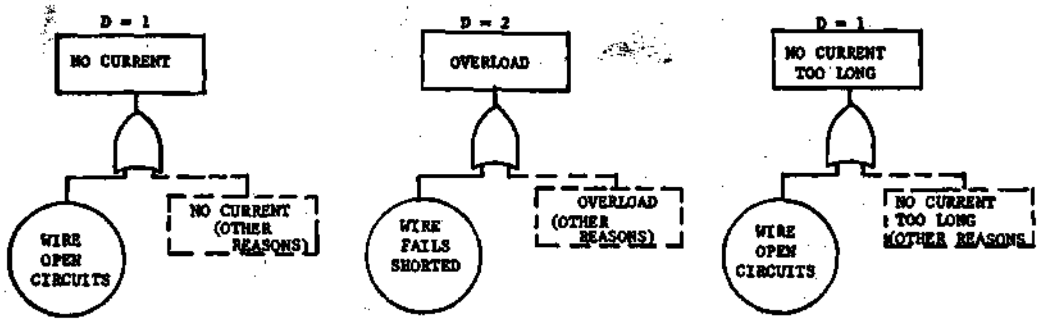


Figure 32. Failure Transfer Functions for Wiring

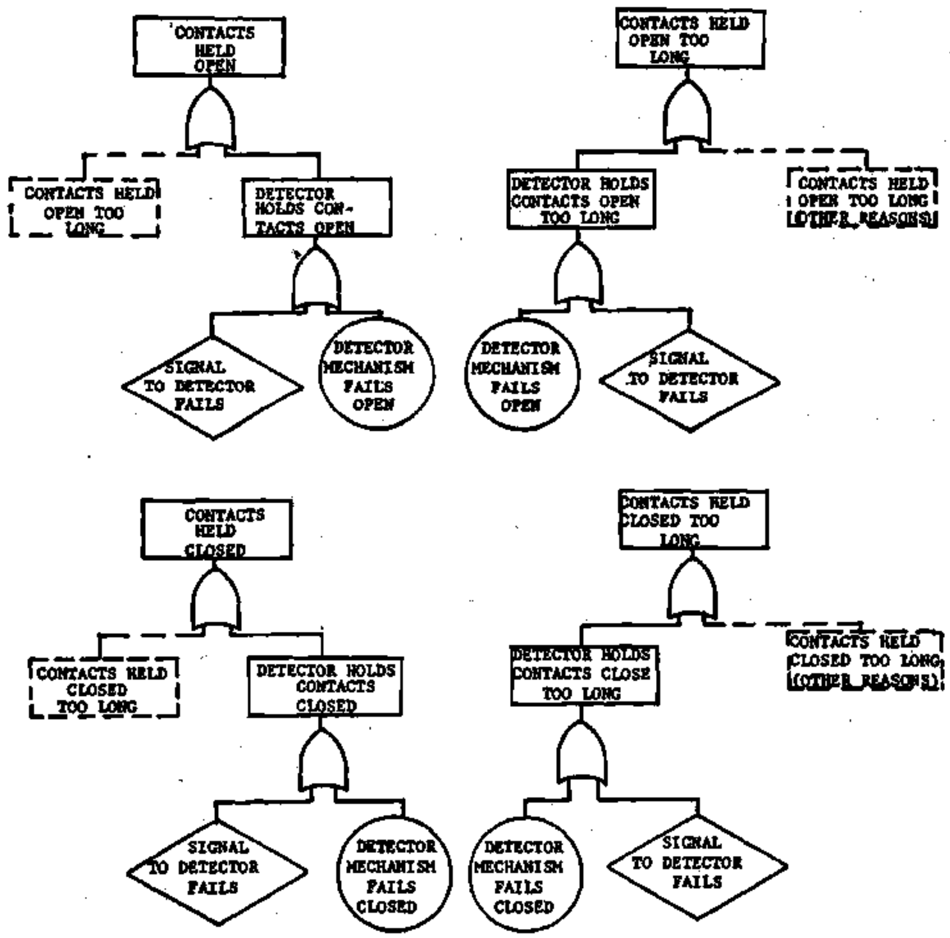

CONTACTS HELD OPEN
CONTACTS HELD OPEN TOO LONG
DETECTOR HOLDS CONTACTS OPEN
SIGNAL TO DETECTOR FAILS
DETECTOR MECHANISM FAILS OPEN
CONTACTS HELD OPEN TOO LONG
DETECTOR HOLDS CONTACTS OPEN TOO LONG
CONTACTS HELD OPEN TOO LONG (OTHER REASONS)
DETECTOR MECHANISM FAILS OPEN
SIGNAL TO DETECTOR FAILS
CONTACTS HELD CLOSED
CONTACTS HELD CLOSED TOO LONG
DETECTOR HOLDS CONTACTS CLOSED
SIGNAL TO DETECTOR FAILS
DETECTOR MECHANISM FAILS CLOSED
CONTACTS HELD CLOSED TOO LONG
DETECTOR HOLDS CONTACTS CLOSE TOO LONG
CONTACTS HELD CLOSED TOO LONG (OTHER REASONS)
DETECTOR MECHANISM FAILS CLOSED
SIGNAL TO DETECTOR FAILS

Figure 33. Failure Transfer Functions for a Pressure Switch

Table 6. Class of the Third Order Fault Event Library Data

Class I	Class II
No Current	Current
No Current Too Long	Current Too Long
	Overload

Table 7. Category of the Second Order Fault Event Library Data

Category I	Category II
No Current	Current
No Current Too Long	Current Too Long
Overload	

Table 8. Inter-Correlation Between Second Order Fault Events and Boundary Condition Library Data

Type 1 Second Order Fault Event Boundary Condition

The Occurrence of This Fault Event in a Given Component Coalition	Generates These Not-allowed Event Boundary Conditions for the Same Component Coalition
No Current	Current Current Too Long Overload
No Current Too Long	Current Current Too Long Overload
Current	No Current No Current Too Long
Current Too Long	No Current No Current Too Long
Overload	No Current No Current Too Long

Table 9. Inter-Correlation Between Second Order Fault Events and Boundary Condition Library Data

Type 2 Second Order Fault Event Boundary Condition

The Occurrence of This Fault Event in a Given Component Coalition	Generates the Transfer Function of Every Component in the Component Coalition with These Output Events, if any, as Not-allowed Event Boundary Conditions
Current	No Current No Current Too Long
Current Too Long	No Current No Current Too Long
Overload	No Current No Current Too Long

In spite of the similarities of this Not-allowed Event Boundary Condition generation to those of Type 1 Second Order Fault Event Boundary Condition generation, they are different since a component can appear in many component coalitions.

Table 10. Inter-Correlation Between Second Order Fault Events and Boundary Condition Library Data

Type 4 Second Order Fault Event Boundary Condition

Events (X)	Initial Conditions (Y)			
	Relay Coil		Circuit Breaker Coil	
	Contacts Open	Contacts Closed	Contacts Open	Contacts Closed
current in any component coalitions containing the coil		contacts closed	contacts open	
no current in every component coalition containing the coil	contacts open			contacts closed
current too long in any component coalition containing the coil		contacts closed too long	contacts open too long	
no current too long in every component coalition containing the coil	contacts open too long			contacts closed too long
overloading the component coalitions containing the coil		contacts closed	contacts open	

For the transfer functions presented here the only Existing Event Boundary Condition generation concerns the relay coil and circuit breaker coil.

This chart is interpreted as follows. Given initial condition, Y, the occurrence of Second Order Fault Events, X, gives rise to the Existing Event Boundary Condition at the intersection of X and Y.

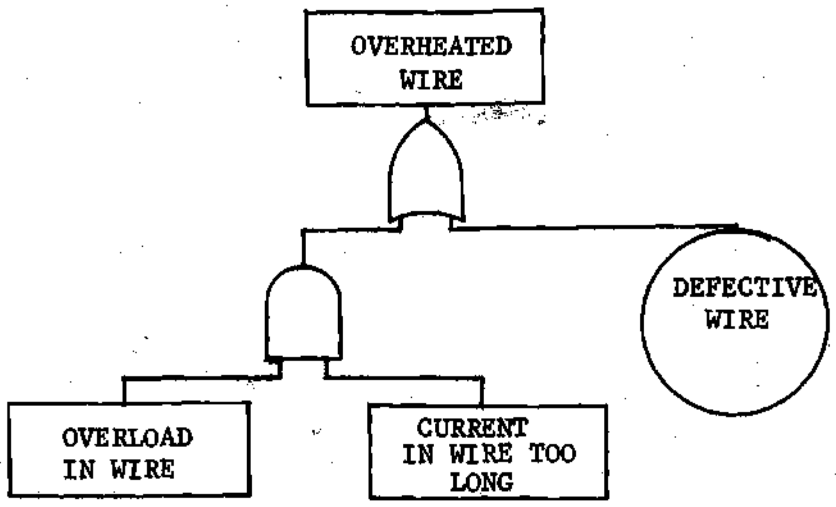


Figure 34. First Order Fault Event Development for Overheated Wire

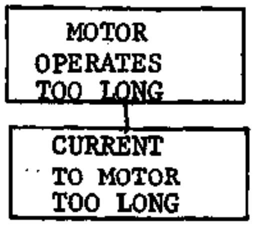


Figure 35. First Order Fault Event Development for Motor Operating Too Long

APPENDIX B

BASIC DESCRIPTION OF DRAFT

DRAFT is a computer program that constructs hardware oriented fault trees for electrical systems to the level of primary failures. DRAFT exercises STM. The input consists of the system schematic and the system boundary conditions. The availability of library data similar to those shown in Appendix A is assumed in coded form.

DRAFT was written for the UNIVAC 1108 computer and will construct a fault tree with 100 gates in typically less than seven seconds. Storage requirements are a limiting factor in the application of DRAFT because of the extensive, necessary bookkeeping associated with the Event Boundary Conditions. At this time, all storage and calculations are done in the 65,000 decimal words of core of the UNIVAC 1108. The fault tree example given in Appendix C is typically the largest that can, thereby, be constructed by DRAFT at this time.

While it is not the purpose of this thesis to present a computer code suitable for industrial applications, indeed such a code has not been developed, the program does verify that STM is formal and does afford a non-intuitive analyst, the UNIVAC 1108, to apply the model.

Input

The electrical schematic must be prepared for input to DRAFT. Each component is given a unique number and is correlated to an item in the library which already has a unique number (see Appendix A). The schematic is divided into panels. A component that does not receive electrical power, such as a mechanical pressure switch, must be in a panel by itself. No two panels can have common wiring; therefore, a component can appear in component coalitions of only one panel. Each panel must have one and may have more than one power supply. There can be interfacing between panels by coupling between one or more components. There can also be such interplay within a given panel. The component coalitions are independent of the mechanical interplay (see Sections 4.3.1 and 4.3.2).

The components of each panel are separated by a minimum number of uniquely numbered nodes. The interplay between components is correlated by noting the component number of component A from which component B receives input.

A contact flag can also be input, indicating if a switch is to be always open. This is useful for masking out certain portions of a system for a given analysis.

The System Boundary Conditions are then input in coded form. The code has two parts, the incident identification and the entity identification. The incident identification is coded to describe the event while the entity identification is simply the component number.

These System Boundary Conditions include the TOP event, the initial conditions, and events that are declared to be existing or not-allowed boundary conditions during the duration of the fault tree construction. This concludes the input description to DRAFT. An actual input edit is given in Table 11 for the system analyzed manually in Chapter VII. A decoding list is given in Table 12.

Table 11. Example Input Edit for DRAFT

NUMBER OF PANELS 3

Panel Number	Component Number	Component Type	Node One	Node Two	Input Flag	Contact Flag
1	2	40	1	2	0	0
1	3	90	5	1	0	0
1	1	10	2	3	0	0
1	4	30	4	5	0	0
1	5	50	4	3	7	0
2	10	40	8	9	0	0
2	6	50	6	7	8	0
2	7	70	8	6	0	0
2	8	80	8	7	0	0
2	9	50	7	9	0	0

Boundary Conditions

TOP Event
```
 5004  5
```

Initial Conditions
```
 2003  5
 2003  6
```

Table 12. Decoding List for Incident Identification

	Code Number	Description
For First Order Fault Events	5004	Device overheats
	5002	Device operates too long
For Second Order Fault Events and Third Order Fault Events	1001	No current
	1002	No current too long
	1003	Current
	1004	Current too long
	1005	Overload
For Fourth Order Fault Events	3001	Input holds device open
	3002	Input holds device open too long
	3003	Input holds device closed
	3004	Input holds device closed too long
	3005	Input to detector fails to reach detector
Failure Transfer Function Internal Events	2001	Contacts open
	2002	Contacts open too long
	2003	Contacts closed
	2004	Contacts closed too long
	2005 - 2600	Dummy identifications
Primary Events	101	Fuse opens
	102	Fuse fails to open
	103	Device fails to perform as designed
	104	Device open circuits
	105	Device short circuits
	106	Power supply fails (OFF)
	107	Power supply surges
	108	Device fails open
	109	Device fails closed
	110	Device defects cause failure
House Events	501	Timer overrun
	502	Timer not overrun

Output

Output from DRAFT includes a description of the component coalitions and the constructed fault tree. The event descriptions are in coded form and are also decoded using Table 12. The output for the system analyzed manually in Chapter VII is given in Tables 13 and 14. The form of the output fault tree is a conventional fault tree representation. The first gate is the TOP event and logic gate with the inputs to the gate are provided. These inputs can be gates or primary events. The graphical fault tree representation is directly implied by this output. It is convenient to draw the fault tree manually directly from this output. Entity identification of over 100 are component coalition numbers while those under 100 indicate component numbers.

Routine

DRAFT executes basically in a manner described by the diagram shown as Figure 36.

Table 13. Example of Component Coalition Output from DRAFT

	Component Coalitions					
Panel Number	Component Coalitions Number	Component I. D.				
1	101	2	1	5	4	3
2	102	10	9	6	7	
2	103	10	9	8		

Table 14. Example of Fault Tree Output from DRAFT

				Gate Input		
	Incident ID	Entity ID	Gate Type	Component Number	Incident ID	Fault Tree Symbol
Gate 1	(5004	3)	OR	Component (3	110)	Circle
				Gate 2		
Gate 2	(2500	3)	AND	Gate 3		
				Gate 4		
Gate 3	(1004	3)		Gate 5		
Gate 4	(1005	3)		Gate 6		
Gate 5	(1004	101)		Gate 7		
Gate 6	(1005	101)	AND	Component (1	102)	Circle
				Gate 8		
Gate 7	(2004	5)	OR	Component (5	109)	Circle
				Gate 9		
Gate 8	(2600	0)	OR	Component (2	107)	Circle
				Component (4	105)	Circle
Gate 9	(3004	5)	OR	Component (7	109)	Circle
				Gate 10		
Gate 10	(1004	7)		Gate 11		
Gate 11	(1004	102)	AND	Gate 12		
				Gate 13		
Gate 12	(2004	9)	OR	Component (9	109)	Circle
				Component (9	3004)	Diamond
Gate 13	(2004	6)	OR	Component (6	109)	Circle
				Gate 14		
Gate 14	(3004	6)	OR	Component (8	109)	Circle
				Gate 15		
Gate 15	(2006	8)	AND	Component (8	110)	Circle
				Gate 16		
Gate 16	(1004	8)		Gate 17		
Gate 17	(1004	103)		Gate 12		

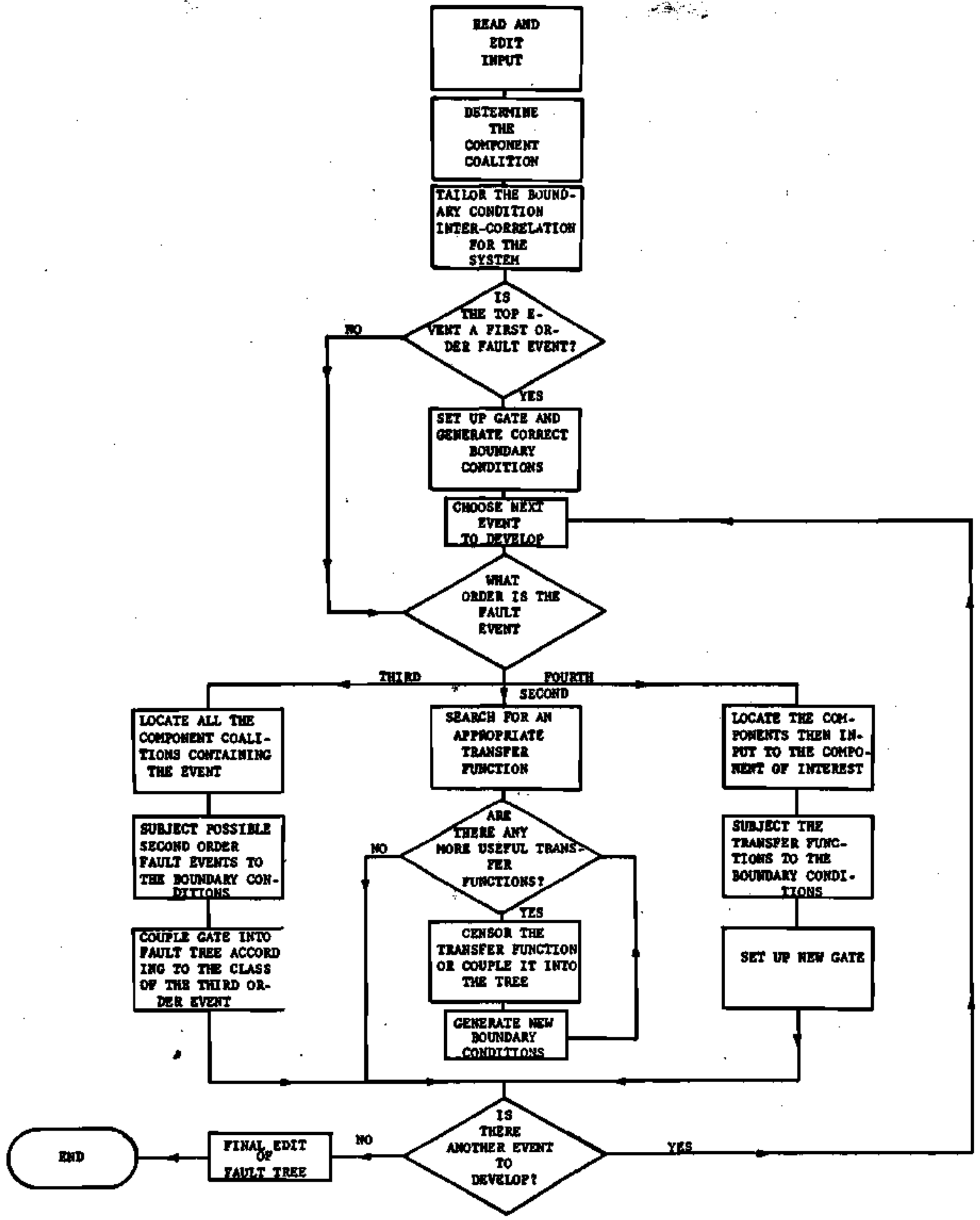

READ AND EDIT INPUT
DETERMINE THE COMPONENT COALITION
TAILOR THE BOUNDARY CONDITION INTER-CORRELATION FOR THE SYSTEM
IS THE TOP EVENT A FIRST ORDER FAULT EVENT?
NO
YES
SET UP GATE AND GENERATE CORRECT BOUNDARY CONDITIONS
CHOOSE NEXT EVENT TO DEVELOP
WHAT ORDER IS THE FAULT EVENT
THIRD
FOURTH
SECOND
LOCATE ALL THE COMPONENT COALITIONS CONTAINING THE EVENT
SUBJECT POSSIBLE SECOND ORDER FAULT EVENTS TO THE BOUNDARY CONDITIONS
COUPLE GATE INTO FAULT TREE ACCORDING TO THE CLASS OF THE THIRD ORDER EVENT
SEARCH FOR AN APPROPRIATE TRANSFER FUNCTION
ARE THERE ANY MORE USEFUL TRANSFER FUNCTIONS?
NO
YES
CENSOR THE TRANSFER FUNCTION OR COUPLE IT INTO THE TREE
GENERATE NEW BOUNDARY CONDITIONS
LOCATE THE COMPONENTS THEN INPUT TO THE COMPONENT OF INTEREST
SUBJECT THE TRANSFER FUNCTIONS TO THE BOUNDARY CONDITIONS
SET UP NEW GATE
IS THERE ANOTHER EVENT TO DEVELOP?
NO
YES
FINAL EDIT OF FAULT TREE
END

Figure 36. Diagram of Procedure of DRAFT

APPENDIX C

EXAMPLE OF FAULT TREE CONSTRUCTION FOR A REACTOR SCRAM SYSTEM USING DRAFT

As a final, and somewhat more involved, example of the DRAFT code, a reactor trip system is chosen. The schematic for this system is shown in Figure 37.

A pressure is sensed by three pressure detectors, components 55, 56, and 57. During normal operation the pressure is below the setpoint and the detector contacts are open. If the pressure exceeds the setpoint, each alarm unit supplies current connections to energize two control relays (for example, detector 55 closes contacts 37 resulting in alarm unit 35 closing contacts 32 that in turn result in relay coils 33 and 34 closing contacts 25 and 13, respectively). Contacts of the relay coils in panels 4, 5, and 6 arranged in two sets (panels 2 and 3) of two-out-of-three logic, energize relay coils 15, 16, 27, and 28. These in turn close contacts 11, 12, 23, and 24, respectively, causing circuit breaker coils 17, 18, 29, and 30 to open contacts 3, 4, 2, and 5, respectively. The power is, thereby, removed from the control rod drive motor and the control rods fall by gravity into the core. The trip action is then complete. The TOP event is the control rod motor operating when it should not. That is, the failure is the control rod motor operating. The initial conditions are the component configurations prior to trip.

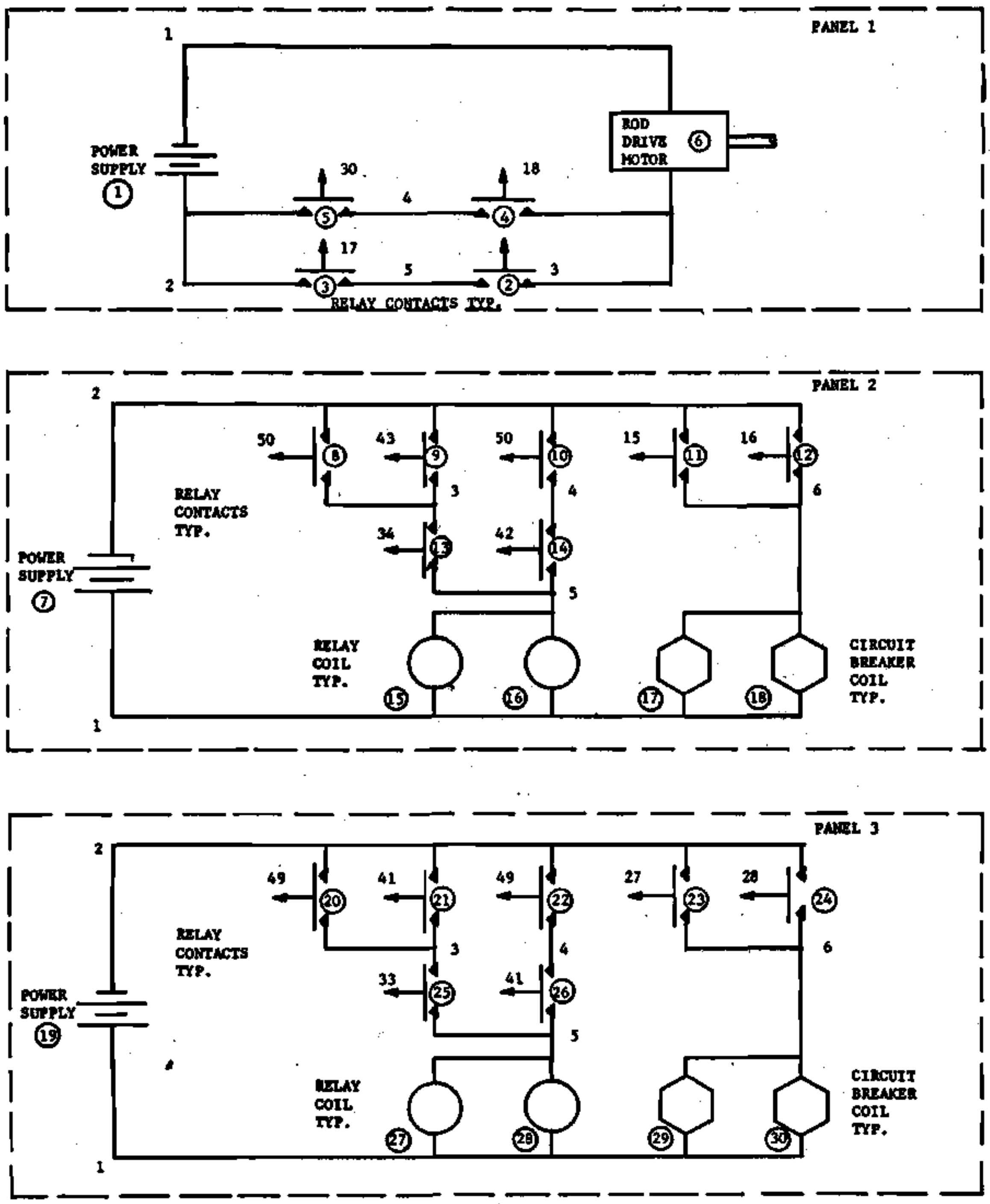


Figure 37. Schematic of Reactor Scram System Example

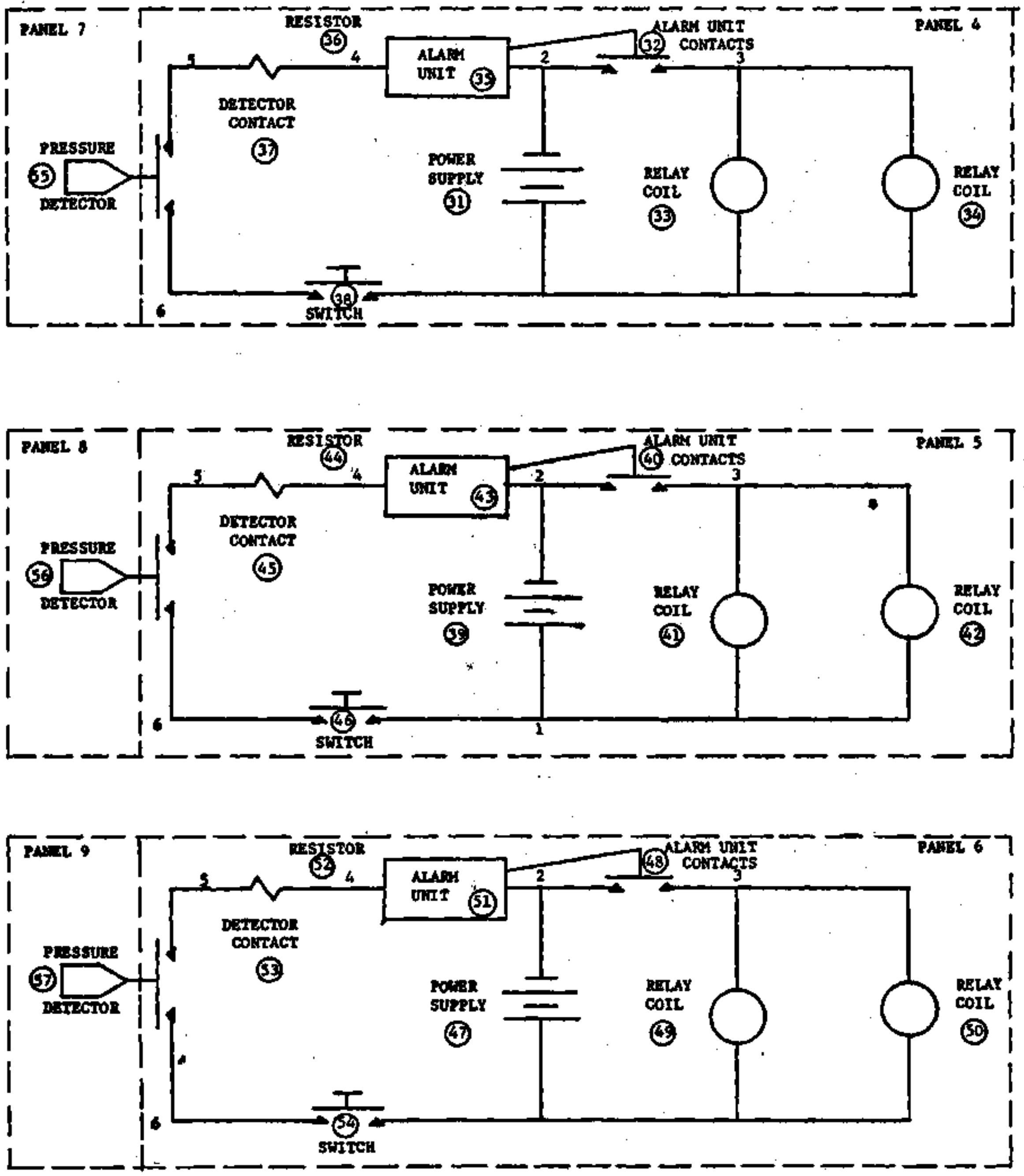


Figure 37. Concluded

It is interesting to note that this explanation of system design intent is not necessary to construct the system fault tree when using DRAFT.

The panel and node layout is also given in Figure 37. The input edit from DRAFT is given in Table 15 and the component coalitions are given in Table 16. The fault tree output from DRAFT is given in Table 17. This output is drawn into a fault tree in Figure 38.

The construction of this fault tree is typical of the largest fault tree DRAFT can construct at present on the UNIVAC 1108 at the Georgia Institute of Technology due to storage requirements. No out of core storage is used, however.

Table 15. DRAFT Input Edit for Reactor Scram System

NUMBER OF PANELS 9

Panel Number	Component Number	Component Type	Node One	Node Two	Input Flag	Contact Flag
1	1	40	1	2	0	0
1	2	50	1	4	29	0
1	3	50	4	3	17	0
1	4	50	1	5	18	0
1	5	50	5	3	30	0
1	6	30	3	2	0	0
2	7	40	1	2	0	0
2	8	50	2	3	50	0
2	9	50	2	3	42	0
2	10	50	2	4	50	0
2	11	50	2	6	15	0
2	12	50	2	6	16	0
2	13	50	3	5	34	0
2	14	50	4	5	42	0
2	15	70	5	1	0	0
2	16	70	5	1	0	0
2	17	60	6	1	0	0
2	18	60	6	1	0	0
3	19	40	1	2	0	0
3	20	50	2	3	49	0
3	21	50	2	3	41	0
3	22	50	2	4	49	0
3	23	50	2	6	27	0
3	24	50	2	6	28	0
3	25	50	3	5	33	0
3	26	50	4	5	41	0
3	27	70	5	1	0	0
3	28	70	5	1	0	0
3	29	60	6	1	0	0
3	30	60	6	1	0	0
4	31	40	1	2	0	0
4	32	50	2	3	35	0
4	33	70	1	3	0	0
4	34	70	1	3	0	0
4	35	70	2	4	0	0
4	36	90	4	5	0	0
4	37	50	5	6	55	0
4	38	50	6	1	0	0

Table 15. Concluded

Panel Number	Component Number	Component Type	Node One	Node Two	Input Flag	Contact Flag
5	39	40	1	2	0	0
5	40	50	2	3	43	0
5	41	70	1	3	0	0
5	42	70	1	3	0	0
5	43	70	2	4	0	0
5	44	90	4	5	0	0
5	45	50	5	6	56	0
5	46	50	6	1	0	0
6	47	40	1	2	0	0
6	48	50	2	3	51	0
6	49	70	1	3	0	0
6	50	70	1	3	0	0
6	51	70	2	4	0	0
6	52	90	4	5	0	0
6	53	50	5	6	57	0
6	54	50	6	1	0	0
7	55	100	1	2	0	0
8	56	100	1	2	0	0
9	57	100	1	2	0	0

Boundary Conditions

TOP Event
1003 6

Initial Conditions

2003	2	2001	11	2001	23
2003	3	2001	12	2001	24
2003	4	2001	13	2001	25
2003	5	2001	14	2001	26
2001	8	2001	20	2001	32
2001	9	2001	21	2001	40
2001	10	2001	22	2001	48

Table 16. Component Coalitions for Reactor Scram System

	Component Coalitions					
Panel Number	Component Coalition Number	Component Number				
1	101	1	6	3	2	
1	102	1	6	5	4	
2	103	7	11	17		
2	104	7	12	17		
2	105	7	8	13	15	
2	106	7	8	13	16	
2	107	7	9	13	15	
2	108	7	9	13	16	
2	109	7	10	14	15	
2	110	7	10	14	16	
2	111	7	11	18		
2	112	7	12	18		
3	113	19	23	29		
3	114	19	24	29		
3	115	19	20	25	27	
3	116	19	20	25	28	
3	117	19	21	25	27	
3	118	19	21	25	28	
3	119	19	22	26	27	
3	120	19	22	26	28	
3	121	19	23	30		
3	122	19	24	30		
4	123	31	32	33		
4	124	31	35	36	37	38
4	125	31	32	34		
5	126	39	40	41		
5	127	39	43	44	45	46
5	128	39	40	42		
6	129	47	48	49		
6	130	47	51	52	53	54
6	131	47	48	50		
7	132	55				
8	133	56				
9	134	57				

Table 17. Fault Tree Output from DRAFT for Reactor Scram System

			Gate Input		
Incident ID	Entity ID	Gate Type	Component Number	Incident ID	Fault Tree Symbol
Gate 1 (1003	6)	OR	Gate 2		
			Gate 3		
Gate 2 (1003	101)	AND	Gate 4		
			Gate 5		
Gate 3 (1003	102)	AND	Gate 6		
			Gate 7		
Gate 4 (2003	3)	OR	Component (3	109)	Circle
			Gate 8		
Gate 5 (2003	2)	OR	Component (2	109)	Circle
			Gate 9		
Gate 6 (2003	5)	OR	Component (5	109)	Circle
			Gate 10		
Gate 7 (2003	4)	OR	Component (4	109)	Circle
			Gate 11		
Gate 8 (3003	3)	OR	Component (17	109)	Circle
			Gate 12		
Gate 9 (3003	2)	OR	Component (29	109)	Circle
			Gate 13		
Gate 10 (3003	5)	OR	Component (30	109)	Circle
			Gate 14		
Gate 11 (3003	4)	OR	Component (18	109)	Circle
			Gate 15		
Gate 12 (1001	17)	AND	Gate 16		
			Gate 17		
Gate 13 (1001	29)	AND	Gate 18		
			Gate 19		
Gate 14 (1001	30)	AND	Gate 20		
			Gate 21		

Table 17. Continued

Incident ID	Entity ID	Gate Type	Gate Input: Component Number	Gate Input: Incident ID	Fault Tree Symbol
Gate 15 (1001	18)	AND	Gate 22		
			Gate 23		
Gate 16 (1001	103)	OR	Component (7	106)	Circle
			Gate 24		
Gate 17 (1001	104)	OR	Component (17	104)	Circle
			Component (7	106)	Circle
			Gate 25		
Gate 18 (1001	113)	OR	Component (17	104)	Circle
			Component (19	106)	Circle
			Gate 26		
Gate 19 (1001	114)	OR	Component (29	104)	
			Component (19	106)	Circle
			Gate 27		
Gate 20 (1001	121)	OR	Component (29	104)	Circle
			Component (19	106)	Circle
			Gate 26		
Gate 21 (1001	122)	OR	Component (30	104)	Circle
			Component (19	106)	Circle
			Gate 27		
Gate 22 (1001	111)	OR	Component (30	104)	Circle
			Component (7	106)	Circle
			Gate 24		
Gate 23 (1001	112)	OR	Component (18	104)	Circle
			Component (7	106)	Circle
			Gate 25		
Gate 24 (2001	11)	OR	Component (18	104)	Circle
			Component (11	108)	Circle
			Gate 28		

Table 17. Continued

Incident ID	Entity ID	Gate Type	Component Number	Gate Input Incident ID	Fault Tree Symbol
****	****	****	****	****	****
Gate 25 (2001	12)	OR	Component (12	108)	Circle
			Gate 29		
Gate 26 (2001	23)	OR	Component (23	108)	Circle
			Gate 30		
Gate 27 (2001	24)	OR	Component (24	108)	Circle
			Gate 31		
Gate 28 (3001	11)	OR	Component (15	108)	Circle
			Gate 32		
Gate 29 (3001	12)	OR	Component (16	108)	Circle
			Gate 33		
Gate 30 (3001	23)	OR	Component (27	108)	Circle
			Gate 34		
Gate 31 (3001	24)	OR	Component (28	108)	Circle
			Gate 35		
Gate 32 (1001	15)	AND	Gate 36		
			Gate 37		
			Gate 38		
Gate 33 (1001	16)	AND	Gate 39		
			Gate 40		
			Gate 41		
Gate 34 (1001	27)	AND	Gate 42		
			Gate 43		
			Gate 44		
Gate 35 (1001	28)	AND	Gate 45		
			Gate 46		
			Gate 47		
Gate 36 (1001	105)	OR	Component (7	106)	Circle
			Gate 48		
			Gate 49		
			Component (15	104)	Circle

Table 17.

Incident ID	Entity ID	Gate Type	Gate Input Component Number	Gate Input Incident ID	Fault Tree Symbol
*****	*****	*****	*****	*****	*****
Gate 37 (1001	107)	OR	Component (7	106)	Circle
			Gate 50		
			Gate 49		
			Component (15	104)	Circle
Gate 38 (1001	109)	OR	Component (7	106)	Circle
			Gate 51		
			Gate 52		
			Component (15	104)	Circle
Gate 39 (1001	106)	OR	Component (7	106)	Circle
			Gate 48		
			Gate 49		
			Component (16	104)	Circle
Gate 40 (1001	110)	OR	Component (7	106)	Circle
			Gate 50		
			Gate 49		
			Component (16	104)	Circle
Gate 41 (1001	110)	OR	Component (7	106)	Circle
			Gate 51		
			Gate 52		
			Component (16	104)	Circle
Gate 42 (1001	115)	OR	Component (19	106)	Circle
			Gate 53		
			Gate 54		
			Component (27	104)	Circle
Gate 43 (1001	117)	OR	Component (19	106)	Circle
			Gate 55		
			Gate 54		
			Component (27	104)	Circle

Table 17. Continued

			Gate Input		
Incident ID	Entity ID	Gate Type	Component Number	Incident ID	Fault Tree Symbol
Gate 44 (1001	119)	OR	Component (19	106)	Circle
			Gate 56		
			Gate 57		
			Component (27	104)	Circle
Gate 45 (1001	116)	OR	Component (19	106)	Circle
			Gate 53		
			Gate 54		
			Component (28	104)	Circle
Gate 46 (1001	118)	OR	Component (19	106)	Circle
			Gate 55		
			Gate 54		
			Component (28	104)	Circle
Gate 47 (1001	120)	OR	Component (19	106)	Circle
			Gate 56		
			Gate 57		
			Component (28	104)	Circle
Gate 48 (2001	8)	OR	Component (8	108)	Circle
			Gate 58		
Gate 49 (2001	13)	OR	Component (13	108)	Circle
			Gate 59		
Gate 50 (2001	9)	OR	Component (9	108)	Circle
			Gate 60		
Gate 51 (2001	10)	OR	Component (10	108)	Circle
			Gate 61		
Gate 52 (2001	14)	OR	Component (14	108)	Circle
			Gate 62		
Gate 53 (2001	20)	OR	Component (20	108)	Circle
			Gate 63		
Gate 54 (2001	25)	OR	Component (25	108)	Circle
			Gate 64		

Table 17. Continued

Incident ID	Entity ID	Gate Type	Gate Input: Component Number	Gate Input: Incident ID	Fault Tree Symbol
Gate 55 (2001	21)	OR	Component (21 Gate 65	108)	Circle
Gate 56 (2001	22)	OR	Component (22 Gate 66	108)	Circle
Gate 57 (2001	26)	OR	Component (26 Gate 67	108)	Circle
Gate 58 (3001	8)	OR	Component (50 Gate 68	108)	Circle
Gate 59 (3001	13)	OR	Component (34 Gate 69	108)	Circle
Gate 60 (3001	9)	OR	Component (42 Gate 70	108)	Circle
Gate 61 (3001	10)	OR	Component (50 Gate 71	108)	Circle
Gate 62 (3001	14)	OR	Component (42 Gate 72	108)	Circle
Gate 63 (3001	20)	OR	Component (33 Gate 73	108)	Circle
Gate 64 (3001	25)	OR	Component (33	108)	Circle
Gate 65 (3001	21)	OR	Component (41 Gate 75	108)	Circle
Gate 66 (3001	22)	OR	Component (49 Gate 76	108)	Circle
Gate 67 (3001	26)	OR	Component (41 Gate 77	108)	Circle
Gate 68 (1001	50)		Gate 78		
Gate 69 (1001	34)		Gate 79		
Gate 70 (1001	42)		Gate 80		

Table 17. Continued

Incident ID******	Entity ID******	Gate Type*****	Gate Input: Component Number*****	Gate Input: Incident ID****	Fault Tree Symbol*****
Gate 71 (1001	50)		Gate 81		
Gate 72 (1001	42)		Gate 82		
Gate 73 (1001	49)		Gate 83		
Gate 74 (1001	33)		Gate 84		
Gate 75 (1001	41)		Gate 85		
Gate 76 (1001	49)		Gate 86		
Gate 77 (1001	41)		Gate 87		
Gate 78 (1001	131)	OR	Component (47	106)	Circle
			Gate 88		
			Component (50	104)	Circle
Gate 79 (1001	125)	OR	Component (31	106)	Circle
			Gate 89		
			Component (34	104)	Circle
Gate 80 (1001	128)	OR	Component (39	106)	Circle
			Gate 90		
			Component (42	104)	Circle
Gate 81 (1001	131)	OR	Component (47	106)	Circle
			Gate 88		
			Component (50	104)	Circle
Gate 82 (1001	128)	OR	Component (39	106)	Circle
			Gate 90		
			Component (42	104)	Circle
Gate 83 (1001	129)	OR	Component (47	106)	Circle
			Gate 88		
			Component (49	104)	Circle
Gate 84 (1001	123)	OR	Component (31	106)	Circle
			Gate 89		
			Component (33	104)	Circle

Table 17. Concluded

Incident ID	Entity ID	Gate Type	Gate Input Component Number	Gate Input Incident ID	Fault Tree Symbol
Gate 98 (1001	124)	OR	Component (31	106)	Circle
			Component (35	104)	Circle
			Component (36	104)	Circle
			Gate 102		
			Gate 103		
Gate 99 (1001	127)	OR	Component (39	106)	Circle
			Component (43	104)	Circle
			Component (44	104)	Circle
			Gate 104		
			Gate 105		
Gate 100(2001	53)	OR	Component (53	108)	Circle
			Gate 106		
Gate 101(2001	54)	OR	Component (54	108)	Circle
			Component (54	3001)	Diamond
Gate 102(2001	37)	OR	Component (37	108)	Circle
			Gate 107		
Gate 103(2001	38)	OR	Component (38	108)	Circle
			Component (38	3001)	Diamond
Gate 104(2001	45)	OR	Component (45	108)	Circle
			Gate 108		
Gate 105(2001	46)	OR	Component (46	108)	Circle
			Component (46	3001)	Diamond
Gate 106(3001	53)	OR	Component (57	108)	Circle
			Component (57	3005)	Diamond
Gate 107(3001	37)	OR	Component (55	108)	Circle
			Component (55	3005)	Diamond
Gate 108(3001	45)	OR	Component (56	108)	Circle
			Component (56	3005)	Diamond

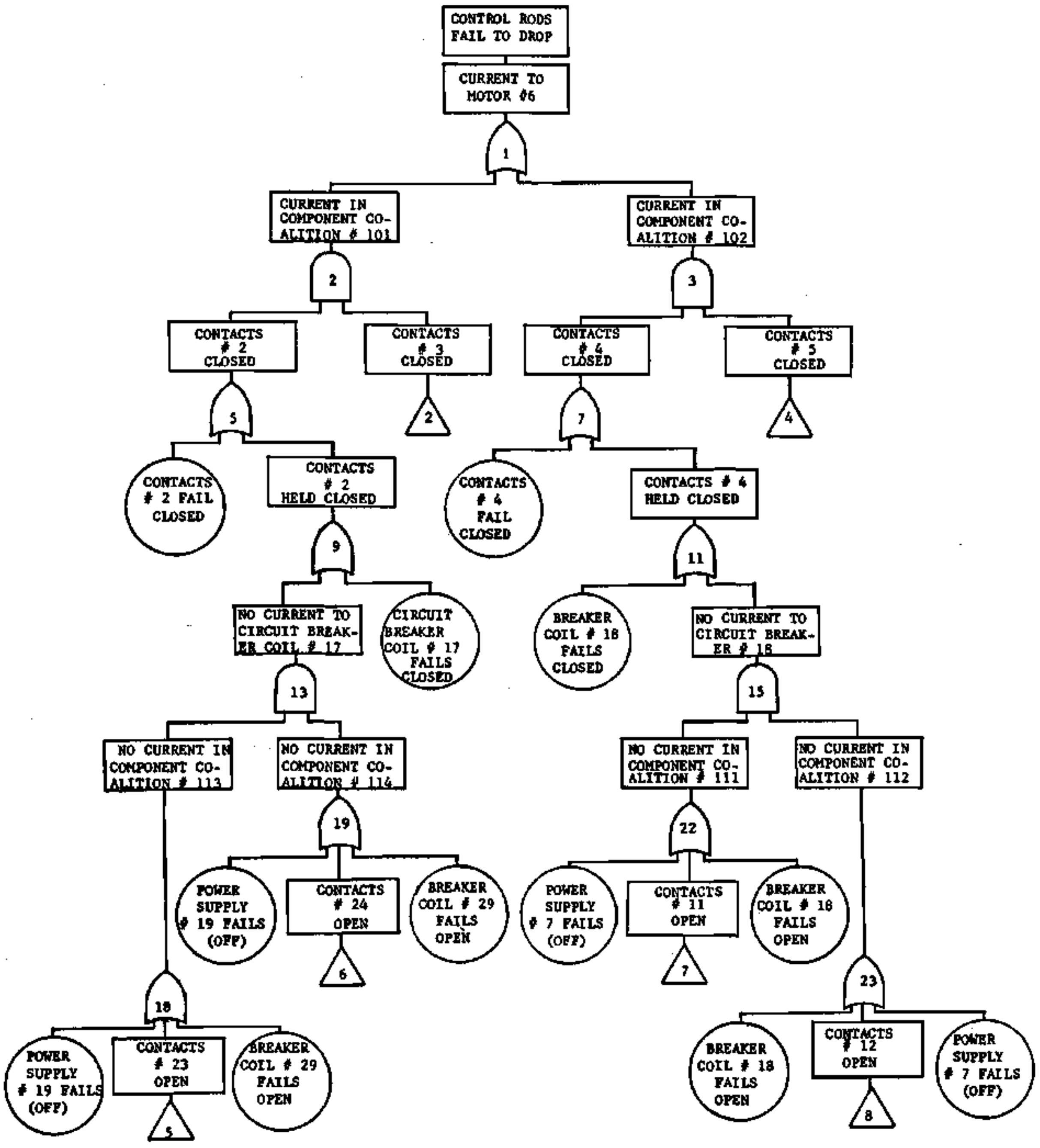

CONTROL RODS FAIL TO DROP
CURRENT TO MOTOR #6
1
CURRENT IN COMPONENT CO-ALITION # 101
CURRENT IN COMPONENT CO-ALITION # 102
2
3
CONTACTS # 2 CLOSED
CONTACTS # 3 CLOSED
CONTACTS # 4 CLOSED
CONTACTS # 5 CLOSED
5
2
7
4
CONTACTS # 2 FAIL CLOSED
CONTACTS # 2 HELD CLOSED
CONTACTS # 4 FAIL CLOSED
CONTACTS # 4 HELD CLOSED
9
11
NO CURRENT TO CIRCUIT BREAK-ER COIL # 17
CIRCUIT BREAKER COIL # 17 FAILS CLOSED
BREAKER COIL # 18 FAILS CLOSED
NO CURRENT TO CIRCUIT BREAK-ER # 18
13
15
NO CURRENT IN COMPONENT CO-ALITION # 113
NO CURRENT IN COMPONENT CO-ALITION # 114
NO CURRENT IN COMPONENT CO-ALITION # 111
NO CURRENT IN COMPONENT CO-ALITION # 112
19
22
POWER SUPPLY # 19 FAILS (OFF)
CONTACTS # 24 OPEN
BREAKER COIL # 29 FAILS OPEN
POWER SUPPLY # 7 FAILS (OFF)
CONTACTS # 11 OPEN
BREAKER COIL # 18 FAILS OPEN
6
7
23
18
POWER SUPPLY # 19 FAILS (OFF)
CONTACTS # 23 OPEN
BREAKER COIL # 29 FAILS OPEN
5
BREAKER COIL # 18 FAILS OPEN
CONTACTS # 12 OPEN
POWER SUPPLY # 7 FAILS (OFF)
8

Figure 38. Fault Tree for Reactor Scram System

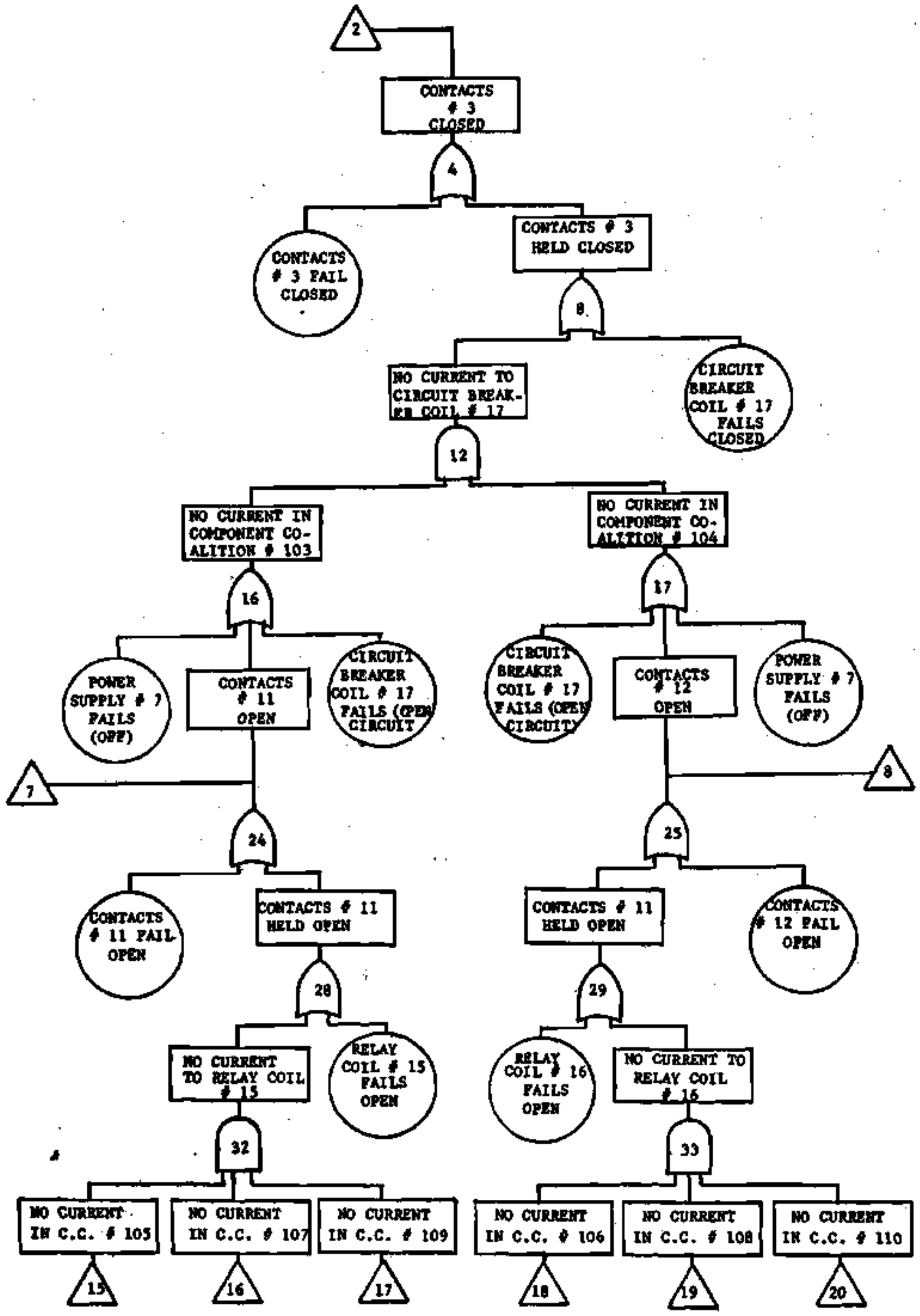

2
CONTACTS # 3 CLOSED
4
CONTACTS # 3 FAIL CLOSED
CONTACTS # 3 HELD CLOSED
8
NO CURRENT TO CIRCUIT BREAKER COIL # 17
CIRCUIT BREAKER COIL # 17 FAILS CLOSED
12
NO CURRENT IN COMPONENT COALITION # 103
NO CURRENT IN COMPONENT COALITION # 104
16
17
POWER SUPPLY # 7 FAILS (OFF)
CONTACTS # 11 OPEN
CIRCUIT BREAKER COIL # 17 FAILS (OPEN CIRCUIT)
CIRCUIT BREAKER COIL # 17 FAILS (OPEN CIRCUIT)
CONTACTS # 12 OPEN
POWER SUPPLY # 7 FAILS (OFF)
7
8
24
25
CONTACTS # 11 FAIL OPEN
CONTACTS # 11 HELD OPEN
CONTACTS # 11 HELD OPEN
CONTACTS # 12 FAIL OPEN
28
29
NO CURRENT TO RELAY COIL # 15
RELAY COIL # 15 FAILS OPEN
RELAY COIL # 16 FAILS OPEN
NO CURRENT TO RELAY COIL # 16
32
33
NO CURRENT IN C.C. # 105
NO CURRENT IN C.C. # 107
NO CURRENT IN C.C. # 109
NO CURRENT IN C.C. # 106
NO CURRENT IN C.C. # 108
NO CURRENT IN C.C. # 110
15
16
17
18
19
20

Figure 38. Continued

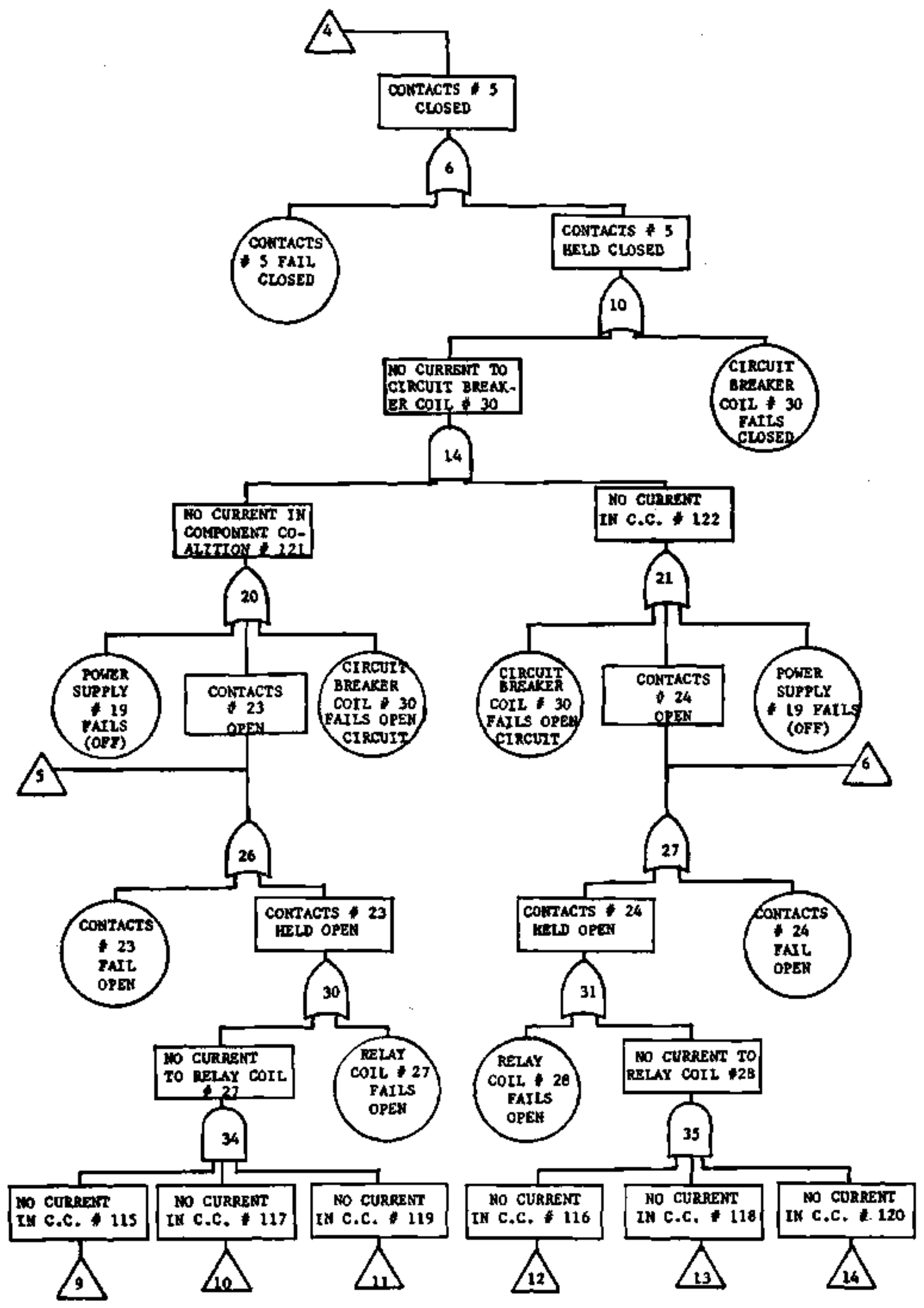


Figure 38. Continued

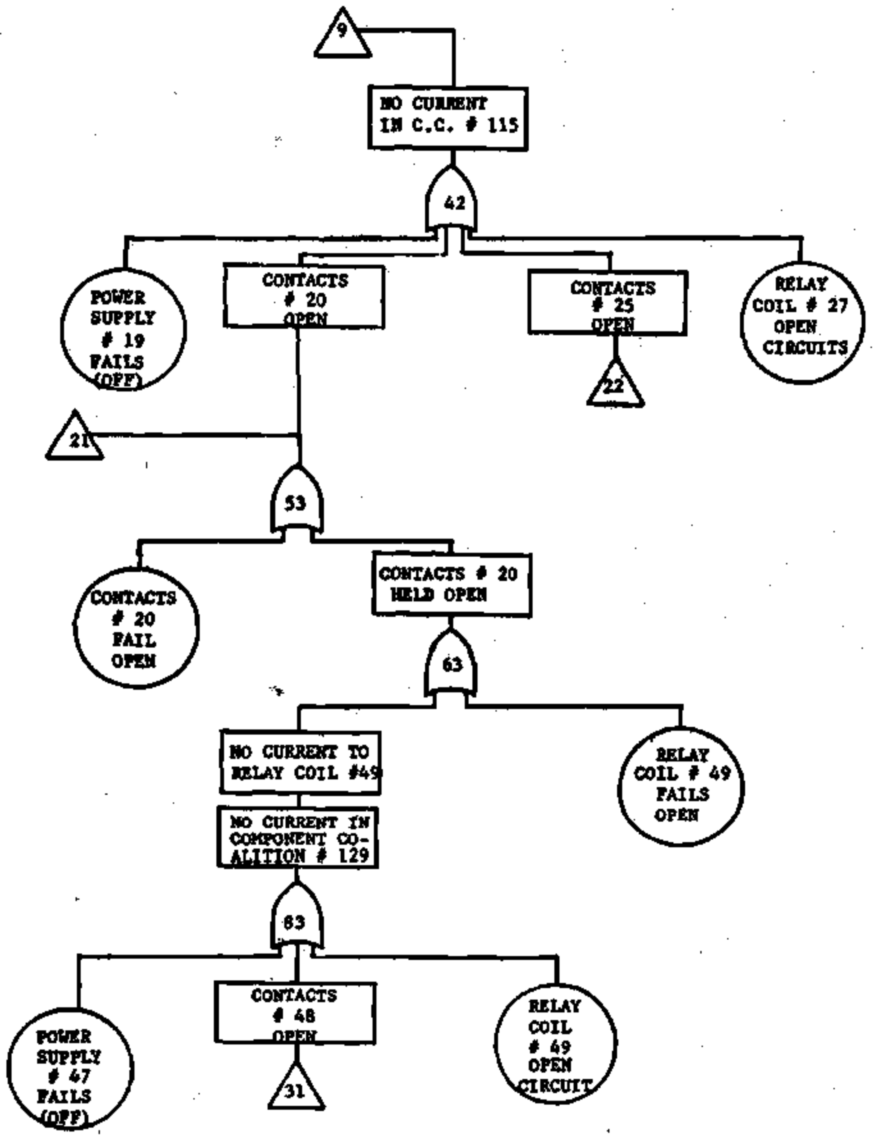

9
NO CURRENT IN C.C. # 115
42
POWER SUPPLY # 19 FAILS (OFF)
CONTACTS # 20 OPEN
CONTACTS # 25 OPEN
22
RELAY COIL # 27 OPEN CIRCUITS
21
53
CONTACTS # 20 FAIL OPEN
CONTACTS # 20 HELD OPEN
63
NO CURRENT TO RELAY COIL #49
RELAY COIL # 49 FAILS OPEN
NO CURRENT IN COMPONENT CO-ALITION # 129
83
POWER SUPPLY # 47 FAILS (OFF)
CONTACTS # 48 OPEN
31
RELAY COIL # 49 OPEN CIRCUIT

Figure 38. Continued

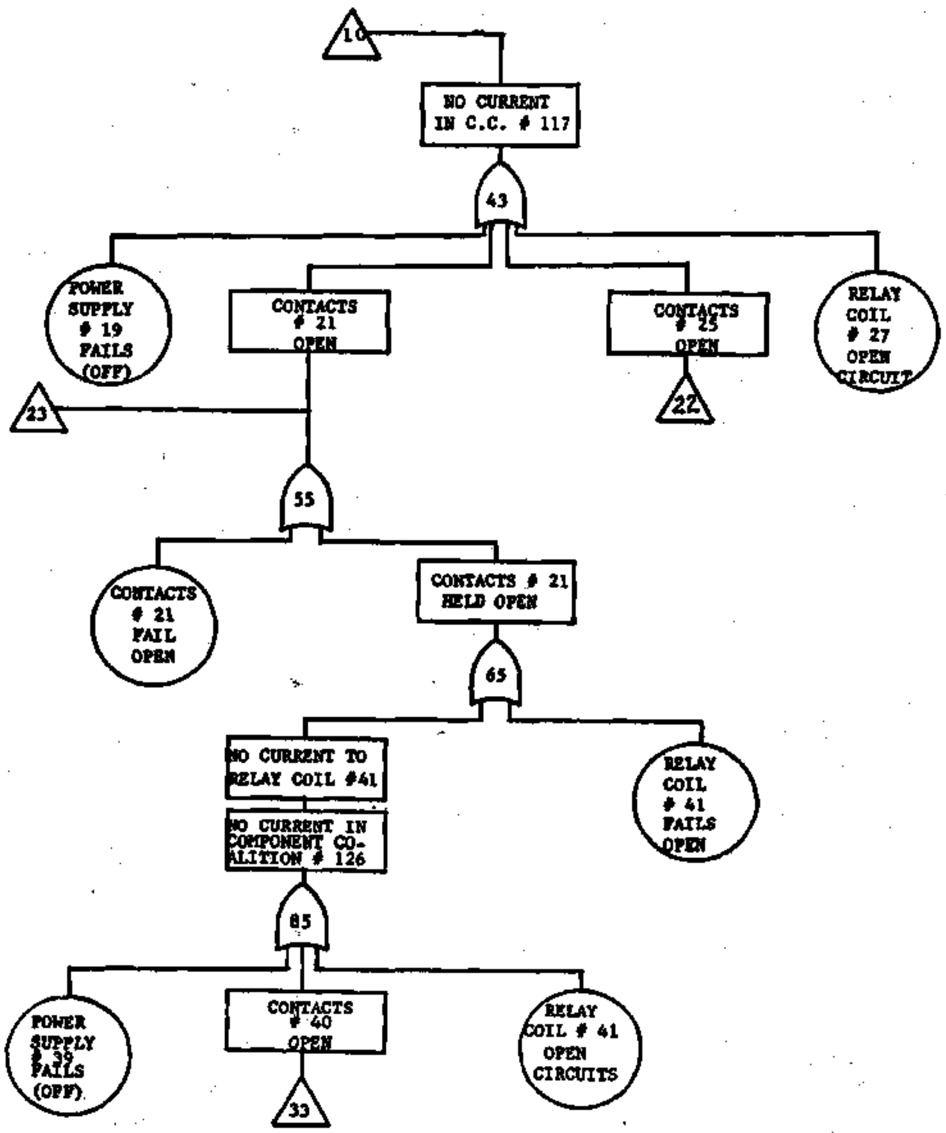

10
NO CURRENT IN C.C. # 117
43
POWER SUPPLY # 19 FAILS (OFF)
CONTACTS # 21 OPEN
CONTACTS # 25 OPEN
RELAY COIL # 27 OPEN CIRCUIT
22
23
55
CONTACTS # 21 FAIL OPEN
CONTACTS # 21 HELD OPEN
65
NO CURRENT TO RELAY COIL #41
RELAY COIL # 41 FAILS OPEN
NO CURRENT IN COMPONENT CO-ALITION # 126
85
POWER SUPPLY # 39 FAILS (OFF)
CONTACTS # 40 OPEN
RELAY COIL # 41 OPEN CIRCUITS
33

Figure 38. Continued

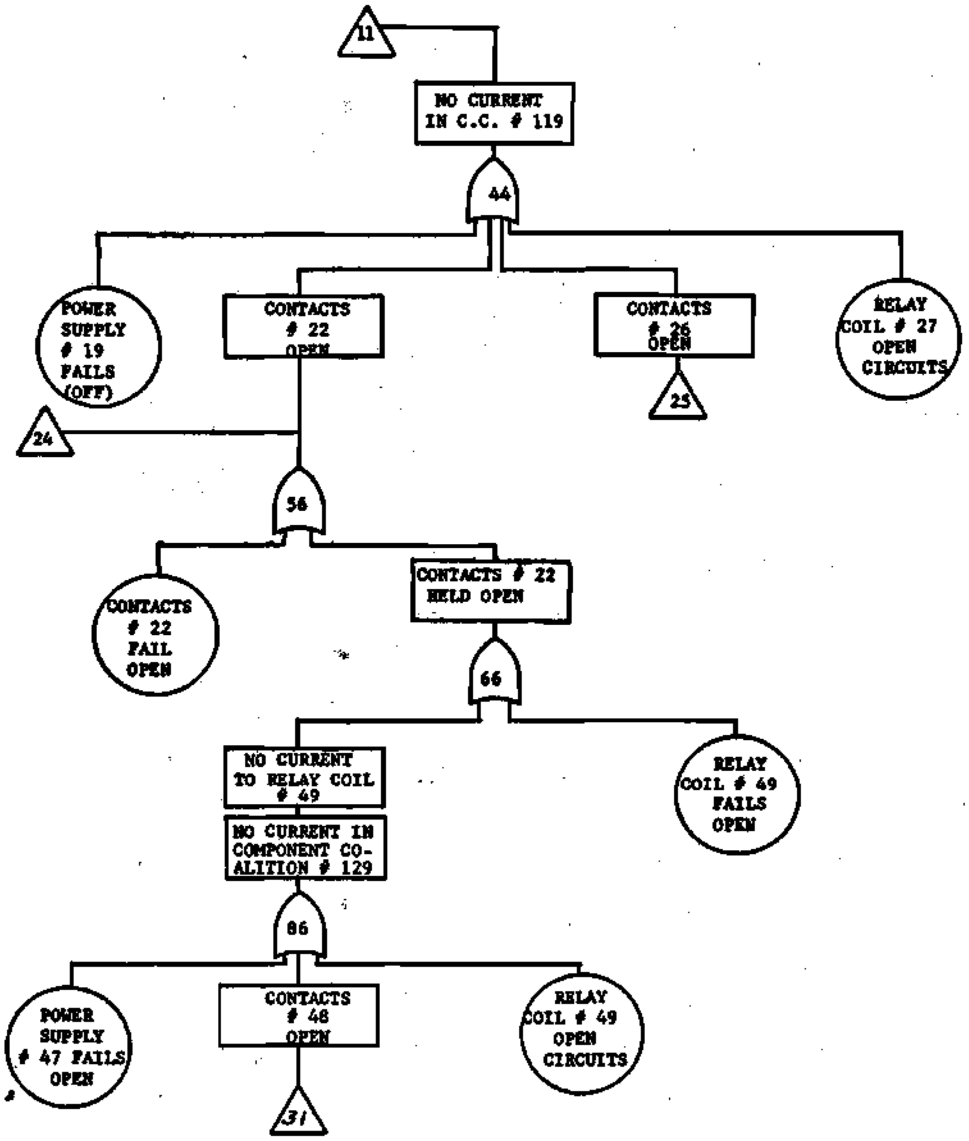

11
NO CURRENT IN C.C. # 119
44
POWER SUPPLY # 19 FAILS (OFF)
CONTACTS # 22 OPEN
CONTACTS # 26 OPEN
RELAY COIL # 27 OPEN CIRCUITS
25
24
56
CONTACTS # 22 FAIL OPEN
CONTACTS # 22 HELD OPEN
66
NO CURRENT TO RELAY COIL # 49
RELAY COIL # 49 FAILS OPEN
NO CURRENT IN COMPONENT CO-ALITION # 129
86
POWER SUPPLY # 47 FAILS OPEN
CONTACTS # 48 OPEN
RELAY COIL # 49 OPEN CIRCUITS
31

Figure 38. Continued

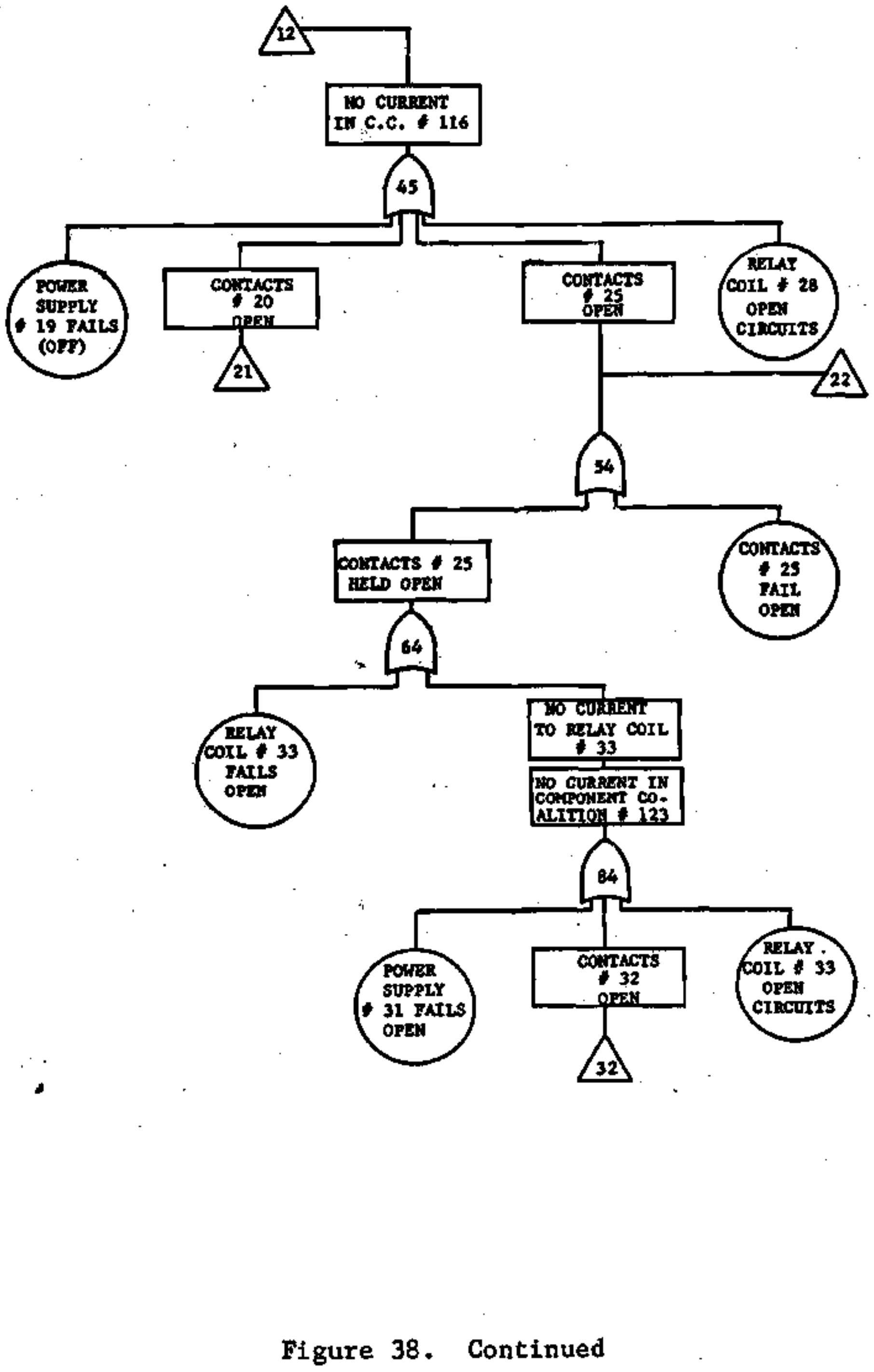
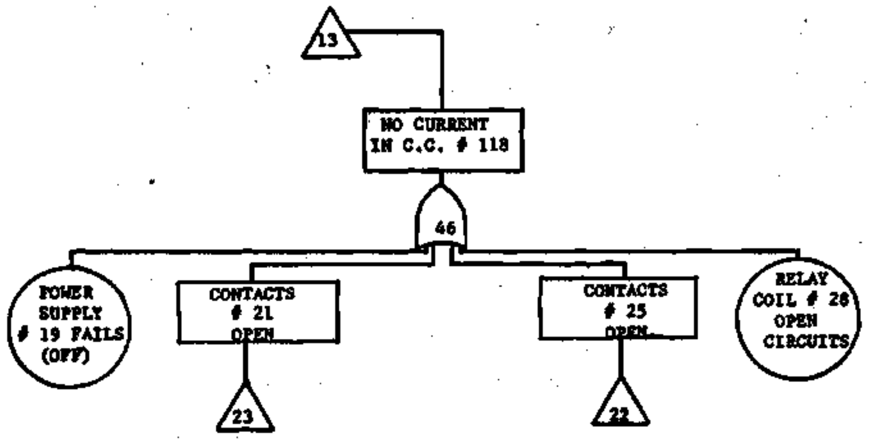

12
NO CURRENT IN C.C. # 116
45
POWER SUPPLY # 19 FAILS (OFF)
CONTACTS # 20 OPEN
21
CONTACTS # 25 OPEN
RELAY COIL # 28 OPEN CIRCUITS
22
54
CONTACTS # 25 HELD OPEN
CONTACTS # 25 FAIL OPEN
64
RELAY COIL # 33 FAILS OPEN
NO CURRENT TO RELAY COIL # 33
NO CURRENT IN COMPONENT CO-ALITION # 123
84
POWER SUPPLY # 31 FAILS OPEN
CONTACTS # 32 OPEN
RELAY COIL # 33 OPEN CIRCUITS
32

Figure 38. Continued



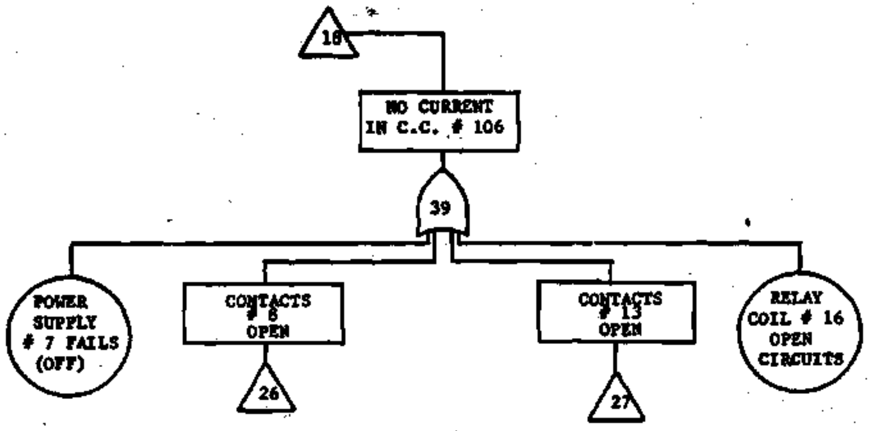


Figure 38. Continued

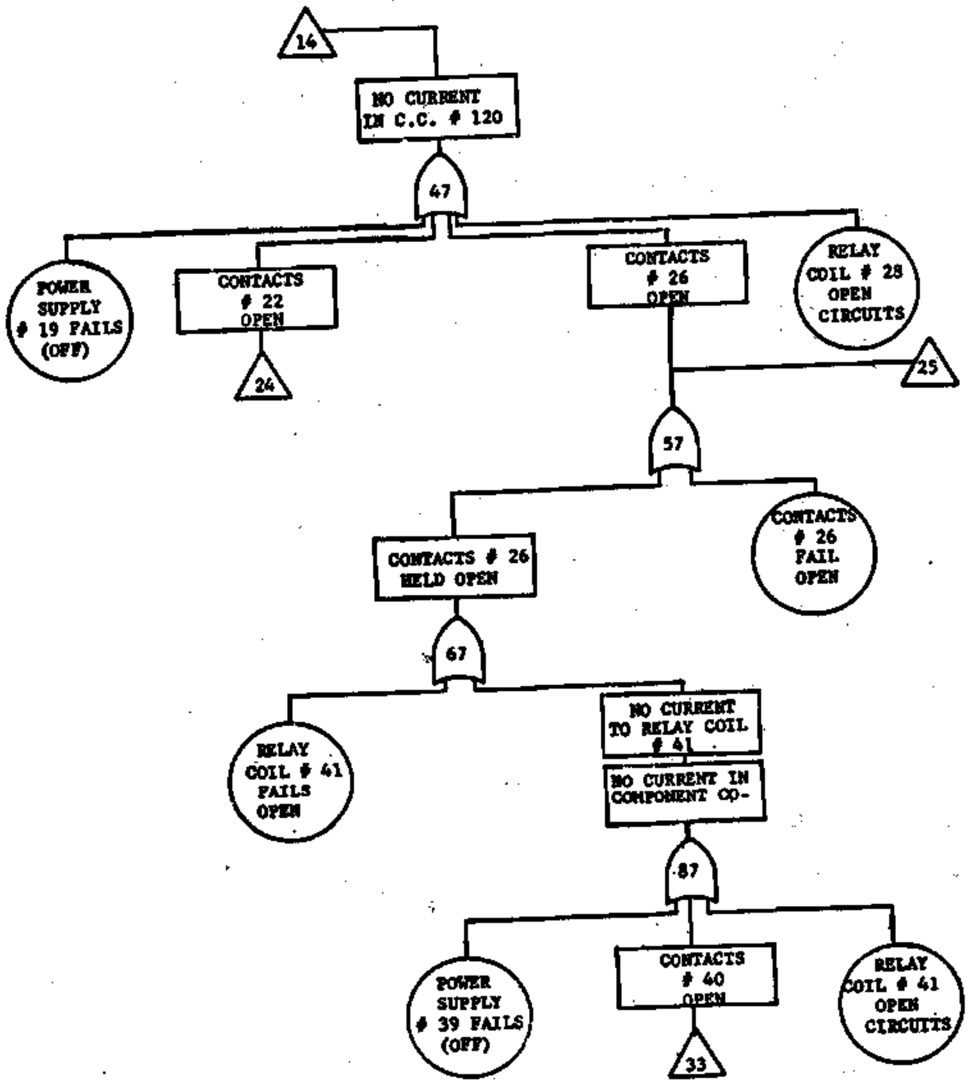

14
NO CURRENT IN C.C. # 120
47
POWER SUPPLY # 19 FAILS (OFF)
CONTACTS # 22 OPEN
24
CONTACTS # 26 OPEN
RELAY COIL # 28 OPEN CIRCUITS
25
57
CONTACTS # 26 HELD OPEN
CONTACTS # 26 FAIL OPEN
67
RELAY COIL # 41 FAILS OPEN
NO CURRENT TO RELAY COIL # 41
NO CURRENT IN COMPONENT CO-
87
POWER SUPPLY # 39 FAILS (OFF)
CONTACTS # 40 OPEN
33
RELAY COIL # 41 OPEN CIRCUITS

Figure 38. Continued

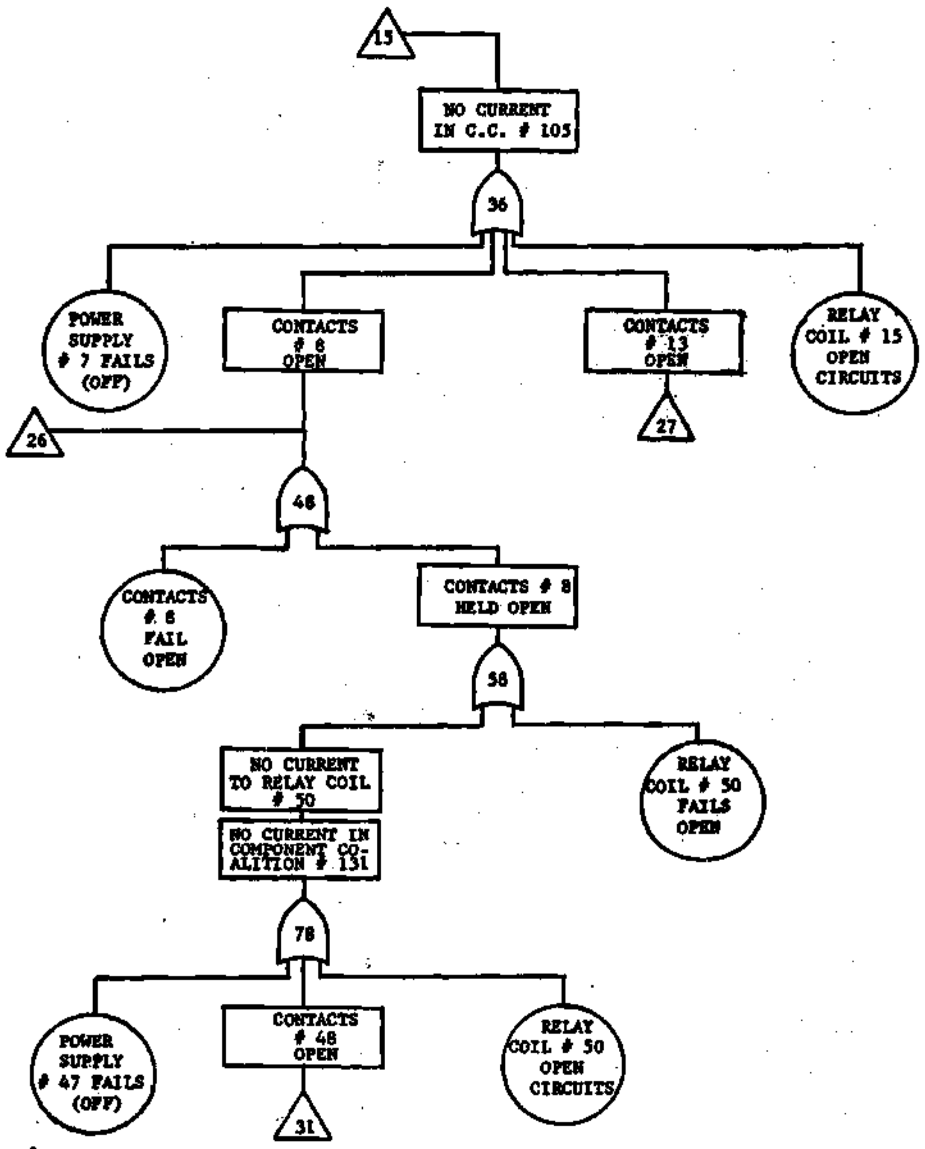

15
NO CURRENT IN C.C. # 105
36
POWER SUPPLY # 7 FAILS (OFF)
CONTACTS # 8 OPEN
CONTACTS # 13 OPEN
RELAY COIL # 15 OPEN CIRCUITS
27
26
46
CONTACTS # 8 FAIL OPEN
CONTACTS # 8 HELD OPEN
58
NO CURRENT TO RELAY COIL # 50
NO CURRENT IN COMPONENT CO-ALITION # 131
RELAY COIL # 50 FAILS OPEN
78
POWER SUPPLY # 47 FAILS (OFF)
CONTACTS # 48 OPEN
31
RELAY COIL # 50 OPEN CIRCUITS

Figure 38. Continued

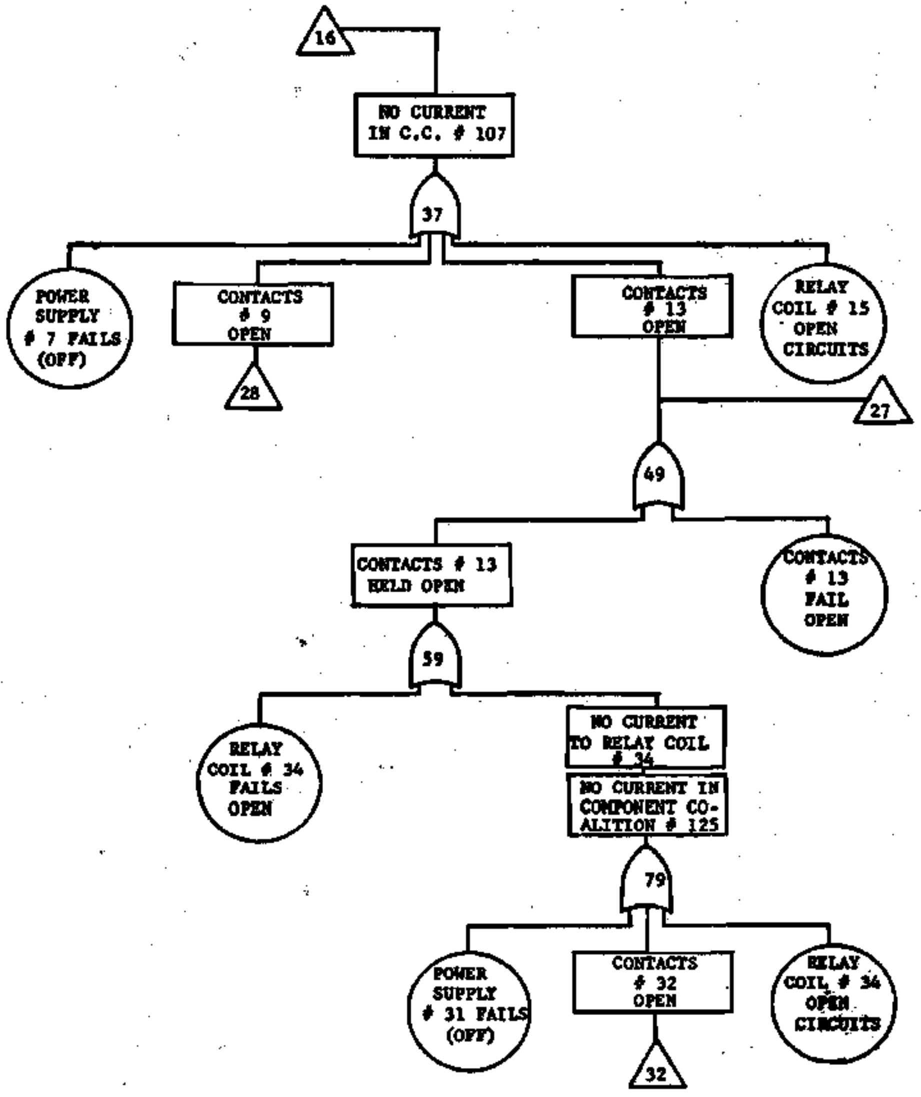

16
NO CURRENT IN C.C. # 107
37
POWER SUPPLY # 7 FAILS (OFF)
CONTACTS # 9 OPEN
28
CONTACTS # 13 OPEN
RELAY COIL # 15 OPEN CIRCUITS
27
49
CONTACTS # 13 HELD OPEN
CONTACTS # 13 FAIL OPEN
59
RELAY COIL # 34 FAILS OPEN
NO CURRENT TO RELAY COIL # 34
NO CURRENT IN COMPONENT CO-ALITION # 125
79
POWER SUPPLY # 31 FAILS (OFF)
CONTACTS # 32 OPEN
32
RELAY COIL # 34 OPEN CIRCUITS

Figure 38. Continued

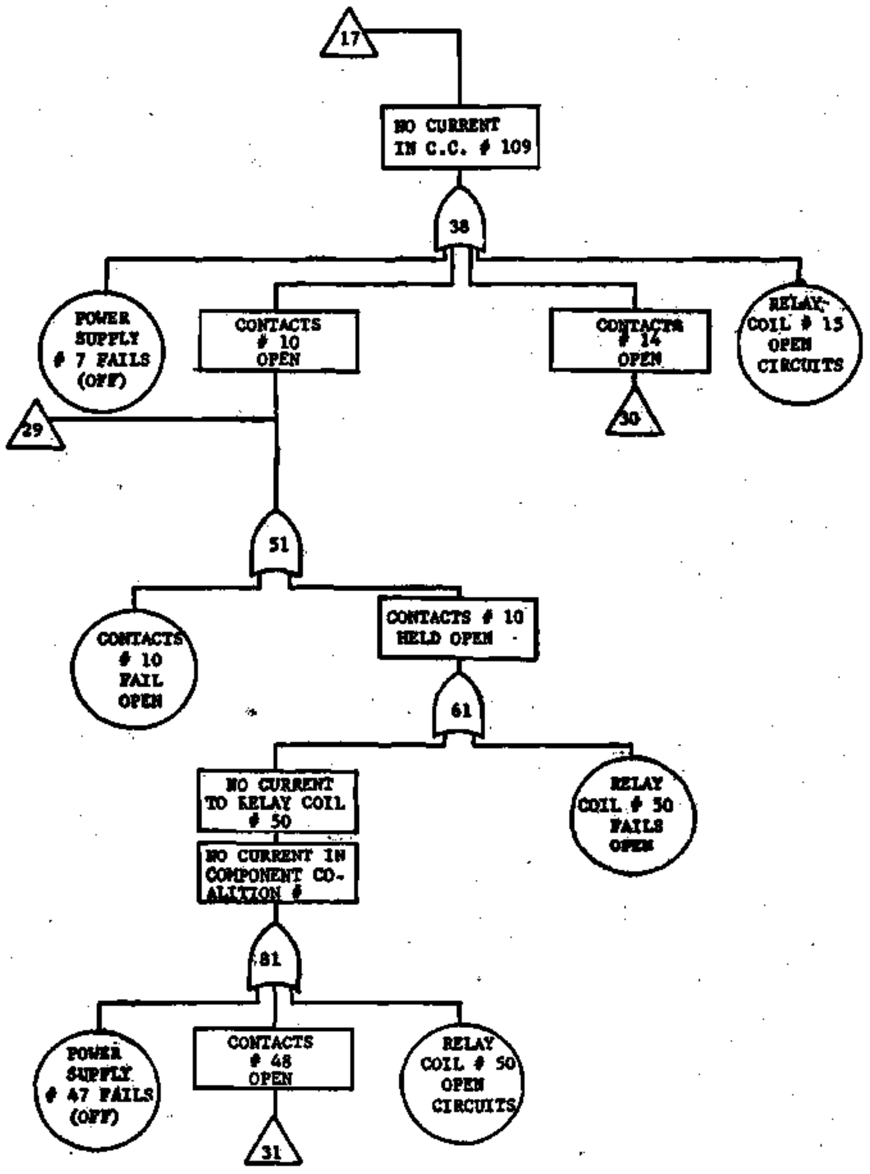

17
NO CURRENT IN C.C. # 109
38
POWER SUPPLY # 7 FAILS (OFF)
CONTACTS # 10 OPEN
CONTACTS # 14 OPEN
RELAY COIL # 15 OPEN CIRCUITS
29
30
51
CONTACTS # 10 FAIL OPEN
CONTACTS # 10 HELD OPEN
61
NO CURRENT TO RELAY COIL # 50
RELAY COIL # 50 FAILS OPEN
NO CURRENT IN COMPONENT CO-ALITION #
81
POWER SUPPLY # 47 FAILS (OFF)
CONTACTS # 48 OPEN
RELAY COIL # 50 OPEN CIRCUITS
31

Figure 38. Continued

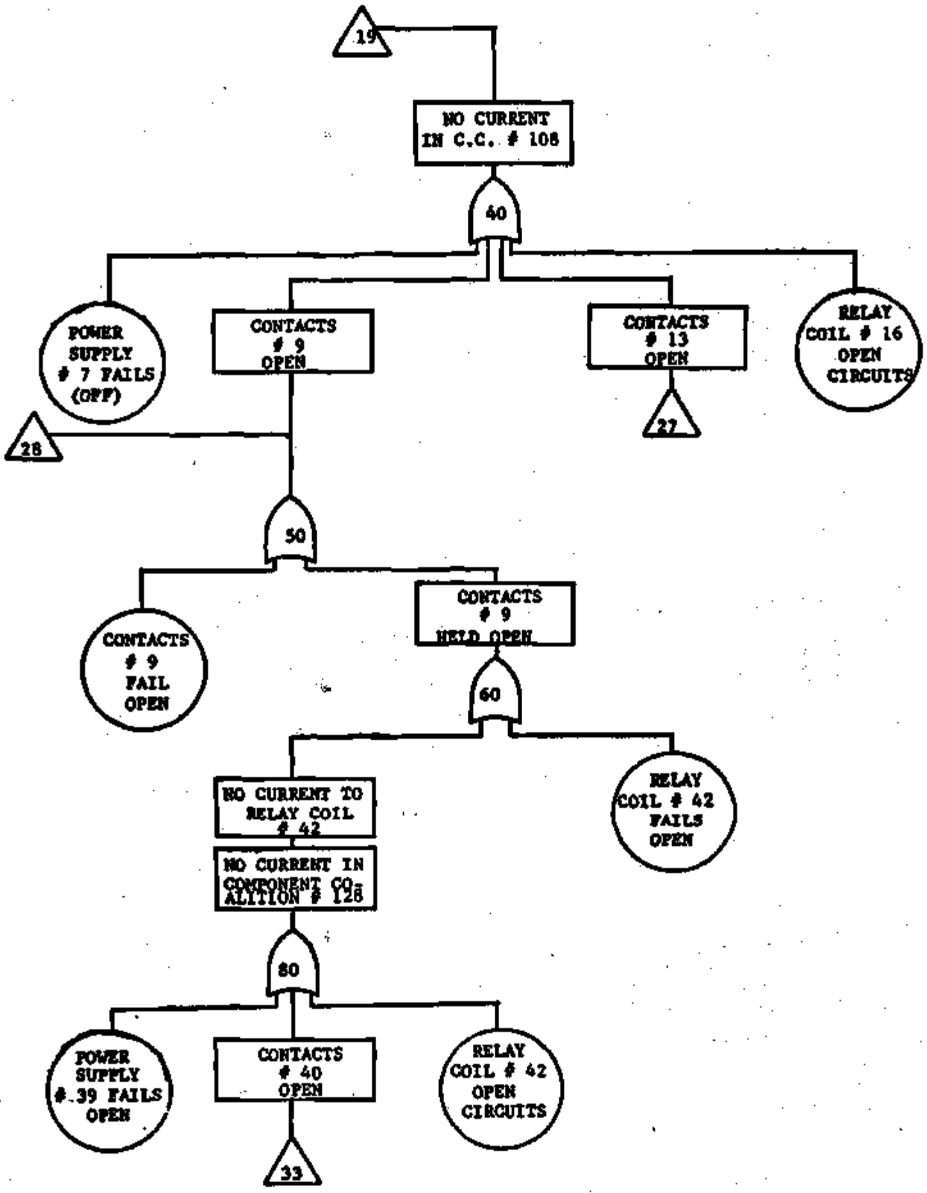

19
NO CURRENT IN C.C. # 108
40
POWER SUPPLY # 7 FAILS (OFF)
CONTACTS # 9 OPEN
CONTACTS # 13 OPEN
RELAY COIL # 16 OPEN CIRCUITS
27
28
50
CONTACTS # 9 FAIL OPEN
CONTACTS # 9 HELD OPEN
60
NO CURRENT TO RELAY COIL # 42
RELAY COIL # 42 FAILS OPEN
NO CURRENT IN COMPONENT CO-ALITION # 128
80
POWER SUPPLY #.39 FAILS OPEN
CONTACTS # 40 OPEN
RELAY COIL # 42 OPEN CIRCUITS
33

Figure 38. Continued

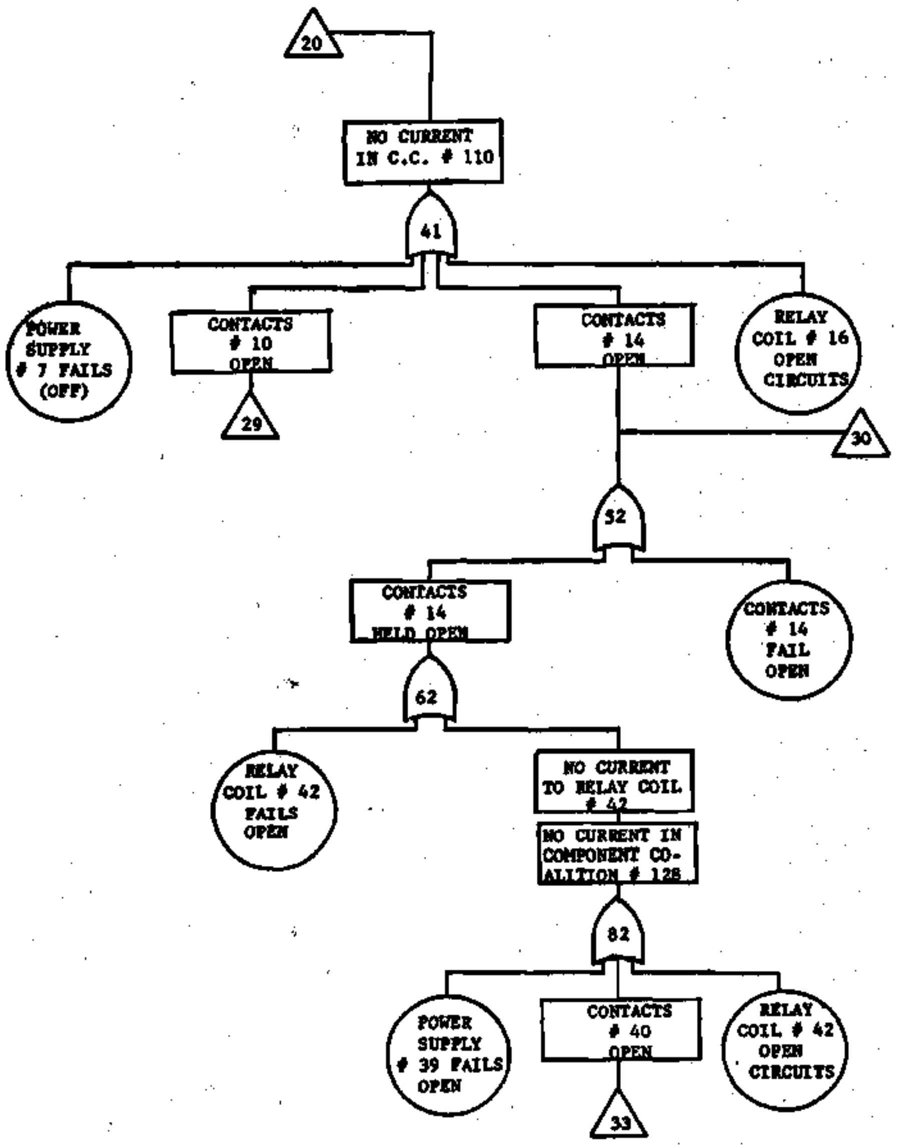

20
NO CURRENT IN C.C. # 110
41
POWER SUPPLY # 7 FAILS (OFF)
CONTACTS # 10 OPEN
29
CONTACTS # 14 OPEN
RELAY COIL # 16 OPEN CIRCUITS
30
52
CONTACTS # 14 HELD OPEN
CONTACTS # 14 FAIL OPEN
62
RELAY COIL # 42 FAILS OPEN
NO CURRENT TO RELAY COIL # 42
NO CURRENT IN COMPONENT CO-ALITION # 128
82
POWER SUPPLY # 39 FAILS OPEN
CONTACTS # 40 OPEN
33
RELAY COIL # 42 OPEN CIRCUITS

Figure 38. Continued

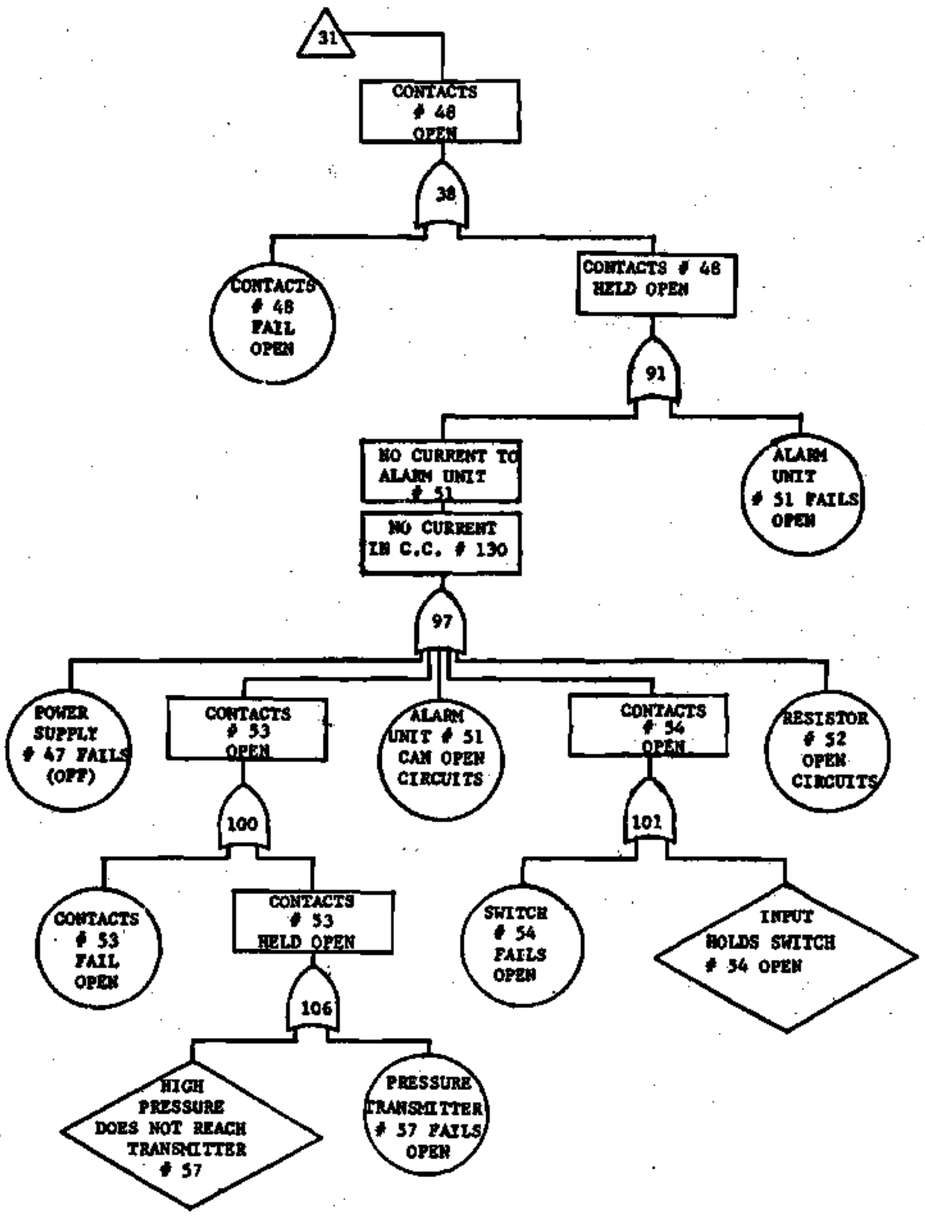
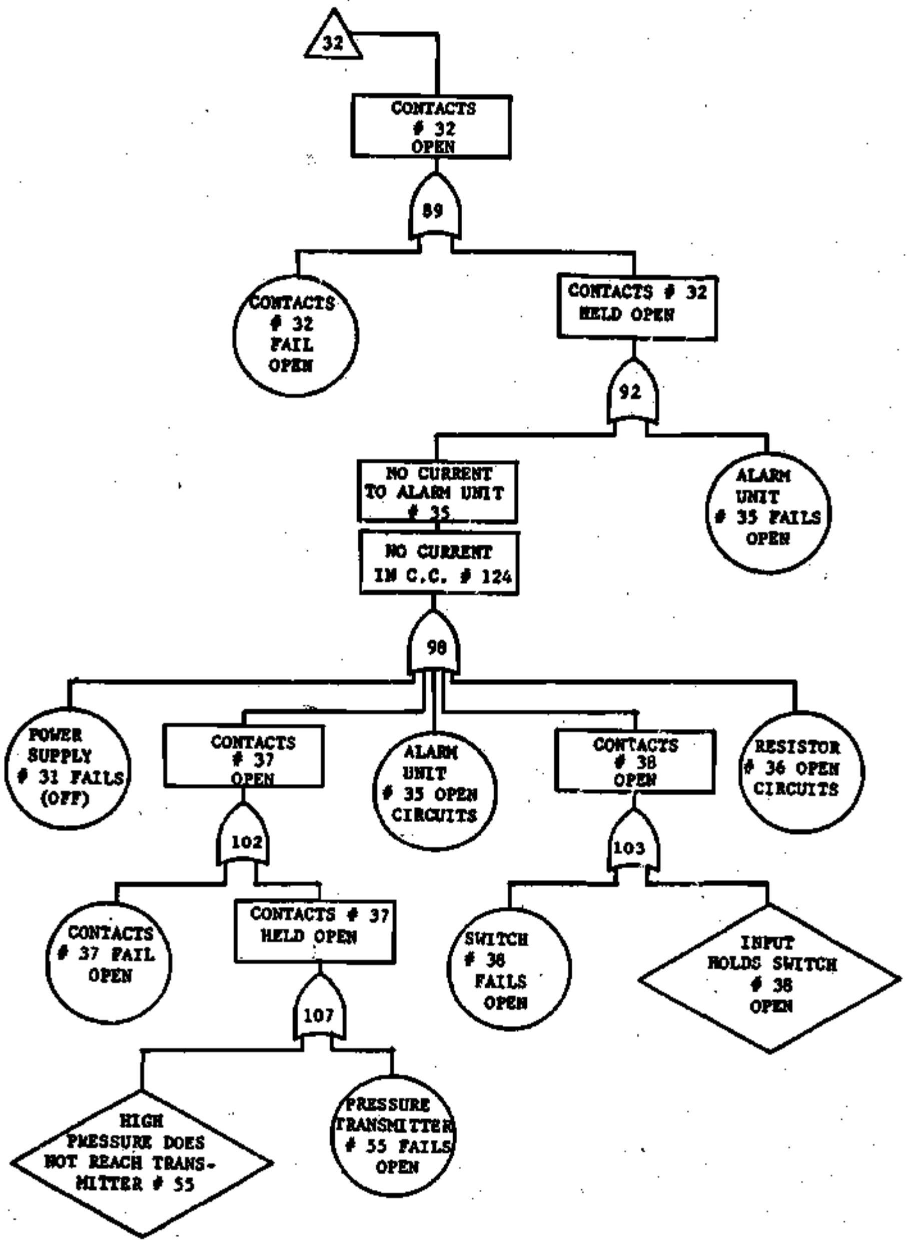

31
CONTACTS # 48 OPEN
38
CONTACTS # 48 FAIL OPEN
CONTACTS # 48 HELD OPEN
91
NO CURRENT TO ALARM UNIT # 51
ALARM UNIT # 51 FAILS OPEN
NO CURRENT IN C.C. # 130
97
POWER SUPPLY # 47 FAILS (OFF)
CONTACTS # 53 OPEN
ALARM UNIT # 51 CAN OPEN CIRCUITS
CONTACTS # 54 OPEN
RESISTOR # 52 OPEN CIRCUITS
100
101
CONTACTS # 53 FAIL OPEN
CONTACTS # 53 HELD OPEN
SWITCH # 54 FAILS OPEN
INPUT HOLDS SWITCH # 54 OPEN
106
HIGH PRESSURE DOES NOT REACH TRANSMITTER # 57
PRESSURE TRANSMITTER # 57 FAILS OPEN

Figure 38. Continued


32
CONTACTS # 32 OPEN
89
CONTACTS # 32 FAIL OPEN
CONTACTS # 32 HELD OPEN
92
NO CURRENT TO ALARM UNIT # 35
ALARM UNIT # 35 FAILS OPEN
NO CURRENT IN C.C. # 124
98
POWER SUPPLY # 31 FAILS (OFF)
CONTACTS # 37 OPEN
ALARM UNIT # 35 OPEN CIRCUITS
CONTACTS # 38 OPEN
RESISTOR # 36 OPEN CIRCUITS
102
103
CONTACTS # 37 FAIL OPEN
CONTACTS # 37 HELD OPEN
SWITCH # 38 FAILS OPEN
INPUT HOLDS SWITCH # 38 OPEN
107
HIGH PRESSURE DOES NOT REACH TRANS-MITTER # 55
PRESSURE TRANSMITTER # 55 FAILS OPEN

Figure

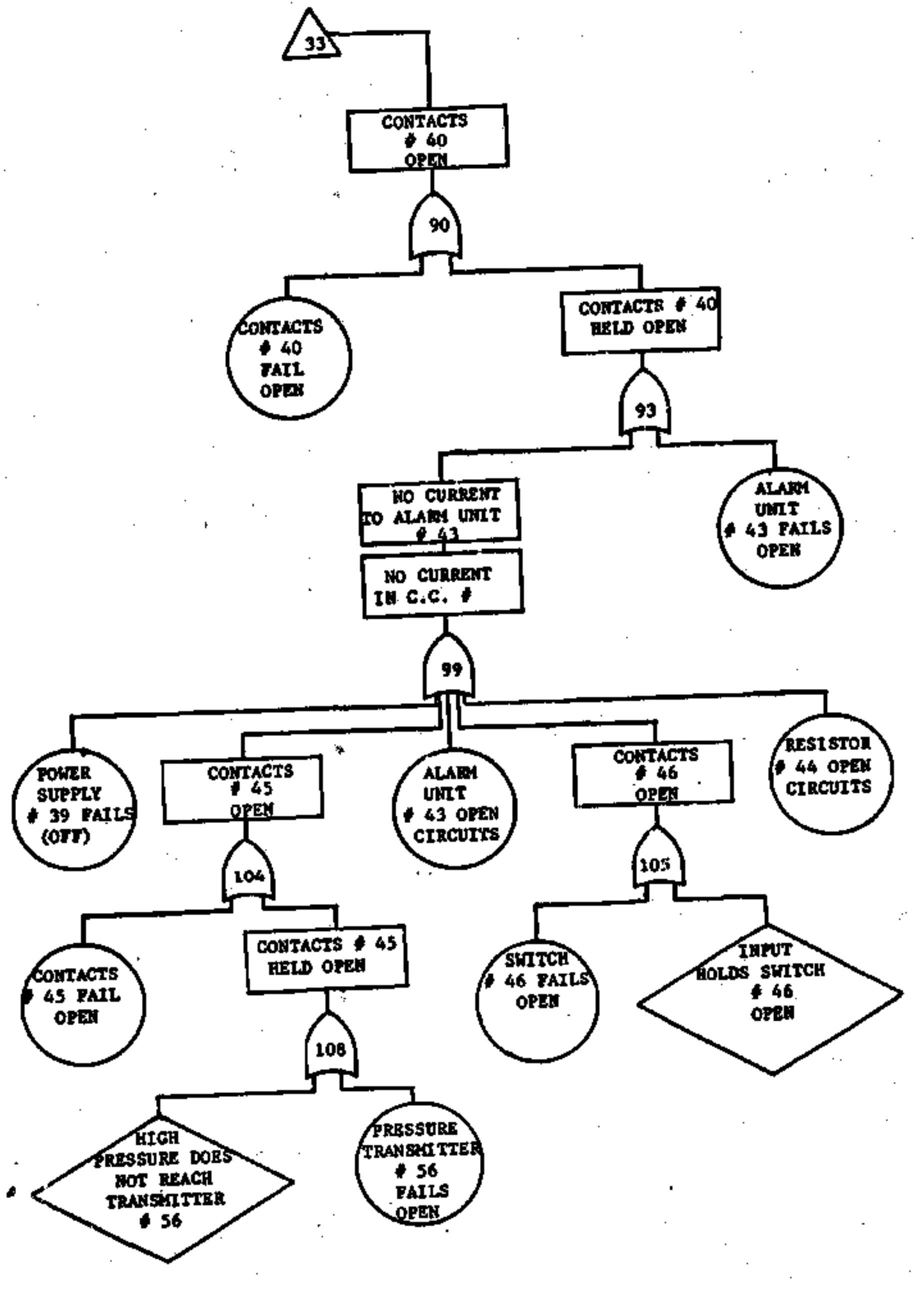

33
CONTACTS # 40 OPEN
90
CONTACTS # 40 FAIL OPEN
CONTACTS # 40 HELD OPEN
93
NO CURRENT TO ALARM UNIT # 43
ALARM UNIT # 43 FAILS OPEN
NO CURRENT IN C.C. #
99
POWER SUPPLY # 39 FAILS (OFF)
CONTACTS # 45 OPEN
ALARM UNIT # 43 OPEN CIRCUITS
CONTACTS # 46 OPEN
RESISTOR # 44 OPEN CIRCUITS
104
105
CONTACTS # 45 FAIL OPEN
CONTACTS # 45 HELD OPEN
SWITCH # 46 FAILS OPEN
INPUT HOLDS SWITCH # 46 OPEN
108
HIGH PRESSURE DOES NOT REACH TRANSMITTER # 56
PRESSURE TRANSMITTER # 56 FAILS OPEN

Figure 38. Concluded

BIBLIOGRAPHY

1. A. M. Polovko, Fundamentals of Reliability Theory, New York, Academic Press, Inc., 1968, p. 12.

2. E. Pieruschka, Principles of Reliability, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1963, p. 75.

3. B. J. Garrick and W. C. Gekler, "Reliability Analysis of Engineered Safeguards," Nuclear Safety, 8 (5), September-October, 1967.

4. K. H. Eagle, "Fault Tree and Reliability Analysis Comparison," Ninth Reliability and Maintainability Conference, Annals of Reliability and Maintainability - 1970.

5. H. W. Von Alven, Reliability Engineering, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1964, p. 200.

6. E. A. Saltarelli and D. G. Fitzgerald, "Reliability Techniques as Applied to Operating Systems and to Design Optimization," Reactor and Fuel Processing Technology, 12 (1), Winter 1968-1969.

7. P. A. Crosetti, "Fault Tree Analysis with Probability Evaluation," IEEE Transactions on Reliability, Seattle, Washington, November, 1970, p. 132.

8. D. F. Haasl, "Advanced Concepts in Fault Tree Analysis," System Safety Symposium, June 8-9, 1965, Seattle: The Boeing Company.

9. "System Safety Symposium," Proceedings sponsored by the University of Washington and the Boeing Company, Seattle, Washington, June 8-9, 1965.

10. S. N. Semanderes, "ELRAFT A Computer Program for the Efficient Logic Reduction Analysis of Fault Trees," IEEE Transactions on Reliability, Seattle, Washington, November, 1970, p. 79.

11. P. Nagel, "Importance Sampling in System Simulation," IEEE Transactions on Reliability, Seattle, Washington, November, 1970, p. [illegible]

12. W. E. Vesely, "Analysis of Fault Trees by Kinetic Tree Theory," [illegible] IN-1330, October, 1969.

13. W. E. Vesely, "A Time-Dependent Methodology for Fault Tre[illegible] tion," Nuclear Engineering and Design, 13 (2), August, [illegible]

BIBLIOGRAPHY (Continued)

14. P. A. Crosetti, "Computer Program for Fault Tree Analysis," DUN-5508, April, 1969.

15. G. E. Greger, D. A. Snyder, and P. D. Gross, "Description and Uses of a Critical Systems Data File for Nuclear Plants," The American Society of Mechanical Engineers, United Engineering Center, 345 East 47th Street, New York, New York 10017, January 9-12, 1972.

16. M. M. Yarosh, "Accident Analysis," Nuclear Safety, 3 (4), 1962.

17. L. Leonardini, "The Third Reliability Meeting at Riso," Nuclear Safety, 11 (4), July-August, 1970.

18. J. R. Penland, Personal Communication, June 1972.

19. R. L. Eisner, "Fault Tree Analysis to Anticipate Potential Failure," The American Society of Mechanical Engineers, United Engineering Center, 345 East 47th Street, New York, New York 10017, May 8-11, 1972.

20. A. B. Mearns, "Fault Tree Analysis: The Study of Unlikely Events in Complex Systems," System Safety Symposium, June 8-9, 1965, Seattle: The Boeing Company.

21. W. E. Vesely, "The Boolean Algebra of a Fault Tree," Unpublished Lecture Notes, 1971.

22. W. E. Vesely, "Fault Tree Algebra," Unpublished Lecture Notes, 1971.

23. P. L. Meyer, Introduction to Probability and Statistical Applications, Second Edition, Addison-Wesley Publishing Company, Reading, Massachusetts, 1970.

24. J. B. Fussell and W. E. Vesely, "A New Methodology for Obtaining Cut Sets for Fault Trees," Transactions of the American Nuclear Society, 15 (1), 1972.

25. P. P. Zemanick, "Failure Mode Analysis to Predict Product Reli- bility," The American Society of Mechanical Engineers, United Engineering Center, 345 East 47th Street, New York, New [illegible] May 8-11, 1972.

26. J. R. Penland, Personal Communication, August 1972.

BIBLIOGRAPHY (Concluded)

27. J. M. Michels, "Computer Evaluation of the Safety Fault Tree Model," System Safety Symposium, June 8-9, 1965, Seattle: The Boeing Company.

28. P. A. Crosetti and R. A. Bruce, "Commercial Application of Fault Tree Analysis," Ninth Reliability and Maintainability Conference, Annals of Reliability and Maintainability - 1970, 9, p. 230.

29. R. Salvatori, "Systematic Approach to Safety Design and Evaluation," IEEE Transactions on Reliability, Seattle, Washington, November, 1970, p. 148.

30. R. J. Schroder, "Fault Trees for Reliability Analysis," Paper Presented at the 1970 Annual Symposium on Reliability, Los Angeles, January, 1970.

31. W. E. Vesely and R. E. Narum, "PREP and KITT: Computer Codes for the Automatic Evaluation of a Fault Tree," IN-1349, August, 1970.

32. W. E. Vesely, "Analysis of Fault Trees by Kinetic Tree Theory," IN-1330, October, 1969.

33. G. H. Sandler, System Reliability Engineering, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1963, pp. 112-132.

VITA

Jerry Bernard Fussell was born on September 13, 1944 in Long Beach, California. He graduated from Dodge County High School in Eastman, Georgia in 1962. In 1964, Mr. Fussell received an Associate in Science degree from Middle Georgia College. He then obtained, in 1967, a Bachelor of Mechanical Engineering degree at the Georgia Institute of Technology. In 1967, Mr. Fussell entered graduate school at the Georgia Institute of Technology and completed his Master of Science in Nuclear Engineering in 1968.

Upon graduation, he worked in the Nuclear Division of Phillips Petroleum Company at the National Reactor Testing Station and was responsible for thermal hydraulic analyses of reactor systems. In 1969, he worked in the Reactor Analysis section of Idaho Nuclear Corporation also at the National Reactor Testing Station.

Mr. Fussell returned to graduate school at the Georgia Institute of Technology in the School of Nuclear Engineering in 1970 where he began investigating the possibility of developing a formal model for fault tree construction as a reliability tool for the nuclear industry and elsewhere.

Mr. Fussell has authored papers in the areas of reliability analysis and reactor physics. He is a member of the American Nuclear Society, Gamma Beta Phi, Phi Theta Kappa, and Pi Tau Sigma.