RELIABILITY ANALYSIS FOR HAZARD AND OPERABILITY STUDIES

By

SAFA YOUSIF RAMADAAN

B.Sc, H.Diploma, M.Sc

Thesis submitted for the degree

of

Doctor of Philosophy

Department of Chemical Engineering

The University of Aston in Birmingham

June 1987

The University of Aston in Birmingham

Reliability Analysis For Hazard And Operability Studies

Safa Yousif Ramadaan PhD 1987

SUMMARY

Fault tree analysis is used as a tool within hazard and operability (Hazop) studies. The present study proposes a new methodology for obtaining the exact TOP event probability of coherent fault trees. The technique uses a top-down approach similar to that of FATRAM. This new Fault Tree Disjoint Reduction Algorithm resolves all the intermediate events in the tree except OR gates with basic event inputs so that a near minimal cut sets expression is obtained. Then Bennetts' disjoint technique is applied and remaining OR gates are resolved. The technique has been found to be appropriate as an alternative to Monte Carlo simulation methods when rare events are encountered and exact results are needed.

The algorithm has been developed in FORTRAN 77 on the Perq workstation as an addition to the Aston Hazop package. The Perq graphical environment enabled a friendly user interface to be created. The total package takes as its input cause and symptom equations using Lihou's form of coding and produces both drawings of fault trees and the Boolean sum of products expression into which reliability data can be substituted directly.

Key words : Hazop, Fault Tree, Boolean Algebra, Reliability

To my wife Tagried and my son Ihab for love, sacrifice, patience and for making my life worthwhile.

Acknowledgments

I am indebted to my supervisor Dr. A.P.H. Jordan for his close supervision, keen interest, patience and encouragement during the performance of this research and writing up the thesis. I am thankful to the departmental electrical staff in particular Mr. M. Lea and Mr. D. Bleby for providing the BREAK box and maintenance for the Perq workstation. My gratitude is also extended to Dr. B. Gay, Head of the Computer Science Department, Dr. J. P. Fletcher and Dr. M.C. Jones for their friendliness and help.

I am grateful to the University of Baghdad for granting me study leave. I am much appreciate the help given to me by Mrs. T.M. Vygus. I would like to thank my colleagues in the computation research group for their comradeship.

Finally I would like to express my indebtedness to my father and my mother for encouraging their children into high academic persuits and to my wife for love, understanding and encouragement especially when our son was born.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

1. INTRODUCTION

Since the industrial revolution, the presence of the potentially destructive physical energies of pressure, heat and motion, and the way in which they may be activated, have been known. But catastrophic accidents continue to happen. From the second world war to the 1960's the industry boomed and more new processes, especially in the field of chemical, petrochemical and petroleum industries, were established. In these processes a number of factors have changed. Process operating conditions such as pressure and temperature have become more severe. The energy stored in the process has increased and represents a greater hazard. Problems in area such as materials of construction and process control are more taxing. At the same time plants have grown in size, typically by a factor of about 10 (1), but are often single-stream. The operation of such plants is relatively difficult. Also the start up and shut down of a large, single-stream plant in an integrated site is much more complex and expensive. These factors have resulted in an increased potential for loss both in human and economic terms. Such loss may occur in various ways. The most obvious is the major accident, frequently taking the form of serious fire, explosion, toxic release or even radioactive releases.

The evolution of automatic control over the years has been in the direction of replacing the human operator by automatic equipment. As the degree of automation has increased, there has emerged a number of functions which have proved rather difficult to automate and which can often be shown to be well suited to execution by the human operator. In more recent years the emphasis has therefore shifted somewhat from complete automation to division

of labour between automatic equipment and man. The sophistication of a system is no longer judged simply by the degree of automation but also by the extent to which it achieves a proper balance between the two.

The application of reliability engineering to the design and operation plant in general and of control systems in particular has led inevitably to a requirement for methods of assessing the reliability of the process operator. This demand is strengthened by the fact that analyses of accident causes show a large proportion to be due to human failures. The reporting of errors in man-machine systems is often deficient, because reporting systems are frequently designed essentially to give information on equipment failure (2,3,4).Work both on the features which cause human error and on the methods of assessing it is a well established aspect of human factors.

Operator selection, training and operating manuals play a great deal in the reduction of the human error factor and help him in decision making. Training may be assisted by the use of a simulator, for example, in flight training, where a subject easily forgets he is in a simulator and not a real aircraft. In the process industry an operator faces a similar situation as the pilot in a plane. Good training, good operating documentation together with a good visual assessment will help to avoiding waste of time in the prevention of accidents (5). The normal approach is to break the task down into its constituent elements, to estimate the reliability of execution of these elements and then to estimate the reliability of the task, using simple reliability relations such as the product law of reliability for series systems or reliability tree diagrams (6). The operator also requires some knowledge of the plant equipment and the instrumentation. In particular, he needs to be able to identify items and to carry out the manipulations

for which he is responsible. There are numerous operating procedures with which he has to become familiar. These include start up, shutdown, batch operation and other sequential routines. So it is the attempt to qualify and quantify risks and the need to have qualitative and quantitative design objectives which have led engineers to try to develop risk criteria and hazard prevention (7).

Hazard and operability studies (Hazop) are used to enhance reliability engineering and design. Studies are used to pinpoint potential malfunctions of a process and to evaluate potential hazards in order to design safer and more reliable plants. Another aim of operability studies is to help in the specification of the required instrumentation and improve the alarm analysis of a particular process (8).

A number of techniques for identifying hazards are used. The most suitable frame work for the analysis of a potential hazard is a fault tree. Fault tree analysis can be carried out as a part of the hazard identification techniques which are known as hazard and operability studies.The hazop study is normally based on a word model, flow sheet diagram or plant layout to be examined. The study should be carried out by a team of experts to provide the knowledge and experience appropriate to the objectives of the examination and to the stage of development of the project. The study takes the form of questions on every part of the process in detail. The team should use its combined experience and imagination to pinpoint the deviations, causes and consequences that may lead to an undesirable event. In large part the technique of hazard and operability studies represents a well-developed form of failure modes and effects analysis. Failure modes and effects analysis involves reviewing systems to discover the mode of failure which may occur and the causes and effects of such failures. It represents a bottom up

approach in contrast with the fault tree where the approach is top down (9).

A fault tree analysis may be used as either a qualitative or a quantitative technique. Qualitatively it shows the links between deviants and the consequences that lead to the occurrence of a particular event. Quantitatively a reliability study may be carried out using the fault tree to calculate the probability of occurrence of a particular event from the basic events at the bottom of the tree. In recent years a lot of work has been done in manipulating fault trees and in using different techniques of analysis. The fault tree analysis technique can be used to investigate system events before they occur. It can be used in cost/benefit studies to compare the level of protection afforded by various design approaches. It can provide a model to study the effects of maintenance activities, operating procedures and in-service testing on system faults (10). Alarm analysis, the design of the layout of control panels and fault tree manipulation can be combined to assist the operator in understanding the cause of an alarm. Further, the operator can use the fault tree to identify likely next events and appropriate causes of action. Studies have found that the operator response to a diagrammatical representation is much better than that for a list of figures on a sheet or a VDU (11,12). Most modern control rooms consist of : 1) A process layout showing all the necessary control instrumentation; 2) Alarm lights and buttons designed to lead the operator to the appropriate next action; 3) A terminal permitting dialogue with the process control computer. The interpretation of an alarm signal can be shown as a fault tree display as a request for action or further analysis (13).

The objective of the present research is to develop the use of fault tree analysis by generating fault trees on a VDU using and modifying the existing Aston

Hazop Package. The analysis is to be enhanced to permit the assessment of event probabilities. Further the work aims to produce a user friendly hazop package for use in operator training and operator decision making.

CHAPTER TWO

2.HAZOP AND FAULT TREE ANALYSIS

2.1 Hazard And Operability Studies (Hazop) :

2.1.1 Introduction :

The major hazards with which the chemical industry is concerned are fire, explosion and toxic release. The problem of avoiding major hazards is extremely important in terms of safety and economy. The first objective of hazard identification is to reveal the substances or processes which have potential for hazard. The second objective is to identify all conceivable threats to the installation or its processes which might lead to loss of containment. The new technology and complexity of chemical processes make identification of hazards difficult: traditional visual inspection is insufficient. It has become necessary, therefore, to develop additional methods for hazard identification (14). Table 2.1 (15) shows some of these methods. Each method has a definite purpose and basis for use. The stage of the project will determine which method is most appropriate for use. For example, at the design stage, hazard and operability (hazop) studies are the more convenient type of analysis to be carried out.

An operability study is defined simply as a structured technique for identifying potential malfunction beforehand (16). The study is aimed at detecting all possible ways in which plants can depart from the intentions of their designers. It is based upon the supposition that most problems are missed because the system is complex rather than because of a lack of knowledge on the part of the design team (17). The technique is based on the critical examination approach.

Table 2.1 : Some methods of hazard identification (15)

Project stage	Hazard identification method
----------	Management and safety audits
All stages	1. Checklists 2. Feedback from workforce
Research and development	1. Screening and testing for, a. Chemical (toxiticy, instability, explosibility) b. Reactions (explosibility) c. Impurities
Predesign	1. Hazard indices 2. Insurance assessments 3. Hazard studies (coarse scale)
Design	1. Process design checks a. Unit processes b. Unit operations c. Plant equipment 2. Hazard and operability studies (fine scale) 3. Failure modes and effects analysis 4. Fault trees and event trees 5. Hazard analysis 6. Reliability assessments 7. Operator task analysis and operating instructions
Commissioning	1. Checks against design, inspection, examination, testing 2. Non-destructive testing, condition monitoring 3. Plant safety audits 4. Emergency planning
Operation	1. Inspection, testing 2. Non-destructive testing, condition monitoring 3. Plant safety audits

2.1.2 Critical Examination Approach :

Essentially, the critical examination approach takes a full description of the process and seeks to answer definite questions in a systematic manner. Each part of the design will be subjected to a number of questions to explore every conceivable way in which that design could deviate from the design intention. This usually produces a number of theoretical deviations. Each deviation is then considered in order to determine whether it is a meaningful or unrealistic one. If the deviation is unrealistic then the derived consequences will be rejected (18). Some of the consequences may be trivial and would be considered no further. The success or failure of the examination will be built on the following aspects :

i) The accuracy of drawings and other data used as the basis for the study.

ii) The technical skills and insights of the study team.

iii) The ability of the study team to use the approach as an aid to their imagination in visualising deviations, causes and consequences.

iv) The ability of the study team to maintain a sense of proportion, particularly when assessing the seriousness of the hazards which are identified.

However, the purpose of the examination is to identify all possible deviations from the way the design is expected to work and to reveal all the hazards or the potential hazards associated with these deviations. The level of detail and the depth of the study are determined by the objectives of the study. Lawley (19,20) has discussed this. Figure 2.1 shows the method of carrying out the critical examination method (21) and figure 2.2 shows the hazop procedure application on each line the process.

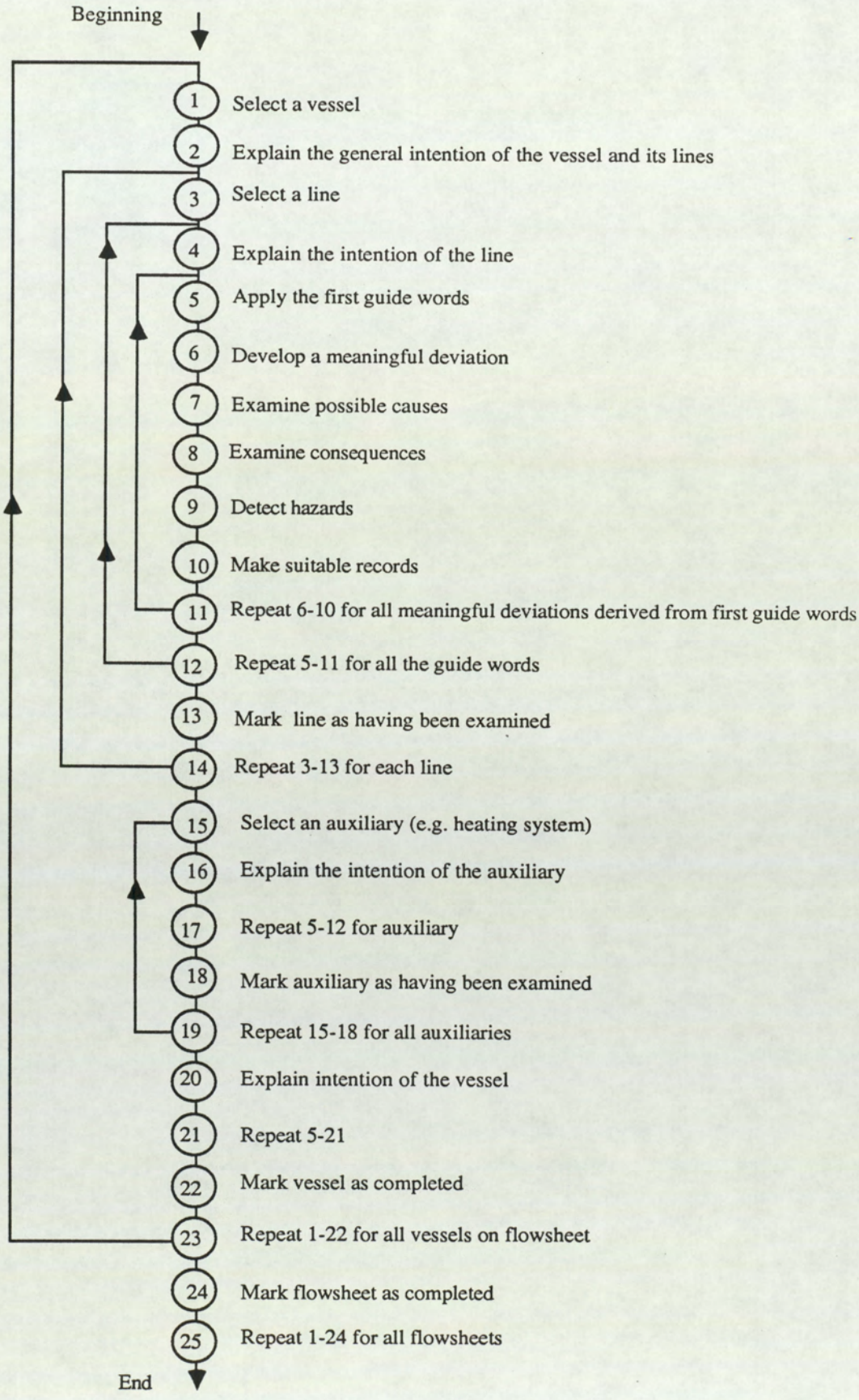


Figure 2.1 : Detailed sequence of critical examination (21)

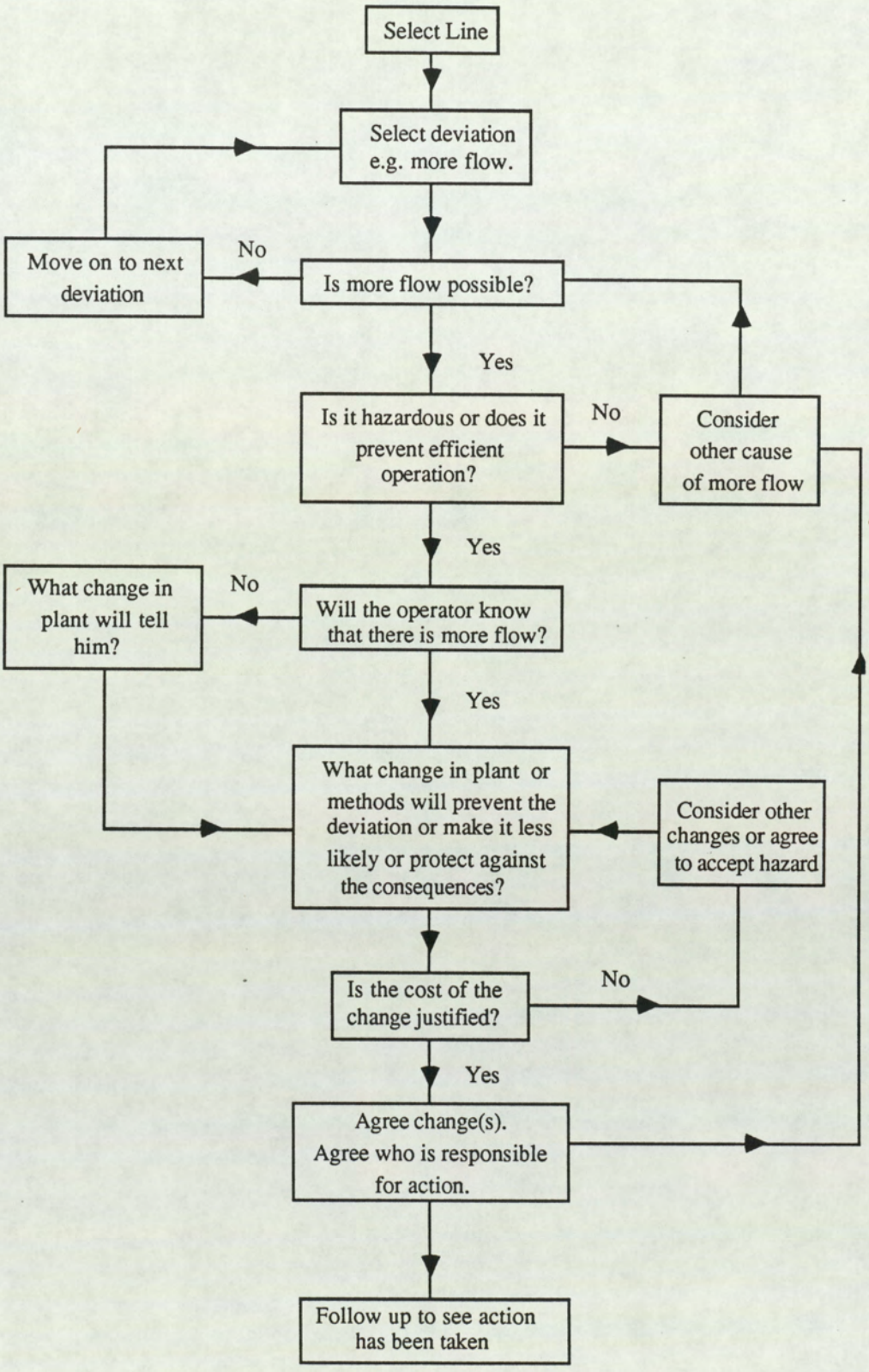


Figure 2.2 : Hazop procedure (21)

2.1.3 Principles of The Hazop Technique :

The basis of the hazop study may be a word model, a process flow sheet, a plant layout or a flow diagram. A word model consists of a set of **KEY WORDS** that stimulate thought. The keywords set consists of two sub-sets of words. The first sub-set consists of **PROPERTY WORDS** which focus attention on the process design conditions. Typical property words are flow, pressure, temperature and level. The second sub-set consists of **GUIDE WORDS** which focus attention onto possible deviations. These are simple words which are used to qualify the intention in order to guide and stimulate the creative thinking process and so discover deviations. Words like **NO, MORE, LESS** and **AS WELL AS** are guide words. Table 2.2 shows the different types of the guide words and their meaning as published by the Chemical Industries Safety and Health Council (22).

2.1.4 The Hazop Study Team :

The study is to be carried out by a team of specialists. The study team should be carefully chosen to provide knowledge and experience appropriate to the objectives of the study and the stage of development of the project. Thus important features of the study are :

1) Intention

2) Deviations

3) Causes

4) Consequences which can be either hazards or operating difficulties.

Table 2.2 : A list of Guide words (22)

Guide words	Meanings	Comments
NO or NOT	The complete negation of these intentions	No part of the intentions is achieved but nothing else happens
MORE LESS	Quantitative increases or decreases	These refer to quantities and properties such as flow rates and temperatures as well as activities like 'HEAT' and 'REACT'
AS WELL AS	A qualitative increase	All the design and operating intentions are achieved together with some additional activity
PART OF	A qualitative decrease	Only some of the intentions are achieved; some are not
REVERSE	The logical opposite of the intention	This is mostly applicable to activities, for example reverse flow or chemical reaction. It can also be applied to substances, e.g. 'POISON' instead of 'ANTIDOTE' or 'D' instead of 'L' optical isomers.
OTHER THAN	Complete substitution	No part of the original intention is achieved. Something quite different happens.

The selection of the team depends on the type of plant which is to be studied and the stage of its development. Kletz (23) discusses the team construction for both a newly design plant and for an existing plant. This is shown in table 2.3 and table 2.4. Sometimes the state of the plant, whether it is a continuous or a batch plant, will determine the team structure. However the team should have a leader who has a role to play through out the study. Kletz also details the way in which the study team should carry out their meetings.

2.2 Recording of Hazop Information :

2.2.1 Introduction :

A hazop study cannot be carried out, for a plant at the design stage, before the Process and Instrumentation (P & I) diagrams are complete. The extracted information from the study may not follow strictly the P & I diagram, depending on the depth of the study (23). The extent to which an operability study is recorded is somewhat debatable. Lawley (17) recorded operability data in terms of a table of entries of guide word, deviation, possible causes, consequences and action required. Another form of hazop information recording is the use of a checklist (24,25). To keep the operability records up to date the team must work out the remote interactions or the different parts of the plant. Sometimes this is too hard to follow due to the complexity and the continuous modification of the process design. It will be easier for the study team to follow these modifications if the recorded operability data are stored in a computer so that any unexpected operating difficulties or changes in the design intentions can be studied. Another benefit of storing operability study information in a computer is that remote interactions can be displayed as fault trees by a computer algorithm, using the stored information. Furthermore the advantage of computer records is that the

Table 2.3 : Team composition for a new plant (23).

Team member	Comments
Design engineer	Usually a mechanical engineer and, at this stage of the project, responsible for minimising the costs but not for hazards or operating problems.
Process engineer	Usually the chemical engineer who drew up the flow sheet.
Commissioning manager	Usually a chemical engineer who will start up and operate the plant.
Instrument design engineer	Requirement for plant with sophisticated control alarm and trip systems.
Research chemist	If new chemistry is involved
Independent chairman	An expert in the hazop technique, not the plant. Should ensure that the team follow the procedure. Leader of the team.

Table 2.4 : Team composition for an existing plant (23).

Team member	Comments
Plant manager	Responsible for operation.
Process foreman	Someone who knows what actually happens rather than what is supposed to happen.
Plant engineer	Resopnsible for mechanical maintenance, and therefore knowledgeable of many of the faults that occur.
Instrument manager	Responsible for instrument maintenance including testing of alarms and trips.
Process investigation manager	Responsible for investigating technical problems.
Independent chairman	An expert in hazop technique and leader of the team.

study need not proceed in the direction of flow on the P & I diagram. If there is some unresolved doubt about an equipment or a line, the team can study the next equipment on the flowsheet and include all the possible deviations in the incoming lines, without having established their causes.

2.2.2 Cause And Symptom Equations :

Lihou (26) has suggested a method of recording operability information. This method, Lihou suggests, will be easy to transfer to a computer. Guide words, property words, deviations, causes and consequences are stored in the form of two types of equations. The first type of equations are called **Cause Equations**, which describe how deviations could arise in pipelines. The second type of equations are called **Symptom Equations**, which show how items or equipments respond to input deviations and transmit deviations into outlet lines.

2.2.2.1 Coding Of Deviations :

On the P & I diagram each pipeline, item or control instrument has a reference number or an item identifier. The identifier is the same with cause and symptom equations but it is not necessary to have the exact lengthy reference name. For example, a pipeline which has the reference number LINE201 can be simply referred to as L201. If there is NO FLOW in this line, due to a malfunction, the deviation state can be recorded as L201 NO FLOW. The guide word is NO and the property word is FLOW. Index numbers are the most convenient way to code deviations in a computer. Table 2.5 forms the basis for coding the deviations and table 2.6 forms the basis for coding causes or failure modes (27). So the deviation in the above example can be coded as L201(11), where the bracket is used to separate

the deviation state from the line reference number. The first index number 1 codes the property word FLOW. The second index number, which is by chance equal to 1, codes the guide word NO. It will read as FLOW NO. This is because in recording the information on the operability sheet it is easier to record deviations under groups of operating conditions like:

a) Flow

b) Temperature

c) Pressure

If the deviation state of a chemical component is wanted to be recorded, for example LESS FLOW of component A in line LINE201, then the coded information will be L201(121). The third number 1 in the brackets refers to component A : this will depend, of course, on the particular system in use. Although seven guide words are sufficient (21), the list of property words can be quite extensive, depending upon the key words used to describe normal operations on a plant. Examples of other property words are React, Purge, Adiabatic, Calorific Value, pH and Viscosity. When an index number exceeds 9 the digits should be separated from adjacent index numbers by a slash; for example, (13/1) indicates a first index number of 13 and a second index number of 1.

Malfunctions of equipment, items or control instruments can be represented by a single number or letter, in brackets, following the item identifier number. The meanings of these codes are shown in table 2.6. The input and output points or critical zones inside the equipment can be considered as nodes to indicate the symptoms caused by these deviations.

Table 2.5 : Meaning of index numbers in brackets following a line or a node number (21).

Index number	Property word	Guide word
1	Flow	NO
2	Temperature	LESS
3	Pressure	MORE
4	Level	AS WELL AS
5	Concentration	PART OF / FLUCTUATION
6	Absorb	REVERSE
7	Heat transfer	OTHER THAN

Table 2.6 : Equipment failure modes indicated by a single index number or L (leaking) in brackets (27).

Equipment	Index number			
Type	0	-1	1	L
Compressor	Stopped, unloaded	Valves passing		Leaking
Controller	No signal	Set low	Set high	
Filter	Fully blocked	Partly blocked		
Heat exchanger	Tubes fully blocked	Tubes partly blocked		Tubes leaking
Indicator	No signal	Indicating low	Indicating high	
Level switch		Stuck low or set high	Stuck high or set low	
Line	Fully blocked	Partly blocked		Leaking
Orifice plate	Blocked	Orifice too large, density low	Orifice too small, density high	
Pnumatic trip Valve (3 way)	Vent branch isolated	Leaking to vent	Open to vent	
Pump	Stopped	Low throughput	Running	Leaking
Safety interlock	Fails to operate		Operates	
Transmitter or transducer	No signal	Indicating too low	Indicating too high	
Valve	Closed, blocked	Insufficien-tly open, leaking	open, open too much	Non-return passing

2.2.2.2 Cause Equations :

Deviation states which appear in pipelines are separated from each other by a + sign to indicate OR condition, and by * sign to indicate AND condition. The causes may be deviations from normal conditions in other lines or they may be due to malfunction of equipment. Appendix A shows the rules for writing cause equations to avoid illogical fault trees.

For the sake of illustration, consider figure 2.3 which simulates the P & I diagram of the Solvay Process for manufacturing sodium bicarbonate (27). To transfer gaseous ammonia from a high pressure cylinder NH3 to the sump of the Ammoniation column at a controlled rate, such that a small excess leaves at node 2, cause equations for line 1 can be written as follows :

L1(11) = V1(0) + PR1(0) + NH3(41) .....[1]

L1(12) = V1(-1) + FI1(1) .....[2]

L1(13) = V1(1) + FI1(-1) .....[3]

L1(32) = PR1(-1) + NH3(42) .....[4]

L1(33) = PR1(1) .....[5]

Equation [1], a cause equation, explains that no flow in line 1 can be caused when valve V1 is closed or the pressure regulator PR1 is closed or the ammonia tank NH3 is empty. Equation [5] explains that more pressure than normal is caused by high setting of the pressure regulator PR1 only.

However, additional parentheses can be used if necessary to state the priority of logical combination of causes. The AND operator always has higher logical priority than the OR operator. For example no flow in line 2 can be shown

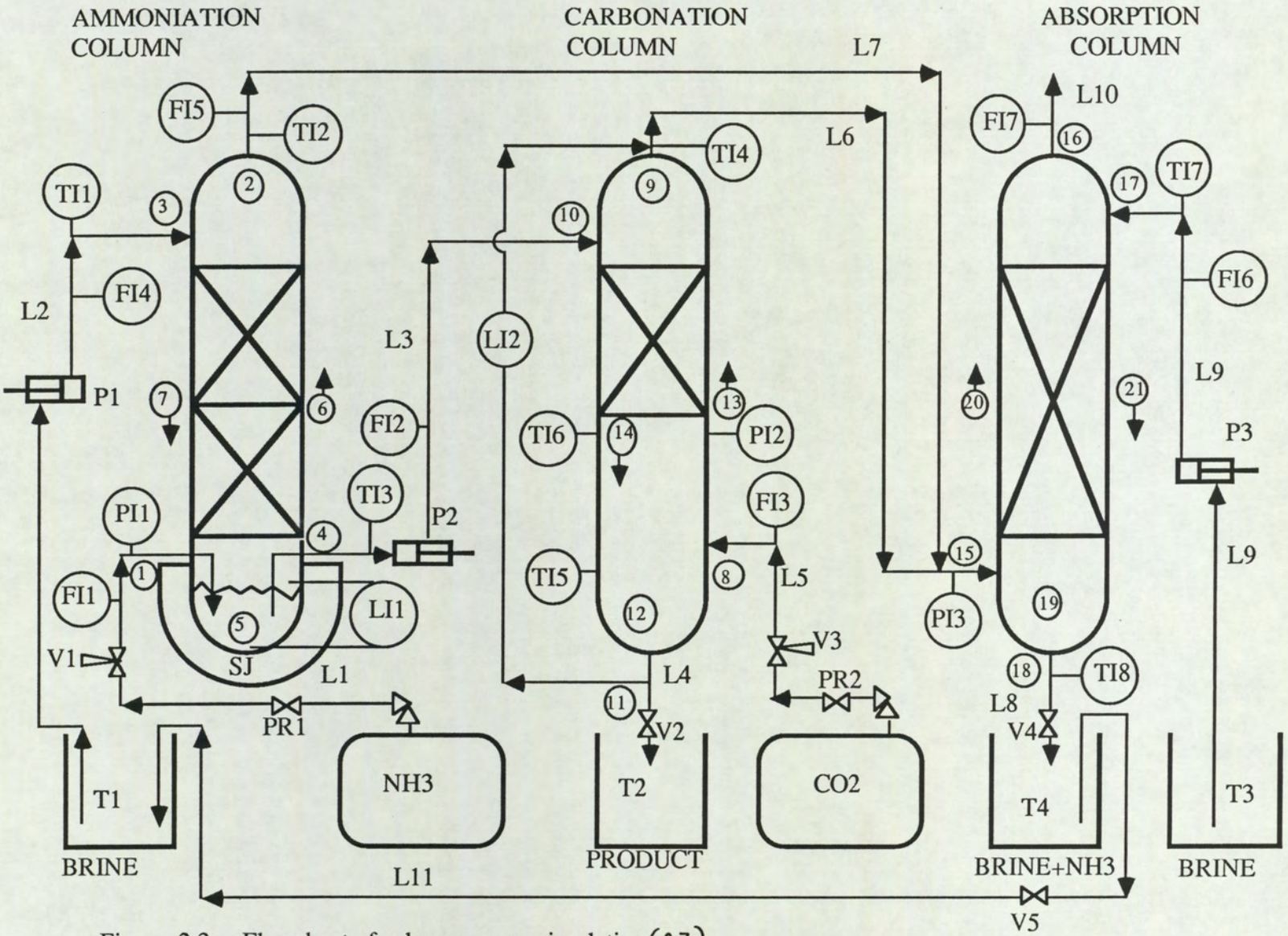


Figure 2.3 : Flowsheet of solvay process simulation (27)

by the following cause equation :

$$L2(11) = P1(0) + T1(41) * (T4(41) + V5(0))$$

No flow in line 2 can be caused by either pump P1 being stopped, or tank T1 being emptied, either with tank T4 being empty or with valve V5 closed.

2.2.2.3 Symptom Equations :

Symptom equations are used to describe how major items of equipment respond to single deviations. Physical and chemical changes inside the equipment are modelled to allow for a multiplicity of simultaneous responses to single causes. The causes are mostly deviations in properties of input streams to the equipment; however in some circumstances, output streams can also cause symptoms in the equipment which the stream is leaving. For example, changes in flow rate of an output liquid stream may cause response in liquid levels in the equipment.

The points where streams enter and leave the equipment are considered as nodes and each node is given a unique number. Immiscible streams that undergo physical, thermal or chemical exchange are each given unique node numbers.

Symptom equations differ from cause equations in that the cause is at the beginning of the equation, followed by an arrow -> sign and then a list of consequences at nodes. Each node symptom is separated from each other by an *, to indicate AND. For example, the symptom equations generated from different causes in line 1, from the previous example, can be written as follows :

L1(11) -> N1(11)*N1(31)*N2(11)*N4(125)*N4(22)*N6(11)

L1(12) -> N1(12)*N1(32)*N2(12)*N2(22)*N4(125)*N4(22)*N6(12)

L1(13) -> N1(13)*N1(33)*N2(13)*N6(13)*N6(23)

The first symptom equation explains that the cause of no flow in line 1 can cause the following symptoms in the Ammoniation Column :

i) no flow in node 1

ii) no pressure in node 1

iii) no flow in node 2

iv) less flow of ammonium hydroxide in node 4

v) less temperature in node 4, and

vi) no flow in node 6.

2.3 Fault Tree Analysis :

2.3.1 Introduction :

Fault tree analysis is an approach to reliability and safety analysis and is generally applicable to complex dynamic systems. Fault tree analysis provides an all inclusive, versatile, mathematical tool for analyzing complex systems (28). The logic of the approach makes it a useful visible tool for both engineering and management. It provides a deductive functional development of a specific undesired event through logic statements of the conditions which could cause the event (29).

In 1961 the concept of fault tree analysis was originated by H. A. Watson of Bell Telephone Laboratories to evaluate the safety of the Minuteman Launch Control System (30). At the 1965 Safety Symposium, sponsored by the

University of Washington and the Boeing Company, several papers were presented that expanded the virtues of fault tree analysis (31). Fussell (28) stated the fault tree analysis major value in :

1) Directing the analyst to ferret out failures deductively.
2) Pointing out the aspects of the system important in respect of the failure of interest.
3) Providing a graphical aid giving visibility to those in system management who are 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 failure at a time.
6) Providing the analyst with genuine insight into system behaviour.

In addition recent studies show that the fault tree analysis is of a good help in the field of operator training, decision making, start up and shut-down procedure, alarm analysis and in writing operating manuals (32,33). In the early 1970's great strides were made in the solution of fault trees to obtain complete reliability information (34,35,36,37).

2.3.2 Fault Tree Terminology :

Different names and terms are used to define a proper act or description being associated with the fault tree analysis. Some of these definitions are described as follows: A **hazard** is the term used for a latent condition or set of conditions, either internal or external to a system, facility, or hardware end item, which when activated, trigger into an event or events that culminate in an accident mechanism. An **accident** is the term used to describe an event or a group

or series of unplanned, unwanted events which have occurred in a static or dynamic system to produce a loss or a near loss. An **event** is the term used for one of a series of concurrent or sequential interacting occurrences that either flow or cascade into other events, thereby constituting the accident mechanism. This event could be a primary or basic failure in a system and can be described as a system component. The development of any group of events results in a branch of the fault tree. The branch is complete only when all events in the branch are developed to the level of primary failures. The event is developed through a logic gate. A logic **gate** defines the input conditions which must be met in order for a failure sequence to propagate up the fault tree toward the final or **TOP** event. The logic gates that are most frequently used to develop fault trees are the basic AND and OR Boolean operators. The **AND** gate provides an output event if, and only if, all the input events are simultaneously present. The **OR** gate transmits an output event if one or more of the input events are present. Other logic operators (gates), frequently in use, are the Exclusive OR and the NOT gates (38,39). An **Exclusive OR** gate, sometimes given an EOR or XOR abbreviation, has two inputs and will produce a TRUE output if one, but not both, of them is TRUE. A fault tree containing only simple AND and OR gates is called an **s-coherent** fault tree (the prefix "s-" implies "statistical"). An **s-noncoherent** fault tree is one which has full or partial statistical dependent relationships among its elements. In other words, an s-noncoherent fault tree contains either EOR or NOT gate among its events (40).

The application of the set theory and the boolean algebra in fault tree analysis is well known (41). A **path set** is represented by the logic configuration of the primary events in the tree. A **minimal path set** is the smallest set of component successes that will ensure that the undesired event will not occur. A **cut set** is a set of basic events whose presence cause the occurrence of an

undesired event. But a **minimal cut set** is the smallest set of primary events in which the presence of all events are necessary to cause an undesired event to occur. The concepts of minimal paths and minimal cuts have been established by Esary and Proschan for coherent structures (42). However, their concepts were built up for systems that did not allow complementary events to occur in a fault tree. If a primary event is X then its complementary event is (1-X) or $\bar{X}$. When complementary events do occur a more general way of specifying fundamental modes of behaviour is required. In non-coherent fault trees, the minimal cut set concept should be replaced by a set of literals called a **prime implicant** set. A **literal** is either a primary event or the complementary event in a tree which is noncoherent. Another well used term is the common mode failure or the common cause failure. **Common mode failures** are multiple failures that result from a single event or failure. Thus, the probabilities associated with the multiple failures become, in reality, dependent probabilities. The single event can be a common environment, common design or external event. A common external event can be caused by a common human operator. Special graphical symbols have been used in the graphical representation of fault trees. Appendix B shows the most commonly used symbols (43,44).

2.3.3 General Procedure Of Fault Tree Analysis :

Generally, the fault tree analysis can be presented in the following four steps (28):

1. System definition
2. Fault tree construction
3. Qualitative evaluation
4. Quantitative evaluation

2.3.3.1 System definition :

The fault tree analysis begins with the determination of the undesired events in a particular piece of equipment in a process or a system (45,46). Since the definition of this system is often the most difficult task, a layout diagram, showing all the system components is essential. The detail of the diagram is dependent upon the study depth. Sufficient information about each component must be available to minimize the number of undesired events and to reduce the range within which they may occur. However, in addition to this the system boundary conditions, such as the top event, must not be confused with the system physical bounds. The system boundary conditions define the situation for which the fault tree is to be drawn.

2.3.3.2 Fault Tree Construction :

The construction of a fault tree is an important task in the overall activity of fault tree analysis. Fault tree actual construction is usually done by hand. Rasmussen stated that the WASH-1400 study took about 25 man-years of effort to complete (47). In the past ten years, computer-aided synthesis of fault trees haS attracted considerable attention and several methodologies have been proposed. Hassl (48) devised a structure that establishes rules to determine the type of gate to use and inputs to the gate. Fussell (45) uses transfer functions as models for component failures to construct the final fault tree for electrical systems. His technique is known as the Synthetic tree Model (STM) and is based on modelling each device in the system by a failure transfer function. The various transfer functions, traced through the schematic, are combined and edited to form the final fault tree. Powers and Tompkins (49,50) devised a method for automated fault tree construction for chemical systems. Their approach is to break down

the system into constituent blocks, and define their operations via unit models, then to combine these systematically to form the fault tree. Salem et al. (51) devised CAT (Computer Automated Tree) code which presented a general computer-implemented approach for modelling nuclear and other complex systems involving electrical, mechanical and human interaction. Lapp and Powers (52) generate a digraph model (direct graph) for system representation and then uses the fault tree synthesis program to deduce the fault tree. Taylor and Hollo (53) use algebraic component models to construct a Cause Consequence Diagram (CCD) and then generate algebraic equations (54). The generated equations are then written for each component and the resulting collection forms the system model. This model can then be used to determine the consequences of any deviation in the input variables. Finally, Camarda et al. (55) proposed an efficient algorithm for fault tree automation synthesis from the reliability graph for large systems.

The qualitative and the quantitative evaluations of fault trees will be discussed in full in chapter three.

CHAPTER THREE

3. RELIABILITY CALCULATIONS

3.1 Introduction :

The discipline which is concerned with the application of probabilistic methods to the problems of failure in systems generally is known as reliability engineering. The earliest developments in reliability engineering occurred during the Second World War when the Germans had problems with the reliability of the V1 missile (56). Since this beginning the study of reliability has become a fully developed discipline. Particular impetus has been given by the reliability requirements in the defence (57), aerospace and nuclear industries (47,58). This has raised the need for new techniques to be developed and adapted to the new technology requirements. In the process industries, reliability analysis is now playing an increasingly important role in quantitative assessment of system performance for assuring safety, for improving plant performance and plant life and for reducing plant operating costs.

Fault tree analysis, as stated earlier, is considered a powerful tool for the qualitative evaluation of potential hazards in a system. The use of reliability assessment in conjunction with fault tree analysis considerably increases the power of the technique. Different techniques and methods of reliability assessment have been developed by the application of probabilistic methods with Boolean Algebra. Some reliability assessment methods use stochastic techniques for prediction. Others apply deterministic techniques using Boolean Algebra to describe the system reliability in terms of the combined failure components of the system.

Different factors and criteria are involved in establishing any newly

developed technique. The major factors are:

1- Aims and goals of the developer

2- Scope and field of the application

3- Availability of failure data, and

4- Type of hardware and software tools used in the study.

3.2 Stochastic Techniques :

These techniques fall into two categories. The first category of techniques involve the direct application of statistical and probabilistic laws for the analysis of fault trees. The second category of techniques involve the use of random variables or random numbers using Monte-Carlo simulation methods. The direct application of probability laws are limited to simple logical relationships and have been found unsuccessful when applied to fault trees which authentically represent large complex systems (44) especially if repeated, repaired events are involved. Nevertheless, these laws have been found to be successful and can be used with some confidence when there is independency among the individual tree components.

3.2.1 Simple Statistical Rules :

The probability of success P in a system in which all the components must work, if the system is to work, is the product of the individual probabilities of success P_i

$$P = \prod_{i=1}^{n} P_i \qquad \text{..... (3.1)}$$

It is appropriate to give, at this stage, a brief description of some basic probability laws.

3.2.1.1 Union Of Events :

The probability of an event X to occur if any of the events A_i occurr is the union ($\bigcup$) of those events .

$$P(X) = P(\bigcup_{i=1}^{n} A_i) \qquad \text{..... (3.2)}$$

for two events

$$P(X) = P(A_1 \cup A_2)$$

$$= P(A_1) + P(A_2) - P(A_1 A_2) \qquad \text{..... (3.3)}$$

If there are n events then

$$P(X) = P(A_1) + P(A_2) + P(A_3) + \text{.....} + P(A_n)$$

- probability of all possible double combination

+ probability of all possible triple combination

- .....

.....

$$+ (-1)^{n-1} P(A_1) P(A_2) \text{.....} P(A_n) \qquad \text{..... (3.4)}$$

If the events are mutually exclusive, equation (3.4) simplifies to

$$P(X) = \sum_{i=1}^{n} P(A_i) \qquad \text{..... (3.5)}$$

In general

$$P\left(\bigcup_{i=1}^{n} A_i\right) \leq \sum_{i=1}^{n} P(A_i) \qquad \text{..... (3.6)}$$

3.2.1.2 Intersection Of Events :

The probability of an event X which occurs only if all the n events A_i occur is the intersection ($\prod$) of these events:

$$P(X) = P(A_1 A_2 \ldots\ldots A_n) = P\left(\prod_{i=1}^{n} A_i\right) \qquad \text{..... (3.7)}$$

For two events

$$P(X) = P(A_1 \prod A_2) = P(A_1) \cdot P(A_2) \qquad \text{..... (3.8)}$$

Generally

$$P(X) = P\left(\prod_{i=1}^{n} A_i\right) \qquad \text{..... (3.9)}$$

3.2.1.3 Bayes' Theorem - Conditional Probability :

The probability of an event X which occurs if the event A and the event B have occurred where the event A depends on the event B is

$$P(X) = P(AB) = P(A/B)P(B) \qquad \text{..... (3.10)}$$

P(A/B) is the conditional probability of A given B.

For n events of A_i dependent upon event B :

$$P(X) = \sum_{i=1}^{n} P(B/A_i)\, P(A_i)$$

The above form is a special case of the more general form of Bayes' theorem; given events A and B, the probability of event X is:

$$P(AB) = P(A/B)P(B) = P(B/A)P(A) \qquad \text{..... (3.11)}$$

$$P(A/B) = \frac{P(AB)}{P(B)} = \frac{P(B/A)P(A)}{P(B)} \qquad \text{..... (3.12)}$$

For n events, the probability of the kth event A_k is :

$$P(A_k/B) = \frac{P(B/A_k)\, P(A_k)}{\sum_{i=1}^{n} P(B/A_i)\, P(A_i)} \qquad \text{..... (3.13)}$$

3.2.2 Failure Rate And Failure Distributions :

The failure of an equipment is not only a function of its design and quality of construction, but also of the environmental conditions which it is placed. Predictions of failure can be made from statistical data based on past observations. The confidence in such predictions will increase with an increase in the number of independent observations. So a large number of independent observations and well defined system conditions will give a high level of confidence in probabilistic predictions. Some other factors which should be considered are failure mode, repair time, true running time and time intervals between failure.

3.2.2.1 Mean Time Between Failures And Failure Rate :

Different terms are in use to describe different types of failure. Mean Time Between Failures (MTBF), Mean Time To Failure (MTTF) and the Mean Time To First Failure (MTTFF) are the most widely used terms in reliability engineering (59). Formal definitions of MTBF and MTTF are given in the appropriate British Standard (60). MTBF has meaning only when applied to a population of components, equipments or systems in which there is repair. It is the total operating time of the items divided by the total number of failures. If n failures have occurred with the interval between successive failures i and i+1 being t_i years, then MTBF and its variance σ^2 is given by equations 3.14 and 3.15 respectively.

$$MTBF = \frac{\sum_{i=1}^{n} t_i}{n} \qquad \text{..... (3.14)}$$

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} (t_i)^2 - \frac{1}{n(n-1)} \left(\sum_{i=1}^{n} t_i \right)^2 \qquad \text{..... (3.15)}$$

MTTF is applied to items without repair and is the mean of distribution of times to failure. MTTFF is applied to items with repair and is the mean of the distribution of times to first failure. Failure rate is defined as the number of failures observed in a very small interval of time divided by the number of non-failed systems (61). For an n component parallel system with an exponential failure distribution of the individual components and without repair, if y is the failure rate of equipment, then :

$$MTTF = \sum_{i=1}^{n} \frac{1}{iy} \qquad \text{..... (3.16)}$$

and for an n component parallel system with repair, if s is the repair rate of equipment , then

$$MTTFF = \frac{1}{y} \sum_{i=0}^{n-1} \frac{(1+s/y)^i}{i+1} \qquad \text{..... (3.17)}$$

If the repair rate s is zero, equation (3.17) reduces to (3.16).

3.2.2.2 Failure Distributions :

The probability that something will fail at a prescribed instant is zero. Failure distributions are represented by functions of time f(t), such that when integrated over a specified time interval t_1 to t_2, the probability of at least one failure being obtained during that time interval is given by :

$$F(t_1,t_2) = \int_{t_1}^{t_2} f(t)\ dt \qquad \text{..... (3.18)}$$

There are two types of statistical distributions which are fundamental in work on reliability. There are either discrete distributions such as binomial, multinomial and Poisson or continuous distributions such as exponential, normal, lognormal, Weibull, rectangular, gamma, Pareto and extreme value distribution. Hastings and Peacock (62) give a comprehensive summary of the properties of the above mentioned statistical distributions. Only the exponential and the Weibull distributions will be described in this work because of their wide use in equipment and systems reliability studies.

3.2.2.2.1 Exponential Distribution :

For the exponential distribution the characteristics of instantaneous failure rate (z), overall failure rate or failure density (f), reliability (R) and failure

distribution (F) can be expressed as follow :

$$z = y \qquad \text{..... (3.19)}$$

$$f = y \exp(-yt) \qquad \text{..... (3.20)}$$

$$R = \exp(-yt) \qquad \text{..... (3.21)}$$

$$F = 1 - R \qquad \text{..... (3.22)}$$

where $0 \leq t \leq \infty$

There is only one parameter y, which is the failure rate. The failure rate is assumed to be independent of time or age or environmental influence; it is the reciprocal of MTBF. The reliability of equipment during any time interval of MTBF is 0.368. This is usually applied to data in the absence of other information and is the most widely used in reliability work.

3.2.2.2.2 Weibull Distribution :

There are two forms of the Weibull distribution, one with three parameters and one with two. The characteristics of the three parameters form are :

$$z = \frac{\beta}{\eta}\left(\frac{t-\gamma}{\eta}\right) \qquad \text{..... (3.23)}$$

$$f = \frac{\beta}{\eta}\left(\frac{t-\gamma}{\eta}\right)^{\beta-1} \exp\left[-\left(\frac{t-\gamma}{\eta}\right)^{\beta}\right] \qquad \text{..... (3.24)}$$

$$R = \exp\left[-\left(\frac{t-\gamma}{\eta}\right)^{\beta}\right] \qquad \text{..... (3.25)}$$

where the range is $0 \leq t \leq \infty$

Here η is the characteristic life, β is the shape factor and γ is the location parameter. If γ is zero, the two-parameter distribution is obtained. A high value of η indicates a well designed system, good quality control, large factors of safety and that equipment is operating below capacity. Table 3.1 shows the significance of the shape factor on the simulated distribution functions. Lihou used the Weibull distribution to calculate the optimum shape factor for some equipment in the process industries (63). The resultant graph is shown in Appendix C.

3.2.3 Monte-Carlo Methods :

Many practical problems cannot be solved by any of the available analytical

Table 3.1 : The effect of the shape factor on the simulated distribution

β	Simulated distribution	Applicability	Hazard rate
< 1	Hyper-exponential	Early failure	Decreasing
1	Exponential	Random failure	Constant
2	Log-normal	Repair failure	Increasing
3	Normal	Wear out	Increasing

methods and are only soluble by simulation (47). The principle methods in use are the Monte-Carlo simulation techniques.

The procedure for the application of a general Monte-Carlo simulation to the determination of reliability or availability of complex systems is as follows:

1. The system configuration, component failure characteristics, system constraints and time period of interest are specified.

2. The time period is divided into small intervals or increments. The first trial is then carried out covering the time period specified.

3. At the first time increment the probability of failure of the first equipment is calculated. This probability is compared with a random number in the range of 0 to 1 generated from a uniform distribution: if the random number is less than or equal to the failure probability the equipment fails, otherwise it survives.

4. The state of all the equipment is computed in the same way. Then the

overall state of the system will be determined.

5. If the total system has failed, the time to first failure has been calculated for this simulation. Otherwise steps 3, 4 and 5 are repeated for following time increments to the end of the specified time period.

The whole simulation is repeated a sufficient number of times so that a result which has reasonable confidence limits has been obtained. To carry out the Monte-Carlo simulation, a computer program is needed. The above procedure represents the application of the Crude or Simple Monte-Carlo method. It is also known as direct simulation. Crosetti (44) described the steps of using Crude Monte-Carlo technique in fault tree analysis.

Other types of Monte-Carlo methods involve the use of some statistical techniques (64, 65, 66). Mazumder (67) proposed a Monte-Carlo method with variance reducing techniques to decrease the variance of the Monte-Carlo estimates reliability. Kumamoto et al.(68) used a Monte-Carlo method for estimating the reliability of large complex systems based on a fault tree or reliability diagram. Levy and Moore (69) used a Monte-Carlo technique for obtaining system reliability confidence limits from component test data. Vesely and Narum (70,71) present PREP and KITT fault tree computer programs using a Monte-Carlo method. SAFTE (72), RELY4 (73), REDIS (74) and SAMPLE-WASH 1400 (47) are computer algorithms which already exist for fault tree analysis that use the Monte-Carlo simulation methods. Karp and Luby (75) show how to exploit knowledge of the failure sets of a network using a Monte-Carlo method.

3.3 Deterministic Techniques :

3.3.1 Introductory Theory :

The fault tree technique has been used by reliability engineers as a general tool for studying system failures. The technique is based on a graphical representation of the logical sequences of causes that lead to the system failure of the top event. A qualitative evaluation of a fault tree involves the application of Set Theory and Boolean Algebra in order to obtain a Boolean expression known as the minimal cut sets of the top event.

The fault trees in themselves are qualitative in nature, but they provide a framework for the probabilistic analysis. A quantitative evaluation of a fault tree basically consists of assigning probabilities to each fault (basic event) and of combining them as prescribed by the fault tree, to obtain the probability of the top event. Usually, this is not a direct substitution of failure data or failure functions of the individual events in the tree. In real systems, complexity, dependency, repairable components or the existence of repeated events make this technique inadequate. So different techniques have been proposed in order to calculate the exact system probability.

3.3.2 Laws of Boolean Algebra :

Fault trees are originally constructed in graphical form using logical operators, mainly AND and OR gates. Boolean algebra is the most appropriate tool to represent them in a mathematical form. For example, the OR gate is equivalent to the Boolean symbol '+' and represents the union of the events attached to the gate as its inputs. The AND gate is equivalent to the Boolean symbol '.' representing the intersection of events. Table 3.2 (76) shows those Laws of

Table 3.2 : Relevant Laws of Boolean Algebra

Feature	Algebraic representation
Commutative law	a. $XY = YX$ b. $X+Y = Y+X$
Associative law	a. $X(YZ) = (XY)Z$ b. $X+(Y+Z) = (X+Y)+Z$
Idempotent law	a. $XX = X$ b. $X+X = X$
Complemented law	a. $X+\bar{X} = 1$ b. $1.f(X) = f(X)$
Absorption law	a. $X(X+Y) = X$ b. $X+XY = X$
Distributive law	a. $X(Y+Z) + XY+XZ$ b. $(X+Y)(X+Z) = X+YZ$
Reduction law	a. $X+\bar{X}Y = X+Y$ b. $XY+XY = XY$ c. $XY+X\bar{Y} = X$

Boolean Algebra most relevant to this work.

3.3.3 Probability Calculations From Fault Trees :

Much of the value of fault tree analysis lies in the fact that if probabilities can be assigned to the events in the tree, then the probability of the top event can be evaluated. This does not necessarily mean that all event probabilities must be known: only the basic events probabilities are the essentials. For example, consider the tree shown in figure 3.1, where A, B, C and D are independent events.From the fault tree logic and probability laws :

$X = A.B.Y \qquad P(X) = P(A)P(B)P(Y)$

$Y = C + D \qquad P(Y) = P(C) + P(D) - P(C)P(D)$

Hence,

$X = A.B.C + A.B.D$

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

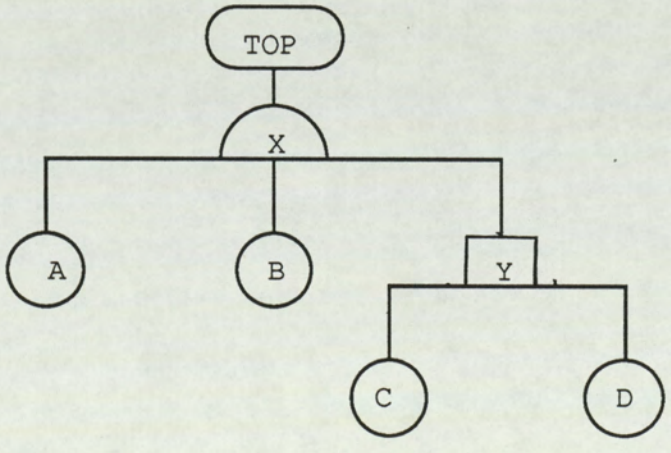


Figure 3.1 : A fault tree

If the probability values of individual events are very small then an approximation for the result can be made by deleting higher order terms. This sometimes helps to reduce the computation time, especially in large trees, without giving too much error in the final result.

3.3.3.1 Effect Of Repeated Events :

If repeated basic events exist then direct application of probability laws are not possible. If they are applied directly in the Boolean expression, a wrong value of the top event probability will be obtained. For example, consider the fault tree in figure 3.2, originally given in Fussell (77), with the basic events A, B, C and D. From the fault tree logic and probability laws :

$X0 = X1.X2 \qquad P(X0) = P(X1)P(X2)$

$X1 = A.X3 \qquad P(X1) = P(A)P(X3)$

$X2 = B + D \qquad P(X2) = P(B)+P(D)-P(B)P(D)$

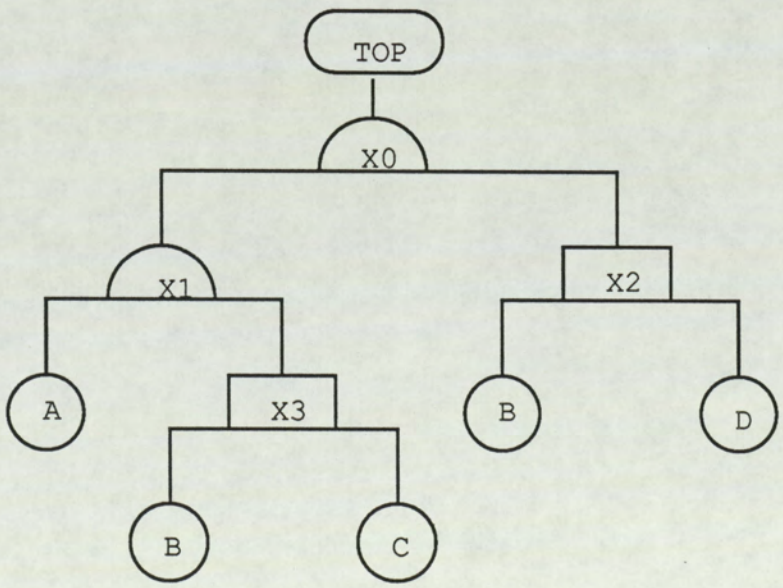


Figure 3.2 : A fault tree (77)

$X3 = B + C \qquad P(X3) = P(B)+P(C)-P(B)P(C)$

Hence,

$X0 = [A.X3][B+D] = A.[B+C].[B+D] = A.B.B+A.B.D+A.B.C+A.C.D$

Bottom up calculation of probabilities in the order P(X3), P(X2), P(X1) and hence P(X0) will give the value of the expression

$P(X0) = P(A)[P(B)+P(C)-P(B)P(C)][P(B)+P(D)-P(B)P(D)]$

However, further reduction of the Boolean expression gives

$X0 = A.B+A.C.D$

and the substitution of probabilities into this Boolean expression will give a different answer.

$P(X0) = P(A)P(B) + P(A)P(C)P(D)$

In fact both are incorrect. Note that the Boolean expressions are both correct. In order to get the right answer, other techniques must be used.

3.3.4 Minimal Cut Sets And Reduced Logical Expressions :

As shown in the above examples, Boolean algebra can be used to reduce the logical expression to a sum of product expression. In the first example, a sum of product expression (SOP) was obtained :

$ABC + ABD$ ..... (3.26)

And in the second example, a sum of product expression (SOP) was obtained too :

$$AB + ABC + ABD + ACD \quad \text{..... (3.27)}$$

This means that the expression in (3.26) consists of two sets or what can be called cut sets. In expression (3.27), the top event occurrence is represented by 4 cut sets. This second expression can be reduced further, using the Boolean absorption law, to yield :

$$AB + ACD \quad \text{..... (3.28)}$$

This is because both of the cut sets ABC and ABD are supersets of the set AB. The set in the expression of (3.28) are called the minimal cut sets of the top event.

The concept of minimal cut sets is very important for fault tree evaluation. Consider the following example given by Semanderes (34) :

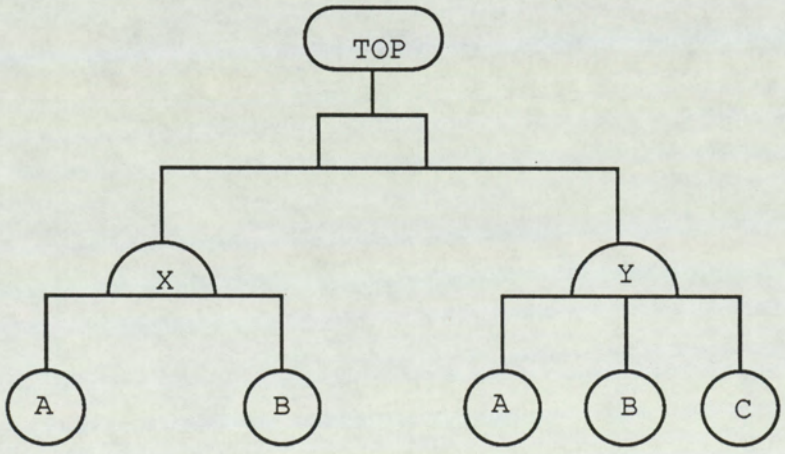


Figure 3.3 : A fault tree (34)

Top = X + Y

= AB + ABC

= AB (Minimal cut set)

This illustrates that the occurrence of the top event is controlled by basic events A and B only but not by C. Thus not all basic events are always required for the calculation of probability of occurrence of top events. However it is not necessarily the case that probabilities can be directly substituted into the minimal cut sets expression.

3.3.5 Methods For Generating Minimal Cut Sets :

An important step in fault tree analysis is determining its minimal cut sets. A minimal cut set is a collection of component failures all of which are necessary and sufficient to cause the system failure by that minimal cut set. A complete set of minimal cut sets are all the failure modes for a given system (28). Thus several algorithms for obtaining the minimal cut sets for the TOP event of a fault tree have been proposed. The first algorithms were based on Monte-Carlo techniques; later methods are deterministic. These deterministic techniques fall into two main approaches, either a bottom-up or a top-down approach.

In 1970 Vesely and Narum (70,71) made available a set of computer programs (PREP) that obtained the minimal cut sets (or minimal path sets) for the fault tree and then obtained quantitative system characteristics from these cut sets (or path sets). It represents one of the earliest deterministic algorithms. Because of the time consuming nature of the algorithm used in PREP, several newer and more efficient deterministic programs have been proposed.

Fussell coded MOCUS (77) starting from the top of the fault tree and proceeding down to the basic events to determine the minimal cut sets of the TOP event. Only AND and OR gates are allowed. The TOP event is replaced by the gates and basic events that are input to it. If the TOP event is an AND gate, the inputs are listed in a row. If the TOP event is an OR gate, the inputs are listed in a column. Each gate in the matrix is now expanded using the same substitution technique. The process continues until only basic events are contained in the matrix. The rows of this final matrix are the cut sets of the tree. Minimal cut sets are obtained by determining supersets from the matrix. The order of a cut set is the number of events in the cut set. Output from MOCUS includes all the cut sets up to a desired order. In addition to obtaining cut sets for the TOP event, cut sets can also be obtained for intermediate events in the tree.

ELRAFT (34), coded by Semanderes, is a FORTRAN program which formulates the simplest logic expression for each secondary event in a fault tree in terms of the basic events which combine to cause it. Other well-known deterministic programs for determining minimal cut sets are SETS (41), ALLCUTS (78), MICSUP (79), BAM-CUTS (80), DICOMICS (81), FATRAM (82), BUP-CUTS (83), RESIN (84) and the Jamson-Kai algorithm (85).

Bengiamin et al. (86) have developed a technique for handling repeated basic events in a fault tree. Koen and Carnino (87) have introduced a pattern recognition technique for fault tree evaluation. The basic idea of pattern recognition is to prune the fault tree by identifying known patterns, retrieving the corresponding mathematical equation, and evaluating the replacement basic events. This process is repeated until the original tree is reduced to a single node or the

system reliability. Appendix D shows the standard patterns and their mathematical forms as given by Koen and Carnino.

Wheeler et al. (88) introduced a fault tree analysis program that uses bit vector representation of cut sets. Each primary event is assigned a single 1 in a unique position in a sequence of zeros (binary digits).

Kumamoto and Henley (89) have introduced a top-down algorithm to obtain prime implicant sets for non-coherent fault trees which are equivalent to minimal cut sets in coherent trees. The algorithm produces a sum of product (SOP) expression.

3.3.5.1 Selected Algorithms And Techniques For Obtaining Minimal Cut Sets In Fault Trees With Repeated Basic Events :

Three different approaches for obtaining the minimal cut sets in fault trees which contain repeated basic events are described below.

3.3.5.1.1 Tree Reduction Technique :

Bengiamin et al. (86) have developed a technique for obtaining the minimal cut set in fault trees that contain repeated basic events. The philosophy of the technique is to reduce the influence of repeated events in the fault tree by reducing the fault tree itself. This technique, as claimed by Bengiamin et al., will reduce the required computing time because of the early elimination of non-minimal cut sets. The algorithm which was written in APL consists of three basic steps :

1. The fault tree containing repeated events is reduced. A reduced fault tree is obtained by eliminating repeated events which are inputs to OR gates.

2. The cut sets of the reduced tree are then obtained by conventional techniques, namely Fussell and Vesely's method (90). The resultant cut sets are referred to as Group 1 cut sets.

3. Finally, the Group 1 cut sets are then further processed to yield the Group 2 cut sets. The cut sets Group 1 and Group 2 are the desired result.

These three steps are described by Bengiamin et al. (86) as follows :

I) Step 1 : Produce a Reduced Fault Tree by deleting repeated events which are inputs to OR gates, as follows. For each OR gate which has one or more repeated event,

a) If one or more of the input events to the same gate are non-repeated, one of them is designated as a partner, and that instance of the repeated event(s) eliminated.

b) If all the input events to the same gate are repeated events, the gate output is examined :

i) If the output event is a transfer event (90) with the same order of repetition as any of the input events, then this event is designated the partner, and that instance of the repeated event(s) elimimnated. The order of repetition is the number of times by which an event is repeated in the same tree.

ii) If the output event is a non-transfer event or an event of different order of repetition than all the input events, then a new event is introduced to represent the input events (artificial events), and that instance of the repeated event(s) eliminated.

II) Step 2 : Find the cut sets of Group 1 by analysing the reduced fault tree as follows,

a) Generate cut sets of the reduced fault tree by Fussell and Vesely's method (90).

b) To get Group 1, process the obtained group of cut sets as follows,

i) If all the repeated events of the original tree were inputs to OR gates and each OR gate had at least one non-repeated event; then all the obtained cut sets are minimal and no processing should be done.

ii) If all the repeated events of the original tree were inputs to OR gates and some of the partners are compared with each other to eliminate any non-minimal ones.

iii) If some of the repeated events are inputs to AND gates, then the obtained cut sets should be compared with each other and with the cut sets of Group 2, as they are generated, to eliminate any non-minimal ones.

III) Step 3 : Find the cut sets of Group 2. Group 2 is obtained from Group 1 by reinserting the previously eliminated repeated events (step 1) to obtain the minimal cut sets which compose these events. The reinsertions follow the following steps,

a) i) Generate all the possible combinations of repeated events.

ii) Discard from the list of (i) any combination which involve more than one repeated event having the same partner.

b) This step requires two main operations.

i) For a combination, consider only the cut sets of Group 1 which involves partners corresponding to events of the combination. This

operation substitutes events in that combination for their partners.

ii) Detection of non-minimal cut sets requires comparing the recently obtained subgroup of cut sets with itself and with the cut sets of the first subgroup (Group 2). This comparison includes the cut sets of Group 1 if Step 2(iii) is the case. This operation discards any cut set previously obtained.

Substitutions and comparisons are done as follows,

1) Substitute the combination of the largest number of events and test the obtained subgroup (Group 2) for non-minimal cut sets.

2) Test all the combinations from Step 3(a) and discard any one existing in Group 2.

3) Substitute the remaining combinations and discard non-minimal cut sets as they occur.

Finally, if any artificial events (Step 1 b(ii)) have been used as partners, then all cut sets containing these events are deleted.

Illustration :

Consider the simple tree given by Fussell (28) and shown in figure 3.2. The partners of the first order repeated event B are the events C and D. The first group can be obtained by considering the reduced fault tree of figure 3.4 which is composed of non-repeated events only. The TOP event of this tree may be obtained by the relation

$X0 = D.X1$ ..... (1)

but,

X1 = A.C ..... (2)

Substituting (2) into (1) yields

X0 = D.A.C

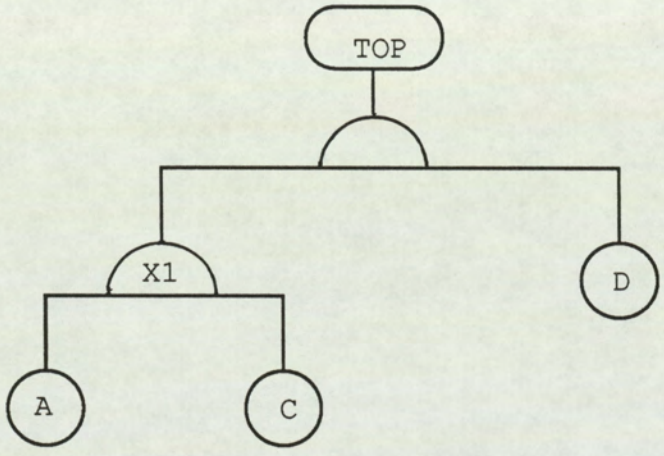


Figure 3.4 : Reduced fault tree

The first group consists, therefore, of the set of events DAC. The second group is obtained from the first group by substituting each of C or D by B. The second group consists, therefore, of the set of events BAB which can be reduced by Boolean algebra to AB. The minimal cut sets of the original fault tree are simply AB and DAC.

3.3.5.1.2 FATRAM (FAult Tree Reduction AlgorithM) :

Rasmuson and Marshall (82) have developed a new top-down algorithm,

similar to MOCUS (77) for finding minimal cut sets of fault trees. FATRAM has been used successfully as part of the Reliability Analysis System (RAS) (91). Rasmuson and Marshall claimed that the main goal for FATRAM, which has been run on the CDC Cyber-76 computer, is to obtain the minimal cut sets as quickly as possible in the smallest amount of main core memory. The steps of the algorithm are :

1. Resolution begins with the TOP event. If the TOP event is an AND gate, all inputs are listed as one set, if it is an OR gate, the inputs are listed as separate sets.

2. Iterate until all OR gates with gate inputs and all AND gates are resolved. OR gates with only basic event inputs are not resolved at this time.

3. Remove any supersets that still exist.

4. Process any repeated basic events remaining in the unresolved OR gates. For each repeated event do the following :

a) The repeated event replaces all unresolved gates of which it is an input to form new sets.

b) These new sets are added to the collection.

c) This event is removed as an input from the appropriate gates.

d) Supersets are removed.

5. Resolve the remaining OR gates. All sets are minimal cut sets.

Illustration :

Consider the same previous example. The repeated event is B. The

minimal cut sets for the fault tree in figure 3.3 is obtained as follows :

1. The TOP gate X0 is an AND gate, thus we obtain (X1.X2).

2. Gate X1 is an AND gate; thus by Rule 2, it is resolved first yielding : (A.X3.X2).

3. Both X2 and X3 are OR gates with only basic events. So the repeated event B should be processed. The repeated basic event B is an input to both the gates X2 and X3, so the new set is obtained by replacing the unresolved gates (X2) and X3 by B. This gives : (A.X3.X2),(A.B.B).

4. The repeated event B is removed as an input from the gates X2 and X3. Thus the input to the gate X2 is D and the input to the gate X3 is C. Resolving these gates gives : (A.C.D), (A.B.B)

5. The minimal cut sets for the tree are obtained by removing the redundant B giving : (A.C.D), (A.B).

3.3.5.1.3 BUP-CUTS :

Nakashima et al. (83) have developed a bottom-up algorithm called BUP-CUTS (Bottom-UP algorithm for enumerating minimal CUT Sets of fault tree) and is coded in FORTRAN. The algorithm uses the ANCHEK algorithm (83) instead of MULTIPL and ORCHEK algorithms in Bennetts' paper (39), upon transforming the logical product of two reduced SOP forms into an equivalent, reduced SOP form. The other parts of the program follow Bennetts' algorithm in principle. Nakashima et al. (83) used the following assumptions :

1. Mutually exclusive primary events are allowed to appear.

2. The same event can appear in several branches.

3. No complemented intermediate events can appear at any gate.

4. Only OR and AND gates are allowed. If logic gates, such as NOT,

Exclusive OR, NAND, appear in the fault tree, then the algorithm can be applied after transforming it into an equivalent fault tree containing only OR and AND gates by using inversion operations by De Morgan's theorms.

5. The Boolean function for the TOP event need not be s- coherent.

Nakashima et al. (83) algorithm begins with primary events and repeats, until reaching the TOP event, the process of expanding the logical product (AND gate) or sum (OR gate) of reduced SOP form for two causative (either primary or intermediate) events into a SOP form by the distribution rule and then discarding redundant terms by applying the idempotence and absorption rules to yield an equivalent reduced SOP form. This structure is known as a bottom-up algorithm (39,41). Nakashima et al. (83) pointed out that the logical combination of two reduced forms having n1 and n2 terms yields a SOP form having n1 x n2 terms (for a product) or n1 + n2 terms (for a sum) by applying the distribution rule; thus there is a sharp increase of terms in the expansion of any combination. All pairs of terms must be checked against each other by applying the idempotence and absorption rules. Since, for a product, the total number of pairs of n1 x n2 terms is (n1 x n2) x (n1 x n2 - 1), the number of checks by two rules becomes greater as the total number of terms increases. Thus the checking for products could dominate the computation time, especially for a fault tree having AND gates near the top. Process plants, particularly those with repairable items, tends to have AND gates towards the top of the fault tree (92,93,94,95). The conventional bottom-up algorithms (39,41) do not take notice of this fact. Accordingly, Nakashima et al. desired to shorten the computation time of expanding and checking for logical product by focusing on this point, and explained this as follows :

Consider the process of obtaining T a reduced SOP form from the logical product (T1.T2), where T1 consists of the union of sets c11, c12,... c1n and T2 consists of the union of sets c21, c22,...c2m. Each primary event either common primary event (primary event appearing in both sets of T1 and T2) or non-common primary event (primary event appearing in only the sets of T1). The algorithm for obtaining the T sets contained in T1 and T2 is based on the following principles :

1. If c contains only non-common primary events, then c is always an element of T1. Thus it is not necessary to check c at all.

2. If c contains at least one common primary event, then only elements of a subset (c1i U c2j), where i and j are the ith set and the jth set respectively) are required to check C.

The use of these principles appreciably decreases the number of checks. The theoretical limit to the number of checks by Nakashima et al. algorithm is (n1 x n2) x (n1+n2-1) while that by Bennetts' algorithm (48) is (n1 x n2) x (n1 x n2 - 1).

Nakashima et al. try not to compare their algorithm with Worrell's algorithm (41) and conventional top-down algorithms (81,88) for the following reasons :

1. Worrell's algorithm is based on the same principle as Bennetts' algorithm.

2. At each intermediate stage of implementation of the algorithm, a top-down algorithm has a SOP form containing not only primary events but also intermediate events. Accordingly, the combined treatment of primary events and intermediate events is required at each intermediate stage.

The BUP-CUTS is efficient but the efficiency becomes noticeable with

the decreasing number of common (repeated) primary events. This means that the more repeated events in the tree the less efficient BUP-CUTS will be and the more processing time is required. Nakashima et al. have used the Bit Manipulation Technique (88) in order to reduce both the computation time and storage requirement.

3.3.6 Disjoint Techniques :

The problem behind the direct substitution into the Boolean expressions of a probabilistic expression is that the Boolean domain does not map directly into the probabilistic domain. This is because the Boolean domain is governed by the Boolean theorems and axioms while the probabilistic domain is governed by the probability laws and axioms. The differences are shown in table 3.3 (76).

3.3.6.1 The Karnaugh Map :

The Karnaugh map (K-map) was developed to order and displays the implicants in a geometrical pattern such that the application of the logic Boolean adjacency theorem becomes obvious (96). The K-map was originally a geometric layout of the Truth-Table. For the purpose of illustration, consider the following SOP expression AB. The K-map plotted for the canonical form (truth table) of this expression is plotted in figure 3.5. The truth table shows that if both A and B are true then the expression is a true otherwise it is false.

The K-maps 3.5-c and 3.6-c show the conditions of each individual cells in terms of the events names. The symbol $\bar{}$ indicates INVERT or the opposite of

Table 3.3 : Comparison of probabilities and Boolean Algebras (76)

Feature	Boolean domain	Probability domain
1. Map	Karnaugh map (K-map)	Probability map (P-map)
2. Variables	Boolean	Probabilities
3. Data manipulations	Boolean axioms and theorems	Probability axioms and theorems
4. Variables can take values of	Discrete values of 0 and 1 corresponding to true and false	Continuous values ranging from 0 to 1 with the extremes corresponding to wholly unreliable and wholly reliable respectively
5. Operators	OR (+) AND (.) INVERT ($\bar{}$)	Plus (+) Minus (-) Multiply (x) Divide (/) invert ($\bar{R} = 1-R$)

Truth table

AB	E
00	0
01	0
10	0
11	1

(a)

B \ A	0	1
0	0	0
1	0	1

(b)

B \ A	0	1
0	$\bar{A}\bar{B}$	$A\bar{B}$
1	$\bar{A}B$	AB

(c)

Figure 3.5 : AB

Truth table

ABC	E
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	0
1 1 0	1
1 1 1	1

(a)

C \ AB	00	01	11	10
0	0	1	1	1
1	1	1	1	1

(b)

C \ AB	00	01	11	10
0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}B\bar{C}$	$AB\bar{C}$	$A\bar{B}\bar{C}$
1	$\bar{A}\bar{B}C$	$\bar{A}BC$	ABC	$A\bar{B}C$

(c)

Figure 3.6 : A+B+C

the existence of the particular event. Each cell maps a specific domain. Grouping of adjacent cells can overlap certain domains. The way to group and separate the individual cells can eliminate the overlapping and thus can introduce direct probability calculation. Thus it can be seen that

$$A+B+C = \bar{A}.B.\bar{C} + A.B.\bar{C} + A.\bar{B}.\bar{C} + \bar{A}.\bar{B}.C + \bar{A}.B.C + A.B.C + \bar{A}.B.C$$

Direct substitution of probabilities into this will give the correct result since none of the product terms overlap.

3.3.6.2 The Disjoint Technique :

The usefulness of this technique is clearer when exact calculations are needed and when repeated events are involved. Bennetts (76) summarised the theory of the disjoint technique is as follows :

a) Two conjunctive Boolean terms (i.e. at least one Boolean variable appears in both terms) are disjoint if and only if one term contains at least one Boolean variable and the other term contains the same variable or variables but in its or their complemented forms. By definition the variable and its inverse's map different domains on the K-map which means that they are disjoint.

b) In general let T1 and T2 be non-disjoint product terms in a Boolean sum-of-product expression and let EC=T1/T2 be the relative complement of T1 and T2 then EC will be defined as the non-empty set of

$$EC = (Y1, Y2,.....,Yk) \qquad k=1,2,3,...$$

where Y1, Y2, ... are the missing variables from T2.

c) The disjoint collection of T1 and T2 is described by the following expression (97) :

$$T1+T2 = T1 + \overline{Y}1\ T2 + Y1\ \overline{Y}2\ T2 + Y1\ Y2\ \overline{Y}3\ T2 + ...$$
$$.. + (Y1\ Y2..Yr\text{-}1\ \overline{Y}r)\ T2 + ... + (Y1\ Y2...\overline{Y}n)\ T2 \quad ..... \quad (3.29)$$

Effectively, equation 3.29 is based on a controlled reintroduction of missing variables to individual products on a variable by variable basis. To illustrate the procedure consider the following SOP expression :

SOP = AB + AC + BCD

1) To apply the procedure refer to the first set (AB) as T1 and the second set (AC) as T2. Compare the individual remaining terms and introduce their complemented form in the second set. The ◊ indicates the set which is being disjointed from the remaining sets. The sets processed at this stage are underlined.

$$SOP = \underset{\lozenge}{\underline{AB}} + \underline{AC} + BCD$$
$$= AB + A\overline{B}C + BCD$$

Now the first set (AB) is disjointed from the second set (ABC).

2) Carry on the same procedure with the remaining sets

$$SOP = \underset{\lozenge}{\underline{AB}} + A\overline{B}C + \underline{BCD}$$

$$= AB + A\overline{B}C + \overline{A}BCD$$

3) Select the next leftmost set (ABC) and compare with all remaining sets

$$SOP = AB + \underset{\diamond}{A\bar{B}C} + \underline{\bar{A}BCD}$$

$$= AB + A\bar{B}C + \bar{A}BCD$$

No modification is necessary on this pass since they are already disjointed.

Consider another example. Here (ABC) is being disjointed from the remainder :

$$SOP = ABC + ADF + BDF$$

$$1)\ SOP = ABC + (\bar{B} + B\bar{C})\,(ADF) + (\bar{A} + A\bar{C})\,(BDF)$$

$$= ABC + A\bar{B}DF + AB\bar{C}DF + \bar{A}BDF + AB\bar{C}DF$$

Before carry on the second step, any superset or redundant sets must be removed. There is one superset in the above expression - $AB\bar{C}DF$. The reduced SOP will be :

$$SOP = ABC + A\bar{B}DF + \bar{A}BDF$$

$$2)\ SOP = ABC + \underset{\Diamond}{A\bar{B}DF} + AB\bar{C}DF + \bar{A}BDF$$

$$= ABC + A\bar{B}DF + AB\bar{C}DF + \bar{A}BDF$$

No modification is needed at this step. All the remaining sets are disjoint from each other and no redundant sets still exist. The above expression can be interpreted directly as a probabilistic expression, i.e.:

$$P_{SOP} = P(A)P(B)P(C)+P(A)P(\overline{B})P(D)P(F)+P(A)P(B)P(\overline{C})P(D)P(F)$$
$$+P(\overline{A})P(B)P(D)P(F)$$

3.3.7 Generation of Probability Expressions From Fault Trees :

The first step in fault tree evaluation is to find the structural representation of the top event in terms of the basic events. One way to accomplish this is to find the minimal cut sets. If the rate of occurrence and fault duration of all basic events are known, and the statistical dependency of each basic event is known, then the probability of the top event can be calculated (98). Chatterjee (99), using the concept of decomposition, evaluates the top event probability directly from the fault tree. Given the basic event probabilities and starting from the bottom of the tree he calculates the lowest level gates which have only basic events as inputs. These gates can now be treated as basic events and repetition of the same procedure eventually yields the top event probability. The statistical independence of basic events is assumed and only an approximate solution can be obtained by this method.

The analysis of noncoherent fault trees proceeds in a similar way. Vesely (35) made an important development to the calculation of the top event probability by introducing the Kinetic Tree Theory for fault trees containing repairable components. Cadarola and Wickenhauser (100) also developed an analytic computer program for fault tree evaluation. Cadarola (101) developed a general analytical theory to calculate the occurrence of the top event of a multistate system. He modified this method to be applicable for s-coherent systems (102).

Koen and Carnino (87) used list processing technique. The procedure relies upon the recognition of patterns, i.e. standard configurations, and replacing known sub-trees or patterns by the equivalent leaves. The process is repeated until the original tree is reduced to a single leaf - the system reliability. Chamow (103) suggests a new approach based on direct graph (digraph) and related matrix methods. The method depends in a major sense on the diagraph representations developed for the OR and AND logic elements.

Lambert (98) developed a computer code called IMPORTANCE to compute various measures and cut sets for a fault tree. The code requires as input the minimal cut sets, the failure rates and the fault duration time (the repair times) of all basic events contained in the minimal cut sets. Finally Clarotti (104) shows the limitation of minimal cut set approach in evaluating reliability of systems with repairable components. He stated that the following rules should satisfied in order to obtain reasonable results:

1. System is s-coherent.
2. Component performances are s-independent of one another.
3. Non-repairable components have increasing failure rates.
4. Repairable components have constant failure rates.
5. Repair rates are non-increasing.

3.3.8 Methods For Disjoining Cut Sets :

In coherent fault trees the logical gates are restricted to AND and OR gates and the minimal cut sets can be determined by the use of one of the previously mentioned techniques. But in non-coherent fault trees, normally of complex systems, other types of gates were included, namely NOT and EOR gates (39,89).

The concepts of minimal cut sets can be applied and the minimal cut sets will be replaced by the prime implicant sets. The qualitative evaluation of coherent and non-coherent fault trees is accomplished by the determination of minimal cut sets and prime implicant sets respectively. In order to carry out the quantitative evaluation, different techniques should be used depending on the complexity of the system, the presence of repeated events and event dependencies (105).

Bennetts (39) has used the technique of reverse polish notation in order to obtain a sum of product (SOP) Boolean expression for the TOP event in a fault tree. The TOP event SOP form is presented by the primary events only. This SOP expression can be converted into an equivalent disjoint SOP by the use of PK (Probability-Karnaugh) map method or what is later known as the disjoint technique (76).

Kumamoto and Henly (89), Locks (106,107) and Worrell et al. (108) have discussed the use of inversion form to obtain near minimal forms by modularizing and minimizing fault trees. Modules are formed by combining certain neighbouring components that have the same mutual logical dependence wherever they appear on the tree. Case (109) introduced a reduction technique to obtain a simplified SOP Boolean expression involving the inverse notation. This technique has been modified later by Gopal (110) in order to reduce the time and effort for the elimination of redundant and non-minimal cut sets.

Recently, Bennetts (39,76) developed a disjoint technique to convert the SOP Boolean expression, for reliability block diagrams and network systems, into an SOP probability expression. Aggarwal (111) commented on the effectiveness of the way of classification of primary events in each minimal cut sets. He reduces

the amount of processing by an audiovisual method which can not be implemented easily in the form of an algorithm. Jinogshing (112) also used the disjoint technique in an visual way which is implemented either directly on a fault tree or on the minimal cut sets of a fault tree. He claimed that his approach reduces the amount of computation to less than a half if it is applied directly to the fault tree. Finally, Rushdi (113) presented an algorithm to evaluate the symbolic system reliability by decomposing the system graph into two subgraphs through a minimal cut. The decomposition method inverted the resultant expression into an equivalent disjoint expression by direct application. He suggested that although the method is manual and graphical it can be computerised but he did not do it.

3.3.9 A New Combined Method :

3.3.9.1 Introduction :

Clarotti (104) has discussed the limitations of the minimal cut set approach in evaluating the reliability of systems and especially if they have repairable components. It is also the case that the quantitative evaluation of a system by examination of its minimal cut sets may not provide enough information. However, as systems become more complex and the consequences of accidents become catastrophic (114), the need for the quantitative evaluation of fault trees is becoming essential in any new technique.

Most of the present quantitative techniques are either manual (76,112,113) or algorithmic (98,100,103,115,116). Some of these algorithms require the minimal cut sets as an input for their evaluation. Nevertheless, for complex systems, determining the list of all the minimal cut sets becomes a difficult and often an impossible task. So any method that requires the list of all the minimal cut

sets as an input is restricted (117). On the other hand the direct implementation of the probability expression on a fault tree, with repeated events, or the use of Monte-Carlo simulation will make the whole very complicated. And the computation time and storage capacity become prohibitive if it is implemented on micro or even minicomputers. In addition approximate results, especially in the case of rare events in the nuclear industries, are unacceptable. So there is a need to develop a technique that has the advantages of giving the exact system reliability (or the TOP event probability), quickly and efficiently. Such a technique would use Boolean reduction techniques and the disjoint technique together throughout the processing of the fault tree. This would reduce redundancy to a minimum.

3.3.9.2 The Fault Tree Disjoint Reduction Algorithm (FTDRA):

The Fault Tree Disjoint Reduction Algorithm (FTDRA) has been developed by the author as part of the present work. The FTDRA program is applicable to coherent fault trees with repeated basic events at the present time but it can be applied to non-coherent systems as well.

The FTDRA algorithm is a top-down algorithm based on both FATRAM (82) and the disjoint technique (76) with some modification in such a way that the generation of redundant sets is minimum. Gates to be resolved and the stage of resolving are selected in such a manner that computation requirements are minimised. The steps of the algorithm are :

1. Resolution begins with the TOP event,

    a) If the TOP event is an AND gate, then all inputs are listed as one set.

    b) If the TOP event is an OR gate, then inputs are listed as separate sets.

2. Iterate until all OR gates with gate inputs and all AND gates are resolved. OR gates with only basic event inputs are not resolved at this time.

3. Remove any supersets that exist.

4. Apply the disjoint technique on the remaining sets. Each set is disjointed from the others in a sequential way by introducing the relative complemented groups to the appropriate sets. Each OR gate involved in the disjoint process will be introduced to the appropriate sets as an AND gate with its inputs inverted according to De Morgan's theorm.

5. If there are still OR gates with repeated basic events as inputs then do the following for each such gate :

a) Delete the OR gate from the set if one of its inputs is in the same set.

b) Remove any repeated event from the gate inputs if its complement does exist in the same set.

c) Otherwise process any repeated basic events remaining in the unresolved OR gate. For each repeated basic event do the following :

i) The repeated event replaces all unresolved gates of which it is an input to form new sets if the event or its complement is not already in the set.

ii) This event is removed as an input from the appropriate gates.

6. Resolve the remaining OR gates. No supersets are existed.

7. Carry on the disjoint technique again.

Illustration :

Consider the fault tree shown in figure 3.7 (82). Two repeated basic events B and C are presented in the tree. Apply the method as follows :

a) The TOP event is an AND gate then all its inputs are put in one set (Rule 1),

SOP = G1 . G2

b) G1 is an AND gate; thus,by Rule 2 it is resolved first to yield,

SOP = A.G3.G2

c) Both G3 and G2 are OR gates, but G3 has only basic event inputs, thus it is unresolved at this stage, but G2 must be resolved since it contains G4 as one of its inputs,

SOP = A.G3.(B+D+G4)

d) G4 is an AND gate and should be resolved.

SOP = A.B.G3 + A.D.G3 + A.E.G3.G5

e) Both of G3 and G5 are OR gates with basic event inputs only. No supersets (Rule 3) exist so the disjoint technique will be applied next (Rule 4). Process the first set and disjoint it from the rest,

SOP = A.B.G3 + A.$\overline{B}$.D.G3 + A.$\overline{B}$.E.G3.G5

continue the disjoint for the second set,

SOP = A.B.G3 + A.$\overline{B}$.D.G3 + A.$\overline{B}$.$\overline{D}$.E.G3.G5

f) No gate has been inverted in the disjoint process, so apply Rule (5a) on gate G3 which contains the repeated event B.

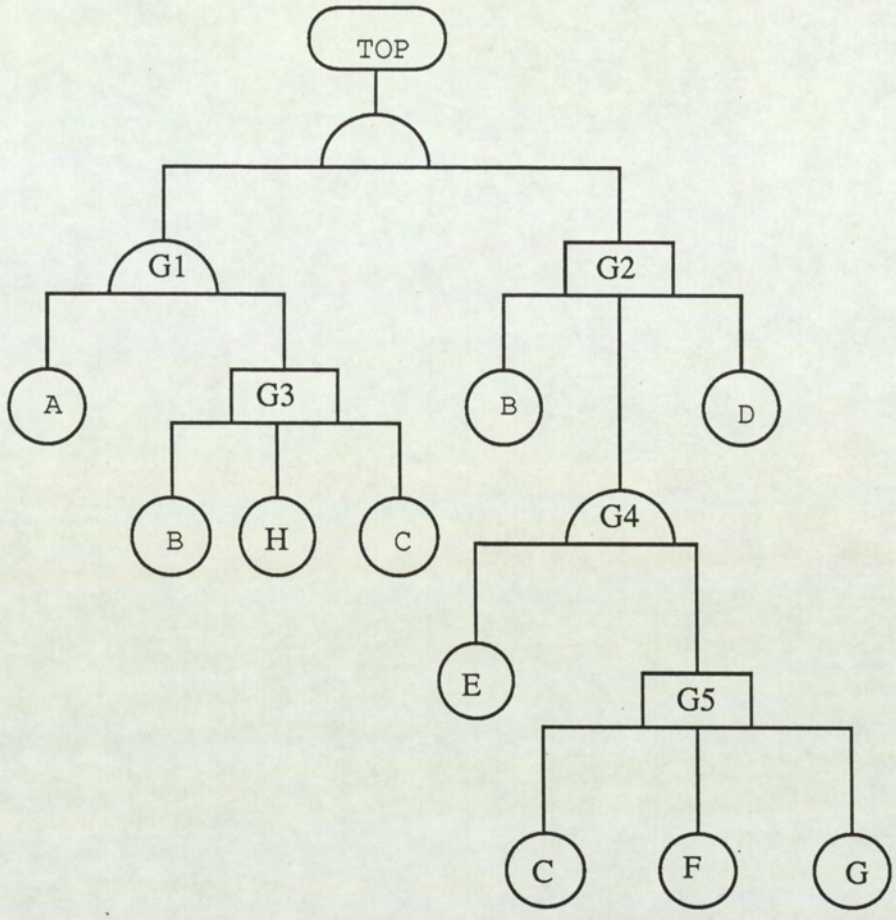


Figure 3.7 : A fault tree (82)

$$SOP = A.B + A.\overline{B}.D.G3 + A.\overline{B}.\overline{D}.E.G3.G5$$

g) Apply Rule (5b) for the two repeated events B and C. For B;

$$SOP = A.B + A.\overline{B}.D.G3_B + A.\overline{B}.\overline{D}.E.G3_B.G5$$

where $G3_B$ represents the gate G3 less the input B. No new sets have been formed but the repeated event B has been removed from the gate inputs. Do the same for C;

$$SOP = A.B + A.\overline{B}.C.D + A.\overline{B}.D.G3_{BC} + A.\overline{B}.C.C.\overline{D}.E$$

$$+ A.\overline{B}.\overline{D}.E.G3_{BC}.G5_{C}$$

In the new set $(A.\overline{B}.\overline{D}.C.\underline{C}.E)$ the redundant C is removed.

h) Resolve the remaining gates (Rule 6),

$$SOP = A.B + A.\overline{B}.C.D + A.\overline{B}.D.H + A.\overline{B}.C.\overline{D}.E + A.\overline{B}.\overline{D}.E.H.(F+G)$$

$$SOP = A.B + A.\overline{B}.C.D + A.\overline{B}.D.H + A.\overline{B}.C.\overline{D}.E + A.\overline{B}.\overline{D}.E.F.H + A.\overline{B}.\overline{D}.E.G.H$$

i) Carry on Rule (7) to complete the disjunction of all the sets in the SOP form. The first set is already disjointed so the second set is processed,

$$SOP = A.B + A.\overline{B}.C.D + A.\overline{B}.\overline{C}.D.H + A.\overline{B}.C.\overline{D}.E + A.\overline{B}.\overline{C}.\overline{D}.E.F.H$$

$$+ A.\overline{B}.\overline{C}.\overline{D}.E.G.H$$

the third and the fourth sets are already disjointed. Finally disjoin the fifth set from the last one,

$$SOP = A.B + A.\overline{B}.C.D + A.\overline{B}.\overline{C}.D.H + A.\overline{B}.C.\overline{D}.E + A.\overline{B}.\overline{C}.\overline{D}.E.F.H$$

$$+ A.\overline{B}.\overline{C}.\overline{D}.E.\overline{F}.G.H$$

the last expression can be interpreted as a probability expression for the TOP event.

$$P(TOP) = P(A)P(B)+P(A)P(\overline{B})P(C)P(D)+P(A)P(\overline{B})P(\overline{C})P(D)P(H)+$$

$$P(A)P(\overline{B})P(C)P(\overline{D})P(E)+P(A)P(\overline{B})P(\overline{C})P(\overline{D})P(E)P(F)P(H)+$$

$$P(A)P(\overline{B})P(\overline{C})P(\overline{D})P(E)P(\overline{F})P(G)P(H)$$

More applications will be discussed in the next chapter.

CHAPTER FOUR

4. APPLICATIONS OF THE FTDRA APPROACH

4.1 Introduction :

The technique used in the FTDRA algorithm is a top-down technique combined with the disjoint techinque in order to get the exact TOP event probability when repeated basic events exist. The FTDRA approach tries to remove redundancy at an early stage by cutting the effect of repeated basic events or repeated gates, with basic event inputs, before the full expansion of those tree gates which have only basic event inputs. This is not the only factor affecting the speed of processing and the amount of storage required. The tree size, the relative number of repeated events to the total number of events in the tree and the tree configuration are also important factors in the speed with which the exact probability of the TOP event is obtained. This can be shown by the following examples taken from the literature.

4.2 Example 1 :

Consider the fault tree given by Vesely (36) shown in figure 4.1 which contains 58 total events, 10 basic events including 9 repeated basic events. Using the FTDRA technique the solution is as follows:

1) SOP = G0 = G1 + G2 = G3 + G4 + G5 + G6

= G7 + G8 + G9 + G10 + G11 + G12 + G13 + G14 + G3.G15

= 1.2 + 1.3 + 2.3 + 1.2.G16 + G18 . G19 + 4.5 + 4.6 + 5.6 + G7.G15 + G8.G15 + G9.G15

= 1.2 + 1.3 + 2.3 + 1.2.G16 + G18.G20 + G18.G21 + 4.5 + 4.6 + 5.6 + 1.2.G15 + 1.3.G15 + 2.3.G15

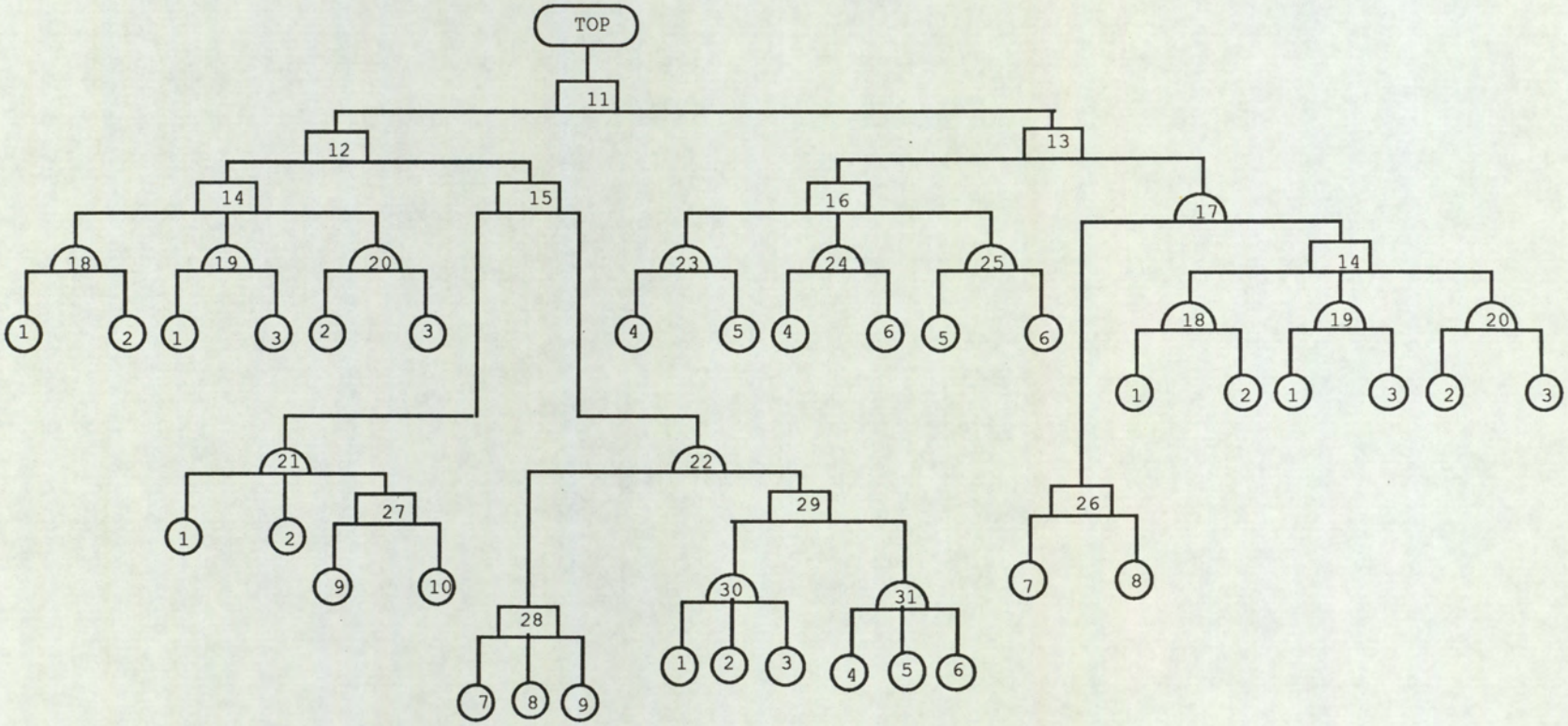


Figure 4.1 : Example 1 (36)

G0 = 11 G1 = 12 G2 = 13 G3 = 14 G20 = 31

G4 = 15 G5 = 16 G6 = 17 G7 = 18

G8 = 19 G9 = 20 G10 = 21 G11 = 22

G12 = 23 G13 = 24 G14 = 25 G15 = 26

G16 = 27 G17 = 28 G18 = 29 G19 = 30

$= 1.2 + 1.3 + 2.3 + 4.5 + 4.6 + 5.6 + 1.2.G15 + 1.3.G15 + 2.3.G15 +$

$1.2.G16 + 1.2.3.G18 + 4.5.6.G18$

2) Removing the supersets yields

$1.2 + 1.3 + 2.3 + 4.5 + 4.6 + 5.6$

3) Applying the disjoint technique to the 6 minimal cut sets yields

$1.2 + 1.2.\overline{3} + 1.\overline{2}.3 + 1.\overline{2}.\overline{4}.5 + 1.\overline{2}.3.\overline{4}.5 + 1.2.\overline{3}.\overline{4}.5 + \overline{1}.\overline{2}.4.\overline{5}.6 + \overline{1}.2.\overline{3}.4.\overline{5}.6 +$

$1.\overline{2}.\overline{3}.4.\overline{5}.6 + \overline{1}.\overline{2}.\overline{4}.5.6 + \overline{1}.2.\overline{3}.\overline{4}.5.6 + 1.\overline{2}.\overline{3}.\overline{4}.5.6$

The above result can be interpreted directly as a probability expression for the TOP event.

4.3 Example 2 :

Consider the fault tree shown in figure 4.2 which has been used by Bengiamin et al.(86) and represents a failure in the protection system of a nuclear power plant. The tree contains a total of 43 events. There are 3 repeated basic events and 8 non-repeated basic events. Using the FTDRA technique the solution is as follows:

1. According to Rule 1 the SOP will be equal to

$SOP = G0 = G1.G2$

2. Resolving all AND gates and all OR gates with gate inputs according to Rule 2 and further removing any supersets according to Rule 3 yields

$D.E.G15.G16 + D.E.G12.G15 + D.E.G12.G16 + D.G9.G13.G15.G16 +$

$D.G9.G14.G15.G16 + D.G13.G14.G15.G16 + D.G9.G12.G13.G15 +$

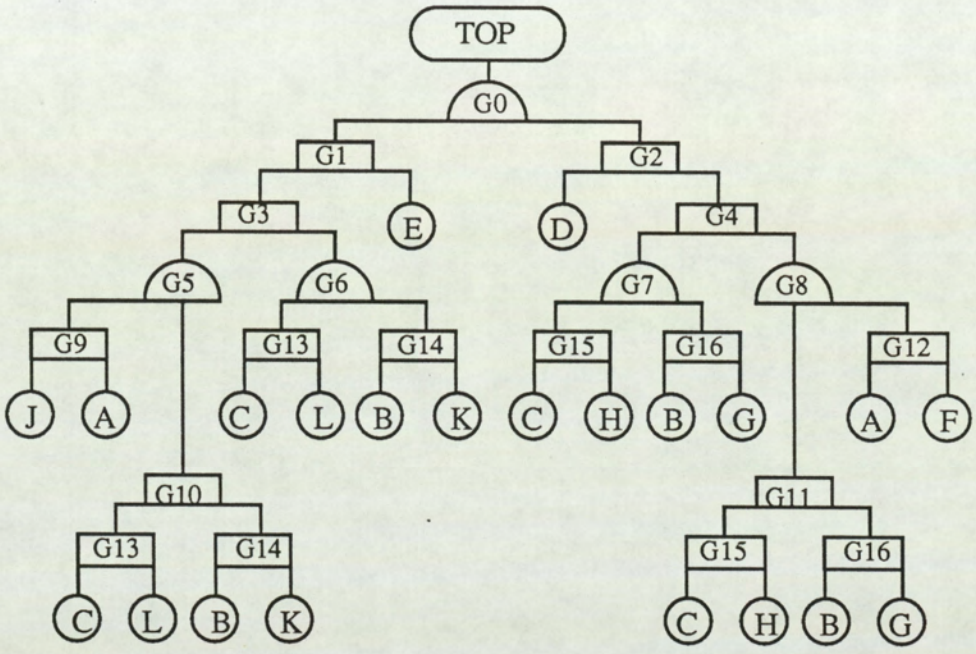


Figure 4.2 : Example 2 (86)

D.G9.G12.G13.G16 + D.G9.G12.G14.G15 + D.G9.G12.G14.G16 + D.G12.G13.G14.G15 + D.12.G13.G14.G16

3. Carry out Rule 4 to disjoin the individual sets from each other. This yields

$D.E.G15.G16+\overline{B}.D.E.\overline{M}.G12.G15+\overline{C}.D.E.\overline{H}.G12.G16+D.\overline{E}.G9.G13.G15.G16$

$+\overline{C}.D.\overline{E}.\overline{L}.G9.G14.G15.G16+\overline{A}.D.\overline{E}.\overline{J}.G13.G14.G15.G16$

$+\overline{B}.D.\overline{E}.\overline{M}.G9.G12.G13.G15+\overline{C}.D.\overline{E}.\overline{H}.G9.G12.G13.G16$

$+\overline{B}.\overline{C}.D.\overline{E}.\overline{M}.\overline{L}.G9.G12.G14.G16+ \overline{C}.D.\overline{E}.\overline{H}.\overline{L}.G9.G12.G14.G16$

$+ \overline{A}.\overline{B}.D.\overline{E}.\overline{M}.\overline{J}.G12.G13.G14.G15 + \overline{A}.\overline{C}.D.\overline{E}.\overline{H}.\overline{J}.G12.G13.G14.G16$

4. Apply the rest of the rules to get the disjointed result as a probability expression for the TOP event

$SOP_P= D.E.M.H + C.D.E.M.\overline{H} + B.D.E.\overline{M}.H + B.C.D.E.\overline{M}.\overline{H} + \overline{B}.D.E.\ F.\overline{M}.H$

$+ \overline{B}.C.D.E.F.\overline{M}.\overline{H} + A.\overline{B}.D.E.\overline{F}.\overline{M}.H + A.\overline{B}.C.D.E.\overline{F}.\overline{M}.\overline{H} + \overline{C}.D.E.F.M.H +$

$B.\overline{C}.D.E.F.\overline{M}.\overline{H} + A.\overline{C}.D.E.\overline{F}.M.\overline{H} + A.B.\overline{C}.D.E.\overline{F}.\overline{M}.\overline{H} + D.\overline{E}.M.H.J.L +$

$C.D.\overline{E}.M.\overline{H}.J + C.D.\overline{E}.M.H.J.\overline{L} + B.D.\overline{E}.\overline{M}.H.J.L + B.C.D.\overline{E}.\overline{M}.\overline{H}.J +$

$B.C.D.\overline{E}.\overline{M}.H.J.\overline{L} + A.D.\overline{E}.M.H.\overline{J}.L + A.C.D.\overline{E}.M.\overline{H}.\overline{J} + A.C.D.\overline{E}.M.H.\overline{J}.\overline{L} +$

$A.B.D.\overline{E}.\overline{M}.H.\overline{J}.L + A.B.C.D.\overline{E}.\overline{M}.\overline{H} + A.B.C.D.\overline{E}.\overline{M}.H.\overline{J}.\overline{L} +$

$\overline{C}.D.\overline{E}.M.H.J.K.\overline{L} + B.\overline{C}.D.\overline{E}.\overline{M}.H.J.\overline{L} + B.\overline{C}.D.\overline{E}.M.H.J.\overline{K}.\overline{L} +$

$A.\overline{C}.D.\overline{E}.M.H.\overline{J}.K.\overline{L} + A.B.\overline{C}.D.\overline{E}.\overline{M}.H.\overline{J}.\overline{L} + A.B.\overline{C}.D.\overline{E}.M.H.\overline{J}.\overline{K}.\overline{L} +$

$\overline{A}.B.C.D.\overline{E}.\overline{J} + \overline{A}.\overline{B}.C.D.\overline{E}.M.\overline{J}.K + \overline{A}.B.\overline{C}.D.\overline{E}.H.\overline{J}.L + \overline{A}.\overline{B}.D.\overline{E}.M.H.\overline{J}.K.L +$

$\overline{B}.D.\overline{E}.F.\overline{M}.J.K.L + A.\overline{B}.D.\overline{E}.\overline{F}.\overline{M}.H.L + A.\overline{B}.D.\overline{E}.F.\overline{M}.H.\overline{J}.L +$

$A.\overline{B}.D.\overline{E}.F.\overline{M}.H.J.\overline{K}.L + A.\overline{B}.C.D.\overline{E}.\overline{F}.\overline{M} + A.\overline{B}.C.D.\overline{E}.F.\overline{M}.\overline{J} +$

$A.\overline{B}.C.D.\overline{E}.F.\overline{M}.J.\overline{K} + A.\overline{B}.C.D.\overline{E}.F.\overline{M}.J.K.\overline{L} + \overline{C}.D.\overline{E}.F.M.\overline{H}.J.L +$

$B.\overline{C}.D.\overline{E}.\overline{M}.\overline{H}.J.L + A.\overline{C}.D.\overline{E}.\overline{F}.M.\overline{H}.L + A.\overline{C}.D.\overline{E}.F.M.\overline{H}.\overline{J}.L +$

$A.B.\overline{C}.D.\overline{E}.\overline{F}.\overline{M}.\overline{H}.L + A.B.\overline{C}.D.\overline{E}.F.\overline{M}.\overline{H}.\overline{J}.L + \overline{C}.D.\overline{E}.F.M.\overline{H}.J.K.\overline{L} +$

$B.\overline{C}.D.\overline{E}.F.\overline{M}.\overline{H}.J.\overline{L} + B.\overline{C}.D.\overline{E}.F.M.\overline{H}.J.\overline{K}.\overline{L} + A.\overline{C}.D.\overline{E}.\overline{F}.M.\overline{H}.K.\overline{L} +$

$A.\overline{C}.D.\overline{E}.F.M.\overline{H}.\overline{J}.K.\overline{L} + A.B.\overline{C}.D.\overline{E}.\overline{F}.\overline{M}.\overline{H}.\overline{L} + A.B.\overline{C}.D.\overline{E}.\overline{F}.M.\overline{H}.\overline{K}.\overline{L} +$
$A.B.\overline{C}.D.\overline{E}.F.\overline{M}.\overline{H}.\overline{J}.\overline{L} + A.B.\overline{C}.D.\overline{E}.F.M.\overline{H}.\overline{J}.\overline{K}.\overline{L} + \overline{A}.\overline{B}.C.D.\overline{E}.F.\overline{M}.\overline{J}.K +$
$\overline{A}.\overline{B}.\overline{C}.D.\overline{E}.F.\overline{M}.H.\overline{J}.K.L + \overline{A}.B.\overline{C}.D.\overline{E}.F.\overline{H}.\overline{J}.L + \overline{A}.\overline{B}.\overline{C}.D.\overline{E}.F.M.\overline{H}.\overline{J}.K.L$

where SOPp means the probability of the TOP event.

4.4 Example 3 :

Consider the fault tree shown in figure 4.3 which has been modified by cutting two levels from the bottom of the tree from a similar tree originally given by Camarda (55). This represents the failure of an electric power supply system. The total number of events despite repetition is 57. The number of basic events is 19 of which 7 basic events are repeated. Also there are 14 repeated gates in the tree. The complete application of the FTDRA technique will not be shown because of the long expressions resulting from the application of the appropriate rules. The analysis will be carried out up to the end of rule 4. This can be shown as follows :

1) Apply Rule 1 to the TOP event

SOP = G22 = G20.G21

2) Apply Rule 2 to resolve all OR gates with gate inputs and all AND gates should be resolved. Then remove any supersets still exist according to Rule 3. This will yield

8.9 + 8.19 + 8.22 + 8.23 + 4.8.G7.G8 + 7.8.G7.G8 +8.17.G7.G8+ 8.G3.G7.G8 + 9.18 + 18.19 + 18.22 + 18.23 + 4.18.G7.G8 + 7.18.G7.G8 + 17.18.G7.G8+ 18.G3.G4.G7.G8 + 19.20.21 + 21.22 + 21.23 + 4.21.G7.G8 + 7.21.G7.G8 +

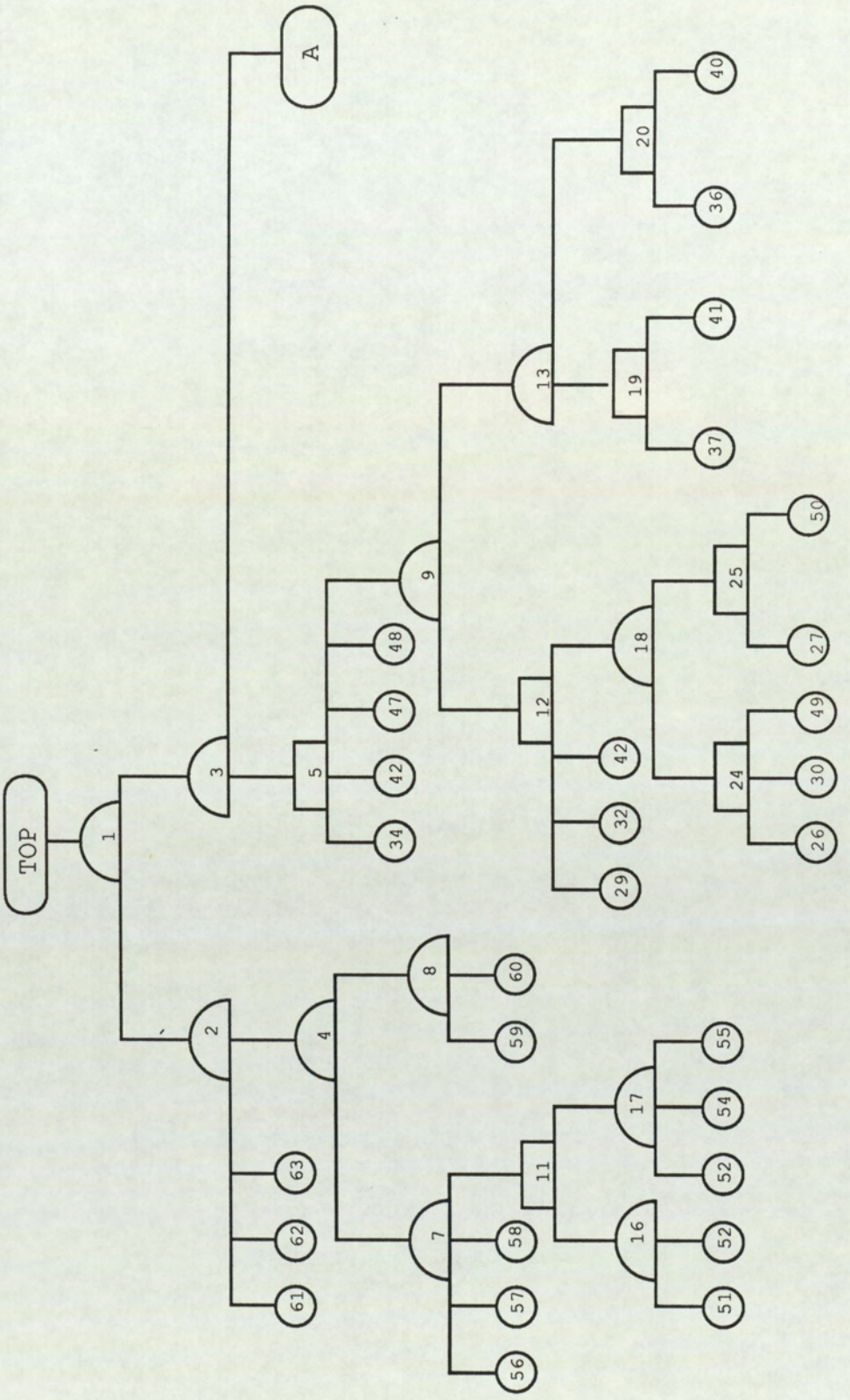


Figure 4.3 : Example 3 (55)

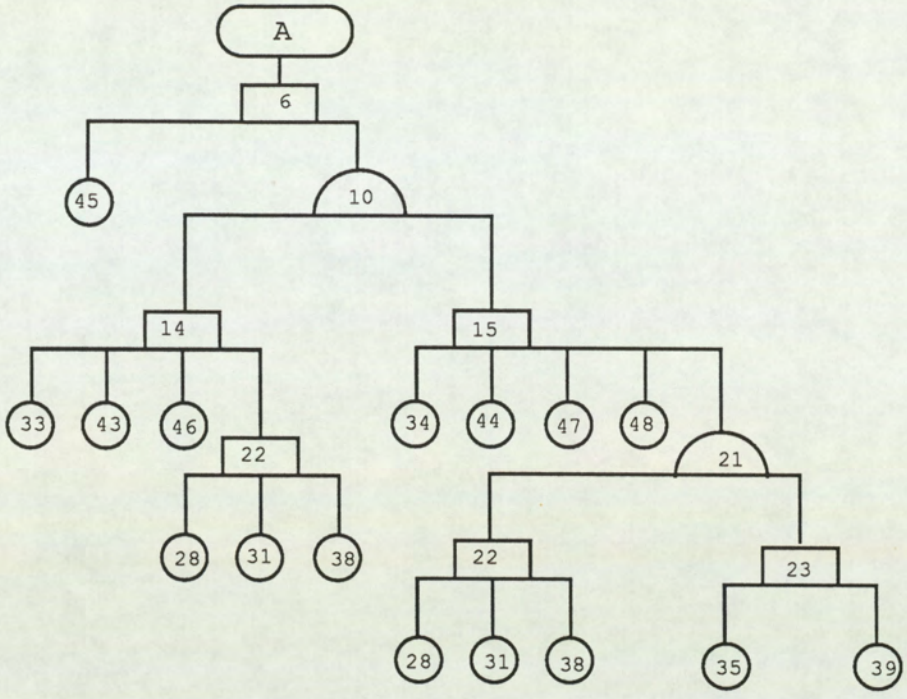


Figure 4.3 : Example 3 (continue)

17.21.G7.G8 + 21.G3.G4.G7.G8 + 3.9.20.G13 + 6.9.20.G13 + 9.13.20.G13 +9.20.G1.G2.G13 + 3.19.20.G13 + 6.19.20.G13 + 13.19.20.G13 +19.20.G1. G2.G13+ 3.4.G7.G8.G13 + 3.7.G7.G8.G13 + 3.17.G7.G8.G13 + 3.G3.G4.G7 .G8.G13 + 4.6.G7.G8.G13 + 6.7.G7.G8.G13 + 6.17.G7.G8.G13 + 6.G3.G4. G7.G8.G13 + 4.13.G7.G8.G13 + 7.13.G7.G8.G13 + 13.17.G7.G8.G13 +13.G3 .G4.G7.G8.G13 + 4.G1.G2.G7.G8.G13 + 7.G1.G2.G7.G8.G13 +17.G1.G2. G7.G8.G13 + G1.G2.G3.G4.G7.G8.G13

The above SOP, after the deletion of 118 supersets, contains OR gates with basic event inputs only.

3) Apply Rule 4 to disjoint the individual terms of the above expression. This yields

SOP = 14.17 + 14.$\overline{17}$.18 +14.$\overline{17}$.$\overline{18}$.19 +14.$\overline{17}$.$\overline{18}$.$\overline{19}$.20 +$\overline{14}$.15.17 +$\overline{14}$.15.$\overline{17}$.18 + $\overline{14}$.15.$\overline{17}$.$\overline{18}$.19 + $\overline{14}$.15.$\overline{17}$.$\overline{18}$.$\overline{19}$.20 + $\overline{14}$.$\overline{15}$. 16.17+ $\overline{14}$.$\overline{15}$.16.$\overline{17}$.18 + $\overline{14}$.$\overline{15}$. 16.$\overline{17}$.$\overline{18}$.19 + $\overline{14}$.$\overline{15}$.16.$\overline{17}$.$\overline{18}$.$\overline{19}$. 20 + 5.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.17.21 + $\overline{5}$.6.13.$\overline{14}$.$\overline{15}$.$\overline{16}$ .17.21+$\overline{5}$.$\overline{6}$.9.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.17.21+1.2.$\overline{5}$.$\overline{6}$.$\overline{9}$.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.17.21+5.13.$\overline{14}$.$\overline{15}$.$\overline{16}$. $\overline{17}$.18.21+$\overline{5}$.6.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.18.21 + $\overline{5}$.$\overline{6}$.9.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.18.21 + 1.$\overline{2}$.$\overline{5}$.$\overline{6}$.9 .$\overline{13}$.$\overline{14}$.$\overline{15}$.$\overline{16}$.17.18.21 +5.$\overline{13}$.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.18.19.21 + $\overline{5}$.6.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$. 19.21 + $\overline{5}$.$\overline{6}$.9.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.19.21 +1.2.$\overline{5}$.$\overline{6}$.$\overline{9}$.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.19.21 + 5.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.$\overline{19}$.20.21 + $\overline{5}$.6.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.$\overline{19}$.20.21 + $\overline{5}$.$\overline{6}$.9.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.$\overline{19}$.20.21 + 1.2.$\overline{5}$.$\overline{6}$.$\overline{9}$.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.$\overline{19}$.20.21+5.13.$\overline{14}$.15.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$+$\overline{5}$.6.13.$\overline{14}$.15.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$+$\overline{5}$.$\overline{6}$.9.13.$\overline{14}$.15.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$+1.2.$\overline{5}$.$\overline{6}$.$\overline{9}$.13 .$\overline{14}$.15.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$ + 5.13.$\overline{14}$.$\overline{15}$.16.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$ + $\overline{5}$.6.13.$\overline{14}$.$\overline{15}$.16.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$ + $\overline{5}$.$\overline{6}$.13.$\overline{14}$.$\overline{15}$.16.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$ +1.2.$\overline{5}$.$\overline{6}$.$\overline{9}$.13.$\overline{14}$.$\overline{15}$.16.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$+5.13.14.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$+$\overline{5}$.6.13.14.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$+$\overline{5}$.$\overline{6}$.9.13.14.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$+1.2.$\overline{5}$.$\overline{6}$.$\overline{9}$.13.14.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$ +5.7.8.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$.G5 + 5.7.8.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.17.$\overline{21}$.G5 + 5.7.8.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.18.$\overline{21}$.G5 + 5.7.8.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.19.$\overline{21}$.G5 + 5.7.8.13 .$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$.G5 + $\overline{5}$.6.7.8.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.18.$\overline{21}$.G5 + $\overline{5}$.6.7.8.13.$\overline{14}$. $\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.19.$\overline{21}$.G5 + $\overline{5}$.6.7.8.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.$\overline{19}$.20.21.G5 + $\overline{5}$.$\overline{6}$.7.8.9 .13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$.G5 + $\overline{5}$.$\overline{6}$.7.8.9.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.18.$\overline{21}$.G5+$\overline{5}$.$\overline{6}$.7.8. 9.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.19.$\overline{21}$.G5+$\overline{5}$.$\overline{6}$.7.8.9.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.$\overline{19}$.20.$\overline{21}$.G5+1.2. $\overline{5}$.$\overline{6}$.$\overline{9}$.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$.G5+1.2.$\overline{5}$.$\overline{6}$.$\overline{9}$.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.18.$\overline{21}$.G5+1.2.$\overline{5}$.$\overline{6}$.$\overline{9}$.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.19.$\overline{21}$.G5+1.2.$\overline{5}$.$\overline{6}$.$\overline{9}$.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.$\overline{19}$.20.$\overline{21}$.G5+1.2. 5.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$.G5+1.2.5.$\overline{7}$.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.17.$\overline{21}$.G5+1.2.5.7.$\overline{8}$.13.$\overline{14}$. $\overline{15}$.$\overline{16}$.17.$\overline{21}$.G5+1.2.5.$\overline{7}$.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.18.$\overline{21}$.G5+1.2.5.7.$\overline{8}$.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$. 18.$\overline{21}$.G5+1.2.5.$\overline{7}$.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.19.$\overline{21}$.G5+1.2.5.7.$\overline{8}$.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$. 19.$\overline{21}$.G5+1.2.5.$\overline{7}$.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$.$\overline{18}$.$\overline{19}$.20.$\overline{21}$.G5+1.2.5.7.$\overline{8}$.13.$\overline{14}$.$\overline{15}$.$\overline{16}$.$\overline{17}$. $\overline{18}$.$\overline{19}$.20.$\overline{21}$.G5+$\overline{1}$.$\overline{5}$.$\overline{6}$.7.8.$\overline{9}$.14.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$.21.G5+5.7.8.$\overline{13}$.14.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$.

21.G5+1.$\overline{2}$.$\overline{5}$.$\overline{6}$.7.8.$\overline{9}$.14.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$.21.G5+1.2.$\overline{5}$.$\overline{6}$.7.8.$\overline{9}$.$\overline{13}$.14.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$.
21.G5 + $\overline{1}$.$\overline{5}$.$\overline{6}$.7.8.$\overline{9}$.$\overline{14}$.15.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$.21.G5+ 5.7.8.$\overline{13}$.$\overline{14}$.15.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$.
21.G5+1.$\overline{2}$.$\overline{5}$.$\overline{6}$.7.8.$\overline{9}$.$\overline{14}$.15.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$.21.G5+1.2.$\overline{5}$.$\overline{6}$.7.8.$\overline{9}$.$\overline{13}$.$\overline{14}$.15.$\overline{17}$.$\overline{18}$.
$\overline{19}$.$\overline{20}$.21.G5+$\overline{1}$.$\overline{5}$.$\overline{6}$.7.8.$\overline{9}$.$\overline{14}$.$\overline{15}$.16.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$.21.G5+5.7.8.$\overline{13}$.$\overline{14}$.$\overline{15}$.16.$\overline{17}$.
$\overline{18}$.$\overline{19}$.$\overline{20}$.21.G5+1.$\overline{2}$.$\overline{5}$.$\overline{6}$.7.8.$\overline{9}$.$\overline{14}$.$\overline{15}$.16.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$.21.G5+1.2.$\overline{5}$.$\overline{6}$.7.8.$\overline{9}$.$\overline{13}$.
$\overline{14}$.$\overline{15}$.16.$\overline{17}$.$\overline{18}$.$\overline{19}$.$\overline{20}$.21.G5

The above expression consists of 75 terms. Only 38 of the terms contain the unresolved gate G5. Using the FATRAM technique, if a gate has at least one of its inputs as a repeated basic event then it will be involved in the treatment process for that particular repeated event. Otherwise, it will not be involved in any processing and can be resolved as usual. This means that gate G5 can be resolved normally since all its inputs are non-repeated basic events. The resolution of gate G5 will increase the total number of terms to 151. Also since gate G5 has not been involved in the disjoint process through its presence in the sets, its elements will also not need to be disjointed again and they will replace the gate position in the particular sets.

As a general rule a gate needs to be disjointed only once if and only if any of its inputs have not been repeated or have not occurred before its resolution. This rule has not been applied in the FTDRA technique at the present time and will be left for future work: modification is required to the present software. In addition an evaluation is required to justify the saving in processing time.

4.5 The Complexity Of Computation :

The complexity of computation can be estimated by the time required to

compute a given tree. Of course it is also dependent upon the size of the computed tree, upon the number of gate levels and upon the number of repeated basic events and their locations in the tree. The size and the number of sets to be handled at each stage of the technique should be kept to a minimum in order to reduce the required time for comparison and searching for supersets. In addition, the intermediate events, i.e. gates, must be resolved as quickly as possible since the top event probability is represented in terms of basic events. Usually OR gates slow down analysis more than AND gates do. This is because OR gates tend to create additional sets for each of their inputs while AND gates do not. From the above examples it is clear that the FTDRA technique makes, in comparison with existing techniques, savings in computation time and cuts the redundant terms gates before resolving them. This is clearly shown in example 1.

CHAPTER FIVE

5. HARDWARE AND SOFTWARE TOOLS

5.1 Introduction :

The hardware used in carrying out the work is the ICL Perq 1 personal computer (119). The Perq has some features that give it some advantages over mainframe computers as well as some other microcomputers. One of the ways in which the personal computer differs from the mainframe computer is that in carrying out a real time analysis the personal computer is in use by one user at a time and this means that processor time is not shared. In the case of the mainframe computer this is not the case. The microcomputer is now sufficiently powerful to carry out many of the analyses required by the engineer. Most of today's microcomputers have a large RAM, a hard disc, good compilers, compact size and are easily fitted near process equipment or in control rooms. This makes them ideal for use in process control analysis especially when user interference is essential.

5.2 Hardware :

The Perq 1 consists of five main components. These are : the keyboard, the high resolution screen, the floppy disc drive, the processor box and the graphics tablet with four button cursor. Figure 5.1 shows these components.

5.2.1 The Keyboard :

One of the weak features of the Perq 1 is its keyboard. The keyboard has

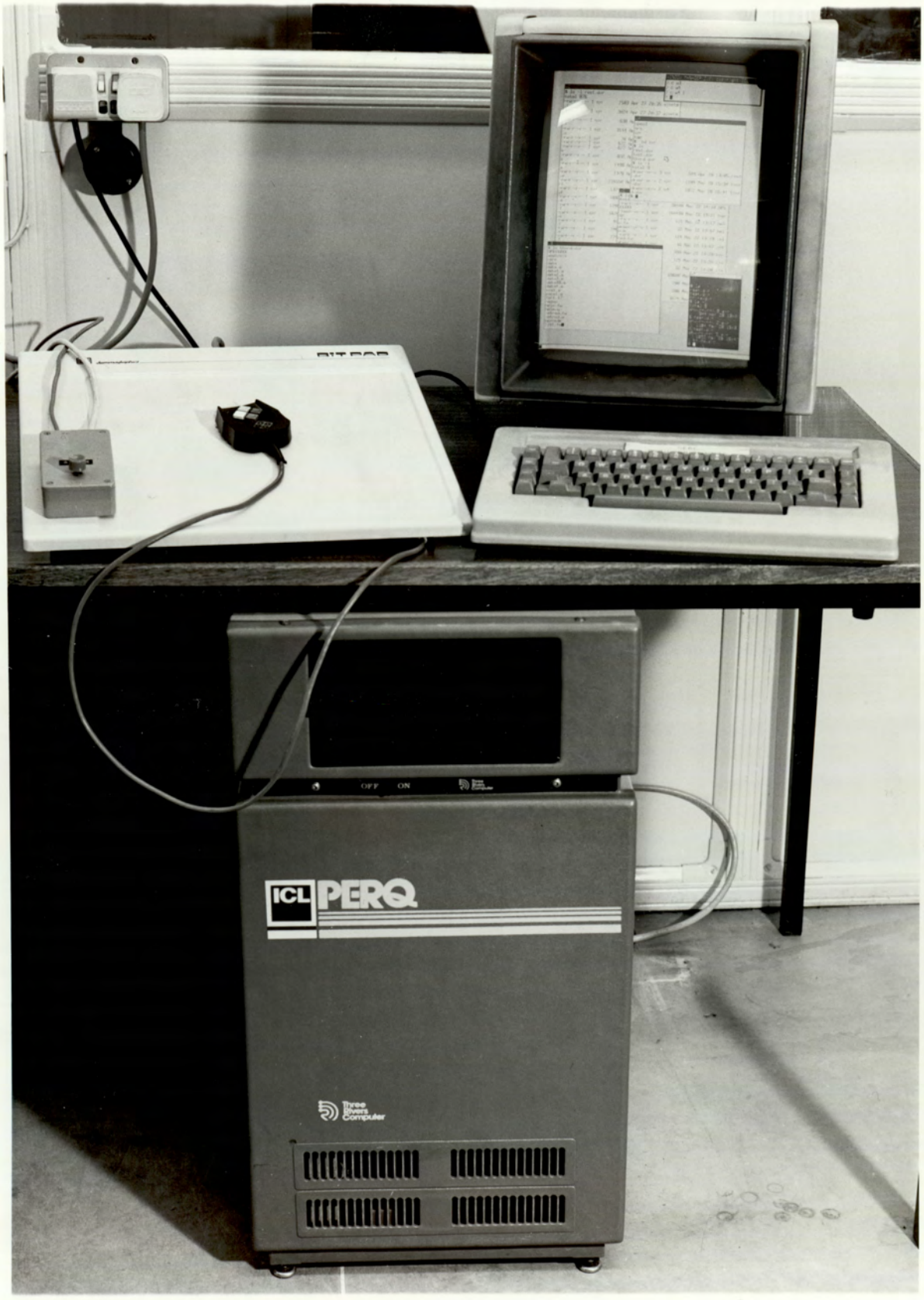


Figure 5.1 : Perq1 workstation

only the basic alphabetic keys and some but not all the control keys that can be found in most of today's terminals or microcomputers. It has a Carriage Return key, Control key, Backspace and Delete keys, Shift and Shift-Lock keys and finally a Help key. In order to enable the Perq to communicate with other computers through its RS232 port, a locally made Break key has been attached to the asynchronous port of the Perq. Although the Perq's compilers accept upper and lower case letters and treat them as the same, the keyboard is unsuitable for text or program typing because it lacks a Lock Capitals key and other word processing facilities. A Lock Capitals key differs in its functioning from that of the Shift-Lock key. Usually the Lock Capitals key locks the alphabetical keys only. It shifts letters from lower case to upper case. The Shift-lock key on the Perq's keyboard locks all the keys on the keyboard to their upper characters. Also the lack of some sort of special function keys on the Perq's keyboard makes the typing of a long program a lengthy business. For example separate numeric keys and the four arithmetic operation keys simplify the typing of text mixed with numeric. Despite the presence of a puck (the four button cursor) the need for cursor control keys is essential especially in communication with other computers in terminal emulation mode.

5.2.2 The High Resolution Screen :

The Perq is supplied with a rectangular monochrome high resolution screen of a 1024 x 768 pixels (27.5 cm in length x 21 cm in width).

5.2.3 The Floppy Disc Drive :

The floppy disc drive, which is built in the processor box, has a 1

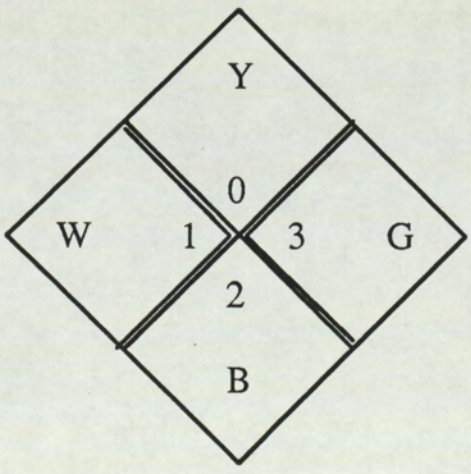
Megabyte capacity. It takes an 8 inch floppy disc.

5.2.4 The Processor Box :

The Perq has a 16 bit proprietary microprogrammable processor. It has a 24 Megabyte Winchester hard disc and 1 Megabyte of RAM. There is a general purpose interface board fixed at the back which has a special printer port and an RS232C serial port.

5.2.5 Tablet and pointing device :

The Perq is supplied with a graphics tablet and a four button cursor. The cursor is placed on the tablet for pointing. The four buttons are arranged in a diamond as shown in figure 5.2.



Y = Yellow (value = 0)

W = White (value = 1)

B = Blue (value = 2)

G = Green (value = 3)

Figure 5.2 : Four button cursor

The cursor on the screen follows the movement of the cursor on the tablet. If the cursor is in the upper-left hand corner of the tablet the screen cursor will be in the upper-left hand corner. The Perq can read the cursor position when the cursor is near the tablet surface. If one of the buttons of the cursor is pressed down, a signal will be sent to the computer. The cursor is vital for carrying out commands.

5.3 The Operating System :

5.3.1 Perq under POS :

The Perq was originally supplied with POS (the Perq Operating System). POS is written in Pascal. POS operating system supports any program written in standard pascal and in FORTRAN 77 (119).

5.3.1.1 The Shell :

POS was supplied with a program environment called the shell. When the user types a command line and presses RETURN, the line is interpreted by the shell and is executed appropriately. The Shell is the vital environment for running and executing the built-in commands.

While the Shell is running, the combined list of built-in commands and the standard Perq utilities can be displayed. Pressing any of the cursor's buttons causes a small window to appear containing an alphabtical list of the commands and the utilities.

5.3.1.2 The HELP key :

When help is required about the use of a utility, the name of the particular utility is selected from the pop-up window and the HELP key is pressed. The utility then displays information and advice about its use.

5.3.2 Software available :

5.3.2.1 Pascal :

The Perq operating system (POS) is programmed in Pascal, a language which is designed to encourage well structured and efficient software. Perq Pascal contains several extensions. These make the language more powerful and helpful for manipulation of the display. The user's own Pascal programs are translated into executable programs by the Perq Pascal compiler and the linker. Appendix E.1 shows a flow diagram of how a Pascal program is converted into an executable form.

5.3.2.2 FORTRAN 77 :

A FORTRAN 77 compiler is available for running under POS. FORTRAN programs are translated into executable programs by the FORTRAN compiler and the linker. Appendix E.2 shows a flow diagram of converting a simple FORTRAN program. Also FORTRAN 77 program can reference Pascal procedures and functions as if they are FORTRAN procedures and functions, but function and procedure names may not be passed as arguments. Appendices E.3 and E.4 show a flow diagram for compiling, consolidating and linking a FORTRAN program which references an independently compiled FORTRAN unit.

5.3.2.3 The POS Editor :

The POS editor makes extensive use of the tablet to enable the user to point to parts of the text that need correction and to display various parts of the file being edited. Help information is available to show the use of different parts of the editor.

5.3.2.4 The window manager :

By default, the whole of the display is regarded as one window, that is, a rectangular area bounded by a thin line and headed by a thick black line containing a title. Since the Perq display has such large capacity, the user will often find it valuable to divide the display into several windows. Each window can act as an input or an output device under the supervision of the window manager. POS is a single tasking operating system and therefore all input and output window devices are connected to the same single process.

5.3.3 Perq under PNX operating system :

After POS had been in use for about one year, an alternative operating system, PNX, was available.

PNX is an operating system developed from the UNIX operating system by ICL (120). UNIX is a multi-tasking operating system originally developed and produced by the Bell Laboratories for program development in a research environment. It has since become extremely popular throughout the world, particularly in universities and has begun to be accepted as an operating system for supporting applications in other environments as well. Perhaps its single most

important feature is the speed at which complex applications can be produced by connecting together simple, existing programs (121).

PNX is implemented on the Perq by microcoding, so that the Perq's apparent machine code is C-code, which is tailored extremely close to the needs of the C language, in which most of the UNIX III operating system version is written. Consequently, the PNX software has been updated to include some UNIX V and UNIX VII software (122). To access PNX on the Perq some of the hardware had to be changed in order to carry on the job. PNX on the Perq has the well-known features of the UNIX operating system, plus the advantage of running on a powerful graphics-oriented personal computer (123).

5.3.4 Software available under PNX :

5.3.4.1 Window management system :

The Perq has only one display, keyboard and tablet, and yet under PNX it could be running several linked or independent processes at the same time, each of which needs to use all the Perq components, which must therefore be shared. The Perq display can be partitioned into independent areas, known as windows, with one window being used by each process or by many processes at the same time.

5.3.4.2 Windows :

A window is that part , or whole, of the screen that is being used by a process or processes and may be linked to the input and/or output devices of the process as required by the user. A window together with a share of the input

devices is considered to be a virtual device under PNX. Many windows may be visible at the same time. They may overlap, in which case one or more windows will be obscured in the overlapped area. All windows are independent of each other and behave like terminals in their own right (124). The window manager maintains the contents of off-display or obscured parts of the windows so that they can be reconstituted as soon as one makes them visible again. While a window is in use, the process can alter the terminal control parameters, i.e. the window opening and closing, and the hardware pattern (the window appearance and its contrast to the background of the screen). For example in using two windows, the first window can be used to compile, consolidate, link and run programs whose output could be piped as an input to the second window which is being used by the user at the same time. The user can determine a window's position and which window appears uppermost. A single process can be connected to several windows and a single window can be connected simultaneously to several processes. Only one window is accessible from the keyboard at a time and this is shown by a thick boundary frame around the particular window.

5.3.5 Languages :

5.3.5.1 C programming language :

The PNX operating system is written in C programming language. C is a general purpose programming language which features economy of expression, modern control flow and data structure, and a rich set of operators. C is not a "very high level" language, nor a "big" one, and is not specialized to any particular area of application. C was originally designed for the implementation of the UNIX operating system on the DEC PDP-11 by Ritchie (125).

5.3.5.2 FORTRAN 77 language :

UNIX FORTRAN 77 adheres closely to the ANSI FORTRAN 77 standard as defined in ANSI-1978. However, PNX FORTRAN 77 has a few extensions to the language to assist compatibility with other FORTRAN dialects, e.g. FORTRAN 66. Also it allows specific features of UNIX on the Perq to be exploited or to facilitate communication with C procedures through the FORTRAN 77 compiler.

5.3.5.3 PNX Pascal :

The version of Pascal available under PNX has some extensions over ISO levels 0 and 1 and differs in compilation and loading from the Pascal under the POS operating system (126).

5.3.6 Spy editor :

The Spy editor is a screen based, interactive, modeless text editor. That it is screen based means that not only is the text the user is editing available on the screen but also all the editing facilities are provided as a display. It is interactive in that all the editing actions the user choose to make are immediately effected on the text on the screen. In other words, what you see is what you get. Modeless means that all the editing facilities are always available. Most other screen based, interactive text editors have two modes; one for issuing editing commands and one for inserting text. The most natural way to edit text is to change the text wherever it is in the file and then move on to the next piece you want to change and Spy provides this environment.

5.3.7 The Make Utility :

One of the important utilities of the UNIX system is '**make**'. This utility, under PNX, can save time and effort in compiling, updating and linking of any program. The usefulness of using this utility can be clearly demonstrated when each part of a package containing is being updated several times. If a part or the whole of the package is being updated, the 'make' utility will try to compile only the recently updated part(s), link the individual object files and prepare the run program. Firstly, it frees the user from errors in recalling the individual parts of the package. Secondly, it will save the user considerable time in figuring out which part has to be compiled and which object files are to be linked to produce the run program. Finally, the utility is also useful in the compilation and the linking of programs written in different programming languages. For example the hazop package has subroutines in FORTRAN 77 and procedures in C and the reliability calculation package is divided into seven parts.

CHAPTER SIX

6. THE HAZOP PACKAGE

6.1 General Introduction :

The existing Hazop package, on the Harris mainframe computer at the Aston University was developed by a number of people (127,128,129,130, 131,132,133,134). Originally the package was written for the ICL 1904S mainframe computer and was then transferred to the University's Harris system (135). Two versions of it have since been developed for the Perq. The first version was developed under the POS operating system. The second version was developed from the POS version in order to run under the PNX operating system. The versionswhich run on the Harris and under POS on the Perq have the same attributes. Both analyse cause and symptom equations and then draw the fault trees but in different environments. However the POS version is more user interactive than the Harris version: this will be discussed later. The version running under PNX does the same functions that the earlier versions do but, in addition, carries out a complete reliability study for the displayed fault tree. This means that the PNX version achieves the goal of assisting the operator in decision making.

6.2 The Data Structure :

The data structure used within the software is common to all the versions. As discussed in Chapter 2, in Lihou coding, the operability study information is prepared in the form of two types of logic equations: cause equations and symptom equations. The operability study information is then translated into a data structure. This data structure consists of three arrays. The first array is an integer array called CELLS and has a dimension of N rows by 10 columns. This array is used to store all the information from the cause equations and some

from the symptom equations. The second array is an integer array called SYMPTS and has a dimension of 50 rows by 20 columns. This array is used to store the remaining information from the symptom equations. The third array is a character array called NAMES. There is an entry in NAMES corresponding to each entry in CELLS and each deviant state appearing in either cause or symptom equations appears once in this array.

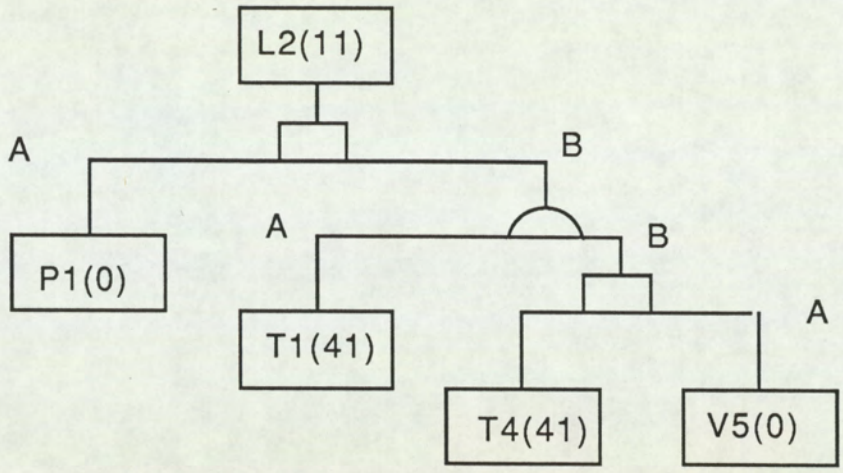
These three arrays are structured in such a way as to represent the logic of the cause and symptom equations. Before discussing each array in detail, the logic of the branches of a tree will be discussed.

6.2.1 Named And Unnamed Branches :

A fault tree consists of many deviants and gates joined by branches. Some of these branches join deviants which have names. Other branches join deviants which have no names. Figure 6.1 shows a fault tree with the two types of branches: this example is taken from an example by Lihou (26). A branch of the type A is called a named branch because it has a named deviant below it. A branch of the type B is called an unnamed branch because it has a gate below it with no name. It will be coded by a combination representing all the gate inputs. An unnamed branch will only occur from a cause equation.

6.2.2 The SYMPTS Array :

SYMPTS is an integer array which stores only the information about the symptom equations. Each row contains the total information coming from a single symptom equation. From the above example, figure 6.1, the following columns contain the following information :



Cause Equation : L2(11) = P1(0)+T1(41)*(T4(41)+V5(0))

Symptom Equation : L2(11) -> N1(32)*N2(13)*N3(11)*N4(12)*N4(141)
*N5(42)*N7(11)

(A) are named branches. (B) are unnamed branches.

Figure 6.1 : Named and unnamed branches.

a) Column 1 : This column will contain the address of the branch that occurs from the left hand side of that symptom equation. That is the row number in array NAMES and consequently the row number in array CELLS.

b) Column 2 : This column will contain the number of branches that are on the right hand side of that symptom equation. This will be equal in the case in figure 6.1 to 7.

c) Columns 3 - 20 : These columns will contain the row numbers in NAMES of these deviant states which are on the right hand side of the arrow sign of that symptom equation.

Suppose, for the sake of argument, that the row numbers in NAMES of the above example are as in the following table :

Deviant state	Row number
.	.
.	.
L2(11)	8
N1(32)	11
N2(13)	12
N3(11)	13
N4(12)	14
N4(141)	15
N5(42)	16
N7(11)	17
.	.
.	.

Initially all the columns in SYMPTS will have the value zero which represents nothing. The final look of particular SYMPTS row representing this symptom equation will be:

8 7 11 12 13 14 15 16 17 0 0 0 0 0 0 0 0 0 0

Symptom equations, at the present, are limited to a maximum of 18 branches on their right hand sides.

6.2.3 The NAMES Array :

NAMES is a character array which stores in words the names of every deviant state appearing in the cause and symptom equations. Initially, all the rows of NAMES contain the word EMPTY. EMPTY, in any row of NAMES, means nothing is stored in that row. If a branch has a name then its name will be placed in the row number in which it appears in CELLS. If a branch is unnamed then an

entry of spaces is made. From the above example the deviant states and their branches, for argument, will be stored as in the following table :

Row no. in NAMES	Deviant state
.	.
.	.
8	L2(11)
9	
10	
11	N1(32)
12	N2(13)
13	N3(11)
14	N4(12)
15	N4(141)
16	N5(42)
17	N7(11)
18	P1(0)
19	T1(41)
20	T4(41)
21	V5(0)
22	EMPTY
23	EMPTY
.	.
.	.

The word EMPTY represents nothing is stored in that row and that that row is available for storage of a new deviant state if required. The blank row numbers 9 and 10 are for the unnamed branches represented by the AND and the OR gates respectively.

6.2.4 The CELLS Array :

CELLS is an integer array, which stores information about all the branches which occur from the cause and symptom equations whether they are named or unnamed branches. Each row of CELLS consists of 10 columns. These columns are classified into four types. The CELLS array should be initialised first by putting a zero value to each element in it. If there is some information to be stored, using the above example, then the following possible

values will be stored accordingly :

a) Column 1: This column may have a value of either 0 or 1 to indicate an unnamed branch or a named branch respectively. In this case it will be 1. This entry is redundant since the information is also available in the corresponding entry in NAMES.

b) Column 2 : This column may have a value of 0 if the branch has occurred from the right hand side of a symptom equation or a value of +1 if the branch is connected to the branches below it through an OR gate. The value is -1 where the branch is connected to the branches below it through an AND gate. A value of +2 occurs if the branch is connected to only one branch below it without a gate and a value of -2 occurs if the branch is not connected to any other branch. This last case is a basic event standing alone. In the previous example an OR gate is connected to the deviant state L2(11). Thus a value of +1 is to be stored in this column for the L2(11) entry.

c) Column 3 : This column has a value dependent upon the state of the branch. If the branch has occurred from the right hand side of a symptom equation, then the value will be equal to the number of symptom equations in which the branch has occurred on the right hand side. If the branch is connected to its branches through a gate, then the value will be equal to the number of the branches below the gate. If the branch is connected to only one branch then the value will be 1, and if the branch is not connected to any branch the value will be 0. From the previous example, since an OR gate is connected to the deviant state L2(11), the value is equal to 2 in row 8 which is equal to the number of the gate inputs.

d) Columns 4 - 10 : These columns contain the addresses in CELLS, which have the information about the branches below the gate. That is they contain the row numbers in the CELLS array of the deviant states below the gate. If the second column has a value of 0 then columns 4 to 10 will contain the addresses in SYMPTS, where the symptom equations, in which the branch has occurred on the right hand side, are stored. From the previous example, these columns will have the values 9,10,18,19,20 and 21. The final look of the CELLS row number 8 will be :

1 1 2 9 10 0 0 0 0 0

The last five columns have values of 0 which indicate that nothing is stored in them.

Consider another example of a CELLS row as follows:

1 0 3 3 4 5 0 0 0 0

The first column indicates that the branch has a name. The second column has a value of zero which indicates that the branch has occurred from the right hand side of a symptom equation. The third column has a value of 3 which indicates that the branch has occurred in three symptom equations on their right hand sides. Columns 4, 5 and 6 contain 3, 4, and equations are stored in the row numbers 3, 4, and 5 of array SYMPTS. Table 6.1 shows a summary of the possible values of each column in CELLS. Appendix F shows a list of cause and symptom equations for the Solvay process as coded by Lihou.

Table 6.1 : Column values in array CELLS and their meanings

Column No.	Value	Meaning
1	0	Branch is unnamed.
	1	Branch has a name.
2	-2	Branch is not connected to any branch below it (i.e. basic event)
	-1	Branch is connected through an AND gate to its branches.
	0	Branch has occurred from the right hand side of a symptom equation.
	+1	Branch is connected through an OR gate to its branches.
	+2	Branch is connected to only one branch below it without a gate.
3	≥0	A) If column 2 has a value equal to 0, then column 3 contains the number of symptom equations from which the branch has occurred on the right hand side. B) If column 2 has a value equal to -1, +1 or +2, then column 3 contains the number of branches below the gate. C) If column 2 has a value equal to -2, then column 3 contains 0.
4 to 10	≥0	A) If column 2 has a value equal to 0, then columns 4 to 10 will contain the addresses in array SYMPTS, where the symptom equations, from which the branch has occurred, are stored. B) If column 2 has a value other than 0, then columns 4 to 10 will contain the addresses in array CELLS, which have the information about the branches below the gate, starting from the left hand side.

6.3 Harris Version of The Package :

The package consists of three major parts. The first part deals with the main calling program and the selection of the available tasks. The second part deals with the analysis of cause and symptom equations and the extraction of information into a data structure. The third part deals with the manipulation and the drawing of fault trees on an appropriate VDU.

6.3.1 Part One :

This part, simply called the main program, starts by asking the user to specify the terminal type. The calling program splits into two types of VDU: graphical VDU and non-graphical VDU.

For graphical terminals the main calling program calls the appropriate subroutines to run the different tasks by simply introducing them in the form of a menu. The available tasks are Analyse, Draw, Translate, Edit and Help. The selection is made by moving the cursor to the appropriate task using a light pen. In fact, from the above menu, only Analyse, Draw and Help are functional. Translate and Edit are simulated forms for future modification.

For non-graphical terminals, the choices are the same and the selection is made by inputting the characters A,B,C,D and E from the keyboard. The characters A,B,C,D and E refer to the menu choices: Analyse, Draw, Translate, Edit and Help respectively. At the present only Analyse and Help choices are in action. Draw cannot work on a non-graphical device.

6.3.2 Part Two :

This part represents the translation of the cause and symptom equations into the specified data structure, viz CELLS, SYMPTS and NAMES. This is done as follows :

1. Initialize the arrays CELLS, SYMPTS and NAMES by putting zero values in each element of CELLS and SYMPTS and the word 'EMPTY' in each element of NAMES. This is done by subroutine PART1.

2. Distinguish whether an equation is a cause or a symptom equation. This is done by the subroutine DISTNT.

3. If the equation is a symptom equation, then :

a) The deviant state on the left hand side of this equation will be allocated a position in NAMES. This is done by subroutine ALLOC.

b) The row number in NAMES will be stored in the first column of the first empty row of SYMPTS.

c) The first deviant state on the right hand side of the symtom equation will be allocated a position in NAMES. The entry number in NAMES will be stored in column 3 of the row in SYMPTS. A position in CELLS corresponding to this entry will be allocated and the number of the row in SYMPTS will be entered in the row of CELLS.

d) If the deviant state has already occurred in another symptom equation then the entry number in NAMES of that state will be stored in column 3 of the current row of SYMPTS. The number of the current row will be placed in the entry row for that state in CELLS.

e) The program processes all the remaining branches from the right hand side of that symptom equation in the same way as that for the first deviant state.

4. If the equation is a cause equation, then the deviant state on the left hand side of this equation will be allocated a position in NAMES.

5. The program then translates the right hand side of the cause equation into a algebraic expression using subroutines TRNSLT, POLISH and CAUSE as discussed in paragraph 6.2.4.

6.3.3 Part Three :

After the analysis is finished, the next step is to draw the appropriate fault tree according to the stored information in the data structure. This is done by subroutine PART2. This can be described briefly as follows :

1. Initialise a character array called FLAG, which is dimensioned (10,40)*40, by putting the word 'EMPTY' in each element of it.

2. Initialise a Graphics Subroutine Library file (136) to assign the appropriate values of the corresponding VDU which is being in use.

3. Check if the data exists by calling subroutine DATA to read the appropriate deviant from the list of names in array NAMES.

4. Call the subroutine ANCELL to analyse the data structure and store the information in FLAG. The way of coding is started with the deviant state being called by subroutine DATA. This is given the code 'Z' and placed in the middle of the first row of array FLAG. The branches below this data are placed in the next row of array FLAG and given the codes AZ, BZ, CZ, DZ and so on. The new branches from AZ will be placed in the third row of array FLAG and given the codes 1AZ,

2AZ, 3AZ and so on. Similarly the branches from BZ will be given the codes 1BZ, 2BZ, 3BZ and so on. The branches from 3C will be given the codes 13C, 23C, 33C and so on. Subroutine BRANCH will check the number of branches and subroutine FILLF will attempt to fill the information inside FLAG.

5. The length, left and right margins of the drawing area are checked by the subroutine EXCEED, CHECK1 and CHECK2 so that the length, left hand side and the right hand side margins of the drawing area are not crossed.

6. Finally subroutine DRAWTR is called to draw the tree from the mapping of array FLAG using the subroutines of the Graphics Subroutine Library (136).

A summary of the course of the package development, which was coded in FORTRAN 77, is shown in table 6.2.

Table 6.2 : The development of the Harris version

Name of the coder	Program purpose
Moraes (128)	To analyse the equations and to form the data structure.
Varelas (129)	To draw the fault trees on a graphical VDU.
Spirakos (131)	To draw the fault trees on a non-graphical VDU.
Jordan (134)	To translate the cause and symptom equations.
Patel (135)	To modify the library HAZLIB which collects the individual parts into a main program.

6.4 POS Version of The Package :

The Harris version did not work directly on the Perq workstation due to the difference in the software support and the non-existence of a Graphics Subroutine Library to handle the graphical part. New subroutines were written in order to make the package more user interactive and friendly. The changes which involve the three major parts are described below.

6.4.1 Changes in Part One :

Because the Perq has a graphical capability, development has been concentrated on the graphical part. The main calling program has been rewritten in order to change the calls to involve the new interactive subroutines which deal with the graphical mode and to remove the non-graphical tasks. So the menu requests the name of the terminal type, a third choice, PERQ. If this last choice is selected then a subroutine called perq is called to initialise the screen parameters and the alternative Graphics Subroutine Library on the Perq. Some subroutines of the original Graphics Subroutine Library, used for the Harris version and dealt with character size and mode as well as for the light pen control, have been omitted because of irrelevancy in the new situation. The new Graphics Subroutine Library will be discussed later. The menu is now provided in a window. The selection is made by moving the cursor to the required option and pressing one of the cursor's buttons.

6.4.2 Changes in Part Three :

This part has undergone considerable changes for two reasons: firstly, the windowing facility of the Perq can be used and, secondly, input and output can be improved with the better facilities of the Perq. This gives the package both more

flexibility and a more friendly user interface than before. Subroutine PART2, especially, is rewritten for this purpose and now acts as follows :

1. Initialise a Graphics subroutine Library file to set up the terminal values.

2. Divide the screen into five parts by creating five windows, each for a specific purpose. The first window is a graphical window and is used to display the fault trees. The second window is for command handling and data input. The third window is to clear the screen. The fourth window is used to switch the process control from the cursor to the keyboard. Finally the fifth window is used to issue the command to end the run. Figure 6.2 shows a schematic diagram for the screen layout.

3. For the first run the input of data is directly from the keyboard. After that the input is made by pointing the cursor to the appropriate deviant box. After the display of the fault tree, the user can continue with the display by pointing the cursor to any deviant state and pressing the cursor button. A new display of the appropriate fault tree is then displayed, beginning with the selected deviant state as the top event. The user can continue this sequence, if there are still unrevealed states, until the last quarter of the display window is reached. At this stage a message window appears on the screen asking the user to clear the screen for more input to be possible. The user then points to the CLEAR window and presses one of the cursor's buttons. All the windows will then be closed and reopened again as fresh windows.

4. If the user does not point accurately inside one of the deviant boxes, then a window carrying an error message will appear on the screen asking the user to try again making his input, this time, from the keyboad. This will happen also if the

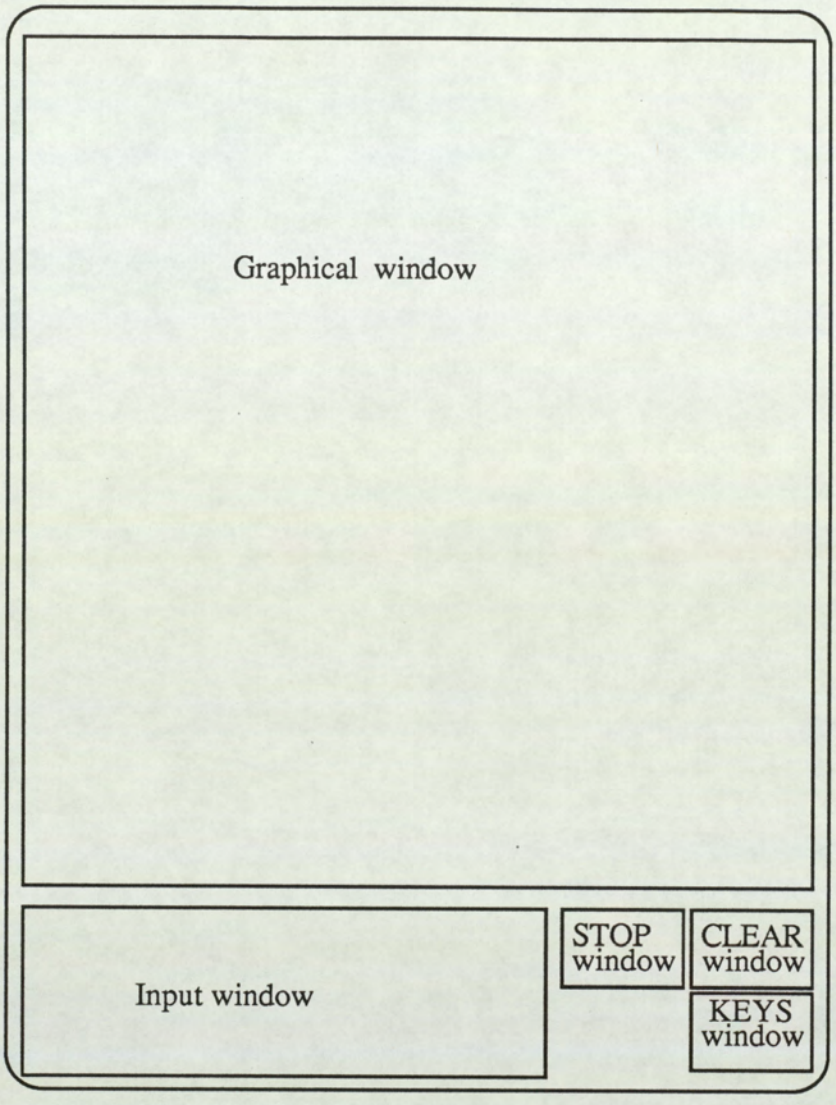


Figure 6.2 : Schematic diagram for the screen layout of the hazop version under POS.

user enters an unknown deviant state: such a state is one which has not occurred in any cause or symptom equation.

5. If the user wants to end his run then he should point to the STOP window and press one of the cursor's buttons. If he is in the keyboard mode he can enter the

word STOP or any part of the word to stop the run. Appendix G shows a listing for the source programs of the hazop POS version.

6.4.3 The TRYOUT Subroutine Library :

This package is coded in Perq Pascal (138). It consists of several procedures and modules to handle information relating to the screen cursor position or the tablet position. The operating system ensures that the screen cursor tracks the position of the tablet other than on those occasions when tracking the tablet would move the screen cursor off the screen. Data can be retrieved for each of the following positions :

a) the tablet position;

b) the cursor position relative to the entire screen;

c) the cursor position relative to the current window.

Different procedures for cursor handling are written. These are:

1- **Procedure getcur :** This procedure reads screen cursor position relative to the top left hand corner of the screen.

2- **Procedure rdcurs :** This procedure reports the depression of any of the tablet buttons.

3- **Procedure strtab :** This procedure senses and reports back the existence of the cursor on the tablet. If the puck is not on the tablet, it will return the value false.

4- **Procedure tabpush :** This procedure recognizes the release of a cursor button following its depression.

5- **Procedure trakcurs :** This procedure enables the application to read the position of the cursor on the screen within defined limits. If the cursor leaves these limits, its position is no longer available to the application.

6- **Procedure setcur :** This procedure sets up the cursor to be an arrow when linked to an application rather than the window manager.

7- **Procedure scrclr :** This procedure clears the screen.

8- **Procedure linexx :** This procedure draws a line on the screen between any two given points. The dimensions are in screen coordinates.

9- **Procedure line77 :** This procedure is the same as linexx but in different coordinates. The dimensions are in the relative current window coordinates.

10- **Procedure window :** This procedure creates a window of a specified origin, width, length and title.

11- **Procedure chnwin :** This procedure makes it possible to switch the input from the current window to another window by changing the current window identifier to that of the new one.

12- **Procedure putchr :** This procedure enables characters to appear on the screen or in a definite window at the required position.

13- **Procedures rs232rd, rs232wrt and rsbaudrd :** These procedures are written for the rs232s serial port for perq communication with other computers. These procedures are used for reading data from remote computers, writing data to remote computers and setting the baud rate for communication.

The above procedures have been written in Pascal and are used as external functions in the Graphics Subroutine Library on the Perq.

6.4.4 The GINO-F Library on the Perq under POS :

The GINO-F Library, earlier referred to as the Graphics Subroutine Library, is a General Purpose Graphical Subroutine Library created by the Computer Aided Design Centre (136). In order to carry out graphics on the Perq using the software written for the Harris a new version of GINO-F had to be

written. Akebe and Jordan (137) wrote a subset of GINO-F coded in FORTRAN 77 which however, was not written on the Perq. The original package required only one system subroutine to draw a line from one screen coordinate to another. During this research, this GINO-F library, was expanded to 40 subroutines and was ported to run under POS. The library contained some of the usual GINO calls such as MOVTO2, MOVBY2, LINTO2, LINBY2, ARCTO2. In addition to these, special subroutines for window selection, handling, switching and cursor control were included. The GINO library is assisted by the TRYOUT package discussed above. Appendix H shows the listing of the source programs of this library.

A brief presentation about the special subroutines of the GINO-F library written to run under POS is described as follows :

1- **PERQ :** This subroutine sets the initial values of the for X and Y extents of the screen, pen position and transformation matrix on the Perq. It clears the screen before the set of the parameters.

2- **PICCLE :** This subroutine clears the screen. It calls the TRYOUT procedure scrclr. It carries out the same job as that of the standard GINO-F subroutine.

3- **CURSOR :** This subroutine calls the procedures strtab and rdcurs to define the cursor set up mode and to read the cursor position. The action is the same as that of the standard GINO-F subroutine.

4- **LINTO2 :** This subroutine carries out the same job as the standard GINO-F subroutine. It is used to draw a line from the current position to a specified point in screen coordinates. It uses the TRYOUT procedure linexx.

5- **CHAHOL :** This subroutine writes any string of characters in the required position on the screen. It uses the TRYOUT procedures getcur, setcur and putchr.

6- **WINDOWI :** This subroutine draws a window as large as the screen itself using the TRYOUT procedure linexx.

7- **MMULT2 :** This subroutine is written to create an axis transformation, in two dimensions, between the screen coordinates and the absolute picture coordinates.

8- **M2INV :** This subroutine finds and returns the inverse of any matrix for the purpose of axis transformation in two dimensions.

In addition to the GINO-F library on the Perq, subroutines for window handling have been written to run under POS as follows :

1- **WINDIN :** This subroutine is written for drawing the layout of the fault tree manipulation. It creates, using the TRYOUT procedure window, five windows for fault tree drawing, input information, clear screen, key input and stop the running.

2- **CHWIN :** This subroutine calls the TRYOUT procedure chwin for window switching.

6.5 PNX Version of The Package :

After the Perq operating system (POS) was changed to the PNX operating system, a new version of the package was written. This new version has a similar layout to that of the POS package but with an extended option for reliability calculations. Thus in addition to the fault tree display, a reliability study can be carried out while running the package. This modification gives the user the power to extend the use of the package in the area of decision making as well as in carrying out a sensitivity analysis on a particular top event. New subroutines have been written for the three major parts of the package. Also some of the old subroutines had to be altered to cope with the new system supported programming language. Thus the auxiliary, TRYOUT, package had to be changed from Pascal to C language. The new auxiliary package is called MAX. Other changes are made in

dealing with the acknowledgement messages. Each question or error message has to be acknowledged first before any action is taken. This will give the user more confidence in making his next choice. Some of the old subroutines have been deleted because of irrelevancy.

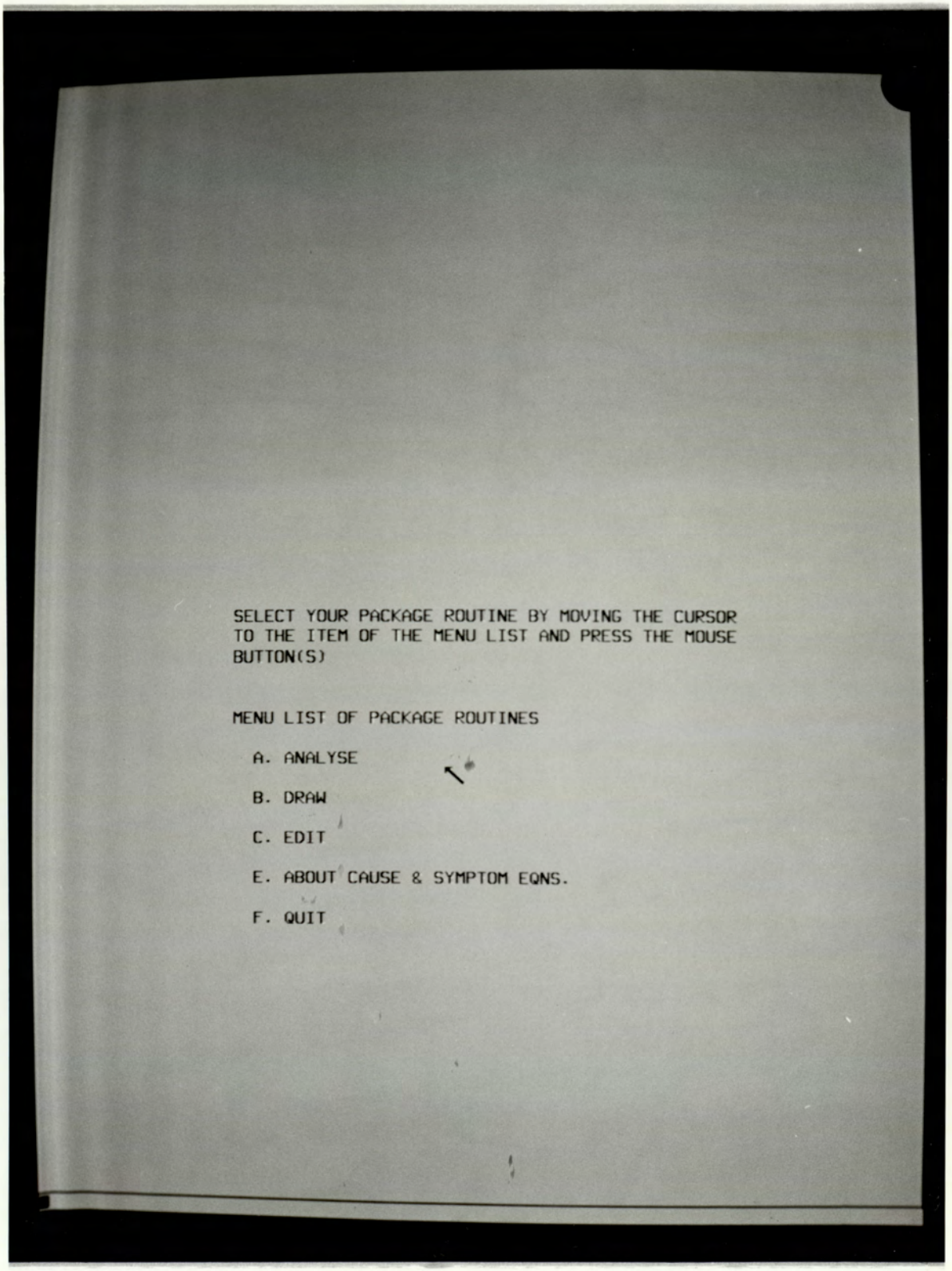
6.5.1 Changes in Part One :

The main calling program was rewritten again to give the user the option of entering the cause and symptom equations from a file of his choice. This is done through a message window on the screen. After this the program asks the user, through another message window, if he wants any reliability calculations. If so, then the program will ask the user to enter the data file name. This data file must contain all the failure rates of the basic events of the process in question.

The menu option 'HELP' was changed to an information option about cause and symptom equations enabling the user to scroll through this file. Previously this option had the same function but without the user ability to abort the choice and return to the menu. The new menu is shown in figure 6.3.

6.5.2 Changes in Part Three :

The changes done in the POS version were also carried out for this version. The whole screen is replaced by a graphical window which in turn is subdivided into six parts. The first and the larger part is used for the fault tree drawing. The second part is occupied by another window for the purpose of input from the keyboard. The third window is used to switch the input from the tablet cursor to the keyboard: selection of this part enables the user



Figur 6.3 : The menu

to input data to the second window. The fourth window is used to clear the screen by closing all the windows and reopening them again. The fifth part is used to stop and end the run. Finally the sixth part is used to carry out the top event probability calculation. Figure 6.4 shows the screen layout. Appendix I shows the listing of source programs of the hazop PNX version.

6.5.3 The MAX Package :

This package is coded in the C language. It consists of a main program with procedures for cursor control, line drawing, communications and window handling. This package supports the GINO-F subset described above. The package has the following procedures :

1- **wopen :** This procedure opens a specified window. This means that a window, firstly, has to be created as a special file using the utility 'Mkwind'. The utility '**Mkwind**' is supplied with the window features such as height, width and the position of the top left hand corner in screen coordinates.

2- **wclose :** This procedure closes a specified window.

3- **wlin :** This procedure draws a line between any two given points. The dimensions are in window coordinates.

4- **wstrng :** This procedure enables the characters to appear inside the current window at the cursor's current position.

5- **wpuck :** This procedure returns the position of the cursor in either the tablet, screen or window coordinates when a depression takes place. By default the cursor with respect to the window coordinates is used.

6.5.4 The GINO-F Library on the Perq under PNX :

When the POS operating system was changed to the PNX operating

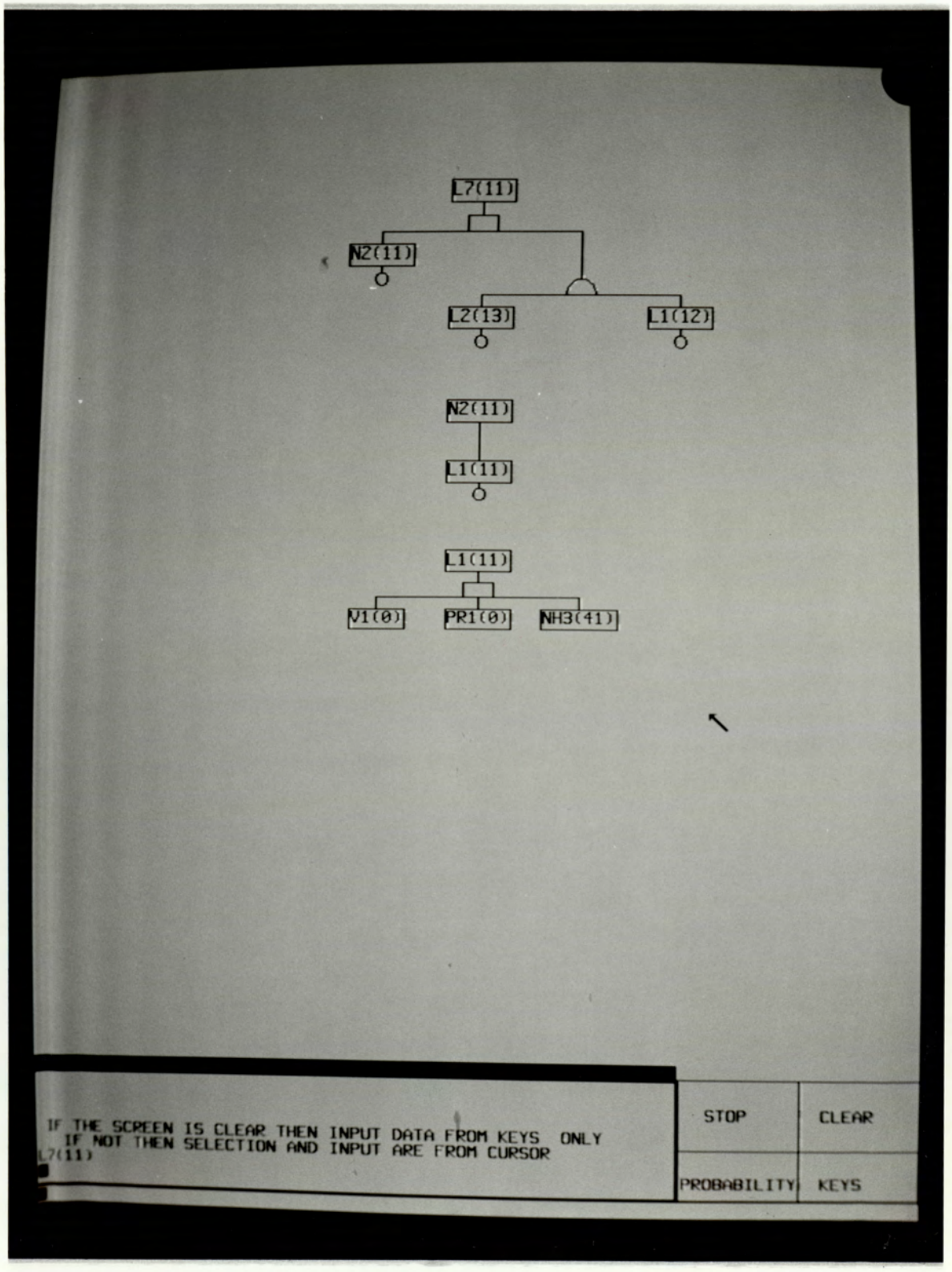


Figure 6.4 : Screen layout under the PNX system

system, the system's software language changed from Pascal to C. This raised the need for another auxiliary package to replace the TRYOUT package in order that most of the GINO-F subroutines could be used as they stood. The new auxiliary package is called MAX. Minor changes were made to the previously mentioned subroutines in order to handle the new system software. MAX procedures were substituted for all the TRYOUT procedures. Also some new subroutines were added to the library which now stands at 49 subroutines.

A brief description for the newly added subroutines is given below :

1- **CCLOSE :** This subroutine calls the MAX procedure wclose to close the graphical window.

2- **COPEN :** This subroutine calls the MAX procedure wopen to open the graphical window.

3- **LINEXX :** This subroutine calls the MAX procedure wlin to draw a line between any two given points in the graphical window.

4- **FINAL :** This function is written to count and find the end of a string of characters.

5- **COMAND :** This subroutine constructs the layout of the graphical window and sets up the commands windows and their positions in the graphical window.

Appendix J shows the listing of source programs of this library.

CHAPTER SEVEN

7. IMPLEMENTING THE FAULT TREE DISJOINT REDUCTION ALGORITHM

7.1 The Data Structure :

7.1.1 Introduction :

The data representation is an important factor affecting the running time of algorithms whatever the programming language in use. Semanderes (34) and Wheeler et.al. (88) have improved the performance of their cut set enumeration algorithm by using better data representations for cut sets. Rosenthal (139) has discussed three data representations for cut sets.

Major concerns of theoretical computer science are estimating the time and storage resource requirements of algorithms, determining the data representations (i.e. data structures) used, and synthesising the programs which implement them (140,141). Better data representations or more clever coding can considerably reduce the resources needed for solving problems involving large and complex fault trees.

7.1.2 Basis Of The Data Structure :

This reduction algorithm uses as input data the results from the analysis of cause and symptom equations. The data are represented in two forms of matrices, namely CELLS and SYMPTS. These have been discussed in Chapter 2. Although FORTRAN 77 is not a very efficient language for handling information which includes sets and pointers it is good at handling input and output files. Also writing the program in FORTRAN 77 renders it almost totally machine independent. To reduce the amount of storage and the searching time

a technique has been developed in which variable length sets are stored in a number of fixed length arrays. Three arrays have been created :

i) Array STARTS - a one dimensional integer array

ii) Array SETS - a two column, multiple row integer array

iii) Array NORMAL - a one dimensional logical array

The elements of sets are held in the array SETS. The first column of SETS holds the element value while the second column holds a pointer to the row of SETS containing the next element of that set. Clearly we need to identify the start and end of each set held in SETS. Identification of the last element of a set is given the pointer in the second column of SETS having a value of zero. The starting row of each set in SETS is given by an entry in the array STARTS. Thus set i has an entry at STARTS(i) which is a pointer to the first element in set i.

For this fault tree disjoint reduction algorithm some elements in SETS have to be identified as being inverted and so for each row in SETS there is a corresponding element of NORMAL which is set to .TRUE. if the set element is not inverted and is set to .FALSE. if it is inverted.

All rows of SETS not currently in use for the storage of set elements are linked together in the empty set.

Two integer variables, which act as pointers, EMPTY and FINISH, have been added to the total data structure. EMPTY contains a pointer to the first entry in the empty set and FINISH points to the last entry in the empty set. Finally the logical variable FULL indicates that no more items can be added to SETS: this happens when all rows of SETS are in use for set element storage.

7.1.3 Subroutines For The Data Structure Manipulation :

The set storage system is explained below in terms of the subroutines used.

7.1.3.1 Subroutine INIT :

This subroutine initialises the set structure. For each element of STARTS a value of -1 is assigned to indicate that this entry is empty and not occupied by a set, i.e. not in use. In SETS all pointers in the second column are set to point to the next row. Thus the entry in row 1 points to row 2, in row 2 to row 3 and so on. Only the last entry of SETS contains a zero pointer. EMPTY is initialised to point to row 1 of SETS and FINISH points to the last row of SETS. Thus SETS is set up initially to consist only of the empty set. The logical variable FULL is set to .FALSE. Figure 7.1 illustrates the individual parts of the data structure which is created by subroutine INIT.

The code permits the size of SETS and STARTS to be set prior to compilation. STARTS currently has 1000 elements. There may therefore be 1000 sets of variable length stored. There are currently 10000 rows in SETS, allowing up to 10000 set elements to be stored.

7.1.3.2 Subroutine NEWSET :

This subroutine creates the entries for a new set in the set structure and returns its entry number in the set structure. Thus the user does not assign a number to a set but has the set number returned by the subroutine. The ith element in STARTS is assigned the value 0 instead of the empty value -1. This subroutine is always called when a new set has to be created within the set structure. The routine checks each entry of the array STARTS and the first -1

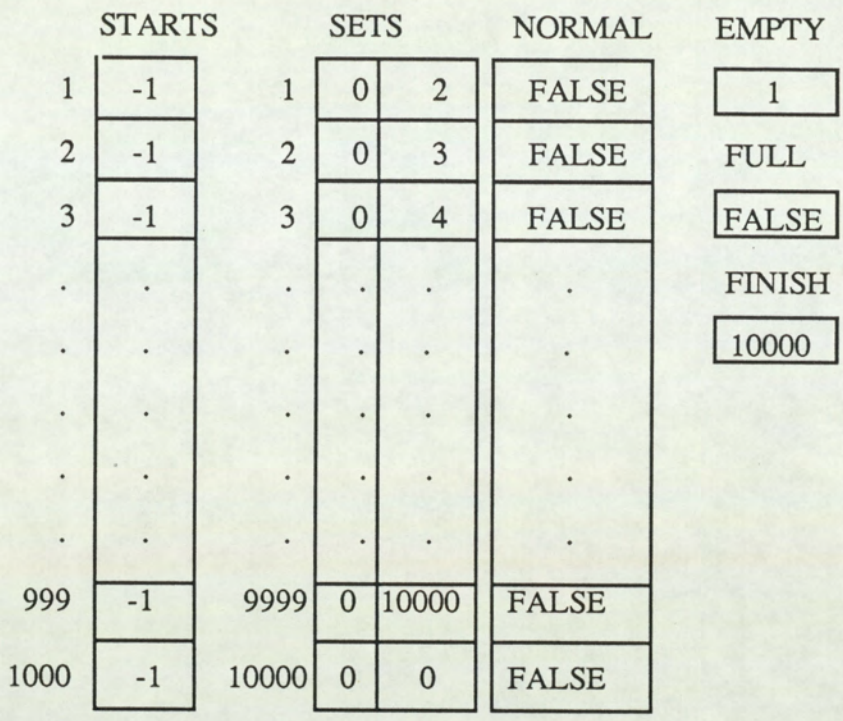


Figure 7.1 : The initialised set structure

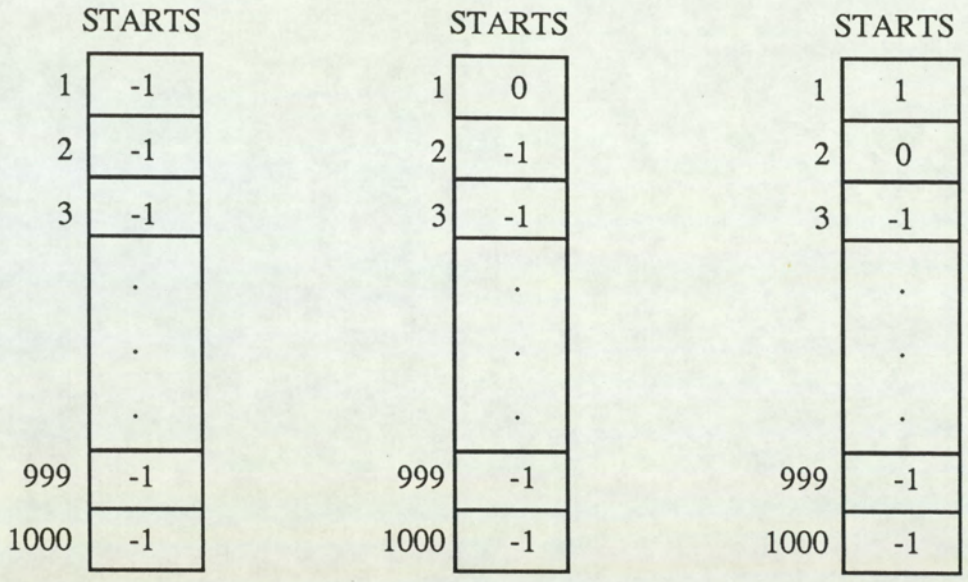
value found will be substituted by 0 and its entry number is then returned to the calling program. Elements of STARTS can therefore have one of three sets of values:

-1 indicates this set number is currently not in use

0 indicates a null set

a positive anumber indicates that there is non-null set and points to the first element entry in STARTS.

This is illustrated in figure 7.2.



(a) (b) (c)

a) No set has been created. b) One set has been created.

c) Two sets are created with the first set as a non-empty set.

Figure 7.2 : Process of creating sets.

7.1.3.3 Subroutine ADDITM :

If an item or items are to be added to the set structure then subroutine ADDITM must be called. The subroutine ADDITM is not only to added items to a particular set but it also adds subsets into a main set. If the added items were subsets then they would be add to the main set sequentially. If the added item are not subsets then subroutine ADDITM sorts the set's elements and executes two Boolean algebra rules, namely the Idempotent Law and the Complemented Law. The work of subroutine ADDITM can be summarised as follows:

a- To add a new item to a particular set in an ascending order.

b- If the same item does exist in the set then no action will take place (Idempotent Law).

c- If the complement of the item(to be added) does exist in the set then the whole set will be eliminated (Complemented Law).

d- The subroutine distinguishes the type of the item to be added via a logical flag. If the item to be added is a subset then it will be added at the end of the set if not then the above steps have to be performed.

Illustration :

The following set of four items 3, 9, 7 and 2 is to be put in the empty set structure. The first step is to initialise the set structure, using subroutine INIT, as shown in figure 7.1. Then create the new set as shown in figure 7.2-a by calling subroutine NEWSET. The subroutine ADDITM is then called. ADDITM will check first whether there is any room in array SETS, i.e. if the first entry in the empty set is equal to the last entry in the set the logical flag FULL is TRUE. The message "There is no more room for further set items" appears across the screen and the file "erreport" will be opened and will have stored in it all the present set structure. If this is not the case one of the following conditions takes place :

a) If no logical flag is setup, i.e. the item to be added is not a subset then :

i) If the value in STARTS of this set is 0, i.e. a new set, then the integer variable EMPTY gives the pointer to the first entry in the empty set row in STARTS in which the element is to be placed; EMPTY initially points to row 1 of SETS. The number of the set, to which the entry is made, is put in STARTS, i.e. the 0 in array STARTS is replaced by number 1. Then the item 3 is placed in the first row of the first column of the first column of the array SETS and the corresponding value in the second column becomes 0 to indicate the end of this set. The original value

in the second column which was 2 is assigned to EMPTY. Figure 7.3 shows this arrangement. In addition the corresponding entry in NORMAL will be assigned to TRUE.

ii) The second item is added to the set in ascending numerical order. If it duplicates an existing value, no change to the set is made. If the value exists but the value of the entry in NORMAL is the inverse of that of the new item, then the entire set is deleted. Thus here the new element 9 is appended to the existing items. This is illustrated in figure 7.4.

iii) Using the same rules, the third item 7 is placed in linked order between the existing items 3 and 9 in order to retain the ascending numerical order. This is shown in figure 7.5.

iv) When the fourth item 2 is to be added three pointer swapping will take place before the addition. The first pointer swapping is between the EMPTY and the STARTS pointer. The second swapping is between the EMPTY and the first column in row SETS. The third swapping is between the second column in row SETS and EMPTY. This is shown in figure 7.6.

b) If the item is a subset then :

i) If the value in STARTS of this set is 0, i.e. a new set, then step (a(i)) will be carried out. Figure 7.7 illustrates this case.

ii) If the value in STARTS is not 0 then the item will be added at the end of the set.

7.1.3.4 Subroutine DELSET :

To delete a set the value of its pointer in STARTS must be replaced by -1. The original value of the pointer will be given to the pointer EMPTY. Since the

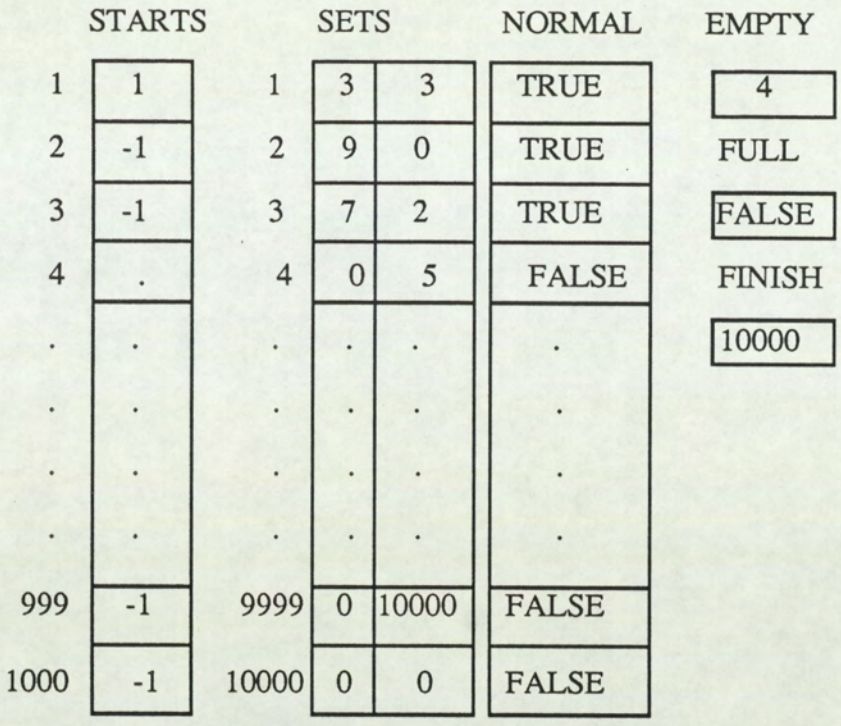


Figure 7.5 : Adding item 7 to the set

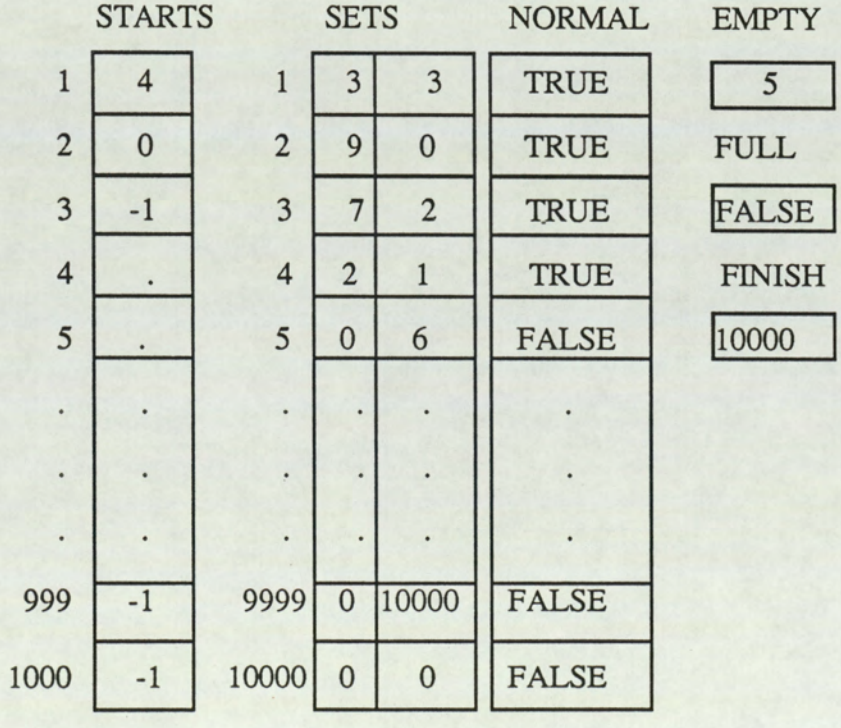


Figure 7.6 : Adding item 2 to the set

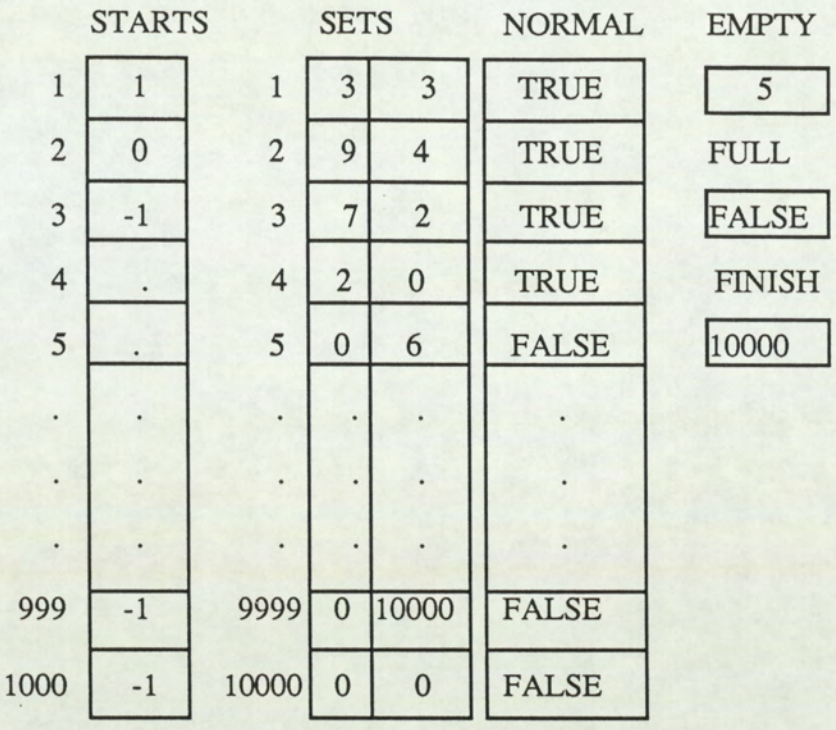


Figure 7.7 : Adding an empty subset to the set.

pointer EMPTY originally has a value, so this original value will be passed to the second column of the last item of the set to replace the zero value. Now the set is deleted since it has no reference pointer in STARTS and the set rows being linked to the empty set. Figure 7.8 shows the deletion of set 1 showed in figure 7.6.

7.1.3.5 Subroutine DELITM :

To delete an item from a set one of the following cases will happen :

i) If the item to be deleted is the first item in the set then pointer swapping will be firstly between STARTS and the second item in the set and secondly between EMPTY and the second column of the row of the deleted item.

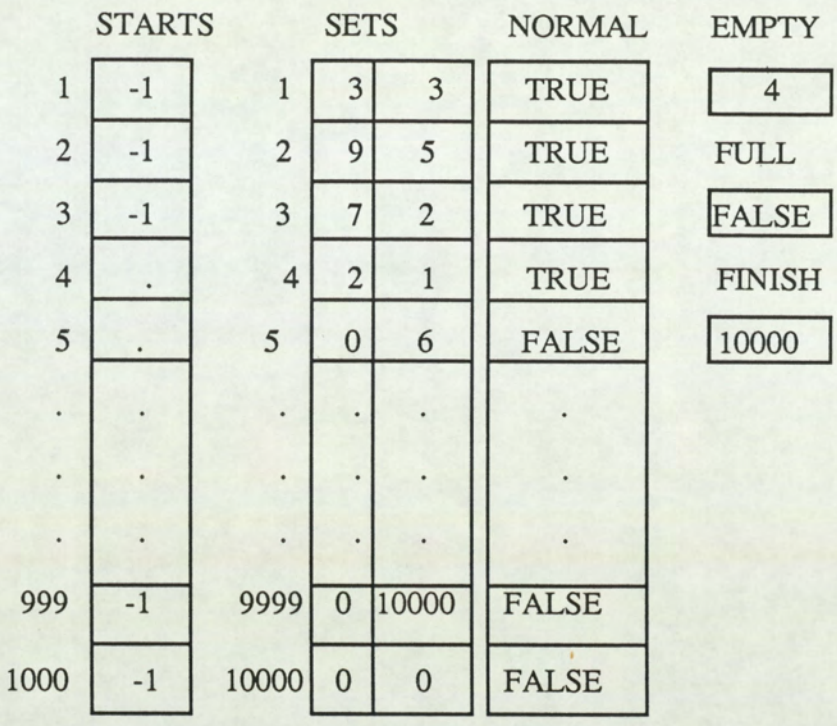


Figure 7.8 : Functioning of subroutine DELSET

ii) If the item to be deleted is neither the first item nor the last item then only swapping of the appropriate pointers within the set will take place. The deleted item will be linked to the empty set as the first empty position in it.

iii) If the deleted item is the last item in the set then a 0 value will be placed in the second column of the item in front of the deleted item to indicate the new end of the set position. The deleted item will be linked to the empty set as the first empty position in it.

Illustration :

Consider the following set which consists of the six items: 6, 9, 12, 14, 15, 20. The above three different cases of item deletion are described into the following figures :

i) Figure 7.9 shows the deletion of the first item in the set, i.e. 6.

ii) Figure 7.10 shows the deletion of item 9.

iii) Figure 7.11 shows the deletion of the last item in the set, i.e. 20.

7.1.3.6 Subroutine INNITTK :

This subroutine is used to initialise a sequential reporting process for the particular set. The usefulness of this subroutine can be shown later in other subroutines.

7.1.3.7 Subroutine TAKITM :

This subroutine is usually called after subroutine INITTK. It takes sequentially an item from the initialised set at a time and retains its value for reporting back.

7.1.3.8 Subroutine NUMSET :

This subroutine counts the total number of the items in a set. The subroutine first looks at STARTS to find the start of the set and then takes, in a sequential manner, an item at a time till it reaches the end of the set. The number of the items in the set will be reported back.

7.1.3.9 Subroutine SETERR :

This subroutine simply returns an error message when on request.

	STARTS
1	2
2	-1
3	-1
4	.
5	.
.	.
.	.
.	.
999	-1
1000	-1

	SETS	
1	6	7
2	9	3
3	12	4
4	14	5
5	15	6
6	20	0
7	0	8
.	.	.
9999	0	10000
10000	0	0

NORMAL
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
FALSE
.
FALSE
FALSE

EMPTY
1
FULL
FALSE
FINISH
10000

Figure 7.9 : Deletion of item 6.

	STARTS
1	3
2	-1
3	-1
4	.
5	.
.	.
.	.
.	.
999	-1
1000	-1

	SETS	
1	6	7
2	9	1
3	12	4
4	14	5
5	15	6
6	20	0
7	0	8
.	.	.
9999	0	10000
10000	0	0

NORMAL
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
FALSE
.
FALSE
FALSE

EMPTY
2
FULL
FALSE
FINISH
10000

Figure 7.10 : Deletion of item 9.

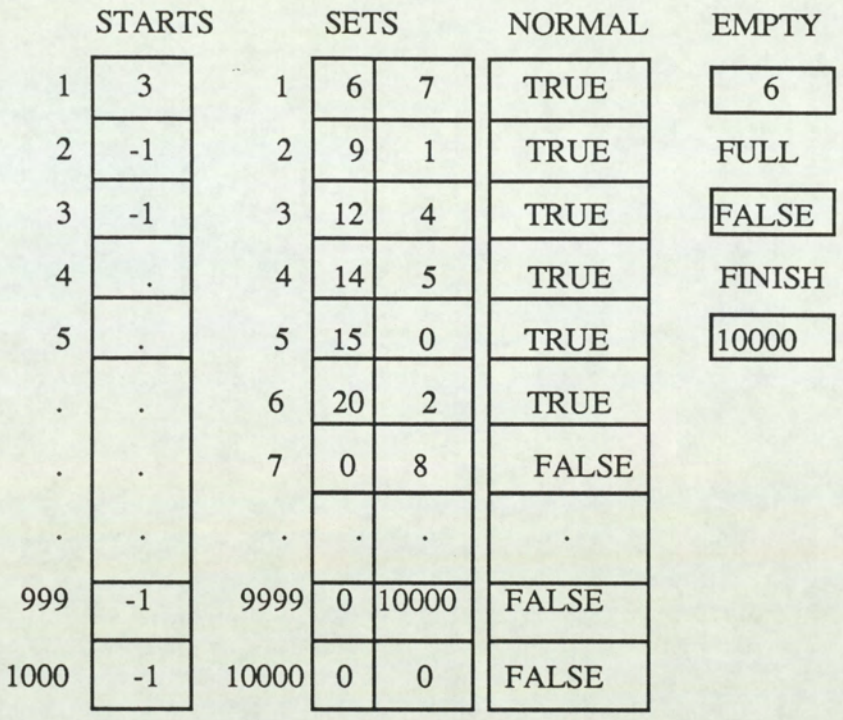


Figure 7.11 : Deletion of item 20.

7.1.3.10 The Logical Function EQUAL :

This logical function returns either .TRUE. or FALSE. on the comparison of any two sets. The number of items in the first set should be less than or equal to the number of items in the second set. If the first set proves to be a sub-set of the second set then a true value will be returned. Otherwise a false value will be returned.

7.1.3.11 Subroutine CPYSET :

This subroutine makes a copy of the required set. This is done by calling the following subroutines successively: NEWSET, NUMSET, INITTAK, TAKITM and ADDITM.

7.1.3.12 Subroutine TYPSET :

This subroutine keeps the sets into three separated and classified groups. The first group 'BASIC' contains sets that consist of basic events only. The second group 'ORBAS' contains sets that consist of basic events together with OR gates with basic event inputs only. The third group 'OTHERS' contains all the sets that have gates with gate inputs and any other mixture of gates and basic events within them. The classification of the sets into the above category will help in the reduction of the search time required for resolving the gates that contain gates inputs and also in solving AND gates. In addition to that reshuffling of sets is possible after any gate(s) resolving.

7.1.4 Subroutines For Rules Manipulation :

7.1.4.1 Subroutines For Rule 1 Manipulation :

7.1.4.1.1 Subroutine STEP1 :

This subroutine carries out Rule 1 of the technique by checking whether the TOP event is an OR gate or an AND gate. If the TOP event is an AND gate then its inputs will be stored in one set. If the TOP event is an OR gate then its inputs will be stored in separate sets in accordence with their nature. That is a basic event is stored in the first group 'BASIC', an OR gate with basic event inputs is stored in the second group 'ORBAS' and any other gate with gate inputs is stored in the third group 'OTHERS'.

7.1.4.2 Subroutines For Rule 2 Manipulation :

7.1.4.2.1 Subroutine STEP2 :

This subroutine carries out Rule 2 of the technique. It resolves all the

intermediate events. This means that no sets in the third group 'OTHERS' will be left since all AND gates and all OR gates with gates inputs have been resolved.

7.1.4.2.2 Logical Function TESTP2 :

This logical function returns a .TRUE. if the questioned item has a gate input. Otherwise it returns .FALSE.

7.1.4.2.3 Subroutine RESTP2 :

If a .TRUE. is returned from the logical function TESTP2 then subroutine RESTP2 is called. This subroutine will carry out the actual job of gate resolution. It reshuffles the resultant sets from any gate resolution into the appropriate group, i.e either 'BASIC' or 'ORBAS'.

7.1.4.3 Subroutines For Rule 3 Manipulation :

7.1.4.3.1 Subroutine STEP3 :

This subroutine contains three calls only for supersets deletion. It calls subroutine RSUP1 for the deletion of supersets with in the group itself. So two calls for this subroutine is needed to clear group 'BASIC' and group 'ORBAS' from supersets. The third call is for subroutine RSUP2 to clear supersets that exist between the two groups.

7.1.4.3.2 Subroutine RSUP1 :

In order to discover a superset and then to delete it a sequential comparison for the individual sets within the group is needed. The time consumed by this

subroutine to carry out a certain job is very important in determining the over all time requires to calculate the TOP event probability. This subroutine has to be called, as will be described later, every time after any disjointed process.

7.1.4.3.3 Subroutine RSUP2 :

This subroutine also is used to search and delete any superset existing when the sets of group 'BASIC' is compared with the sets of group 'ORBAS'.

7.1.4.4 Subroutines For Rule 4 Manipulation :

7.1.4.4.1 Subroutine COLCT :

The job of this subroutine is to collect the sets of the two groups in one group in order to prepare the set for the disjoint process.

7.1.4.4.2 Subroutine DISJON :

The Bennetts disjoint procedure has been carried out by subroutine DISJON. The subroutine disjoints one set from the rest at a time. When a set is compared with another set the items to be disjointed are kept first in a temporary set and are not to be disjointed until the comparison between the two sets is finished. If it is found that a disjoint process is needed then subroutine MAKDIS is called. Otherwise the temporary set will be deleted and the process continues in the same manner for the remaining sets. After the completion of each set from the remainder a call to subroutine RSUP1 is made if and only if there is more than one item to be disjointed, to ensure no supersets are present when another set is to be disjointed.

7.1.4.4.3 Subroutine MAKDIS :

This subroutine carries out the actual changes on the set to be disjointed and introduces the appropriate inverted items accordingly.

7.1.4.5 Subroutines For Rule 5 Manipulation :

7.1.4.5.1 Subroutine STEP5 :

This is the main subroutine in which all the remaining steps of the technique are carried out. It calls several subroutines to carry out the classification of sets, the breeding of new sets, if required, for the appropriate repeated basic events, the removal of repeated events from gate inputs and finally the evaluation of the disjoint of the non-disjointed sets.

7.1.4.5.2 Subroutine RESOLV :

This subroutine has the main task of processing the remaining unresolved gates in sequence. It calls another subroutine to check if the particular gate has a repeated event in them or not. If the gate does not have a repeated event in it then its inputs will replace it in its set. If the gate does have a repeated event then the following will take place :

a) If the repeated event does occur in the same set with the gate then the whole gate is deleted from the set.

b) If the complement of the repeated event does exist in the set then the repeated event will be removed from the gate inputs and the remainder will replace the gate in its set.

7.1.4.5.3 Subroutine CHECK :

This subroutine is called in order to check whether a gate does has a repeated event among its inputs. If so then .TRUE. is returned otherwise .FALSE. is returned.

7.1.4.5.4 Subroutine COLSET :

This subroutine checks for subsets and transfers them from a two level set to a one level set. A two level set means that the set elements are subsets while a one level set is the one whose elements are basic events only.

7.1.4.6 Reporting The Set Structure :

The reporting back of the data set structure is done by subroutine REPORT. It returns the contents of the elements of the arrays STARTS and SETS. Appendix K shows the listing of the FTDRA source programs.

7.2 Real Time Processing :

The real processing time for several examples has been recorded in terms of the CPU (Central Processing Unit) time. Table 7.1 shows the tree characteristics for fault trees whose processing times have been recorded. Table 7.2 shows the number of cut sets after each stage of FTDRA application as well as the time required to carry out the individual stages and total time needed to obtain the TOP event probability expression in terms of the basic events. Example 3 proved to be too large for the program due to more than 1000 sets being generated. The program failed in step 4. The full output for these trees is given in Appendix L.

Table 7.1 : A general comparison between the characteristics of some examples.

Item	Number of events				
	Ex.5	Ex.4	Ex.3	Ex.2	Ex.1
Tree size (total)	16	16	71	43	59
Basic events	9	9	19	11	10
Repeated basic events	2	2	7	3	9
Figure number	7.12	3.7	4.3	4.2	4.1

Table 7.2 : A general comparison between the number of sets after the application of FTDRA rules with their CPU times for the examples given in table 7.1.

FTDRA application	Number of sets					Time (CPU) seconds				
Up to including	Ex.5	Ex.4	Ex.3	Ex.2	Ex.1	Ex.5	Ex.4	Ex.3	Ex.2	Ex.1
STEP1	2	2	2	2	2	0.7	0.6	0.9	0.7	0.9
STEP2	5	13	506	14	32	0.7	0.6	3.4	0.9	0.9
STEP3	5	5	278	10	28	0.7	0.7	17.4	0.9	0.9
STEP4	4	4	----	13	8	0.7	0.7	----	1.0	26.3
STEP5 (ALL)	5	7	----	13	6	0.7	0.7	----	1.1	26.6

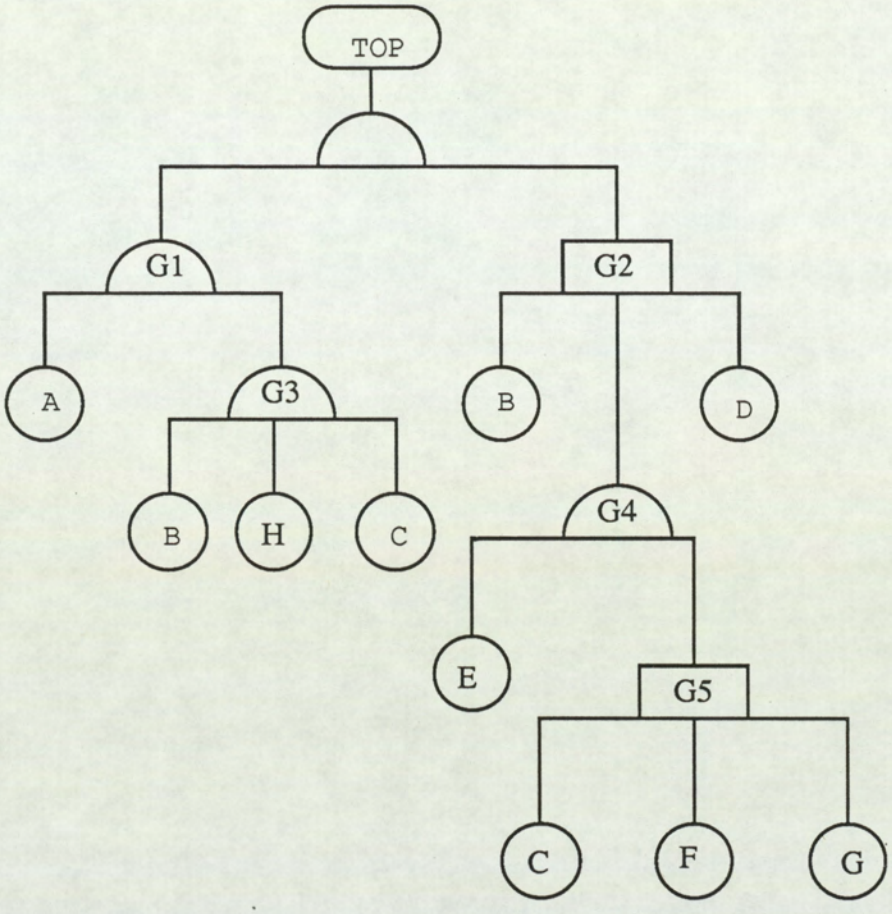


Figure 7.12 : Example 5

CHAPTER EIGHT

8. DISCUSSION AND CONCLUSION

8.1 Discussion of work done :

8.1.1 Introduction :

Lihou (26) suggested a method to code data from hazop studies in the form of cause and symptom equations. The Boolean logical operators OR and AND are used to form these equations. This means that the method is restricted to coding coherent systems only. Symptom equations can have only an AND operator while cause equations can have both OR and AND operators. The method uses a word model based on keywords that stimulate thought. The keyword set is based on two subsets: property words and guide words. These words are represented in the cause and the symptom equations by index numbers. The numbers, shown between brackets, are used to describe any deviant state.

The Aston Hazop Package only deals with the Boolean logic used in these equations. It takes no account of the coding method for expressing deviant states or failure modes. It simply reproduces the codes used in a particular equation. So how deviant states and failure modes are expressed is up to the user. For these reasons the Lihou method of coding is versatile and is convenient to use with many computerisation techniques and languages. Alternatively the usefulness of having index numbers to express deviant states is more convenient from the view of storage on a computer and in passing the codes to other packages. Boolean Algebra manipulation has been used successfully with these codes as shown in the Fault Tree Disjoint Reduction Algorithm.

8.1.2 The Fault Tree Disjoint Reduction Algorithm (FTDRA) :

A number of algorithms for obtaining the minimal cut sets have been considered. A fault tree may be one of the following types :

i) No repeated basic events.

ii) Only one repeated basic event in a gate.

iii) More than one repeated event (basic or intermediate) in a gate.

The TOP event probability can easily be obtained for the first type. In the second type the TOP event probability can be obtained using some techniques like tree reduction (90), decomposition and modularisation techniques (142) or the transformation of the tree into a binary tree (143). The third type is the more difficult and more complex one to analyse. Page and Perry (143) have recently published an algorithm for the evaluation of fault tree probabilities without determining the minimal cut sets of that tree. However they specifically state that this algorithm is most appropriate to trees with fewer than 100 nodes and acknowledge that the determination of cut sets may be necessary for trees with hundreds of nodes. The alternative to an analytical method is a Monte-Carlo approach. However the applicability of Monte-Carlo methods is dependent not only on the size of the tree but also on the probability of events within the tree (144). If rare events occur a large number of Monte-Carlo simulation runs may be required to obtain a result with a reasonable degree of confidence. The processing time of the analytical methods is independent of the event probabilities. The disjoint technique has been introduced to suit this application. The disjoint approach is a top-down type so its application in conjunction with other top-down minimal cut sets reduction techniques is natural. A disjoint technique can be applied on its own (112) but more time will be spent in cutting out redundant sets. From the above description the FTDRA technique has been derived and based. It

uses the benefits of reduction techniques and the disjoint approach together in such a way that the processing requirements are minimised. Recently, Page and Perry (145) have modified the use of their method to cater for non-coherent fault trees of size 50 - 100 events. They claimed that, by using the recursion technique offered by Pascal programming language, more savings could be made. The tree has to be changed into a binary tree with each parent having exactly two children (145). This is done by expanding the tree until it is totally converted into a binary tree. By doing this more intermediate events are generated and hence the recursion technique may not give substantial processing time reduction.

The FTDRA technique does not waste time by finding the minimal cut sets first and then carrying out the disjoint process. It actually gets to near minimal cut sets by leaving the OR gates with basic inputs unresolved, so that the number of cut sets is minimum, and then applies the disjoint technique. By doing this no time is wasted on redundant events. In reality all the intermediate events are redundant items. The FATRAM technique gets rid of most intermediate events first and on this basis it has been chosen for the FTDRA technique. To justify this selection two other significant techniques, tree reduction and bottom-up analysis, were compared with FATRAM in Chapter 3. Limnios and Ziani (146) tested the use of MOCUS, Benjamin technique and FATRAM in their method to obtain the minimal cut sets of the TOP event. They commented that FATRAM applies if the fault tree has at least one OR gate with only events as inputs. They also suggested that the algorithm of Benjamin et.al.(86) does not apply if the repeated events of the fault tree are all inputs to AND gates. FATRAM was found more natural and simple in its approach.

8.1.3 Implementation Of The FTDRA On The Perq :

8.1.3.1 The Programming Language :

FTDRA has been coded in FORTRAN 77 because :

a) FTDRA has been written firstly as an addition to the existing Aston Hazop Package which is coded in FORTRAN 77 and secondly as a stand alone reliability calculation package.

b) FORTRAN 77 is the most familiar programming languages to scientists and engineers.

c) Almost all the reliability calculation programs found in the literature were coded in one of the FORTRAN language family of which FORTRAN 77 is the most advanced type (142,147).

d) Standard FORTRAN 77 coding is portable and machine independent. This makes the transfer of any program written in FORTRAN 77 language to another machine or operating system very easy.

The other language available throughout this work has been Pascal. C was not available on the Perq under POS. Pascal would have had undoubted advantages in data structuring and recursion. But the non-standard nature of Pascal particularly for input / output would have caused the transfer of the program between machines or operating systems to be significantly more complicated.

All the above facts have been taken into consideration as a starting point for the development of the FTDRA technique and as a basis for the present work. Not only that but the FORTRAN 77 on the Perq has the ability to deal with up to 19 files being opened simultaneously of which some could be special files. These special files are windows: either graphical windows or text ones or both.

8.1.3.2 The Set Structure :

An efficient set structure has been constructed. It consists of three main arrays with relevant pointers. This was described in Chapter 7. The set structure makes use of all the available storage space in contrast with the CELLS and SYMPTS array matrices which waste some of the storage space available to the cause and symptom equations. In the set structure a deleted set space will end up in the empty set, so no gap between the sets is lost. The pointers, namely STARTS and EMPTY ensure that.

Two Boolean laws have been implemented in subroutine ADDITM. This will eliminate redundant items within the set and also eliminate the set as a whole when an item and its complement exist in the same set. In addition the items are added in ascending order. This will also eliminate the need for any sorting.

One of the features used in programming the technique is classifying the sets into three groups. The groups are:

i) BASIC : This group contains sets that consist of basic events only.

ii) ORED : This group contains sets that have basic events with OR gates with basic inputs.

iii) OTHERS: This group contains sets with a mixture of basic events, OR gates and AND gates.

Before the application of the disjoint technique only two groups exist, namely BASIC and ORED. In other words no intermediate events exist in the sets except OR gates with basic event inputs only. These gates are left unresolved in order to reduce the total number of sets to a minimum. In addition to this any OR gate involved in the disjoint process will be changed to an AND gate with its inputs

inverted. This means that the number of redundant sets is reduced and that the resolution of a gate does not increase the total number of sets.

8.1.4 The Hazop Package On The Perq Under POS :

At the beginning of this work a major task was the transfer of the existing Aston Hazop package from the Harris system to the Perq coumputer running under POS. The reasons for this transfer are :

1. The package ran on a mainframe computer while a strategy more appropriate to current technology is to develop computer packages using single user workstations.

2. The existing package depends on the GINO-F graphical library on the Harris and this meant a degree of machine dependency. GINO-F is not universally available in all machines.

3. The updating, linking of the various parts of the package and the running of the package on the Harris was most user unfriendly. In particular the process as a whole was very slow.

All these reasons lead to the need to carry out the development of the existing Hazop package on a workstation with graphics. The chosen workstation was the Perq. Its features were most appropriate to the work in hand: the major drawback was the lack of a printer connected directly to the Perq.

The Aston package consists mainly of two parts in addition to the main

calling program. The main calling program has been rewritten to suit the Perq. The first part deals with the analysis of cause and symptom equations. This part has not required any major changes. The second part deals with the graphics and the calling of the appropriate GINO-F subroutine library. This part has undergone modification especially with the part that involves the input and the output of information being written to the screen. For example the original package supplied the user with a menu and the pointing device had to be either the cursor control key if the terminal was one of the Newbury 8000 series or the light pen if the IMLAC terminal was being used. On the Perq, the tablet with the cursor is used. The screen menu consists of the following:

A. Analysis
B. Draw
C. Edit
D. Help

The Edit option has not been implemented: its purpose is to permit modification to a cause or symptom equation. The Help option is only an explanation file for what is the meaning of Lihou's cause and symptom equations. No exit is possible from this file unless the end of the file is reached.

On the Perq a new layout has been achieved using the window manager. Two types of special files (windows) have been established. The first type of the special files is that which deals with text input and text output. The second type is that which deals with graphical output only. The screen is divided into several parts to offer the user a choice of entering the data either from the keyboard or via the tablet using the cursor. Other commands such as CLEAR, STOP or

switching between the cursor and the keyboard are also available as a part of the friendly screen manipulation. In addition if the user makes a mistake an acknowledgement notice will appear across the screen, so the user can know what the mistake was.

Most of the GINO-F library calls have been kept the same and this has been achieved through a new GINO library written for this purpose using the POS system subroutine. To accomplish this goal, procedures for cursor manipulation, windows, line and character configuration had to be written in Pascal (the POS system language) and then linked with the FORTRAN subroutines through the use of EXTERNAL functions and procedures.

8.1.5 The Hazop Package On The Perq Under PNX :

When the Perq operating system was replaced by the PNX operating system the system language was the C programming langauge. This meant that all the user interface subroutines had to be rewritten and the package was now written in the FORTRAN 77 and C programming languages only. The same screen layout has been carried over to the new version of the package but in addition a new facility was added which was PROBABILITY. This new cursor command gives the user the choice to carry out reliability calculation and to find out the TOP event probability in terms of the basic events CELLS numbers in the data file. If the probability data of the basic events are supplied then it is possible to calculate the exact TOP event probability using the FTDRA technique. At the present moment the probability expression of the TOP event is expressed in terms of the basic event entry numbers. This expression is stored in a file which can be viewed on request.

The PNX is an ICL version of the UNIX multi-tasking and time sharing technique but the present work does not use this facility since it will slow down the running speed of the package but it would be interesting to carry out such development in the future.

8.2 Uses of This Work:

The present work presents a computer aided hazop package with a new technique for obtaining the exact top event reliability. The package can be used in a number of different fields:

a) Hazard and operability studies.

b) Documentation.

c) Design.

d) Alarm system analysis.

e) Operator training and determination of human errors.

8.2.1 Hazard And Operability Studies:

The risk evaluation of a chemical plant for fire, explosion and other hazards requires a close study of many factors, including the site, structure, materials, processes, operations and material handling, operator traning, equipment, and the loss prevention program (32). To investigate the existence of potential hazards and malfunctions within a process, two main methods were in use. The first method is the hazard and operability study (hazop) and the second method is the hazard analysis study (Hazan) (23). The hazop study is a qualitative method which can somtimes extend to contain a quantitaive evaluation as well. The

Hazan method is a quantitative method which can provide full information after a hazard has been identified. The former method is the target of the present work.

The present work found that the use of cause and symptom equations was a very appropriate way to code hazop information and then to store them on a computer. Lihou had presented several rules for writing cause equations. These rules can be extended, depending on the process characteristics. The present work uses as its input already formatted cause and symptom equations. These equations are at the present time generated by hand by the hazop team. A way of automating their generation is by the use of Lihou rules in the form of modules. The modules can be put into two main categories: Linkers and units. The linkers refer to pipelines and control equipments that occur on P and I diagrams. The category of units refers to storage tanks, evaporators, separation columns, reactors and the like. From the two categories, sub-categories can be generated which may introduce the process physical states such as flow, temperature, pressure and level. Another sub-category could include the extension of the deviant states as either normal or abnormal. Further work is needed to write additional rules to generate cause and symptom equations by computer.

8.2.2 Documentation :

One of the difficult tasks for any plant management and process engineer is the writing and carrying out of start up and shutdown procedures (149). The start up procedure is normally easier than the shutdown procedure. An improper shutdown can cause expensive wastes, incomplete products and a long delay. The present work can be of help in understanding the sequences of events that arise from specific actions. The present system can help the qualitative

evaluation of the appropriate action which can be then carried out more safely and quickly. In addition to that a quantitative evaluation can be carried out using the present work to help in setting up action priorities.

8.2.3 Design :

When a scale up of a process is needed a new evaluation for the process individual components has to be carried out. Sizing up and process alteration is not straight forward and the type and sequences of deviation may change (150).

8.2.4 Operator Training And Determination Of Human Errors :

Human errors and slow response play the major part in most disastrous accidents in the chemical industries (151, 152,114). Visual aids can be of help for operator training to realise how a specific cause can lead to other causes and this could be of great help when this can be presented, say as a fault tree, on a VDU screen in the control room.

8.2.5 Alarm System Analysis :

One goal of the alarm systems is to help the operator to take necessary and the appropriate actions. It will not however guide his decisions. The operator must decide for humself the cause of the alarm (8,153). Visual aids can be of great benefit in minimising errors but the logic of how one alarm situation can initiate others is not shown. The present work could be used as a help to the operator and also to the process control engineer in setting up a more easily and understandable alarm system. It will help him in grouping and cutting down the

number of unnecessary alarms. It could be of help in setting up some linked alarms. For example a failure of a cooling pump will lead to high temperature inside a reactor. In the normal case two alarms will be triggered, high temperature and pump failure. If the pump failure alarm is not triggered then the operator will be confused and may treat the high temperature as a false alarm or as being due to a chain reaction. Hence the operator will take the wrong decision. But if the control engineer follows the causes and their consequences using a visual hazop study then the possible cause of the high temperature might be closed valve or reverse flow. So more alarms can be added to indicate the accurate situation.

8.2.6 Reliability Data :

The source of reliability data can be either available or predicted. The present package uses Boolean algebra for processing and end up with an expression that could be used as a reliability expression. So the source and the accuracy of the data will determine the accuracy of the final result (33). Since the resultant disjointed expression is in terms of the basic events and their complements then only the basic event data will influence the final result.

A sensitivity analysis can be of interest in predicting the influence of the individual items on the final result as well as the relative sensitivity of the final result to repeated basic events. The sensitivity analysis can be treated as a qualitative evaluation but evaluated by quantitative means.

One advantage of the Boolean expression generated by the FTDRA technique is that many terms will be the complement of a basic event. If the probability of a basic event is very low, the probability of its component can be

approximated by 1.0. Examination of the expression without direct substitution of the reliability data but with a knowledge of those basic events with a very low probability of occurrence may permit identification of those event sequences which are most likely to cause a given dangerous situation.

8.3 Proposals For Future Work :

8.3.1 Improving The FTDRA Set Structure :

As described in Chapter 7, the set structure consists of STARTS, SETS and NORMAL arrays. The beginning of each set has a pointer number in array STARTS. To find out the number of items in a set the subroutine NUMSET has to be called. This subroutine calls subroutines INITTAK and TAKITM and through a Do loop it counts the number of terms in the particular set. Table 8.1 shows the different subroutines that call subroutine NUMSET within them and whether the calls are within a Do loop or not. Since some of these calls are within Do loops one way of cutting the processing time would be to store the number of items in each set. The way to accomplish this is by redefining the array STARTS to be a two dimensional array. Each row represents a set. The first element in the row will store a pointer of the set starting point as before. The second element will store a counter containing the number of items in the set. This modification will substantially cut the processing time. To carry out this modification a number of changes have to be carried out on the following subroutines :

1. Subroutine INIT :-

Replace STARTS (I) = -1 by STARTS (I,1) = -1

Add STARTS (I,2) = 0

Table 8.1 : List of names of subroutines that call subroutine NUMSET within them.

Name of subroutine	Number of calls of subroutine NUMSET	
	In a Do loop	Not in a Do loop
CPYSET	-	1
TYPSET	-	1
STEP2	1	1
RSUP1	1	2
RSUP2	2	2
COLCT	-	1
DISJON	1	2
MAKDIS	-	1
RESOLV	3	1
COLSET	1	1

2. Subroutine NEWSET :-

Replace STARTS (I) = 0 by STARTS (I,1) = 0

Add STARTS (I,2) = 0

3. Subroutine ADDITM :-

Replace STARTS (SETNO) by STARTS (SETNO,1) before each RETURN.

Add STARTS (SETNO,2) = STARTS (SETNO,2) + 1

4. Subroutine DELSET :-

Add STARTS (SETNO,2) = 0

5. Subroutine DELITM :-

After the deletion of the item add

STARTS (SETNO,2) = STARTS (SETNO,2) -1

8.3.2 Alternative Languages :

Normally, FORTRAN 77 is rated as a good language for solving mathematical and algebraic equations and other trigonometric functions. It fulfils the scientists and engineers needs. The standard FORTRAN 77 source files are portable.

In hazard and operability studies (Hazop), fault tree analysis and even in reliability calculations a great deal of the process is carried out by Boolean Algebra. Much of the work involves algebraic manipulation rather than arithmetic operation. FORTRAN is not particularly appropriate to such manipulation tasks.

Many alternative high level languages can be suggested for use in carrying out fault tree analysis (148). Each language has its strengths with reference to its application from reliability techniques point of view. The suggested alternative languages are:

1) Pascal : Although there are various dialects of the Pascal programming language, they all share the strong features of a well structured high level language and one of them is the recursion facility. Recursion is simply defined as a technique that

allows a formulated procedure to call itself again within itself. This means that the number of calls and the stages for carrying out a certain task are minimised. FORTRAN in general does not have this facility but some BASIC programming languages like QUICKBASIC and FASTBASIC do have it. Page and Perry (143,145) have demonstrated the usefulness of the recursion impementation in their technique for calculating the exact TOP event probability using the USCD Pascal.

2) LISP : Like Pascal, LISP is a high level language with different versions and language extensions. LISP is ideal for sets, groups and Boolean manipulations. It has more logic orientation than the FORTRAN. Its advanced facilities make it a most appropriate language for coding fault tree logic and cut sets evaluation.

8.3.3 Alternative Coding Of Hazop Data :

The aims of carrying out this work have been discussed in Chapter 1. One of the problems of handling the information of hazop studies is how to store it on a computer in such a form that it can be coded with ease according to some rules. Lihou coding has been used in The Aston Hazop Package and found to be the most appropriate way of storing the hazop information. As mentioned in Chapter 2, two types of equations are in use: cause and symptom equations. A symptom equation uses the AND operator only while a cause equation uses both the AND and the OR operators. This means that Lihou coding is limited to coherent systems only. Alternatively only a 2-state system for failures can be expressed by this coding. A 2-state system for failures shows its components' state as either a failure or a success.

There are other expressions used in industry for stating the malfunctions in a process which are still within workable conditions. For example a flow controller may have the following responses :

a- Very high flow

b- High flow

c- Normal flow

d- Low flow

e- Very low flow

Obviously there are three states for this system :

1- The process is functioning satisfactorily (third response).

2- The process is operating with some malfunctions and some alarms set: second and fourth responses are such states.

3- The process is in an unworkable condition and most of the alarms are set: such responses are the first and the fifth.

Similar arguments can be applied to pressure controllers, temperature controllers and level controllers. Such systems are 3-state system with two workable states and one failure state. In order to apply this example to the present work an assumption has to be made first. The 3-state system has to be treated as one success state and one failure state according to the TOP event. This will mean that the probability of the top event has a lower bound value and a higher bound value. The second state represents the lower bound value of the TOP event which has a lower probability value for the system failure while the third state represents the higher bound value of the TOP event which is the Higher and possible system failure. In both cases Lihou coding can be used but with some

modification and care.

To modify the Lihou coding, a new symbol could be introduced. This would be a colon (:) to go in between the guide word and the property word. For example a No FLow in line 1 is coded according to Lihou as L1(11). In the new way the coding will be L1(1:1). This will not affect the other rules found by Lihou but this modification will be useful in the 3-state system coding. For example an L(12) means LESS FLOW in L1. The Guide word LESS indicates both slight deviation and severe deviation. In the 3-state system another constraint has to be added such as LOW which is not a serious deviation: if no action is taken then this could develop the next VERY LOW (LESS) deviation. To code this deviation another digit could be used with any guide word:

a) The figure 1 would indicate the normal deviation;

b) The figure 2 would indicate the extreme deviation.

So a VERY LOW flow in L1 would be L1(1:22). Similarly a high flow in L1 is, using Lihou's coding, L1(13). The new coding would be L1(1:32).

A similar modification could be carried out to separate the component identification number from the two words. For the sake of argument, suppose the component to be ammonia gas and its identification number 3. Then the new coding for no flow in L1 would be L1(1:1:3). This gives clearer presentation of the coding. To express very high flow of ammonia in L1 the new coding would be L1(1:32:3). This is quite useful when two-phase fluids exist in the same line: for example when a gas and a liquid flow countercurrently or cocurrently in the same line. Similarly consider saturated steam flowing vertically in line L1 with a control valve V1 in the line. If the temperature drops for any reason

condensation would appear in the line and the resultant water will flow downward and partially or totally block the control valve. According to Lihou coding less flow of steam in L1 could be either due to valve V1 being partially blocked owing to scaling or due to reverse flow of condensed water.

L1(126) = V1(-1) + L1(165)

The figures in the notation have the following meaning:

1 refers to flow

2 refers to LESS

5 refers to water

6 refers to steam/reverse flow

-1 refers to a partial blockage

With the new coding the reason for less flow (normal deviation) has the possibility of steam condensation or valve partially blocked. If very much less flow of steam does occur (severe deviation) then this could be due to simultaneous steam condensation and valve partial blockage. Each case has its own cause equation as follows:

L1(1:21:6) = V1(-1) + L1(1:61:5)

L1(1:22:6) = V1(-1) * L1(1:62:5)

The first equation represents the normal deviation state while the second equation represents the severe deviation.

Another modification to Lihou coding would be the addition of a slash

(/) to represent an Exclusive OR gate. This is a crucial modification to ensure a complete presentation for all coherent fault trees and not only special cases where only OR and AND gates exist.

8.5 Conclusion :

This work set out :

- to investigate the use of fault tree analysis in hazop studies.

- to generate and analyse fault trees on a graphics screen through a friendly user interface;

- to enhance the analysis to permit the assessment of top event probabilities.

The major result of the work has been the development of a new technique for the calculation of the exact probability of occurrence of the TOP event in a tree containing repeated events.

The achievements of the work can be summarised as follows :

a) A new technique, based on FATRAM top-down Boolean reduction approach, has been developed together with the disjoint technique. The resultant algorithm is called the Fault Tree Disjoint Reduction Algorithm (FTDRA).

b) FTDRA technique, in one of its steps, resolves all the intermediate events except OR gates which have its inputs as basic events. This results in a near minimal cut sets expression.

c) A disjoint technique based on Bennetts' disjoint technique (76) has been applied to convert the near minimal cut sets into a sum of products expression. The Boolean domain is mapped into the probability domain, permitting

the resultant expression to be interpreted directly as a probability expression.

d) A hazop package has been developed on the Perq workstation under two different operating systems. The first package was coded in FORTRAN 77 and Pascal under the POS operating system. The second package has been coded in FORTRAN 77 and C under the PNX operating system.

e) A friendly menu drive user interface has been developed for the hazop package on the Perq workstation.

f) A version of the graphical library GINO-F has been written for the Perq workstation.

g) A windowing environment has been successfully developed for the hazop package on the Perq workstation.

h) A method has been suggested to modify Lihou's method of coding to identify more accurately the causes of different deviations.

The use of hazard and operability studies is now accepted throughout the process industries. Computer aided design is increasingly widely used at all stages of the design process. The integration of hazard and operability studies into CAD using techniques such as those developed in this work seems an inevitable development.

APPENDICES

APPENDIX A

RULES FOR WRITING CAUSE EQUATIONS

APPENDIX A-1 : CAUSE EQUATIONS FOR PIPELINES

APPENDIX A-2 : CAUSE EQUATIONS FOR VESSELS

APPENDIX A-1 : CAUSE EQUATIONS FOR PIPELINES

In writing cause equations, the following rules have been devised by Lihou, to avoid illogical fault trees.

1. Pipelines

Index	Meaning	Cause
11	FLOW NO	a) No flow in the line(s) immediately upstream b) No flow at the node where the line leaves an equipment c) The supply tank empty d) A valve shut in the line e) A filter fully blocked in the line f) A pump in the line stopped
12	FLOW LESS	a) Less flow in the supply line(s) immediately upstream b) Less flow at node where the line leaves an equipment c) Vent blocked on a storage tank d) More flow in a branch line (judged by the relative magnitude of the normal flows) e) A valve insufficiently open, filter or exchanger tubes partly blocked, in a line without flow or pressure control f) The flow control valve fully open and a valve in the line insufficiently open or some other restrictions g) A valve insufficiently open down stream of a pressure control h) A pressure or flow controller set too low i) A pressure or flow transmitter indicating too high j) A pneumatic trip valve leaking to vent k) Leak(s) down stream of a flow transmitter l) Bypass around a pump open or leaking m) Pump delivery reduced by cavitation or speed reduction
13	FLOW MORE	a) More flow in the line(s) down stream b) More flow at the nodes where the line leaves or enters an equipment c) Leak in the line(s) down stream without flow control d) Leak upstream of flow controllers in down stream line(s). Leaks from a line are equated to drain or bleed valves open or leaking, filters leaking, etc. e) Pressure or flow controller set too high f) Pressure or flow transmitter indicating too low

Index	Meaning	Cause
		g) Control valve stuck open h) Bypass around a control valve fully open i) Bypass flow around a pump too low
14	FLOW AS WELL AS	a) Contamination of inlet lines b) Contamination in supply tank c) Supply tank level low allowing air plus liquid to enter the discharge pipe d) Steam or nitrogen purge valves leaking e) Tubes leaking in exchangers f) Steam trap leaking or its bypass valve leaking
15	FLOW FLUCTUAT-ION	a) Caused by an on/off controller or an unstable control loop
16	FLOW REVERSE	a) Caused by differential pressures, unequal levels unequal levels and no non-return valves leaking
17	FLOW OTHER THAN	a) Caused by gas or vapour entering the discharge pipe from an empty supply tank
22	TEMPERAT-URE LESS	a) Low temperature at the node where the line leaves an equipment b) Trace heating not on or failed c) Temperature controller set low d) Temperature indicator indicating too high
23	TEMPERAT-URE MORE	a) High temperature at the node where the line leaves an equipment b) Trace heating on when it should be off c) Temperature controller set high d) Temperature indicator indicating too low
32	PRESSURE LESS	a) Less pressure in the supply line b) Less pressure at the node where the line leaves an equipment c) A valve insufficiently open down stream of a pressure controller d) Pressure or flow controller set low e) Presssure or flow transmitter indicating too high f) Pneumatic trip valve leaking to vent g) Control valve fully open and a valve in the line insufficiently open or other blockage
33	PRESSURE HIGH	a) High pressure at the beginning of a line and no pressure or flow control in the line b) High pressure at the end of the line and flow is controlled c) Pressure or flow controller set high d) Control valve stuck open e) Bypass cvalve around a control valve is fully open f) Flow is controlled and flow tramsmitter is indicating

Index	Meaning	Cause
		too low g) Pressure is controlled and pressure transmitter is indicating too low h) Pressure regulator set high or its valve stuck open (down stream pressure controller valve only if two are used in series)

2. Pipeline Junctions

Where line L3 is supplied by lines L1 and L2, the following rules are used to construct cause equations for line L3 :

1. No flow is caused by no flow in L1 AND no flow in L2.

2. For less and more flow, temperature and pressure an

OR gate is use between the equivalent deviations in L1

and L2.

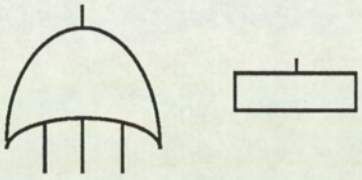
APPENDIX A-2 : CAUSE EQUATIONS FOR VESSELS

In writing cause equations, the following rules have been devised by Lihou, to avoid illogical fault trees.

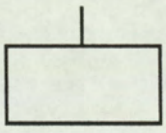
Index	Meaning	Cause
41	LEVEL NO	a) Means the vessel is empty
42	LEVEL LOW	a) No flow into vessel b) Less flow into vessel c) Level transmitter indicating too high d) Level controller set low e) Low isolating valve on the level indicator or transmitter is closed f) Level control valve on the discharge line is stuck open g) Bypass valve open around a down stream level control valve
42	LEVEL HIGH	a) Level transmitter indicating too low b) Level controller set high c) Upper isolation valve on the level indicator or transmitter is closed d) Level control valve stuck open if on a supply line or stuck closed if on the discharge line e) Bypass valve open around the level control valve in a supply line f) Kick-back liquid flow going to a non-supply storage tank

APPENDIX B : SYMBOLS COMMONLY USED IN FAULT TREE GRAPHICAL REPRESENTATION.

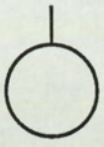
AND Gates : Coexistance to all inputs required to produce output.



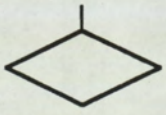
OR Dates : Output will exist if at least one input is present.



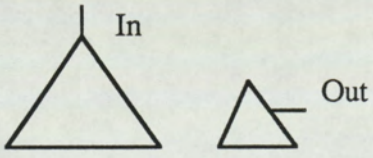
RECTANGLE : A fault tree usually resulting from the combination of more basic faults acting through logic gates.



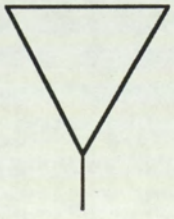
CIRCLE : A basic component fault - An independent event.



DIAMOND : A fault event not developed to its cause.

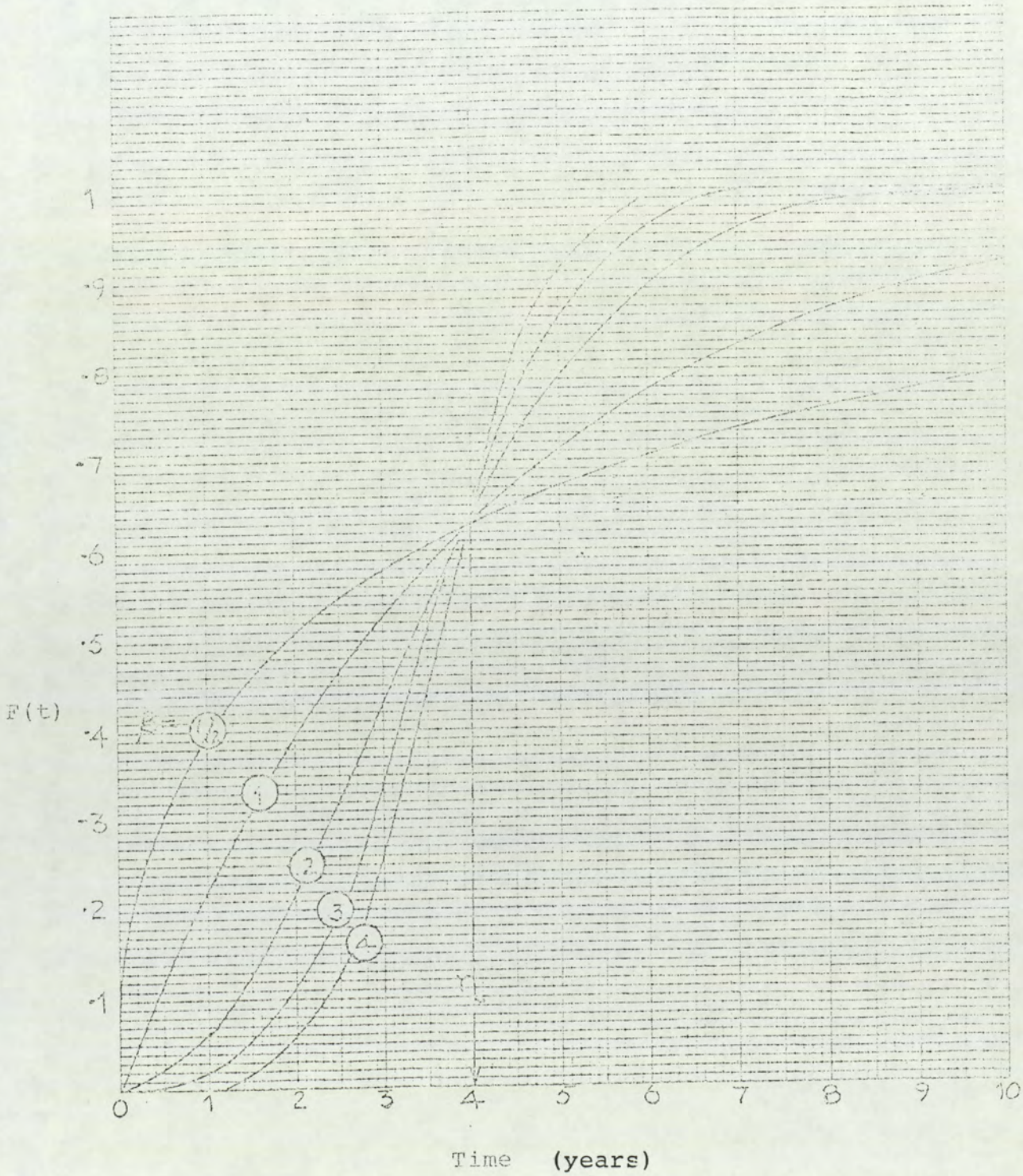


TRIANGLE : A connecting or transfer symbol.

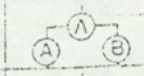
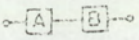
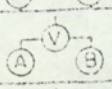
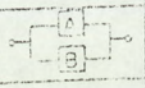
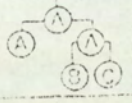
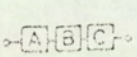
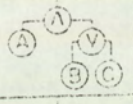
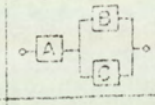
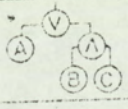
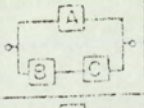
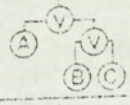
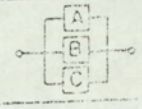
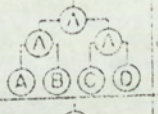
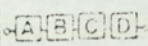
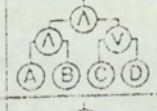
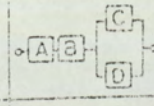
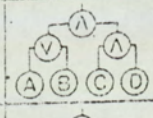
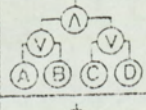
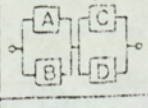
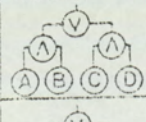
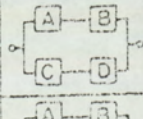
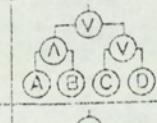
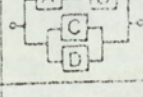
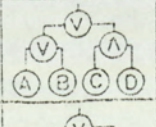
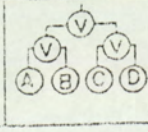
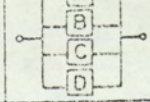


UPSIDE DOWN TRAINGLE : A similar transfer but not identical to the like identified input.

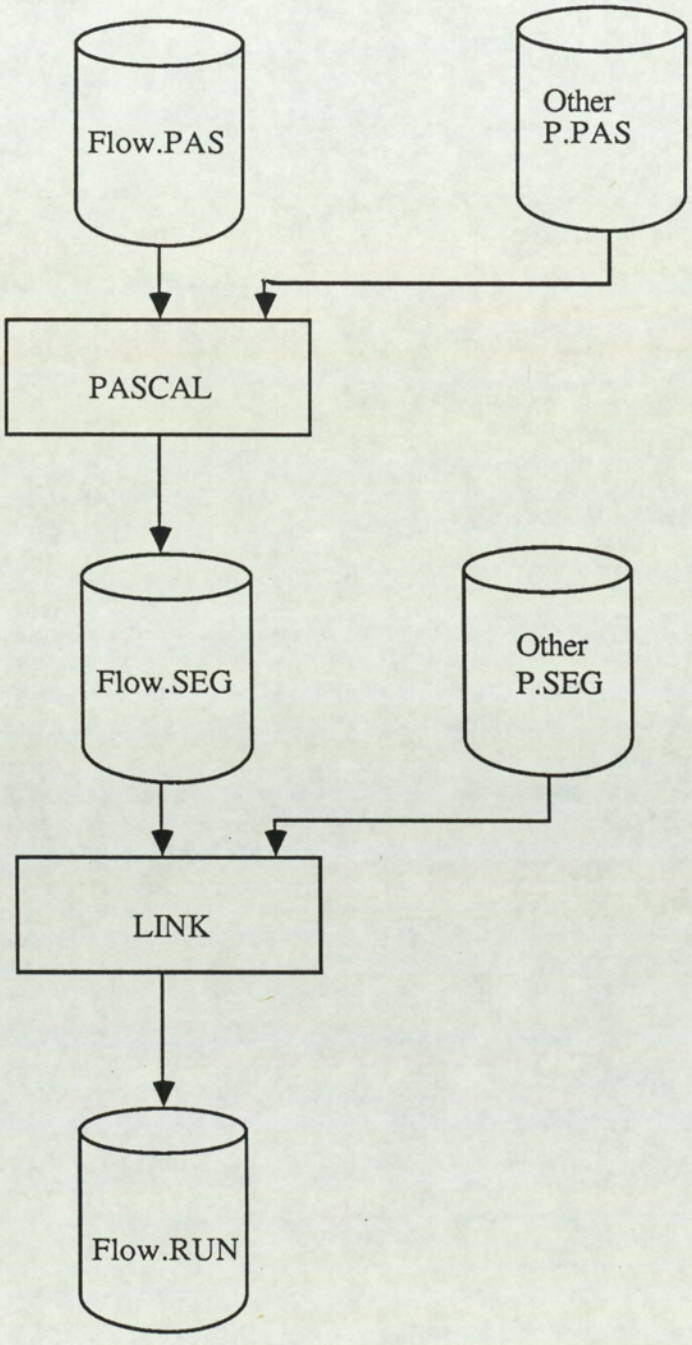
APPENDIX C : FAILURE PROBABILITY BY THE WEIBULL DISTRIBUTION



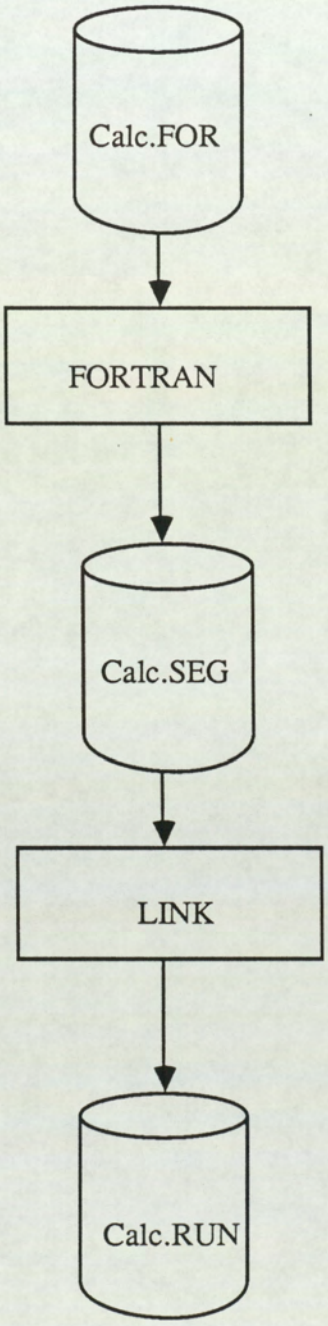
APPENDIX D : STANDARD PATTERNS AND THEIR MATHEMATICAL FORMS

Pattern Number	Pattern	Reliability Diagram	Pattern EOT	Mathematical Form
P_1			ABΛ	ab
P_2			ABV	1-(1-a)x(1-b)
P_3			ABCΛΛ	abc
P_4			ABCVΛ	a(b+c-bc)
P_5			ABCΛV	1-(1-a) x (1-bxc)
P_6			ABCVV	1-(1-a)x (1-b)(1-c)
P_7			ABΛCDΛΛ	abcd
P_8			ABΛCDVΛ	ab(1-[1-c]x [1-d])
P_9		FORBIDDEN FOR ORIENTED TREES		
P_{10}			ABVCDVΛ	[1-(1-a)x(1-b)] x [1-(1-c)x(1-d)]
P_{11}			ABΛCDΛV	1-(1-axb)x (1-cxd)
P_{12}			ABΛCDVV	1-(1-axb)x (1-c)(1-d)
P_{13}		FORBIDDEN FOR ORIENTED TREES		
P_{14}			ABVCDVV	1-(1-a)(1-b) x(1-c)(1-d)

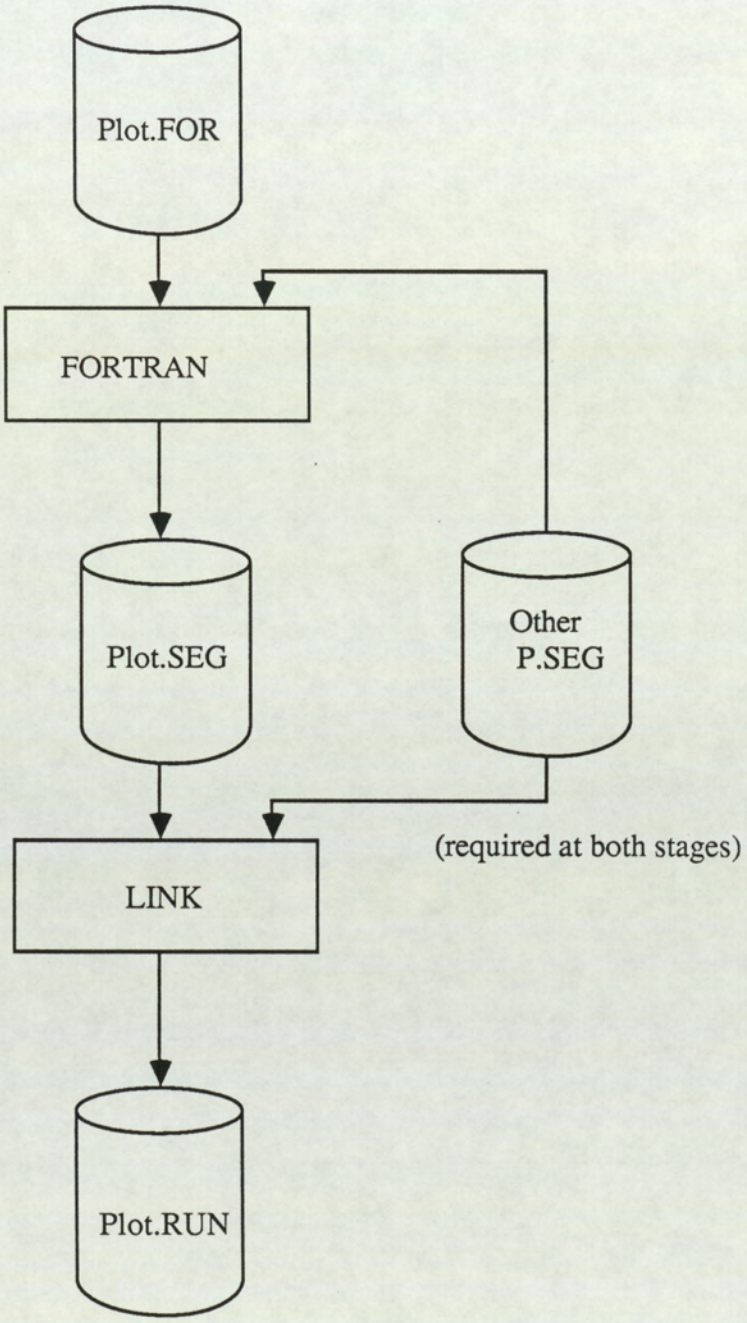
APPENDIX E.1 : CONVERTING A PASCAL PROGRAM



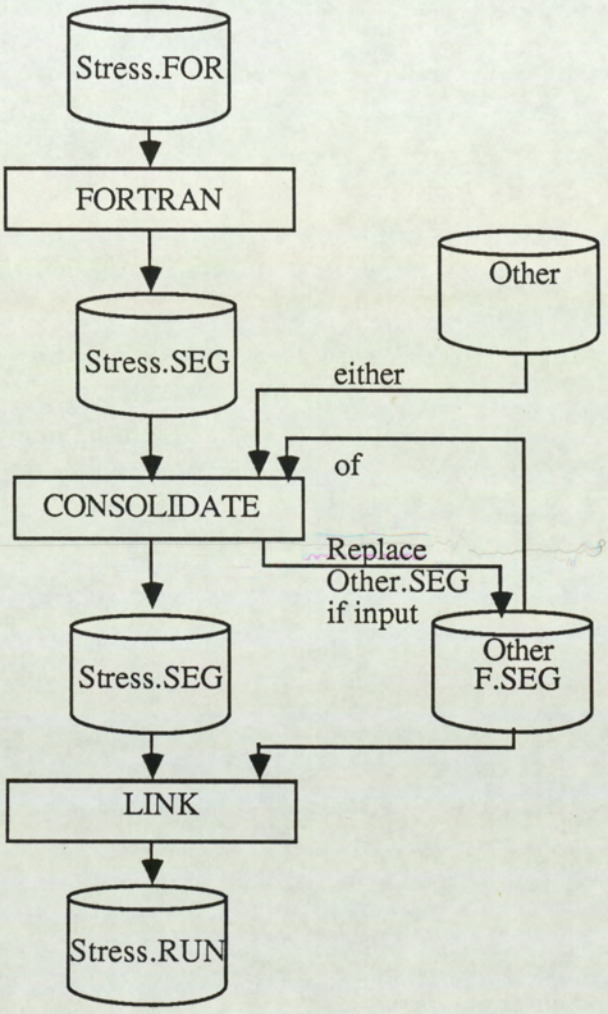
APPENDIX E.2 : CONVERTING A SIMPLE FORTRAN PROGRAM



APPENDIX E.3 : CONVERTING A FORTRAN PROGRAM WHICH REFERENCES A PASCAL MODULE



APPENDIX E.4 : CONVERTING A FORTRAN PROGRAM WHICH REFERENCES AN INDEPENDENTLY COMPILED FORTRAN UNIT



APPENDIX F : LISTING OF CAUSE AND SYMPTOM EQUATIONS OF THE SOLVAY PROCESS

NOTE : SYMPTOM EQUATIONS ARE WRITTEN WITH - RATHER THAN ->

'L1(11)=V1(0)+PR1(0)+NH3(41)'
'L1(12)=V1(-1)+FI1(1)'
'L1(13)=V1(1)+FI1(-1)'
'L1(32)=PR1(-1)+NH3(42)'
'L1(33)=PR1(1)'
'L2(11)=P1(0)+T1(41)*(T4(41)+V5(0))'
'L2(12)=P1(-1)'
'L2(13)=P1(1)'
'L2(142)=T1(23)*T1(532)*L2(22)+L1(142)'
'L2(22)=L2(73)'
'L2(23)=T1(23)+T4(23)'
'L2(522)=T1(533)+T3(533)'
'L2(535)=T4(535)'
'L1(11)→N1(11)*N1(31)*N2(11)*N4(125)*N4(22)*N6(11)'
'L1(12)→N1(12)*N1(32)*N2(12)*N2(22)*N4(125)*N4(22)*N6(12)'
'L1(13)→N1(13)*N1(33)*N2(13)*N6(13)*N6(23)'
'L2(11)→N1(32)*N2(13)*N3(11)*N4(12)*N4(141)*N5(42)*N7(11)'
'L2(13)→N1(33)*N2(142)*N3(13)*N4(525)*N5(43)*N7(13)'
'L2(142)→N1(33)*N5(142)*N7(142)'
'L2(22)→N2(22)*N3(22)*N7(22)'
'L2(23)→N2(13)*N2(23)*N3(23)*N4(125)*N4(23)*N7(23)'
'L2(522)→N4(522)'
'L2(533)→N2(13)'
'L3(12)→N1(33)*N5(43)'
'L3(13)→N1(32)*N2(13)*N4(13)*N4(125)*N4(23)*N5(42)*N6(23)'
'SJ(72)→N4(23)*N5(23)'
'L3(11)=P2(0)'
'L3(12)=P2(-1)+N4(12)'
'L3(13)=P2(1)'
'L3(125)=N4(125)'
'L3(141)=N4(141)'
'L3(23)=N4(23)'
'L3(522)=N4(522)'
'L5(11)=V3(0)+PR2(0)+CO2(41)'
'L5(12)=V3(-1)+FI3(1)'
'L5(13)=V3(1)+FI3(-1)'
'L5(32)=PR2(-1)+CO2(42)'
'L5(33)=PR2(1)'

'L3(11)→N9(13)*N9(22)*N10(11)*N11(12)*N12(42)*N14(22)*N13(32)*N14(11)'
'L3(12)→N9(13)*N9(23)*N10(12)*N11(12)*N11(146)*N14(22)*N14(12)'
'L3(13)→N10(13)*N11(13)*N11(532)*N14(22)*N13(33)*N14(13)'
'L3(125)→N9(13)*N9(23)*N11(128)*N11(532)*N14(22)'
'L3(141)→N9(131)*N11(128)*N14(22)'
'L3(23)→N9(131)*N9(23)*N10(23)*N11(128)*N12(23)*N14(23)'
'L3(522)→N11(128)*N11(146)'
'L5(11)→N8(11)*N9(11)*N11(128)*N11(145)*N12(22)*N13(11)*N13(32)'
'L5(12)→N8(12)*N9(12)*N11(128)*N12(22)*N13(32)*N14(22)'
'L5(13)→N8(13)*N9(13)*N12(23)*N13(33)'
'L4(13)→N9(131)*N9(23)*N11(144)*N12(42)*N13(32)*N14(22)'
'L6(11)=N9(11)'
'L6(12)=N9(12)'
'L6(13)=N9(13)'
'L6(131)=N9(131)'
'L6(22)=N9(22)'
'L6(23)=N9(23)'
'L7(11)=N2(11)+L2(13)*L1(12)'
'L7(12)=N2(12)'
'L7(13)=N2(13)'
'L7(22)=N2(22)'
'L7(23)=N2(23)'
'L9(11)=P3(0)+T3(41)'
'L9(12)=P3(-1)'
'L9(13)=P3(1)'
'L9(23)=T3(23)'
'L9(522)=T3(533)'
'L7(11)→N15(12)*N15(32)*N18(22)'
'L7(12)→N15(12)*N15(32)*N18(22)'
'L7(13)→N15(13)*N15(33)*N18(148)*N18(23)*N21(23)'
'L7(23)→N18(23)*N21(23)'
'L6(11)→N15(12)*N15(32)*N16(11)*N20(11)'
'L6(12)→N15(12)*N15(32)*N16(12)*N20(12)'
'L6(13)→N15(13)*N15(33)*N16(13)*N20(13)'
'L6(131)→N18(148)*N18(23)*N21(23)'
'L6(23)→N18(23)*N21(23)'
'L9(11)→N15(32)*N16(13)*N16(141)*N17(11)*N18(12)*N18(22)*N19(42)*N21(11)'
'L9(12)→N15(32)*N17(12)*N18(12)*N18(23)*N19(42)*N21(12)'

'L9(13)→N15(33)*N17(13)*N18(13)*N19(43)*N21(13)'
'L9(23)→N16(141)*N17(23)*N18(23)*N21(23)'
'L9(522)→N18(522)'
'L8(12)→N15(33)*N19(43)'
'L8(13)→N15(32)*N18(22)*N20(13)*N21(23)'
'L10(11)=N16(11)'
'L10(12)=N16(12)'
'L10(13)=N16(13)'
'L10(141)=N16(141)'
'L8(11)=V4(0)+L8(0)'
'L8(12)=V4(-1)+N18(12)'
'L8(13)=V4(1)+N18(13)'
'L8(23)=N18(23)'
'L8(522)=N18(522)'
'L11(11)=V5(0)+T4(41)'
'L11(12)=L8(12)+V5(-1)'
'L11(13)=L8(13)+V5(1)'
'L11(142)=T3(23)*T3(532)*T4(2$\bar{2}$)'
'L4(11)=V2(0)+L4(0)'
'L4(12)=N11(12)+V2(-1)'
'L4(128)=L4(12)+N11(128)'
'L4(13)=N11(13)+V2(1)'
'L4(145)=N11(145)'
'L4(146)=N11(146)'
'L4(22)=N11(22)'
'L4(23)=N11(23)'
'L4(532)=N11(532)'
'END'

APPENDIX G : LISTING OF THE SOURCE PROGRAMS OF THE HAZOP POS VERSION

```
**************************************************************************
*OBJECTIVE : THE HAZOP PACKAGE WRITTEN TO RUN ON THE HARRIS HAS
*           BEEN MODIFIED TO BE RUN ON THE PERQ WORKSTATION
*           UNDER THE POS OPERATING SYSTEM. IT ENABLES THE USER
*           TO SELECT THE PACKAGE ROUTINE OF HIS/HER CHOICE AND
*           THE PROGRAM PROVIDES A CONSTANT INTERACTION BETWEEN
*           HIM/HER AND THE TERMINAL. THE PACKAGE MAKES USE OF
*          THE TABLET'S CURSOR TO CHOOSE FROM THE PROVIDED MENU.
**************************************************************************


    INTEGER CLR,CLC, SMR, SMC, NML, ERRNO, VALUE

    PARAMETER ( CLR = 500,
   +        CLC = 10,
   +        SMR = 50,
   +        SMC = 20,
   +        NML = 12)

    INTEGER CELLS ( CLR, CLC ), SYMPTS ( SMR, SMC )
    CHARACTER NAMES ( CLR ) * ( NML )

    LOGICAL RQST,CHECK,TEST,YESNO
    CHARACTER SELCTN(5)*80,HELP(5)*80

     DATA HELP(1),HELP(2),HELP(3),HELP(4)/'  ANALYSE CALLS SBR.
    1 EQTS FOR THE ANALYSIS OF CAUSE & SYMPTOM EQNS',' DRAW CALLS SBR.
    1TREES FOR THE DRAWING OF FAULT TREES FROM C & S EQNS',' TRANSLATE
    1 CALLS SBR. WORDS FOR THE TRANSLATION OF C & S EQNS INTO WORDS','
    1 EDIT CALLS SBR.  EDIT FOR THE EDITING OF C & S EQNS'/


    DATA(SELCTN(I),I=1,5)/'  A.  ANALYSE CAUSE & SYMPTOM EQNS USING SBR
    1. EQTS',' B.  DRAW FAULT TREES FROM CAUSE & SYMPTOM EQNS USING SBR
    1. TREES',' C.  TRANSLATE CAUSE & SYMPTOM EQNS.INTO WORDS USING SBR
    1. WORDS',' D.  EDIT CAUSE & SYMPTOM EQNS. USING SBR. EDIT',' E.
    1 DISPLAY DETAILED EXPLANATION OF PACKAGE'/


 1   ERRNO = 0


C
C****  SELECT GRAPHICAL OR NON-GRAPHICAL OPTIONS
C
    PRINT *,'-------------------------------------------------'
    PRINT *,' ARE THERE GRAPHICAL FACILITIES AVAILABLE AT YOUR TERMINA
    1L'
    PRINT *
    PRINT *,' INPUT YES OR NO'
    PRINT *,'-------------------------------------------------'
C
C**** SUBPROGRAM YESNO(TYPE) ENABLE YES OR NO ANSWER
```

```
C
      RQST=YESNO(TYPE)
      IF(RQST)THEN

       CALL DRIVER ( 1, VALUE, ERRNO )
       IF ( ERRNO .NE. 0 ) THEN

        GO TO 1

       END IF


      DO 5 MM=1,4
      CALL GRFTR1(IFLAG,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
C
C**** SIMULATION OF EXECUTION OF INDIVIDUAL PARTS OF THE PACKAGE FROM
C**** GRAPHICAL MODE OF OPERATION BY MEANS OF A FLAG RETURNED FROM EACH
SBR.
C
      IF(IFLAG.EQ.1)THEN
      PRINT *,'CAUSE & SYMPTOM EQNS ANALYSED'
      PRINT *
      ELSEIF(IFLAG.EQ.2)THEN
      PRINT *,'FAULT TREES DRAWN'
      PRINT *
      ELSEIF(IFLAG.EQ.3)THEN
      PRINT *,'CAUSE & SYMPTOM EQNS TRANSLATED'
      PRINT *
      ELSEIF(IFLAG.EQ.4)THEN
      PRINT *,'CAUSE & SYMPTOM EQNS EDITED'
      PRINT *
      ENDIF
    5 CONTINUE
      STOP
      ELSE
C
C****
C****         SELECT  NON-GRAPHICAL OPTIONS

      PRINT *
      PRINT *,'-------------------------------------------------'
      PRINT *,'HAVE YOU HAD PREVIOUS EXPERIENCE USING THIS PACKAGE?'
      PRINT *,'PLEASE ENTER YES OR NO'
      PRINT *,'-------------------------------------------------'
      RQST=YESNO(TYPE)
      DO 15 I=1,4
      IF(RQST)THEN
      CALL EXPRC(HELP,IFLAG,SBR,I,CHECK,TEST,CELLS,CLR,CLC,SYMPTS,
     *         SMR,SMC,NAMES,NML)
      ELSE
      CALL GUIDE(SELCTN,IFLAG,M1,I,SBR,CELLS,CLR,CLC,SYMPTS,
     *         SMR,SMC,NAMES,NML)
      ENDIF
      IF(SBR.EQ.1.)THEN
      PRINT *,'CONFIRMATION OF USER CHOICE '
```

```
      PRINT 20,SELCTN(M1)
   20 FORMAT(A)
      PRINT *
      PRINT *
      PRINT *,'-----------------------------------------------'
      ENDIF
C
C****   SIMULATION OF EXECUTION OF INDIVIDUAL PARTS OF PACKAGE BY MEANS
C****   OF FLAG MESSAGES FROM SBR'S USING NON-GRAPHICAL MODE.
C
      IF(IFLAG.EQ.1)THEN
      PRINT *,'CAUSE & SYMPTOM EQNS ANALYSED'
      PRINT *
      ELSEIF(IFLAG.EQ.2)THEN
      PRINT *,'FAULT TREES DRAWN ON NON-GRAPHICAL VDU'
      PRINT *
      ELSEIF(IFLAG.EQ.3)THEN
      PRINT *,'CAUSE & SYMPTOM EQNS TRANSLATED'
      PRINT *
      ELSEIF(IFLAG.EQ.4)THEN
      PRINT *,'CAUSE & SYMPTOM EQNS EDITED'
      PRINT *
      ENDIF
   15 CONTINUE
      ENDIF
      STOP
      END


      SUBROUTINE DRIVER ( ENTRY, VALUE, ERRNO )
*
*
****   SUBROUTINE TO SET UP GINO DRIVER

*           ENTRY = 1 FOR INITIALISATION OF TERMINAL
*           ENTRY = 2 FOR CALL OF DRIVER
*           ENTRY = 3 TO RETURN VALUE OF TERMINAL TYPE


*           ERRORS GENERATED ARE
*
*       FOR ENTRY = 1

*            ERRNO = 1 IF TERMINAL TYPE NUMBER NOT WITHIN RANGE
*
*       FOR ENTRY = 3
*
*            ERRNO = 50 IF DRIVER NOT PREVIOUSLY CALLED
*


*     THIS S/R TO BE UPDATED FOR THE USE OF NEW GRAPHICS TERMINALS
*

      INTEGER ENTRY, TRMNL, ERRNO, VALUE
```

```
      LOGICAL YESNO
      REAL TYPE
      SAVE TRMNL

      IF ( ENTRY .EQ. 1 ) THEN

*       INITIALISATION

 1     PRINT *, ' ENTER THE TYPE OF TERMINAL YOU ARE USING:'
       PRINT *, ' IS IT'
       PRINT *
       PRINT *, '     1. A NEWBURY 8000 SERIES'
       PRINT *, '     2. A DYNAGRAPHICS ( IMLAC )'
       PRINT *, '     3. A POS OPERATING SYSTEM ( PERQ )'
       PRINT *
       PRINT *, ' ENTER THE NUMBER OF THE TYPE, I.E. 1 , 2 OR 3'
       READ *, TRMNL
       IF ( TRMNL .LT. 1 .OR. TRMNL .GT. 3 ) THEN

        PRINT *, 'WAS YOUR ENTRY CORRECT?'

        IF ( .NOT. YESNO ( TYPE ) ) THEN

         GO TO 1

        ELSE

         ERRNO = 1
         RETURN

        END IF

       ELSE

        RETURN

       END IF

      ELSE IF ( ENTRY .EQ. 2 ) THEN

*       CALL GINO DRIVER

101    IF ( TRMNL .EQ. 1 .OR. TRMNL . EQ. 2 ) THEN

*      CALL PERQ
       PRINT *,' SORRY, NO ACCESS TO THIS TYPE AT THE MOMENT'
       PRINT *,'DO YOU WANT TO TRY AGAIN ?'
        IF ( YESNO ( TYPE ) ) THEN
        PRINT *,'INPUT THE NUMBER OF THE TYPE'
         READ *,TRMNL
         GO TO 101
          ELSE
         STOP
        END IF
```

```
      ELSE IF( TRMNL .EQ. 3 ) THEN
     CALL PERQ
     END IF
     RETURN

       ELSEIF ( ENTRY .EQ. 3 ) THEN
      IF ( TRMNL .LT. 0 ) THEN

       ERRNO = 100

       RETURN

      END IF

      VALUE = TRMNL

     END IF

     END


C****  SUBROUTINE GUIDE OPERATES A MORE ILLUSTRATED NON-GRAPHICAL
C****  SELECTION MODE IN WHICH THE USER IS PRESENTED WITH A MENU
C****  LIST OF THE PACKAGE ROUTINES - SELECTION IS MADE BY INPUTING
C****  A, B, C, D OR E  EACH CHARACTER CORRESPONDING TO A PACKAGE
C****  ROUTINE.
C

     SUBROUTINE GUIDE(SELCTN,IFLAG,M1,K,SBR,CELLS,CLR,CLC,SYMPTS,
     *              SMR,SMC,NAMES,NML)

     INTEGER CLR,CLC, SMR, SMC, NML
     INTEGER CELLS ( CLR, CLC ), SYMPTS ( SMR, SMC )
     CHARACTER NAMES ( CLR ) * ( 12 )
     LOGICAL FILE,CHECK,TEST,TEST1
     CHARACTER TEX*4,X1*1,SELCTN(5)*(*),X*1
     INTEGER FIRST

C
C****  ARRAY SELCTN CONTAINS THE MENU LIST OF THE PACKAGE ROUTINES
C

     DO 25 M=K,5
     PRINT 15,(SELCTN(J),J=1,5)
  15  FORMAT(A/)
     PRINT *
     PRINT *,'INPUT YOUR CHOICE BY PRINTING A, B, C, D OR E'
      PRINT *
      PRINT *,'TO END RUN PRINT STOP'
     PRINT *,'-----------------------------------------------'
     READ(UNIT=5,FMT='(A)')TEX
     IF(TEX.EQ.'STOP') STOP
     IF(.NOT.(TEX.GE.'A'.AND.TEX.LE.'E'))THEN
     PRINT *,' YOUR INPUT SHOULD BE A, B, C, D OR E'
```

```
      ENDIF
      SBR=1.

C
C****      TEST FOR SELECTION OF PACKAGE ROUTINE
C

      DO 30 M1=1,5
      N=FIRST(SELCTN(M1))
      N1=FIRST(TEX)
      X1=TEX(N1:N1)
      X=SELCTN(M1)(N:N)

C
C****   CHECK IF DATA FROM ANALYSIS OF CAUSE AND SYMPTOM EQNS. HAVE
C****   BEEN STORED IN DETAILS FROM PREVIOUS RUN IF NOT OPTION IS
C****   GIVEN TO STORE THE INTERMEDIATE DATA IN A DATAFILE AND USED
C****   AS INPUT TO THE OTHER PACKAGE ROUTINES.- THE DATA OBTAINED
C****   FROM THE ANALYSIS OF CAUSE AND SYMPTOM EQNS FORM THE DATA
C****   STRUCTURE FOR THE REST OF THE ROUTINES IN THE PACKAGE.
C


C
C****      TEST FOR SELECTION OF ROUTINE
C

 9000 IF ( .NOT. X1.EQ.X) GO TO 9010
      IF(M1.EQ.1)THEN
      TEST1=TEST.AND.CHECK
      IF(TEST1)THEN
      PRINT *,'-------------------------------------------------------'
      PRINT *,'WARNING CAUSE & SYMPTOM EQNS HAVE ALREADY BEEN ANALYSED'
      PRINT *,'DATA HAVE BEEN STORED IN DATA FILE OPEREQNS'
      PRINT *
      PRINT *,'-------------------------------------------------------'
      IFLAG=5
      RETURN
      ELSE
      CHECK=FILE(I)
      CALL EQTS(IFLAG,CHECK,TEST,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
      ENDIF
      ELSEIF(M1.EQ.2)THEN
      CALL TREES1 (IFLAG,TEST)
      ELSEIF(M1.EQ.3)THEN
      CALL WORDS(IFLAG,TEST)
      ELSEIF(M1.EQ.4)THEN
      CALL EDIT(IFLAG,TEST)
      ELSEIF(M1.EQ.5)THEN
      CALL DEXPLN
      IFLAG=5
      ENDIF
      RETURN
      GO TO 9000
 9010 CONTINUE
```

```
 30 CONTINUE
 25 CONTINUE
    END


C****     SUBROUTINE EXPRC OPERATES NON-GRAPHICAL SELECTION MODE
C****     BY INPUT OF KEYWORDS
C
      SUBROUTINE EXPRC(HELP,IFLAG,SBR,K,CHECK,TEST,CELLS,CLR,CLC,
     *            SYMPTS,SMR,SMC,NAMES,NML)

      INTEGER CLR,CLC, SMR, SMC, NML
      INTEGER CELLS ( CLR, CLC ), SYMPTS ( SMR, SMC )
      CHARACTER NAMES ( CLR ) * ( 12 )
      LOGICAL FILE,COMPR,TEST,TEST1,CHECK,A
      CHARACTER C1*9,C2(5)*9,HELP(4)*80
      DATA(C2(I),I=1,5)/'ANALYSE','DRAW','TRANSLATE','EDIT','HELP'/
      SBR=2
      DO 22 J=1,5
      PRINT *,'INPUT KEYWORD OR STOP TO END RUN'
      READ(UNIT=5,FMT='(A)')C1
      IF(C1.EQ.'STOP') STOP
C
C****  SUBPROGRAM COMPR(C1,C2(I)) COMPARES INPUT KEYWORD WITH SPECIFIED
C****  KEYWORD
C
      DO 20 I=1,5
      A=COMPR(C1,C2(I))
 9000 IF ( .NOT. A ) GO TO 9010
C
C****   OPTION GIVEN AS TO WHETHER TO STORE DATA OBTAINED FROM SBR. EQTS
C****   INTO INTERMEDIATE FILE FOR LATER USE.
C

C
C*****   TEST FOR SELECTION OF PACKAGE ROUTINE
C

      IF(I.EQ.1)THEN
      TEST1=TEST.AND.CHECK
      IF(TEST1)THEN
      PRINT *,'-------------------------------------------------'
      PRINT *,'WARNING CAUSE & SYMPTOM EQNS HAVE ALREADY BEEN ANALYSED'
      PRINT *,'DATA HAVE BEEN STORED IN DATAFILE OPEREQNS'
      PRINT *,'-------------------------------------------------'
      IFLAG=5
      RETURN
      ELSE
C
C**** IF DATA HAVE ALREADY BEEN STORED IN DATAFILE WARNING IS GIVEN IF
C**** NEXT SELECTION IS TO ANALYSE CAUSE & SYMPTOM EQNS ELSE OPTION IS
C**** GIVEN FOR THE STORAGE OF THE DATA FROM THE ANALYSIS OF CAUSE &
C**** SYMPTOM EQNS INTO A DATAFILE.
```

```
      CHECK=FILE(I)
      CALL EQTS(IFLAG,CHECK,TEST,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
      ENDIF
      ELSEIF(I.EQ.2)THEN
      CALL TREES1(IFLAG,TEST)
      ELSEIF(I.EQ.3)THEN
      CALL WORDS(IFLAG,TEST)
      ELSEIF(I.EQ.4)THEN
      CALL EDIT(IFLAG,TEST)
      ELSEIF(I.EQ.5)THEN
      CALL DEXPLN
      IFLAG=5
      ENDIF
      RETURN
      GO TO 9000
 9010 CONTINUE
  20  CONTINUE
      CALL EXPLN(HELP)
  22  CONTINUE
      END



C****   SUBROUTINE GRFTR1 HAS BEEN PREPARED FOR GRAPHICAL TERMINALS
C****   WITHOUT LIGHT PEN FACILITIES ,IN WHICH CASE THE USER IS PRESENTED
C****   WITH A MENU LIST OF THE PACKAGE ROUTINES.-SELECTION IS MADE BY
C****   MOVING THE CURSOR TO THE ITEM TO BE SELECTED AND PRESSING THE
C****   RETURN BUTTON.THIS SUBROUTINE HAS MORE COMMON APPLICATION SINCE
C****   MOST GRAPHICAL TERMINALS HAVE GOT CURSOR FACILITIES.
C
      SUBROUTINE GRFTR1(IFLAG,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)

      INTEGER CLR,CLC,SMR,SMC,NML, ERRNO, VALUE
      INTEGER CELLS ( CLR, CLC ), SYMPTS ( SMR, SMC )
      INTEGER ICOM
      REAL X, Y
      CHARACTER NAMES ( CLR ) * ( 12 )

      CHARACTER CHS(5)*15
      LOGICAL TEST,CHECK,TEST1,FILE
      DATA (CHS(I),I=1,5)/'  A. ANALYSE','  B. DRAW','  C. TRANSLATE','
     1 D. EDIT','  E. HELP'/
      DATA TEST, CHECK / 2 * .FALSE. /


      CALL DRIVER ( 2, VALUE, ERRNO )
      CALL UNITS(5.)
      CALL SHIFT2(10.,20.)
      CALL MOVTO2(0.,0.)
      CALL MOVTO2(0.,5.)
      CALL PICCLE
      CALL MOVTO2(0.,5.)
```

```
      CALL CHAHOL('SELECT YOUR PACKAGE ROUTINE BY MOVING THE CURSOR')
      CALL MOVTO2(0.,4.)
      CALL CHAHOL(' TO THE ITEM OF THE MENU LIST AND PRESS ANY')
      CALL MOVTO2(0.,3.)
      CALL CHAHOL(' CHARACTER KEY')
      CALL MOVTO2(0.,0.)
      CALL CHAHOL('MENU LIST OF PACKAGE ROUTINES')

C
C****       SET UP MENU LIST
C

      DO 10 I=2,10,2
      X=I
      CALL MOVTO2(0.,-X)
      KK=I/2
      CALL AKCHAR(CHS(KK))
   10 CONTINUE
C
C****     MAKE THE CURSOR APPEAR
C
      CALL CURSOR(ICOM,X,Y)
C
C****    MOVE THE CURSOR TO THE ITEM OF THE MENU LIST TO BE SELECTED
C****    AND PRESS RETURN
C

      CALL CHAMOD
C
C****        CLEAR THE GRAPHICAL SCREEN
C

      CALL CHAHOL ( ' ')
      CALL PICCLE
C
C****     IDENTIFY ITEM SELECTED BY ANALYSING PICTURE SEGMENT
C****     COORDINATE  Y.
C

      IF(Y.GT.18.0.AND.Y.LT.19.0)THEN
      TEST1=TEST.AND.CHECK
      IF(TEST1)THEN
      PRINT *,'-----------------------------------------------'
      PRINT *
      PRINT *,'WARNING CAUSE & SYMPTOM EQNS HAVE ALREADY BEEN ANALYSED'
      PRINT *,'DATA FROM ANALYSIS OF CAUSE & SYMPTOM EQNS HAVE BEEN '
      PRINT *,'STORED IN DATAFILE OPXXXXXX. PRESS RETURN TO CONTINUE'
      PRINT *
      PRINT *,'-----------------------------------------------'

*     READ *

      IFLAG=5
      RETURN
      ELSE IF ( TEST ) THEN
```

```
      PRINT *,'-------------------------------------------------'
      PRINT *
      PRINT *,'THE ANALYSED CAUSE AND SYMPTOM EQUATIONS HAVE ALREADY'
      PRINT *,'BEEN READ IN FROM DATAFILE OPXXXXXX'
      PRINT *
      PRINT *,'-------------------------------------------------'

     ELSE
     CHECK=FILE(I)
     CALL EQTS(IFLAG,CHECK,TEST,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
     ENDIF
     ELSEIF(Y.GT.16.0.AND.Y.LT.17.0)THEN
     CALL TREES(IFLAG,TEST,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
     ELSEIF(Y.GT.14.0.AND.Y.LT.15.0)THEN
     CALL WORDS(IFLAG,TEST)
     ELSEIF(Y.GT.12.0.AND.Y.LT.13.0)THEN
     CALL EDIT(IFLAG,TEST)
     ELSEIF(Y.GT.10.0.AND.Y.LT.11.0)THEN
     CALL CCLOSE
     CALL DEXPLN
     IFLAG=5
     ENDIF
     CALL DEVEND
     END



C****    SUBROUTINE EQTS  IS USED TO SIMULATE THE ANALYSIS OF
C****    CAUSE & SYMPTOM EQNS AND MAY EASILY FACILITATE THE
C****    PROGRAM TREES WHICH ANALYSES CAUSE & SYMPTOM EQNS.
C

     SUBROUTINE EQTS(IFLAG,CHECK,TEST,CELLS,CLR,CLC,SYMPTS,SMR,SMC,
    *           NAMES,NML)

     INTEGER CLR,CLC,SMR,SMC,NML
     LOGICAL TEST,CHECK
     INTEGER CELLS ( CLR, CLC ), SYMPTS ( SMR, SMC )
     CHARACTER NAMES ( CLR ) * ( 12 )

     CALL PART1(CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
     TEST = .TRUE.

C
C****    CHECK IF THE USER HAS REQUESTED TO STORE DATA FROM ANALYSIS
C****    OF CAUSE & SYMPTOM EQNS INTO A FILE.
C
     IF(CHECK)THEN
     OPEN(3,FILE='OPXXXXXX',FORM='UNFORMATTED',
    * ACCESS = 'SEQUENTIAL')
     REWIND 3
     DO 11 I = 1, CLR
     WRITE(3)(CELLS(I,J),J=1,CLC)
```

```
 11  CONTINUE
     DO 12 K = 1,SMR
     WRITE(3)(SYMPTS(K,L),L=1,SMC)
 12  CONTINUE
     DO 13 M = 1,CLR
     WRITE(3)NAMES(M)
 13  CONTINUE
     CLOSE(3)
     ENDIF
     IFLAG=1
     END


     SUBROUTINE TREES1(IFLAG,TEST)
     LOGICAL TEST
     CHARACTER DEQTS*90
     IF(TEST)THEN
     OPEN(3,FILE='OPXXXXXX',FORM='UNFORMATTED',
    * ACCESS='SEQUENTIAL')
     REWIND 3
     READ(3)DEQTS
     WRITE(3)DEQTS
     CLOSE(3)
     ELSE
C****      INSERT PROGRAM TREES1 FOR NON-GRAPHICAL FAULT TREES.
C
     IFLAG=2
     ENDIF
     END

C
C****    SUBROUTINE WORDS SIMULATES THE TRANSLATION OF CAUSE & SYMPTOM
C****    EQNS INTO WORDS AND USES AS ITS DATA STRUCTURE THE ANALYSIS
C****    OF CAUSE AND SYMPTOM EQNS . THE EXISTING PROGRAM WORDS MAY
C****    BE INSERTED WITHIN THIS SUBROUTINE .
C
     SUBROUTINE WORDS(IFLAG,TEST)
     LOGICAL TEST
     CHARACTER DEQTS*90
C
C****    TEST WHETHER DATA STRUCTURE FOR THE SUBROUTINE IS TO BE READ
C****    FROM FILE CONTAINING THE ANALYSIS OF CAUSE & SYMPTOM EQNS
C****    OR IS TO BE TRANSFERED VIA SUBROUTINE ARGUMENTS.
C
     IF(TEST)THEN
     OPEN(3,FILE='OPXXXXXX',FORM='UNFORMATTED',
    * ACCESS='SEQUENTIAL')
     REWIND(3)
     READ(3)DEQTS
     CLOSE(3)
     ENDIF
C
C****          INSERT PROGRAM WORDS
C
     IFLAG=3
```

```
      END

C
C****   SUBROUTINE EDIT SIMULATES THE EDITING OF CAUSE & SYMPTOM
C****   EQNS AND ONCE MORE THE DATA STRUCTURE FOR THIS SBR. COMES
C****   FROM THE ANALYSIS OF CAUSE & SYMPTOM EQNS. THE SBR. MAY
C****   FACILITATE THE NEWLY PREPARED  PROGRAM FOR EDITING.
C

      SUBROUTINE EDIT(IFLAG,TEST)
      LOGICAL TEST
      CHARACTER DEQTS*90
C
C****     TEST WHETHER DATA OF ANALYSIS OF C&S EQNS IS TO BE READ FROM
C****     A FILE OR TRANSFERED AS AN ARGUMENT OF THE SBR.
C
      IF(TEST)THEN
      OPEN(3,FILE='OPXXXXXX',FORM='UNFORMATTED',
     *  ACCESS='SEQUENTIAL')
      REWIND(3)
      READ(3)DEQTS
      CLOSE(3)
      ENDIF
      IFLAG=4
      END

C
C****   SUBROUTINE DEXPLN OUTPUTS A DETAILED ILLUSTRATED EXPLANATION
C****   OF THE PACKAGE
C
      SUBROUTINE DEXPLN
      CHARACTER LINE*80,REP*4

*     OPEN ( UNIT = 4, FILE = 'ZXC' )
      DO 15 J=1,4
C
C****   OUTPUT EXPLANATION OF PACKAGE AT INTERVALS OF 50 LINES
C
      DO 20 I=1,60
C
C****   READ THE TEXT OF THE EXPLANATION OF THE PACKAGE FROM FILE ZXC
C
*     READ(4,100)LINE
*  100 FORMAT(A)
      READ(9,'(A)')LINE
      IF(LINE.EQ.'@@@')THEN

*       CLOSE ( UNIT = 4 )
*       RETURN
      GO TO 15
      ELSE
C
C****     OUTPUT THE TEXT OF EXPLANATION FROM FILE ZXC TO THE SCREEN.
C
      WRITE(UNIT=*,FMT='(A)')LINE
```

```
      ENDIF
   20 CONTINUE
      WRITE(16,FMT=*)'PRESS RETURN TO CONTINUE'
      READ(UNIT=*,FMT='(A)')REP
   15 CONTINUE
       RETURN
      END



C
C****    SUBROUTINE TREES SIMULATES THE DRAWING OF FAULT TREES
C****    FROM CAUSE & SYMPTOM EQNS ON A GRAPHICAL TERMINAL
C****    AND CAN EASILY FACILITATE THE PROGRAM TREES FOR THE
C****    ACTUAL DRAWING OF FAULT TREES.
C
      SUBROUTINE TREES(IFLAG,TEST,CELLS,CLR,CLC,SYMPTS,SMR,SMC,
     *            NAMES,NML)

      INTEGER CLR,CLC,SMR,SMC,NML,IFLAG
      INTEGER CELLS ( CLR, CLC ), SYMPTS ( SMR, SMC )
      CHARACTER NAMES ( CLR ) * ( 12 )
      LOGICAL TEST
C
C****    TEST WHETHER DATA FROM ANALYSIS OF CAUSE & SYMPTOM EQNS
C****    ARE GOING TO BE READ FROM A FILE OR TO BE TRANSFERED AS
C****    ARGUMENTS FROM THE SUBROUTINE.
C
       IF(.NOT. TEST)THEN
      OPEN(3,FILE='OPXXXXXX',FORM = 'UNFORMATTED',
     * ACCESS = 'SEQUENTIAL')
      REWIND(3)
      DO 11 I = 1, CLR
      READ(3)(CELLS(I,J),J=1,CLC)
   11  CONTINUE
      DO 12 K = 1,SMR
      READ(3)(SYMPTS(K,L),L=1,SMC)
   12  CONTINUE
      DO 13 M = 1, CLR
      READ(3) NAMES(M)
   13  CONTINUE
      CLOSE(3)
      ENDIF
C
      CLOSE(18)
      CALL PART2(CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
C
      IFLAG=2
      END



C
```

```
C****   SUBROUTINE GRFTRM HAS BEEN SPECIALLY PREPARED FOR TERMINALS
C****   WITH GRAPHICAL FAILITIES.- THE USER IS PRESENTED WITH A LIST
C****   OF PACKAGE ROUTINES IN THE FORM OF A MENU .- SELECTION IS
C****   MADE BY POINTING A LIGHT PEN AT THE ITEM OF YOUR CHOICE.
C****   THIS SUBROUTINE COMPRICES OF SPECIAL GINO GRAPHICS COMMANDS
C****   WHICH DO NOT COMPLY WITH MOST GRAPHICAL TERNMINALS WITHOUT
C****   LIGHT PEN FACILITIES.
C
      SUBROUTINE GRFTRM(IFLAG,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
      INTEGER CLR,CLC,SMR,SMC,NML, ERRNO, VALUE
      INTEGER CELLS ( CLR, CLC ), SYMPTS ( SMR, SMC )
      CHARACTER NAMES ( CLR ) * ( 12 )
      CHARACTER CHS(5)*15
      LOGICAL CHECK,TEST,TEST1,FILE
      COMMON/GFEVEN/KEY,IMPKEY,IMPDAT,NSEG,XPIC,YPIC,NARGS,ARGS(80)
      DATA (CHS(I),I=1,5)/' A. ANALYSE',' B. DRAW',' C. TRANSLATE','
     1 D. EDIT',' E. HELP'/
      CALL DRIVER ( 2, VALUE,ERRNO )
      CALL UNITS(5.)
      CALL SHIFT2(2.,15.)
      CALL MOVTO2(0.,0.)

C
C****         SET UP MENU LIST
C
C****         SET UP PICTURE SEGMENTS
C
      DO 10 I=1,5
      CALL  PICBEG(I)
      X=I
      CALL MOVTO2(0.,-X)
      CALL AKCHAR(CHS(I))
   10 CONTINUE
C
C****       CLOSE PICTURE SEGMENTS
C
      CALL PICEND
C
C****          PREPARE FOR PENHIT
C
      CALL EVESET(2)
      DO 11 J=1,5

C
C****      MAKE PICTURE SEGMENT SENSITIVE TO PENHIT
C

      CALL  PICSEN(J,1)
   11 CONTINUE
      J1=2

C
C****       CALL TO EVENT EXPECTS EVENT OF TYPE PENHIT
C
```

```
      CALL EVENT(J1)

C
C****      ELIMINATE PICTURE SEGMENT SENSITIVITY TO PENHIT
C

      DO 12 N=1,5
      CALL PICSEN(N,0)
   12 CONTINUE

C
C****          END OF EVENT
C

      CALL  EVEDEL(2)

C
C****          ENABLE TRANSFER OF PICTURE SEGMENT NUMBER NSEG FROM
C****          GRAPHICAL SCREEN TO ALPHANUMERIC SCREEN.
C

      CALL CHAMOD

C
C****          IDENTIFY ITEM SELECTED
C

      IF(NSEG.EQ.1)THEN
      TEST1=TEST.AND.CHECK
      IF(TEST1)THEN
      PRINT *,'-----------------------------------------------------'
      PRINT *
      PRINT *,' WARNING CAUSE & SYMPTOM EQNS HAVE ALREADY BEEN ANALYSED'
      PRINT *,' DATA HAVE BEEN STORED IN DATA FILE OPEREQNS'
      PRINT *
      PRINT *,'-----------------------------------------------------'
      IFLAG=5
      RETURN
      ELSE
      CHECK=FILE(I)
      CALL EQTS(IFLAG,CHECK,TEST,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
      ENDIF
      ELSEIF(NSEG.EQ.2)THEN
      CALL TREES(IFLAG,TEST,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
      ELSEIF(NSEG.EQ.3)THEN
      CALL WORDS(IFLAG,TEST)
      ELSEIF(NSEG.EQ.4)THEN
      CALL EDIT(IFLAG,TEST)
      ELSEIF(NSEG.EQ.5)THEN
      CALL DEXPLN
      IFLAG=5
      ENDIF
      CALL DEVEND
      END
```

```
C
C****    SUBROUTINE AKCHAR PREPARES CHARACTER STRINGS TO BE READILY
C****    USED WITH THE GINO COMMAND CALL CHAARR(L,M,N) WHICH OUTPUTS
C****    A CHARACTER STRING.
C
      SUBROUTINE AKCHAR(STRING)
      PARAMETER(LENGTH=20)
      CHARACTER STRING*(*)
      INTEGER FINAL
      NCHAR=FINAL(STRING)
C
C****    OUTPUT STRING OF CHARACTERS
C

      CALL CHAHOL(STRING(1:FINAL(STRING)))
      END
      LOGICAL FUNCTION YESNO(TYPE)
      CHARACTER ANSWER*3
      LOGICAL REPLY
 9000 CONTINUE
      READ (*,16)ANSWER
  16  FORMAT(A)
      IF(ANSWER.EQ.'YES'.OR.ANSWER.EQ.'YE'.OR.ANSWER.EQ.'Y'.OR.ANSWER.
     +EQ.'yes'.OR.ANSWER.EQ.'ye'.OR.ANSWER.EQ.'y')THEN
      REPLY=.TRUE.
      YESNO=.TRUE.
      ELSEIF(ANSWER.EQ.'NO'.OR.ANSWER.EQ.'N'.OR.ANSWER.EQ.'NO'.OR.
     +ANSWER.EQ.'N')THEN
      REPLY=.TRUE.
      YESNO=.FALSE.
      ELSE
      PRINT *,'YOUR ANSWER SHOULD BE YES OR NO'
      REPLY=.FALSE.
      ENDIF
      IF ( .NOT. REPLY) GO TO 9000
      RETURN
      END
      SUBROUTINE EXPLN(HELP)
      CHARACTER HELP(4)*80
      LOGICAL RQST,YESNO
      PRINT *,'-----------------------------------------------------'
      PRINT *,'PLEASE TRY AGAIN KEYWORD WRONGLY INPUTED'
      PRINT *,'DO YOU REQUIRE EXPLANATION OF KEYWORDS?ENTER YES OR NO'
      PRINT *
      PRINT *,'-----------------------------------------------------'
      RQST=YESNO(TYPE)
      IF(RQST)THEN
      PRINT 20,(HELP(I),I=1,4)
  20  FORMAT(A/)
      PRINT *,'-----------------------------------------------'
      PRINT *
      PRINT *,'IF DETAILED EXPLANATION OF PACKAGE IS REQUIRED PRINT'
      PRINT *,'HELP- THIS PRODUCES LISTING TO THE SCREEN '
```

```
      PRINT *
      PRINT *,'-------------------------------------------------'
      ELSE
      PRINT *,'-------------------------------------------------'
      ENDIF
      END


      LOGICAL FUNCTION COMPR(C1,C2)
      CHARACTER C1*(*),C2*(*)
      INTEGER FIRST,FINAL
      IF(C1(FIRST(C1):FINAL(C1)).EQ.C2(FIRST(C2):FINAL(C2)))THEN
      COMPR=.TRUE.
      ELSE
      COMPR=.FALSE.
      ENDIF
      RETURN
      END


      INTEGER FUNCTION FIRST(C)
      CHARACTER C*(*)
      INTEGER COUNT
      DO 10 COUNT=1,LEN(C)
      IF(C(COUNT:COUNT).NE.' ')THEN
      FIRST=COUNT
      RETURN
      ENDIF
   10 CONTINUE
      FIRST=LEN(C)
      RETURN
      END


      INTEGER FUNCTION FINAL(C)
      CHARACTER C*(*)
      INTEGER COUNT
      DO 15 COUNT=LEN(C),1,-1
      IF(C(COUNT:COUNT).NE.' ')THEN
      FINAL=COUNT
      RETURN
      ENDIF
   15 CONTINUE
      FINAL=0
      RETURN
      END


      LOGICAL FUNCTION FILE(I)
      LOGICAL RPLY,YESNO
      PRINT *,'-------------------------------------------------'
      PRINT *,'DO YOU INTEND TO TERMINATE RUN BEFORE OBTAINING FINAL RES
     1ULTS?'
```

```
      PRINT *,'IF YES INTERMEDIATE RESULTS FROM ANALYSIS OF CAUSE AND SY
     1MPTOM EQNS '
      PRINT *,'MAY BE STORED IN DATA FILE AND USED ON REQUEST'
      PRINT *
      PRINT *,'-------------------------------------------------'
      RPLY=YESNO(TYPE)
      IF(RPLY)THEN
      FILE=.TRUE.
      ELSE
      FILE=.FALSE.
      ENDIF
      RETURN
      END



C
C*******
C
      SUBROUTINE PART1(CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)

      INTEGER CLR,CLC, SMR, SMC, NML
      INTEGER AT,OP,PO,EQ,AX,TP,NAM,SYMP,SIGN,I,Q,N,POS,M,K,P,S,R,J
      INTEGER LIMT,FIM,TNML
      PARAMETER (AT=30,OP=20,PO=30,EQ=200,AX=30,TP=9,TNML=12)
      INTEGER CELLS ( CLR, CLC ), SYMPTS ( SMR, SMC )
      INTEGER DIST,TEST,ATRNS(AT),OPSTK(OP),POLLSH(PO),AUX(AX),TEMP(TP)
      CHARACTER NAMES(CLR)*(12),BRANCH*(TNML)
      CHARACTER EQN*(EQ)
*
*     'INITIALISING ARRAYS'
*
      DO 10 I=1,CLR
       NAMES(I)='EMPTY'
   10 CONTINUE


      DO 20 I=1,SMR
       DO 21 J=1,SMC
        SYMPTS(I,J)=0
   21  CONTINUE
   20  CONTINUE

      DO 22 I=1,CLR
       DO 23 J=1,CLC
        CELLS(I,J)=0
   23  CONTINUE
   22  CONTINUE

      WRITE ( UNIT = *, FMT = 24 )
   24  FORMAT (T30,'D A T A'/T30,7('*'))
      WRITE ( UNIT = *, FMT =*)
      OPEN(8,FILE='SOLVAY',FORM='FORMATTED')
      REWIND(8)
  100 READ( UNIT = 8, FMT = * ) EQN
```

```
      WRITE ( UNIT = *, FMT = '(2A)' ) 'EQN  :  ',EQN

      Q=0
      N=0
      IF(EQN.EQ.'END')THEN
       GO TO 2000
      END IF

      CALL DISTNT (EQN,DIST,Q )
      POS=Q-1
      BRANCH=EQN(1:POS)
      CALL ALLOC(BRANCH,R,TEST,NAMES,NML,CLR)
      NAM=R
      IF(DIST.EQ.0)THEN
       GO TO 350
      END IF
      IF(TEST.EQ.-1)THEN
       CELLS(NAM,1)=1
       CELLS(NAM,2)=-2
      END IF
      M=0
   50 M=M+1
      IF(M.GT.SMR)THEN
       PRINT*, 'SYSTEM CANNOT ACCEPT MORE EQS.'
       GO TO 300
      END IF
      IF(SYMPTS(M,1).NE.0)THEN
       GO TO 50
      END IF
      SYMPTS(M,1)=NAM
      SYMP=M
      K=1
   60 P=Q+1
      S=0
      SIGN=0
   70 Q=Q+1
      S=S+1
      IF(S.GT.NML+1)THEN
       PRINT*, 'THERE IS A MISTAKE,IT IS NOT A SYMPTON EQN.'
       PRINT*, 'INPUT EQUATION WAS'
       PRINT*, EQN
       GO TO 300
      END IF
      IF(EQN(Q:Q).EQ.'*')THEN
       BRANCH=EQN(P:Q-1)
      ELSE
       IF(EQN(Q+1:Q+1).EQ.' ')THEN
        BRANCH=EQN(P:Q)
        SIGN=-1
       ELSE
        GO TO 70
       END IF
      END IF
      CALL ALLOC(BRANCH,R,TEST,NAMES,NML,CLR)
      IF(TEST.EQ.-1)THEN
```

```
      CELLS(R,1)=1
      CELLS(R,2)=0
      CELLS(R,3)=1
     ELSE
      CELLS(R,3)=CELLS(R,3)+1
     END IF
     J=CELLS(R,3)
     CELLS(R,3+J)=SYMP
     SYMPTS(SYMP,2+K)=R
     K=K+1
     IF(K.GT.SMC-2)THEN
      PRINT*, 'THERE ARE MORE THAN 18 BRANCHS IN A SYMPTON EQN.'
      GO TO 300
     END IF
     IF(SIGN.EQ.0)THEN
      GO TO 60
     END IF
     SYMPTS(SYMP,2)=K-1
     GO TO 100

350  IF(TEST.EQ.-1)THEN
      CELLS(NAM,1)=1
     END IF


     CALL TRNSLT(EQN,Q,ATRNS,I,LIMT,EQ,AT,CELLS,CLR,CLC,NAMES,NML)


     CALL POLISH(ATRNS,I,POLLSH,N,LIMT,FIM,OP,PO,AT,OPSTK,AUX)


     CALL CAUSE(POLLSH,NAM,FIM,PO,TP,CELLS,CLR,CLC,NAMES,NML,TEMP)


     GO TO 100


2000 DO 1500 J=1,CLR
      IF(CELLS(J,3).EQ.0)THEN
       IF(NAMES(J).NE.'EMPTY')THEN
        CELLS(J,2)=-2
        CELLS(J,1)=1
       ELSE
        GO TO 200
       END IF
      END IF
1500 CONTINUE


 200 WRITE ( UNIT = *, FMT =*)
     WRITE ( UNIT = *, FMT = 205 )
 205 FORMAT (T42,'C E L L S'/T42,9('*'))
     WRITE ( UNIT = *, FMT = * )
     DO 210 I=1,CLR
      WRITE ( UNIT = *, FMT = 220 ) I, (CELLS ( I,J ), J = 1, CLC )
```

```
220  FORMAT (T5,I3,T20,10(3X,I3))
210 CONTINUE


    WRITE ( UNIT = *, FMT = * )
    WRITE ( UNIT = *, FMT = 215 )
215 FORMAT (T60,'S Y M P T S'/T60,11('*'))
    WRITE ( UNIT = *, FMT = * )
    DO 230 I=1,SMR
     WRITE ( UNIT = *, FMT = 240 )I,(SYMPTS(I,J),J=1,SMC)
240  FORMAT (T5,I3,T20,(10(2X,I3)))
230 CONTINUE


    WRITE ( UNIT = *, FMT = * )
    WRITE ( UNIT = *, FMT = 265 )
265 FORMAT (T10,'N A M E S'/T10,9('*'))
    WRITE ( UNIT = *, FMT = * )
    DO 250 I=1,CLR
     WRITE ( UNIT = *, FMT = 260 ) I,NAMES(I)
260  FORMAT (T5,I3,T10,A)
250  CONTINUE
300 RETURN
    END


    SUBROUTINE DISTNT(EQN,DIST,Q )

C*** *     THIS SUBROUTINE DETERMINES WHETHER AN EQUATION IS A CAUSE OR
C*** *      A SYMPTOM EQUATION.
C*** *
C*** *      EQN   - CHARACTER - THE EQUATION IN QUESTION
C*** *      DIST  - INTEGER   - RETURNS -1 FOR SYMPTOM
C*** *                          RETURNS 0 FOR CAUSE
C*** *      Q     - INTEGER   - RETURNS POSITION OF - OR = IN EQN
C*** *

    CHARACTER EQN * ( * )
    INTEGER DIST, Q, NXTNSP

    Q = INDEX ( EQN, ')' )

    IF ( Q .EQ. 0 ) THEN

      CALL INERR ( EQN )
      STOP

    ELSE

*
*     LOOK FOR NEXT NON-SPACE CHARACTER
*

      Q = Q + NXTNSP ( EQN ( Q + 1 : LEN ( EQN ) ) )
```

```
      IF ( EQN ( Q : Q ) .EQ. '-' ) THEN

        DIST = -1
        RETURN

      ELSE IF ( EQN ( Q : Q ) .EQ. '=' ) THEN

        DIST = 0
        RETURN

      ELSE

        CALL INERR ( EQN )
        STOP

      END IF

    END IF

    END


    SUBROUTINE ALLOC(BRANCH,R,TEST,NAMES,NML,CLR)

    INTEGER CLR,NML
    INTEGER R, TEST, NEXT
    PARAMETER ( TNML = 12 )
    CHARACTER BRANCH*(*),NAMES(CLR)*(12)

    DATA NEXT / 1 /

    IF ( BRANCH .NE. ' ' ) THEN

      DO 10, R = 1 , NEXT

        IF ( BRANCH .EQ. NAMES ( R ) ) THEN

          TEST = 0
          RETURN

        END IF

10      CONTINUE

    END IF

    IF ( NEXT .EQ. CLR ) THEN

      PRINT *, 'SYSTEM CANNOT ACCEPT MORE EQUATIONS '
      STOP

    ELSE

      NAMES ( NEXT ) = BRANCH
```

```
      R = NEXT
      NEXT = NEXT + 1
      TEST = -1

      RETURN

    END IF

    END
*************************************************************************
***** SUBROUTINE PART2
*****   TO INTERPRETE A DATA STRUCTURE, WHICH REPRESENTS
*****   CAUSE AND SYMPTOM EQUATIONS AS CODED BY
*****   DR. D. A. LIHOU FROM OPERABILITY STUDY RECORDS,
*****   AND TO DRAW A REPRESENTATION OF THE ABOVE
*****   EQUATIONS IN THE FORM OF A FAULT TREE USING
*****   GINO LIBRARY
*************************************************************************

    SUBROUTINE PART2 (CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)

    INTEGER CLR,CLC,SMR,SMC,NML, ERRNO, NOTERM
    PARAMETER ( IW = 10, IL = 40 )
    PARAMETER ( YMAX=350.0, XMAX=1030.0 )
    COMMON /ONE/ FLAG
    COMMON /TWO/ JLJ, KLJ, KR, IOPT
    COMMON/SHIN/SHX,SHY,SCX,SCY
    INTEGER  CELLS(CLR,CLC), IOPT(3), SYMPTS ( SMR, SMC )
    CHARACTER NAMES(CLR)*(12), INDATA*40, FLAG(IW,IL)*40
    REAL X,Y
    INTEGER ICOM
    LOGICAL CLEAR,EXIST,ERROR

    CLEAR = .FALSE.
C**
 1   CONTINUE
    JLJ = 1
    KR = -1
C**** INITIALISE THE ARRAY FLAG
    DO 10 JK = 1,IL
      DO 10 JR = 1,IW
  10 FLAG(JR,JK) = 'EMPTY'
     CALL BOXSTR(1,'    ',0.,0.,0.,0.,ERROR)
    IF ( CLEAR ) THEN
     GO TO 19
    END IF
C**** READ THE OPTIONS AND ACT APPROPRIATELY
    CALL DAWRI( 'EMPTY',1,CELLS,CLR,CLC,NAMES,NML)
C**** INITIALISE A GINO FILE
    CALL DRIVER ( 2, NOTERM, ERRNO )
C*****  SET UP THE SHIFT FACTORS ( SHX AND SHY ) AND THE SCALE FACTORS
C*****  (SCX AND SCY ) FOR THE GINO GRAPHICS
    CALL DRIVER ( 3, NOTERM, ERRNO )
    IF ( NOTERM .EQ. 1 ) THEN
     CALL HARCHA
```

```
      END IF
      IF ( NOTERM .EQ. 1 ) THEN

C****     I.E. A NEWBURY SERIES 8000
        SHX = -25
        SHY = 140
        SCX = 0.7
        SCY = 0.7
      ELSE
C****     I.E. A DYNAGRAPHICS
        SHX = 0
        SHY = 250
        SCX = 0.7
        SCY = 0.7
      END IF

      CALL SHIFT2 ( SHX,SHY )
      CALL SCALE2(SCX,SCY)
      CALL ROTAT2 ( 270. )
      CALL CHASWI ( 1 )
      CALL PENSEL( 1,0.0,0 )
 19   CLEAR = .FALSE.

      CALL WINDIN(1)
      CALL WINDIN(2)
      CALL WINDIN(3)
      CALL WINDIN(4)
      CALL WINDIN(5)
      CALL CHWIN(2)
C**** READ, WRITE OUT IF REQUIRED, AND CHECK THE INPUT DATA
      PRINT *,'IF THE SCREEN IS CLEAR THEN INPUT DATA FROM KEYS
     +ONLY'
      PRINT *,'  IF NOT THEN YOU CAN CHOOSE INPUT BY;'
      PRINT *,'  1. CURSOR'
      PRINT *,'  2. KEYS'
 20   READ(10, '( A )')INDATA
 99   CALL DAWRI( INDATA,2,CELLS,CLR,CLC,NAMES,NML )
      CALL DATA( INDATA,I,CELLS,CLR,CLC,NAMES,NML, CLEAR, EXIST )

      IF ( CLEAR ) THEN
      GO TO 1
      END IF
      IF ( .NOT. EXIST ) THEN
       GO TO 20
      END IF
      KR = KR+1
C**** ANALYSE THE DATA STRUCTURE AND STORE THE INFORMATION
C**** IN THE ARRAY FLAG
      KLJ = JLJ
      CALL ANCELL(KLJ,I,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
      KLJ = KLJ+1
C**** USING GINO DRAW THE FAULT TREE CREATED IN ARRAY FLAG
       CALL CHWIN(1)
      CALL DRAWTR(I,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
C**** UPDATE THE COUNTER JLJ AND READ THE NEXT ITEM OF DATA
```

```
      JLJ = KLJ+1
       CALL CHWIN(4)
       PRINT *
       PRINT *,'KEYS'
       PRINT *
       CALL CHWIN(2)
       PRINT *
       PRINT *,'**NOW YOU MUST USE CURSOR FOR CHOOSING INPUT**'
       PRINT *
   88  CALL CURSOR(ICOM,X,Y)
      IF(X.GE.182.AND.X.LE.210.AND.Y.GE.18.AND.Y.LE.36)THEN
        GO TO 1
      ELSEIF(X.GE.182.AND.X.LE.210.AND.Y.GE.0.AND.Y.LT.15)THEN
       CALLCHWIN(4)
       PRINT *
       PRINT *,'   *****'
       PRINT *,'**KEYS**'
       PRINT *,'   *****'
       CALL CHWIN(2)
      GO TO 20
      ELSEIF(X.GE.0.AND.X.LE.210.AND.Y.GE.40.AND.Y.LE.120)THEN
      CALL CHWIN(2)
      PRINT *,'*********************************************'
      PRINT *,'***NOW CLEAR THE SCREEN FOR MORE INPUT*'
      PRINT *,'*********************************************'

      GO TO 20
      ELSEIF(X.GE.154.AND.X.LT.182.AND.Y.GE.18.AND.Y.LE.36)THEN
      CALL CHWIN(2)
       PRINT *,'*** END OF DATA88'
       STOP
      ELSE
       CALL BOXSTR(3,INDATA,X,Y,0.,0.,ERROR)
       CALL CHWIN(2)
       PRINT *, INDATA
      IF(ERROR)THEN
      PRINT *,'*******************************'
      PRINT *,'********* TRY AGAIN ***********'
      PRINT *,'*******************************'
       ENDIF
      GO TO 99
      END IF
      END



       SUBROUTINE DATA( INDATA,I,CELLS,CLR,CLC,NAMES,NML, CLEAR, EXIST )

      INTEGER CLR,CLC,NML,I
      CHARACTER NAMES(CLR)*(12), INDATA*40
      INTEGER CELLS(CLR,CLC)
      LOGICAL CLEAR, EXIST

      IF (INDATA .EQ. 'STOP'.OR.INDATA .EQ. 'STO'.OR.INDATA.EQ.'ST'
     +.OR.INDATA.EQ.'S'.OR.INDATA.EQ.'STOP'.OR.INDATA.EQ.'STO'.OR.
```

```
     +INDATA.EQ.'ST'.OR.INDATA.EQ.'S') THEN
*       PRINT *
*       PRINT *,'*****  END OF INPUT DATA  *****'
        CALL GINEND
        CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
        STOP

     ELSE IF ( INDATA .EQ. 'CLEAR' .OR. INDATA .EQ. 'CLEA'.OR.INDATA
     +.EQ.'CLE'.OR.INDATA .EQ. 'CL'.OR.INDATA.EQ.'C'.OR.INDATA.EQ.
     +'CLEAR'.OR.INDATA.EQ.'CLEA'.OR.INDATA.EQ.'CLE'.OR.INDATA.EQ.
     +'CL'.OR.INDATA.EQ.'C') THEN

       CALL PICCLE
       CLEAR = .TRUE.
       RETURN
     END IF
     DO 10 I = 1,CLR
       IF (INDATA .EQ. NAMES(I)) THEN

         EXIST = .TRUE.
         RETURN
       END IF
   10 CONTINUE
       PRINT *
       PRINT *,'****  DATA  GIVEN  NOT  CORRECT'
       PRINT *,'****  THIS DEVIANT STATE WAS NOT CONTAINED IN THE'
       PRINT *,'****  INPUT EQUATIONS  -  ENTER CLEAR TO CLEAR SCREEN'

     EXIST = .FALSE.

     RETURN
C
     END
C
C****
C
     SUBROUTINE TRNSLT(EQN,Q,ATRNS,I,LIMT,EQ,AT,CELLS,CLR,CLC,
     *               NAMES,NML)

     INTEGER EQ,AT,CLR,CLC,NML
     INTEGER Q,ATRNS(AT),I,LIMT,PS,MEM,R,TEST,CELLS(CLR,CLC)
     CHARACTER BRANCH*(12),EQN*(*),NAMES(CLR)*(12)
     DO 51 I=1,AT
       ATRNS(I)=0
   51 CONTINUE
     I=1
     PS=Q+1
   55 Q=Q+1
     IF(EQN(Q:Q).EQ.'(')THEN
       IF(EQN(Q+1:Q+1).NE.'(')THEN
         IF(Q.EQ.PS)THEN
           ATRNS(I)=-4
           I=I+1
           PS=PS+1
           GO TO 55
```

```
        ELSE
         MEM=INDEX(EQN(PS:  ),')')
         BRANCH=EQN(PS:PS+MEM-1)
          CALL ALLOC(BRANCH,R,TEST,NAMES,NML,CLR)
         IF(TEST.EQ.-1)THEN
          CELLS(R,1)=1
         END IF
         ATRNS(I)=R
         I=I+1
         PS=PS+MEM
         Q=PS-1
         GO TO 55
        END IF
       ELSE
        ATRNS(I)=-4
        I=I+1
        PS=PS+1
        GO TO 55
       END IF
      END IF
      IF(EQN(Q:Q).EQ.')')THEN
       ATRNS(I)=-3
       I=I+1
       PS=Q+1
       GO TO 55
      END IF
      IF(EQN(Q:Q).EQ.'*')THEN
       ATRNS(I)=-1
       I=I+1
       PS=Q+1
       GO TO 55
      END IF
      IF(EQN(Q:Q).EQ.'+')THEN
       ATRNS(I)=-2
       I=I+1
       PS=Q+1
       GO TO 55
      END IF
      IF(EQN(Q:Q).NE.' ')THEN
       GO TO 55
      ELSE
       LIMT=I-1
      END IF
      RETURN
      END

C
C*****
C
      SUBROUTINE POLISH(ATRNS,I,POLLSH,N,LIMT,FIM,OP,PO,AT,OPSTK,AUX)

      INTEGER OP,PO,AT,FIM,N,I,L,J,LIMT,X
      INTEGER POLLSH(PO),ATRNS(AT),OPSTK(OP),AUX(OP)

      I=0
```

```
      N=0
      J=0
      X=0
500   I=I+1
      IF(I.GT.LIMT)THEN
       IF(J.NE.0)THEN
        DO 555 L=J,1,-1
         N=N+1
         POLLSH(N)=OPSTK(L)
555     CONTINUE
       END IF
       FIM=N
       RETURN
      END IF
      IF(ATRNS(I).LT.0)THEN
       IF(J.EQ.0)THEN
        J=J+1
        OPSTK(J)=ATRNS(I)
        GO TO 500
       END IF
       X=X+1
       AUX(X)=ATRNS(I)
       IF(AUX(X).GT.OPSTK(J))THEN
        J=J+1
        OPSTK(J)=AUX(X)
        AUX(X)=0
        X=X-1
        GO TO 560
       ELSE
        IF(AUX(X).NE.OPSTK(J))THEN
         IF(AUX(X).EQ.-4)THEN
          J=J+1
          OPSTK(J)=AUX(X)
          AUX(X)=0
          X=X-1
          GO TO 500
         END IF
         IF(AUX(X).EQ.-3)THEN
570        N=N+1
          POLLSH(N)=OPSTK(J)
          OPSTK(J)=0
          J=J-1
          IF(OPSTK(J).NE.-4)THEN
           GO TO 570
          ELSE
           J=J+1
           OPSTK(J)=AUX(X)
           AUX(X)=0
           X=X-1
           GO TO 560
          END IF
         END IF
580       N=N+1
         POLLSH(N)=OPSTK(J)
         OPSTK(J)=0
```

```
        J=J-1
        IF(J.GT.0)THEN
         IF(OPSTK(J).EQ.POLLSH(N))THEN
          GO TO 580
         END IF
        END IF
        J=J+1
        OPSTK(J)=AUX(X)
        AUX(X)=0
        X=X-1
        GO TO 500
       ELSE
        IF(AUX(X).GT.-3)THEN
         DO 590 L=I+1,LIMT
          IF(ATRNS(L).LT.0)THEN
           IF(ATRNS(L).NE.-3)THEN
            IF(ATRNS(L).NE.AUX(X))THEN
             J=J+1
             OPSTK(J)=AUX(X)
             AUX(X)=0
             X=X-1
             GO TO 500
            END IF
           END IF
          END IF
590       CONTINUE
         N=N+1
         POLLSH(N)=AUX(X)
         AUX(X)=0
         X=X-1
         GO TO 500
        END IF
       END IF
      END IF
560   IF(OPSTK(J)+OPSTK(J-1).EQ.-7)THEN
       OPSTK(J)=0
       OPSTK(J-1)=0
       J=J-2
      END IF
      GO TO 500
     ELSE
      N=N+1
      POLLSH(N)=ATRNS(I)
      GO TO 500
     END IF
     END

C
C****
C

     SUBROUTINE CAUSE(POLLSH,NAM,F1M,PO,TP,CELLS,CLR,CLC,
     *           NAMES,NML,TEMP)

     INTEGER CLR,CLC,NML,TP,PO
```

```
      CHARACTER NAMES(CLR)*(12)
      INTEGER CELLS(CLR,CLC),POLLSH(PO),TEMP(TP),FLAG,STEP,F1M,NAM
     +,LASTOP,L,K,S,M,I, TEST
      DO 700 L=1,TP
       TEMP(L)=0
700   CONTINUE
      K=0
      FLAG=1
710   K=K+1
      IF(POLLSH(K).GE.0)THEN
       IF(K.GE.F1M)THEN
        IF(FLAG.EQ.1)THEN
         CELLS(NAM,2)=2
         CELLS(NAM,3)=1
         CELLS(NAM,4)=POLLSH(K)
         RETURN
        ELSE
         DO 720 S=1,TP
          CELLS(NAM,S+1)=TEMP(S)
720      CONTINUE
         RETURN
        END IF
       ELSE
        GO TO 710
       END IF
      END IF
      IF(FLAG.EQ.1)THEN
       LASTOP=POLLSH(K)
       IF(LASTOP.EQ.-1)THEN
        TEMP(1)=-1
       ELSE
        TEMP(1)=1
       END IF
       TEMP(2)=2
       TEMP(3)=POLLSH(K-2)
       TEMP(4)=POLLSH(K-1)
       IF(F1M.EQ.3)THEN
        DO 730 M=1,3
         POLLSH(M)=0
730     CONTINUE
        F1M=1
        K=0
        FLAG=0
        GO TO 710
       END IF
       CALL COMPRS (POLLSH,PO,K,F1M,FLAG)
       GO TO 710
      END IF
      IF(POLLSH(K).EQ.LASTOP)THEN
       IF(POLLSH(K-2).EQ.0)THEN
        TEMP(2)=TEMP(2)+1
        STEP=TEMP(2)
        TEMP(2+STEP)=POLLSH(K-1)
        IF(F1M.EQ.3)THEN
         DO 750 M=1,3
```

```
          POLLSH(M)=0
750       CONTINUE
          F1M=1
          FLAG=0
          K=0
          GO TO 710
         END IF
         CALL COMPRS (POLLSH,PO,K,F1M,FLAG)
         GO TO 710
        END IF
        IF(POLLSH(K-1).EQ.0)THEN
         TEMP(2)=TEMP(2)+1
         STEP=TEMP(2)
         TEMP(2+STEP)=POLLSH(K-2)
         IF(F1M.EQ.3)THEN
          DO 780 M=1,3
           POLLSH(M)=0
780       CONTINUE
          F1M=1
          FLAG=0
          K=0
          GO TO 710
         END IF
         CALL COMPRS (POLLSH,PO,K,F1M,FLAG)
         GO TO 710
        END IF
       END IF
       LASTOP=POLLSH(K)
850    CALL ALLOC ('      ',I,TEST,NAMES,NML,CLR)
       NAMES(I)='          '
       DO 920 S=1,TP
        CELLS(I,S+1)=TEMP(S)
920    CONTINUE
       DO 930 S=1,TP
        TEMP(S)=0
930    CONTINUE
       DO 931 L=K-1,1,-1
        IF(POLLSH(L).EQ.0)THEN
         POLLSH(L)=I
         GO TO 935
        END IF
931    CONTINUE
935    IF(LASTOP.EQ.-1)THEN
       TEMP(1)=-1
      ELSE
       TEMP(1)=1
      END IF
      TEMP(2)=2
      TEMP(3)=POLLSH(K-2)
      TEMP(4)=POLLSH(K-1)
      IF(F1M.EQ.3)THEN
       DO 940 M=1,3
        POLLSH(M)=0
940    CONTINUE
       F1M=1
```

```
      K=0
      FLAG=0
      GO TO 710
    END IF
    CALL COMPRS (POLLSH,PO,K,F1M,FLAG)
    GO TO 710
    END
    SUBROUTINE COMPRS (POLLSH,PO,K,F1M,FLAG)
    INTEGER PO
    INTEGER POLLSH(PO),K,F1M,FLAG,S
    POLLSH(K-2)=0
    DO 950 S=K+1,F1M
      POLLSH(S-2)=POLLSH(S)
950  CONTINUE
    F1M=F1M-2
    FLAG=0
    K=0
    END



    SUBROUTINE ANCELL( ILJ,KS,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
C
    INTEGER CLR,CLC,SMR,SMC,NML

    PARAMETER (IW=10, IL=40, LVECT=50)
    COMMON /ONE/ FLAG
    COMMON/SYGIN/FD
C
    INTEGER  CELLS(CLR,CLC), NM(IW), NJ(IW), K(IW), NZ(IW),
   +      VECTOR ( LVECT ),SYMPTS ( SMR,SMC )
    CHARACTER NAMES(CLR)*(12), FLAG(IW,IL)*40
C
C**** INITIALISE THE ARRAY K
C
    K(1) = KS
    DO 5, I=2,IW

      K ( I ) = 0

 5   CONTINUE

C
C**** CHECK THE LENGTH OF THE DRAWING AREA
C
    CALL EXCEED( ILJ,IL,CELLS,CLR,CLC,NAMES,NML )
C
C**** PUT THE TOP BOX OF THE TREE IN THE MIDDLE OF A ROW
C**** OF ARRAY FLAG AND CODE NAME IT
C
    IF (2*(IW/2) .EQ. IW) THEN
      FLAG(IW/2,ILJ) = 'Z'
      KIW = IW/2
    ELSE
```

```
      FLAG(1+IW/2,ILJ) = 'Z'
      KIW = 1+IW/2
    END IF
C
C**** FIND THE NUMBER OF BRANCHES AND MARK THEM IN THE NEXT
C**** ROW OF ARRAY FLAG
    CALL BRANCH(ILJ,K(1),KIW,CELLS,CLR,CLC,NAMES,NML)
C
C**** CHECK TO SEE IF ALL THE BOXES MARKED IN THE LATEST ROW
C**** OF ARRAY FLAG HAVE NAMES
C
  10 L = 0
    J = 1
    N8 = 0
    DO 40 I = 1,IW

      IF (K(I) .NE. 0) THEN

        CALL SETROW ( VECTOR, K ( I ), CELLS, SYMPTS, CLR, CLC, SMR,
   +              SMC )

        N1 = 3
        DO 20 M = J,IW
          IF (FLAG(M,ILJ+1) .NE. 'EMPTY') THEN
            N1 = N1+1
            IF ( VECTOR ( N1 ) . EQ. 0 ) THEN
              GO TO 30
            END IF
            N8 = N8+1
            IF (CELLS(VECTOR(N1),1) .EQ. 0) THEN
              L = L+1
              NM(L) = I
              NJ(L) = N1
              NZ(L) = N8
            END IF
          END IF
 20     CONTINUE
C
 30     J = M
      ELSE
        GO TO 50
      END IF
 40 CONTINUE
C
 50 IF (L .EQ. 0) THEN
      RETURN
    END IF
C
C**** IF THERE ARE UNNAMED BOXES CONTINUE WITH THE ANALYSIS
C**** OF THE DATA IN ARRAY CELLS
C
    ILJ = ILJ+1
C
C**** CHECK THE LENGTH OF THE DRAWING AREA
C
```

```
      CALL EXCEED( ILJ,IL,CELLS,CLR,CLC,NAMES,NML )
C
C**** DEAL WITH EACH UNNAMED BOX IN TURN
C
      DO 80 LA = 1,L
        KA = 0
        DO 60 LB = 1,IW
          IF (FLAG(LB,ILJ) .NE. 'EMPTY') THEN
            KA = KA+1
            IF (KA .EQ. NZ(LA)) THEN
              GO TO 70
            END IF
          END IF
 60     CONTINUE
C
C**** FIND THE ROW OF ARRAY CELLS WHICH HAS INFORMATION ABOUT
C**** THE UNNAMED BOX AND USE THE SUBROUTINE BRANCH TO FIND
C**** THE NUMBER OF BRANCHES BELOW IT AND TO MARK THEM IN
C**** THE NEXT ROW OF ARRAY FLAG
C
 70     K1 = CELLS(K(NM(LA)),NJ(LA))
        CALL BRANCH(ILJ,K1,LB,CELLS,CLR,CLC,NAMES,NML)
 80   CONTINUE
C
C**** FIND THE ROWS OF CELLS WHICH HOLD INFORMATION ABOUT THE
C**** MARKED BOXES IN THE LATEST ROW OF ARRAY FLAG
C
      ML = 0
      DO 100 LD = 1,IW
        IF (K(LD) .NE. 0) THEN
          DO 90 LF = 4,CLC
            IF (CELLS(K(LD),LF) .NE. 0) THEN
              ML = ML+1
              NM(ML) = CELLS(K(LD),LF)
            ELSE
              GO TO 100
            END IF
 90       CONTINUE
        ELSE
          GO TO 110
        END IF
100   CONTINUE
C
C**** STORE THE ROWS FOUND ABOVE IN ARRAY K AND START AGAIN
C**** CHECKING THE LATEST ROW OF ARRAY FLAG
C
110   DO 120 LP = 1,IW
        IF (LP .LE. ML) THEN
          K(LP) = NM(LP)
        ELSE
          K(LP) = 0
        END IF
120   CONTINUE
      GO TO 10
C
```

```
      END


      SUBROUTINE BRANCH(MLJ,K2,KMW,CELLS,CLR,CLC,NAMES,NML)
C
      INTEGER CLR,CLC,NML

      PARAMETER (IW=10, IL=40)

      COMMON /ONE/FLAG
      COMMON/SYGIN/FD
C
      INTEGER  CELLS(CLR,CLC)

      CHARACTER FLAG(IW,IL)*40, NAMES(CLR)*(12)
C
C**** CHECK THE SECOND ELEMENT IN THE ROW OF ARRAY CELLS
C
C**** FIND THE NUMBER OF BRANCHES AND CHECK THE WIDTH
C**** OF THE DRAWING AREA
C
      NUM = CELLS(K2,3)
      IF (NUM .GT. IW) THEN
        PRINT *,'*****  WIDTH OF DRAWING AREA TOO SMALL  *****'
        CALL GINEND
        CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
        STOP
      END IF
C
C**** CHECK TO SEE IF THE NUMBER OF BRANCHES IS EVEN OR ODD
C**** AND USE SUBROUTINE FILLF TO MARK THEM IN THE NEXT
C**** ROW OF ARRAY FLAG
C
      IF (2*(NUM/2) .EQ. NUM) THEN
        CALL FILLF(KMW,NUM,MLJ,CELLS,CLR,CLC,NAMES,NML)
      ELSE
        FLAG(KMW,MLJ+1) = '10'
        CALL FILLF(KMW,NUM,MLJ,CELLS,CLR,CLC,NAMES,NML)
      END IF
C
      END


      SUBROUTINE FILLF(KMW,NUM,MLJ,CELLS,CLR,CLC,NAMES,NML)

      INTEGER CLR,CLC,NML

      PARAMETER (IW=10, IL=40)

      COMMON /ONE/ FLAG
       COMMON/SYGIN/FD

      CHARACTER FLAG(IW,IL)*40, CHARA, NAMES(CLR)*(12)
      INTEGER CELLS(CLR,CLC)
```

```
C**** DERIVE A CODED NAME FOR EACH MARKED ELEMENT OF ARRAY FLAG
      IF (FLAG(KMW,MLJ) .EQ. 'Z') THEN
        IT = ICHAR( 'A' )
      ELSE
        IT = ICHAR( '1' )
      END IF
C
C**** CODE NAME THE MIDDLE ELEMENT OF THE NEXT ROW OF THE
C**** ARRAY FLAG IF IT IS MARKED
C
      IF (FLAG(KMW,MLJ+1) .EQ. '10') THEN
        CHARA = CHAR( IT+NUM/2 )
        FLAG(KMW,MLJ+1) = CHARA(1:1)//FLAG(KMW,MLJ)
      END IF
C
C**** MARK THE LEFT HAND SIDE BRANCHES IN THE NEXT ROW OF
C**** THE ARRAY FLAG AND CODE NAME THEM
C
      KLW = KMW
      DO 20 N = 1,NUM/2
        IF (KLW-N .LT. 1) THEN
C
C****     USE SUBROUTINE CHECK1 TO SEE IF THE MARKED BOXES
C****     CAN BE MOVED TO THE RIGHT
C
          CALL CHECK1( KLW,MLJ,CELLS,CLR,CLC,NAMES,NML )
        ELSE
C
C****     USE SUBROUTINE SHIFT TO SEE IF THE ELEMENT OF THE
C****     ARRAY FLAG ABOUT TO BE MARKED CAN BE MARKED AND
C****     IF NOT TRY TO MOVE THE ALREADY MARKED BRANCHES
C
          CALL SHIFT( MLJ,KMW,KLW,N,CELLS,CLR,CLC,NAMES,NML )
        END IF
        CHARA = CHAR( IT+NUM/2-N )
        FLAG(KLW-N,MLJ+1) = CHARA(1:1)//FLAG(KMW,MLJ)
 20   CONTINUE
C
C**** MARK THE RIGHT HAND SIDE BRANCHES AND CODE NAME THEM
C
      DO 40 N = 1,NUM/2
        IF (KLW+N .GT. IW) THEN
C
C****     USE SUBROUTINE CHECK2 TO TRY AND MOVE THE MARKED
C****     BRANCHES TO THE LEFT
C
          CALL CHECK2( MLJ,KLW,CELLS,CLR,CLC,NAMES,NML )
        END IF
        CHARA = CHAR( IT+NUM/2+N )
        FLAG(KLW+N,MLJ+1) = CHARA(1:1)//FLAG(KMW,MLJ)
 40   CONTINUE

      END
```

```
      SUBROUTINE CHECK1( KLW,MLJ,CELLS,CLR,CLC,NAMES,NML )
C
      INTEGER CLR,CLC,NML

      PARAMETER (IW=10, IL=40)

      COMMON /ONE/ FLAG


      INTEGER CELLS(CLR,CLC)

      CHARACTER FLAG(IW,IL)*40, NAMES(CLR)*(12)
C
      KLW = KLW+1
      DO 20 IA = 1,IW
        IF (FLAG(IA,MLJ+1) .EQ. 'EMPTY') THEN
          DO 10 IB = IA,2,-1
   10       FLAG(IB,MLJ+1) = FLAG(IB-1,MLJ+1)
          FLAG(1,MLJ+1) = 'EMPTY'
          GO TO 30
        END IF
   20 CONTINUE
C
   30 IF (FLAG(1,MLJ+1) .NE. 'EMPTY') THEN
        PRINT *,'*****  DRAWING AREA NOT WIDE ENOUGH  *****'
        CALL GINEND
        CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
        STOP
      END IF
C
      END



      SUBROUTINE CHECK2( MLJ,KLW,CELLS,CLR,CLC,NAMES,NML )
C
      INTEGER CLR,CLC,NML

      PARAMETER (IW=10, IL=40)

      COMMON /ONE/ FLAG

      CHARACTER FLAG(IW,IL)*40, NAMES(CLR)*(12)
C
      INTEGER CELLS(CLR,CLC)

      KLW = KLW-1
      DO 20 IA = IW,1,-1
        IF (FLAG(IA,MLJ+1) .EQ. 'EMPTY') THEN
          DO 10 IB = IA,IW-1
   10       FLAG(IB,MLJ+1) = FLAG(IB+1,MLJ+1)
          FLAG(IW,MLJ+1) = 'EMPTY'
          GO TO 30
        END IF
   20 CONTINUE
```

```
C
 30 IF (FLAG(IW,MLJ+1) .NE. 'EMPTY') THEN
      PRINT *,'*****  DRAWING AREA TOO NARROW  *****'
      CALL GINEND
      CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
      STOP
    END IF
C
    END



    SUBROUTINE SHIFT( MLJ,KMW,KLW,N,CELLS,CLR,CLC,NAMES,NML )
C
    INTEGER CLR,CLC,NML

    PARAMETER (IW=10, IL=40)

    COMMON /ONE/ FLAG

    CHARACTER FLAG(IW,IL)*40, NAMES(CLR)*(12)

    INTEGER CELLS(CLR,CLC)
C
 10 KSW = KLW
    K = 0
    DO 110 IA = IW,KSW-N,-1
      IF (FLAG(IA,MLJ+1) .NE. 'EMPTY') THEN
        IF (FLAG(IA,MLJ+1)(2:) .NE. FLAG(KMW,MLJ)) THEN
          IF (NALEN( FLAG(KMW,MLJ) ) .EQ. 2) THEN
            IF (FLAG(IW,MLJ) .EQ. 'EMPTY' .AND.
     1              FLAG(IW,MLJ+1) .EQ. 'EMPTY') THEN
              DO 20 IB = IW,KMW+1,-1
 20             FLAG(IB,MLJ) = FLAG(IB-1,MLJ)
              FLAG(KMW,MLJ) = 'EMPTY'
              KMW = KMW+1
              DO 30 IC = IW,IA+2,-1
 30             FLAG(IC,MLJ+1) = FLAG(IC-1,MLJ+1)
              FLAG(IA+1,MLJ+1) = 'EMPTY'
              KLW = KLW+1
              K = 1
            END IF
            IF (FLAG(1,MLJ) .EQ. 'EMPTY' .AND.
     1              FLAG(1,MLJ+1) .EQ. 'EMPTY') THEN
              DO 40 ID = 1,KMW-2
 40             FLAG(ID,MLJ) = FLAG(ID+1,MLJ)
              FLAG(KMW-1,MLJ) = 'EMPTY'
              DO 50 IE = 1,IA-1
 50             FLAG(IE,MLJ+1) = FLAG(IE+1,MLJ+1)
              FLAG(IA,MLJ+1) = 'EMPTY'
              K = K+2
            END IF
          END IF
          IF (K .EQ. 3) THEN
            GO TO 120
```

```
          ELSE IF (K .EQ. 2) THEN
            GO TO 90
          END IF
          DO 60 IG = 1,IA-1
            IF (FLAG(IG,MLJ+1) .EQ. 'EMPTY') THEN
              GO TO 70
            END IF
 60         CONTINUE
C
 70         IF (IG .LT. IA) THEN
            DO 80 JA = IG,IA-1
 80           FLAG(JA,MLJ+1) = FLAG(JA+1,MLJ+1)
            FLAG(IA,MLJ+1) = 'EMPTY'
            K = K+5
          END IF
          IF (K .EQ. 6) THEN
            GO TO 120
          END IF
 90         IF (FLAG(IW,MLJ+1) .EQ. 'EMPTY') THEN
            DO 100 JB = IW,IA+2,-1
 100          FLAG(JB,MLJ+1) = FLAG(JB-1,MLJ+1)
            FLAG(IA+1,MLJ+1) = 'EMPTY'
            KLW = KLW+1
            K = 8
            GO TO 120
          END IF
        END IF
      END IF
110  CONTINUE
    IF (K .EQ. 0) THEN
      IF (FLAG(KLW-N,MLJ+1) .NE. 'EMPTY') THEN
        PRINT *,'*****  WIDTH OF DRAWING AREA EXCEEDED  *****'
        CALL GINEND
        CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
        STOP
      END IF
      RETURN
    END IF
120  GO TO 10
C
    END



    SUBROUTINE BOXSTR(KEY,ANAME,XX1,YY1,XX2,YY2,ERROR)

    PARAMETER (NML=12)
    REAL COORD(40,4),XX1,YY1,XX2,YY2
    INTEGER KEY,POINTA
    CHARACTER ANAME*(*),LIST(40)*(NML)

    LOGICAL ERROR
    SAVE COORD,POINTA

    IF(KEY.EQ.1)THEN
```

```
      POINTA = 1
      DO 10 I =1,40
      DO 5 J =1,4
         COORD(I,J) = 0
 5    CONTINUE
10    CONTINUE
     ERROR = .FALSE.
     ELSEIF(KEY.EQ.2)THEN
       IF(POINTA.GT.40)THEN
       ERROR = .TRUE.
       RETURN
       ENDIF
     LIST(POINTA) = ANAME
     COORD(POINTA,1) = XX1
     COORD(POINTA,2) = YY1
     COORD(POINTA,3) = XX2
     COORD(POINTA,4) = YY2
     POINTA = POINTA + 1
     ERROR = .FALSE.
     ELSEIF(KEY.EQ.3)THEN
       DO 100 I = 1,POINTA-1
         IF(XX1.GE.COORD(I,1).AND.XX1.LE.COORD(I,3).AND.YY1
    +.GE.COORD(I,2).AND.YY1.LE.COORD(I,4))THEN
         ANAME = LIST(I)
         ERROR = .FALSE.
         RETURN
         ENDIF
100  CONTINUE
     ERROR =.TRUE.
     ENDIF
     RETURN
     END



     SUBROUTINE DRAWTR(I,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
C
     INTEGER CLR,CLC,SMR,SMC,NML
     PARAMETER (IW=10, IL=40, LVECT=50)
     COMMON /ONE/ FLAG
     COMMON /TWO/ JLJ, KLJ, KR, IOPT
C
     INTEGER  CELLS(CLR,CLC), TAV(IW), SAV(IW), IOPT(3),
    +       SYMPTS ( SMR, SMC ), VECTOR ( LVECT )
     CHARACTER NAMES(CLR)*(12), FLAG(IW,IL)*40

     DIMENSION  X1(IW), X2(IW), Y1(IW), Y2(IW)

C
C**** INITIALISE THE ARRAYS: TAV SAV X1 X2 Y1 Y2
     DO 5, INIT = 1, IW

       TAV ( INIT ) = 0
       SAV ( INIT ) = 0
       X1  ( INIT ) = 0
```

```
      X2  ( INIT ) = 0
      Y1  ( INIT ) = 0
      Y2  ( INIT ) = 0

 5   CONTINUE

C
C**** POSITION THE PEN TO DRAW THE TOP BOX OF THE FAULT TREE
    XPOS = (JLJ-1)*20.0 + KR*10.0
C
    DO 10 LB = 1,IW
      IF (FLAG(LB,JLJ) .EQ. 'Z') THEN
        GO TO 20
      END IF
 10 CONTINUE
C
 20 YMID = (2*LB-1)*17.0
    CALL MOVTO2( XPOS,YMID )
C
C**** DRAW THE BOX AND WRITE IN IT
    CALL ABOX(NAMES(I),NML)

C
C**** DRAW THE GATE IF THERE IS ONE
      CALL MOVTO2(XPOS+7,YMID)

    IF (CELLS(I,2) .EQ. 0 .AND. CELLS(I,3) .GT. 1
   *   .OR. CELLS ( I, 2 ) .EQ. 1 ) THEN
      CALL LINBY2( 4.0,0.0 )
      CALL ORED
    ELSE IF (CELLS(I,2) .EQ. +2 .OR. CELLS(I,2) .EQ.0 .AND.
   *    CELLS(I,3) .EQ.1) THEN
      CALL LINBY2( 9.0,0.0 )
    ELSE IF (CELLS(I,2) .EQ. -1) THEN
      CALL LINBY2( 4.0,0.0 )
      CALL ANDED
    END IF
C
C**** FIND THE POSITION OF EACH BOX IN THE NEXT ROW OF FLAG
    L5 = 0
    L6 = 4
    DO 50 LD = 1,IW
      IF (FLAG(LD,JLJ+1) .NE. 'EMPTY') THEN
        DO 30 LF = L6,CLC

          CALL SETROW ( VECTOR, I, CELLS, SYMPTS,
   +                CLR, CLC, SMR, SMC )
          IF (VECTOR(LF) .NE. 0) THEN
            L5 = L5+1
            IF (L5 .EQ. 1) THEN
              LD1 = LD
              LD2 = LD
            ELSE
              LD2 = LD
            END IF
```

```
C
C****         SAVE THE ROW OF ARRAY CELLS WHICH HOLDS
C****         INFORMATION ABOUT THE BOX
          TAV(L5) = VECTOR(LF)
          L6 = LF+1
          GO TO 40
        ELSE
   PRINT *,'*****  ERROR IN ANALYSIS OF DATA STRUCTURE  *****'
          CALL GINEND
          CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
          STOP
        END IF
 30     CONTINUE
C
C****     POSITION THE PEN FOR DRAWING
 40     XPOS1 = XPOS + 20.0
       YMID = (2*LD-1)*17.0
       CALL MOVTO2( XPOS1,YMID )
C
C****     SAVE THE CO-ORDINATES OF THE PEN'S POSITION
       X1(LD) = XPOS1 - 4.0
       Y1(LD) = YMID
C
C****     CHECK TO SEE IF THE BOX HAS A NAME INSIDE IT
       IF (CELLS(TAV(L5),1) .EQ. 1) THEN
C
C****       DRAW THE BOX AND WRITE IN IT
         CALL ABOX( NAMES(TAV(L5)),NML)

         CALL MOVTO2(X1(LD)+7,Y1(LD))
         CALL MOVBY2( -7.0,0.0 )
         CALL LINBY2( 4.0,0.0 )
       END IF
     END IF
50   CONTINUE
C
C**** CONNECT THE DRAWN BOXES BETWEEN THEM AND TO THE GATE
C
   IF ( L5 .NE. 0 ) THEN

     CALL MOVTO2( X1(LD1),Y1(LD1) )
   CALL LINTO2( X1(LD2),Y1(LD2) )

   END IF

C
C**** DO THE REST OF THE DRAWING BY CONSIDERING THE ROWS
C**** OF ARRAY FLAG ONE AT A TIME
   DO 110 MD = JLJ+1,KLJ
     L7 = 0
     L9 = 0
C
C****  FIND THE POSITION OF EACH BOX IN THE NEXT ROW
C****  OF ARRAY FLAG
     DO 90 MA = 1,IW
```

```
      IF (FLAG(MA,MD) .NE. 'EMPTY') THEN
       L7 = L7+1
C
C****     FIND OUT IF THE BOX IS NAMED
       IF (CELLS(TAV(L7),1) .EQ. 1) THEN
         IF (CELLS(TAV(L7),2) .EQ. -2) THEN
           CONTINUE
         ELSE
C
C****        SHOW THE NUMBER OF BRANCHES BELOW THE BOX
           NUM = CELLS(TAV(L7),3)
           CALL MOVTO2( X1(MA)+11.0,Y1(MA) )
           CALL NUMBRA( NUM )
         END IF
       ELSE
C
C****      FOR EACH UNNAMED BOX DRAW THE BRANCHES BELOW IT
         L5 = 0
         L6 = 4
         DO 80 KA = 1,IW
           IF (FLAG(MA,MD) .EQ. FLAG(KA,MD+1)(2:)) THEN
             L9 = L9+1
             DO 60 LF = L6,CLC

             CALL SETROW ( VECTOR, TAV ( L7 ), CELLS,
     +                  SYMPTS, CLR, CLC, SMR, SMC )
               IF (VECTOR(LF) .NE. 0) THEN
                 L5 = L5+1
                 IF (L5 .EQ. 1) THEN
                   KA1 = KA
                   KA2 = KA
                 ELSE
                   KA2 = KA
                 END IF
C
C****               SAVE THE ROW OF CELLS WHICH HAS
C****               INFORMATION ABOUT THE BOX
                 SAV(L9) = VECTOR(LF)
                 L6 = LF+1
                 GO TO 70
               ELSE
         PRINT *,'*****  DATA STRUCTURE NOT ANALYSED WELL  *****'
                 CALL GINEND
                 CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,
     *                NAMES,NML )
                 STOP
               END IF
   60        CONTINUE
C
C****        POSITION THE PEN TO DRAW THE BOX
   70        XPOS1 = XPOS + (MD+1-JLJ)*20.0
             YMID = (2*KA - 1)*17.0
             CALL MOVTO2( XPOS1,YMID )
C
C****        SAVE THE CO-ORDINATES OF THE PEN'S POSITION
```

```
            X2(KA) = XPOS1-4.0
            Y2(KA) = YMID
C
C****          CHECK TO SEE IF THE BOX HAS A NAME
            IF (CELLS(SAV(L9),1) .EQ. 1) THEN
C
C****            DRAW THE BOX AND WRITE IN IT
             CALL ABOX( NAMES(SAV(L9)),NML )

              CALL MOVTO2(X2(KA)+7,Y2(KA))
              CALL MOVBY2( -7.0,0.0 )
              CALL LINBY2( 4.0,0.0 )
            END IF
           END IF
 80        CONTINUE
C
C****       CONNECT THE DRAWN BOXES BETWEEN THEM
C****       AND TO THE GATE
          CALL MOVTO2( X2(KA1),Y2(KA1) )
          CALL LINTO2( X2(KA2),Y2(KA2) )
C
C****       DRAW THE GATE
          Y3 = (Y2(KA1) + Y2(KA2)) / 2.0
          CALL MOVTO2( X1(MA),Y1(MA) )
          CALL LINTO2( X2(KA1)-5.0,Y3 )
          IF (CELLS(TAV(L7),2) .EQ. 1) THEN
            CALL ORED
          ELSE IF (CELLS(TAV(L7),2) .EQ. -1) THEN
            CALL ANDED
          ELSE
           PRINT *,'*****  DATA STRUCTURE CONTAINS ERRORS  *****'
            CALL GINEND
            CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
            STOP
          END IF
        END IF
      END IF
 90   CONTINUE
     DO 100 KB = 1,IW
       TAV(KB) = SAV(KB)
       X1(KB) = X2(KB)
100   Y1(KB) = Y2(KB)
C
110  CONTINUE
     CALL CHAMOD

    END


    SUBROUTINE ABOX(A,NML)

    INTEGER NML, ERRNO, VALUE,FINAL
    REAL POSX, POSY, POSZ, AWID, ALEN1
    PARAMETER (ALEN=24.0, BLENG=50 )
    CHARACTER  A*(12)
```

```
      LOGICAL ERROR
C
      CALL DRIVER ( 3, VALUE, ERRNO )

      IF ( VALUE .EQ. 1 ) THEN

       ALEN1 = 30
       AWID = 7

       ELSE

       ALEN1 =  FINAL ( A ) * 3.68
       AWID = 7

      END IF
      ALET = (ALEN1 / 2.0)
      CALL LINBY2( 0.0,ALET )
      CALL POSPIC(XX2,YY2)
      CALL LINBY2( AWID,0.0 )
      CALL LINBY2( 0.0,-ALEN1 )
      CALL POSPIC(XX1,YY1)
      CALL LINBY2( -AWID,0.0 )
      CALL LINBY2( 0.0,ALET )
      CALL POSSPA ( POSX, POSY, POSZ )
      CALL BOXSTR(2,A,XX1,YY1,XX2,YY2,ERROR)


      CALL MOVBY2( AWID - 1, -ALET+1 )
      NC = NALEN( A )
      CALL CHAHOL ( A ( 1 : FINAL ( A ) ) )


*
****   TAKE ALTERNATIVE ACTIONS FOR DIFFERENT OUTPUT DEVICES
****   FOR CHARACTER OUTPUT
*

      IF ( VALUE .EQ. 1 ) THEN

       CALL CHASIZ ( 2.6, 4.2 )
       CALL MOVBY2 ( 2.5, ( 15. - NC ) / 2 + 1 )

      ELSE

       CALL CHASIZ( (ALEN1-1)/NC,5.0 )
       CALL CHAANG ( 90.0 )
       CALL PENSEL( 7,0.0,0 )

      END IF

      IF ( VALUE .EQ. 1 ) THEN

       CALL MOVTO2 ( POSX + 7.0 , POSY )

      ELSE
```

```
      YP = -(NML*(ALEN1-1)/NC) + ALET - 1.0
      CALL MOVBY2( 1.0,YP )
      CALL PENSEL( 1,0.0,0 )

    END IF
    END


    SUBROUTINE ORED

    CALL LINBY2( 0.0,5.0 )
    CALL LINBY2( 5.0,0.0 )
    CALL LINBY2( 0.0,-10.0 )
    CALL LINBY2( -5.0,0.0 )
    CALL LINBY2( 0.0,5.0 )
    CALL MOVBY2( 5.0,0.0 )
    END


    SUBROUTINE AND
    CALL MOVBY2( 5.0,-5.0 )
    CALL ARCBY2( 0.0,5.0,0.0,10.0,0 )
    CALL MOVBY2( 0.0,-5.0 )
    END


    SUBROUTINE STAR
    INTEGER NOTERM, ERRNO
    CALL PENSEL( 2,0.0,0 )
    CALL LINBY2( 2.0,0.0 )
    CALL ARCBY2( 2.0,0.0,0.0,0.0,0 )
    CALL MOVBY2( 3.0,-1.0 )

    CALL DRIVER ( 3, NOTERM, ERRNO )

    IF ( NOTERM .EQ. 1 ) THEN

*       NEWBURY TERMINAL

     CONTINUE

    ELSE

     CALL CHASIZ( 3.0,2.0 )
     CALL CHAANG ( 90.0 )

    END IF

    CALL PENSEL( 5,0.0,0 )
    CALL CHAASC( 42 )
```

```
      CALL PENSEL( 1,0.0,0 )
      END



      SUBROUTINE NUMBRA( I )

      INTEGER NOTERM, ERRNO
      CALL PENSEL( 2,0.0,0 )
      CALL LINBY2( 2.0,0.0 )
      CALL ARCBY2( 2.0,0.0,0.0,0.0,0 )
      CALL MOVBY2( 3.0,-1.0 )

      CALL DRIVER ( 3, NOTERM, ERRNO )

      IF ( NOTERM .EQ. 1 ) THEN

*       NEWBURY

       CALL MOVBY2 ( 3., 2. )

      ELSE

       CALL CHASIZ( 3.0,2.0 )
       CALL CHAANG ( 90.0 )

       CALL PENSEL( 5,0.0,0 )
       CALL CHAINT( I,1 )

      END IF
      CALL PENSEL( 1,0.0,0 )
      END



      SUBROUTINE EXCEED( ILJ,IL,CELLS,CLR,CLC,NAMES,NML )
C
      INTEGER CLR,CLC,NML
      COMMON /TWO/ JLJ, KLJ, KR, IOPT
      CHARACTER NAMES(CLR)*(12)
      INTEGER  IOPT(3), CELLS(CLR,CLC)
C
C**** CHECK TO SEE IF THE LENGTH OF THE DRAWING AREA
C**** HAS  BEEN  EXCEEDED
      IF ( ILJ+1 .GT. IL ) THEN
        IF (JLJ .EQ. 1) THEN
      PRINT *,'*****  LENGTH OF DRAWING AREA TOO SMALL  *****'
          CALL GINEND
          CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
          STOP
        END IF
      PRINT *,'*****  PROGRAM RUN OUT OF DRAWING AREA  *****'
      PRINT *, '*****  PLEASE START AGAIN WITH THE REST OF THE
     1DATA  *****'
        CALL GINEND
```

```
      CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
      STOP
    END IF
    END



    SUBROUTINE GINEND
C
    PARAMETER ( YMAX=350.0, XMAX=1030.0 )
    COMMON /TWO/ JLJ, KLJ, KR, IOPT
    INTEGER  IOPT(3)
C
C**** DRAW A FRAME AND CLOSE GINO
    CALL MOVTO2( 0.0,0.0 )
    CALL SHIFT2( -15.0,-5.0 )
    CALL MOVTO2( 0.0,0.0 )
    XPOS = KLJ*20.0 + KR*10.0 + 30.0
*
*   NO FRAME ON INTERACTIVE TERMINAL
*
    GO TO 10

    CALL FRAME( XPOS,YMAX,1 )
    CALL FRAME( XPOS-2.0,YMAX-2.0,2 )
    CALL FRAME( XPOS-4.0,YMAX-4.0,5 )
    CALL FRAME( XPOS-6.0,YMAX-6.0,7 )
 10 CALL DEVEND
    END



    SUBROUTINE FRAME( XPOS,YMAX,ICOL )
C
    CALL PENSEL( ICOL,0.0,0 )
    CALL LINBY2( XPOS,0.0 )
    CALL LINBY2( 0.0,YMAX )
    CALL LINBY2( -XPOS,0.0 )
    CALL LINBY2( 0.0,-YMAX )
    CALL MOVBY2( 1.0,1.0 )
    END



    SUBROUTINE DAWRI(INDATA,INDEX,CELLS,CLR,CLC,NAMES,NML)
C
    INTEGER CLR,CLC,SMR,SMC,NM
    PARAMETER (IW=10, IL=40)
    COMMON /ONE/ FLAG
    COMMON /TWO/ JLJ, KLJ, KR, IOPT
    INTEGER  CELLS(CLR,CLC), IOPT(3)
    CHARACTER NAMES(CLR)*(12), INDATA*(*), FLAG(IW,IL)*40
C
    IF (INDEX .EQ. 1) THEN
*     IF (IOPT(1) .EQ. 1) THEN
```

```
C
C****     WRITE OUT THE CONTENTS OF ARRAYS CELLS AND NAMES
*        WRITE (32,10) ( (CELLS(I2,I1), I1 = 1,CLC), I2 = 1,CLR)
*  10     FORMAT (1H1, 'CONTENTS OF ARRAY CELLS:' / 1H , 27('*')
*   1          /// 50(1H , 2(5(5X,I5) /)/ ) )
*        WRITE (32,25) (NAMES(I),I=1,CLC)
*  25     FORMAT (1H1, 'CONTENTS OF ARRAY NAMES:' / 1H , 27('*')
*   1          /// (1H , A20) )
*      END IF
*    ELSE IF (INDEX .EQ. 2) THEN
*      IF (IOPT(2) .EQ. 1) THEN
C
C****     WRITE OUT THE INPUT DATA USED
*        IF (JLJ .EQ. 1) THEN
*          WRITE (32, '(1H1, "INPUT DATA USED:" /
*   1          1H , 18("*") // )' )
*        END IF
*        WRITE (32, '(1H , A20)' ) INDATA
*      END IF
*    ELSE IF (INDEX .EQ. 3) THEN
*      IF (IOPT(3) .EQ. 1) THEN
C
C****     WRITE OUT THE CONTENTS OF ARRAY FLAG
*        WRITE (32,30) ( (FLAG(J1,J2), J1 = 1,IW), J2 = 1,IL+1)
*  30     FORMAT (1H1, 'CONTENTS OF ARRAY FLAG:' / 1H , 26('*')
*   1          /// 40(1H , 10A10 / ) )
*      END IF
    END IF
C
    END
    INTEGER FUNCTION NALEN( A )

C
C**** IT COUNTS THE CHARACTERS OF A CHARACTER VARIABLE
C**** BEFORE THE FIRST BLANK CHARACTER
C
    PARAMETER ( NML=12 )
C
    CHARACTER A*(*)
C
    DO 10 K = 1,NML
      IF (A(K:K) .EQ. ' ') THEN
        GO TO 20
      END IF
   10 CONTINUE
C
   20 NALEN = K-1
    END



    SUBROUTINE FSOPEN ( FILNAM, UNIT, FORMAT )

*      THIS SUBROUTINE OPENS A SEQUENTIAL FILE
*      IF THE FILE ALREADY EXISTS, IT IS JUST OPENED
```

```
*     IF THE FILE DOES NOT EXIST, IT IS CREATED FIRST
*
*     FILNAM - CHARACTER - THE NAME OF THE FILE
*     UNIT - INTEGER - UNIT NUMBER FOR OPENING
*     FORMAT - CHARACTER - EITHER 'FORMATTED' OR 'UNFORMATTED'
*
      INTEGER UNIT
       CHARACTER FILNAM * 12, FORMAT * ( * ), STATUS * 7
      LOGICAL THERE



      INQUIRE ( FILE = '       '//FILNAM, EXIST = THERE )

      IF ( THERE ) THEN

        STATUS = 'MODIFY'

      ELSE

        STATUS = 'NEW'

      END IF

      OPEN ( FILE = '       '//FILNAM, UNIT = UNIT,
     +     STATUS = STATUS, FORM = FORMAT, ERR = 900 )
      REWIND(UNIT)
      RETURN

  900 PRINT *, 'ERROR IN OPENING FILE', FILNAM
      STOP
      END



      INTEGER FUNCTION NXTNSP ( TEXT )

*
****    RETURNS THE POSITION OF THE FIRST NON-SPACE CHARACTER IN TEXT
****    RETURNS 0 IF NO NON-SPACE CHARACTER
*
      CHARACTER TEXT * ( * )
      INTEGER TEMP

      DO 10, TEMP = 1, LEN ( TEXT )

        IF ( TEXT ( TEMP : TEMP ) .NE. ' ' ) THEN

          NXTNSP = TEMP
          RETURN

        END IF
```

```
10  CONTINUE

    NXTNSP = 0
    RETURN

    END
    SUBROUTINE INERR ( EQN )

*
****    PRINTS OUT ERROR MESSAGE AND THE FAULTY EQUATION
*

    CHARACTER EQN * ( * )

    OPEN(UNIT=9,FILE='ERROR')
    WRITE ( UNIT = 9, FMT = 900 )
900 FORMAT ( /// 1X, 80('*') /// )
    PRINT *, 'THE FOLLOWING LINE OF INPUT IS NOT A CORRECT EQUATION'
    PRINT *
    PRINT *, EQN
    WRITE ( UNIT = 9, FMT = 900 )
    RETURN
    END



    SUBROUTINE SETROW ( VECTOR, NUMBER, CELLS, SYMPTS,
   +                CLR, CLC, SMR, SMC )

    INTEGER CLR, CLC, SMR, SMC
    INTEGER VECTOR (CLC ),  CELLS ( CLR, CLC ), SYMPTS ( SMR, SMC ),
   +      COUNT, NUMBER


         IF ( CELLS ( NUMBER, 2 )  .EQ. 0 ) THEN

           DO 104, COUNT = 1, CLC

             VECTOR ( COUNT ) = 0

104          CONTINUE

           DO 105, COUNT = 1, 3

             VECTOR ( COUNT ) = CELLS ( NUMBER, COUNT )

105          CONTINUE

           DO 106, COUNT = 4, 3 + CELLS ( NUMBER, 3 )

             VECTOR ( COUNT ) = SYMPTS ( CELLS ( NUMBER,
   +                      COUNT ), 1 )
106          CONTINUE
```

```
      ELSE
      DO 11, COUNT = 1, CLC
        VECTOR ( COUNT ) = CELLS ( NUMBER, COUNT )
11      CONTINUE
      END IF
  RETURN
  END
```

APPENDIX H : LISTING OF THE SOURCE PROGRAMS OF THE GINO-POS VERSION

```
      SUBROUTINE PERQ
C      THIS SUBROUTINE SETS THE DEFAULT VALUES FOR GRAPHICAL OUTPUT.
      REAL PPOS,OTR
      REAL VPOS,XMIN,XMAX,YMIN,YMAX
      INTEGER IW
      LOGICAL FLAG,LOGIC
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      CALL PICCLE
      XMIN = 0.0
      XMAX = 210.0
      YMIN = 0.0
      YMAX = 275.0
      IW   = 0
      R    = 1.0
      TOL  = 0.05
      ITYPE = 0
      FLAG =.TRUE.
      LOGIC = .FALSE.
C      NOW INITIALISE THE TRANSFORMATION MATRIX AND THE START POINT.
       DO 10 I  = 1,3
        DO 5 J = 1,3
         OTR(I,J) = 0.0
    5   CONTINUE
       OTR(I,I) = 1.0
       VPOS(I) = 0.0
       PPOS(I) = 0.0
   10  CONTINUE
      VPOS(3) = 1.0
      RETURN
      END


      SUBROUTINE PICCLE
       EXTERNAL / PASCAL / SCRCLR
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

       CALL SCRCLR
      RETURN
      END


      SUBROUTINE CURSOR(I,X,Y)
       EXTERNAL / PASCAL / STRTAB
       EXTERNAL / PASCAL / RDCURS
       LOGICAI  FLAG, LOGIC
      REAL X,Y
       INTEGER I,IX,IY
       CHARACTER*3 ANS
       CHARACTER*6 BUTTON(0:3)
       COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
```

```
      BUTTON(0) = 'YELLOW'
      BUTTON(1) = 'WHITE'
      BUTTON(2) = 'BLUE'
      BUTTON(3) = 'GREEN'
      CALL STRTAB
      CALL RDCURS(I,IX,IY)
      X = 210.*X1/767/R
      Y = ( 275. - 275.*Y1/1023)/R
      RETURN
      END


      SUBROUTINE ROTAT2(ALPHA)
C     ANGLE ALPHA IN DEGREES
C     ALPHA POSITIVE = ANTICLOCKWISE ROTATION ABOUT ORIGIN
C     ALPHA NEGATIVE = CLOCKWISE ROTATION ABOUT ORIGIN
      REAL PI,A,ALPHA,ROT(3,3)
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      DO 2 I = 1,3
       DO 2 J = 1,3
        ROT(I,J) = 0.0
    2 CONTINUE
      ROT(3,3) = 1.0
      PI = 4.0 * ATAN(1.0)
      A = PI*ALPHA / 180.0
      ROT(1,1) = COS(A)
      ROT(2,2) = COS(A)
      ROT(2,1) = SIN(A)
      ROT(1,2) = -SIN(A)
      CALL RFRESH(ROT)
      CALL MATINV(ROT)
      RETURN
      END



      SUBROUTINE SCALE2(SX,SY)
      REAL SX,SY,SCA(3,3)
      INTEGER I,J
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      DO 2  I = 1,3
      DO 2  J = 1,3
        SCA(I,J) = 0.0
    2 CONTINUE
      SCA(1,1) = SX
      SCA(2,2) = SY
      SCA(3,3) = 1.0
      CALL RFRESH(SCA)
      CALL MATINV(SCA)
      RETURN
      END
```

```
      SUBROUTINE SCALE(S)
      REAL S,SCA(3,3)
      INTEGER I,J
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      DO 10 I = 1,3
       DO 5 J = 1,3
        SCA(I,J) = 0.0
 5     CONTINUE
10    CONTINUE
      SCA(1,1) = S
      SCA(2,2) = S
      SCA(3,3) = 1.0
      CALL RFRESH(SCA)
      CALL MATINV(SCA)
      RETURN
      END



      SUBROUTINE SHEAR2(IDEP,A)
      REAL A,SHE(3,3)
      INTEGER IDEP,I,J
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      DO 10 I=1,3
       DO 5  J = 1,3
        SHE(I,J) = 0.0
 5     CONTINUE
      SHE(I,I) = 1.0
10    CONTINUE
      IF(IDEP.EQ.1) THEN
       SHE(2,1) = A
      ELSE IF(IDEP.EQ.2) THEN
       SHE(1,2) = A
      ELSE
       WRITE(3,100)
      END IF
      CALL RFRESH(SHE)
      CALL MATINV(SHE)
100   FORMAT(1X,'ERROR: FIRST ARGUMENT IN ROUTINE SHEAR2
     ! MUST BE 1 OR 2')
      RETURN
      END



      SUBROUTINE SHIFT2(DX,DY)
      REAL DX,DY,SHI(3,3)
      INTEGER I,J
```

```
      LOGICAL FLAG,LOGIC

      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      DO 10  I = 1,3
       DO 5  J = 1,3
         SHI(I,J) = 0.0
 5      CONTINUE
       SHI(I,I) = 1.0
10     CONTINUE
      SHI(1,3) = DX
      SHI(2,3) = DY
      CALL RFRESH(SHI)
      RETURN
      END



      SUBROUTINE VIEWSE(NHORIZ)
      REAL VIE(3,3)
      INTEGER I,J,NHORIZ
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      DO 10  I = 1,3
       DO 5 J = 1,3
         VIE(I,J) = 0.0
 5      CONTINUE
       VIE(I,I) = 1.0
10     CONTINUE
      IF(NHORIZ.EQ.2) THEN
       VIE(1,1) = 0.0
       VIE(1,2) = 1.0
       VIE(2,1) = 1.0
       VIE(2,2) = 0.0
      CALL RFRESH(VIE)
      ELSE IF(NHORIZ.EQ.1) THEN
      RETURN
      ELSE
       WRITE(3,100)
      END IF
100   FORMAT(1X,'ERROR: ARGUMENT IN ROUTINE VIEWSE MUST BE 2
     ! IF AXIS TO BE PERMUTED, OR 1 IF NOT TO BE')
      RETURN
      END

      SUBROUTINE LINEXX(M1,N1,M2,N2)
      INTEGER I,WLIN,M1,N1,M2,N2

      I=WLIN(FD,M1,N1,M2,N2)
      RETURN
      END

      SUBROUTINE LINTO2(S,T)
       EXTERNAL / PASCAL / LINEXX
      REAL X(2),Y(2),U(2),V(2)
```

```
      INTEGER M(2),N(2),IW
      LOGICAL FLAG,LOGIC
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      CALL MMULT(OTR,VPOS,PPOS)
      X(1) = PPOS(1)
      Y(1) = PPOS(2)
      VPOS(1) = S
      VPOS(2) = T
      CALL MMULT(OTR,VPOS,PPOS)
      X(2) = PPOS(1)
      Y(2) = PPOS(2)
      IF(IW .EQ.2 .OR. IW .EQ. 1) THEN
       CALL WCUTS(X,Y,U,V)
       IF(U(1).EQ.-50.0 .OR. U(2).EQ.-50.0)  RETURN
       DO 10 I = 1,2
        X(I) = U(I)
        Y(I) = V(I)
10     CONTINUE
      END IF
      DO 20 I = 1,2
       M(I) = NINT(767/210.0 * X(I) )
       N(I) = NINT(1023 - (1023/275.0 *Y(I)) )
20    CONTINUE
      CALL LINEXX(M(1),N(1),M(2),N(2))
      RETURN
      END



      SUBROUTINE LINBY2(X,Y)
      REAL X,Y,V1,V2,VPOS
      LOGICAL FLAG,LOGIC
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      V1 = VPOS(1) + X
      V2 = VPOS(2) + Y
      CALL LINTO2(V1,V2)
      RETURN
      END



      SUBROUTINE MOVTO2(X,Y)
      REAL X,Y,VPOS
      LOGICAL FLAG,LOGIC
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      VPOS(1) = X
      VPOS(2) = Y
      RETURN
```

```
END


SUBROUTINE MOVBY2(X,Y)
REAL X,Y,VPOS
LOGICAL FLAG,LOGIC
COMMON/SPCD/VPOS(3),OTR(3,3)
COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

VPOS(1) = VPOS(1) + X
VPOS(2) = VPOS(2) + Y
RETURN
END


SUBROUTINE DRAW2(X,Y,IABS,IVIS)
REAL X,Y
LOGICAL FLAG,LOGIC
INTEGER IABS,IVIS

COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
 IF(IABS.EQ.0 .AND.IVIS.EQ.0) THEN
  CALL MOVBY2(X,Y)
 ELSE IF(IABS.EQ.0 .AND.IVIS.EQ.1) THEN
  CALL LINBY2(X,Y)
 ELSE IF(IABS.EQ.1 .AND.IVIS.EQ.0) THEN
  CALL MOVTO2(X,Y)
 ELSE IF(IABS.EQ.1 .AND.IVIS.EQ.1) THEN
  CALL LINTO2(X,Y)
 ELSE
  PRINT *,'ERROR IN IABS OR IVIS IN ROUTINE DRAW2'
 END IF
RETURN
END


SUBROUTINE CHAHOL( STRING )
 EXTERNAL / PASCAL / GETCUR, SETCUR, PUTCHR
CHARACTER STRING * ( * )
REAL TEMP ( 3, 3)
INTEGER X,Y
LOGICAL FLAG,LOGIC
COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
COMMON / SPCD / VPOS(3),OTR(3,3)
COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW

CALL MMULT(OTR,VPOS,PPOS)
RX = PPOS( 1 )
RY = PPOS( 2 )
X = INT( 767 /210.* RX )
Y = INT(1023- 1023 / 275.* RY )
 CALL SETCUR(INT(767 * RX / 210.),
```

```
     +       INT( 1023 - 1023 * RY / 275.)
          DO 10, I = 1, LEN (STRING)
             CALL PUTCHR ( STRING (I:I))
   10    CONTINUE
         CALL GETCUR ( X,Y)
         Y = 1023 - Y
         VPOS ( 1 ) = 210. * X / ( 767 )
         VPOS ( 2 ) = 275. * Y /  ( 1023  )
         VPOS ( 3 ) = 1
         DO 20, I= 1, 3
          DO 15, J = 1, 3
           TEMP ( I, J ) = OTR ( I, J )
   15     CONTINUE
   20    CONTINUE
         CALL MATINV ( TEMP )
         RETURN
         END



         SUBROUTINE ARCTOL(T)
         REAL T,TOL
         LOGICAL FLAG,LOGIC

         COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
         IF(T.EQ.0.0) THEN
          TOL = 0.05
          LOGIC = .FALSE.
         ELSE
          TOL = T
         END IF
         RETURN
         END


         SUBROUTINE ARCINC(N)
         INTEGER N,INC
         LOGICAL FLAG,LOGIC

         COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
         IF(N.EQ.0 ) THEN
          INC = 0
          LOGIC = .FALSE.
         ELSE
          INC = N
          LOGIC = .TRUE.
         END IF
         RETURN
         END


         SUBROUTINE ARCENQ(I,NINCS,TOL)
         REAL TOL
         INTEGER I,NINCS
         LOGICAL FLAG,LOGIC
```

```
      COMMON/ENQY/FLAG,R,ATOL,INC,LOGIC,ITYPE

      I = ITYPE
      NINCS = INC
      TOL = ATOL
      RETURN
      END


      SUBROUTINE IRCBY2(DXC,DYC,DXE,DYE,ISENSE)
      REAL DXC,DYC,DXE,DYE,XC,YC,XE,YE
      INTEGER ISENSE
      LOGICAL FLAG,LOGIC
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      XC = VPOS(1) + DXC
      YC = VPOS(2) + DYC
      XE = VPOS(1) + DXE
      YE = VPOS(2) + DYE
      CALL IRCTO2(XC,YC,XE,YE,ISENSE)
      RETURN
      END


      SUBROUTINE IRCTO2(XC,YC,XE,YE,ISENSE)
      REAL XC,YC,XE,YE,AX,AY,AL,X,Y,RAD
      INTEGER ISENSE
      LOGICAL FLAG,LOGIC
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      RAD = SQRT( (XC-VPOS(1))**2 + (YC-VPOS(2))**2 )
      AX = XE - XC
      AY = YE - YC
      AL = SQRT( AX**2 + AY**2 )
      X  = RAD * AX/AL
      Y  = RAD * AY/AL
      CALL MOVTO2(X,Y)
      RETURN
      END



      SUBROUTINE POLBY2(DXARR,DYARR,NPTS)
      REAL DXARR(NPTS),DYARR(NPTS)
      INTEGER I
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      DO 10 I = 1,NPTS
       CALL LINBY2(DXARR(I),DYARR(I))
10    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE POLTO2(XARR,YARR,NPTS)
      REAL XARR(NPTS),YARR(NPTS)
      INTEGER I
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      DO 10 I = 1,NPTS
       CALL LINTO2(XARR(I),YARR(I))
10    CONTINUE
      RETURN
      END



      SUBROUTINE WINDO2(X1,X2,Y1,Y2)
      REAL X1,X2,Y1,Y2
      INTEGER IW
      LOGICAL FLAG,LOGIC
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      XMIN = X1
      XMAX = X2
      YMIN = Y1
      YMAX = Y2
      RETURN
      END



      SUBROUTINE WIND(I)
       EXTERNAL / PASCAL / LINEXX
      INTEGER IW,M(4),N(4)
      LOGICAL FLAG,LOGIC
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      IW = I
      IF(IW .EQ. 2) THEN
       XMIN = 0.0
       XMAX = 210.0
       YMIN = 0.0
       YMAX = 275.0
      END IF
      IF(IW.EQ.2 .OR. IW.EQ.1 ) THEN
       M(1) = NINT(767/210.0*XMIN)
       N(1) = NINT(1023 - (1023/275.0*YMIN))
       M(2) = NINT(767/210.0*XMAX)
       N(2) = N(1)
       M(3) = M(2)
       N(3) = NINT(1023 - (1023/275.0*YMAX))
       M(4) = M(1)
```

```
      N(4) = N(3)
       DO 10 I = 1,3
        CALL LINEXX(M(I),N(I),M(I+1),N(I+1))
10      CONTINUE
       CALL LINEXX( M(4),N(4),M(1),N(1) )
      END IF
      RETURN
      END



      SUBROUTINE WCUTS(X,Y,U,V)
C     MAKES USE OF EQUATION OF STRAIGHT LINE :
C          X = X1 + DELTA * (X2-X1)
C          Y = Y1 + DELTA * (Y2-Y1)
C     TO FIND WHERE INTEDED LINE CUTS THE WINDOW.
C     FUNCTIONS DEFINED ARE : DXY AND FNXY
C
      REAL X(2),Y(2),U(2),V(2),A(2),B(2),DELTA,HOR,VER
      REAL XMIN,XMAX,TMIN,TMAX
      INTEGER I,J,K
      LOGICAL FLAG,LOGIC
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      DXY(XY1,XY2,XY) =    (XY - XY1)/(XY2 - XY1)
      FNXY(XY1,XY2,DELTA) = XY1 + DELTA * (XY2 - XY1)
       IF(X(1).GE.XMIN .AND. X(1).LE.XMAX  .AND.
     !  Y(1).GE.YMIN .AND. Y(1).LE.YMAX ) THEN
        IF(X(2).GE.XMIN .AND. X(2).LE.XMAX  .AND.
     !   Y(2).GE.YMIN .AND.Y(2).LE.YMAX ) THEN
         DO 3 I = 1,2
          U(I) = X(I)
          V(I) = Y(I)
3        CONTINUE
         RETURN
        END IF
       END IF
C      CASE WHERE ONE OR BOTH POINTS LIE OUTSIDE THE WINDOW.
      J = 1
      I = 1
      HOR = YMIN
      VER = XMIN
      DO 4 K = 1,2
       A(K) = -50.0
       B(K) = -50.0
4     CONTINUE
5      IF(Y(1).NE.Y(2)) THEN
        DELTA = DXY( Y(1),Y(2),HOR)
        TEMP = FNXY( X(1),X(2),DELTA)
         IF(DELTA.GE.0.0 .AND. DELTA.LE.1.0  .AND.
     !    TEMP.GE.XMIN .AND.TEMP.LE.XMAX) THEN
            A(J) = TEMP
            B(J) = HOR
            HOR  = YMAX
```

```
            J = 2
          ELSE
           HOR = YMAX
          END IF
        ELSE
         HOR = YMAX
        END IF
        IF( X(1).NE.X(2) ) THEN
         DELTA = DXY( X(1),X(2),VER )
         TEMP = FNXY( Y(1),Y(2),DELTA)
          IF(DELTA.GE.0.0 .AND. DELTA.LE.1.0  .AND.
     !     TEMP.GE.YMIN .AND.TEMP.LE.YMAX ) THEN
             A(J) = VER
             B(J) = TEMP
             VER  = XMAX
              J = 2
          ELSE
           VER = XMAX
          END IF
        ELSE
         VER = XMAX
        END IF
       I = I + 1
       IF(I.NE.3) GOTO 5
C      TEST FOR NUMBER OF INTERSECTIONS.
       IF( X(1).GT.XMIN .AND. X(1).LT.XMAX .AND.
     ! Y(1).GT.YMIN .AND. Y(1).LT.YMAX ) THEN
        IF( X(2).LT.XMIN .OR. X(2).GT.XMAX .OR.
     !   Y(2).LT.YMIN .OR. Y(2).GT.YMAX ) THEN
         U(1) = X(1)
         V(1) = Y(1)
         U(2) = A(1)
         V(2) = B(1)
         RETURN
        END IF
       ELSE IF( X(1).LT.XMIN .OR.X(1).GT.XMAX .OR.
     ! Y(1).LT.YMIN .OR. Y(1).GT.XMAX ) THEN
        IF( X(2).GT.XMIN .AND. X(2).LT.XMAX .AND.
     !   Y(2).GT.YMIN .AND.Y(2).LT.YMAX ) THEN
         U(1) = A(1)
         V(1) = B(1)
         U(2) = X(2)
         V(2) = Y(2)
         RETURN
        END IF
       END IF
C      HAVING GOT THIS FAR, CONTINUE TO THE END
       SMIN = MIN( A(1),A(2) )
       TMIN = MIN( B(1),B(2) )
       SMAX = MAX( A(1),A(2) )
       TMAX = MAX( B(1),B(2) )
        IF( X(2).GE.X(1) .AND. Y(2).GE.Y(1) .OR.
     !   X(2).LE.X(1) .AND. Y(2).LE.Y(1) ) THEN
         IF( X(2).GT.X(1) .OR. Y(2).GT.Y(1) ) THEN
          U(1) = SMIN
```

```
          V(1) = TMIN
          U(2) = SMAX
          V(2) = TMAX
         ELSE
          U(1) = SMAX
          V(1) = TMAX
          U(2) = SMIN
          V(2) = TMIN
         END IF
        ELSE IF( X(2).GT.X(1) ) THEN
          U(1) = SMIN
          V(1) = TMAX
          U(2) = SMAX
          V(2) = TMIN
        ELSE
          U(1) = SMAX
          V(1) = TMIN
          U(2) = SMIN
          V(2) = TMAX
      END IF
      RETURN
      END


      SUBROUTINE RFRESH(PTR)
C     DOES MATRIX MULTIPLICATION, C = OTR*PTR
      REAL PTR(3,3),C(3,3)
      INTEGER I,J,K
      LOGICAL FLAG,LOGIC
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      DO 20 I= 1,3
       DO 10 J = 1,3
        C(I,J) = 0.0
        DO 5 K = 1,3
         C(I,J) = C(I,J) + OTR(I,K) * PTR(K,J)
 5      CONTINUE
 10     CONTINUE
 20    CONTINUE
      DO 100 I = 1,3
       DO 90 J = 1,3
        OTR(I,J) = C(I,J)
 90     CONTINUE
 100   CONTINUE
      RETURN
      END



      SUBROUTINE MATINV(A)
C     FINDS INVERSE OF MATRIX PTR AND RETURNS IT IN RTP
      REAL A(3,3),B(3,3),VAP(3)
      LOGICAL FLAG,LOGIC
```

```
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

C      DEFINE INDENTITY MATRIX OF SAME DIM. AS A(3,3)
      DO 10 I = 1,3
       DO 5 J = 1,3
        B(I,J) = 0.0
 5      CONTINUE
       B(I,I) = 1.0
 10    CONTINUE
C
      DO 4 I = 1,3
      DO 2 K = 1,3
      IF (K.EQ.I) GO TO 2
      CONST = -A(K,I)/A(I,I)
      DO 1 J =1,3
      A(K,J) = A(K,J) + CONST*A(I,J)
      B(K,J) = B(K,J) + CONST*B(I,J)
      IF (J.EQ.I) A(K,J) = 0.0
   1  CONTINUE
   2  CONTINUE
      CONST = A(I,I)
      DO 3 J = 1,3
      A(I,J) = A(I,J)/CONST
   3  B(I,J) = B(I,J)/CONST
      A(I,I) = 1.0
   4  CONTINUE
C      INVERSE OF MATRIX A IS IN MATRIX B
C      NOW FIND POSITION(VPOS), W.R.T. NEWLY DEFINED AXIS
      CALL MMULT(B,VPOS,VAP)
      DO 100 I = 1,3
       VPOS(I) = VAP(I)
 100   CONTINUE
      RETURN
      END


      SUBROUTINE MMULT(OTR,VPOS,APOS)
      REAL OTR(3,3),VPOS(3),APOS(3)
      INTEGER I,J
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      DO 10 I = 1,3
       APOS(I) = 0.0
        DO 5 J = 1,3
         APOS(I) = APOS(I) + OTR(I,J) * VPOS(J)
 5       CONTINUE
 10    CONTINUE
      RETURN
      END



      SUBROUTINE WINENQ(IW,IB1,NB,BOUNDS)
```

```
      REAL BOUNDS(NB)
      INTEGER IW,IB1,I
      LOGICAL FLAG,LOGIC
      COMMON/PICC/PPOS(3),W(4),JW
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      IW = JW
      IF(IB1.EQ.0 .OR. NB.EQ.0 ) RETURN
      DO 5 I = 1,NB
       BOUNDS(I) = W(IB1)
       IB1 = IB1 + 1
       IF(IB1.GT.4) IB1 = 1
    5 CONTINUE
      RETURN
      END



      SUBROUTINE TURNA4
      LOGICAL FLAG,LOGIC
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      CALL SHIFT2(0.0,275.0)
      CALL ROTAT2(-90.0)
      PERM1 = XMIN
      PERM2 = XMAX
      XMIN  = YMIN
      XMAX  = YMAX
      YMIN  = PERM1
      YMAX  = PERM2
      RETURN
      END



      SUBROUTINE UNITS(XMILS)
C     DEFINES RATIO OF SCALES AND PLACES IT IN COMMON.
      REAL R,XMILS
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      R = XMILS
      CALL SCALE(R)
      RETURN
      END


      SUBROUTINE POSPIC(X,Y)
C     RETURNS CURRENT PEN POSITION W.R.T. THE PICTURE COORDINATES.
      REAL X,Y
      LOGICAL FLAG,LOGIC
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
```

```
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      X = PPOS(1) * R
      Y = PPOS(2) * R
      RETURN
      END


      SUBROUTINE POSSPA(X,Y,Z)
C     RETURNS CURRENT PEN POSITION W.R.T. MOST RECENTLY DEFINED
C     SPACE COORDINATES.
      REAL X,Y,Z
      LOGICAL FLAG,LOGIC
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE


      X = VPOS(1) * R
      Y = VPOS(2) * R
      RETURN
      END


      SUBROUTINE DEVEND
      REAL R
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      FLAG = .FALSE.
      RETURN
      END




      SUBROUTINE M2INV(A)
C     FINDS INVERSE OF MATRIX PTR AND RETURNS IT IN RTP
      REAL A(2,2),B(2,2),VAP(2)
      LOGICAL FLAG,LOGIC
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

C     DEFINE INDENTITY MATRIX OF SAME DIM. AS A(2,2)
      DO 10 I = 1,2
       DO 5 J = 1,2
        B(I,J) = 0.0
 5     CONTINUE
       B(I,I) = 1.0
 10    CONTINUE
C
      DO 4 I = 1,2
      DO 2 K = 1,2
      IF (K.EQ.I) GO TO 2
```

```
      CONST = -A(K,I)/A(I,I)
      DO 1 J =1,2
      A(K,J) = A(K,J) + CONST*A(I,J)
      B(K,J) = B(K,J) + CONST*B(I,J)
      IF (J.EQ.I) A(K,J) = 0.0
    1 CONTINUE
    2 CONTINUE
      CONST = A(I,I)
      DO 3 J = 1,2
      A(I,J) = A(I,J)/CONST
    3 B(I,J) = B(I,J)/CONST
      A(I,I) = 1.0
    4 CONTINUE
C     INVERSE OF MATRIX A IS IN MATRIX B
C     NOW FIND POSITION(VPOS), W.R.T. NEWLY DEFINED AXIS
      CALL MMULT2(B,VPOS,VAP)
      DO 100 I = 1,2
      VPOS(I) = VAP(I)
  100 CONTINUE
      RETURN
      END


      SUBROUTINE MMULT2(OTR,VPOS,APOS)
      REAL OTR(2,2),VPOS(2),APOS(2)
      INTEGER I,J
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      DO 10 I = 1,2
      APOS(I) = 0.0
        DO 5 J = 1,2
        APOS(I) = APOS(I) + OTR(I,J) * VPOS(J)
    5   CONTINUE
   10 CONTINUE
      RETURN
      END



      SUBROUTINE ARCBY2(DXC,DYC,DXE,DYE,ISENSE)
      REAL DXC,DYC,DXE,DYE,XC,YC,XE,YE
      INTEGER ISENSE
      LOGICAL FLAG,LOGIC
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

      XC = VPOS(1) + DXC
      YC = VPOS(2) + DYC
      XE = VPOS(1) + DXE
      YE = VPOS(2) + DYE
      CALL ARCTO2(XC,YC,XE,YE,ISENSE)
      VPOS(1) = XE - DXE
      VPOS(2) = YE - DYE
      RETURN
```

```
      END


      SUBROUTINE ARCTO2(XC,YC,XE,YE,ISENSE)
      REAL XC,YC,XE,YE,RAD,DX,DY,X,Y,THETA,ALPHA
      INTEGER ISENSE,N
      LOGICAL FLAG,LOGIC
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE

C     FIND RADIUS OF ARC
      RAD = SQRT( (XC-VPOS(1))**2 + (YC-VPOS(2))**2 )
      THETA = 2 * ACOS(1 - TOL/RAD/5.)
      L = 2 * (RAD -TOL) * TAN(THETA/2)
      DX = VPOS(1) - XC
      DY = VPOS(2) - YC
      N = 3.14159/THETA
      IF(ISENSE.EQ.1)THEN
      THETA = THETA
      ELSE
      THETA = -THETA
      ENDIF
      ALPHA = 0.0
      IF(YC.EQ.YE)THEN
      N = 2*N
      ELSE
      N = N
      ENDIF
      DO 10 I =1,N+1
      ALPHA = THETA + ALPHA
      X = DX*COS(ALPHA) - DY*SIN(ALPHA) + XC
      Y = DX*SIN(ALPHA) + DY*COS(ALPHA) + YC
      CALL LINTO2(X,Y)
   10 CONTINUE
      RETURN
      END


      SUBROUTINE WINDIN (I)
      EXTERNAL / PASCAL / WINDOW
      INTEGER I,J,K,L,M,N
      IF (I.EQ.1 )THEN
        CALL WINDOW (1,0,0,767,873,'          DRAWING OF FAULT TREES')
      ELSEIF (I.EQ.2)THEN
        CALL WINDOW (2,0,873,555,150,'     INPUT INFORMATION  ')
      ELSEIF (I.EQ.3) THEN
        CALL WINDOW (3,660,873,107,75,'COMMAND')
        PRINT *
        PRINT *,'  CLEAR'
      ELSEIF (I.EQ.4) THEN
        CALL WINDOW (4,660,948,107,75,'INPUT FROM')
        PRINT *
        PRINT *,'    KEYS'
```

```
ELSEIF (I.EQ.5)THEN
   CALL WINDOW (5,560,873,100,75,'COMMAND')
   PRINT *
   PRINT *,'  STOP'
END IF
RETURN
END


SUBROUTINE CHWIN (I)
EXTERNAL / PASCAL / CHNWIN
INTEGER I
CALL CHNWIN(I)
RETURN
END


SUBROUTINE CHAMOD
END


SUBROUTINE PICBEG(I)
END


SUBROUTINE PICEND
END

SUBROUTINE EVESET(I)
END

SUBROUTINE PICSEN(I,J)
END

SUBROUTINE EVENT(I)
END


SUBROUTINE EVEDEL(I)
END


SUBROUTINE HARCHA
END


SUBROUTINE CHASWI(I)
END


SUBROUTINE PENSEL(I,W,J)
END
```

```
      SUBROUTINE CHASIZ(W,H)
      END


      SUBROUTINE CHAANG(A)
      END

      SUBROUTINE T4010
      END

      SUBROUTINE CHAASC(I)

      CALL CHAHOL(CHAR(I))
      END


      SUBROUTINE CHAINT(I,J)
      END
```

```
module drawings;
 exports
 imports screen from screen;
 imports io_others from io_others;
 imports io_unit from io_unit;
 imports rs232baud from rs232baud;
 procedure getcur ( var x, y : long );
 procedure scrclr;
 procedure rs232rd(var str : string);
 procedure rsbaudwt;
 procedure rsbaudrd;
 procedure rs232wrt(var str : string);
 procedure line77;
 procedure linexx(var x1,y1,x2,y2 : long );
 procedure rdcurs( var i,x, y : long );
 procedure strtab;
 procedure trakcurs;
 procedure tabpush ( var down : long );
 procedure window (var windx, orgx, orgy, width, height : long ; var title : string );
 procedure setcur ( var x, y : long );
 procedure putchr ( var letter : string );
 procedure chnwin ( var windx : long );
 private
 procedure getcur;
 var
  sx, sy : integer;
 begin
  sreadcursor ( sx, sy );
  x := stretch ( sx );
  y := stretch ( sy );
 end;
 procedure scrclr;
 begin
  screenreset;
```

```
end;
procedure rs232rd;
var ch : char;
   error : integer;
begin
   repeat
   until iocread (9, ch ) = 1;
   str[1] := ch;
end;
procedure rsbaudwt;
begin
  setbaud ( '9600', false );
end;
procedure rsbaudrd;
begin
  setbaud ( '9600', true );
end;
procedure rs232wrt;
var ch : char;
begin
 ch := str [1];
 repeat
 until iocwrite (10,ch ) = 1;
end;
procedure chnwin;
begin
  changewindow ( shrink ( windx ) );
end;
procedure putchr;
var ch : char;
begin
  ch := letter [ 1 ];
  sputchr ( ch );
end;
procedure linexx ( var x1, y1, x2, y2 : long );
begin
  line ( drawline, shrink(x1), shrink(y1), shrink(x2), shrink(y2), sscreenp );
end;
procedure rdcurs;
var xx,yy : integer;
begin
  trakcurs;
  repeat until tabswitch;
  while tabswitch do
    begin
    ioreadtablet ( xx,yy );
    if tabyellow then i := 0;
    if tabwhite then i := 1;
    if tabblue then i := 2;
    if tabgreen then i := 3;
  end;
  x := stretch (xx);
  y := stretch (yy);
end;
procedure strtab;
```

```
begin
   iosetmodetablet ( reltablet );
end;
procedure trakcurs;
begin
   iocursormode ( trackcursor );
end;
procedure tabpush ( var down : long );
begin
   if tabswitch then down :=1 else down := 0;
end;
procedure line77;
var style : linestyle;
   x1,x2,y1,y2 : integer;
   p : rasterptr;
   orgx,orgy,width,height : integer;
   title : string;
   windx :winrange;
   letter : char;
   m : cursmode;
begin
  screenreset;
  m := trackcursor;
  iosetmodetablet(reltablet);
  iocursormode ( m );
  for windx := 1 to 2 do
begin
  orgx := 100*windx;
  orgy := 100*windx;
  width := 300;
  height := 500;
  title := 'window 1';
  letter := 'A';
  style := drawline;
  x1:= 10;
  x2 :=700;
  y1 := 20;
  y2 := 650;
  p := sscreenp;
  for x1 := 1 to 76 do
       line ( style, x1*10,20,x1*10,1000, p );
  for y1 := 1 to 100 do
       line ( style, 5, y1*10, 765, y1 * 10,  p );
  createwindow ( windx, orgx, orgy, width, height, title );
  for x1 := 1 to 100 do
     sputchr ( letter );
end;
repeat until tabswitch;
end;
procedure setcur ( var x, y : long );
begin
  ssetcursor ( shrink ( x ), shrink ( y ) );
end;
procedure window ;
begin
  createwindow(shrink(windx),shrink(orgx),shrink(orgy),shrink(width),
           shrink(height), title)
end.
```

APPENDIX I : LISTING OF THE SOURCE PROGRAMS OF THE HAZOP PNX VERSION

```
*********************************************************************
*  COMMENT : THIS PACKAGE IS WRITTEN FOR THE ICL-PERQ TO CARRY    *
*            OUT THE HAZARD AND OPERABILITY STUDY (HAZOP) AND GIVE *
*            ACCESS TO RELIABILITY CALCULATIONS ESPECIALLY WITH     *
*            REPEATED BASIC EVENTS                                  *
*********************************************************************


    INTEGER CLR,CLC, SMR, SMC, NML

    PARAMETER ( CLR = 500,
   +          CLC = 10,
   +          SMR = 50,
   +          SMC = 20,
   +          NML = 12)

    INTEGER CELLS ( CLR, CLC ), SYMPTS ( SMR, SMC )
    CHARACTER NAMES ( CLR ) * ( NML ), S*80


    OPEN (19,FILE='SCLEAR.W')
     S='WELCOME TO THE HAZOP PACKAGE RUNNING ON THE PERQ_ICL'
    CALL FORM1(S)

    DO 5 MM=1,4
    CALL GRFTR1(IFLAG,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
C
C**** SIMULATION OF EXECUTION OF INDIVIDUAL PARTS OF THE PACKAGE FROM
C**** GRAPHICAL MODE OF OPERATION BY MEANS OF A FLAG RETURNED FROM EACH
C**** SBR.
C
    IF(IFLAG.EQ.1)THEN

     S='CAUSE & SYMPTOM EQNS ANALYSED'
    ELSEIF(IFLAG.EQ.2)THEN

     S='FAULT TREES DRAWN'
    ELSEIF(IFLAG.EQ.3)THEN

     S='CAUSE & SYMPTOM EQNS TRANSLATED'
    ELSEIF(IFLAG.EQ.4)THEN

     S='CAUSE & SYMPTOM EQNS EDITED. '
    ELSEIF(IFLAG.EQ.5)THEN
     GO TO 5
    ENDIF
     CALL FORM1(S)
 5  CONTINUE
    CLOSE(19)
    END
```

```
C****   SUBROUTINE GRFTR1  HAS BEEN  PREPARED FOR  GRAPHICAL TERMINALS
C****   WITH A MOUSE ,IN  WHICH CASE THE USER IS PRESENTED WITH A MENU
C****   LIST OF THE PACKAGE ROUTINES.  SELECTION IS MADE BY MOVING THE
C****   CURSOR TO  THE  ITEM TO  BE  SELECTED  AND  PRESSING THE MOUSE
C****   BUTTON. THIS SUBROUTINE HAS MORE COMMON APPLICATION SINCE MOST
C****   MODERN TERMINALS HAVE GOT MOUSE-CURSOR FACILITIES.
C
      SUBROUTINE GRFTR1(IFLAG,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)

      INTEGER CLR,CLC,SMR,SMC,NML
      INTEGER CELLS ( CLR, CLC ), SYMPTS ( SMR, SMC )
      INTEGER ICOM
      REAL X, Y
      CHARACTER NAMES ( CLR ) * ( 12 )

      CHARACTER CHS(5)*45 ,S*300
      LOGICAL TEST,CHECK,TEST1,FILE

      DATA (CHS(I),I=1,5)/'  A. ANALYSE','  B. DRAW','  C. EDIT','  E. A
     1BOUT CAUSE & SYMPTOM EQNS.','  F. QUIT'/
      DATA TEST, CHECK / 2 * .FALSE. /

      CALL PERQ
      CALL PICCLE
      CALL UNITS(5.)
      CALL SHIFT2(10.,25.)
      CALL MOVTO2(0.,5.)
      CALL CHAHOL('SELECT YOUR PACKAGE ROUTINE BY MOVING THE CURSOR')
      CALL MOVTO2(0.,4.)
      CALL CHAHOL('TO THE ITEM OF THE MENU LIST AND PRESS THE MOUSE')
      CALL MOVTO2(0.,3.)
      CALL CHAHOL('BUTTON(S)')
      CALL MOVTO2(0.,0.)
      CALL CHAHOL('MENU LIST OF PACKAGE ROUTINES')

C
C****       SET UP MENU LIST
C

      DO 10 I=2,12,2
      X=I
      CALL MOVTO2(0.,-X)
      KK=I/2
      CALL AKCHAR(CHS(KK))
 10   CONTINUE
C
C****     MAKE THE CURSOR APPEAR
C
      CALL CURSOR(ICOM,X,Y)
C
C****    MOVE THE CURSOR TO THE ITEM OF THE MENU LIST TO BE SELECTED
C****    AND PRESS THE MOUSE BUTTON
C
C****     IDENTIFY ITEM SELECTED BY ANALYSING PICTURE SEGMENT
```

```
C****    COORDINATE Y.
C

    IF(Y.GT.23.0.AND.Y.LT.24.0)THEN
    TEST1=TEST.AND.CHECK
    IF(TEST1)THEN

      S='WARNING CAUSE & SYMPTOM EQNS HAVE ALREADY BEEN ANALYSED.
     +      DATA FROM ANALYSIS OF CAUSE & SYMPTOM EQNS HAVE BEEN STO
     +RED IN DATAFILE OPXXXXXX.'
    CALL FORM1 (S)

    IFLAG=5
    RETURN
    ELSE IF ( TEST ) THEN

     S='THE ANALYSED CAUSE AND SYMPTOM EQUATIONS HAVE ALREADY BEEN
     +READ IN FROM DATAFILE OPXXXXXX.'
     CALL FORM1 (S)

    ELSE
    CHECK=FILE(I)
C**** CLEAR THE GRAPHICAL SCREEN AND REOPEN IT FOR DRAWING FAULT TREES
    CALL CCLOSE
    CALL EQTS(IFLAG,CHECK,TEST,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
    ENDIF
    ELSEIF(Y.GT.21.0.AND.Y.LT.22.0)THEN
    CALL TREES(IFLAG,TEST,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
    ELSEIF(Y.GT.19.0.AND.Y.LT.20.0)THEN
    CALL EDIT(IFLAG,TEST)
    ELSEIF(Y.GT.17.0.AND.Y.LT.18.0)THEN
    CALL CCLOSE
    CALL DEXPLN
    IFLAG=5
    ELSEIF(Y.GT.15.0.AND.Y.LT.16.0)THEN
    STOP
    ENDIF
    CALL DEVEND
    RETURN
    END



C****    SUBROUTINE EQTS  IS USED TO SIMULATE THE ANALYSIS OF
C****    CAUSE & SYMPTOM EQNS AND MAY EASILY FACILITATE THE
C****    PROGRAM TREES WHICH ANALYSES CAUSE & SYMPTOM EQNS.
C


    SUBROUTINE EQTS(IFLAG,CHECK,TEST,CELLS,CLR,CLC,SYMPTS,SMR,SMC,
    *           NAMES,NML)

    INTEGER CLR,CLC,SMR,SMC,NML
    LOGICAL TEST,CHECK
    INTEGER CELLS ( CLR, CLC ), SYMPTS ( SMR, SMC )
```

```
      CHARACTER NAMES ( CLR ) * ( 12 )

      CALL PART1(CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
      TEST = .TRUE.

C
C****    CHECK IF THE USER HAS REQUESTED TO STORE DATA FROM ANALYSIS
C****    OF CAUSE & SYMPTOM EQNS INTO A FILE.
C
      IF(CHECK)THEN
      OPEN(3,FILE='OPXXXXXX',FORM='UNFORMATTED',
     * ACCESS = 'SEQUENTIAL')
      REWIND (3)
      DO 11 I = 1, CLR
      WRITE(3)(CELLS(I,J),J=1,CLC)
 11   CONTINUE
      DO 12 K = 1,SMR
      WRITE(3)(SYMPTS(K,L),L=1,SMC)
 12   CONTINUE
      DO 13 M = 1,CLR
      WRITE(3)NAMES(M)
 13   CONTINUE
      CLOSE(3)
      ENDIF
      IFLAG=1
      END


      SUBROUTINE FORM1 (S)

      CHARACTER S*(*), KEY

      OPEN (1,FILE='INTROD.W')
      OPEN (2,FILE='INTROD.W')
      WRITE (1,FMT=*)'************************************************
     +****************************'
      WRITE (1,FMT=*)S
      WRITE (1,FMT=*)'************************************************
     +****************************'
      WRITE (1,FMT=*)
      WRITE (1,FMT=*)'  PLEASE, PRESS RETURN KEY TO CONTINUE'
 11   READ (2,'(A)')KEY
      IF (KEY .EQ. CHAR(32)) THEN
      CLOSE (1)
      CLOSE (2)
      ELSE
      GO TO 11
      END IF
      RETURN
      END


      SUBROUTINE FORM1A (S1,S2)

      CHARACTER S1*(*),S2*(*), KEY
```

```
      OPEN (1,FILE='INTROD.W')
      OPEN (2,FILE='INTROD.W')
       WRITE (1,FMT=*)S1
       WRITE (1,FMT=*)S2
       WRITE (1,FMT=*)'  PLEASE, PRESS RETURN KEY TO CONTINUE'
 11    READ (2,'(A)')KEY
      IF (KEY .EQ. CHAR(32)) THEN
      CLOSE (1)
      CLOSE (2)
      ELSE
      GO TO 11
      END IF
      RETURN
      END

      SUBROUTINE FORM2 (S,YESNO)

      CHARACTER S*160, ANSWER*3
      LOGICAL YESNO, REPLY

      OPEN (3,FILE='INTROD.W')
      OPEN (4,FILE='INTROD.W')
9000 CONTINUE
       WRITE (3,FMT=*)S
      READ (4,16)ANSWER
 16  FORMAT(A)
      IF(ANSWER.EQ.'YES'.OR.ANSWER.EQ.'YE'.OR.ANSWER.EQ.'Y'.OR.ANSWER.
     +EQ.'YES'.OR.ANSWER.EQ.'YE'.OR.ANSWER.EQ.'Y')THEN
      REPLY=.TRUE.
      YESNO=.TRUE.
      ELSEIF(ANSWER.EQ.'NO'.OR.ANSWER.EQ.'N'.OR.ANSWER.EQ.'NO'.OR.
     +ANSWER.EQ.'N')THEN
      REPLY=.TRUE.
      YESNO=.FALSE.
      ELSE
      S='YOUR ANSWER SHOULD BE YES OR NO [Y/N]'
      REPLY=.FALSE.
      ENDIF
      IF ( .NOT. REPLY) GO TO 9000
      CLOSE(3)
      CLOSE(4)
      RETURN
      END



C
 C****   SUBROUTINE EDIT SIMULATES THE EDITING OF CAUSE & SYMPTOM
 C****   EQNS AND ONCE MORE THE DATA STRUCTURE FOR THIS SBR. COMES
 C****   FROM THE ANALYSIS OF CAUSE & SYMPTOM EQNS. THE SBR. MAY
 C****   FACILITATE THE NEWLY PREPARED  PROGRAM FOR EDITING.
 C
```

```
      SUBROUTINE EDIT(IFLAG,TEST)
      LOGICAL TEST
      CHARACTER DEQTS*90
C
C****    TEST WHETHER DATA OF ANALYSIS OF C&S EQNS IS TO BE READ FROM
C****    A FILE OR TRANSFERED AS AN ARGUMENT OF THE SBR.
C
      IF(TEST)THEN
      OPEN(3,FILE='OPXXXXXX',FORM='UNFORMATTED',
     *  ACCESS='SEQUENTIAL')
      REWIND(3)
      READ(3)DEQTS
      CLOSE(3)
      ENDIF
      IFLAG=4
      END

C
C****    SUBROUTINE DEXPLN OUTPUTS A DETAILED ILLUSTRATED EXPLANATION
C****    OF THE PACKAGE
C
      SUBROUTINE DEXPLN
      CHARACTER LINE*80,REP*4

      OPEN(UNIT=15,FILE='HELP.W')
      OPEN(UNIT=16,FILE='HELP.W')
      OPEN(UNIT=9,FILE='ZXC.TEXT')
      REWIND(9)
C
C****    OUTPUT THE TEXT OF EXPLANATION FROM FILE ZXC TO THE SCREEN.
C
 10   J=1
 20   READ(9,'(A)')LINE
      WRITE(UNIT=16,FMT='(A)')LINE
      IF (LINE.NE.' @@@')THEN
       J=J+1
C
C****    OUTPUT EXPLANATION OF PACKAGE AT INTERVALS OF 65 LINES
C
       IF (J.EQ.65)THEN
         WRITE(16,FMT=*)'PRESS RETURN TO CONTINUE'
        READ(UNIT=15,FMT='(A)')REP
        IF(REP.NE.CHAR(32))THEN
         CLOSE(9)
         CLOSE(16)
         CLOSE(15)
         RETURN
        ELSE
         GO TO 10
        END IF
       END IF
       GO TO 20
      ELSE
      WRITE(16,FMT=*)'PRESS ANY KEY TO QUIT'
      READ(15,FMT='(A)')REP
```

```
      END IF
      CLOSE(9)
      CLOSE(16)
      CLOSE(15)
      RETURN
      END



C
C****    SUBROUTINE TREES SIMULATES THE DRAWING OF FAULT TREES
C****    FROM CAUSE & SYMPTOM EQNS ON A GRAPHICAL TERMINAL
C****    AND CAN EASILY FACILITATE THE PROGRAM TREES FOR THE
C****    ACTUAL DRAWING OF FAULT TREES.
C
      SUBROUTINE TREES(IFLAG,TEST,CELLS,CLR,CLC,SYMPTS,SMR,SMC,
     *            NAMES,NML)

      INTEGER CLR,CLC,SMR,SMC,NML,IFLAG
      INTEGER CELLS ( CLR, CLC ), SYMPTS ( SMR, SMC )
      CHARACTER NAMES ( CLR ) * ( 12 )
      LOGICAL TEST
C
C****    TEST WHETHER DATA FROM ANALYSIS OF CAUSE & SYMPTOM EQNS
C****    ARE GOING TO BE READ FROM A FILE OR TO BE TRANSFERED AS
C****    ARGUMENTS FROM THE SUBROUTINE.
C
      OPEN(UNIT=18,FILE='SYEND.W')
      WRITE(18,FMT=*)'          READING DATA FROM ANALYSIS OF'
      WRITE(18,FMT=*)'          CAUSE & SYMPTOM EQNS'
      CALL FLASH(18)
      IF(.NOT. TEST)THEN
      OPEN(3,FILE='OPXXXXXX',FORM = 'UNFORMATTED',
     * ACCESS = 'SEQUENTIAL')
      REWIND(3)
      DO 11 I = 1, CLR
      READ(3)(CELLS(I,J),J=1,CLC)
   11 CONTINUE
      DO 12 K = 1,SMR
      READ(3)(SYMPTS(K,L),L=1,SMC)
   12 CONTINUE
      DO 13 M = 1, CLR
      READ(3) NAMES(M)
   13 CONTINUE
      CLOSE(3)
      CALL ASYR
      ENDIF
      CLOSE(18)
      CALL PICCLE

      CALL PART2(CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
C
      IFLAG=2
      END
```

```
C
C****    SUBROUTINE AKCHAR PREPARES CHARACTER STRINGS TO BE READILY
C****    USED WITH THE GINO COMMAND CALL CHAARR(L,M,N) WHICH OUTPUTS
C****    A CHARACTER STRING.
C
      SUBROUTINE AKCHAR(STRING)
      PARAMETER(LENGTH=20)
      CHARACTER STRING*(*)
      INTEGER FINAL
      NCHAR=FINAL(STRING)
C
C****    OUTPUT STRING OF CHARACTERS
C
      CALL CHAHOL(STRING(1:FINAL(STRING)))
      END


      LOGICAL FUNCTION FILE(I)
      LOGICAL RPLY
      CHARACTER S*400

      S='DO YOU INTEND TO TERMINATE RUN BEFORE OBTAINING FINAL RESULTS?
     1              IF [Y] INTERMEDIATE RESULTS FROM ANALYSIS O
     2F CAUSE AND SYMPTOM              EQNS MAY BE STORED IN DAT
     3A FILE AND USED ON REQUEST '
      CALL FORM2 (S,RPLY)
      IF(RPLY)THEN
      FILE=.TRUE.
      ELSE
      FILE=.FALSE.
      ENDIF
      RETURN
      END

C
C*******
C
      SUBROUTINE PART1(CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)

      INTEGER CLR,CLC, SMR, SMC, NML
      INTEGER AT,OP,PO,EQ,AX,TP,NAM,SYMP,SIGN,I,Q,N,POS,M,K,P,S,R,J
      INTEGER LIMT,FIM,TNML
      PARAMETER (AT=30,OP=20,PO=30,EQ=200,AX=30,TP=9,TNML=12)
      INTEGER CELLS ( CLR, CLC ), SYMPTS ( SMR, SMC )
      INTEGER DIST,TEST,ATRNS(AT),OPSTK(OP),POLLSH(PO),AUX(AX),TEMP(TP)
      CHARACTER NAMES(CLR)*(12),BRANCH*(TNML)
      CHARACTER EQN*(EQ), DNAME*8,SS*50,SA*(EQ)
      LOGICAL SAY2

*
*     'INITIALISING ARRAYS'
*
      DO 10 I=1,CLR
        NAMES(I)='EMPTY'
   10 CONTINUE
```

```
      DO 20 I=1,SMR
       DO 21 J=1,SMC
        SYMPTS(I,J)=0
   21  CONTINUE
   20 CONTINUE

      DO 22 I=1,CLR
       DO 23 J=1,CLC
        CELLS(I,J)=0
   23  CONTINUE
   22  CONTINUE

      OPEN (13,FILE='INTROD.W')
      OPEN (17,FILE='INTROD.W')
      WRITE (17,FMT=*)'INPUT THE NAME OF THE CAUSE & SYMPTOMS EQNS. FILE
     +'
      READ(13,'(A)')DNAME
      SS='DO YOU WANT TO SEE A DISPLAY OF THE CAUSE & SYMPTOM EQNS. WITH
     + THEIR ANALSIS?'
      CALL FORM2(SS,SAY2)
      CLOSE (17)
      CLOSE (13)
      IF (.NOT.SAY2)THEN
      OPEN(UNIT=16,FILE='SYEND.W')
      WRITE(16,FMT=*)'    WAITING FOR THE ANALYSIS TO BE FINISHED'
      CALL FLASH(16)
      END IF
      OPEN(8,FILE=DNAME,FORM='FORMATTED')
      REWIND(8)
  100 READ( UNIT = 8, FMT = * ) EQN
      Q=0
      N=0
      IF(EQN.EQ.'END')THEN
       GO TO 2000
      END IF

      CALL DISTNT (EQN,DIST,Q )
      POS=Q-1
      BRANCH=EQN(1:POS)
      CALL ALLOC(BRANCH,R,TEST,NAMES,NML,CLR)
      NAM=R
      IF(DIST.EQ.0)THEN
       GO TO 350
      END IF
      IF(TEST.EQ.-1)THEN
       CELLS(NAM,1)=1
       CELLS(NAM,2)=-2
      END IF
      M=0
   50 M=M+1
      IF(M.GT.SMR)THEN
      SS= 'SYSTEM CANNOT ACCEPT MORE EQS.'
      CALL FORM1(SS)
       GO TO 300
      END IF
```

```
      IF(SYMPTS(M,1).NE.0)THEN
       GO TO 50
      END IF
      SYMPTS(M,1)=NAM
      SYMP=M
      K=1
   60 P=Q+1
      S=0
      SIGN=0
   70 Q=Q+1
      S=S+1
      IF(S.GT.NML+1)THEN
       SS='THERE IS A MISTAKE,IT IS NOT A SYMPTON EQN. INPUT EQUATION
     +WAS'
       SA=EQN
       CALL FORM1A(SS, SA)
            GO TO 300
      END IF
      IF(EQN(Q:Q).EQ.'*')THEN
       BRANCH=EQN(P:Q-1)
      ELSE
       IF(EQN(Q+1:Q+1).EQ.' ')THEN
        BRANCH=EQN(P:Q)
        SIGN=-1
       ELSE
        GO TO 70
       END IF
      END IF
      CALL ALLOC(BRANCH,R,TEST,NAMES,NML,CLR)
      IF(TEST.EQ.-1)THEN
       CELLS(R,1)=1
       CELLS(R,2)=0
       CELLS(R,3)=1
      ELSE
       CELLS(R,3)=CELLS(R,3)+1
      END IF
      J=CELLS(R,3)
      CELLS(R,3+J)=SYMP
      SYMPTS(SYMP,2+K)=R
      K=K+1
      IF(K.GT.SMC-2)THEN
       SS='THERE ARE MORE THAN 18 BRANCHS IN A SYMPTON EQN.'
       SA=EQN
       CALL FORM1A(SS, SA)
       GO TO 300
      END IF
      IF(SIGN.EQ.0)THEN
       GO TO 60
      END IF
      SYMPTS(SYMP,2)=K-1
      GO TO 100

  350 IF(TEST.EQ.-1)THEN
       CELLS(NAM,1)=1
      END IF
```

```
      CALL TRNSLT(EQN,Q,ATRNS,I,LIMT,EQ,AT,CELLS,CLR,CLC,NAMES,NML)

      CALL POLISH(ATRNS,I,POLLSH,N,LIMT,FIM,OP,PO,AT,OPSTK,AUX)

      CALL CAUSE(POLLSH,NAM,FIM,PO,TP,CELLS,CLR,CLC,NAMES,NML,TEMP)

      GO TO 100
2000  CLOSE(8)
      CLOSE(16)

      DO 1500 J=1,CLR
       IF(CELLS(J,3).EQ.0)THEN
        IF(NAMES(J).NE.'EMPTY')THEN
         CELLS(J,2)=-2
         CELLS(J,1)=1
        ELSE
         GO TO 200
        END IF
       END IF
1500 CONTINUE
 200  IF (SAY2)THEN
      OPEN(14 ,FILE = 'HELP.W')
      OPEN(18,FILE=DNAME,FORM='FORMATTED')
      REWIND(18)
      WRITE(14,*)
      WRITE (14,*)'******* ',DNAME,' *******'
      WRITE(14,*)'============================'
      WRITE(14,*)
111   READ( UNIT = 18, FMT = * ) EQN
      WRITE ( UNIT = 14, FMT = '(2A)' ) 'EQN  :  ',EQN
      IF(EQN.NE.'END')THEN
       GO TO 111
      END IF
      CLOSE(18)
      WRITE ( UNIT = 14, FMT =*)
      WRITE ( UNIT = 14, FMT = 205 )
205 FORMAT (T42,'C E L L S'/T42,9('*'))
      WRITE ( UNIT = 14, FMT = * )
      DO 210 I=1,CLR
       WRITE ( UNIT = 14, FMT = 220 ) I, (CELLS ( I,J ), J = 1, CLC )
220   FORMAT (T3,I3,T20,10(3X,I3)/)
210  CONTINUE


      WRITE ( UNIT = 14, FMT = * )
      WRITE ( UNIT = 14, FMT = 215 )
215  FORMAT (T42,'S Y M P T S'/T42,11('*'))
      WRITE ( UNIT = 14, FMT = * )
      DO 230 I=1,SMR
       WRITE ( UNIT = 14, FMT = 240 )I,(SYMPTS(I,J),J=1,SMC)
240   FORMAT (T3,I3,T6,(10(3X,I3)/))
230  CONTINUE
```

```
      WRITE ( UNIT = 14, FMT = * )
      WRITE ( UNIT = 14, FMT = 265 )
  265  FORMAT (T10,'N A M E S'/T10,9('*'))
      WRITE ( UNIT = 14, FMT = * )
      DO 250 I=1,CLR
       WRITE ( UNIT = 14, FMT = 260 ) I,NAMES(I)
  260  FORMAT (T5,I4,T10,A)
  250 CONTINUE
      CALL FLASH(14)
      CLOSE(14)
      END IF
      OPEN(7, FILE = 'LOCAL' ,FORM = 'FORMATTED')
      REWIND(7)
      DO 400 L=1,CLR
        WRITE(UNIT=7,FMT=420)(CELLS(L,K),K=1,CLC)
  420   FORMAT(10(I3,3X))
  400 CONTINUE
      DO 450 L=1,SMR
        WRITE(UNIT=7,FMT=440)(SYMPTS(L,K),K=1,SMC)
  440   FORMAT(10(I3,3X))
  450 CONTINUE
  300 CONTINUE
      CLOSE (7)
      RETURN
      END


 C
 C*******
 C
      SUBROUTINE DISTNT(EQN,DIST,Q )

 C*** *  THIS SUBROUTINE DETERMINES WHETHER AN EQUATION IS A CAUSE OR
 C*** *  A SYMPTOM EQUATION.
 C*** *
 C*** *   EQN   - CHARACTER - THE EQUATION IN QUESTION
 C*** *   DIST  - INTEGER   - RETURNS -1 FOR SYMPTOM
 C*** *                       RETURNS 0 FOR CAUSE
 C*** *   Q     - INTEGER   - RETURNS POSITION OF [-] OR [=] IN EQN
 C*** *

      CHARACTER EQN * ( * )
      INTEGER DIST, Q, NXTNSP

      Q = INDEX ( EQN, ')' )

      IF ( Q .EQ. 0 ) THEN

        CALL INERR ( EQN )
        STOP

      ELSE

 *
 *      LOOK FOR NEXT NON-SPACE CHARACTER
```

```
      Q = Q + NXTNSP ( EQN ( Q + 1 : LEN ( EQN ) ) )

      IF ( EQN ( Q : Q ) .EQ. '-' ) THEN

        DIST = -1
        RETURN

      ELSE IF ( EQN ( Q : Q ) .EQ. '=' ) THEN

        DIST = 0
        RETURN

      ELSE

        CALL INERR ( EQN )
        STOP

      END IF

    END IF
  END


  SUBROUTINE ALLOC(BRANCH,R,TEST,NAMES,NML,CLR)

  INTEGER CLR,NML
  INTEGER R, TEST, NEXT
  PARAMETER ( TNML = 12 )
  CHARACTER BRANCH*(*),NAMES(CLR)*(12)

  DATA NEXT / 1 /

  IF ( BRANCH .NE. ' ' ) THEN

    DO 10, R = 1 , NEXT

      IF ( BRANCH .EQ. NAMES ( R ) ) THEN

        TEST = 0
        RETURN

      END IF

10    CONTINUE

  END IF

  IF ( NEXT .EQ. CLR ) THEN

    PRINT *, 'SYSTEM CANNOT ACCEPT MORE EQUATIONS '
    STOP

  ELSE

    NAMES ( NEXT ) = BRANCH
```

```
      R = NEXT
      NEXT = NEXT + 1
      TEST = -1

      RETURN

    END IF

    END

C
C****
C
    SUBROUTINE TRNSLT(EQN,Q,ATRNS,I,LIMT,EQ,AT,CELLS,CLR,CLC,
   *              NAMES,NML)

    INTEGER EQ,AT,CLR,CLC,NML
    INTEGER Q,ATRNS(AT),I,LIMT,PS,MEM,R,TEST,CELLS(CLR,CLC)
    CHARACTER BRANCH*(12),EQN*(*),NAMES(CLR)*(12)
    DO 51 I=1,AT
     ATRNS(I)=0
51  CONTINUE
    I=1
    PS=Q+1
55  Q=Q+1
    IF(EQN(Q:Q).EQ.'(')THEN
     IF(EQN(Q+1:Q+1).NE.'(')THEN
      IF(Q.EQ.PS)THEN
       ATRNS(I)=-4
       I=I+1
       PS=PS+1
       GO TO 55
      ELSE
       MEM=INDEX(EQN(PS:  ),')')
       BRANCH=EQN(PS:PS+MEM-1)
        CALL ALLOC(BRANCH,R,TEST,NAMES,NML,CLR)
       IF(TEST.EQ.-1)THEN
        CELLS(R,1)=1
       END IF
       ATRNS(I)=R
       I=I+1
       PS=PS+MEM
       Q=PS-1
       GO TO 55
      END IF
     ELSE
      ATRNS(I)=-4
      I=I+1
      PS=PS+1
      GO TO 55
     END IF
    END IF
    IF(EQN(Q:Q).EQ.')')THEN
     ATRNS(I)=-3
     I=I+1
```

```
      PS=Q+1
      GO TO 55
     END IF
     IF(EQN(Q:Q).EQ.'*')THEN
      ATRNS(I)=-1
      I=I+1
      PS=Q+1
      GO TO 55
     END IF
     IF(EQN(Q:Q).EQ.'+')THEN
      ATRNS(I)=-2
      I=I+1
      PS=Q+1
      GO TO 55
     END IF
     IF(EQN(Q:Q).NE.' ')THEN
      GO TO 55
     ELSE
      LIMT=I-1
     END IF
     RETURN
     END

C
C*****
C
     SUBROUTINE POLISH(ATRNS,I,POLLSH,N,LIMT,FIM,OP,PO,AT,OPSTK,AUX)

     INTEGER OP,PO,AT,FIM,N,I,L,J,LIMT,X
     INTEGER POLLSH(PO),ATRNS(AT),OPSTK(OP),AUX(OP)

     I=0
     N=0
     J=0
     X=0
500  I=I+1
     IF(I.GT.LIMT)THEN
      IF(J.NE.0)THEN
       DO 555 L=J,1,-1
        N=N+1
        POLLSH(N)=OPSTK(L)
555    CONTINUE
      END IF
      FIM=N
      RETURN
     END IF
     IF(ATRNS(I).LT.0)THEN
      IF(J.EQ.0)THEN
       J=J+1
       OPSTK(J)=ATRNS(I)
       GO TO 500
      END IF
      X=X+1
      AUX(X)=ATRNS(I)
      IF(AUX(X).GT.OPSTK(J))THEN
```

```
      J=J+1
      OPSTK(J)=AUX(X)
      AUX(X)=0
      X=X-1
      GO TO 560
     ELSE
      IF(AUX(X).NE.OPSTK(J))THEN
       IF(AUX(X).EQ.-4)THEN
        J=J+1
        OPSTK(J)=AUX(X)
        AUX(X)=0
        X=X-1
        GO TO 500
       END IF
       IF(AUX(X).EQ.-3)THEN
570      N=N+1
        POLLSH(N)=OPSTK(J)
        OPSTK(J)=0
        J=J-1
        IF(OPSTK(J).NE.-4)THEN
         GO TO 570
        ELSE
         J=J+1
         OPSTK(J)=AUX(X)
         AUX(X)=0
         X=X-1
         GO TO 560
        END IF
       END IF
580     N=N+1
       POLLSH(N)=OPSTK(J)
       OPSTK(J)=0
       J=J-1
       IF(J.GT.0)THEN
        IF(OPSTK(J).EQ.POLLSH(N))THEN
         GO TO 580
        END IF
       END IF
       J=J+1
       OPSTK(J)=AUX(X)
       AUX(X)=0
       X=X-1
       GO TO 500
      ELSE
       IF(AUX(X).GT.-3)THEN
        DO 590 L=I+1,LIMT
         IF(ATRNS(L).LT.0)THEN
          IF(ATRNS(L).NE.-3)THEN
           IF(ATRNS(L).NE.AUX(X))THEN
            J=J+1
            OPSTK(J)=AUX(X)
            AUX(X)=0
            X=X-1
            GO TO 500
           END IF
```

```
            END IF
           END IF
590         CONTINUE
           N=N+1
           POLLSH(N)=AUX(X)
           AUX(X)=0
           X=X-1
           GO TO 500
          END IF
         END IF
        END IF
560   IF(OPSTK(J)+OPSTK(J-1).EQ.-7)THEN
        OPSTK(J)=0
        OPSTK(J-1)=0
        J=J-2
       END IF
       GO TO 500
      ELSE
       N=N+1
       POLLSH(N)=ATRNS(I)
       GO TO 500
      END IF
      END

C
C****
C

      SUBROUTINE CAUSE(POLLSH,NAM,F1M,PO,TP,CELLS,CLR,CLC,
     *            NAMES,NML,TEMP)

      INTEGER CLR,CLC,NML,TP,PO
      CHARACTER NAMES(CLR)*(12)
      INTEGER CELLS(CLR,CLC),POLLSH(PO),TEMP(TP),FLAG,STEP,F1M,NAM
     +,LASTOP,L,K,S,M,I, TEST
      DO 700 L=1,TP
       TEMP(L)=0
700   CONTINUE
      K=0
      FLAG=1
710   K=K+1
      IF(POLLSH(K).GE.0)THEN
       IF(K.GE.F1M)THEN
        IF(FLAG.EQ.1)THEN
         CELLS(NAM,2)=2
         CELLS(NAM,3)=1
         CELLS(NAM,4)=POLLSH(K)
         RETURN
        ELSE
         DO 720 S=1,TP
          CELLS(NAM,S+1)=TEMP(S)
720       CONTINUE
         RETURN
        END IF
       ELSE
```

```
      GO TO 710
     END IF
    END IF
    IF(FLAG.EQ.1)THEN
     LASTOP=POLLSH(K)
     IF(LASTOP.EQ.-1)THEN
      TEMP(1)=-1
     ELSE
      TEMP(1)=1
     END IF
     TEMP(2)=2
     TEMP(3)=POLLSH(K-2)
     TEMP(4)=POLLSH(K-1)
     IF(F1M.EQ.3)THEN
      DO 730 M=1,3
       POLLSH(M)=0
730    CONTINUE
      F1M=1
      K=0
      FLAG=0
      GO TO 710
     END IF
     CALL COMPRS (POLLSH,PO,K,F1M,FLAG)
     GO TO 710
    END IF
    IF(POLLSH(K).EQ.LASTOP)THEN
     IF(POLLSH(K-2).EQ.0)THEN
      TEMP(2)=TEMP(2)+1
      STEP=TEMP(2)
      TEMP(2+STEP)=POLLSH(K-1)
      IF(F1M.EQ.3)THEN
       DO 750 M=1,3
        POLLSH(M)=0
750     CONTINUE
       F1M=1
       FLAG=0
       K=0
       GO TO 710
      END IF
      CALL COMPRS (POLLSH,PO,K,F1M,FLAG)
      GO TO 710
     END IF
     IF(POLLSH(K-1).EQ.0)THEN
      TEMP(2)=TEMP(2)+1
      STEP=TEMP(2)
      TEMP(2+STEP)=POLLSH(K-2)
      IF(F1M.EQ.3)THEN
       DO 780 M=1,3
        POLLSH(M)=0
780     CONTINUE
       F1M=1
       FLAG=0
       K=0
       GO TO 710
      END IF
```

```
      CALL COMPRS (POLLSH,PO,K,F1M,FLAG)
      GO TO 710
     END IF
    END IF
    LASTOP=POLLSH(K)
850  CALL ALLOC ('      ',I,TEST,NAMES,NML,CLR)
    NAMES(I)='          '
    DO 920 S=1,TP
     CELLS(I,S+1)=TEMP(S)
920  CONTINUE
    DO 930 S=1,TP
     TEMP(S)=0
930  CONTINUE
     DO 931 L=K-1,1,-1
      IF(POLLSH(L).EQ.0)THEN
       POLLSH(L)=I
       GO TO 935
      END IF
931   CONTINUE
935  IF(LASTOP.EQ.-1)THEN
     TEMP(1)=-1
    ELSE
     TEMP(1)=1
    END IF
    TEMP(2)=2
    TEMP(3)=POLLSH(K-2)
    TEMP(4)=POLLSH(K-1)
    IF(F1M.EQ.3)THEN
     DO 940 M=1,3
      POLLSH(M)=0
940  CONTINUE
     F1M=1
     K=0
     FLAG=0
     GO TO 710
    END IF
    CALL COMPRS (POLLSH,PO,K,F1M,FLAG)
    GO TO 710
    END
    SUBROUTINE COMPRS (POLLSH,PO,K,F1M,FLAG)
    INTEGER PO
    INTEGER POLLSH(PO),K,F1M,FLAG,S
    POLLSH(K-2)=0
    DO 950 S=K+1,F1M
      POLLSH(S-2)=POLLSH(S)
950  CONTINUE
    F1M=F1M-2
    FLAG=0
    K=0
    END



    SUBROUTINE ANCELL( ILJ,KS,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
```

```
      INTEGER CLR,CLC,SMR,SMC,NML

      PARAMETER (IW=10, IL=40, LVECT=50)
      COMMON /ONE/ FLAG
      COMMON/SYGIN/FD
C
      INTEGER  CELLS(CLR,CLC), NM(IW), NJ(IW), K(IW), NZ(IW),
     +      VECTOR ( LVECT ),SYMPTS ( SMR,SMC )
      CHARACTER NAMES(CLR)*(12), FLAG(IW,IL)*40
C
C**** INITIALISE THE ARRAY K
C
      K(1) = KS
      DO 5, I=2,IW

        K ( I ) = 0

 5    CONTINUE

C
C**** CHECK THE LENGTH OF THE DRAWING AREA
C
      CALL EXCEED( ILJ,IL,CELLS,CLR,CLC,NAMES,NML )
C
C**** PUT THE TOP BOX OF THE TREE IN THE MIDDLE OF A ROW
C**** OF ARRAY FLAG AND CODE NAME IT
C
      IF (2*(IW/2) .EQ. IW) THEN
        FLAG(IW/2,ILJ) = 'Z'
        KIW = IW/2
      ELSE
        FLAG(1+IW/2,ILJ) = 'Z'
        KIW = 1+IW/2
      END IF
C
C**** FIND THE NUMBER OF BRANCHES AND MARK THEM IN THE NEXT
C**** ROW OF ARRAY FLAG
      CALL BRANCH(ILJ,K(1),KIW,CELLS,CLR,CLC,NAMES,NML)
C
C**** CHECK TO SEE IF ALL THE BOXES MARKED IN THE LATEST ROW
C**** OF ARRAY FLAG HAVE NAMES
C
 10   L = 0
      J = 1
      N8 = 0
      DO 40 I = 1,IW

        IF (K(I) .NE. 0) THEN

          CALL SETROW ( VECTOR, K ( I ), CELLS, SYMPTS, CLR, CLC, SMR,
     +               SMC )

          N1 = 3
          DO 20 M = J,IW
            IF (FLAG(M,ILJ+1) .NE. 'EMPTY') THEN
```

```
            N1 = N1+1
            IF ( VECTOR ( N1 ) . EQ. 0 ) THEN
              GO TO 30
            END IF
            N8 = N8+1
            IF (CELLS(VECTOR(N1),1) .EQ. 0) THEN
              L = L+1
              NM(L) = I
              NJ(L) = N1
              NZ(L) = N8
            END IF
          END IF
 20     CONTINUE
C
 30     J = M
      ELSE
        GO TO 50
      END IF
 40   CONTINUE
C
 50   IF (L .EQ. 0) THEN
      RETURN
     END IF
C
C**** IF THERE ARE UNNAMED BOXES CONTINUE WITH THE ANALYSIS
C**** OF THE DATA IN ARRAY CELLS
C
    ILJ = ILJ+1
C
C**** CHECK THE LENGTH OF THE DRAWING AREA
C
    CALL EXCEED( ILJ,IL,CELLS,CLR,CLC,NAMES,NML )
C
C**** DEAL WITH EACH UNNAMED BOX IN TURN
C
    DO 80 LA = 1,L
      KA = 0
      DO 60 LB = 1,IW
        IF (FLAG(LB,ILJ) .NE. 'EMPTY') THEN
          KA = KA+1
          IF (KA .EQ. NZ(LA)) THEN
            GO TO 70
          END IF
        END IF
  60   CONTINUE
C
C**** FIND THE ROW OF ARRAY CELLS WHICH HAS INFORMATION ABOUT
C**** THE UNNAMED BOX AND USE THE SUBROUTINE BRANCH TO FIND
C**** THE NUMBER OF BRANCHES BELOW IT AND TO MARK THEM IN
C**** THE NEXT ROW OF ARRAY FLAG
C
  70   K1 = CELLS(K(NM(LA)),NJ(LA))
       CALL BRANCH(ILJ,K1,LB,CELLS,CLR,CLC,NAMES,NML)
  80 CONTINUE
C
```

```
C**** FIND THE ROWS OF CELLS WHICH HOLD INFORMATION ABOUT THE
C**** MARKED BOXES IN THE LATEST ROW OF ARRAY FLAG
C
    ML = 0
    DO 100 LD = 1,IW
      IF (K(LD) .NE. 0) THEN
        DO 90 LF = 4,CLC
          IF (CELLS(K(LD),LF) .NE. 0) THEN
            ML = ML+1
            NM(ML) = CELLS(K(LD),LF)
          ELSE
            GO TO 100
          END IF
 90     CONTINUE
      ELSE
        GO TO 110
      END IF
100  CONTINUE
C
C**** STORE THE ROWS FOUND ABOVE IN ARRAY K AND START AGAIN
C**** CHECKING THE LATEST ROW OF ARRAY FLAG
C
110  DO 120 LP = 1,IW
       IF (LP .LE. ML) THEN
         K(LP) = NM(LP)
       ELSE
         K(LP) = 0
       END IF
120  CONTINUE
     GO TO 10
     END



     SUBROUTINE BRANCH(MLJ,K2,KMW,CELLS,CLR,CLC,NAMES,NML)
C
     INTEGER CLR,CLC,NML
     PARAMETER (IW=10, IL=40)
     COMMON /ONE/FLAG
     COMMON/SYGIN/FD
     INTEGER  CELLS(CLR,CLC)
     CHARACTER FLAG(IW,IL)*40, NAMES(CLR)*(12)
C
C**** CHECK THE SECOND ELEMENT IN THE ROW OF ARRAY CELLS
C
C**** FIND THE NUMBER OF BRANCHES AND CHECK THE WIDTH
C**** OF THE DRAWING AREA
C
     NUM = CELLS(K2,3)
     IF (NUM .GT. IW) THEN
       PRINT *,'*****  WIDTH OF DRAWING AREA TOO SMALL  *****'
       CALL GINEND
       CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
       STOP
     END IF
```

```
C**** CHECK TO SEE IF THE NUMBER OF BRANCHES IS EVEN OR ODD
C**** AND USE SUBROUTINE FILLF TO MARK THEM IN THE NEXT
C**** ROW OF ARRAY FLAG
C
    IF (2*(NUM/2) .EQ. NUM) THEN
      CALL FILLF(KMW,NUM,MLJ,CELLS,CLR,CLC,NAMES,NML)
    ELSE
      FLAG(KMW,MLJ+1) = '10'
      CALL FILLF(KMW,NUM,MLJ,CELLS,CLR,CLC,NAMES,NML)
    END IF
    END



    SUBROUTINE FILLF(KMW,NUM,MLJ,CELLS,CLR,CLC,NAMES,NML)

    INTEGER CLR,CLC,NML
    PARAMETER (IW=10, IL=40)
    COMMON /ONE/ FLAG
    COMMON/SYGIN/FD
    CHARACTER FLAG(IW,IL)*40, CHARA, NAMES(CLR)*(12)
    INTEGER CELLS(CLR,CLC)

C**** DERIVE A CODED NAME FOR EACH MARKED ELEMENT OF ARRAY FLAG
    IF (FLAG(KMW,MLJ) .EQ. 'Z') THEN
      IT = ICHAR( 'A' )
    ELSE
      IT = ICHAR( '1' )
    END IF
C
C**** CODE NAME THE MIDDLE ELEMENT OF THE NEXT ROW OF THE
C**** ARRAY FLAG IF IT IS MARKED
C
    IF (FLAG(KMW,MLJ+1) .EQ. '10') THEN
      CHARA = CHAR( IT+NUM/2 )
      FLAG(KMW,MLJ+1) = CHARA(1:1)//FLAG(KMW,MLJ)
    END IF
C
C**** MARK THE LEFT HAND SIDE BRANCHES IN THE NEXT ROW OF
C**** THE ARRAY FLAG AND CODE NAME THEM
C
    KLW = KMW
    DO 20 N = 1,NUM/2
      IF (KLW-N .LT. 1) THEN
C
C****     USE SUBROUTINE CHECK1 TO SEE IF THE MARKED BOXES
C****     CAN BE MOVED TO THE RIGHT
C
        CALL CHECK1( KLW,MLJ,CELLS,CLR,CLC,NAMES,NML )
      ELSE
C
C****     USE SUBROUTINE SHIFT TO SEE IF THE ELEMENT OF THE
C****     ARRAY FLAG ABOUT TO BE MARKED CAN BE MARKED AND
C****     IF NOT TRY TO MOVE THE ALREADY MARKED BRANCHES
C
```

```
        CALL SHIFT( MLJ,KMW,KLW,N,CELLS,CLR,CLC,NAMES,NML )
      END IF
      CHARA = CHAR( IT+NUM/2-N )
      FLAG(KLW-N,MLJ+1) = CHARA(1:1)//FLAG(KMW,MLJ)
  20 CONTINUE
C
C**** MARK THE RIGHT HAND SIDE BRANCHES AND CODE NAME THEM
C
    DO 40 N = 1,NUM/2
      IF (KLW+N .GT. IW) THEN
C
C****     USE SUBROUTINE CHECK2 TO TRY AND MOVE THE MARKED
C****     BRANCHES TO THE LEFT
C
        CALL CHECK2( MLJ,KLW,CELLS,CLR,CLC,NAMES,NML )
      END IF
      CHARA = CHAR( IT+NUM/2+N )
      FLAG(KLW+N,MLJ+1) = CHARA(1:1)//FLAG(KMW,MLJ)
 40  CONTINUE

    END




*****    SUBROUTINE PART2 : TO INTERPRETE A DATA STRUCTURE,
*****     WHICH REPRESENTS CAUSE AND SYMPTOM EQUATIONS
*****     AS CODED BY DR. D. A. LIHOU FROM OPERABILITY STUDY
*****     RECORDS, AND TO DRAW A REPRESENTATION OF THE ABOVE
*****    EQUATIONS IN THE FORM OF A FAULT TREE USING  PERQ'S GINO PACKAGE


    SUBROUTINE PART2 (CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)

    INTEGER CLR,CLC,SMR,SMC,NML
    PARAMETER ( IW = 10, IL = 40 )
    PARAMETER ( YMAX=350.0, XMAX=1030.0 )
    COMMON /ONE/ FLAG
    COMMON /TWO/ JLJ, KLJ, KR, IOPT
    COMMON/SYGIN/FD
    COMMON/SHIN/SHX,SHY,SCX,SCY
    INTEGER  CELLS(CLR,CLC), IOPT(3), SYMPTS ( SMR, SMC )
    CHARACTER NAMES(CLR)*(12), INDATA*40, FLAG(IW,IL)*40
    REAL X,Y
    INTEGER ICOM
    LOGICAL CLEAR,EXIST,ERROR
    CLEAR = .FALSE.
 1   CONTINUE
    JLJ = 1
    KR = -1
C**** INITIALISE THE ARRAY FLAG
    DO 10 JK = 1,IL
      DO 10 JR = 1,IW
 10 FLAG(JR,JK) = 'EMPTY'
C**** INITIALISE THE BOXSTORE COORDINATES'S ARRAY FLAG
```

```
      CALL BOXSTR(1,'    ',0.,0.,0.,0.,ERROR)
     IF ( CLEAR ) THEN
      CALL COMAND
       GO TO 19
     END IF
C**** READ THE OPTIONS AND ACT APPROPRIATELY
     CALL DAWRI( 'EMPTY',1,CELLS,CLR,CLC,NAMES,NML)
     CALL COMAND

C**** INITIALISE A GINO FILE
C*****  SET UP THE SHIFT FACTORS ( SHX AND SHY ) AND THE SCALE FACTORS
C*****  (SCX AND SCY ) FOR THE GINO GRAPHICS ON THE PERQ

      SHX = 0
      SHY = 250
      SCX = 0.7
      SCY = 0.7

 19   CALL SHIFT2 ( SHX,SHY )
     CALL SCALE2(SCX,SCY)
     CALL ROTAT2 ( 270. )
     CLEAR = .FALSE.
C**** CONSTRUCT THE DRAWING AREA BY OPENING A SPECIAL WINDOW FILES

      OPEN(10,FILE='SINPUT.W')

      OPEN(11,FILE='SINPUT.W')
C**** READ, WRITE OUT IF REQUIRED, AND CHECK THE INPUT DATA
     WRITE(11,FMT=*)'IF THE SCREEN IS CLEAR THEN INPUT DATA FROM KEYS
    +ONLY'
     WRITE(11,FMT=*)'  IF NOT THEN SELECTION AND INPUT ARE FROM CURSOR
    +'
 20   READ(10, '( A )')INDATA
C**** CLOSE THE ACKNOWLEDGEMENT WINDOWS
     CLOSE (12)
     CLOSE(14)
*    CALL DAWRI( INDATA,2,CELLS,CLR,CLC,NAMES,NML )
 77   CALL DATA( INDATA,I,CELLS,CLR,CLC,NAMES,NML, CLEAR, EXIST )

      IF ( CLEAR ) THEN
     DO 25 I=10,19
     CLOSE(I)
 25  CONTINUE
     CALL PICCLE
      GO TO 1
     END IF
     IF ( .NOT. EXIST ) THEN
      GO TO 20
     END IF
     KR = KR+1
C**** ANALYSE THE DATA STRUCTURE AND STORE THE INFORMATION
C**** IN THE ARRAY FLAG
     KLJ = JLJ
     CALL ANCELL(KLJ,I,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
     KLJ = KLJ+1
```

```
C**** USING GINO DRAW THE FAULT TREE CREATED IN ARRAY FLAG
   CALL DRAWTR(I,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
C**** UPDATE THE COUNTER JLJ AND READ THE NEXT ITEM OF DATA
   JLJ = KLJ+1
C****  NOW USE THE CURSOR TO DIRECT INPUT THROUGH DIFFERENT COMMAND ZONES
88  CALL CURSOR(ICOM,X,Y)
      IF(X.GE.182.AND.X.LE.210.AND.Y.GE.18.AND.Y.LE.36)THEN
C**** CLEAR THE SCREEN
      INDATA = 'CLEAR'
      GO TO 77
      ELSEIF(X.GE.182.AND.X.LE.210.AND.Y.GE.0.AND.Y.LT.18)THEN
C**** INPUT THROUGH KEYBOARD
      GO TO 20
      ELSEIF(X.GE.0.AND.X.LE.210.AND.Y.GE.40.AND.Y.LE.120)THEN
C**** OPEN ACKNOWLEDGEMENT WINDOW TO FRESH (CLEAR) THE SCREEN
      CALL NOTE(1)
      GO TO 88
      ELSEIF(X.GE.154.AND.X.LT.182.AND.Y.GE.18.AND.Y.LE.36)THEN
C**** STOP THE PACKAGE
      INDATA = 'STOP'
      GO TO 77
      ELSEIF(X.GE.154.AND.X.LT.182.AND.Y.GE.0.AND.Y.LT.18)THEN
C**** DO THE RELIABILITY CALCULATION OF CURRENT TOP EVENT USING
C**** FTDRA TECHNIQUE
      CALL RLIABL(I)
      GO TO 88
      ELSE
C**** INPUT DATA BY POINTING TO A SPECIFIC EVENT-BOX
66  CALL BOXSTR(3,INDATA,X,Y,0.,0.,ERROR)
   IF(ERROR)THEN
   CALL NOTE(2)
   ELSE
   GO TO 77
    ENDIF
   GO TO 20
   END IF
   RETURN
   END


   SUBROUTINE NOTE(KEY)
   INTEGER KEY
   IF(KEY.EQ.1)THEN
   OPEN(13,FILE='SYEND.W')
   WRITE(13,FMT=*)'****************************************'
   WRITE(13,FMT=*)'*****NOW CLEAR THE SCREEN FOR MORE INPUT*****'
   WRITE(13,FMT=*)'************ USING THE CURSOR *************'
   WRITE(13,FMT=*)'****************************************'
   CALL FLASH(13)
   ELSEIF(KEY.EQ.2)THEN
   OPEN(12,FILE='SERROR.W')
   WRITE(12,FMT=*)'****************************'
   WRITE(12,FMT=*)'******* TRY AGAIN ***********'
   WRITE(12,FMT=*)'****INPUT FROM KEYS ONLY*****'
   WRITE(12,FMT=*)'****************************'
```

```
 CALL FLASH(12)
 ENDIF
 RETURN
 END



  SUBROUTINE DATA( INDATA,I,CELLS,CLR,CLC,NAMES,NML, CLEAR, EXIST )

 INTEGER CLR,CLC,NML,I
 CHARACTER NAMES(CLR)*(12), INDATA*40
 INTEGER CELLS(CLR,CLC)
 LOGICAL CLEAR, EXIST

 IF (INDATA .EQ. 'STOP'.OR.INDATA .EQ. 'STO'.OR.INDATA.EQ.'ST'
+.OR.INDATA.EQ.'S'.OR.INDATA.EQ.'STOP'.OR.INDATA.EQ.'STO'.OR.
+INDATA.EQ.'ST'.OR.INDATA.EQ.'S') THEN
 OPEN(14,FILE='SYEND.W')
 WRITE(14,FMT=*)'   ****************************'
 WRITE(14,FMT=*)'     **** END OF DATA ********'
 WRITE(14,FMT=*)'       ********************'
 CALL FLASH(14)
 CLOSE(14)
 STOP

 ELSE IF ( INDATA .EQ. 'CLEAR' .OR. INDATA .EQ. 'CLEA'.OR.INDATA
+.EQ.'CLE'.OR.INDATA .EQ. 'CL'.OR.INDATA.EQ.'C'.OR.INDATA.EQ.
+'CLEAR'.OR.INDATA.EQ.'CLEA'.OR.INDATA.EQ.'CLE'.OR.INDATA.EQ.
+'CL'.OR.INDATA.EQ.'C') THEN
   CLEAR = .TRUE.
   RETURN
 END IF

 DO 10 I = 1,CLR
   IF (INDATA .EQ. NAMES(I)) THEN
     EXIST = .TRUE.
     RETURN
   END IF
10 CONTINUE

 OPEN(14,FILE='SYEND.W')
 WRITE(14,FMT=*)'******* DATA GIVEN NOT CORRECT *******'
 WRITE(14,FMT=*)'******* THIS DEVIANT STATE WAS NOT CONTAINED IN THE
+INPUT EQUATIONS ***'
 WRITE(14,FMT=*)'******* TRY AGAIN *******'
 CALL FLASH(14)
 CLOSE(14)
 EXIST = .FALSE.
 RETURN
 END
```

```
      SUBROUTINE CHECK1( KLW,MLJ,CELLS,CLR,CLC,NAMES,NML )
C
      INTEGER CLR,CLC,NML

      PARAMETER (IW=10, IL=40)
      COMMON /ONE/ FLAG
      COMMON/SYGIN/FD
      INTEGER CELLS(CLR,CLC)
      CHARACTER FLAG(IW,IL)*40, NAMES(CLR)*(12)
C
      KLW = KLW+1
      DO 20 IA = 1,IW
        IF (FLAG(IA,MLJ+1) .EQ. 'EMPTY') THEN
          DO 10 IB = IA,2,-1
   10       FLAG(IB,MLJ+1) = FLAG(IB-1,MLJ+1)
          FLAG(1,MLJ+1) = 'EMPTY'
          GO TO 30
        END IF
   20 CONTINUE
C
   30 IF (FLAG(1,MLJ+1) .NE. 'EMPTY') THEN
        PRINT *,'*****  DRAWING  AREA  NOT  WIDE  ENOUGH  *****'
        CALL GINEND
        CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
        STOP
      END IF
       END



      SUBROUTINE CHECK2( MLJ,KLW,CELLS,CLR,CLC,NAMES,NML )
C
      INTEGER CLR,CLC,NML
      PARAMETER (IW=10, IL=40)
      COMMON /ONE/ FLAG
      COMMON/SYGIN/FD
      CHARACTER FLAG(IW,IL)*40, NAMES(CLR)*(12)
      INTEGER CELLS(CLR,CLC)

      KLW = KLW-1
      DO 20 IA = IW,1,-1
        IF (FLAG(IA,MLJ+1) .EQ. 'EMPTY') THEN
          DO 10 IB = IA,IW-1
   10       FLAG(IB,MLJ+1) = FLAG(IB+1,MLJ+1)
          FLAG(IW,MLJ+1) = 'EMPTY'
          GO TO 30
        END IF
   20 CONTINUE
   30 IF (FLAG(IW,MLJ+1) .NE. 'EMPTY') THEN
        PRINT *,'*****  DRAWING  AREA  TOO  NARROW  *****'
        CALL GINEND
        CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
        STOP
      END IF
      END
```

```
      SUBROUTINE SHIFT( MLJ,KMW,KLW,N,CELLS,CLR,CLC,NAMES,NML )
C
      INTEGER CLR,CLC,NML
      PARAMETER (IW=10, IL=40)
      COMMON /ONE/ FLAG
      COMMON/SYGIN/FD
      CHARACTER FLAG(IW,IL)*40, NAMES(CLR)*(12)
      INTEGER CELLS(CLR,CLC)
C
   10 KSW = KLW
      K = 0
      DO 110 IA = IW,KSW-N,-1
        IF (FLAG(IA,MLJ+1) .NE. 'EMPTY') THEN
          IF (FLAG(IA,MLJ+1)(2:) .NE. FLAG(KMW,MLJ)) THEN
            IF (NALEN( FLAG(KMW,MLJ) ) .EQ. 2) THEN
              IF (FLAG(IW,MLJ) .EQ. 'EMPTY' .AND.
     1              FLAG(IW,MLJ+1) .EQ. 'EMPTY') THEN
                DO 20 IB = IW,KMW+1,-1
     20           FLAG(IB,MLJ) = FLAG(IB-1,MLJ)
                FLAG(KMW,MLJ) = 'EMPTY'
                KMW = KMW+1
                DO 30 IC = IW,IA+2,-1
     30           FLAG(IC,MLJ+1) = FLAG(IC-1,MLJ+1)
                FLAG(IA+1,MLJ+1) = 'EMPTY'
                KLW = KLW+1
                K = 1
              END IF
              IF (FLAG(1,MLJ) .EQ. 'EMPTY' .AND.
     1              FLAG(1,MLJ+1) .EQ. 'EMPTY') THEN
                DO 40 ID = 1,KMW-2
     40           FLAG(ID,MLJ) = FLAG(ID+1,MLJ)
                FLAG(KMW-1,MLJ) = 'EMPTY'
                DO 50 IE = 1,IA-1
     50           FLAG(IE,MLJ+1) = FLAG(IE+1,MLJ+1)
                FLAG(IA,MLJ+1) = 'EMPTY'
                K = K+2
              END IF
            END IF
            IF (K .EQ. 3) THEN
              GO TO 120
            ELSE IF (K .EQ. 2) THEN
              GO TO 90
            END IF
            DO 60 IG = 1,IA-1
              IF (FLAG(IG,MLJ+1) .EQ. 'EMPTY') THEN
                GO TO 70
              END IF
     60       CONTINUE
C
     70       IF (IG .LT. IA) THEN
              DO 80 JA = IG,IA-1
     80         FLAG(JA,MLJ+1) = FLAG(JA+1,MLJ+1)
              FLAG(IA,MLJ+1) = 'EMPTY'
              K = K+5
            END IF
```

```
          IF (K .EQ. 6) THEN
            GO TO 120
          END IF
 90        IF (FLAG(IW,MLJ+1) .EQ. 'EMPTY') THEN
           DO 100 JB = IW,IA+2,-1
100          FLAG(JB,MLJ+1) = FLAG(JB-1,MLJ+1)
           FLAG(IA+1,MLJ+1) = 'EMPTY'
           KLW = KLW+1
           K = 8
           GO TO 120
          END IF
        END IF
      END IF
110 CONTINUE
     IF (K .EQ. 0) THEN
      IF (FLAG(KLW-N,MLJ+1) .NE. 'EMPTY') THEN
        PRINT *,'*****  WIDTH OF DRAWING AREA EXCEEDED  *****'
        CALL GINEND
        CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
        STOP
      END IF
      RETURN
    END IF
120 GO TO 10
    END


    SUBROUTINE BOXSTR(KEY,ANAME,XX1,YY1,XX2,YY2,ERROR)

    PARAMETER (NML=12)
    REAL COORD(40,4),XX1,YY1,XX2,YY2
    INTEGER KEY,POINTA
    CHARACTER ANAME*(*),LIST(40)*(NML)
    COMMON/SYGIN/FD
    LOGICAL ERROR
    SAVE COORD,POINTA

    IF(KEY.EQ.1)THEN
      POINTA = 1
      DO 10 I =1,40
      DO 5 J =1,4
        COORD(I,J) = 0
5     CONTINUE
10    CONTINUE
    ERROR = .FALSE.
    ELSEIF(KEY.EQ.2)THEN
       IF(POINTA.GT.40)THEN
       ERROR = .TRUE.
       RETURN
       ENDIF
     LIST(POINTA) = ANAME
     COORD(POINTA,1) = XX1
     COORD(POINTA,2) = YY1
     COORD(POINTA,3) = XX2
     COORD(POINTA,4) = YY2
```

```
      POINTA = POINTA + 1
      ERROR = .FALSE.
     ELSEIF(KEY.EQ.3)THEN
       DO 100 I = 1,POINTA-1
         IF(XX1.GE.COORD(I,1).AND.XX1.LE.COORD(I,3).AND.YY1
    +.GE.COORD(I,2).AND.YY1.LE.COORD(I,4))THEN
          ANAME = LIST(I)
          ERROR = .FALSE.
          RETURN
          ENDIF
  100 CONTINUE
     ERROR =.TRUE.
     ENDIF
     RETURN
     END



     SUBROUTINE DRAWTR(I,CELLS,CLR,CLC,SYMPTS,SMR,SMC,NAMES,NML)
C
     INTEGER CLR,CLC,SMR,SMC,NML
     PARAMETER (IW=10, IL=40, LVECT=50)
     COMMON /ONE/ FLAG
     COMMON /TWO/ JLJ, KLJ, KR, IOPT
     COMMON/SYGIN/FD
     INTEGER  CELLS(CLR,CLC), TAV(IW), SAV(IW), IOPT(3),
    +       SYMPTS ( SMR, SMC ), VECTOR ( LVECT )
     CHARACTER NAMES(CLR)*(12), FLAG(IW,IL)*40
     DIMENSION  X1(IW), X2(IW), Y1(IW), Y2(IW)

C
C**** INITIALISE THE ARRAYS: TAV SAV X1 X2 Y1 Y2
     DO 5, INIT = 1, IW

        TAV ( INIT ) = 0
        SAV ( INIT ) = 0
        X1  ( INIT ) = 0
        X2  ( INIT ) = 0
        Y1  ( INIT ) = 0
        Y2  ( INIT ) = 0

  5   CONTINUE

C
C**** POSITION THE PEN TO DRAW THE TOP BOX OF THE FAULT TREE
     XPOS = (JLJ-1)*20.0 + KR*10.0
C
     DO 10 LB = 1,IW
       IF (FLAG(LB,JLJ) .EQ. 'Z') THEN
         GO TO 20
       END IF
  10 CONTINUE
C
  20 YMID = (2*LB-1)*17.0
     CALL MOVTO2( XPOS,YMID )
```

```
C**** DRAW THE BOX AND WRITE IN IT
    CALL ABOX(NAMES(I),NML)

C
C**** DRAW THE GATE IF THERE IS ONE
      CALL MOVTO2(XPOS+7,YMID)

    IF (CELLS(I,2) .EQ. 0 .AND. CELLS(I,3) .GT. 1
   *   .OR. CELLS ( I, 2 ) .EQ. 1 ) THEN
      CALL LINBY2( 4.0,0.0 )
      CALL ORED
    ELSE IF (CELLS(I,2) .EQ. +2 .OR. CELLS(I,2) .EQ.0 .AND.
   *    CELLS(I,3) .EQ.1) THEN
      CALL LINBY2( 9.0,0.0 )
    ELSE IF (CELLS(I,2) .EQ. -1) THEN
      CALL LINBY2( 4.0,0.0 )
      CALL ANDED
    END IF
C
C**** FIND THE POSITION OF EACH BOX IN THE NEXT ROW OF FLAG
    L5 = 0
    L6 = 4
    DO 50 LD = 1,IW
      IF (FLAG(LD,JLJ+1) .NE. 'EMPTY') THEN
        DO 30 LF = L6,CLC

          CALL SETROW ( VECTOR, I, CELLS, SYMPTS,
   +                CLR, CLC, SMR, SMC )
          IF (VECTOR(LF) .NE. 0) THEN
            L5 = L5+1
            IF (L5 .EQ. 1) THEN
              LD1 = LD
              LD2 = LD
            ELSE
              LD2 = LD
            END IF
C
C****         SAVE THE ROW OF ARRAY CELLS WHICH HOLDS
C****         INFORMATION ABOUT THE BOX
            TAV(L5) = VECTOR(LF)
            L6 = LF+1
            GO TO 40
          ELSE
        PRINT *,'*****  ERROR IN ANALYSIS OF DATA STRUCTURE  *****'
            CALL GINEND
            CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
            STOP
          END IF
  30    CONTINUE
C
C****     POSITION THE PEN FOR DRAWING
  40    XPOS1 = XPOS + 20.0
        YMID = (2*LD-1)*17.0
        CALL MOVTO2( XPOS1,YMID )
C
```

```
C****    SAVE THE CO-ORDINATES OF THE PEN'S POSITION
       X1(LD) = XPOS1 - 4.0
       Y1(LD) = YMID
C
C****    CHECK TO SEE IF THE BOX HAS A NAME INSIDE IT
       IF (CELLS(TAV(L5),1) .EQ. 1) THEN
C
C****      DRAW THE BOX AND WRITE IN IT
         CALL ABOX( NAMES(TAV(L5)),NML)

         CALL MOVTO2(X1(LD)+7,Y1(LD))
         CALL MOVBY2( -7.0,0.0 )
         CALL LINBY2( 4.0,0.0 )
       END IF
     END IF
 50  CONTINUE
C
C**** CONNECT THE DRAWN BOXES BETWEEN THEM AND TO THE GATE
C
   IF ( L5 .NE. 0 ) THEN

     CALL MOVTO2( X1(LD1),Y1(LD1) )
   CALL LINTO2( X1(LD2),Y1(LD2) )

   END IF

C
C**** DO THE REST OF THE DRAWING BY CONSIDERING THE ROWS
C**** OF ARRAY FLAG ONE AT A TIME
   DO 110 MD = JLJ+1,KLJ
     L7 = 0
     L9 = 0
C
C****  FIND THE POSITION OF EACH BOX IN THE NEXT ROW
C****  OF ARRAY FLAG
     DO 90 MA = 1,IW
       IF (FLAG(MA,MD) .NE. 'EMPTY') THEN
         L7 = L7+1
C
C****      FIND OUT IF THE BOX IS NAMED
         IF (CELLS(TAV(L7),1) .EQ. 1) THEN
           IF (CELLS(TAV(L7),2) .EQ. -2) THEN
             CONTINUE
           ELSE
C
C****         SHOW THE NUMBER OF BRANCHES BELOW THE BOX
             NUM = CELLS(TAV(L7),3)
             CALL MOVTO2( X1(MA)+11.0,Y1(MA) )
             CALL NUMBRA( NUM )
           END IF
         ELSE
C
C****       FOR EACH UNNAMED BOX DRAW THE BRANCHES BELOW IT
           L5 = 0
           L6 = 4
```

```
          DO 80 KA = 1,IW
            IF (FLAG(MA,MD) .EQ. FLAG(KA,MD+1)(2:)) THEN
              L9 = L9+1
              DO 60 LF = L6,CLC

              CALL SETROW ( VECTOR, TAV ( L7 ), CELLS,
     +                    SYMPTS, CLR, CLC, SMR, SMC )
                IF (VECTOR(LF) .NE. 0) THEN
                  L5 = L5+1
                  IF (L5 .EQ. 1) THEN
                    KA1 = KA
                    KA2 = KA
                  ELSE
                    KA2 = KA
                  END IF
C
C****               SAVE THE ROW OF CELLS WHICH HAS
C****               INFORMATION ABOUT THE BOX
                  SAV(L9) = VECTOR(LF)
                  L6 = LF+1
                  GO TO 70
                ELSE
       PRINT *,'*****  DATA STRUCTURE NOT ANALYSED WELL  *****'
                  CALL GINEND
                  CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,
     *                NAMES,NML )
                  STOP
                END IF
   60           CONTINUE
C
C****           POSITION THE PEN TO DRAW THE BOX
   70           XPOS1 = XPOS + (MD+1-JLJ)*20.0
              YMID = (2*KA - 1)*17.0
              CALL MOVTO2( XPOS1,YMID )
C
C****           SAVE THE CO-ORDINATES OF THE PEN'S POSITION
              X2(KA) = XPOS1-4.0
              Y2(KA) = YMID
C
C****           CHECK TO SEE IF THE BOX HAS A NAME
              IF (CELLS(SAV(L9),1) .EQ. 1) THEN
C
C****             DRAW THE BOX AND WRITE IN IT
               CALL ABOX( NAMES(SAV(L9)),NML )

                CALL MOVTO2(X2(KA)+7,Y2(KA))
                CALL MOVBY2( -7.0,0.0 )
                CALL LINBY2( 4.0,0.0 )
              END IF
            END IF
   80       CONTINUE
C
C****       CONNECT THE DRAWN BOXES BETWEEN THEM
C****       AND TO THE GATE
          CALL MOVTO2( X2(KA1),Y2(KA1) )
```

```
          CALL LINTO2( X2(KA2),Y2(KA2) )
C
C****        DRAW  THE  GATE
          Y3 = (Y2(KA1) + Y2(KA2)) / 2.0
          CALL MOVTO2( X1(MA),Y1(MA) )
          CALL LINTO2( X2(KA1)-5.0,Y3 )
          IF (CELLS(TAV(L7),2) .EQ. 1) THEN
            CALL ORED
          ELSE IF (CELLS(TAV(L7),2) .EQ. -1) THEN
            CALL ANDED
          ELSE
      PRINT *,'*****  DATA STRUCTURE CONTAINS ERRORS  *****'
            CALL GINEND
            CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
            STOP
          END IF
        END IF
      END IF
 90   CONTINUE
      DO 100 KB = 1,IW
        TAV(KB) = SAV(KB)
        X1(KB) = X2(KB)
100     Y1(KB) = Y2(KB)
C
 110 CONTINUE
C
     CALL CHAMOD
     END


     SUBROUTINE ABOX(A,NML)

     INTEGER NML, ERRNO, VALUE,FINAL
     REAL POSX, POSY, POSZ, AWID, ALEN1

     COMMON/SYGIN/FD
     PARAMETER (ALEN=24.0, BLENG=50 )
     CHARACTER  A*(12)
     LOGICAL ERROR
C
     CALL DRIVER ( 3, VALUE, ERRNO )

     IF ( VALUE .EQ. 1 ) THEN

      ALEN1 = 30
      AWID = 7

     ELSE

      ALEN1 =  FINAL ( A ) * 3.68
      AWID = 7

     END IF
     ALET = (ALEN1 / 2.0)
     CALL LINBY2( 0.0,ALET )
```

```
      CALL POSPIC(XX2,YY2)
      CALL LINBY2( AWID,0.0 )
      CALL LINBY2( 0.0,-ALEN1 )
      CALL POSPIC(XX1,YY1)
      CALL LINBY2( -AWID,0.0 )
      CALL LINBY2( 0.0,ALET )
      CALL POSSPA ( POSX, POSY, POSZ )
      CALL BOXSTR(2,A,XX1,YY1,XX2,YY2,ERROR)


      CALL MOVBY2( AWID - 1, -ALET+1 )
      NC = NALEN( A )
      CALL CHAHOL ( A ( 1 : FINAL ( A ) ) )


*
****   TAKE ALTERNATIVE ACTIONS FOR DIFFERENT OUTPUT DEVICES
****   FOR CHARACTER OUTPUT
*

      IF ( VALUE .EQ. 1 ) THEN

       CALL CHASIZ ( 2.6, 4.2 )
       CALL MOVBY2 ( 2.5, ( 15. - NC ) / 2 + 1 )

      ELSE

       CALL CHASIZ( (ALEN1-1)/NC,5.0 )
       CALL CHAANG ( 90.0 )
       CALL PENSEL( 7,0.0,0 )

      END IF



      IF ( VALUE .EQ. 1 ) THEN

       CALL MOVTO2 ( POSX + 7.0 , POSY )

      ELSE

       YP = -(NML*(ALEN1-1)/NC) + ALET - 1.0
       CALL MOVBY2( 1.0,YP )
       CALL PENSEL( 1,0.0,0 )

      END IF
C
      END


      SUBROUTINE ORED
      COMMON/SYGIN/FD
C
      CALL LINBY2( 0.0,5.0 )
      CALL LINBY2( 5.0,0.0 )
```

```
      CALL LINBY2( 0.0,-10.0 )
      CALL LINBY2( -5.0,0.0 )
      CALL LINBY2( 0.0,5.0 )
      CALL MOVBY2( 5.0,0.0 )
      END


      SUBROUTINE ANDED
C
      COMMON/SYGIN/FD
      CALL MOVBY2( 5.0,-5.0 )
      CALL ARCBY2( 0.0,5.0,0.0,10.0,0 )
      CALL MOVBY2( 0.0,-5.0 )
      END


      SUBROUTINE STAR
C
      INTEGER NOTERM, ERRNO
      COMMON/SYGIN/FD

      CALL PENSEL( 2,0.0,0 )
      CALL LINBY2( 2.0,0.0 )
      CALL ARCBY2( 2.0,0.0,0.0,0.0,0 )
      CALL MOVBY2( 3.0,-1.0 )
      CALL DRIVER ( 3, NOTERM, ERRNO )

      IF ( NOTERM .EQ. 1 ) THEN

*       NEWBURY TERMINAL

       CONTINUE

      ELSE

       CALL CHASIZ( 3.0,2.0 )
       CALL CHAANG ( 90.0 )

      END IF

      CALL PENSEL( 5,0.0,0 )
      CALL CHAASC( 42 )
      CALL PENSEL( 1,0.0,0 )
      END


      SUBROUTINE NUMBRA( I )
      COMMON/SYGIN/FD
      INTEGER NOTERM, ERRNO

      CALL PENSEL( 2,0.0,0 )
      CALL LINBY2( 2.0,0.0 )
      CALL ARCBY2( 2.0,0.0,0.0,0.0,0 )
      CALL MOVBY2( 3.0,-1.0 )
```

```
      CALL DRIVER ( 3, NOTERM, ERRNO )

      IF ( NOTERM .EQ. 1 ) THEN

*       NEWBURY

        CALL MOVBY2 ( 3., 2. )

      ELSE

        CALL CHASIZ( 3.0,2.0 )
        CALL CHAANG ( 90.0 )

        CALL PENSEL( 5,0.0,0 )
        CALL CHAINT( I,1 )

      END IF
      CALL PENSEL( 1,0.0,0 )
      END


      SUBROUTINE EXCEED( ILJ,IL,CELLS,CLR,CLC,NAMES,NML )
C
      INTEGER CLR,CLC,NML
      COMMON /TWO/ JLJ, KLJ, KR, IOPT
      COMMON/SYGIN/FD
      CHARACTER NAMES(CLR)*(12)
      INTEGER  IOPT(3), CELLS(CLR,CLC)
C
C**** CHECK TO SEE IF THE LENGTH OF THE DRAWING AREA
C**** HAS BEEN EXCEEDED
      IF ( ILJ+1 .GT. IL ) THEN
        IF (JLJ .EQ. 1) THEN
      PRINT *,'*****  LENGTH OF DRAWING AREA TOO SMALL  *****'
          CALL GINEND
          CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
          STOP
        END IF
      PRINT *,'*****  PROGRAM RUN OUT OF DRAWING AREA  *****'
      PRINT *, '*****  PLEASE START AGAIN WITH THE REST OF THE
     1DATA  *****'
        CALL GINEND
        CALL DAWRI( 'EMPTY',3,CELLS,CLR,CLC,NAMES,NML )
        STOP
      END IF
      END


      SUBROUTINE GINEND
C
      PARAMETER ( YMAX=350.0, XMAX=1030.0 )
      COMMON /TWO/ JLJ, KLJ, KR, IOPT
      COMMON/SYGIN/FD
      INTEGER  IOPT(3)
C
```

```
C**** DRAW A FRAME AND CLOSE GINO
      CALL MOVTO2( 0.0,0.0 )
      CALL SHIFT2( -15.0,-5.0 )
      CALL MOVTO2( 0.0,0.0 )
      XPOS = KLJ*20.0 + KR*10.0 + 30.0
*
*     NO FRAME ON INTERACTIVE TERMINAL
*
      GO TO 10

      CALL FRAME( XPOS,YMAX,1 )
      CALL FRAME( XPOS-2.0,YMAX-2.0,2 )
      CALL FRAME( XPOS-4.0,YMAX-4.0,5 )
      CALL FRAME( XPOS-6.0,YMAX-6.0,7 )
10    CALL DEVEND
      END


      SUBROUTINE FRAME( XPOS,YMAX,ICOL )
C
      CALL PENSEL( ICOL,0.0,0 )
      CALL LINBY2( XPOS,0.0 )
      CALL LINBY2( 0.0,YMAX )
      CALL LINBY2( -XPOS,0.0 )
      CALL LINBY2( 0.0,-YMAX )
      CALL MOVBY2( 1.0,1.0 )
      END


      SUBROUTINE DAWRI(INDATA,INDEX,CELLS,CLR,CLC,NAMES,NML)
C
      INTEGER CLR,CLC,SMR,SMC,NML
      PARAMETER (IW=10, IL=40)
      COMMON /ONE/ FLAG
      COMMON /TWO/ JLJ, KLJ, KR, IOPT
      INTEGER  CELLS(CLR,CLC), IOPT(3)
      CHARACTER NAMES(CLR)*(12), INDATA*(*), FLAG(IW,IL)*40
C
      IF (INDEX .EQ. 1) THEN
*       IF (IOPT(1) .EQ. 1) THEN
C
C****       WRITE OUT THE CONTENTS OF ARRAYS CELLS AND NAMES
*         WRITE (32,10) ( (CELLS(I2,I1), I1 = 1,CLC), I2 = 1,CLR)
*  10      FORMAT (1H1, 'CONTENTS OF ARRAY CELLS:' / 1H , 27('*')
*    1           /// 50(1H , 2(5(5X,I5) /)/ ) )
*         WRITE (32,25) (NAMES(I),I=1,CLC)
*  25      FORMAT (1H1, 'CONTENTS OF ARRAY NAMES:' / 1H , 27('*')
*    1           /// (1H , A20) )
*       END IF
*     ELSE IF (INDEX .EQ. 2) THEN
*       IF (IOPT(2) .EQ. 1) THEN
C
C****       WRITE OUT THE INPUT DATA USED
*         IF (JLJ .EQ. 1) THEN
*           WRITE (32, '(1H1, "INPUT DATA USED:" /
```

```
*    1          1H , 18("*") // )' )
*         END IF
*         WRITE (32, '(1H , A20)' ) INDATA
*       END IF
*     ELSE IF (INDEX .EQ. 3) THEN
*       IF (IOPT(3) .EQ. 1) THEN
C
C****     WRITE OUT THE CONTENTS OF ARRAY FLAG
*         WRITE (32,30) ( (FLAG(J1,J2), J1 = 1,IW), J2 = 1,IL+1)
*  30     FORMAT (1H1, 'CONTENTS OF ARRAY FLAG:' / 1H , 26('*')
*     1          /// 40(1H , 10A10 / ) )
*       END IF
      END IF
      END


      INTEGER FUNCTION NALEN( A )
      COMMON/SYGIN/FD
C
C**** IT COUNTS THE CHARACTERS OF A CHARACTER VARIABLE
C**** BEFORE THE FIRST BLANK CHARACTER
C
      PARAMETER ( NML=12 )
      CHARACTER A*(*)
C
      DO 10 K = 1,NML
        IF (A(K:K) .EQ. ' ') THEN
          GO TO 20
        END IF
   10 CONTINUE
C
   20 NALEN = K-1
      END



      SUBROUTINE FSOPEN ( FILNAM, UNIT, FORMAT )

*       THIS SUBROUTINE OPENS A SEQUENTIAL FILE
*       IF THE FILE ALREADY EXISTS, IT IS JUST OPENED
*       IF THE FILE DOES NOT EXIST, IT IS CREATED FIRST
*
*       FILNAM - CHARACTER - THE NAME OF THE FILE
*       UNIT - INTEGER - UNIT NUMBER FOR OPENING
*       FORMAT - CHARACTER - EITHER 'FORMATTED' OR 'UNFORMATTED'
*

      INTEGER UNIT
      CHARACTER FILNAM * 12, FORMAT * ( * ), STATUS * 7
      LOGICAL THERE



      INQUIRE ( FILE = '       '//FILNAM, EXIST = THERE )
```

```
      IF ( THERE ) THEN

        STATUS = 'MODIFY'

      ELSE
        STATUS = 'NEW'
      END IF

      OPEN ( FILE = '       '//FILNAM, UNIT = UNIT,
     +     STATUS = STATUS, FORM = FORMAT, ERR = 900 )
      REWIND(UNIT)
      RETURN

  900 PRINT *, 'ERROR IN OPENING FILE', FILNAM
      STOP
      END


      INTEGER FUNCTION NXTNSP ( TEXT )

  ****    RETURNS THE POSITION OF THE FIRST NON-SPACE CHARACTER IN TEXT
  ****    RETURNS 0 IF NO NON-SPACE CHARACTER
  *
      CHARACTER TEXT * ( * )
      INTEGER TEMP

      DO 10, TEMP = 1, LEN ( TEXT )

        IF ( TEXT ( TEMP : TEMP ) .NE. ' ' ) THEN

          NXTNSP = TEMP
          RETURN

        END IF

   10 CONTINUE

      NXTNSP = 0
      RETURN
      END


      SUBROUTINE INERR ( EQN )

  ****    PRINTS OUT ERROR MESSAGE AND THE FAULTY EQUATION

      CHARACTER EQN * ( * )

      OPEN(UNIT=9,FILE='ERROR')
      WRITE ( UNIT = 9, FMT = 900 )
  900 FORMAT ( /// 1X, 80('*') /// )
      PRINT *, 'THE FOLLOWING LINE OF INPUT IS NOT A CORRECT EQUATION'
      PRINT *
      PRINT *, EQN
      WRITE ( UNIT = 9, FMT = 900 )
```

```
      RETURN
      END


      SUBROUTINE SETROW ( VECTOR, NUMBER, CELLS, SYMPTS,
     +                CLR, CLC, SMR, SMC )

      INTEGER CLR, CLC, SMR, SMC
      INTEGER VECTOR (CLC ),  CELLS ( CLR, CLC ), SYMPTS ( SMR, SMC ),
     +      COUNT, NUMBER
      COMMON/SYGIN/FD
           IF ( CELLS ( NUMBER, 2 )  .EQ. 0 ) THEN

             DO 104, COUNT = 1, CLC

               VECTOR ( COUNT ) = 0

104            CONTINUE

             DO 105, COUNT = 1, 3

               VECTOR ( COUNT ) = CELLS ( NUMBER, COUNT )

105            CONTINUE

             DO 106, COUNT = 4, 3 + CELLS ( NUMBER, 3 )

               VECTOR ( COUNT ) = SYMPTS ( CELLS ( NUMBER,
     +                      COUNT ), 1 )

106            CONTINUE

           ELSE

           DO 11, COUNT = 1, CLC

             VECTOR ( COUNT ) = CELLS ( NUMBER, COUNT )

11         CONTINUE

           END IF

      RETURN

      END
```

APPENDIX J : LISTING OF THE SOURCE PROGRAMS OF THE GINO-PNX VERSION

```
C     THIS IS THE SAME FILE AS THE PROJECT FILE

      SUBROUTINE PERQ
C     THIS SUBROUTINE SETS THE DEFAULT VALUES FOR GRAPHICAL OUTPUT.
      REAL PPOS,OTR
      REAL VPOS,XMIN,XMAX,YMIN,YMAX
      INTEGER IW
      LOGICAL FLAG,LOGIC
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      CALL PICCLE
      XMIN = 0.0
      XMAX = 210.0
      YMIN = 0.0
      YMAX = 275.0
      IW   = 0
      R    = 1.0
      TOL  = 0.05
      ITYPE = 0
      FLAG =.TRUE.
      LOGIC = .FALSE.
C     NOW INITIALISE THE TRANSFORMATION MATRIX AND THE START POINT.
      DO 10 I  = 1,3
        DO 5 J = 1,3
          OTR(I,J) = 0.0
 5      CONTINUE
        OTR(I,I) = 1.0
        VPOS(I) = 0.0
        PPOS(I) = 0.0
10      CONTINUE
      VPOS(3) = 1.0
      RETURN
      END


      SUBROUTINE PICCLE
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      CALL CCLOSE
      CALL COPEN
      RETURN
      END

      SUBROUTINE CCLOSE
      INTEGER KK,WCLOSE
      COMMON/SYGIN/FD
      KK = WCLOSE(FD)
      RETURN
      END


      SUBROUTINE COPEN
```

```
      INTEGER FD,WOPEN
      COMMON/SYGIN/FD
      FD = WOPEN('SGINO.W')
      RETURN
      END



      SUBROUTINE CURSOR(I,X,Y)
      REAL X,Y
      LOGICAL FLAG,LOGIC
       INTEGER FD,K,WPUCK,X1,Y1,X2,Y2,X3,Y3,PRESS,I
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
       COMMON/SYGIN/FD
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      K=WPUCK(FD,X1,Y1,X2,Y2,X3,Y3,PRESS)
      I = PRESS
       X = 210.*X1/767/R
       Y = ( 275. - 275.*Y1/1023)/R
      RETURN
      END


      SUBROUTINE ROTAT2(ALPHA)
C     ANGLE ALPHA IN DEGREES
C     ALPHA POSITIVE = ANTICLOCKWISE ROTATION ABOUT ORIGIN
C     ALPHA NEGATIVE = CLOCKWISE ROTATION ABOUT ORIGIN
      REAL PI,A,ALPHA,ROT(3,3)
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      DO 2 I = 1,3
       DO 2 J = 1,3
        ROT(I,J) = 0.0
    2 CONTINUE
      ROT(3,3) = 1.0
      PI = 4.0 * ATAN(1.0)
      A = PI*ALPHA / 180.0
      ROT(1,1) = COS(A)
      ROT(2,2) = COS(A)
      ROT(2,1) = SIN(A)
      ROT(1,2) = -SIN(A)
      CALL RFRESH(ROT)
      CALL MATINV(ROT)
      RETURN
      END
      SUBROUTINE SCALE2(SX,SY)
      REAL SX,SY,SCA(3,3)
      INTEGER I,J
      LOGICAL FLAG,LOGIC
      COMMON/SYGIN/FD
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      DO 2  I = 1,3
      DO 2  J = 1,3
       SCA(I,J) = 0.0
```

```
 2  CONTINUE
    SCA(1,1) = SX
    SCA(2,2) = SY
    SCA(3,3) = 1.0
    CALL RFRESH(SCA)
    CALL MATINV(SCA)
    RETURN
    END



    SUBROUTINE SCALE(S)
    REAL S,SCA(3,3)
    INTEGER I,J
    LOGICAL FLAG,LOGIC
    COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
     COMMON/SYGIN/FD
    DO 10 I = 1,3
     DO 5 J = 1,3
      SCA(I,J) = 0.0
5     CONTINUE
10   CONTINUE
    SCA(1,1) = S
    SCA(2,2) = S
    SCA(3,3) = 1.0
    CALL RFRESH(SCA)
    CALL MATINV(SCA)
    RETURN
    END



    SUBROUTINE SHEAR2(IDEP,A)
    REAL A,SHE(3,3)
    INTEGER IDEP,I,J
    LOGICAL FLAG,LOGIC
    COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
    COMMON/SYGIN/FD
    DO 10 I=1,3
     DO 5  J = 1,3
      SHE(I,J) = 0.0
5     CONTINUE
    SHE(I,I) = 1.0
10   CONTINUE
    IF(IDEP.EQ.1) THEN
     SHE(2,1) = A
    ELSE IF(IDEP.EQ.2) THEN
     SHE(1,2) = A
    ELSE
     WRITE(3,100)
    END IF
    CALL RFRESH(SHE)
    CALL MATINV(SHE)
100  FORMAT(1X,'ERROR: FIRST ARGUMENT IN ROUTINE SHEAR2
  ! MUST BE 1 OR 2')
```

```
      RETURN
      END



      SUBROUTINE SHIFT2(DX,DY)
      REAL DX,DY,SHI(3,3)
      INTEGER I,J
      LOGICAL FLAG,LOGIC
      COMMON/SYGIN/FD
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      DO 10  I = 1,3
       DO 5  J = 1,3
        SHI(I,J) = 0.0
5      CONTINUE
      SHI(I,I) = 1.0
10    CONTINUE
      SHI(1,3) = DX
      SHI(2,3) = DY
      CALL RFRESH(SHI)
      RETURN
      END



      SUBROUTINE VIEWSE(NHORIZ)
      REAL VIE(3,3)
      INTEGER I,J,NHORIZ
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      DO 10  I = 1,3
       DO 5 J = 1,3
        VIE(I,J) = 0.0
5      CONTINUE
      VIE(I,I) = 1.0
10    CONTINUE
      IF(NHORIZ.EQ.2) THEN
       VIE(1,1) = 0.0
       VIE(1,2) = 1.0
       VIE(2,1) = 1.0
       VIE(2,2) = 0.0
      CALL RFRESH(VIE)
      ELSE IF(NHORIZ.EQ.1) THEN
      RETURN
      ELSE
       WRITE(3,100)
      END IF
100   FORMAT(1X,'ERROR: ARGUMENT IN ROUTINE VIEWSE MUST BE 2
     ! IF AXIS TO BE PERMUTED, OR 1 IF NOT TO BE')
      RETURN
      END

      SUBROUTINE LINEXX(M1,N1,M2,N2)
      INTEGER I,WLIN,M1,N1,M2,N2
```

```
      COMMON/SYGIN/FD
      I=WLIN(FD,M1,N1,M2,N2)
      RETURN
      END

      SUBROUTINE LINTO2(S,T)
      REAL X(2),Y(2),U(2),V(2)
      INTEGER M(2),N(2),IW
      LOGICAL FLAG,LOGIC
      COMMON/SYGIN/FD
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      CALL MMULT(OTR,VPOS,PPOS)
      X(1) = PPOS(1)
      Y(1) = PPOS(2)
      VPOS(1) = S
      VPOS(2) = T
      CALL MMULT(OTR,VPOS,PPOS)
      X(2) = PPOS(1)
      Y(2) = PPOS(2)
      IF(IW .EQ.2 .OR. IW .EQ. 1) THEN
       CALL WCUTS(X,Y,U,V)
       IF(U(1).EQ.-50.0 .OR. U(2).EQ.-50.0)  RETURN
       DO 10 I = 1,2
         X(I) = U(I)
         Y(I) = V(I)
10     CONTINUE
      END IF
      DO 20 I = 1,2
       M(I) = NINT(767/210.0 * X(I) )
       N(I) = NINT(1023 - (1023/275.0 *Y(I)) )
20    CONTINUE
      CALL LINEXX(M(1),N(1),M(2),N(2))
      RETURN
      END



      SUBROUTINE LINBY2(X,Y)
      REAL X,Y,V1,V2,VPOS
      LOGICAL FLAG,LOGIC
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      V1 = VPOS(1) + X
      V2 = VPOS(2) + Y
      CALL LINTO2(V1,V2)
      RETURN
      END



      SUBROUTINE MOVTO2(X,Y)
      REAL X,Y,VPOS
```

```
LOGICAL FLAG,LOGIC
COMMON/SPCD/VPOS(3),OTR(3,3)
COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
COMMON/SYGIN/FD
VPOS(1) = X
VPOS(2) = Y
RETURN
END


SUBROUTINE MOVBY2(X,Y)
REAL X,Y,VPOS
LOGICAL FLAG,LOGIC
COMMON/SPCD/VPOS(3),OTR(3,3)
COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
COMMON/SYGIN/FD
VPOS(1) = VPOS(1) + X
VPOS(2) = VPOS(2) + Y
RETURN
END


SUBROUTINE DRAW2(X,Y,IABS,IVIS)
REAL X,Y
LOGICAL FLAG,LOGIC
INTEGER IABS,IVIS
COMMON/SYGIN/FD
COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
 IF(IABS.EQ.0 .AND.IVIS.EQ.0) THEN
  CALL MOVBY2(X,Y)
 ELSE IF(IABS.EQ.0 .AND.IVIS.EQ.1) THEN
  CALL LINBY2(X,Y)
 ELSE IF(IABS.EQ.1 .AND.IVIS.EQ.0) THEN
  CALL MOVTO2(X,Y)
 ELSE IF(IABS.EQ.1 .AND.IVIS.EQ.1) THEN
  CALL LINTO2(X,Y)
 ELSE
  PRINT *,'ERROR IN IABS OR IVIS IN ROUTINE DRAW2'
 END IF
RETURN
END


INTEGER FUNCTION FINAL(S)
CHARACTER S*(*)
INTEGER COUNT
COMMON/SYGIN/FD
FINAL = 0
DO 15 COUNT =LEN(S),1,-1
IF(S(COUNT:COUNT).NE.' ') THEN
FINAL=COUNT
RETURN
```

```
      ENDIF
   15 CONTINUE
      RETURN
      END

      SUBROUTINE CHAHOL( STRING )
      CHARACTER STRING * ( * )
      REAL TEMP ( 3, 3)
      INTEGER X,Y,I
      INTEGER OFFSET,MAXBYT,XMAX,WSTRNG,LAST,FINAL
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON / SPCD / VPOS(3),OTR(3,3)
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
      COMMON/SYGIN/FD

      CALL MMULT(OTR,VPOS,PPOS)
      RX = PPOS( 1 )
      RY = PPOS( 2 )
      OFFSET=0
      XMAX=767
      X = INT( 767 /210.* RX )
      Y = INT(1023- 1023 / 275.* RY )

      LAST=FINAL(STRING)
      MAXBYT = LAST
      I=WSTRNG(FD,X,Y,STRING,LAST,OFFSET,MAXBYT,XMAX)
      Y = 1023 - Y
      VPOS ( 1 ) = 210. * X / ( 767 )
      VPOS ( 2 ) = 275. * Y /  ( 1023  )
      VPOS ( 3 ) = 1
      DO 20, I= 1, 3
       DO 15, J = 1, 3
        TEMP ( I, J ) = OTR ( I, J )
   15  CONTINUE
   20 CONTINUE
      CALL MATINV ( TEMP )
      RETURN
      END


      SUBROUTINE ARCTOL(T)
      REAL T,TOL
      LOGICAL FLAG,LOGIC
      COMMON/SYGIN/FD
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      IF(T.EQ.0.0) THEN
       TOL = 0.05
       LOGIC = .FALSE.
      ELSE
       TOL = T
      END IF
      RETURN
      END
```

```
SUBROUTINE ARCINC(N)
INTEGER N,INC
LOGICAL FLAG,LOGIC
COMMON/SYGIN/FD
COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
IF(N.EQ.0 ) THEN
 INC = 0
 LOGIC = .FALSE.
ELSE
 INC = N
 LOGIC = .TRUE.
END IF
RETURN
END


SUBROUTINE ARCENQ(I,NINCS,TOL)
REAL TOL
INTEGER I,NINCS
LOGICAL FLAG,LOGIC
COMMON/ENQY/FLAG,R,ATOL,INC,LOGIC,ITYPE
COMMON/SYGIN/FD
I = ITYPE
NINCS = INC
TOL = ATOL
RETURN
END


SUBROUTINE IRCBY2(DXC,DYC,DXE,DYE,ISENSE)
REAL DXC,DYC,DXE,DYE,XC,YC,XE,YE
INTEGER ISENSE
LOGICAL FLAG,LOGIC
COMMON/SPCD/VPOS(3),OTR(3,3)
COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
COMMON/SYGIN/FD
XC = VPOS(1) + DXC
YC = VPOS(2) + DYC
XE = VPOS(1) + DXE
YE = VPOS(2) + DYE
CALL IRCTO2(XC,YC,XE,YE,ISENSE)
RETURN
END


SUBROUTINE IRCTO2(XC,YC,XE,YE,ISENSE)
REAL XC,YC,XE,YE,AX,AY,AL,X,Y,RAD
INTEGER ISENSE
LOGICAL FLAG,LOGIC
COMMON/SPCD/VPOS(3),OTR(3,3)
COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
COMMON/SYGIN/FD
RAD = SQRT( (XC-VPOS(1))**2 + (YC-VPOS(2))**2 )
AX = XE - XC
```

```
      AY = YE - YC
      AL = SQRT( AX**2 + AY**2 )
      X  = RAD * AX/AL
      Y  = RAD * AY/AL
      CALL MOVTO2(X,Y)
      RETURN
      END


      SUBROUTINE POLBY2(DXARR,DYARR,NPTS)
      REAL DXARR(NPTS),DYARR(NPTS)
      INTEGER I
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      DO 10 I = 1,NPTS
       CALL LINBY2(DXARR(I),DYARR(I))
10    CONTINUE
      RETURN
      END



      SUBROUTINE POLTO2(XARR,YARR,NPTS)
      REAL XARR(NPTS),YARR(NPTS)
      INTEGER I
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      DO 10 I = 1,NPTS
       CALL LINTO2(XARR(I),YARR(I))
10    CONTINUE
      RETURN
      END


      SUBROUTINE WINDO2(X1,X2,Y1,Y2)
      REAL X1,X2,Y1,Y2
      INTEGER IW
      LOGICAL FLAG,LOGIC
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      XMIN = X1
      XMAX = X2
      YMIN = Y1
      YMAX = Y2
      RETURN
      END


      SUBROUTINE WIND(I)
      INTEGER IW,M(4),N(4)
      LOGICAL FLAG,LOGIC
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
```

```
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      IW = I
      IF(IW .EQ. 2) THEN
       XMIN = 0.0
       XMAX = 210.0
       YMIN = 0.0
       YMAX = 275.0
      END IF
      IF(IW.EQ.2 .OR. IW.EQ.1 ) THEN
       M(1) = NINT(767/210.0*XMIN)
       N(1) = NINT(1023 - (1023/275.0*YMIN))
       M(2) = NINT(767/210.0*XMAX)
       N(2) = N(1)
       M(3) = M(2)
       N(3) = NINT(1023 - (1023/275.0*YMAX))
       M(4) = M(1)
       N(4) = N(3)
        DO 10 I = 1,3
         CALL LINEXX(M(I),N(I),M(I+1),N(I+1))
10      CONTINUE
       CALL LINEXX( M(4),N(4),M(1),N(1) )
      END IF
      RETURN
      END


      SUBROUTINE WCUTS(X,Y,U,V)
C     MAKES USE OF EQUATION OF STRAIGHT LINE :
C          X = X1 + DELTA * (X2-X1)
C          Y = Y1 + DELTA * (Y2-Y1)
C     TO FIND WHERE INTEDED LINE CUTS THE WINDOW.
C     FUNCTIONS DEFINED ARE : DXY AND FNXY
C
      REAL X(2),Y(2),U(2),V(2),A(2),B(2),DELTA,HOR,VER
      REAL XMIN,XMAX,TMIN,TMAX
      INTEGER I,J,K
      LOGICAL FLAG,LOGIC
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      DXY(XY1,XY2,XY) =    (XY - XY1)/(XY2 - XY1)
      FNXY(XY1,XY2,DELTA) = XY1 + DELTA * (XY2 - XY1)
       IF(X(1).GE.XMIN .AND. X(1).LE.XMAX  .AND.
     !  Y(1).GE.YMIN .AND. Y(1).LE.YMAX ) THEN
         IF(X(2).GE.XMIN .AND. X(2).LE.XMAX  .AND.
     !    Y(2).GE.YMIN .AND.Y(2).LE.YMAX ) THEN
           DO 3 I = 1,2
            U(I) = X(I)
            V(I) = Y(I)
 3         CONTINUE
           RETURN
         END IF
       END IF
C      CASE WHERE ONE OR BOTH POINTS LIE OUTSIDE THE WINDOW.
```

```
      J = 1
      I = 1
      HOR = YMIN
      VER = XMIN
      DO 4 K = 1,2
       A(K) = -50.0
       B(K) = -50.0
    4 CONTINUE
    5  IF(Y(1).NE.Y(2)) THEN
        DELTA = DXY( Y(1),Y(2),HOR)
        TEMP = FNXY( X(1),X(2),DELTA)
         IF(DELTA.GE.0.0 .AND. DELTA.LE.1.0  .AND.
     !    TEMP.GE.XMIN .AND.TEMP.LE.XMAX) THEN
             A(J) = TEMP
             B(J) = HOR
             HOR  = YMAX
              J  = 2
         ELSE
          HOR = YMAX
         END IF
       ELSE
        HOR = YMAX
       END IF
       IF( X(1).NE.X(2) ) THEN
        DELTA = DXY( X(1),X(2),VER )
        TEMP = FNXY( Y(1),Y(2),DELTA)
         IF(DELTA.GE.0.0 .AND. DELTA.LE.1.0  .AND.
     !    TEMP.GE.YMIN .AND.TEMP.LE.YMAX ) THEN
             A(J) = VER
             B(J) = TEMP
             VER  = XMAX
              J  = 2
         ELSE
          VER = XMAX
         END IF
       ELSE
        VER = XMAX
       END IF
      I = I + 1
      IF(I.NE.3) GOTO 5
C     TEST FOR NUMBER OF INTERSECTIONS.
      IF( X(1).GT.XMIN .AND. X(1).LT.XMAX .AND.
     ! Y(1).GT.YMIN .AND. Y(1).LT.YMAX ) THEN
       IF( X(2).LT.XMIN .OR. X(2).GT.XMAX .OR.
     !  Y(2).LT.YMIN .OR. Y(2).GT.YMAX ) THEN
        U(1) = X(1)
        V(1) = Y(1)
        U(2) = A(1)
        V(2) = B(1)
        RETURN
       END IF
      ELSE IF( X(1).LT.XMIN .OR.X(1).GT.XMAX .OR.
     ! Y(1).LT.YMIN .OR. Y(1).GT.XMAX ) THEN
       IF( X(2).GT.XMIN .AND. X(2).LT.XMAX .AND.
     !  Y(2).GT.YMIN .AND.Y(2).LT.YMAX ) THEN
```

```
        U(1) = A(1)
        V(1) = B(1)
        U(2) = X(2)
        V(2) = Y(2)
        RETURN
       END IF
      END IF
C     HAVING GOT THIS FAR, CONTINUE TO THE END
      SMIN = MIN( A(1),A(2) )
      TMIN = MIN( B(1),B(2) )
      SMAX = MAX( A(1),A(2) )
      TMAX = MAX( B(1),B(2) )
      IF( X(2).GE.X(1) .AND. Y(2).GE.Y(1) .OR.
     !  X(2).LE.X(1) .AND. Y(2).LE.Y(1) ) THEN
        IF( X(2).GT.X(1) .OR. Y(2).GT.Y(1) ) THEN
          U(1) = SMIN
          V(1) = TMIN
          U(2) = SMAX
          V(2) = TMAX
        ELSE
          U(1) = SMAX
          V(1) = TMAX
          U(2) = SMIN
          V(2) = TMIN
        END IF
      ELSE IF( X(2).GT.X(1) ) THEN
          U(1) = SMIN
          V(1) = TMAX
          U(2) = SMAX
          V(2) = TMIN
      ELSE
          U(1) = SMAX
          V(1) = TMIN
          U(2) = SMIN
          V(2) = TMAX
     END IF
     RETURN
     END


     SUBROUTINE RFRESH(PTR)
C     DOES MATRIX MULTIPLICATION, C = OTR*PTR
     REAL PTR(3,3),C(3,3)
     INTEGER I,J,K
     LOGICAL FLAG,LOGIC
     COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
     COMMON/SPCD/VPOS(3),OTR(3,3)
     COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
     COMMON/SYGIN/FD
     DO 20 I= 1,3
      DO 10 J = 1,3
       C(I,J) = 0.0
       DO 5 K = 1,3
        C(I,J) = C(I,J) + OTR(I,K) * PTR(K,J)
5       CONTINUE
```

```
10     CONTINUE
20    CONTINUE
      DO 100 I = 1,3
       DO 90 J = 1,3
        OTR(I,J) = C(I,J)
90     CONTINUE
100   CONTINUE
      RETURN
      END


      SUBROUTINE MATINV(A)
C     FINDS INVERSE OF MATRIX PTR AND RETURNS IT IN RTP
      REAL A(3,3),B(3,3),VAP(3)
      LOGICAL FLAG,LOGIC
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
C     DEFINE INDENTITY MATRIX OF SAME DIM. AS A(3,3)
      DO 10 I = 1,3
       DO 5 J = 1,3
        B(I,J) = 0.0
5      CONTINUE
      B(I,I) = 1.0
10    CONTINUE
      DO 4 I = 1,3
      DO 2 K = 1,3
      IF (K.EQ.I) GO TO 2
      CONST = -A(K,I)/A(I,I)
      DO 1 J =1,3
      A(K,J) = A(K,J) + CONST*A(I,J)
      B(K,J) = B(K,J) + CONST*B(I,J)
      IF (J.EQ.I) A(K,J) = 0.0
 1    CONTINUE
 2    CONTINUE
      CONST = A(I,I)
      DO 3 J = 1,3
      A(I,J) = A(I,J)/CONST
 3    B(I,J) = B(I,J)/CONST
      A(I,I) = 1.0
 4    CONTINUE
C     INVERSE OF MATRIX A IS IN MATRIX B
C     NOW FIND POSITION(VPOS), W.R.T. NEWLY DEFINED AXIS
      CALL MMULT(B,VPOS,VAP)
      DO 100 I = 1,3
       VPOS(I) = VAP(I)
100   CONTINUE
      RETURN
      END


      SUBROUTINE MMULT(OTR,VPOS,APOS)
      REAL OTR(3,3),VPOS(3),APOS(3)
      INTEGER I,J
      LOGICAL FLAG,LOGIC
```

```
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      DO 10 I = 1,3
       APOS(I) = 0.0
        DO 5 J = 1,3
         APOS(I) = APOS(I) + OTR(I,J) * VPOS(J)
5       CONTINUE
10    CONTINUE
      RETURN
      END


      SUBROUTINE WINENQ(IW,IB1,NB,BOUNDS)
      REAL BOUNDS(NB)
      INTEGER IW,IB1,I
      LOGICAL FLAG,LOGIC
      COMMON/PICC/PPOS(3),W(4),JW
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      IW = JW
      IF(IB1.EQ.0 .OR. NB.EQ.0 ) RETURN
      DO 5 I = 1,NB
       BOUNDS(I) = W(IB1)
       IB1 = IB1 + 1
       IF(IB1.GT.4) IB1 = 1
5     CONTINUE
      RETURN
      END


      SUBROUTINE TURNA4
      LOGICAL FLAG,LOGIC
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      CALL SHIFT2(0.0,275.0)
      CALL ROTAT2(-90.0)
      PERM1 = XMIN
      PERM2 = XMAX
      XMIN  = YMIN
      XMAX  = YMAX
      YMIN  = PERM1
      YMAX  = PERM2
      RETURN
      END


      SUBROUTINE UNITS(XMILS)
C     DEFINES RATIO OF SCALES AND PLACES IT IN COMMON.
      REAL R,XMILS
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      R = XMILS
```

```
      CALL SCALE(R)
      RETURN
      END


      SUBROUTINE POSPIC(X,Y)
C     RETURNS CURRENT PEN POSITION W.R.T. THE PICTURE COORDINATES.
      REAL X,Y
      LOGICAL FLAG,LOGIC
      COMMON/PICC/PPOS(3),XMIN,XMAX,YMIN,YMAX,IW
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      X = PPOS(1) * R
      Y = PPOS(2) * R
      RETURN
      END


      SUBROUTINE POSSPA(X,Y,Z)
C     RETURNS CURRENT PEN POSITION W.R.T. MOST RECENTLY DEFINED
C     SPACE COORDINATES.
      REAL X,Y,Z
      LOGICAL FLAG,LOGIC
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD

      X = VPOS(1) * R
      Y = VPOS(2) * R
      RETURN
      END


      SUBROUTINE DEVEND
      REAL R
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      FLAG = .FALSE.
      RETURN
      END


      SUBROUTINE M2INV(A)
C     FINDS INVERSE OF MATRIX PTR AND RETURNS IT IN RTP
      REAL A(2,2),B(2,2),VAP(2)
      LOGICAL FLAG,LOGIC
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
C     DEFINE INDENTITY MATRIX OF SAME DIM. AS A(2,2)
      DO 10 I = 1,2
       DO 5 J = 1,2
        B(I,J) = 0.0
5      CONTINUE
```

```
      B(I,I) = 1.0
10    CONTINUE
C
      DO 4 I = 1,2
      DO 2 K = 1,2
      IF (K.EQ.I) GO TO 2
      CONST = -A(K,I)/A(I,I)
      DO 1 J =1,2
      A(K,J) = A(K,J) + CONST*A(I,J)
      B(K,J) = B(K,J) + CONST*B(I,J)
      IF (J.EQ.I) A(K,J) = 0.0
    1 CONTINUE
    2 CONTINUE
      CONST = A(I,I)
      DO 3 J = 1,2
      A(I,J) = A(I,J)/CONST
    3 B(I,J) = B(I,J)/CONST
      A(I,I) = 1.0
    4 CONTINUE
C     INVERSE OF MATRIX A IS IN MATRIX B
C     NOW FIND POSITION(VPOS), W.R.T. NEWLY DEFINED AXIS
      CALL MMULT2(B,VPOS,VAP)
      DO 100 I = 1,2
      VPOS(I) = VAP(I)
100   CONTINUE
      RETURN
      END


      SUBROUTINE MMULT2(OTR,VPOS,APOS)
      REAL OTR(2,2),VPOS(2),APOS(2)
      INTEGER I,J
      LOGICAL FLAG,LOGIC
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      DO 10 I = 1,2
      APOS(I) = 0.0
        DO 5 J = 1,2
        APOS(I) = APOS(I) + OTR(I,J) * VPOS(J)
5       CONTINUE
10    CONTINUE
      RETURN
      END


      SUBROUTINE ARCBY2(DXC,DYC,DXE,DYE,ISENSE)
      REAL DXC,DYC,DXE,DYE,XC,YC,XE,YE
      INTEGER ISENSE
      LOGICAL FLAG,LOGIC
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
      XC = VPOS(1) + DXC
      YC = VPOS(2) + DYC
      XE = VPOS(1) + DXE
```

```
      YE = VPOS(2) + DYE
      CALL ARCTO2(XC,YC,XE,YE,ISENSE)
      VPOS(1) = XE - DXE
      VPOS(2) = YE - DYE
      RETURN
      END


      SUBROUTINE ARCTO2(XC,YC,XE,YE,ISENSE)
      REAL XC,YC,XE,YE,RAD,DX,DY,X,Y,THETA,ALPHA
      INTEGER ISENSE,N
      LOGICAL FLAG,LOGIC
      COMMON/SPCD/VPOS(3),OTR(3,3)
      COMMON/ENQY/FLAG,R,TOL,INC,LOGIC,ITYPE
      COMMON/SYGIN/FD
C     FIND RADIUS OF ARC
      RAD = SQRT( (XC-VPOS(1))**2 + (YC-VPOS(2))**2 )
      THETA = 2 * ACOS(1 - TOL/RAD/5.)
      L = 2 * (RAD -TOL) * TAN(THETA/2)
      DX = VPOS(1) - XC
      DY = VPOS(2) - YC
      N = 3.14159/THETA
      IF(ISENSE.EQ.1)THEN
      THETA = THETA
      ELSE
      THETA = -THETA
      ENDIF
      ALPHA = 0.0
      IF(YC.EQ.YE)THEN
      N = 2*N
      ELSE
      N = N
      ENDIF
      DO 10 I =1,N+1
      ALPHA = THETA + ALPHA
      X = DX*COS(ALPHA) - DY*SIN(ALPHA) + XC
      Y = DX*SIN(ALPHA) + DY*COS(ALPHA) + YC
      CALL LINTO2(X,Y)
 10   CONTINUE
      RETURN
      END

      SUBROUTINE COMAND
      COMMON/SYGIN/FD
      CALL PERQ
      CALL MOVTO2(154.,0.)
      CALL LINBY2(0.,36.)
      CALL LINBY2(56.,0.)
      CALL MOVBY2(0.,-18.)
      CALL LINBY2(-56.,0.)
      CALL MOVTO2(182.,0.)
      CALL LINBY2(0.,36.)
      CALL MOVTO2(185.,7.)
      CALL CHAHOL(' KEYS')
      CALL MOVTO2(158.,25.)
```

```
 CALL CHAHOL(' STOP')
 CALL MOVTO2(185.,25.)
 CALL CHAHOL(' CLEAR')
 CALL MOVTO2(155.,7.)
 CALL CHAHOL('PROBABILITY')
 RETURN
 END


SUBROUTINE CHAMOD
END


SUBROUTINE PICBEG(I)
END


SUBROUTINE PICEND
END

SUBROUTINE EVESET(I)
END

SUBROUTINE PICSEN(I,J)
END

SUBROUTINE EVENT(I)
END


SUBROUTINE EVEDEL(I)
END


SUBROUTINE HARCHA
END


SUBROUTINE CHASWI(I)
END


SUBROUTINE PENSEL(I,W,J)
END


SUBROUTINE CHASIZ(W,H)
END


SUBROUTINE CHAANG(A)
END

SUBROUTINE T4010
END
```

```
      SUBROUTINE CHAASC(I)

      CALL CHAHOL(CHAR(I))
      END


      SUBROUTINE CHAINT(I,J)
      END


      SUBROUTINE FLASH (UNIT)
      INTEGER UNIT
      WRITE(UNIT=UNIT,FMT='(1000(3A1))')(CHAR(27),'X',CHAR(33),I=1,170)
      RETURN
      END


#include <wuser.h>
 int wopen_(s)
 char *s;
 {
 char *s1;
 int fd;
 s1=s;
 fd=open(s1,2);
 return(fd);
 }
 int wlin_(fd,x1,y1,x2,y2)
 int *fd;
 int *x1,*y1,*x2,*y2;
 {
 int i,fd1;
 short xx1,yy1,xx2,yy2;
 struct LINCtl linargp;
 struct ClipCtl clipp;
 xx1=(*x1);
 yy1=(*y1);
 xx2=(*x2);
 yy2=(*y2);
 linargp.LINX1=xx1;
 linargp.LINY1=yy1;
 linargp.LINX2=xx2;
 linargp.LINY2=yy2;
 linargp.LINDstInc=1;
 linargp.LINDstBase=0;
 linargp.LINStyle=LDraw;
 fd1=(*fd);
 i=wline(fd1,&linargp,0);
 return(i);
 }
 int wclose_(fd)
 int *fd;
 {
 int i,fd1;
```

```
fd1=(*fd);
i=close(fd1);
return (i);
}
int wstrng_(fd,x,y,s,last,offset,maxbyte,xmax)
int *fd,*last;
int *x,*y,*offset,*maxbyte,*xmax;
char *s;
{
char str[90];
char *ps;
short xx,yy,offs,xxmax,xmb;
int fd1;
int i;
struct DBCtl dbargp;
struct ClipCtl clipp;
for(i=0;i<(*last);i++)
  str[i]=(*(s+i));
ps=(&str[0]);
fd1=(*fd);
xx=(*x);
offs=(*offset);
yy=(*y);
xxmax=(*xmax);
xmb=(*maxbyte);
dbargp.DBX=xx;
dbargp.DBByteOfset=offs;
dbargp.DBFunc=ROr;
dbargp.DBY=yy;
dbargp.DBMaxX=xxmax;
dbargp.DBMaxByte=xmb;
dbargp.DBSrcString=ps;
dbargp.DBScreen=0;
dbargp.DBFont=0;
dbargp.DBDstInc=1;
i=wdbyte(fd1,&dbargp,0);
return(i);
}
int wpuck_(fd,x1,y1,x2,y2,x3,y3,press)
int *fd,*x1,*y1,*x2,*y2,*x3,*y3,*press;
{
int i;
short button;
struct Wgrec buf;
do {
i=wgread(*fd,&buf);
}
while (buf.grButtons != 0 );
do {
i=wgread(*fd,&buf);

}
while (buf.grButtons == 0);
(*x1)=buf.grWTabX;
(*y1)=buf.grWTabY;
```

```
(*x2)=buf.grSTabX;
(*y2)=buf.grSTabY;
(*x3)=buf.grTTabX;
(*y3)=buf.grTTabY;
button=buf.grButtons;
if (button&WB_YELLOW){
    *press=0;
}
else if(button&WB_WHITE){
    *press=1;
}
else if(button&WB_BLUE){
     *press=2;
}
else if(button&WB_GREEN){
    *press=3;
}
else {*press=4;
}
return(i);
}
```

APPENDIX K : LISTING OF THE FTDRA SOURCE PROGRAMS

```
      SUBROUTINE INIT

CC    TO CREATE SET STRUCTURE AND TO INITIALIZE IT

      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      LOGICAL FULL
      LOGICAL NORMAL
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)

      DO 10 I=1,NOSETS
       STARTS(I) = -1
10    CONTINUE
      DO 20 J=1,NUMENT-1
       SETS(J,2) = J+1
20    CONTINUE
       SETS(NUMENT,2) = 0
       EMPTY = 1
       FINISH = 10000
       FULL =.FALSE.
      RETURN
      END



      SUBROUTINE NEWSET (NUMBER)

CC    TO INITIALIZE NEW SET AND TO RETURN ITS ENTRY NUMBER
CC    IN THE SET STRUCTURE

      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      LOGICAL FULL

      LOGICAL NORMAL
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)
      DO 10 I = 1,NOSETS
        IF (STARTS(I).EQ. -1)THEN
          NUMBER = I
          STARTS(I) = 0
          RETURN
        END IF
10    CONTINUE
       OPEN ( UNIT = 7, FILE='ERREPORT')
       REWIND (7)
       WRITE (7,*) 'EMPTY = ',EMPTY
       WRITE (7,*) 'FINISH = ', FINISH
       WRITE (7,*) 'STARTS ='
       WRITE (7, '(I5,I7)')  (IAJ,STARTS(IAJ),IAJ=1,1000)
       WRITE (7,*) 'SETS ='
        WRITE ( 7,'(I5, I7,I5, L3)') (IAJ,SETS(IAJ,1),SETS(IAJ,2),
     +    NORMAL(IAJ),IAJ=1,10000)
       CLOSE (7)
```

```
      CALL SETERR (' NO MORE SETS CAN BE CREATED')
      RETURN
      END



      SUBROUTINE ADDITM (ITEM ,SETNO ,STATE ,CLASS)

CC    TO ADD AN ITEM TO SET SETNO

      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      INTEGER ITEM ,SETNO ,NEXT ,LAST ,BEGIN ,OLD ,FOLO
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      LOGICAL FULL
      LOGICAL NORMAL ,STATE ,CLASS
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)



      IF (EMPTY.EQ.FINISH)THEN
        FULL = .TRUE.
      END IF

      IF (FULL)THEN
       OPEN ( UNIT = 7, FILE='ERREPORT')
       REWIND (7)
       WRITE (7,*) 'EMPTY = ',EMPTY
       WRITE (7,*) 'FINISH = ', FINISH
       WRITE (7,*) 'STARTS ='
       WRITE (7, '(I5,I7)')  (IAJ,STARTS(IAJ),IAJ=1,1000)
       WRITE (7,*) 'SETS ='
        WRITE ( 7,'(I5, I7,I5, L3)') (IAJ,SETS(IAJ,1),SETS(IAJ,2),
     +    NORMAL(IAJ),IAJ=1,10000)
       CLOSE (7)
       CALL SETERR (' THERE IS NO MORE ROOM FOR FURTHER SET
     + ITEMS')
      END IF

      IF (STARTS(SETNO).EQ.0)THEN
         STARTS(SETNO) = EMPTY
         EMPTY = SETS(EMPTY,2)
         SETS(STARTS(SETNO),1) = ITEM
         SETS(STARTS(SETNO),2) = 0
         NORMAL(STARTS(SETNO)) = STATE
         RETURN
      ELSEIF (ITEM.EQ.SETS(STARTS(SETNO),1))THEN
            RETURN
      ELSEIF (ITEM.LT.SETS(STARTS(SETNO),1))THEN
            IF (CLASS) GO TO 9
           BEGIN = STARTS(SETNO)
           STARTS(SETNO) = EMPTY
           EMPTY = SETS(EMPTY,2)
           SETS(STARTS(SETNO),1) = ITEM
           SETS(STARTS(SETNO),2) = BEGIN
```

```
        NORMAL(STARTS(SETNO)) = STATE
        RETURN
      END IF

 9    NEXT = STARTS(SETNO)
10      LAST = NEXT
        NEXT = SETS(NEXT,2)

          IF (NEXT.NE.0)THEN
            IF (ITEM.EQ.SETS(NEXT,1))THEN
              IF (NORMAL(NEXT).EQV.STATE)THEN
                RETURN
              ELSE
                CALL DELSET (SETNO)
                RETURN
              END IF
            ELSEIF (ITEM .LT.SETS(NEXT,1))THEN
              IF (CLASS) GO TO 10
              OLD = EMPTY
              FOLO = SETS(LAST,2)
              SETS(EMPTY,1) = ITEM
              SETS(LAST,2) = EMPTY
              NORMAL(EMPTY) = STATE
              EMPTY = SETS(EMPTY,2)
              SETS(OLD,2) = FOLO
             RETURN
            END IF
              GO TO 10
          ELSE

99              SETS(LAST,2) = EMPTY
              EMPTY = SETS(EMPTY,2)
              SETS(SETS(LAST,2),1) = ITEM
              NORMAL(SETS(LAST,2)) = STATE
              SETS(SETS(LAST,2),2) = 0
              RETURN
          END IF
      RETURN
      END



      SUBROUTINE DELSET (SETNO)

CC    TO DELETE SET SETNO

      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      INTEGER SETNO ,LAST ,NEXT
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      LOGICAL FULL
      LOGICAL NORMAL
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)
```

```
      LAST = STARTS(SETNO)

      IF (LAST.EQ.0)THEN
         STARTS(SETNO) = -1
        RETURN
      END IF

10     NEXT = SETS(LAST,2)
      IF (NEXT.EQ.0)THEN
       SETS(LAST,2) = EMPTY
       EMPTY = STARTS(SETNO)
       STARTS(SETNO) = -1
      RETURN
      END IF

       LAST = NEXT
       GO TO 10
      END



      SUBROUTINE DELITM (ITEM ,SETNO ,STATE)

CC   TO DELETE ONE OCCURENCE OF ITEM FROM SET SETNO IF IT
 CC   OCCURS

      INTEGER ITEM ,SETNO ,NEXT ,LAST ,FOLO
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      LOGICAL FULL
      LOGICAL NORMAL ,STATE
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)

       NEXT = STARTS(SETNO)
       LAST = -1
10   IF (NEXT.EQ.0)THEN
      RETURN
      END IF

      IF ((SETS(NEXT,1).EQ.ITEM).AND.(NORMAL(NEXT).EQV.STATE))THEN
       IF (LAST .EQ.-1)THEN
          FOLO = STARTS(SETNO)
          STARTS(SETNO) = SETS(NEXT,2)
        ELSE
          FOLO = SETS(LAST,2)
          SETS(LAST,2) = SETS(NEXT,2)
       END IF
         SETS(FOLO,2) = EMPTY
         EMPTY = FOLO
       RETURN
      END IF

       LAST = NEXT
       NEXT = SETS(NEXT,2)
```

```
      GO TO 10

      END



      SUBROUTINE INITTK (SETNO)

CC    TO INITIALIZE THE SEQUENTIAL REPORTING BACK BY SUB.
CC    TAKITM OF THE ITEMS IN SET SETNO

      INTEGER SETNO
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      LOGICAL FULL
      LOGICAL NORMAL
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +      FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)

       TAKNUM(SETNO) = STARTS(SETNO)

      RETURN
      END



      SUBROUTINE TAKITM (SETNO ,VALUE ,STATE)

CC    TO TAKE THE NEXT SEQUENTIAL ITEM FROM SET SETNO AND
CC     RETURN IT IN VALUE

      INTEGER SETNO ,VALUE
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      LOGICAL FULL
      LOGICAL NORMAL ,STATE

      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +      FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)

       VALUE = SETS(TAKNUM(SETNO),1)
       STATE = NORMAL(TAKNUM(SETNO))
       TAKNUM(SETNO) = SETS(TAKNUM(SETNO),2)
      RETURN
      END



      SUBROUTINE NUMSET (SETNO ,NUMBER)

CC    TO DETERMINE THE NUMBER OF ITEMS IN SET SETNO
```

```
      INTEGER SETNO ,NUMBER ,NEXT
      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      LOGICAL FULL
      LOGICAL NORMAL
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)

        NUMBER = 0
        NEXT = STARTS(SETNO)
10      IF (NEXT.LE.0)THEN
        RETURN
        END IF
          NUMBER = NUMBER + 1
          NEXT = SETS(NEXT,2)
          GO TO 10
      END


      SUBROUTINE CPYSET (SETOLD ,SETNEW)

CC    TO GENERATE A NEW SET SETNEW AND TO COPY SETOLD INTO IT

      INTEGER SETOLD ,SETNEW ,VALUE ,NUMBER
      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      LOGICAL FULL
      LOGICAL NORMAL ,STATE
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)

        CALL NEWSET (SETNEW)
        CALL NUMSET (SETOLD ,NUMBER)
        CALL INITTK (SETOLD)

      DO 10 I=1,NUMBER
        CALL TAKITM (SETOLD ,VALUE ,STATE)
        CALL ADDITM (VALUE ,SETNEW ,STATE, .FALSE.)
10    CONTINUE
      RETURN
      END



      SUBROUTINE SETERR (MESS)

CC    TO OUTPUT ERROR MESSAGE MESS

      CHARACTER MESS * (*)

      PRINT 900, MESS
900   FORMAT( // 2 (60('*') /),10X,A/2(60('*')/))
      STOP
      END
```

```
      LOGICAL FUNCTION EQUAL (SETNO1 ,NUM1 ,SETNO2 ,NUM2)

CC    TO RETURN TRUE IF SETS SETNO1 AND SETNO2 HAVE IDENTICAL
CC     CONTENTS; OTHERWISE RETURNS FALSE.

      INTEGER SETNO1 ,SETNO2 ,NUM1 ,NUM2 ,VAL1 ,VAL2
      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      LOGICAL FULL
      LOGICAL NORMAL ,STATE1 ,STATE2
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)

        CALL INITTK (SETNO1)
      DO 10 I =1,NUM1
        CALL TAKITM (SETNO1 ,VAL1 ,STATE1)
        CALL INITTK (SETNO2)
      DO 20 J=1, NUM2
        CALL TAKITM (SETNO2 ,VAL2 ,STATE2)
       IF (VAL1.EQ.VAL2)THEN
          IF (STATE1.EQV.STATE2)THEN
             EQUAL = .TRUE.
             GO TO 10
          END IF
          EQUAL = .FALSE.
          RETURN
       END IF
20    CONTINUE
      EQUAL = .FALSE.
      RETURN
10    CONTINUE
      EQUAL = .TRUE.
      RETURN
      END


      SUBROUTINE TYPSET (VALUE ,BASIC ,ORBAS ,OTHERS ,CLASS)

CC    TO DETERMINE WHETHER THE PRIMARY EVENTS OF THE EVENT IN
CC     ROW  NEXT OF CELLS FALL INTO THE CATEGORY OF PUTTING NEXT
CC     INTO THE  CATEGORY OF BASIC OR ORBAS OR OTHERS. IF NEXT IS
CC     A BASIC EVENT,  THEN VALUE IS PUT INTO SET BASIC.  IF NEXT
CC     IS AN OR GATE WITH ONLY BASIC EVENT INPUTS,NEXT IS PUT
CC    INTO ORBAS. IF NEXT IS ANYTHING ELSE, NEXT IS PUT INTO
CC     OTHERS.

      INTEGER CLR ,CLC ,SMR ,SMC
      PARAMETER ( NOSETS =1000 ,NUMENT =10000,
     +          CLR = 500 ,CLC = 10 ,SMR = 50 ,SMC = 20 )

      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
```

```
      INTEGER VALUE ,TYPE ,CELLS ,SYMPTS
      INTEGER NEXT ,BASIC ,ORBAS ,OTHERS ,ELEM ,NUMITM
      LOGICAL FULL
      LOGICAL NORMAL ,STATE ,CLASS
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)
      COMMON /SYRS/ CELLS(CLR,CLC),SYMPTS(SMR,SMC)

CC    TYPE = 1 IF BASIC ; 2 IF ORBAS ; 3 IF OTHERS

      TYPE = 1
      CALL NUMSET (VALUE ,NUMITM)
      CALL INITTK (VALUE)
      DO 15 I=1 , NUMITM
       CALL TAKITM (VALUE ,ELEM ,STATE)
       IF (CELLS(ELEM,2).EQ.-1 .OR.
     +     CELLS(ELEM,2) .EQ. +2)THEN
           CALL ADDITM (VALUE ,OTHERS ,STATE ,CLASS)
           TYPE = 3
           RETURN
       ELSEIF (CELLS(ELEM,2).EQ.+1.AND.(.NOT.STATE))THEN
           CALL ADDITM (VALUE ,OTHERS ,STATE ,CLASS)
           TYPE = 3
           RETURN
       ELSEIF (CELLS(ELEM,2).EQ.+1)THEN
           DO 20 J =4,CELLS(ELEM,3)+3
           NEXT = CELLS(ELEM,J)
             IF (CELLS(NEXT,2).NE.-2)THEN
               CALL ADDITM(VALUE ,OTHERS ,STATE ,CLASS)
              TYPE = 3
              RETURN
             END IF
20         CONTINUE
        TYPE = 2
       ELSEIF (CELLS(ELEM,2).EQ.0)THEN
           DO 30 M =4,CELLS(ELEM,3)+3
           NEXT = SYMPTS(CELLS(ELEM,M),1)
             IF (CELLS(NEXT,2).NE.-2)THEN
               CALL ADDITM (VALUE ,OTHERS ,STATE ,CLASS)
                TYPE = 3
              RETURN
             END IF
30         CONTINUE
        TYPE = 2
      ELSEIF (CELLS(ELEM,2).EQ.-2)THEN
           CONTINUE
      END IF
15    CONTINUE

     IF (TYPE.EQ.1)THEN
     CALL ADDITM (VALUE ,BASIC ,STATE ,CLASS)
     ELSE IF (TYPE.EQ.2)THEN
     CALL ADDITM (VALUE ,ORBAS ,STATE ,CLASS)
     END IF
     RETURN
```

```
      END


      SUBROUTINE STEP1 (TOP ,BASIC ,ORBAS ,OTHERS)

CC    TO CARRY OUT STEP 1 OF THE FTDRA ALGORITHM

      INTEGER CLR, CLC, SMR, SMC
      PARAMETER ( NOSETS =1000 ,NUMENT =10000,
     +          CLR = 500, CLC = 10, SMR = 50, SMC = 20 )
      INTEGER TOP ,BASIC ,ORBAS ,OTHERS ,VALUE ,NEXT
      INTEGER CELLS ,SYMPTS
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      LOGICAL FULL
      LOGICAL NORMAL
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +        FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)
      COMMON /SYRS/ CELLS(CLR,CLC),SYMPTS(SMR,SMC)
       IF ( CELLS(TOP,2).EQ.-1 )THEN
          CALL NEWSET (VALUE)
          DO 10 I=4,CELLS( TOP,3 )+3
            NEXT = CELLS(TOP,I)
            CALL ADDITM (NEXT ,VALUE ,.TRUE. ,.FALSE.)
10        CONTINUE
          CALL TYPSET (VALUE ,BASIC ,ORBAS ,OTHERS ,.TRUE.)
         RETURN
       ELSE IF ( CELLS(TOP,2).EQ.+1 )THEN
          DO 20 I=4,CELLS( TOP,3 )+3
            CALL NEWSET (VALUE)
            NEXT = CELLS(TOP,I)
            CALL ADDITM ( NEXT ,VALUE ,.TRUE. ,.FALSE.)
            CALL TYPSET ( VALUE ,BASIC ,ORBAS ,OTHERS ,.TRUE.)
20        CONTINUE
         RETURN
       ELSE IF ( CELLS(TOP,2).EQ.+2 )THEN
            CALL NEWSET ( VALUE )
            NEXT = CELLS(TOP,4)
            CALL ADDITM ( NEXT ,VALUE ,.TRUE. ,.FALSE.)
            CALL TYPSET ( VALUE ,BASIC ,ORBAS ,OTHERS ,.TRUE.)
           RETURN
       ELSE IF ( CELLS(TOP,2).EQ.-2 )THEN
            CALL NEWSET ( VALUE )
            CALL ADDITM ( TOP ,VALUE ,.TRUE. ,.FALSE.)
            CALL ADDITM ( VALUE ,BASIC ,.TRUE. ,.TRUE.)
           RETURN
       ELSE IF ( CELLS(TOP,2).EQ.0 )THEN
          DO 30 I =4,CELLS( TOP,3 )+3
            CALL NEWSET ( VALUE )
            NEXT = SYMPTS( CELLS( TOP,I ),1 )
            CALL ADDITM ( NEXT ,VALUE ,.TRUE. ,.FALSE.)
            CALL TYPSET ( VALUE ,BASIC ,ORBAS ,OTHERS ,.TRUE.)
30        CONTINUE
       END IF
       RETURN
```

```
      END


      SUBROUTINE STEP2 (BASIC ,ORBAS ,OTHERS)

      INTEGER CLR, CLC, SMR, SMC
      PARAMETER ( NOSETS =1000 ,NUMENT =10000,
     +          CLR = 500, CLC = 10, SMR = 50, SMC = 20 )

      INTEGER SET ,CELLS ,SYMPTS
      INTEGER NXTITM ,BASIC ,ORBAS ,OTHERS ,NUMITM ,NUMBER
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      LOGICAL FULL
      LOGICAL NORMAL ,STATE ,TESTP2
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)
      COMMON /SYRS/ CELLS(CLR,CLC),SYMPTS(SMR,SMC)

1       CALL NUMSET (OTHERS ,NUMBER)
       IF (NUMBER.EQ.0)THEN
         CALL DELSET (OTHERS)
        RETURN
       END IF
       CALL INITTK (OTHERS)

        DO 10 I =1,NUMBER
         CALL TAKITM (OTHERS ,SET ,STATE)
         CALL NUMSET (SET ,NUMITM)
         CALL INITTK (SET)

           DO 100 J =1,NUMITM
            CALL TAKITM (SET ,NXTITM ,STATE)
            IF (TESTP2(NXTITM))THEN
             CALL RESTP2 (SET ,NXTITM ,BASIC ,ORBAS ,OTHERS)
             GO TO 1
            END IF
100          CONTINUE
10      CONTINUE
      GO TO 1
      END



      LOGICAL FUNCTION TESTP2 (ITEM)

      INTEGER CLR, CLC, SMR, SMC
      PARAMETER ( NOSETS =1000 ,NUMENT =10000,
     +          CLR = 500, CLC = 10, SMR = 50, SMC = 20 )
      INTEGER J ,ITEM ,NXTITM ,CELLS ,SYMPTS
      LOGICAL TESTP2
      COMMON /SYRS/ CELLS(CLR,CLC),SYMPTS(SMR,SMC)

       IF ( CELLS(ITEM,2).EQ. -2 )THEN
         TESTP2 = .FALSE.
```

```
        RETURN
      ELSE IF ( CELLS(ITEM,2).EQ.+1)THEN
        DO 15 J =4,CELLS(ITEM,3 )+3
          NXTITM = CELLS(ITEM,J)
            IF ( CELLS(NXTITM,2).EQ.-2 )THEN
             CONTINUE
            ELSE
             TESTP2 = .TRUE.
            RETURN
            END IF
15          CONTINUE
          TESTP2 = .FALSE.
          RETURN
      ELSE IF (CELLS(ITEM,2).EQ.-1.OR.CELLS(ITEM,2).EQ.+2 ) THEN
        TESTP2 = .TRUE.
        RETURN
      ELSE IF ( CELLS(ITEM,2).EQ.0 )THEN
        DO 20 J =4,CELLS(ITEM,3 )+3
          NXTITM = SYMPTS(CELLS(ITEM,J),1)
           IF ( CELLS(NXTITM,2).EQ.-2 )THEN
            CONTINUE
           ELSE
            TESTP2 = .TRUE.
           RETURN
           END IF
20         CONTINUE
        TESTP2 = .FALSE.
      END IF
       RETURN
     END



     SUBROUTINE RESTP2 (SET ,NXTITM ,BASIC ,ORBAS ,OTHERS)


     INTEGER CLR, CLC, SMR, SMC
     PARAMETER ( NOSETS =1000 ,NUMENT =10000,
    +          CLR = 500, CLC = 10, SMR = 50, SMC = 20 )
     INTEGER SET ,NXTITM ,J ,CELLS ,SYMPTS
     INTEGER NEXT ,BASIC ,ORBAS ,OTHERS
     INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
     LOGICAL FULL
     LOGICAL NORMAL
     COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
    +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)
     COMMON /SYRS/ CELLS(CLR,CLC),SYMPTS(SMR,SMC)

      IF (CELLS(NXTITM,2).EQ.-1)THEN
       CALL DELITM (NXTITM ,SET ,.TRUE.)
       DO 50 J=4,CELLS(NXTITM,3 )+3
          NEXT = CELLS(NXTITM,J)
          CALL ADDITM (NEXT ,SET ,.TRUE. ,.FALSE.)
```

```
50     CONTINUE

       CALL DELITM (SET ,OTHERS ,.TRUE.)
       CALL TYPSET (SET ,BASIC ,ORBAS ,OTHERS ,.TRUE.)
     ELSE IF (CELLS(NXTITM,2).EQ.+1)THEN
       DO 60 J=4,CELLS(NXTITM,3 )+3
          NEXT = CELLS(NXTITM,J)
          CALL DELITM (NXTITM ,SET ,.TRUE.)
          CALL DELITM (SET ,OTHERS ,.TRUE.)
          CALL CPYSET (SET ,VALUE)
          CALL ADDITM (NEXT ,VALUE ,.TRUE. ,.FALSE.)
          CALL TYPSET (VALUE ,BASIC ,ORBAS ,OTHERS ,.TRUE.)
60     CONTINUE
       CALL DELSET (SET)
     ELSE IF (CELLS(NXTITM,2).EQ.0)THEN
       DO 70 J=4,CELLS(NXTITM,3 )+3
         NEXT = SYMPTS(CELLS(NXTITM,J),1)
          CALL DELITM (NXTITM ,SET ,.TRUE.)
          CALL DELITM (SET ,OTHERS ,.TRUE.)
          CALL CPYSET (SET ,VALUE)
          CALL ADDITM (NEXT ,VALUE ,.TRUE. ,.FALSE.)
          CALL TYPSET (VALUE ,BASIC ,ORBAS ,OTHERS ,.TRUE.)
70     CONTINUE
       CALL DELSET (SET)
     ELSE IF (CELLS(NXTITM,2).EQ.-2)THEN
       CONTINUE
     ELSE IF (CELLS(NXTITM,2).EQ.+2)THEN
       CALL DELITM (NXTITM ,SET ,.TRUE.)
          NEXT = CELLS(NXTITM,4)
          CALL ADDITM (NEXT ,SET ,.TRUE. ,.FALSE.)
          CALL TYPSET (SET ,BASIC ,ORBAS ,OTHERS ,.TRUE.)
     RETURN
     END IF
    RETURN
    END


    SUBROUTINE STEP3 (BASIC ,ORBAS)

    INTEGER BASIC ,ORBAS
    PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
    INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
    LOGICAL FULL
    LOGICAL NORMAL
    COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
   +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)

    CALL RSUP1 (BASIC)
    CALL RSUP1 (ORBAS)
    CALL RSUP2 (BASIC ,ORBAS)
    RETURN
    END


    SUBROUTINE RSUP1 (SETYPE)
```

```
      INTEGER SETYPE ,NUM ,VAL1 ,VAL2 ,VNUM1 ,VNUM2
      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      INTEGER VAL1 ,VNUM1 ,VAL2 ,VNUM2
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      LOGICAL FULL
      LOGICAL NORMAL ,STATE1 ,STATE2 ,EQUAL
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)


      K = 1
15    CALL NUMSET (SETYPE ,NUM)
      IF (K.GE.NUM)THEN
       RETURN
      END IF
      CALL INITTK (SETYPE)
        DO 50 I=1 ,K
         CALL TAKITM (SETYPE, VAL1, STATE1)
50      CONTINUE
        CALL NUMSET (VAL1, VNUM1)
        DO 100 J=K+1, NUM
         CALL TAKITM (SETYPE, VAL2, STATE2)
         CALL NUMSET (VAL2, VNUM2)
         IF (VNUM1.LE.VNUM2) THEN
           IF(EQUAL (VAL1,VNUM1,VAL2,VNUM2))THEN
             CALL DELSET(VAL2)
             CALL DELITM (VAL2, SETYPE, STATE2)
             GO TO 100
           END IF
         ELSE
           IF (EQUAL(VAL2,VNUM2,VAL1,VNUM1))THEN
             CALL DELSET (VAL1)
             CALL DELITM (VAL1, SETYPE, STATE1)
             K = I
             GO TO 15
           END IF
         END IF
100     CONTINUE
        K = K+1
        GO TO 15
      END


      SUBROUTINE RSUP2(SETYP1 ,SETYP2)

      INTEGER SETYP1 ,SETYP2,NUM1 ,NUM2 ,VNUM1 ,VNUM2, VAL1 ,VAL2
      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      LOGICAL FULL
      LOGICAL NORMAL ,STATE1 ,STATE2 ,EQUAL
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)

      CALL NUMSET (SETYP1 ,NUM1)
```

```
      CALL NUMSET (SETYP2 ,NUM2)
      IF (NUM1.EQ.0.OR.NUM2.EQ.0)THEN
       RETURN
      END IF
      M = 1
      CALL INITTK (SETYP1)
 90   IF (M.NE.1)THEN
        CALL INITTK (SETYP1)
        DO 50 K=1,M-1
        CALL TAKITM (SETYP1, VAL1, STATE1)
 50      CONTINUE
      END IF
      N = 1
      DO 100 I=M ,NUM1
        CALL TAKITM (SETYP1 ,VAL1 ,STATE1)
        CALL NUMSET (VAL1 ,VNUM1)
        CALL INITTK (SETYP2)
110      IF (N.NE.1)THEN
          CALL INITTK (SETYP2)
          DO 60 L=1,N-1
            CALL TAKITM (SETYP2, VAL2, STATE2)
60          CONTINUE
        END IF
        DO 200 J=N, NUM2
          CALL TAKITM (SETYP2 ,VAL2 ,STATE2)
          CALL NUMSET (VAL2 ,VNUM2)
          IF (VNUM1.LE.VNUM2)THEN
            IF(EQUAL(VAL1,VNUM1,VAL2,VNUM2))THEN
              CALL DELSET(VAL2)
              CALL DELITM(VAL2,SETYP2,STATE2)
              NUM2 = NUM2 - 1
              N = J
              GO TO 110
            END IF
          ELSE
            IF(EQUAL(VAL2,VNUM2,VAL1,VNUM1))THEN
              CALL DELSET (VAL1)
              CALL DELITM (VAL1,SETYP1,STATE1)
              NUM1 = NUM1 - 1
              M = I
              GO TO 90
            END IF
          END IF
200     CONTINUE
100   CONTINUE
      RETURN
      END


       SUBROUTINE STEP4 (BASIC ,ORBAS)

      INTEGER BASIC ,ORBAS
      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
```

```
      LOGICAL FULL ,NORMAL
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)

      CALL COLCT (BASIC ,ORBAS)
      CALL DISJON (BASIC)
      RETURN
      END


       SUBROUTINE COLCT (BASIC ,ORBAS)

      INTEGER BASIC ,ORBAS ,NUMORB ,VALUE
      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      LOGICAL FULL
      LOGICAL NORMAL ,STATE
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)


      CALL NUMSET (ORBAS ,NUMORB)
       IF (NUMORB.NE.0)THEN
         CALL INITTK(ORBAS)
         DO 20 J=1,NUMORB
          CALL TAKITM (ORBAS ,VALUE ,STATE)
          CALL ADDITM (VALUE ,BASIC ,STATE ,.TRUE.)
20       CONTINUE
       END IF
      CALL DELSET (ORBAS)
      RETURN
      END


      SUBROUTINE DISJON (BASIC)

      INTEGER BASIC ,NUMG ,ACUMM ,VAL1 ,VAL2 ,ITM1 ,ITM2
      INTEGER VNUM1 ,VNUM2
      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      LOGICAL FULL
      LOGICAL NORMAL ,STATE ,STATE1 ,STATE2 ,ACT
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)


      K = 1
20    CALL NUMSET (BASIC ,NUMG)
      ACT = .FALSE.
      IF (K.LE.NUMG-1)THEN
        CALL INITTK (BASIC)
        DO 10 I=1,K
          CALL TAKITM (BASIC ,VAL1 ,STATE)
10      CONTINUE
      ELSE
```

```
      RETURN
    END IF
    CALL NUMSET (VAL1 ,VNUM1)
    DO 100 M=1+K ,NUMG
      CALL TAKITM (BASIC ,VAL2 ,STATE)
      CALL NUMSET (VAL2 ,VNUM2)
      CALL NEWSET (ACUMM)
      CALL INITTK (VAL1)
     DO 40 J= 1,VNUM1
       CALL TAKITM (VAL1 ,ITM1 ,STATE1)
       CALL INITTK (VAL2)
       DO 50 L=1, VNUM2
         CALL TAKITM (VAL2 ,ITM2 ,STATE2)
         IF ((ITM1.EQ.ITM2).AND.(STATE1.EQV.STATE2))THEN
           GO TO 40
         ELSEIF ((ITM1.EQ.ITM2).AND.(STATE1.NEQV.STATE2))THEN
           CALL DELSET (ACUMM)
           GO TO 100
         END IF
 50    CONTINUE
       CALL ADDITM (ITM1 ,ACUMM ,STATE1 ,.FALSE.)
 40  CONTINUE
       CALL MAKDIS (ACUMM ,VAL2 ,BASIC ,ACT)
       CALL DELSET (ACUMM)
100  CONTINUE
     IF (ACT)THEN
      CALL RSUP1 (BASIC)
    END IF
    K = K+1
    GO TO 20
    END


    SUBROUTINE MAKDIS (ACUMM ,NVAL2 ,BASIC ,ACT)

    INTEGER CLR, CLC, SMR, SMC
    PARAMETER ( NOSETS =1000 ,NUMENT =10000,
   +          CLR = 500 ,CLC = 10 ,SMR = 50 ,SMC = 20 )
    INTEGER BASIC ,NVAL2 ,ACUMM ,NUMAC ,LOCAL ,JASS ,NEXT
    INTEGER CELLS ,SYMPTS
    INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
    LOGICAL FULL
    LOGICAL NORMAL ,STATE ,STATE1 ,ACT
    COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
   +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)
    COMMON /SYRS/ CELLS(CLR,CLC),SYMPTS(SMR,SMC)

      CALL NUMSET (ACUMM ,NUMAC)
      CALL INITTK (ACUMM)
      IF (NUMAC.EQ.1)THEN
       CALL TAKITM (ACUMM ,LOCAL ,STATE)
       IF (CELLS(LOCAL,2).EQ.1)THEN
         DO 10 I = 4,CELLS( LOCAL,3 )+3
           NEXT = CELLS(LOCAL,I)
           CALL ADDITM (NEXT ,NVAL2 ,.FALSE. ,.FALSE.)
```

```
10        CONTINUE
        ELSE
          CALL ADDITM (LOCAL ,NVAL2 ,.FALSE. ,.FALSE.)
        END IF
        RETURN
      ELSE
        DO 20 M=1,NUMAC
          CALL TAKITM (ACUMM ,LOCAL ,STATE)
          CALL CPYSET (NVAL2 ,JASS)
          IF (CELLS(LOCAL,2).EQ.1)THEN
            DO 15 J = 4,CELLS(LOCAL,3)+3
              NEXT = CELLS(LOCAL,J)
              CALL ADDITM (NEXT ,NVAL2 ,.FALSE. ,.FALSE.)
15          CONTINUE
            CALL ADDITM (LOCAL ,JASS ,.TRUE. ,.FALSE.)
            NVAL2 = JASS
          ELSE
            STATE1 = .NOT.STATE
            CALL ADDITM (LOCAL ,NVAL2 ,STATE1 ,.FALSE.)
            CALL ADDITM (LOCAL ,JASS ,STATE ,.FALSE.)
            NVAL2 = JASS
          END IF
          CALL ADDITM (NVAL2 ,BASIC ,.TRUE. ,.TRUE.)
20        CONTINUE
        CALL DELITM (NVAL2 ,BASIC ,.TRUE.)
        CALL DELSET (NVAL2)
        ACT = .TRUE.
      END IF
      RETURN
      END


    SUBROUTINE STEP5 (GENRAL ,GROUP)

    INTEGER GENRAL ,GROUP
    INTEGER CLR, CLC, SMR, SMC
    PARAMETER ( NOSETS =1000 ,NUMENT =10000,
   +          CLR = 500, CLC = 10, SMR = 50, SMC = 20 )
    INTEGER CELLS ,SYMPTS
    INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
    LOGICAL FULL
    LOGICAL NORMAL
    COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
   +        FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)
    COMMON /SYRS/ CELLS(CLR,CLC),SYMPTS(SMR,SMC)

     CALL RESOLV (GENRAL ,GROUP)
     CALL COLSET (GROUP ,GENRAL)
     CALL RSUP1(GENRAL)
     CALL DISJON (GENRAL)
     RETURN
     END
```

```
      SUBROUTINE RESOLV (GENRAL ,GROUP)

      INTEGER GENRAL ,GROUP ,BREED ,SETNUM ,ITMG ,FINE ,NEXT
      INTEGER NUMITM ,PART ,NUMPAR ,LOOM ,LOCAL, LOCNUM ,ELASP
      INTEGER CLR, CLC, SMR, SMC
      PARAMETER ( NOSETS =1000 ,NUMENT =10000,
     +          CLR = 500, CLC = 10, SMR = 50, SMC = 20 )
      INTEGER CELLS ,SYMPTS
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      LOGICAL FULL
      LOGICAL NORMAL ,STATE ,GATE ,PASS ,SAME
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)
      COMMON /SYRS/ CELLS(CLR,CLC),SYMPTS(SMR,SMC)
      KK = 1
      CALL NEWSET (GROUP)
      CALL NUMSET (GENRAL ,SETNUM)
    9 CALL INITTK (GENRAL)
      IF (KK.GT.SETNUM)THEN
        RETURN
      END IF
      CALL NEWSET (BREED)
      DO 33 I=1,KK
        CALL TAKITM (GENRAL ,ITMG ,STATE)
   33 CONTINUE
      M = 0
      GATE = .FALSE.
      CALL NUMSET (ITMG ,NUMITM)
      CALL INITTK (ITMG)
      DO 22 J = 1, NUMITM
        CALL TAKITM (ITMG ,FINE ,STATE)
        IF (CELLS(FINE,2).EQ.+1)THEN
          GATE = .TRUE.
          CALL ADDITM (FINE ,BREED ,.TRUE. ,.FALSE.)
          CALL DELITM (FINE ,ITMG ,STATE)
          M = M+1
        END IF
   22 CONTINUE
      IF (GATE)THEN
        CALL NEWSET(PART)
        CALL INITTK (BREED)
        SAME = .FALSE.
        DO 11 K = 1,M
          CALL TAKITM (BREED ,LOOM ,STATE)
          CALL NUMSET(PART, NUMPAR)
          IF (NUMPAR.EQ.0)THEN
            CALL CHECK (LOOM ,ITMG ,NUMITM-M ,PASS)
            IF (PASS)THEN
              GO TO 11
            END IF
           DO 44 N=4,CELLS(LOOM,3)+3
             NEXT = CELLS(LOOM,N)
             CALL CPYSET (ITMG, LOCAL)
             CALL ADDITM (NEXT ,LOCAL ,.TRUE. ,.FALSE.)
             CALL ADDITM (LOCAL ,PART ,.TRUE. ,.TRUE.)
```

```
44          CONTINUE
          CALL DELITM (LOOM ,BREED ,STATE)
            SAME = .TRUE.
          ELSE
            CALL INITTK(PART)
            DO 50 L=1,NUMPAR
              CALL TAKITM(PART ,LOCAL ,STATE)
              CALL NUMSET(LOCAL ,LOCNUM)
              CALL CHECK (LOOM ,LOCAL ,LOCNUM ,PASS)
            IF (PASS)THEN
              GO TO 50
            END IF
             DO 55 NN=4,CELLS(LOOM,3)+3
                NEXT = CELLS(LOOM,NN)
                CALL CPYSET (LOCAL ,ELASP)
                CALL ADDITM (NEXT ,ELASP ,.TRUE. ,.FALSE.)
                CALL ADDITM (ELASP ,PART ,.TRUE. ,.TRUE.)
55           CONTINUE
            CALL DELSET(LOCAL)
            CALL DELITM (LOCAL ,PART ,STATE)
50          CONTINUE
           SAME = .TRUE.
         END IF
11     CONTINUE
       IF(SAME)THEN
         CALL DELSET (ITMG)
         CALL DELITM (ITMG ,GENRAL ,STATE)
         SETNUM = SETNUM-1
         CALL ADDITM (PART ,GROUP ,.TRUE. ,.TRUE.)
         KK = KK
       ELSE
         CALL DELSET (PART)
         KK = KK+1
       END IF
       CALL DELSET (BREED)
       GO TO 9
     ELSE
       CALL DELSET (BREED)
       KK = KK+1
       GO TO 9
     END IF
     END


     SUBROUTINE CHECK (GATE ,SETT ,STNUM ,PASS)

     INTEGER GATE ,SETT ,STNUM ,NEXT ,SLIM
     INTEGER CLR, CLC, SMR, SMC
     PARAMETER ( NOSETS =1000 ,NUMENT =10000,
    +          CLR = 500, CLC = 10, SMR = 50, SMC = 20 )
     INTEGER CELLS ,SYMPTS
     INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
     LOGICAL FULL
     LOGICAL NORMAL ,STATE1,PASS
     COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
```

```
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)
      COMMON /SYRS/ CELLS(CLR,CLC),SYMPTS(SMR,SMC)

        PASS = .FALSE.
        DO 44 N=4,CELLS(GATE,3)+3
        NEXT = CELLS(GATE,N)
        CALL INITTK(SETT)
        DO 40 NN=1,STNUM
          CALL TAKITM (SETT ,SLIM ,STATE1)
          IF((NEXT.EQ.SLIM).AND.(STATE1.EQV.(.TRUE.)))THEN
           PASS = .TRUE.
           RETURN
          END IF
   40     CONTINUE
   44     CONTINUE
        RETURN
        END



       SUBROUTINE COLSET (SET1 ,SET2)

      INTEGER SET1 ,SET2 ,NUM1 ,VAL1 ,NUMVAL ,EXAC
      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      LOGICAL FULL
      LOGICAL NORMAL ,STATE ,STATE1
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)

      CALL NUMSET (SET1 ,NUM1)
       IF (NUM1.NE.0)THEN
         CALL INITTK(SET1)
         DO 20 J=1,NUM1
          CALL TAKITM (SET1 ,VAL1 ,STATE)
          CALL NUMSET (VAL1 ,NUMVAL)
          CALL INITTK(VAL1)
          DO 10 I=1, NUMVAL
            CALL TAKITM(VAL1 ,EXAC ,STATE1)
            CALL ADDITM (EXAC ,SET2 ,STATE1 ,.TRUE.)
   10     CONTINUE
          CALL DELSET (VAL1)
   20    CONTINUE
       END IF
      CALL DELSET (SET1)
      RETURN
      END



      SUBROUTINE REPORT

      INTEGER NEXT ,LAST
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
```

```
      PARAMETER ( NOSETS =1000 ,NUMENT =10000 )
      LOGICAL USED ( NUMENT )
      LOGICAL FULL
      LOGICAL NORMAL
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)

      DO 3, I= 1, NUMENT
       USED ( I ) = .TRUE.
    3 CONTINUE
      NEXT = EMPTY
    5 USED ( NEXT ) = .FALSE.
      IF ( SETS ( NEXT, 2 ) .EQ. 0 ) THEN
       GO TO 7
      ELSE
       LAST = NEXT
       NEXT = SETS ( LAST, 2 )
       GO TO 5
      END IF
    7 OPEN (1,FILE='RESULT',FORM='FORMATTED')
      REWIND(1)
      WRITE(1,*)'     ** SETS ENTRIES **'
      WRITE(1,*)'     ------------------'
      WRITE(1,*)
      WRITE(1,*)'SERIAL NO. ','SET NO.    ','START NO.'
      WRITE(1,*)'---------- ','-------    ','---------'
      J=0
      DO 150, I= 1, NOSETS
       IF ( STARTS ( I ) .NE. -1 ) THEN
         J=J+1
      WRITE(1,FMT=31)J ,I , STARTS ( I )
   31 FORMAT(4X, I4, 9X, I4, 8X, I7)
      END IF
  150 CONTINUE
      J=0
      WRITE(1,*)
      WRITE(1,*) '         ** SETS ELEMENTS **'
      WRITE(1,*) '         ------------------'
      WRITE(1,*)
      WRITE(1,*)'SERIAL NO.',' SET ROW ','SET ELEM.','NEXT ROW ','STATE'
      WRITE(1,*)'----------',' ------- ','---------','---------','-----'
      DO 200, I= 1, NUMENT
       IF ( USED ( I ) ) THEN
        J = J+1
      WRITE(1,FMT=33)J,I,SETS(I,1),SETS(I,2),NORMAL(I)
   33 FORMAT(5(5X,I4))
       END IF
  200 CONTINUE
      WRITE(1,*)
      WRITE(1,*)
      WRITE(1,*)'EMPTY = ', EMPTY
      WRITE(1,*) 'FINISH = ', FINISH
      WRITE(1,*) ' FULL = ', FULL
      END
```

```
      SUBROUTINE RLIABL(TOP)

      INTEGER CLR, CLC, SMR, SMC
      PARAMETER ( NOSETS =500 ,NUMENT =10000,
     +          CLR = 500, CLC = 10, SMR = 50, SMC = 20 )
      INTEGER TOP ,BASIC ,ORBAS ,OTHERS ,GROUP
      INTEGER CELLS,SYMPTS
      INTEGER SETS ,EMPTY ,FINISH ,STARTS ,TAKNUM
      LOGICAL FULL
      LOGICAL NORMAL
      COMMON / AJSETS / STARTS(NOSETS) ,SETS(NUMENT,2) ,EMPTY ,
     +       FINISH ,FULL ,TAKNUM(NOSETS) ,NORMAL(NUMENT)
      COMMON /SYRS/ CELLS(CLR,CLC),SYMPTS(SMR,SMC)

      OPEN (19, FILE='SHOW.W')
      CALL INIT
      CALL NEWSET ( BASIC )
      CALL NEWSET ( ORBAS )
      CALL NEWSET ( OTHERS )
      CALL STEP1 (TOP ,BASIC ,ORBAS ,OTHERS)
      CALL STEP2 (BASIC ,ORBAS ,OTHERS)
      CALL STEP3 (BASIC ,ORBAS)
      CALL STEP4 (BASIC ,ORBAS)
      WRITE(19,*)'     *** DONE ***'
      WRITE(19,*)'**READ FILE "RESULTS"**'
      WRITE(19,*)'** FOR FULL ANALYSIS**'
      CALL STEP5 (BASIC ,GROUP)
      CALL FLASH(19)
      CALL REPORT
      CLOSE (19)
      RETURN
      END
```

APPENDIX L : REAL TIME RUNS FOR EXAMPLES 1, 2, 4 AND 5

RESULT OF EXAMPLE 1

*** reporting step5 ***

** SETS ENTRIES **

SERIAL NO.	SET NO.	START NO.
1	1	1229
2	3	417
3	4	255
4	5	398
5	6	403
6	7	805

** SETS ELEMENTS **

SERIAL NO.	SET ROW	SET ELEM.	NEXT ROW	STATE
1	14	9	542	1
2	36	8	14	1
3	255	4	256	1
4	256	7	381	1
5	261	7	408	1
6	263	4	571	1
7	264	7	0	0
8	381	8	0	0
9	393	8	400	1
10	398	4	1060	1
11	400	9	0	0
12	403	4	261	1
13	408	8	466	1
14	417	4	264	1
15	418	3	263	1
16	466	9	817	1
17	542	12	915	0
18	571	5	1084	1
19	589	7	36	1
20	771	7	0	1
21	805	4	589	1
22	817	12	0	1
23	915	13	0	0
24	1060	7	393	1
25	1084	6	771	1
26	1229	129	418	1

EMPTY = 229
FINISH = 10000
FULL = f

RESULT OF EXAMPLE 2

*** reporting step5 ***

** SETS ENTRIES **

SERIAL NO.	SET NO.	START NO.
1	1	16
2	5	3
3	6	7
4	8	2
5	9	23

** SETS ELEMENTS **

SERIAL NO.	SET ROW	SET ELEM.	NEXT ROW	STATE
1	1	7	15	1
2	2	2	20	1
3	3	2	5	1
4	4	4	9	1
5	5	3	0	1
6	7	2	10	1
7	8	9	0	1
8	9	5	19	1
9	10	3	4	0
10	11	8	8	1
11	12	9	0	0
12	15	8	0	1
13	16	5	18	1
14	18	6	11	1
15	19	6	1	0
16	20	3	21	0
17	21	6	22	1
18	22	9	0	1
19	23	2	24	1
20	24	3	26	0
21	25	6	12	1
22	26	4	25	1

EMPTY = 17
FINISH = 10000
FULL = f

RESULT OF EXAMPLE 4

*** reporting step5 ***

** SETS ENTRIES **

SERIAL NO.	SET NO.	START NO.
1	1	4
2	5	3
3	8	2
4	9	21
5	11	28
6	12	11
7	14	47

** SETS ELEMENTS **

SERIAL NO.	SET ROW	SET ELEM.	NEXT ROW	STATE
1	2	2	18	1
2	3	2	5	1
3	4	5	8	1
4	5	3	0	1
5	7	9	12	1
6	8	8	7	1
7	9	14	0	1
8	11	2	34	1
9	12	11	25	1
10	18	3	19	0
11	19	6	20	1
12	20	9	0	1
13	21	2	22	1
14	22	3	24	0
15	23	6	40	1
16	24	4	23	1
17	25	12	9	1
18	28	2	29	1
19	29	3	32	0
20	30	5	31	1
21	31	6	0	0
22	32	4	30	1
23	34	3	41	0
24	35	5	36	1
25	36	6	38	0
26	37	9	0	1
27	38	7	37	1
28	40	9	0	0
29	41	4	35	0
30	42	7	52	0
31	45	4	49	0
32	47	2	48	1
33	48	3	45	0

34	49	5	50	1
35	50	6	42	0
36	51	9	0	1
37	52	8	51	1

EMPTY = 10
FINISH = 10000
FULL = f

RESULT OF EXAMPLE 5

*** reporting step5 ***

** SETS ENTRIES **

SERIAL NO.	SET NO.	START NO.
1	1	16
2	5	3
3	6	7
4	8	2
5	9	23

** SETS ELEMENTS **

SERIAL NO.	SET ROW	SET ELEM.	NEXT ROW	STATE
1	1	7	15	1
2	2	2	20	1
3	3	2	5	1
4	4	4	9	1
5	5	3	0	1
6	7	2	10	1
7	8	9	0	1
8	9	5	19	1
9	10	3	4	0
10	11	8	8	1
11	12	9	0	0
12	15	8	0	1
13	16	5	18	1
14	18	6	11	1
15	19	6	1	0
16	20	3	21	0
17	21	6	22	1
18	22	9	0	1
19	23	2	24	1
20	24	3	26	0
21	25	6	12	1
22	26	4	25	1

EMPTY = 17
FINISH = 10000
FULL = f

REFERENCES

1. Lees, F.P. 'Loss prevention in the process industries', Vol. 1, pp. 2, Butterworths, London, (1980).

2. Cornell, C.E. 'Minimizing human errors', Space Aeronautics, Vol. 49, pp.72-81, March (1968).

3. Scott, R.L. 'A review of safety-related occurrences in nuclear power reactors from 1967-1970', Oak Ridge National Laboratories, Tenn., USA, ORNL-TM-3435, pp. 13, May (1971).

4. Ovenu, H. 'Inspection service - vital factor in securing maximum plant availibilty', Proceedings of American Power stations Conference, (1969).

5. James, R. and H.P. Bloch 'Instrumentation for predictive maintenance monitoring', Loss Prevention, CEP, AIChE, Vol.10, pp. 101-107, (1976).

6. Swain, A.D. 'Some problems in the measurement of human performance in man-machine systems', Human Factors, Vol. 6, pp. 687-700, December (1964).

7. Klaassen, P.L. 'Importance of software/hardware for safe processing', Loss Prevention, Vol. 12, CEP, AIChE, pp. 118-123, (1979).

8. Hix, A. H. 'Safety and instrumentation systems', Loss Prevention, CEP, AIChE, Vol. 6, pp. 4-13, (1972).

9. King, C.F. and D.F. Rudd 'Design and maintenance of economically failure-tolerant processes', AIChEJ, Vol. 18, pp. 257, (1972).

10. Young, J. 'Using the fault tree analysis technique' Reliability and Fault Tree Analysis, SIAM, Philadelphia, pp. 827-847, (1975).

11. Keener, E. L. 'Operator training with a dynamic simulation of a process', ISACC, No. 1, pp. 90, (1970).

12. Edwards, E. and F. Lees 'Man and computer in process control', Institute of Chemical Engineers, pp. 118, (1972).

13. Edwards, E. and F. Lees 'Man and computer in process control', Institute of Chemical Engineers, pp. 171, (1972).

14. Kletz, T.A. 'Practical applications of hazard analysis', Loss prevention, CEP, AIChE, Vol. 12, pp. 34-40, (1978).

15. Lees, F.P. 'Loss Prevention in the process industries', Vol. 2, pp. 1049, Butturworths, London, (1980).

16. Trepa de Faria, F.A. and D.A. Lihou 'Operability studies and fault finding', Chempor 78 Conference at Braga, Portugal pp. 10/1-15, September (1978).

17. Lawley, H.G. 'Operability studies and hazard analysis', Loss prevention, CEP, AIChE, Vol. 8, pp. 105-116, (1974).

18. Elliott, D.M. and J.M. Owen 'Critical Examination In Process Design', No. 223, The Chemical Engineer (London), pp. 377- 383, November (1968).

19. Lawley, H.G. 'Operability Studies And Hazard Analysis', CEP, Vol.70, No.4, pp.45-56, (1974).

20. Lawley, H.G. 'Size Up Plant Hazards This Way', Hydrocarbon Processing, Vol. 55, No. 4, pp.247-261, April (1976).

21. Chemical Industries safety and health council 'A guide for hazard and operability studies', Chemical Industries Association Ltd. (London), pp. 5, (1977).

22. Chemical Industries safety and health council 'A guide for hazard and operability studies', Chemical Industries Association Ltd. (London), pp. 7, (1977).

23. Kletz, T.A. 'Hazop and Hazan', Institute of Chemical Engineers (London), pp. 8, (1983).

24. Chemical Industries safety and health council 'A guide for hazard and operability studies', Chemical Industries Association Ltd. (London), pp. 6, (1977).

25. Taylor, J.R. 'Evaluation of costs, quality and benefits for six risk analysis procedures', Internal report, Electronics Department, RISO, N-14-82, May (1982).

26. Lihou, D.A. 'Computer aided operability studies for loss control', 3rd International Symposium In Loss Prevention And Safety Promotion In The Process Industries. Basle- Switzerland, Vol. 2, pp. 7/579, September (1980).

27. Jordan, A.P.H. 'Evaluating hazard ranges using the Aston computer package' A short Course On Hazard Evaluation And Control, Dept. of Chemical Eng., University of Aston in Birmingham, 27-30 September (1982).

28. Fussell, J.B. 'Fault tree analysis - concepts and techniques' NATO Advanced Study, Inst. on Generic Techniques of System Reliability Assessment, Nordhoff Publishing Company, pp. 133- 162, July (1973).

29. Jones, G.P. 'Risk analysis in hazardous materials transportation', University of Southern California Los Angles, Institute of Aerospace safety and Management, RAPO 737, Vol. 1, pp. 199, March (1973).

30. Bell Telephone Laboratories 'Launch control Safety Study', Bell Telephone Labs., Murray Hill, New Jersey, USA, Vol. 1, section VII, (1961).

31. System Safety Symposium, Seattle, Washington : The Boeing Company, June 8-9, (1965).

32. Spiegelman, A. 'Risk evaluation of chemical plants', Loss Prevention, CEP, AIChE, Vol. 3, pp. 1-10, (1969).

33. Browning, R.L. 'Use a fault tree to check safeguards', Loss Prevention, CEP, AIChE, Vol. 12, pp. 20-26, (1979).

34. Semanderes, S. N. 'ELRAFT a computer program for the efficient logic reduction analysis of fault trees', IEEE Trans. Nuclear Science, Vol. NS-18, No.1, pp. 481-487, (1971).

35. Vesely, W. E. 'Analysis of fault trees by kinetic theory', IN-1330, Idaho Nuclear Corp., Idhao Fall, October (1969).

36. Vesely, W. E. 'A time-dependent methodology for fault tree analysis', Nuclear Engineering and Design, Vol. 13, pp. 337- 360, August (1970).

37. Crosetti, P. 'Computer program for fault tree analysis', Douglas United Nuclear Inc., Richard, Washington, DUN-5508, April (1969).

38. Lapp, S. A. and G. J. Powers, 'Computer-aided synthesis of fault trees', IEEE Trans. Reliability, Vol. R-23, p. 51-55, April (1974).

39. Bennetts, R. G., 'On the analysis of fault trees', IEEE Trans. Reliability, Vol. R-24, pp.175-185, August (1975).

40. Lee, K. K. 'A compilation technique for exact system Reliability' IEEE Trans. Reliability Vol. R-30, No.3, pp. 284-288, August (1981).

41. Worrell, R. B. ' Using the set equation transformation system in fault tree analysis', Reliability and fault tree analysis, SIAM, Philadelphia, pp. 165-185, (1975).

42. Esary, J. and F. Proschan 'Coherent structures of non-identical components', Technometrics, Vol. 5, pp.191-209, (1963).

43. Mearns, A. B.' Fault tree analysis : The study of unlikely events in complex systems', System Safety Symposium, June 8-9, (1965), Seattle: The Boeing Company. (Cited in ref.13)

44. Crosetti, P.A. 'Fault tree analysis with probability evaluation', IEEE Trans. Nuclear Science, Vol. NS-18, No. 1, pp. 465-471, (1971).

45. Fussell, J. B. 'A formal methodology for fault tree construction', Nuclear Engineering and Design, Vol. 52, pp. 337-360, (1973).

46. Salem, S. L., G. E. Apostolakis and D. Okrent 'A new methodology for the computer-aided construction of fault tree' Annals of Nuclear Energy, Vol. 4, pp. 417-433, (1977).

47. Reactor Safety Study - An assessment of accident risks in U.S. commercial nuclear power plants, WASH-1400 (NUREG-75/014), U.S. Nuclear Regulatory Commission, Washington, D.C., Appendix II, October (1975).

48. Haasl, D. F. 'Advanced concepts on fault tree analysis', System Safety Symposium, The Boeing Company, Seattle, Washington, June 8-9 (1965).

49. Powers, G.J. and F.C. Tompkins 'Computer aided synthesis of fault tree for complex processing', NATO Advanced Study Institute on Generic Techniques of System Reliability Assessment, Nordhoff Publishing Company, Liverpool, pp. 307- 314, July (1973).

50. Powers, G. J. and F. C. Tompkins 'Fault tree synthesis for chemical processes', AIChEJ, Vol. 20, pp. 376-387, March (1974).

51. Salem, S. L., G. E. Apostolakis and D. Okrent 'A computer-oriented approach to fault tree construction', EPRI NP-288, Electric Power Research Institute, November (1976).

52. Lapp, S. A. and G. J. Powers 'Computer-aided synthesis of fault trees', IEEE Trans. Reliability, Vol. R-26, pp. 2-13, April (1977).

53. Hollo, E. and J.R. Taylor 'Algorithms and programs for consequence diagram and fault tree construction', Report No. RISO-M-1907, Danish Atomic Energy Commission, Roskilde, Denmark, December (1976).

54. Nielsen, D. 'Use of cause-consquence charts in practical systems analysis', Reliability and Fault Tree Analysis, SIAM, Philadelphia, pp. 849-880, (1975).

55. Camarda, P., F. Corsi and A. Trentadue 'An efficient simple algorithm for fault tree automatic synthesis from reliability graph', IEEE Trans. Reliability, vol. R-27, pp. 215-221, August (1978).

56. Bazovisky, I. 'Reliability theory and Practice', Prentice-Hall, Inc., Englewood Cliffs, New Jersey, USA, (1961).

57. Atkinson, J.H. 'The set equation transformation system used in analysis of a typical naval weapon system', Reliability and Fault Tree Analysis, SIAM, Philadelphia, USA, pp. 187- 202, (1975).

58. Cummings, G.E. 'Application of the fault tree technique to a nuclear reactor containment system', Reliability and Fault Tree Analysis, SIAM, Philadelphia, USA, pp. 805-825, (1975).

59. Myers, R.H., K.L. Wong and H.M. Gordy 'Reliability engineering for electronic systems', New York, Wiley, (1964).

60. British Standard Institution, BS4200 : part2 : 1967 'Guide on the reliability of electronic equipment and parts used there in terminology'.

61. Laviron, A. 'ESCAF for MTBF, time evolution, sensitivity coefficients, cut-set importance, and non-coherence of large systems' IEEE Trans. Reliability, vol. R-35, No. 2, pp. 139-144, June (1986).

62. Hastings, N.A. and J.B. Peacock 'Statistical distributions', Butterworths, London, (1974).

63. Lihou, D. A. ,'Estimation/Calculation of probabilities', Short Course on Hazard Evaluation and Control, Chem. Eng. Dept., Aston University, pp. C/3.1-3.17, September (1982).

64. Kamat, S.J. and M.W. Riley 'Determination of reliability using event-based Monte Carlo simulation', IEEE Trans. Reliability, Vol. R-24, pp. 73-75, April (1975).

65. Kamat, S.J. and W.E. Franzmeier 'Determination of reliability using event-based Monte Carlo simulation part II', IEEE Trans. Reliability, Vol. R-25, pp. 254-255, October (1976).

66. Hammersley, J.M. and D.C. Handscomb 'Monte Carlo Method', Methuen and Co. Ltd., London, (1967).

67. Mazumdar, M. 'Importance sampling in reliability estimation', Reliability and Fault Tree Analysis, SIAM, Philadelphia, pp. 153-163, (1975).

68. Kumamoto, H., K. Tanaka and K. Inone 'Efficient evaluation of system reliability by Monte Carlo method', IEEE Trans. Reliability, Vol. R-26, pp. 311-315, December (1977).

69. Levy, L.L. and A.H. Moore 'A Monte Carlo technique for obtaining system reliability confidence limits from component test data', IEEE Trans. Reliability, Vol. R-16, pp. 69-72, September (1967).

70. Vesely, W.E. 'Reliability and fault tree applications at the NRYS', IEEE Trans. Nuclear Science, Vol. NS-18, No. 1, pp. 472-480, (1971).

71. Vesely, W.E. and R.E. Narum 'PREP and KITT: Computer codes for the automatic evaluation of a fault tree', IN-1349, National Technical Information Service, Springfield, VA22161 USA, August (1970).

72. Garrick, B.J. 'Principles of unified system safety analysis', Nuclear Engineering and Design, Vol. 13, pp. 245-321, (1970).

73. Kongsoe, H.E. 'RELY4, a Monte Carlo computer program for systems reliability analysis', Danish Atomic Energy Commission, RISO-M-1500, June (1972).

74. Kongsoe, H.E. 'REDIS, a computer program for system reliability analysis by direct simulation', International Symposium on Reliability of Nuclear Power Plants, Innsbruck, Austria, April 14-18, (1975).

75. Karp, R. and M.G. Luby 'A new Monte Carlo method for estimating the failure probability of an n-compnent system', Computer Science Division, University of California, Berekeley, USA, (1983).

76. Bennetts, R.G. 'Analysis of reliability block diagrams by Boolean technique' IEEE Trans. Reliability, Vol. R-31, pp. 159-166, June (1982).

77. Fussell, J.B. and N.H. Marshall 'MOCUS- a computer program to obtain minimal sets from fault trees', ANCR-1156, Aerojet Nuclear Company, Idaho Falls, Idaho, USA, March (1974).

78. Van Slyke, W.J. and D.E. Griffing 'ALLCUTS- a fast comprehensive fault tree analysis code', ARH-ST-112, July (1975).

79. Pande, P.K., M.E. Spector and P. Chatterjee 'Computerised fault tree analysis: TREEL and MICSUP', ORC-75-3, Operations Research Center, University of California, Berkeley, California USA, April (1975).

80. Erdmann, R.C., J.E. Kelly, H.R. Kirch, F.L. Leverenz and E.T. Rumble 'A method for quantifying logic models for safety analysis', Nuclear Systems Reliability Engineering and Risk Assessment, SIAM, Philadelphia, USA, pp. 732-754, (1975).

81. Garribba, S., P. Mussio, F. Naldi, G. Reina and G. Volta 'Efficient construction of minimal cut sets from fault trees', IEEE Trans. Reliability, Vol. R-26, No. 2, pp. 88-94, June (1977).

82. Rasmuson, D.M. and N.H. Marshall 'FATRAM - A core efficient cut-set algorithm', IEEE Trans, Reliability, Vol. R-27, No. 4, pp. 250-253, October (1978).

83. Nakashima, K. and Y. Hattori 'An efficient bottom-up algorithm for enumerating minimal cut sets of fault trees', IEEE Trans. Reliability, Vol. R-28, No. 5, pp. 353-357, December (1979).

84. Magee, D. and A. Refsum 'RESIN, A desktop-computer program for finding cut sets', IEEE Trans. Reliability, Vol. R-30, No. 5, pp. 407-410, December (1981).

85. Jasmon, G.B. and O.S. Kai 'A new technique in minimal path and cut set evaluation', IEEE Trans. Reliability, Vol. R-34, No. 2, pp. 136-143, June (1985).

86. Bengiamin, N.N., B.A. Bowen and K.F. Schenk 'An efficient algorithm for reducing the complexity of computation in fault tree analysis', IEEE Trans. Nuclear Science, Vol. NS-23, No. 5, pp. 1442-1446, October (1976).

87. Koen, B.V. and A. Carnino 'Reliability calculations with a list processing technique', IEEE Trans. Reliability, Vol. R-23, pp.43-50, April (1974).

88. Wheeler, D.B., J.S. Hsuan and G.M. Roe 'Fault tree analysis using bit manipulation', IEEE Trans. Reliability, Vol. R-26, No. 2, pp. 95-99, June (1977).

89. Kumamoto, H. and E.J. Henley 'Top-down algorithm for obtaining prime implicant sets of non-coherent fault trees', IEEE Trans. Reliability, Vol. R-27, No. 4, pp. 242-249, October (1978).

90. Fussell, J.B. and W.E. Vesely 'A new methodology for obtaining cut sets for fault trees', Trans. American Nuclear Society, Vol. 15, No.1, pp. 263, (1972).

91. Rasmuson, D.M., N.M. Marshall and G.R. Burdick 'User's Guide for the reliability analysis system (RAS)', Tree-116, NTIS, Sprigfield, VA 22161 USA, September (1977).

92. Prugh, R.W. 'Applications of fault tree analysis', Loss Prevention, CEP, AIChE, Vol. 14, pp. 1-9, (1981).

93. Lapp, S.A. 'Computer aided fault tree synthesis', Msc. thesis,

Carnegie-Mellon University, pp. 20-38, (1976).

94. Reactor Safety Study -An assessment of accident risks in U.S. commercial nuclear power plants, WASH-1400 (NUREG-75/014), U.S. Nuclear Regulatory Commission, Washington, D.C., Appendix II, pp. 19-25, October (1975).

95. Reactor Safety Study -An assessment of accident risks in U.S. commercial nuclear power plants, WASH-1400 (NUREG-75/014), U.S. Nuclear Regulatory Commission, Washington, D.C., Appendix II, pp. 255-263, October (1975).

96. Fletcher, W.I. 'Engineering approach to digital design', Prentice-Hall, International editions, Englewood Cliffs, N.J., pp. 134, (1980).

97. Hirst, K.E. and F. Rhodes 'Conceptual models in mathematics', George Allen and Unwin Ltd., 1971.

98. Lambert, H.E. 'Fault trees for decision making in system analysis', Lawrence Livermore Laboratory, University of California, Livermore, UCRL-51829, October (1975).

99. Chatterjee, P. 'Modularization of fault trees: A method to reduce the cost of analysis', Reliability and fault tree analysis,SIAM, Philadeliphia, pp.101-126, (1975).

100. Caldarola, L. and A. Wickenhauser 'The Karlsruhe computer program for the evaluation of the availability and reliability of complex repairable systems', Nuclear Engineering and Design, Vol. 43, pp.463-470, (1977).

101. Caldarola, L. 'fault tree analysis of multistate systems with multistate components', Probabilistic Analysis of Nuclear Reactor Safety, Los Angeles, Calefornia, May 8-10, pp.VIII/1- 28, (1978).

102. Caldarola, L. 'Coherent systems with multistate systems', Nuclear Engineering and Design, Vol. 58, pp. 127-139, (1980).

103. Chamow, M.F. 'Directed graph techniques for the analysis of fault trees', IEEE Trans. Reliability, Vol. R-27, No. 1, pp. 7-15, April (1978).

104. Clarotti, C.A. 'Limitations of minimal cut set approach in evaluating reliability of systems with repairable components', IEEE Trans. Reliability, Vol. R-30, pp.335-338, October (1981).

105. Rumble, E.T. and J. Olmos 'Fault tree analysis incorporating dependent events', Probabilistic Analysis of Nuclear Reactor Safety, Topical Meeting May 8-10, Los Angeles, California, USA, pp. X.4/1-12, (1978).

106. Locks, M.O. 'Fault trees, prime implicants and noncoherence', E.I. Ogunbiyi 'Author reply #1', H. Kumamoto and E.J. Henley 'Author reply #2', M.O. Locks 'Rebuttal', IEEE Trans. Reliability, Vol. R-29, No. 2, pp. 130-135, June (1980).

107. Locks, M.O. 'Modularizing, minimizing and interpreting the K&H fault tree', IEEE Trans. Reliability, Vol. R-30, No. 5, pp. 411-417, December (1981).

108. Worrell, R.B, D.W. Stack and B.L. Hulme 'Prime implicant of noncoherent fault trees', IEEE Trans. Reliability, Vol. R-30, No. 2, pp. 98-100, June (1981).

109. Case, I. 'A reduction technique for obtaining a simplified reliability expression', IEEE Trans. Reliability, Vol. R-26, No. 4, pp. 248-249, October (1977).

110. Gopal, K. and S. Rai 'Discussion on -A reduction technique for obtaining simplified reliability expression', IEEE Trans. Reliability, Vol. R-28, No. 1, pp.66-66, April (1979).

111. Aggarwal, K.K. 'Comments on -On the analysis of fault trees', R.G. Bennetts 'Author's Reply', IEEE Trans. Reliability, Vol. R25, No. 2, pp. 126-127, June (1976).

112. Jiongsheng, L. 'A new approach for fault tree analysis', Scientia Sinica, Series A, Vol. 25, No. 9, pp. 983-992, September (1982).

113. Rushdi, A.M. 'On Reliability Evaluation by network decomposition', IEEE Trans. Reliability, Vol. R-33, No. 5, pp. 379-383, December (1984).

114. Lihou, D. 'Bhopal and byond', The Chemical Engineer, No. 414, UK, pp. 15-15, May (1985).

115. Burdick, G.R.'COMCAN -A computer code for common cause analysis', IEEE Trans. Reliability, Vol. R-26, No. 2, pp. 100-102, June (1977).

116. Astolfi, M., S. Contini, C.L. Van der Muyzenberg and G. Volta 'Fault tree analysis by list processing techniques', Synthesis and Analysis Methods for Safety and Reliability Studies, Editors, G. Apostdakis, S. Garribba and G. Volta, Plenum, pp. 5-32, (1978).

117. Lee, W.S., D.L. Grosh, F.A. Tillman and C.H. Lie 'Fault tree analysis, methods and applications- A review', IEEE Trans. Reliability, Vol. R-34, No. 3, pp. 194-203, August (1985).

118. Garrick, B.J. 'Principles of unified system safety analysis', Nuclear Engineering and Design, Vol. 13, pp. 245-321, (1970).

119. ICL-Perq 'Interoduction to Perq', first edition, International Computers Limited, May (1982).

120. ICL-Perq 'PNX operating system for Perq1', International Computers Limited, Program product set notice, November (1985).

121. Bidmead, C. 'The octopus in your tank', Practical Computing, Vol. 7, No. 4, pp. 107-109, April (1984).

122. ICL-Perq 'Unix program's manual', Seventh edition, Vollume 1, pp. 189, International Computers Limited, November (1985).

123. ICL-Perq 'Guide to PNX version 5', International Computers Limited, ICL Hous, Putney, London, (1985).

124. Gay, B. and A.P.H. Jordan 'The effect of advanced workstations on CAD', People and Computers : Designing the interface, Proceedings of the Conference of the British Computer Society, Human Computer Interaction, University of East Anglia, pp.(36-401), 17-20 September (1985).

125. Kernighan, B.W. and D.M. Ritchie 'The C programming language' Prentice-Hall, Inc., Englewood Cliffs, New Jersey, (1978).

126. ICL-Perq 'Developing ICL-Pascal programs under PNX', First edition, International Computers Limited, London, (1984).

127. Patel, V.H. 'Computing plant reliability from Hazop studies' M.Sc.Thesis, Department of Chemical Engineering, University of Aston in Birmingham, pp. 31, October (1982).

128. Moraes, I., Computer Project (Course Work), Department of Chemical Engineering, University of Asron in Birmingham, March (1981).

129. Varelas, T., Computer Project (Course Work), Department of Chemical Engineering, University of Aston in Birmingham, March (1981).

130. Napinda, A., Computer Project (Course Work), Department of Chemical Engineering, University of Aston in Birmingham, March (1981).

131. Spirakos, S. E., Computer Project (Course Work), Department of Chemical Engineering, University of Aston in Birmingham, March (1982).

132. Collins, D., Computer Project (Course Work), Department of Chemical Engineering, University of Aston in Birmingham, March (1982).

133. Kyriacou, A., Computer Project (Course Work), Department of Chemical Engineering, University of Aston in Birmingham, March (1982).

134. Jordan, A.P.H., Private Communication, 1982.

135. Patel, V.H. 'Computing plant reliability from Hazop studies' M.Sc. Thesis, Department of Chemical Engineering, University of Aston in Birmingham, pp. 35-40, October (1982).

136. User Guide 'GINO-F the general purpose graphical package', Version 2.7A, CAD centre ltd., High Cross, Madingley road, Cambridge, October (1983).

137. Akabe, R. and A.P.H. Jordan, Computer Project (Course Work), Deptartment of Chemical Engineering, University of Aston in Birmingham, November (1983).

138. Ramadaan, S.Y. 'The use of hazard and operability studies in design and control', First Year Report, Department of Chemical Engineering, University of Aston in Birmingham, May (1984).

139. Rosenthal, A. 'Approaches to comparing cut set enumeration algorithms', IEEE Trans. Reliability, Vol. R-28, No. 1, pp. 62-65, April (1979).

140. Aho, A., J. Hopcroft and J. Ullman 'The design and analysis of computer algorithms', Addison-Wesley, Reading, Massachusetts, (1974).

141. Knuth, D.E. 'The art of computer programming', Vol. 1, Second Edition, Addison-Wesley, Reading, Massachusetts, (1969).

142. Strecher, K. 'Evaluation of large fault trees with repeated events using an efficient bottom-up algorithm', IEEE Trans. Reliability, Vol. R-35, No. 1, pp. 51-58, April (1986).

143. Page, L.B. and J.E. Perry 'A simple approach to fault tree probabilities', Computers and Chemical engineering, Vol. 10, No. 3, pp. 249-257, (1986).

144. Fishman, G.S. 'A comparison of four Monte Carlo methods for estimating the probability of s-t connectedness', IEEE Trans. Reliability, Vol. R-35, No. 2, pp. 145-155, June (1986).

145. Page, L.B. and J.E. Perry 'An algorithm for exact fault tree probabilities without cut sets', IEEE trans. Reliability, Vol. R-35, No. 5, pp. 544-558, December (1986).

146. Limnios, N. and R. Ziani 'An algorithm for reducing cut sets in fault tree analysis', IEEE Trans. Reliability, Vol. R-35, No. 5, pp. 559-565, December (1986).

147. Henley, E.J. and H. Kumamoto 'Reliability engineering and risk assessment', Prentice-Hill, Inc., Englewood Cliffs, New Jersey, (1981).

148. Andow P. 'Expert system in Safety and Reliability', a lecture orginised by the Department of Chemical Engineering, Asto University, 2nd of December (1986).

149. Nisenfeld, A.E. 'Shutdown features of in-line process control', Loss Prevention, CEP, AIChE, Vol. 6, pp.1-3, (1972).

150. Russell, W.W. 'Hazard control of plant process changes', Loss Prevention, CEP, AIChE, Vol. 10, pp. 80-87, (1976).

151. Lees, F.P. 'Loss Prevention in the process industries', Vol. 2, Appendix 1, pp. 863-881, Butturworths, London, (1980).

152. Toth, L.M., A.P. Malimauskas, G.R. Eiden and H.M. Burton 'The three miles accident -diagnosis and prognosis', ACS Symposium Series, American Chemical Society, Washington, D.C., pp. 2-26, (1986).

153. Andow, P.K. and F. P. Lees 'Process plant alarm systems: General considerations', Loss Prevention and Safety Promotion In The Process Industries,Elsevier, Amstrdam, Vol. 1, Pp. 299-307, (1974)