This is to certify that the

dissertation entitled

Catastrophic Fault Diagnosis in Dynamic Systems
Using Bond Graph Methods

presented by

Tamar Yarom

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Mechanical Engineering

R.C. Rosenberg
Major professor

Date July 23, 1990

CATASTROPHIC FAULT DIAGNOSIS IN DYNAMIC SYSTEMS USING BOND GRAPH METHODS

By

Tamar Yarom

A DISSERTATION

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

DOCTOR OF PHILOSOPHY

Mechanical Engineering Department

1990

ABSTRACT

CATASTROPHIC FAULT DIAGNOSIS IN DYNAMIC SYSTEMS USING BOND GRAPH METHODS

By

Tamar Yarom

Detection and diagnosis of faults has become a critical issue in high performance engineering systems as well as in mass produced equipment. It is particularly helpful when the diagnosis can be made at the initial design level with respect to a prospective fault list. A number of powerful methods have been developed for aiding in the general fault analysis of designs. Catastrophic faults represent the limit case of complete local failure of connections or components. They result in the interruption of energy transfer between corresponding points in the system.

In this work the conventional approach to fault detection and diagnosis is extended by means of bond graph methods to a wide variety of engineering systems. Attention is focussed on catastrophic fault diagnosis. A catastrophic fault dictionary is generated from the system model based on topological properties of the bond graph. The dictionary is processed by existing methods to extract a catastrophic fault report to aid the engineer in performing a design analysis.

The proposed fault dictionary was found applicable for testability design.

An extension of the software for constructing the fault dictionary has been developed in order to use it as a design tool for various engineering applications.

ACKNOWLEDGMENTS

I would like to acknowledge all those who have contributed to the success of this thesis.

Special thanks are extended to my adviser, Dr. Ronald Rosenberg, not only for his expert guidance but also for his sincere friendship and encouragement. I would like to express my gratitude to the Guidance Committee, Dr. Benjamin Bachrach for his careful reading and useful suggestions, Dr. Steven Shaw, for clarifying physical definitions and Dr. Hassan Khalil for his comments concerning this dissertation.

During this research I was supported, in part, By Zonta International organization and by Ford Motor Company. The support is gratefully acknowledged.

But most of all, I wish to thank ny husband, Eli, and my two daughters, Orli and Michal, for their love, support and patient, without which this work would never come to end.

TABLE OF CONTENTS

page

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

D(l,m,n)	- Fault dictionary array
e(t)	- effort variable
P_{ij}	- Equivalent path from U_i to Y_j
f(t)	- flow variable
F_k	- The kth fault in the fault list
l	- Number of inputs
m	- Number of test nodes (outputs)
n	- Number of faults in the fault list
P_{ij}	- Path from U_i to Y_j
P(t)	- Power variable; $P(t)=e(t)*f(t)$
S_{ij}	- Sensitivity of Y_j with respect to U_i
U_i	- The ith input of the system
U_{ci}	- The ith constant input of the system
Y_j	- The jth output of the system
Y_{ssj}	- The jth steady-state output of the system

ABBREVIATIONS

BG	-	Bond Graph
BS	-	Bond Structure
CI	-	Constant Input
CF	-	Catastrophic Failure
CFD	-	Catastrophic Fault Dictionary
CEF	-	Catastrophic Effort Failure
CFF	-	Catastrophic Flow Failure
DFI	-	Depth First Search
DFL	-	Detectable Fault List
DFS	-	Depth-First Search
EP	-	Equivalent Path
FA	-	"Father"
FD	-	Fault Dictionary
FDD	-	Fault Detection and Diagnosis
FL	-	Fault List
FS	-	Fault Source
FSE	-	Fault Source of Effort
FSF	-	Fault Source of Flow
IJ	-	Identity Junction
JS	-	Junction Structure
KCL	-	Kirchhoff Current Law
KVL	-	Kirchhoff Voltage Law
LSI	-	Large-Scale Integration
MBG	-	Modified Block Graph
PBG	-	Proper Bond Graph

SAT	-	Simulation After Test
SBT	-	Simulation Before Test
SJ	-	Simple Junction
SM	-	Sensitivity Matrix
SSM	-	Structural Sensitivity Matrix
SRU	-	Shop Replaceable Unit
SUT	-	System Under Test
TNM	-	Test-Node Matrix
VLSI	-	Very-Large-Scale Integration

1. INTRODUCTION

1.1 Anticipating failures in engineering systems

The proliferation of large engineering systems of ever-increasing complexity makes the detection and diagnosis of faults in them an important task. The early indication of incipient failures can help avoid major system breakdowns and catastrophes. Similarly, failure detection and isolation has become a critical issue in the operation of high-performance ships, submarines, airplanes, space vehicles, and structures, where safety, mission satisfaction, and significant material value are at stake. Quite recently, computerized diagnostic systems have been applied to such mass-produced consumer equipment as automobiles and household appliances.

From the very beginning of computerized testing, most practical systems have contained some form of failure detection and diagnosis. In the majority of these systems, the detection and diagnosis function is rather simple and is based on straight limit checking. The development of computational equipment and techniques has set the scene for the general application of more sophisticated and powerful methods.

1.2 Review of literature

The problem of detecting changes in dynamical systems has received growing attention during the last twenty years, as can be seen from the long list of survey papers and books written about it (Willsky, 1976; Duhamel, 1979; Mironovski, 1980; Iserman, 1984; Bandler, 1985; Basseville, 1988; Gertler, 1988; Ozawa, 1988). Most of the papers deal with the subject of fault location and estimation in circuits. The applications are mainly to analog circuits (Hochwald, 1979; Wu, 1982; Elcherif, 1983; Lin, 1982, 1985) and digital circuits (Williams, 1983; Markas 1990). Other papers give a generic approach to dynamic systems, and a few treat particular mechanical systems such as the automobile system (de Benito, 1988, 1989) and the space shuttle engine (Cikanek, 1987).

1.2.1 Basic conceptual terminology

A *system fault* is to be understood as a deviation of a characteristic property which leads to an inability of the system to fulfill its intended purpose.

The fault diagnosis process includes three steps: the *alarm, isolation,* and *estimation* of the fault. The *alarm* step is usually done by checking if measurable variables are within a certain tolerance of the normal value. If this check is not passed, this leads to a fault message. If necessary, this is followed by *fault isolation*, determining where the fault is located. The next step, the *fault estimation*, defines the type and extent of the fault. After the effect of the fault is known, a *decision* on the action to be taken can be made. Since the decision depends upon the system design, failure detection methods are limited to the diagnosis stage. Moreover, the problem which is mainly addressed is the fault location and identification, assuming that the system has already been identified as faulty, i.e., a fault message has already been received.

Failures in a system could be *catastrophic (hard) failures* or *deviation (soft) failures*. Hard failures in circuits means that the faulty element produces either a short circuit or an open circuit. Soft failure in an element occurs when the faulty element deviates from its nominal value without reaching its extreme bounds (typically zero or infinity). Field data show that short circuits and open circuits account for about 70-80% of the faults in analog equipment (Hockwald, 1979).

1.2.2 Fault detection methods

Several criteria have been used for categorizing fault detection techniques. The most popular one is categorization according to the stage in the testing process at which simulation of the tested system occurs. In particular, we have *simulation-before-test (SBT)* and *simulation-after-test (SAT)*.

1.2.2.1 Simulation-before-test method

The SBT method (Lin, 1985), also called the fault dictionary method, is based upon pattern recognition. The first step in constructing the dictionary is *fault definition*, where the most likely and/or important faults are specified. This is a very critical aspect of this approach since only these faults can be identified and the number of faults defines the memory size needed for the algorithm. The system under test (SUT) is then simulated for these hypothesized faulty cases, in order to develop sets of stimuli and responses which will be used to detect and isolate the faults. The *signatures* of the responses are stored in a dictionary for use in the on-line identification of faults. At the time of testing, the faulty SUT is excited by the same stimuli that are used in constructing the dictionary. The signatures are compared to the prestored signatures using an isolation criterion, and the actual failure is detected. Various types of inputs have been proposed for the fault dictionary method. These include constant (DC) inputs (Hockwald, 1979), sinusoidal (AC) inputs (Pahwa, 1982), and piecewise constant inputs (Schreiber, 1979). Unfortunately, most of these methods either are limited to linear networks or have not progressed beyond the feasibility stage of development. At present the constant input fault dictionary is the only one used in practice. Figure 1.1 (from Hochwald, 1979) presents a block diagram of a DC approach for the construction of a fault dictionary of an analog circuit.

In this work, a new approach to fault dictionary is proposed. Thus, it is necessary first to review in details the conventional approach. A numerical example (a modification of an example from Ozawa, 1988) is given to illustrate the procedure of constructing the fault dictionary and the notation. The system (circuit) is deliberately chosen to be simple so that all steps can be verified without the aid of a computer.

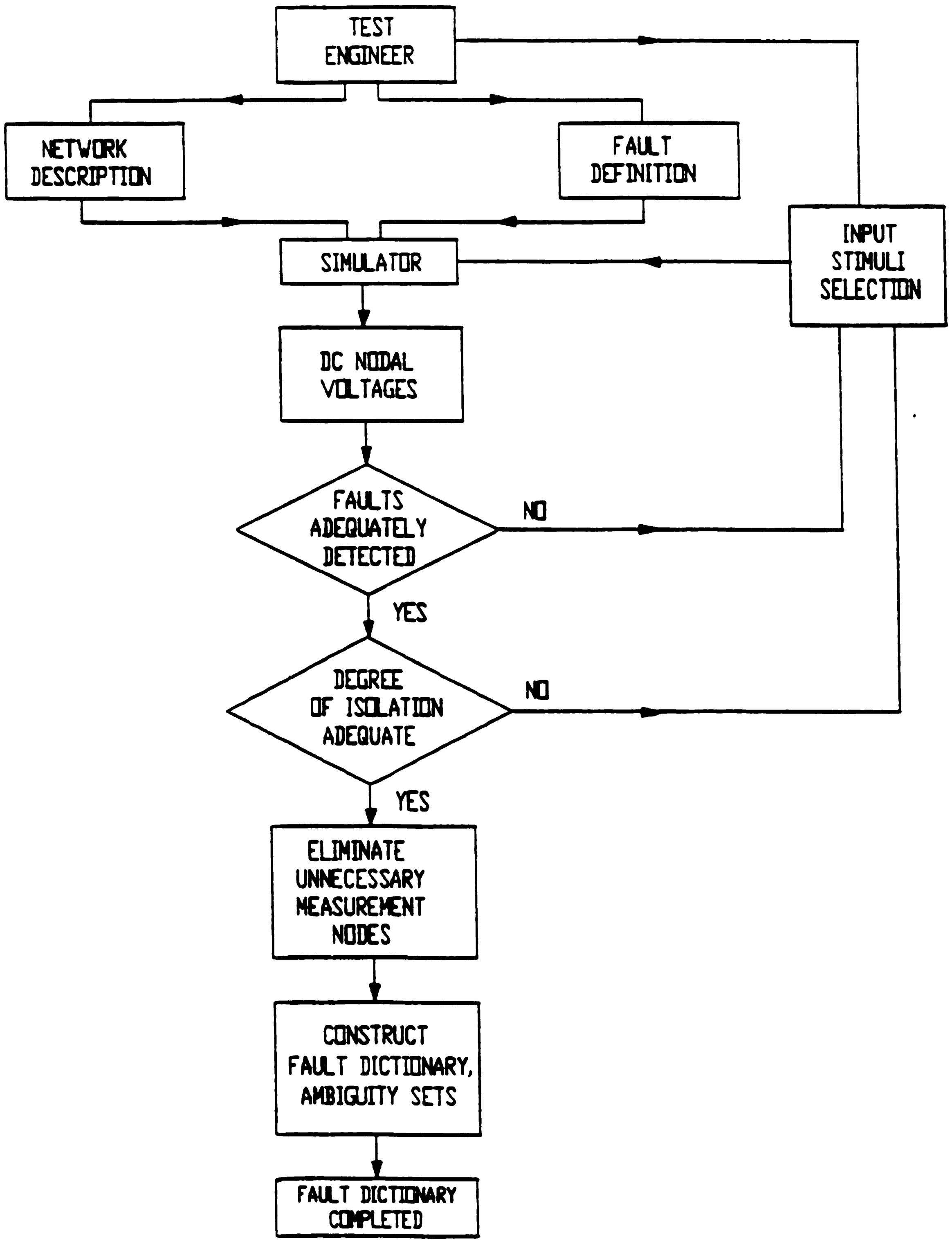

TEST ENGINEER
NETWORK DESCRIPTION
FAULT DEFINITION
SIMULATOR
INPUT STIMULI SELECTION
DC NODAL VOLTAGES
FAULTS ADEQUATELY DETECTED
NO
YES
DEGREE OF ISOLATION ADEQUATE
NO
YES
ELIMINATE UNNECESSARY MEASUREMENT NODES
CONSTRUCT FAULT DICTIONARY, AMBIGUITY SETS
FAULT DICTIONARY COMPLETED

Figure 1.1: Block diagram of a DC approach for the construction of a fault dictionary of an analog circuit

Example: Figure 1.2 shows a circuit in its normal operating condition. Suppose that the prospective faults consist of R1 shorted, R2 open, R3 open and R4 shorted. The circuit contains two test-nodes which measure the voltage at points A and B. The test-node measurements are defined as VA and VB respectively.

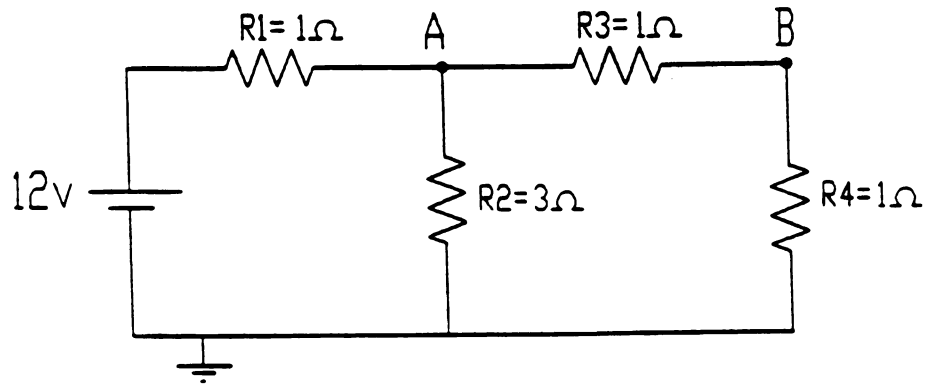


Figure 1.2: A simple circuit example

The list of prospective faults is given in Table 1.1. Note that the nominal case is just one of the nominal case is just one of the circuit conditions and is listed as case 1 in Table 1.1.

Table 1.1: Definition of faults

Fault	Description
N	Nominal case
F1	R1 shorted
F2	R2 open
F3	R3 open
F4	R4 shorted

In this stage, a simulation program is used to determine the test-node measurements for each condition defined by the fault list. For our simple circuit VA and VB are calculated directly from the circuit description. The values are provided in Table 1.2.

Table 1.2: Fault dictionary defined by two test-nodes

Circuit condition	VA	VB
N	6.54	3.27
F1	12	6
F2	8	4
F3	9	0
F4	5.14	0

Table 1.2 defines the fault dictionary. Here each fault-signature consists of a vector containing VA and VB. When actual measurements of VA and VB are taken, the fault is identified by comparison to the fault dictionary. Usually a fault isolation algorithm is needed for identification (e.g., minimum sum of squared error, as in Hochwald).

In general, the nominal and fault circuits are simulated under more than one *input combination* in order to achieve adequate separation of faults. Each input combination is also called an *input vector*. For multi-input multi-test nodes the corresponding *fault signature* is a matrix, called the *fault page*. The fault-dictionary is, therefore, a three dimensional array which contains the *pages* for all prospective faults.

Simulation-before-test method in digital systems

Fault dictionary, as a diagnosis method, is well known in combinatorial digital circuits (Vishnubhotla, 1980; Sugasaki, 1989; Markas 1990). Here several input sequences are applied to the network and the resultant outputs are then compared with a fault dictionary. Single stuck-type faults (faults in which one wire or gate are stuck at either 0 or 1) are usually identified in those dictionaries. Some researches have tried to extend the dictionary to multiple faults (Breuer, 1976).

1.2.2.2 Simulation-after-test method

The SAT approach (Wu, 1982; Salama, 1984) is most often used in analog circuits. In this approach, also called the claim-and-check method, the system is partitioned into subsets, where some subsets are assumed faulty and the rest are assumed fault free. This claim is then checked by the test data. If it is found to be true the faulty subset is partitioned again. The procedure is continued until the faulty element is detected.

1.2.2.3 Summary

It is reasonable to believe that a sound fault diagnosis strategy should incorporate both the SBT and SAT approaches. The fault dictionary is first used to locate hard failures. For some cases (for example analog circuits) this approach will be adequate. For the remaining cases of soft failures, we rely on the SAT approach or a combination of both approaches.

Although FDD has received much attention from researchers in the last twenty years, there are still many important issues to be studied. A short list is offered here: more effective and efficient fault isolation algorithms including the selection of test nodes and stimuli design; FDD of shop replaceable units (SRU); and investigation of the nondeterministic aspects of fault diagnosis (statistical system parameters and inputs).

1.3 ORGANIZATION OF THE DISSERTATION

The main goal of this dissertation is to develop a Catastrophic-Fault-Dictionary (CFD) of a dynamic system. Modeling of the system is done by bond graph techniques. Bond graph methods and graph algorithms are used to develop the CFD, while combinatorial methods are used to apply the CFD to an engineering problem. Several examples of using CFD for mechanical and electrical systems are provided as part of the dissertation. All the software needed for constructing a CFD for a specific system is given as well as subroutines for using this information as a design tool for engineering purposes.

The dissertation contains three main parts:

Part I: Theory and development of the proposed approach for fault detection and diagnosis;

Part II: Construction of the catastrophic fault dictionary;

Part III: Application of the CFD as a design tool.

The first part, covered in chapters two and three, includes all the theory, definitions and theoretical foundations for fault modeling using bond graph techniques, sensitivity, and the principles of the proposed fault dictionary.

In the second part, described in chapter four, the actual CFD is constructed. Here the graphical presentation of a fault diagnosis problem, together with graph algorithms, are used in order to set up the CFD for a given system. This part also provides the computer program which constructs the CFD for a given system. Mechanical and electrical examples are included.

The third part, chapter five, develops the CFD as a "design for testability" tool. The problem of how to design your system such that it will be testable is addressed here, by using the CFD as the input for those questions. Typical fault diagnosis problems in engineering systems are solved. Algorithms provided in this part can be used as a design-for-testability tool for choosing test nodes location, input location and detection of specific faults in the system.

Chapter six summarizes the current approach and describes some additional problem areas. Certain research topics were found to be very suitable for the application of fault diagnosis using bond graph methods. Those are described in detail. Other topics are mentioned briefly as possible further research areas.

The software which has been developed during this research uses the data base defined in the ENPORT software (Rosenberg, 1987). Hence it could be imbedded into the ENPORT program. At this point of the research it can be used independently on a personal computer. Listing of the software is done in Appendix E.

2. FAILURE MODELING USING BOND GRAPHS

2.1 Bond graphs - a modeling tool

Modeling of a dynamic system can take various forms such as schematic diagrams, mathematical equations, block diagrams, signal-flow graphs or bond graphs. Bond graphs are concise pictorial system representations that are useful for systems analysis and equation formulation. The bond graph language was invented by Paynter in the early sixties. Its early methodology was developed and documented by Rosenberg and Karnopp (1983).

The advantages of bond graphs are many. First, they provide insight into the system by describing the flow of energy within the physical system. Second, they elegantly support energy transfer between different energy domains. Third, bond graph elements support nonlinear constitutive relationships involving multiple output variables. Fourth, they are easily adapted for systematic input to computers for simulation of the modeled dynamic system. Last, state equations which have physical meaning can be derived directly from the bond graph models.

Bond graphs have been used in a wide variety of applications for simulation, design and analysis of mechanical systems, vibration studies, networks, hydraulic circuits and chemical systems. They have also been used in thermodynamics, heat transfer and biomedical and medical physics. A successful effort has been made in this study to adapt bond graph techniques for detection of faults in various dynamic systems.

2.2 Potential advantages of bond graph techniques for fault detection and diagnosis (FDD)

FDD is done using various simulation methods and programs. Programs like INPICE (Jagdnik,1979), SYSCAPII (Hochwald,1979) and HAFDIC (Elcherif,1983) are common computer programs used for failure investigations. The bond graph technique, as a simulation tool, may provide some potential advantages for failure detection.

An important feature of a bond graph model, from a fault detection point of view, is its graphical presentation. Graphical methods have been previously used for FDD. Ozawa (1988) solved a failure detection problem of an electrical circuit by using two graphs, the voltage graph and the current graph, and applying KCL and KVL to each. A bond graph includes both circuit graphs in the definition of the junction structure. The graphical nature of the bond graph junction structure permits easy access to every junction of the model. Associated with those junctions are flows and efforts of the physical system. In practice we associate flows with 1-junctions and efforts with 0-junctions. In that sense we can say that zero and one junctions represent effort and flow variables, respectively. This property of the bond graph plays an important role in FDD, as will be shown later.

Another important feature of bond graphs is the variety of systems it can model. Most conventional methods of fault diagnosis are limited to a specific energy domain, for example, analog circuit fault detection. A fault detection and diagnosis method which uses bond graphs as a modeling tool is more general and widely applicable to various engineering systems.

2.3 Catastrophic failures

2.3.1 Catastrophic failure modeling in bond graphs

Definition: A *Catastrophic Failure (CF)* in a bond graph is defined by a persistent zero power level on one of its bonds where the input set previously produced nonzero power.

Since the power equals effort times flow, ($P=e * f$), zero power arises from one of the following conditions:

Catastrophic Flow Failure	(CFF)	$f = 0$.
Catastrophic Effort Failure	(CEF)	$e = 0$.

<u>Definitions:</u>

A *Catastrophic Flow Failure (CFF)* is a catastrophic failure in which the flow is identically zero.

A *Catastrophic Effort Failure (CEF)* is a catastrophic failure in which the effort is identically zero.

Consider the physical interpretation of CF in electrical, mechanical and hydraulic systems. By catastrophic failures in electrical systems we refer to open circuits and short circuits. As mentioned in the literature, 70-80% of faults in analog circuits are of this type. Since flow and effort are interpreted as current and voltage respectively, a CFF corresponds to an open circuit and a CEF corresponds to short circuit.

In mechanical systems flow means velocity (or relative velocity) and effort means force. Thus zero velocity (CFF) is interpreted as sticking of a component to ground (absolute velocity failure) or sticking of two components together (relative velocity failure). Zero force on a bond is interpreted as a disconnection between two elements of the system.

We refer to volume flow and pressure as flow and effort, respectively, in fluidpower systems. CFF is interpreted as zero volume flow, meaning complete blocking of the flow, while CEF is interpreted as a significant leakage in the system, for example, due to a ruptured line.

2.3.2 Fault Sources

Two types of catastrophic failures can occur in a system. A *component failure* is associated with the component parameter or function tending to zero or infinity. A *connection failure* describes a failure in the connection between two or more subsystems or components. The third type of failure, the *SRU failure*, is actually a combination of the other failures, i.e., it consists of all possible failures that can occur in a certain subsystem.

Modeling of a component failure can be done by two approaches. In the *parameter approach* we set the parameter or function of the faulty element to zero or infinity without changing the bond graph model. In the *fault source approach* we add to the model an external input that will constrain the system to behave in the faulty mode. In bond graph terms this means we add a source of zero flow (SF) to a 1-junction, or a source of zero effort to a 0-junction, associated with the failure. The latter approach is adopted in the current study.

Modeling of a connection failure is done in the same way, by adding an external source to the simple junction which is associated with the failure.

<u>Definitions:</u>

A *Fault Source of Effort (FSE)* is a source of zero effort added to a 0-junction in order to define a catastrophic effort failure (CEF) associated with that junction.

A *Fault Source of Flow (FSF)* is a source of zero flow added to a 1-junction in order to define a catastrophic flow failure (CFF) associated with that junction.

For convenience in distinguishing fault sources from physical sources, their bonds will be drawn in dashed lines.

FSE $\overset{e\,=\,0}{- - - - \rightharpoonup}$

FSF $\overset{f\,=\,0}{- - - - \rightharpoonup}$

Let us illustrate the preceding ideas by an example. Refer to Figure 2.1. The longitudinal motion of a robotic arm can modeled by a set of masses, springs and dampers. For simplicity consider a schematic model as shown in Figure 2.1a, which contains three masses *(m1, m2, m3)*, two springs *(k1, k2)*, and two dampers *(b1, b2)*. An input force, F, is acting on one end and the velocity, V, is measured at the other end. The system bond graph is

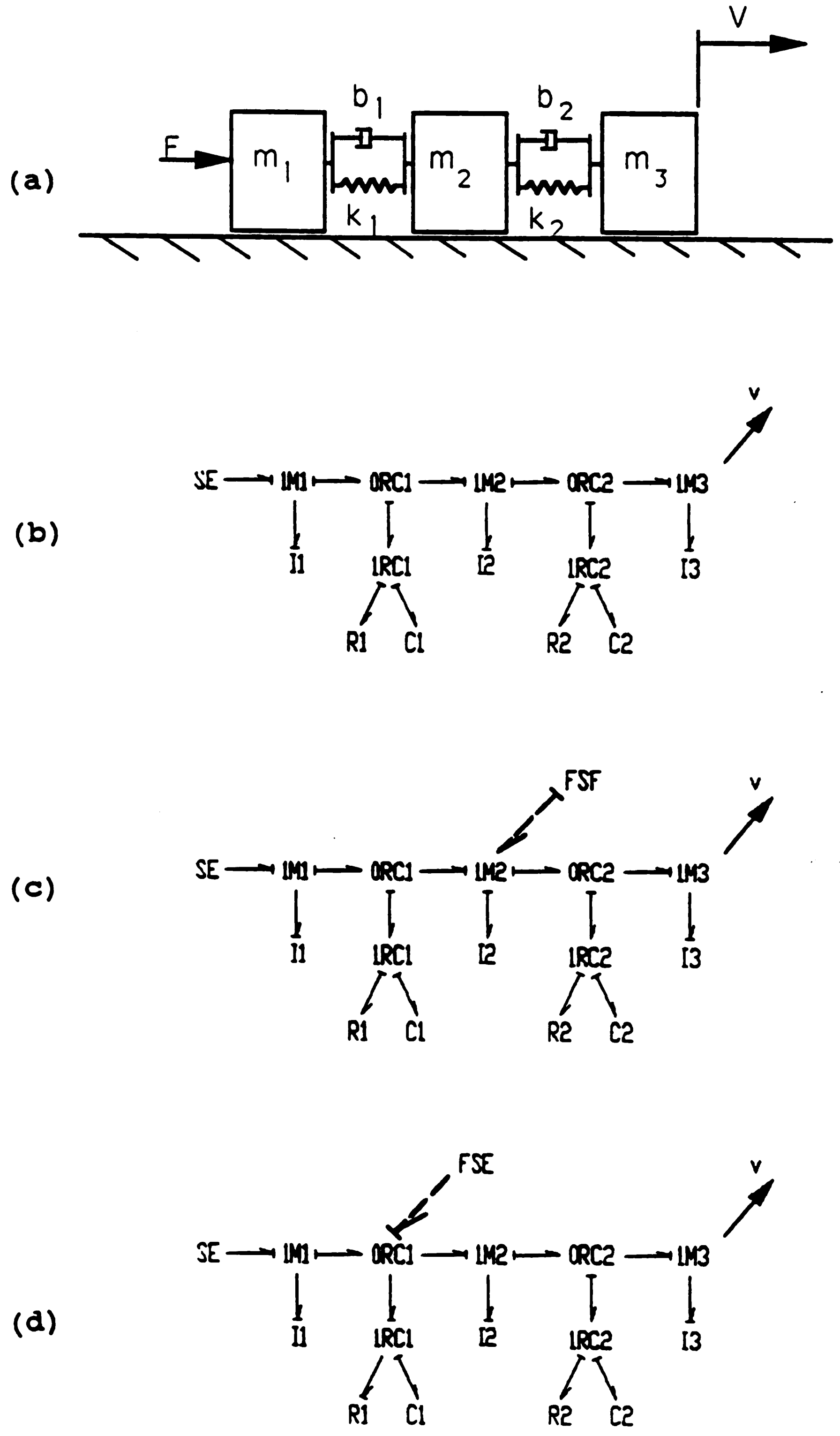


Figure 2.1: Robotic arm: (a) Schematic diagram; (b) bond graph model; (c) modeling of a component failure; (d) modeling of a connection failure.

shown in Figure 2.1b. Suppose the mass *m2* is stuck (i.e., the velocity of *m2* is zero), which is a component failure. How do we model it in the bond graph? In the *parameter approach* we set the mass *m2* to infinity which leaves the model graph unchanged. In the *fault source approach* we add a FSF to 1M2 junction (Figure 2.1c).

A connection failure can be illustrated as follows:
Suppose mass *m2* is disconnected from the left spring and damper, i.e., there is no force between the two subsystems. This connection failure can be modeled using the *fault source approach* by addition of a FSE to the 0-junction that describes this force (0RC1), as shown in Figure 2.1d. Note that the causality of the bond graph, shown by the short transverse strokes on each of the bonds is changed as the fault source is added to the system. Causality concepts, operations, and the standard causal assignment procedure are discussed in several references (e.g., Rosenberg and Karnopp, 1983), and will not be reviewed here. The causality change and its physical interpretation plays an important role in constructing the CFD as will be shown later.

2.3.3 Modeling considerations of connection failures

Connection failures are basically defined in a similar way as component failures. It is known that there may be different bond graphs that represent the same physical system in terms of equations implied. When the purpose of constructing the bond graph is fault detection the situation is different. Those equivalent graphs may represent different connections between components of the system. Thus, when modeling is done for fault detection, special consideration should be made when constructing the graph. Here, not only the components and the connections should be presented in the graph, but also the subassemblies of the system. In order to be able to detect connection failures between subassemblies, each physical connection should be associated with a simple junction.

Let us illustrate this idea by an example. Figure 2.2a represents two masses connected by two springs and a damper. From the conventional modeling point of view, all four bond graphs shown in Figure 2.2b - 2.2e describe the same dynamic system. However, when connection failures are of concern, each bond graph defines a different assembly of the system, as shown in the schematic beside each bond graph. For example, a broken connection at point A can only be modeled by using the bond graph in Figure 2.2b. This is done by setting the effort e.1 to zero. This failure can not be modeled in the other bond graphs since this variable does not exist there.

In order to construct an appropriate bond graph for a given system, one can a use the macro mode approach (Rosenberg, 1986). The graph should be built in steps, starting from the top level assembly. In each step the macros (or subassemblies) are defined in detail. A demonstration of the procedure is shown in Figure 2.3. The bond graph is constructed in three steps, using MACRO1 and MACRO2 as subassemblies. Note that the connection forces at points A, B, and D, are defined explicitly by e.2, e.3 and e.10 respectively. The fact that connection forces are defined as variables in the bond graph explicitly aids the modeling of connection failures, as

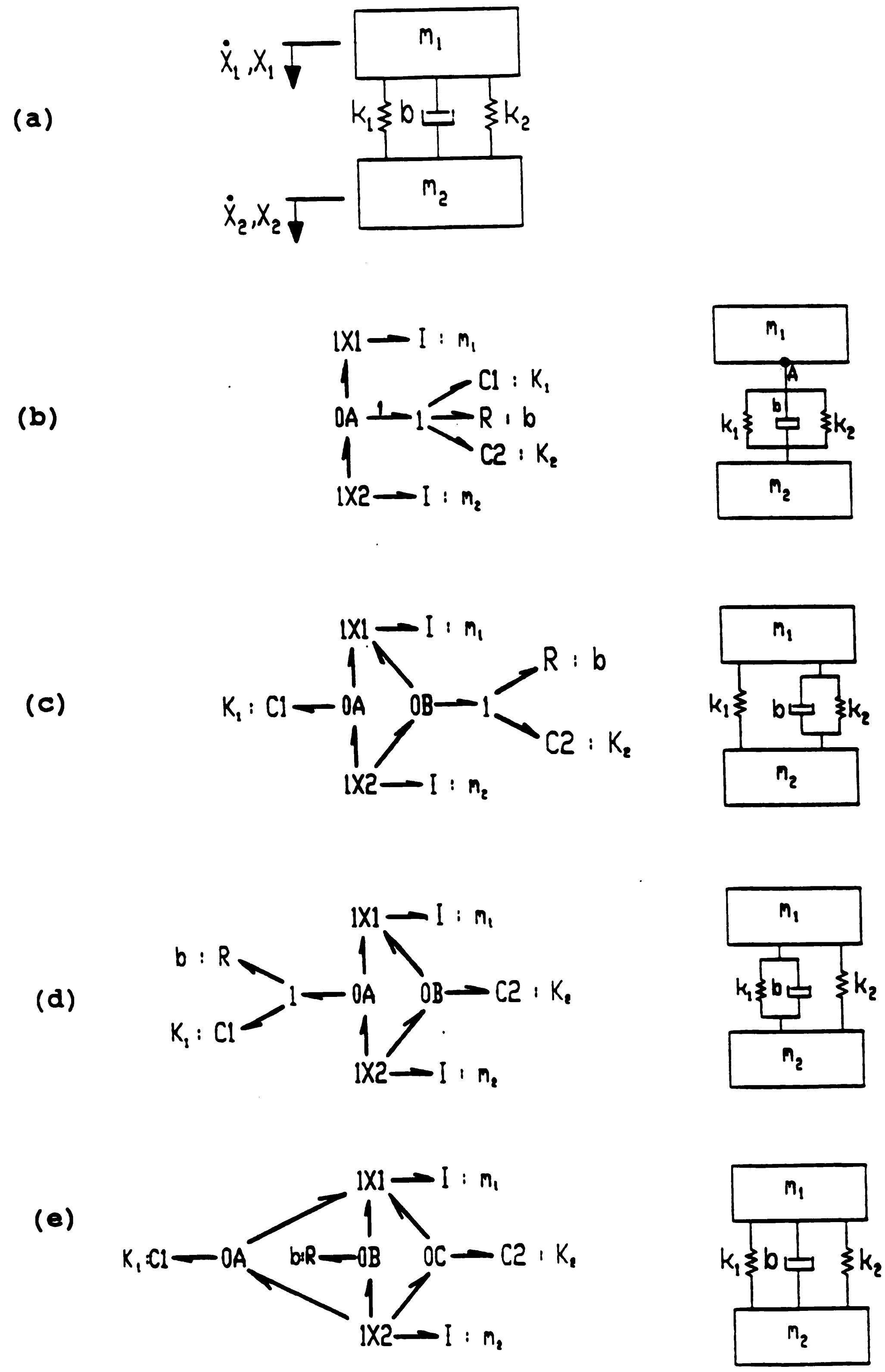


Fig. 2.2: Modeling of connection failures

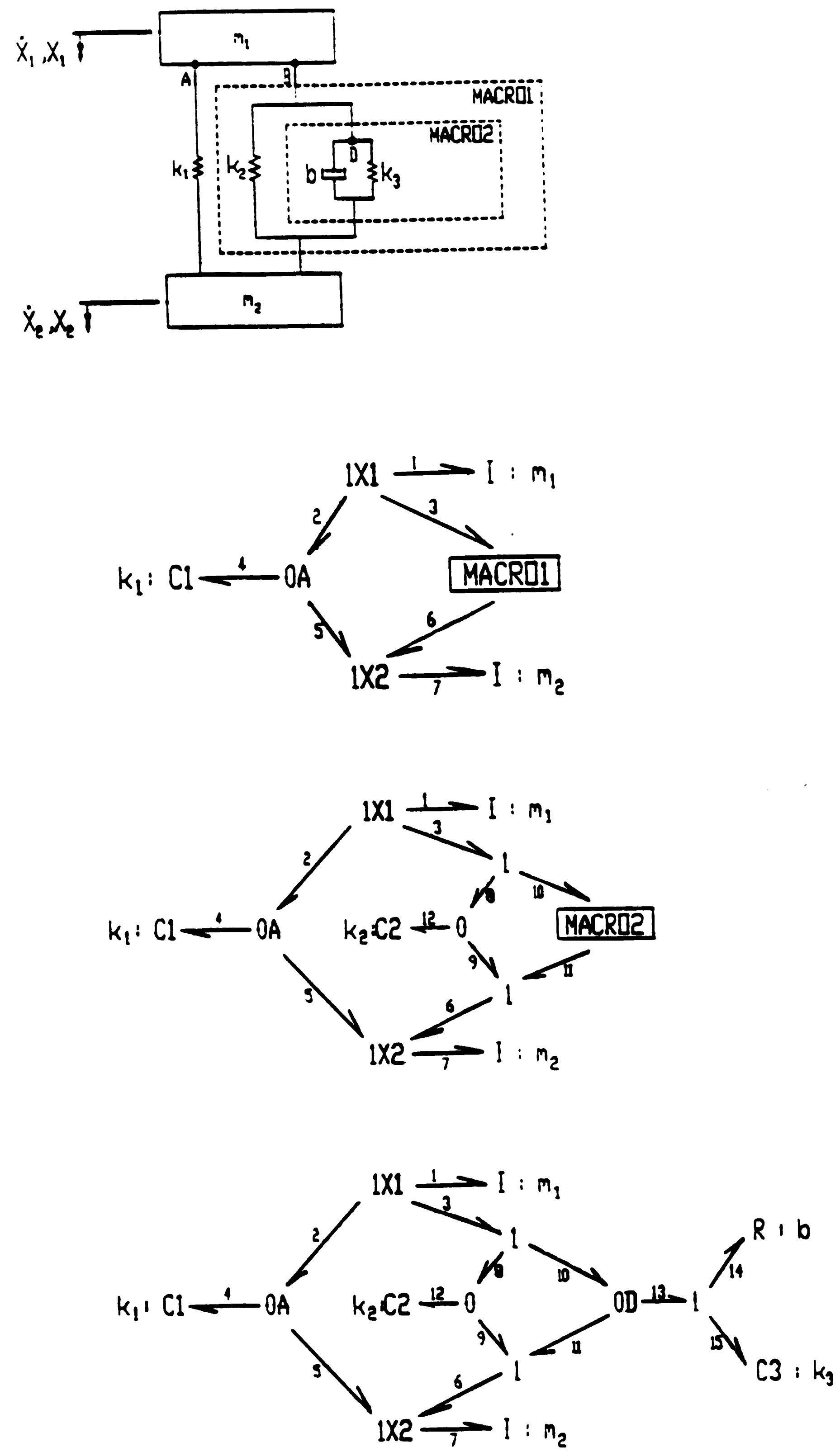


Fig. 2.3: Macro modeling

well as other design purposes, perhaps.

2.4 Proper bond graph

Every flow and effort that appears in the bond graph can be treated as a candidate for failure. Therefore each flow and effort should be associated with a simple junction in order to define that failure in the fault source approach. We may need to augment our initial bond graph model such that it will satisfy this property.

Definitions:

A *proper bond graph (PBG)* is a bond graph in which every flow variable is associated with a 1-junction and every effort variable is associated with a 0-junction.

A *Bond Structure (BS)* contains a specific bond together with the two nodes adjacent to it.

A *Simple Junction (SJ)* is a 0-junction or a 1-junction (Rosenberg, 1979a).

An *Identity Junction (IJ)* is a simple junction that has degree equal to two and contains one inwardly-directed bond and one outwardly-directed bond.

The PBG is constructed by adding identity junctions to the original bond graph. Addition of identity junctions to a bond graph does not change the behavior implied by the bond graph. The PBG and the original graph present the same physical system. The following algorithm will construct a proper bond graph from a given bond graph.

<u>An algorithm for constructing the PBG</u>

1. Scan all bonds (edges) of the graph and add identity junctions and edges according to Table 2.1. N1, N2 are any nodes except simple junctions.

Table 2.1: Insertions for constructing a proper bond graph

bond structure in a given bond graph	equivalent bond structure in the proper bond graph
$0 \xrightarrow{i} 0$	$0 \xrightarrow{i} 1 \xrightarrow{i1} 0$
$0 \xrightarrow{i} 1$	unchanged
$1 \xrightarrow{i} 0$	unchanged
$1 \xrightarrow{i} 1$	$1 \xrightarrow{i} 0 \xrightarrow{i1} 1$
$1 \xrightarrow{i} N1$	$1 \xrightarrow{i} 0 \xrightarrow{i1} N1$
$0 \xrightarrow{i} N1$	$0 \xrightarrow{i} 1 \xrightarrow{i1} N1$
$N1 \xrightarrow{i} 1$	$N1 \xrightarrow{i} 0 \xrightarrow{i1} 1$
$N1 \xrightarrow{i} 0$	$N1 \xrightarrow{i} 1 \xrightarrow{i1} 0$
$N1 \xrightarrow{i} N2$	$N1 \xrightarrow{i} 0 \xrightarrow{i1} 1 \xrightarrow{i2} N2$

Using the definitions of a proper bond graph, a catastrophic failure and a fault source, consider the following theorem.

Theorem 2.1: Every catastrophic failure in a multiport system can be modeled by addition of either a fault source of flow or a fault source of effort to the proper bond graph of the system.

Proof: For the sake of brevity proofs are provided in APPENDIX A.

Thus, there is a group of simple junctions that are associated with a given list of faults. These junctions play a role in the graphical formulation of the failure detection problem.

2.5 Graphical presentation of FDD problem

As shown before, every prospective catastrophic failure ($e=0$, $f=0$) in a dynamic system is associated with a specific simple junction. The fault list which is provided by the designer, based on his/her experience, can be translated into a list of simple junctions named the *fault junctions*.

Physical inputs are represented in the bond graph by two types of sources: source of flow (SF) and source of effort (SE). The flow variable of a SF is associated with the 1-junction adjacent to it and the effort variable is associated with the 0-junction adjacent to it. These junctions will always appear in the proper bond graph and will be referred to as *input junctions*.

For this study we restrict ourselves to test node measurements (outputs) that are flows, efforts or a function of them. This restriction covers most practical systems, since voltage measurements commonly are used in electrical systems; force, velocity, acceleration and displacement are effort, flow, and functions of them in mechanical systems; and pressure (effort variable) is common as a measured signal in hydraulic systems.

Since every flow and effort is associated with a simple junction of the proper bond graph, we can refer to these nodes as *test-node junctions* or *output junctions*. Test-nodes/outputs will be used as synonyms throughout this work.

So far we realize that a FDD problem in a bond graph is associated with three special groups of simple junctions: the fault junctions, the input junctions and the test nodes junctions. Figure 2.4 illustrates that situation abstractly. One group of simple junctions represents the inputs and another group represents the output junctions. In the remaining junction structure we can find the junctions that are associated with the faults of the fault list. Note that there may be simple junctions in the graph that do not belong to any of the special groups above.

Test node measurements are actually signals extracted from simple junctions. In order to preserve the structure of the graph and the duality between inputs and outputs, test node measurements are represented by either a source of zero effort extracted from a 1-junction (where flow is measured) or a source of zero flow extracted from a 0-junction. Figure 2.5 shows the equivalency between these signals and sources.

By defining the input, fault and test node junctions we have transferred a failure detection problem into a graphical problem. Thus graphical methods are applicable for solving the problem, as will be developed in the subsequent chapters.

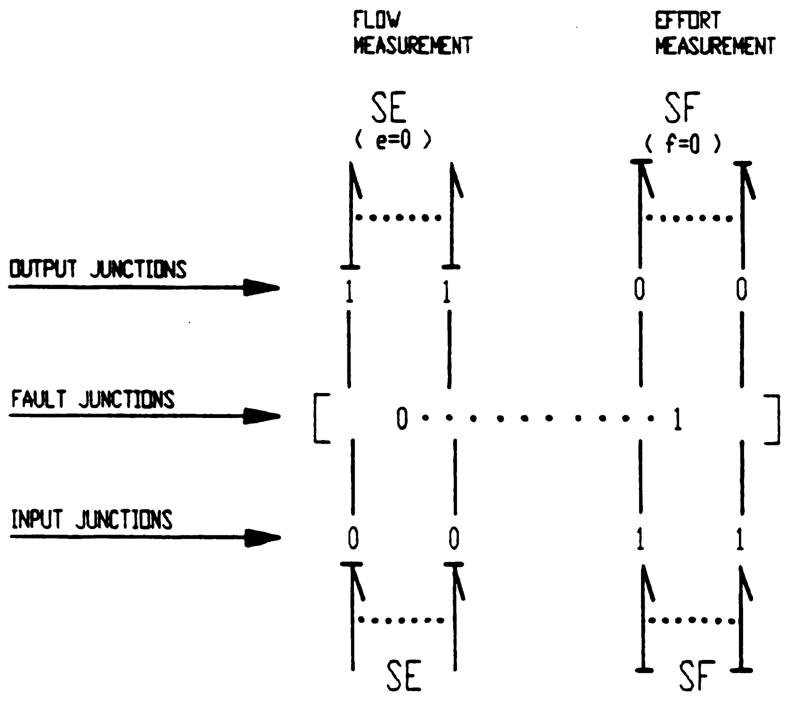


Figure 2.4: Abstract description of FDD problem in a bond graph model.

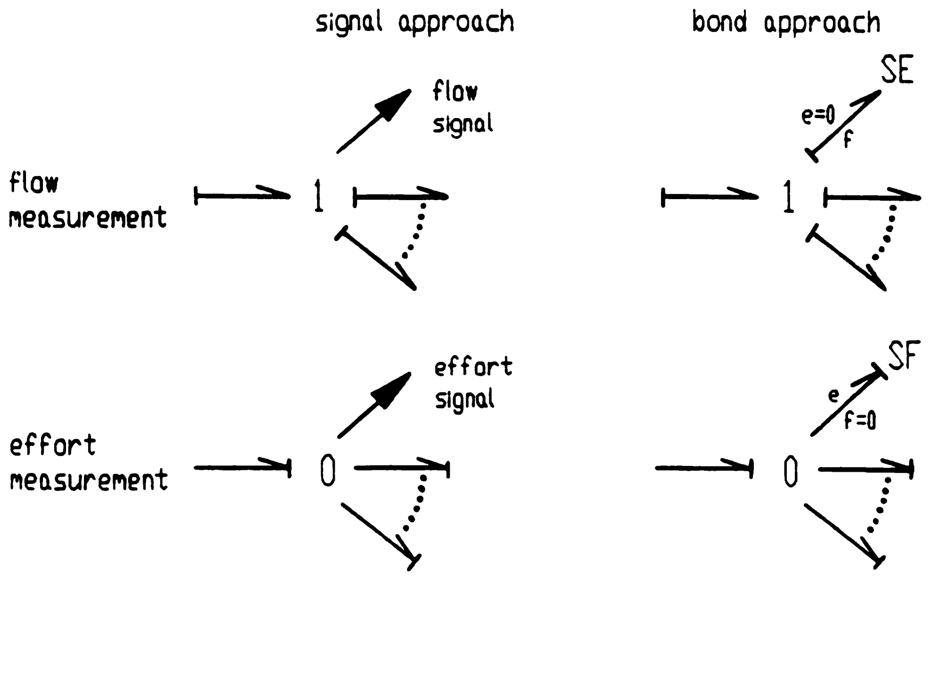


Figure 2.5: Equivalent representation of test-nodes measurements

3. THE CATASTROPHIC FAULT DICTIONARY

The conventional fault dictionary contains a "page" of information for every fault in the fault list. This data consists, usually, of the test-node measurements. We suggest a special type of sensitivity measure to be the entries of the dictionary. This dictionary can be directly derived from the graphical presentation. The special type of sensitivity, called *Structural Sensitivity* is defined in section 3.2. The procedure of deriving those entries from the graphical presentation is described in section 3.3 and following. It is first done for a tree graph and then expanded to a general connected graph. For the sake of continuity, proofs and minor cases are treated in appendices.

3.1 System bond graph model

The requirements for the system bond graph model are:

(1) The bond graph model contains only bonds (no signals allowed).

(2) The bond graph model is a connected graph. If the graph is unconnected, every component can be treated individually.

(3) The standard bond graph elements are:

C:	generalized capacitance,
I:	generalized inertance,
R:	generalized resistance,
SE:	effort source,
SF:	flow source,
0:	zero junction,
1:	one junction,
TF:	transformer, and

GY: gyrator.

C, I, and R elements can have any number of ports. Constitutive equations for the C, I, and R elements, linear or nonlinear, are functions only of the local port variables (and their appropriate integrals, for C and I). Source nodes (SE and SF) are functions of time only. TF and GY nodes have constant parameters.

3.2 The Structural Sensitivity Matrix (SSM)

Suppose a dynamic system contains l inputs and m outputs. Let $U_i(t)$ be the ith input and $Y_j(t)$ be the jth output of the system. $Y_j(t)$ is defined for given inputs and initial conditions. Let us restrict ourselves to stable systems in which a constant input defines a unique constant steady-state output.

Let S_{ij} denote a *sensitivity* of the j^{th} output with respect to the i^{th} input, where there is a perturbation in a constant input and the steady state output is observed. The sensitivity, S_{ij}, is

$$S_{ij} = \frac{\Delta Y_{ssj}}{\Delta U_{ci}} \qquad (3.1)$$

where ΔU_{ci} is a constant perturbation in the ith input and ΔY_{ssj} is the resulted change in the jth steady-state output.

In the same way we can define a lxm *Sensitivity Matrix, SM,* of the system as

$$SM = \begin{bmatrix} \frac{\Delta Y_{ss1}}{\Delta U_{c1}} & \cdots & \frac{\Delta Y_{ssm}}{\Delta U_{c1}} \\ \vdots & & \vdots \\ \frac{\Delta Y_{ss1}}{\Delta U_{cl}} & \cdots & \frac{\Delta Y_{ssm}}{\Delta U_{cl}} \end{bmatrix} \qquad (3.2)$$

If the output Y_{ssj} does not depend on the input U_{ci}, then the entry S_{ij} in the sensitivity matrix is identically zero. Starting with the SM concept we define a *Structural Sensitivity Matrix, SSM*.

<u>Definition:</u> A *Structural Sensitivity Matrix, SSM*, is a sensitivity matrix whose entries are zero when S_{ij} is identically zero, and one otherwise.

From a practical point of view, every entry of the SM (and SSM) can be found by perturbing one input while holding the others fixed, and observing the resulting output. If a perturbation in U_{ci} does not change the output Y_{ssj}, then the corresponding entry to SSM is zero; otherwise it is one. Note that the initial conditions for both cases (original and perturbed) should be the same.

Some nonlinear systems may provide a zero entry in the SSM at a specific singular operating point. These situations should be avoided by determining the SSM entries at different operating points of the system.

A SSM whose all entries are one is a *full SSM*. Otherwise the matrix is *partial*.

3.3 Connection between graphical presentation and SSM

Consider a physical system with defined inputs, outputs (test-nodes measurements) and fault-list variables. We wish to derive the CFD for this system. Let us restrict ourselves to the following:

(1) The system bond graph is a tree.

(2) The SSM of the nominal system is full, i.e., every output is a function of every input.

(3) The fault-list contains only single catastrophic faults (i.e., each fault corresponds to a single $e=0$ or $f=0$ statement).

In the following discussion we treat the bond graph as an undirected graph, since power orientation is not relevant.

Let P_{ij} denote a path from node i to node j in a bond graph that is a tree, where node i is an input junction node and node j is an output junction node. The path between any two nodes in a tree graph is unique. We express the path P_{ij} as the set of nodes between (and including) i and j.

The following theorem is essential in deriving the CFD.

<u>Theorem 3.1:</u> S_{ij} is identically zero if and only if there exists a catastrophic failure associated with a simple junction on P_{ij}.

Proof: Provided in APPENDIX A

Let us demonstrate the preceding theorem in a physical system. As an example consider the robotic arm shown in Figure 2.1. Here we have one input (force, F) and one output (velocity, V). The system satisfies all the preceding restrictions. Any catastrophic failure along the path from SE to 1M3 is either a stuck mass (CFF associated with 1M1, 1M2 or 1M3) or a connection failure between two masses (CEF associated with 0RC1 or 0RC2). Either type of failure will cause zero transmission of input force effect to the system downstream of the fault. Thus the sensitivity of V with respect to F will be identically zero. On the other hand, if a failure of a consisting stuck damper *b2* occurs, which is a CFF associated with 1RC2, two adjacent masses will be connected into one rigid body. This will cause a change in the sensitivity but will not make it identically zero.

3.4 Definition of CFD

As a result of Theorem 3.1, a zero entry in the SSM is an indication of the existence of a catastrophic fault. This is true when the system contains a full nominal SSM. Thus, we can use the SSM as a signature of a specific fault. Suppose we derive the SSM for every faulty system that contains a fault from the fault list. Then their collection forms the fault dictionary. For convenience, we put all these matrices in a three dimensional array. The catastrophic dictionary is, therefore, a three dimensional boolean array. This array is referred to as the *CFD array*. Figure 3.1 shows this array symbolically, where each horizontal matrix in it is the SSM of a specific fault. Each SSM is referred to as a *page* of the dictionary. During testing the experimental SSM of the actual system is compared to these pages and the actual fault is defined.

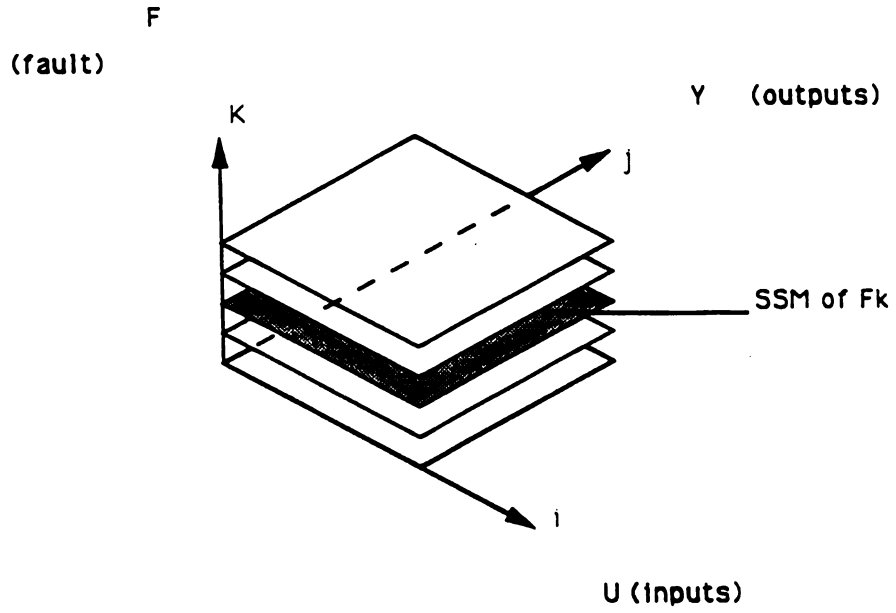


Figure 3.1: CFD array

3.5 Construction of CFD for a tree bond graph

The CFD can be constructed directly from the bond graph of the system. Since the graph is a tree, every path P_{ij} is unique. The procedure of constructing the CFD consists of two main steps. In the first step the FDD problem is converted to the graphical representation. In the second step we use graph algorithms to find the paths from each input to each output. Each path is checked for existence of fault-junctions along it, and the corresponding entries are put in the CFD array. The software developed for this task, referred as the *CFD tree algorithm*, is described in detail in the next chapter. Figure 3.2 shows its flow diagram.

3.6 CFD for a general graph

Since most systems do not have a tree bond graph, we have to expand our method to a general bond graph. We still restrict ourselves to the following assumptions:

(1) The system bond graph is a connected graph;

(2) The SSM of the nominal system is full;

(3) The fault-list contains only single catastrophic faults.

In a general connected graph Pij is not unique. Our approach is to generate an equivalent graph which abstracts the relevant information and is a tree graph. Then the CFD tree algorithm can operate on it. To this end we introduce the *Modified-Block-Graph, MBG*.

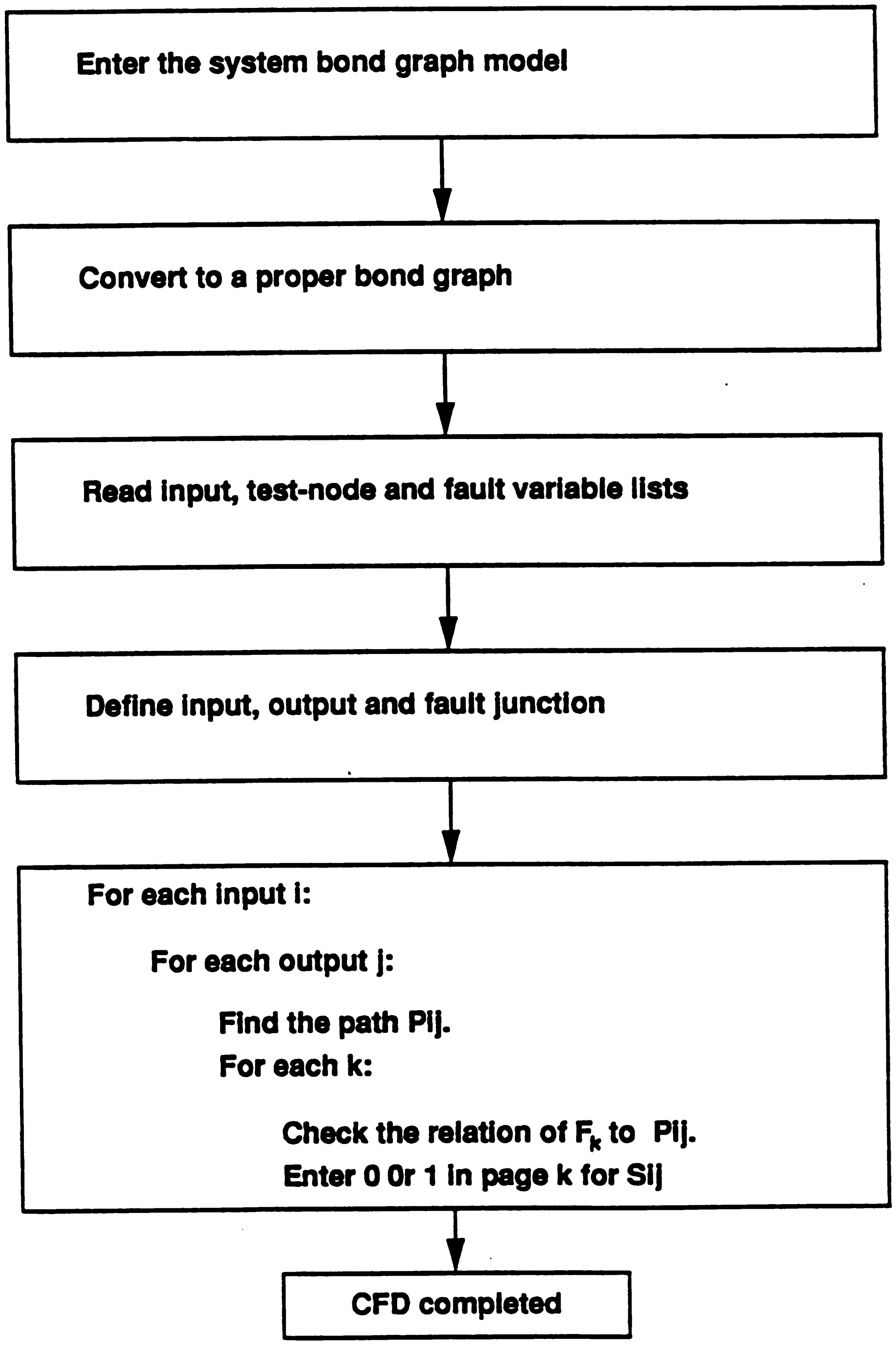

Enter the system bond graph model
Convert to a proper bond graph
Read input, test-node and fault variable lists
Define input, output and fault junction
For each input i:
For each output j:
Find the path Pij.
For each k:
Check the relation of F_k to Pij.
Enter 0 0r 1 in page k for Sij
CFD completed

Figure 3.2: Flow diagram of CFD tree algorithm

3.6.1 The Modified-Block-Graph

Definitions:

A *Modified-Block-Graph, MBG(G)*, of a connected graph G is a graph derived from G such that every block in G is defined by a node, and the cut nodes that connect the blocks remain in the modified block graph.

A *block node* is a node in MBG(G) which defines a block in G.

A *cut node* is a node whose removal from a connected graph (together with its adjacent edges) results in an unconnected graph.

MBG(G) is a modification form of the *Block-Cut-vertex-graph, BC(G)*, which is known in graph theory (Chartrand and Lesniak, 1986). The only difference between the two lies in efficiency, since in our case we ignore the trivial blocks (blocks which contain one bond). MBG(G) and BC(G) have the same properties (proved in the above mentioned reference).

The interesting properties of MBG(G) are:

(1) The MBG is a tree.

(2) Any path from node S to node T in G contains the same cut nodes as the path from S to T in MBG(G), where S and T are either defined in MBG(G) or captured in the block-nodes of MBG(G).

The first property allows us to use the CFD algorithm developed for a tree graph, while the second helps us to convert paths from G to MBG(G). An illustration of a graph G and its corresponding MBG is shown in Figure 3.3. Note that all the cut-nodes of G remain in MBG(G), while the nodes that are inside a block disappear since the block is replaced by a block-node. A detailed procedure for constructing the MBG(G) is represented in APPENDIX B.

Determination of the blocks of a given graph is a known algorithm in graph theory. It is one of the applications of the Depth-First Search method, and is known as "The Depth-First Search for blocks algorithm" (Even, 1979; Gibbons, 1985). The subroutines that find the blocks and construct the modified-block-graph are described in detail in Chapter 4.

3.6.2 Detectable and undetectable faults

In a general connected graph G all the paths from U_i to Y_j are reduced to one *Equivalent Path*, *EP_{ij}*, in MBG(G). This path is defined by U_i and Y_j, since MBG(G) is a tree. EPij represents the original paths. It contains all cut-nodes of P_{ij}, but it does not represent nodes that were inside the blocks.

Definitions:

A *detectable fault* is a fault whose existence in the system changes the SSM of the nominal system.

An *undetectable fault* is a fault whose existence in the system does not change the SSM of the nominal system.

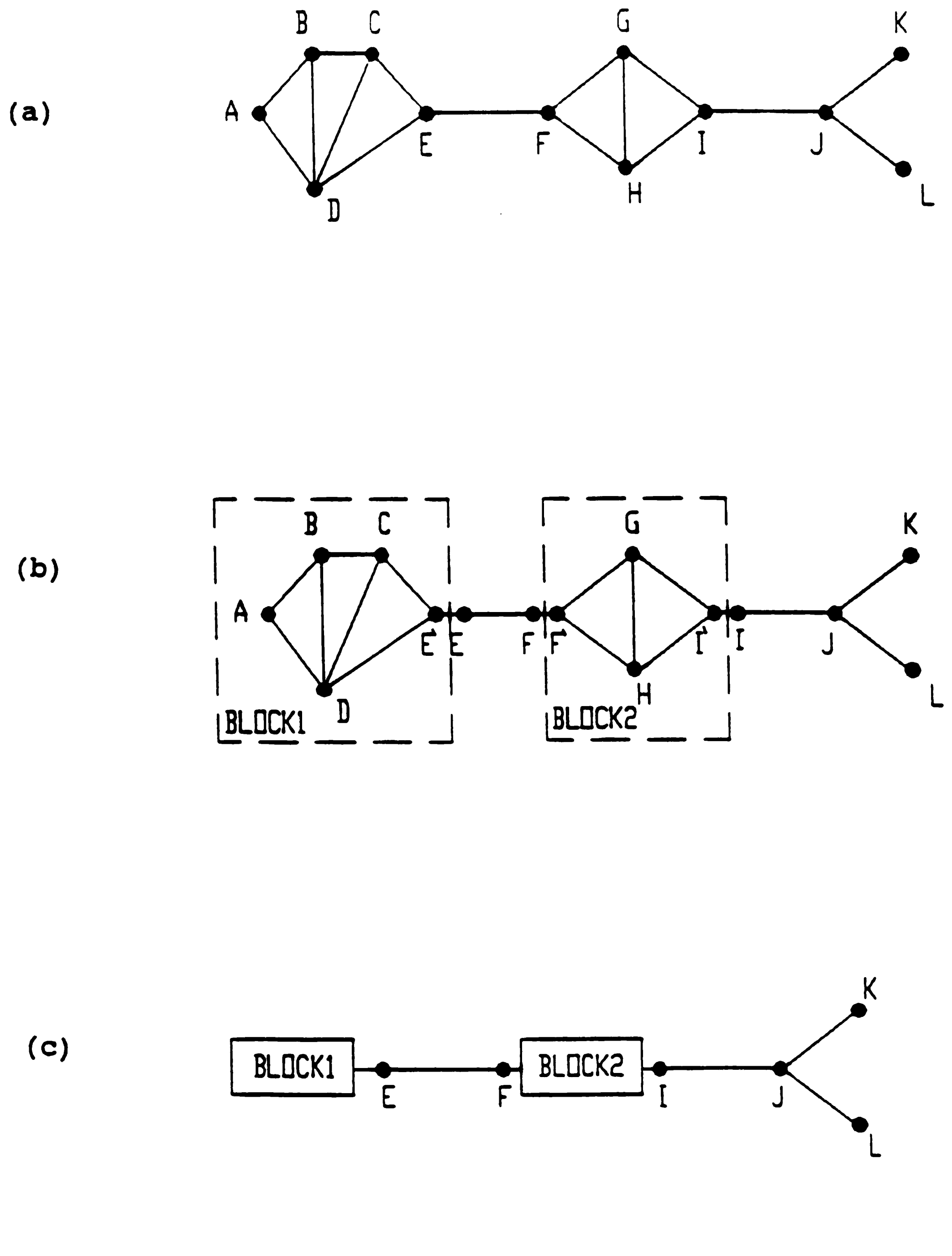


Figure 3.3: Construction of the Modified-Block-Graph: (a) Original graph; (b) Definition of blocks; (c) MBG completed.

Faults associated with junctions that disappear through the process of converting the graph to the MBG are undetectable by the current approach since they do not change the SSM entries. From a graphical point of view it can be explained as the following: If a path from point S to point T in the MGB contains a block-node, there is no information about the path inside the block. For example, refer to Figure 3.3: point H is inside BLOCK2. A path from K to E may contain H (K-J-I-H-F-E) but there exists an alternative path that does not contain H (K-J-I-G-F-E). Cut points, on the other hand, like I or J, appear in both paths, thus they are contained in the equivalent path (Figure 3.3c, equivalent path defined by K-J-I-BLOCK2-F-E).

A fault may be undetectable for either of two reasons. Either its fault-junction disappears during the conversion of the graph to the MBG, or its fault-junction does not appear on any input-output path. The two types are defined as:

(1) *Structurally undetectable fault:* this is a fault associated with a junction that disappears through the process of converting the graph into the MBG. Such faults can not be detected by the CFD approach. Fortunately, their appearance in engineering systems is rare.

(2) *Undetectable fault:* fault whose junction does not appear on any path P_{ij}. A change in test-node locations can change their status. A discussion about this is presented in the design-for-testability chapter (Chapter 5).

Two examples are discussed next to give some insight into the detectability of faults. The mechanical system of Figure 3.4a contains a thick beam with two input forces, F1 and F2. A small mass, m, is connected to the beam through a spring. The bond graph model, Figure 3.4b, contains two loops. The MBG is shown in Figure 3.4c. Since no simple junction has disappeared through the process of converting the graph into the block-graph, there are no structurally-undetected faults in the system. Note that the transformer nodes do not appear in the MBG. Nevertheless, no prospective faults were lost since they are associated with the simple junctions connected to those transformers. Suppose an accelerometer is located on the small mass, which means that the output junction is 0D. Using F1 and F2 as inputs, we can observe that faults associated with 1T and 1R are undetectable since they do not appear on the path from the inputs to 0D. A change of output location may convert those faults into detectable faults. Thus, faults associated with 1R and 1T are undetectable, but not structurally undetectable.

The network shown in Figure 3.5a (from Rosenberg and Karnopp, 1983) contains two inputs, a voltage source and a current source. Its bond graph is unicyclic. The corresponding MBG is shown in Figure 3.5c. Note that none of the junctions disappear during the transformation. Here, again, none of the faults is structurally-undetectable. The detectability of the faults depends on the choice of test-node location.

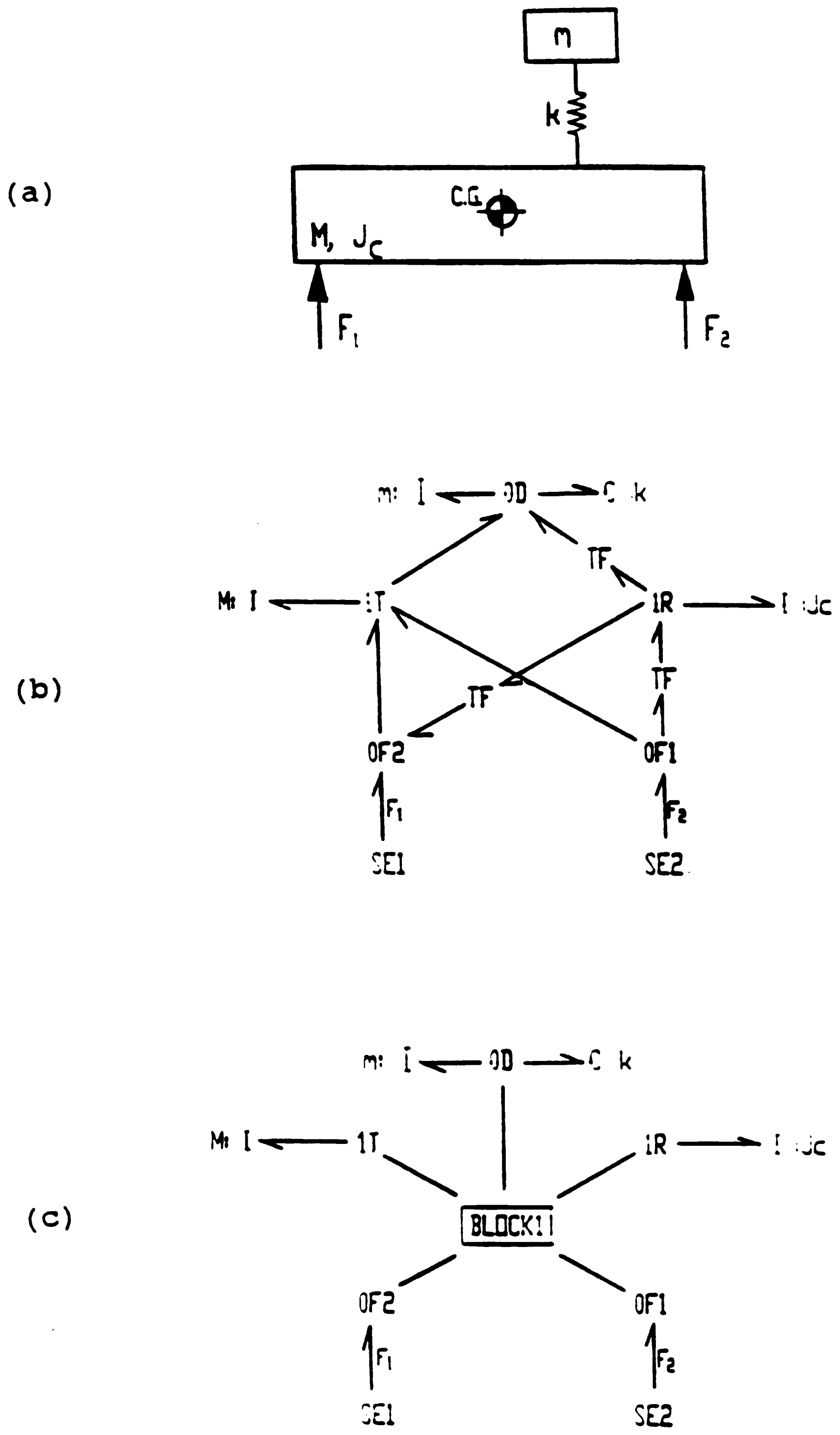


Figure 3.4: MBG of a mechanical system: (a) schematic diagram; (b) bond graph model; (c) MBG.

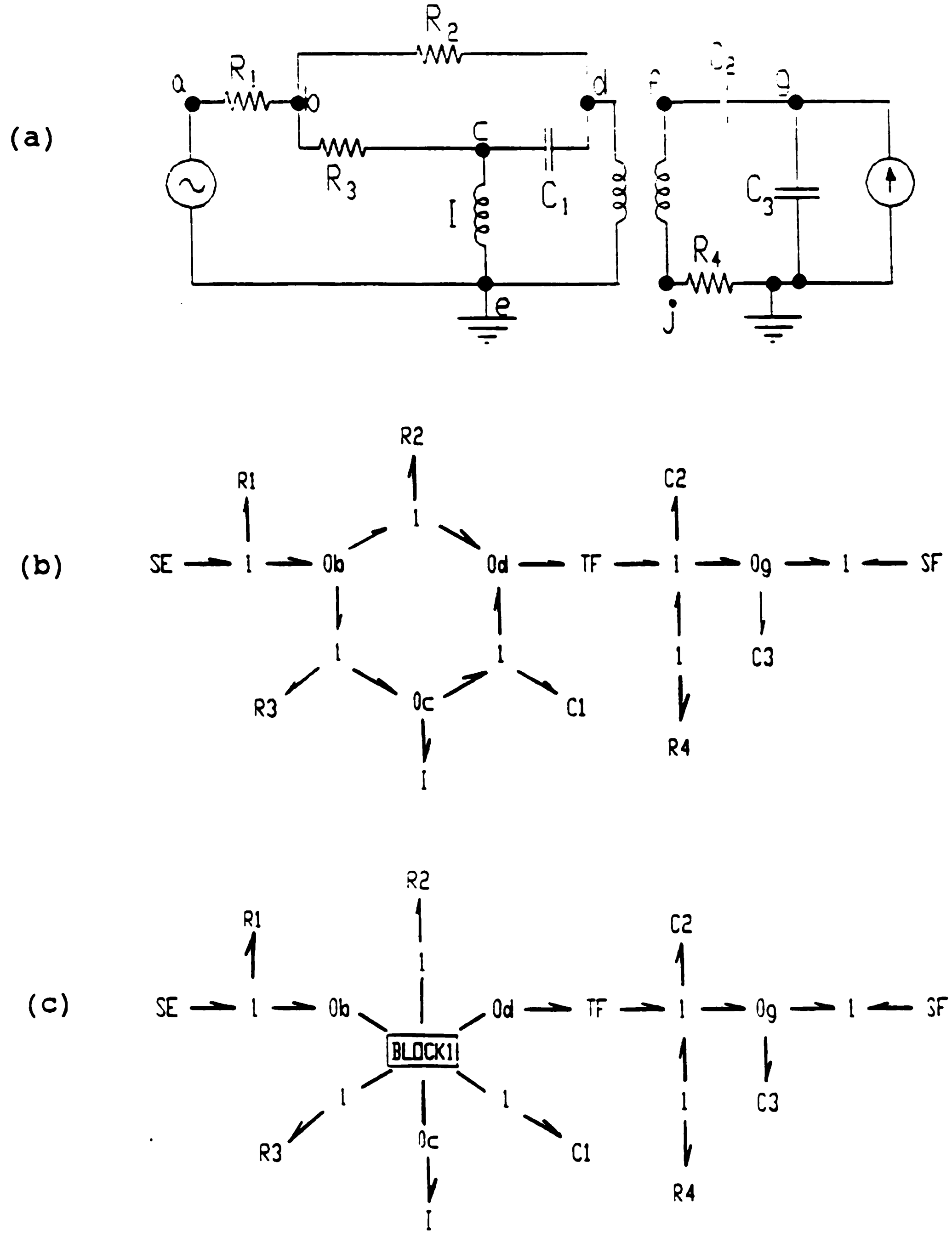


Figure 3.5: MBG of a network: (a) schematic diagram; (b) bond graph model; (c) MBG.

The only loss of fault information when a proper-bond-graph is transformed to MBG is associated with simple junctions that do not appear in the MBG. In both examples the number of simple junctions did not change when the bond graph was converted into the MBG. This situation is very common to systems whose bond graph contains 1-port elements and 2-port transformers and gyrators. Thus most of the prospective faults in those systems are detectable. We note that the existence of loops (and therefore blocks) in the bond graph corresponds to multiple paths of energy in the system. Many common dynamic engineering systems, with the notable exception of electronic circuits and reduntant mechanical structures, do not have many parallel energy paths. Therefore the number of structurally-undetectable faults in them typically is very small.

3.7 The partial SSM

So far we have dealt with a full SSM. Thus a zero entry in an experimental SSM was an indication of a fault. Most engineering systems satisfy that condition. But what if the nominal SSM contains zero entries? This is common to systems whose bond graph models are unconnected, but it occurs in connected bond graphs too. Actually, the entries of the nominal SSM can be derived directly from its bond graph representation. See details in APPENDIX C. In this section we provide a procedure for constructing the CFD for a given partial nominal SSM (i.e., the nominal system has a partial SSM whose zero locations are known).

Construction of the fault dictionary for partial SSM

Suppose that the entry S_{ij} in a nominal SSM is zero. In this case any fault along EP_{ij} is undetectable. Moreover, a zero entry in an SSM is no longer an indication of a fault. For this case we need to add the nominal SSM to the dictionary. Since every SSM of a faulty system will contain those initial zero entries, the SSM of the nominal system will be added to each SSM. Mathematically it is computed as the following:

$$SSM_{F1} = SSM_{F1}^{*} \oplus SSM_{N} \qquad (3.3)$$

where

SSM_N is the SSM of the nominal system;

SSM_{F1}^{*} is the SSM of the faulty system, derived from the bond graph as shown before;

SSM_{F1} is the final page of the dictionary.

The addition property ($\oplus$) of equation 3.3 is identical to the *intersection* operation in boolean algebra (Goodstein, 1964). Its truth table is

p	q	$p \oplus q$
0	0	0
0	1	0
1	0	0
1	1	1

After the CFD has been constructed, the experimental SSM is compared to the dictionary as before, but in this case the nominal SSM is also a page in the dictionary, usually defined as F0 (fault 0). A fault is undetectable if its fault-page equals page F0, the nominal SSM.

3.8 Additional properties of CFD

3.8.1 Ambiguity sets

Sometimes different faults contain the same SSM. They create an *ambiguity set* and are detected as a group. This phenomenon is well known in detection of faults in analog circuits (Lin, 1985) as well as in digital systems (Markas, 1990). If the unseparated faults are located in the same "Shop Replaceable Unit", further isolation may not be necessary. Otherwise, if unique detection is required, more or different test-nodes are needed. This topic is also discussed in the design chapter.

3.8.2 Fault identification by dictionary lookup

As mentioned earlier, actual faults are identified by comparison of the actual SSM to all SSMs of the fault dictionary. In a conventional fault dictionary this is a time consuming process, since outputs are defined as real numbers. An attempt to reduce it into a an integer coded dictionary has been made by Lin (1985). One of the main advantages of the CFD is the easy lookup, due to its binary structure.

3.8.3 Superposition

A multifault (i.e., the simultaneous occurrence of a set of single faults) is treated in conventional dictionaries as an independent fault. These dictionaries usually do not contain multifaults due to limitation of memory size. The binary structure of the CFD allows us to use superposition, i.e., the SSM of a fault defined by two or more faults from the fault list is a combination (using the + operation as in section 3.7) of their SSMs. This fact allows us to detect multifaults without increasing the memory significantly.

3.9 Summary

The theoretical foundations for constructing the CFD were described. A procedure for deriving the CFD directly from the graphical representation of the FDD problem was presented for a tree bond graph. A general bond graph was treated in two steps. First, the graph is converted to a modified block graph which captures all the relevant information and is also a tree. Second, the CFD tree algorithm is applied to the MBG to obtain the CFD. Faults associated with junctions that disappear in the procedure of converting the original graph to the MBG are undetected faults. They are rare in practical engineering problems.

4. CONSTRUCTION OF THE CATASTROPHIC FAULT DICTIONARY

4.1 General description

A FORTRAN computer program called CFDIC (Catastrophic Fault Dictionary) has been developed to generate the fault dictionary. The program is based on the theoretical foundations described in Chapter 3. The current chapter describes the basic features of this program. The design applications of the CFDIC software are presented in the following chapter. CFDIC program can be run independently on a personal computer using FORTRAN-77 compiler. The software is described in detail in two appendices. APPENDIX D, the user manual of CFDIC, provides directions for installing and running CFDIC including a detailed example. APPENDIX E contains the listing and implementation notes of the program.

Figure 4.1 shows the main inputs and outputs of the program. The inputs are: (1) the model description, defined by its bond graph (i.e., node list and incidence list); (2) input, test-node and fault variables definitions, stored in a data file.

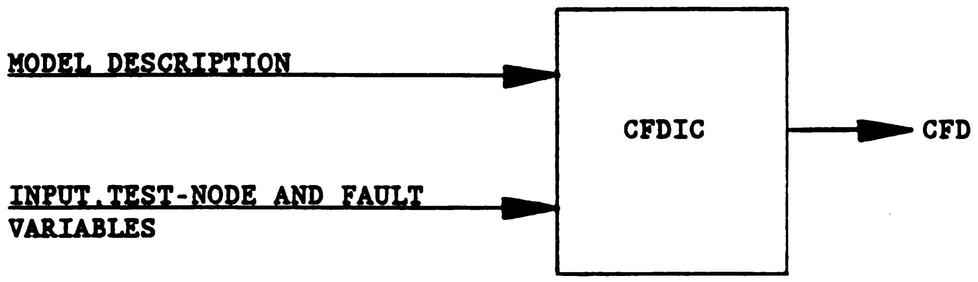


Figure 4.1: Overview of CFD

The program contains two main parts:

1. Conversion of the graph into a modified block-graph;
2. Construction of CFD of a tree (or a modified-block-graph).

The flow diagram of the program is shown in Figure 4.2. Before starting the operation of the main parts, the computer reads the system data and checks whether or not the graph is connected. Disconnected graphs are not accepted by this program; thus a message is provided about this to the user. Then inputs, test-node and fault variables are read and changed into input, test-node and fault junctions. If the original graph is already a tree, no extra step is needed and the tree-algorithm is activated. The tree algorithm is described in the preceding chapter. Please refer to Figure 3.2. Otherwise the graph is converted into the modified block graph. Then the tree algorithm is applied to this MBG and the fault dictionary is generated.

4.2 Inputs

4.2.1 Model description

Model structure is defined by its bond graph. The graph is introduced to the program by two lists: the node list and the incidence list. The node list defines the names of all nodes in the graph together with their type. The incidence list defines the bond names and their connection to the nodes.

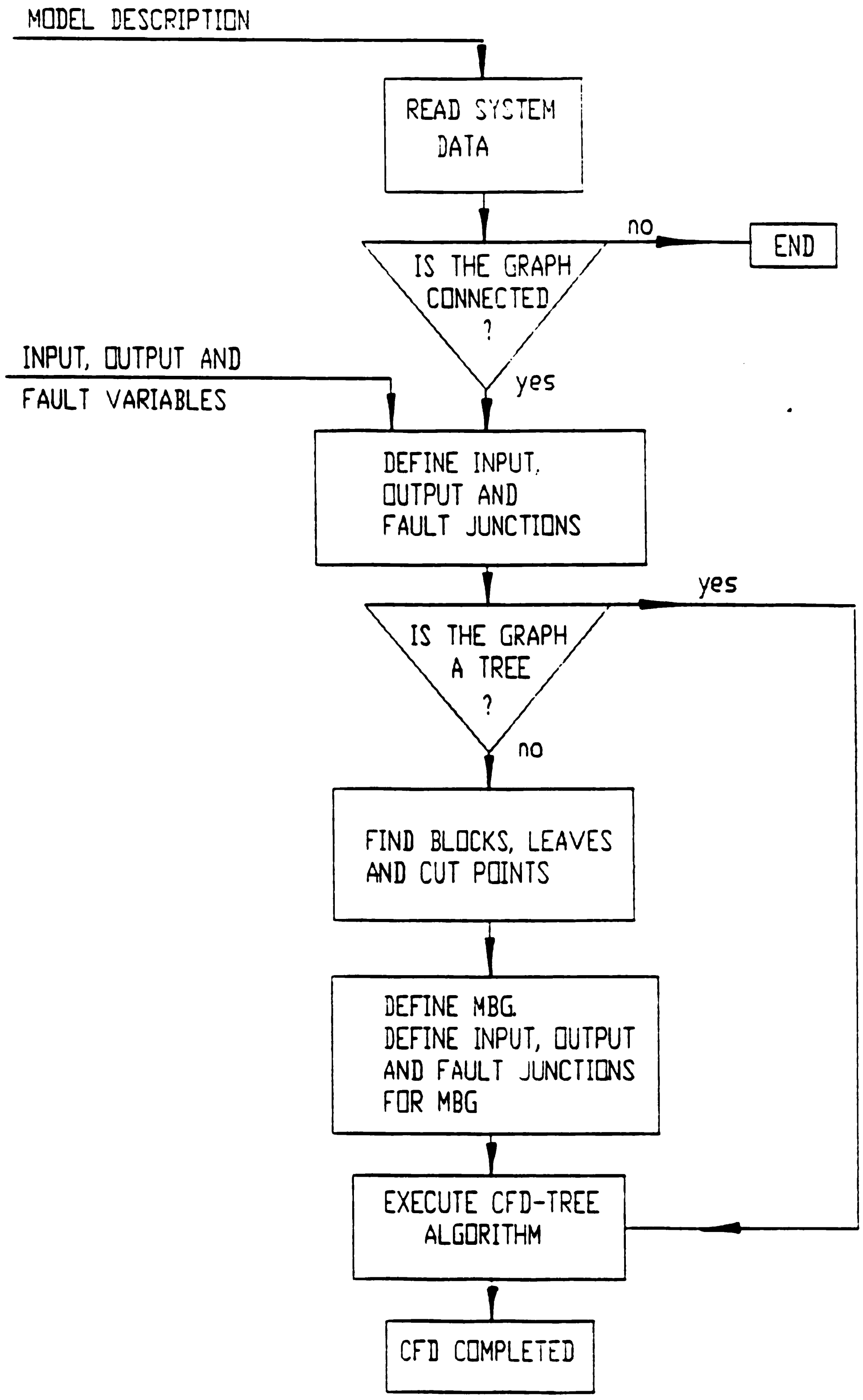

MODEL DESCRIPTION
READ SYSTEM DATA
IS THE GRAPH CONNECTED ?
no
END
yes
INPUT, OUTPUT AND FAULT VARIABLES
DEFINE INPUT, OUTPUT AND FAULT JUNCTIONS
yes
IS THE GRAPH A TREE ?
no
FIND BLOCKS, LEAVES AND CUT POINTS
DEFINE MBG. DEFINE INPUT, OUTPUT AND FAULT JUNCTIONS FOR MBG
EXECUTE CFD-TREE ALGORITHM
CFD COMPLETED

Figure 4.2: The CFDIC main flowchart

4.2.2 Input, test-node and fault variables

The special variables (i.e., input, test-node and fault variables) are defined in a data file called VAR.DAT. Input variables should be efforts or flows which are adjacent to a source node (SE, SF). A test-node variable can be any effort or flow that appears in the graph except an input variable. If there is no simple junction adjacent to the test-node variable, the program will automatically add the corresponding identity junction. Fault variables are defined by efforts or flows. For example, if a fault variable is e.i it means that it represents a catastrophic fault defined by e.i=0. Identity junctions are added to the model in the same way as for the test-node variables. The running OPTION will be discussed in Chapter 5.

4.3 Special subroutines

The CFD program contains thirty one subroutines. Most of them are an integral part of the construction of the fault dictionary. Some subroutines are of more interest or can be used independently is other graphical applications. These subroutines are discussed here briefly. Implementation notes, the calling tree structure and the listing of all the subroutines in alphabetic order, are provided in APPENDIX E.

4.3.1 Depth-First Search labeling (subroutine DFSLBL)

The *Depth-First Search (DFS)* is a well known systematic method of labeling vertices of a graph (Gibbons, 1985). Many graph algorithms are especially efficient when based on depth-first search labeling, for example, finding paths in a graph or finding blocks and strongly connected components. In our program DFSLBL is used for three different applications. These are (1) to check if the graph is connected, (2) to find paths in the tree-graph, and (3) to find the blocks of the graph.

The input to the subroutine includes the graph description (node list and incidence list) and the starting node. The subroutine provides the DFI(v), which is the *Depth-First Index* of every vertex v, and the *"father"* of v, FA(v), which defines the previsited vertex.

4.3.2 Finding paths in a tree graph (subroutine PATHFN)

Finding a path is one of the direct application of the Depth-First Search algorithm. The subroutine finds the path (defined by a list of nodes) between two given nodes in a tree graph. The algorithm is executed after the graph has been labeled by DFSLBL. A path is defined directly by following the *"father"* of each node. It is known to be a very efficient algorithm (Even, 1979).

4.3.3 Block finding (subroutine BLCKFND)

BLCKFND is executed after the graph has been labeled by DFSLBL. In this algorithm the graph is scanned and every block found is popped out from the subroutine. Blocks are defined as group of nodes. The algorithm uses the DFI and the FA of each node to find the blocks. BLCKFND uses two other subroutines, SSTACK and POPSTK, to execute the algorithm. SSTACK is used to store nodes that are in the same block. When the nodes in SSTACK define a block, the list is send to POPSTK. The procedure continues until all blocks are defined.

4.3.4 Generating the modified block graph (subroutine BLCKGR)

Information about blocks, cut points and leaves (nodes of degree one) is captured in this subroutine. It processes all this data and converts it to a Modified Block Graph, which is stored in the place of the original graph. This information is made available to all other subroutines (through a common block) for generating the fault dictionary.

4.3.5 Generating CFD for a tree graph (subroutine TREDIC)

Subroutine TREDIC derives the CFD for a tree bond graph. The tree is either the original graph or the modified block graph.

The subroutine contains the following steps:

(1) Finding paths from an input to all test nodes;

(2) Checking the existence of all faults, defined by fault list, on those paths and adding the corresponding entries (one or zero) to the CFD.

The algorithm is run for each input until the CFD is completed. Input to TREDIC includes the system description and lists of input, test-node, and fault junctions. The output is the corresponding CFD.

5. THE FAULT DICTIONARY AS A DESIGN TOOL

The fault dictionary is basically used for detecting failures by comparison of experimental data to the FD data. From an engineering point of view, the more important and interesting problem is in the design stage, which is known as "Design for Testability". The design for testability problem can be described in general as: the problem of finding, in a cost-effective way, the faulty component, module, or connections in a dynamic system (Williams, 1983). In practice, many design-for-testability problems dealt with reducing the number of test-nodes needed to detect given faults (Lin, 1982, 1985; Ozawa, 1988).

The special structure of the CFD as a three dimensional boolean array is very attractive for design-for-testability. Figure 5.1 shows the CFD abstractly, from different points of view. So far we are used to think about the CFD as a collection of fault pages, as shown in Figure 5.1a, but it can also be interpreted from a test-node point of view as a collection of test-node pages (Figure 5.1b) or input pages (Figure 5.1c). Each test-node page contains all the information about the specific test-node, and the same is true for an input page. Thus the information which is kept in the CFD can be generated in various ways for solving design problems.
Using this structure, the CFDIC software was extended to solve several design problems. In this chapter we present some typical applications and the software needed to solve them. A five degree-of-freedom ground vehicle model is used to illustrate the various design options. The design problems are imbedded in the CFDIC software and can be run under different options of the same program.

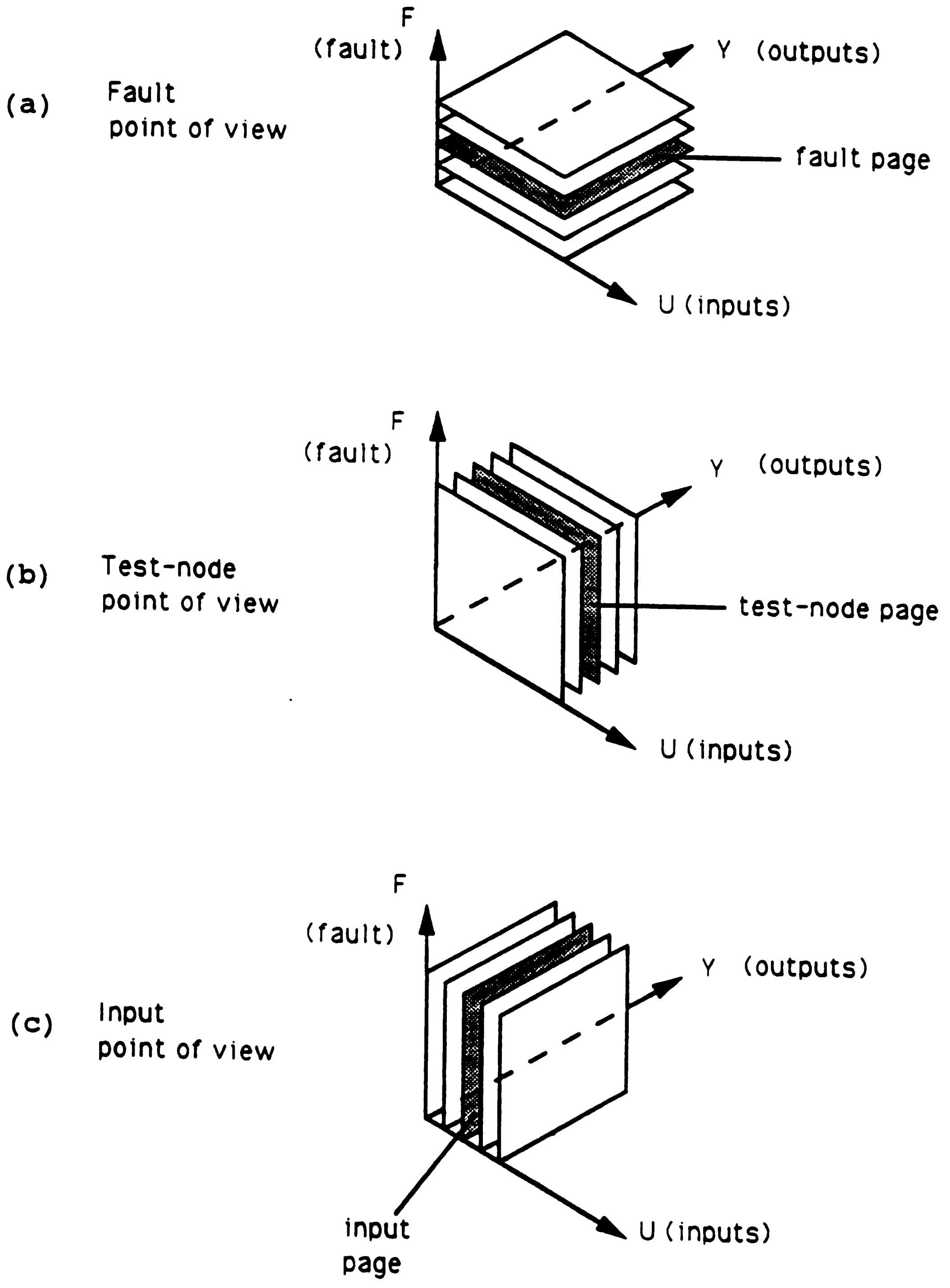


Figure 5.1: CFD representation from fault, test-node, and input point of view

5.1 Fault detection properties (CFDIC option 1)

5.1.1 Fault report

For given input, output and fault lists, the fault dictionary may provide additional information about the status of every fault in the system (i.e., whether the fault is uniquely detectable, undetectable and so on). CFDIC Option 1 provides this data in addition to the fault dictionary.

The message about each fault in the original fault-list is one of the three categories:

(1) The fault is uniquely detected.

(2) The fault is detected but creates an ambiguity set with other faults. In this case the list of all faults in that ambiguity set is provided.

(3) The fault is undetectable by the current inputs and test nodes. A change in either test nodes or inputs is needed in order to detect this fault.

Percentage of detection (number of detected faults out of total number of faults) and unique detection is also provided. Option 1 of CFDIC provides the fault report. Inputs are the model description and input, output and fault variables.

Example

Consider the five degree-of freedom vehicle model of Figure 5.2. In the figure *ml* represents the mass of the seat and driver, which is supported by spring *kl* and damper *cl* that are attached to the main body of the vehicle. The masses of the main body of the vehicle, the wheels, and the

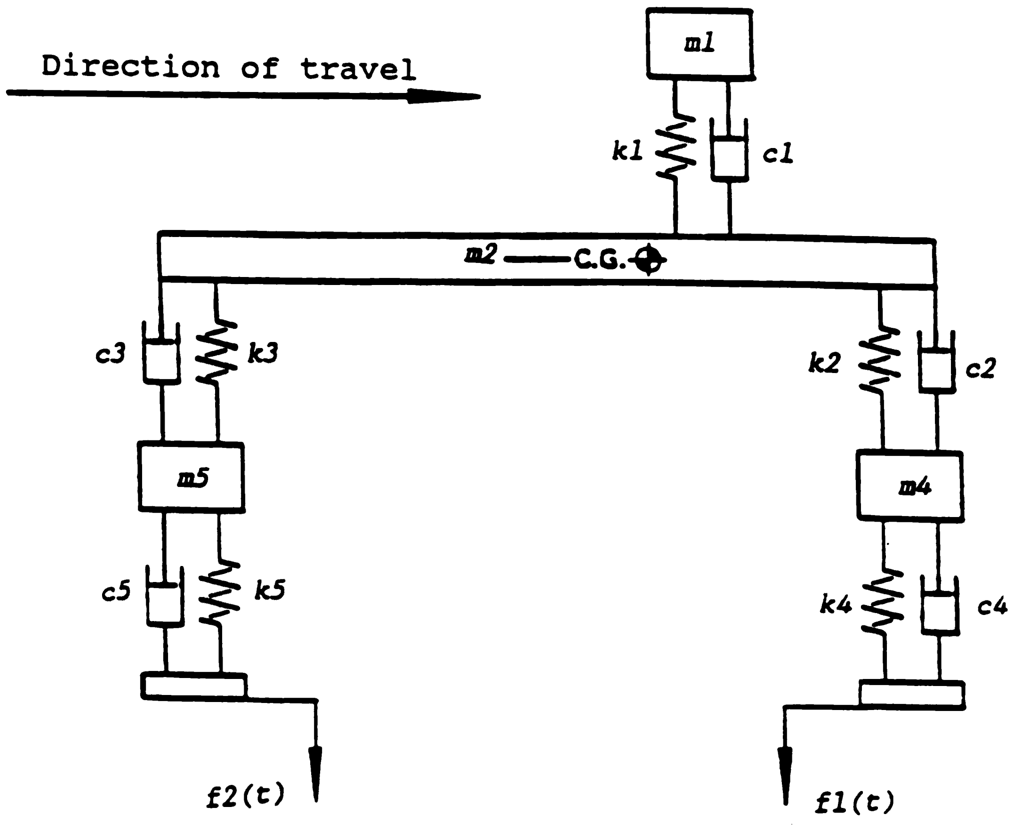


Figure 5.2: Five degree of freedom vehicle model

axles are *m2*, *m4*, and *m5*, respectively. The main body of the vehicle is supported by springs *k2* and *k3* and the dampers *c2* and *c3*, which are attached to the axles. The parameters *k4*, *k5*, *c4*, and *c5* represent the stiffness and damping coefficients of the tires. The functions *f1(t)* and *f2(t)* represent vertical velocities of the front and the rear tire contact patch, respectively, due to undulations in the road surface on which the vehicle is traveling.

The bond graph of the system is shown in Figure 5.3. Here, SF1 and SF2 represent the input velocities of the front and rear tire contact patch,

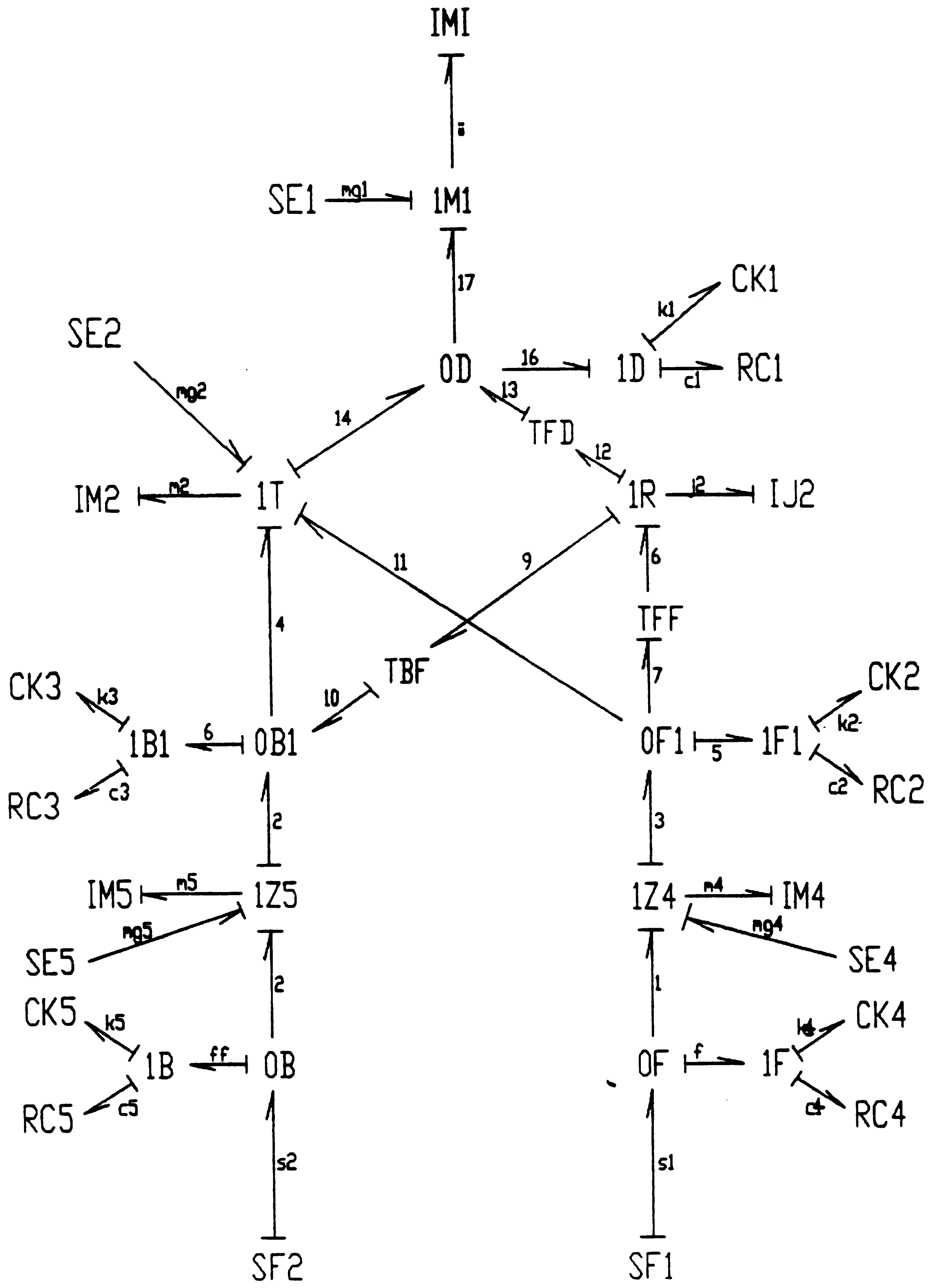


Figure 5.3: Bond graph model of five-degree-of-freedom vehicle suspension system

the I-elements represent masses and inertia (IM4 represents *m4*, etc.), C-elements define springs, and R-elements represent dampers. It is interesting to observe the modified block-graph of this system, shown in Figure 5.4. Note that the tree part of the graph remains unchanged while the loops were reduced to one node (block1). Model description, as an input file, is attached in APPENDIX F1.

Fault-list, test-nodes and inputs

Suppose a catastrophic fault list contains the faults defined in Table 5.1.

Table 5.1: Fault list

Fault	Fault variable	Description
F1	f.c2	Stuck suspension damper of front wheel
F2	e.3	suspension and body disconnected (front)
F3	e.k2	Broken suspension spring (front)
F4	e.2	wheel and suspension disconnected (rear)
F5	f.k5	Flat rear tire

In order to detect the faults, three accelerometers and one load-cell were located in the system. Test-node locations and measurements are described in Table 5.2.

Table 5.2: Test-node list

Test-node	Test-node variable	Location	Measurement
Y1	e.m4	Front wheel axle	Vertical acceleration
Y2	e.m2	Main body	Vertical acceleration
Y3	e.ii	Seat	Vertical acceleration
Y4	e.k2	Suspension	Spring force

Velocities of front and rear wheels were used as inputs of the system. Their description is shown in Table 5.3.

Table 5.3: Input list

Input	Input variable	Description
U1	f.s1	Vertical front wheel velocity
U2	f.s2	Vertical rear wheel velocity

Fault, test-nodes and input variables

Faults, test-nodes and input variables are defined as bond graph variables using the bond graph model shown in Figure 5.3. Table 5.1 shows the bond graph variables associated with each fault. Note that a fault is defined such that its variable is identically zero. Tables 5.2 and 5.3 show the test-node variables and the input variables respectively. Input, output and fault lists as an input file for the program are shown in APPENDIX F2.

Construction of CFD

Using the preceding algorithms, the CFD is constructed. The SSM for each fault are listed below for clarity.

SSM for F1: $\begin{matrix}1&1&1&0\\1&1&1&0\end{matrix}$

SSM for F2: $\begin{matrix}1&0&0&0\\0&1&1&0\end{matrix}$

SSM for F3: $\begin{matrix}1&1&1&0\\1&1&1&0\end{matrix}$

SSM for F4: $\begin{matrix}1&1&1&1\\0&0&0&0\end{matrix}$

SSM for F5: $\begin{matrix}1&1&1&1\\1&1&1&1\end{matrix}$

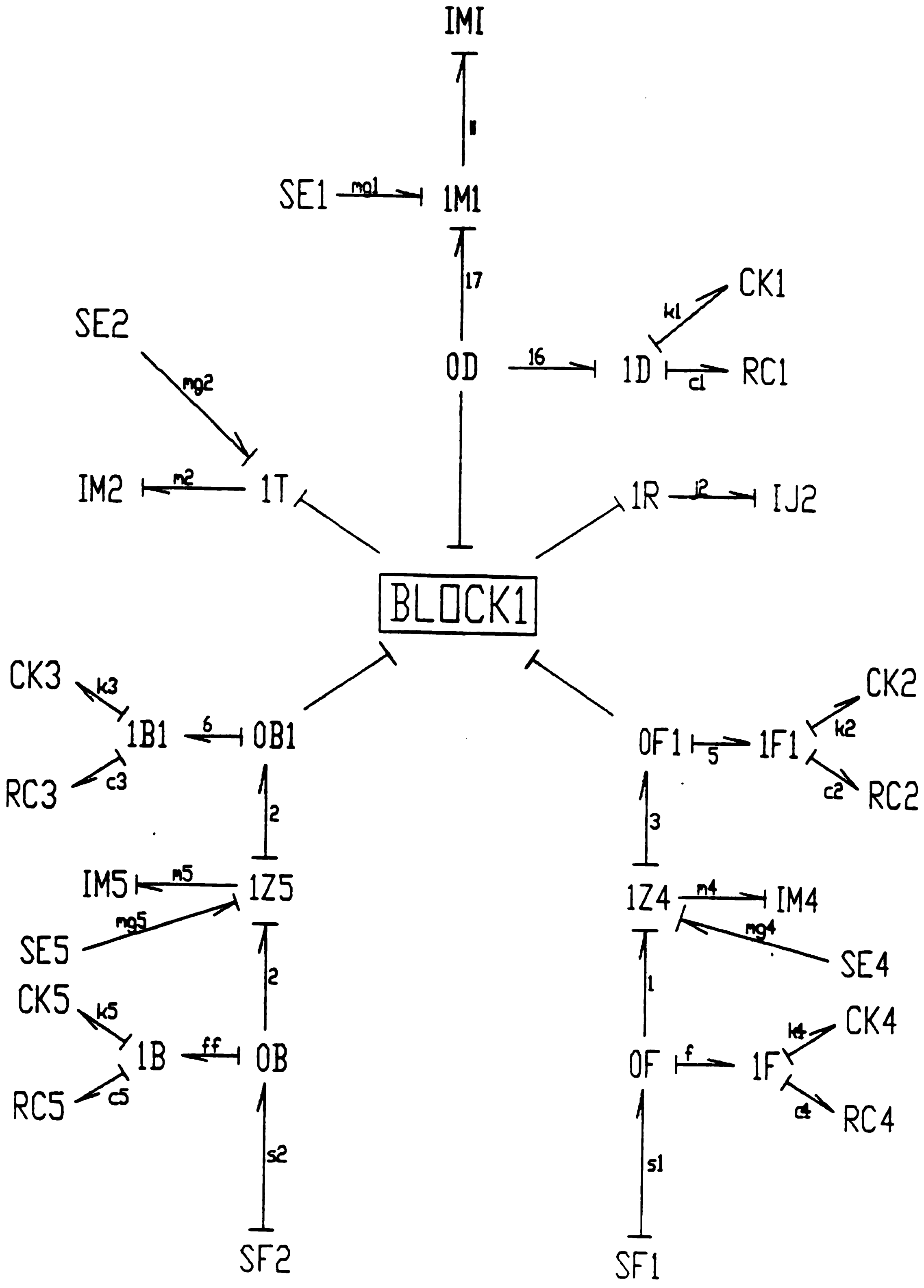


Figure 5.4: Modified-Block-graph of five-degree-of-freedom vehicle suspension system

The fault report for this (short) fault list, as provided by CFDIC option 1 is:

Table 5.4: Fault report

FAULT	DEFINED BY	DIAGNOSIS
F1	f.c2 =0	DETECTABLE, AMBIGUITY SET #1
F2	e.3 =0	DETECTABLE, UNIQUELY
F3	e.k2 =0	DETECTABLE, AMBIGUITY SET #1
F4	e.2 =0	DETECTABLE, UNIQUELY
F5	e.k5 =0	UNDETECTABLE

```
TOTAL DETECTION PERCENTAGE: 80.0%
UNIQUE DETECTION PERCENTAGE: 40.0%

AMBIGUITY SETS
**************

   AMBIGUITY SET #1 CONTAINS:
                    FAULT #1
                    FAULT #3
```

5.1.2 Initial fault list

This option is a subset of the previous one. Here, inputs and outputs are given, and the user wishes to find the detectable fault list. A list of all faults that are detected by the given inputs and outputs is provided by the software. This can work as an initial fault list, since other faults are undetected by those inputs/outputs.

This problem is addressed by finding all fault junctions on the paths from every input to every output. Option 1 of CFDIC provides a list of faults that are detected by known inputs and test-nodes. The input to this option includes the model description, and the lists of the known inputs and test-nodes.

The *detected faults* option may be helpful for the definition of the fault list, since it is the complete fault list detected by given inputs and test-nodes. This fault list can be used as an initial list from which the actual fault list is defined. The initial fault-list option is actually a default of option 1, i.e., whenever the fault list data is missing, the software provides the complete detected fault list.

Example

Let us assume that three accelerometers are already located in the suspension system (Figure 5.2) which measure the main body rotation, main body vertical acceleration and the seat vertical acceleration. Thus the output variables are e.j2, e.m2 and e.ii respectively. The input is the front wheel vertical velocity, f.sl. Which faults may be detected by these inputs-outputs? CFDIC option 1 (the default option) was run using this data and the following results were provided (direct copy of the printout).

Table 5.5: List of faults detected by the given inputs and test-nodes

FAULT	DEFINED BY	ASSOCIATED WITH S.J.	DIAGNOSIS
F1	e.j2 =0	*	DETECTABLE, A.S. #1
F2	f.j2 =0	1R	DETECTABLE, A.S. #1
F3	e.3 =0	0F1	DETECTABLE, A.S. #2
F4	f.3 =0	1Z4	DETECTABLE, A.S. #2
F5	e.1 =0	0F	DETECTABLE, A.S. #2
F6	e.m2 =0	*	DETECTABLE, A.S. #3
F7	f.m2 =0	1T	DETECTABLE, A.S. #3
F8	f.m2 =0	0D	DETECTABLE, A.S. #3

* THE SJ ASSOCIATED WITH THIS FAULT DO NOT APPEAR IN THE ORIGINAL GRAPH

ABBREVIATIONS:
S.J. - SIMPLE JUNCTIONS
A.S. - AMBIGUITY SET

AMBIGUITY SETS
****

AMBIGUITY SET #1 CONTAINS:
FAULT #1
FAULT #2

AMBIGUITY SET #2 CONTAINS:
FAULT #3
FAULT #4
FAULT #5

AMBIGUITY SET #3 CONTAINS:
FAULT #6
FAULT #7

5.2 Test-node detectability characteristics (CFDIC Option 2)

The design of number and location of test-nodes, or choosing test-nodes in an efficient way, is the most important and most addressed problem in design-for-testability (Lin, 1985). Usually, an initial list of possible test-node locations with their cost is provided. We will address this problem in a simplified way, i.e., every test-node has the same installation complexity (defined as "cost"), every fault has the same importance (defined by "weight") and the inputs to the system are fixed. More complex problems can be consider from this approach modified suitably.

In the general case, the determination of a *minimum* set of test nodes to achieve the highest percentage (not always 100%) of fault detection is a very time consuming process for large systems. Lin (1985) suggested a more practical approach. For practical applications there is no need to obtain the *theoretical minimum* number of test nodes. Any *near-minimum* solution will serve our purpose if the solution is simple. In other words, a heuristic method might be more useful for solving practical problems. Our approach combines the CFD properties with the practical approach mentioned above.

The following algorithm should address the questions:

(1) Which of the test-nodes (defined in the initial list) is unnecessary?

(2) Which test-nodes are equivalent (i.e., provide the same information)?

(3) Which test-node detects the maximum number of faults?

(4) If the number of test-nodes is limited, which test-nodes should we choose in order to detect the maximum number of faults?

The first two questions can be directly answered by observing the CFD. Let us define a *Test-Node Matrix, TNM*, for test-node #jo as:

$$TNM(jo) = D(i,jo,k) \qquad (5.1)$$

where i=1, ... ,l (l is number of inputs);

k=1, ... ,n (n is number of faults);

and D is the CFD array.

Fig. 5.1 shows this matrix abstractly. A test-node is unnecessary if all its entries are 1, which means that there is no prospective fault on any path from any input to this test-node. If two test-nodes have the same TNM, they are equivalent. Thus only one of these should be selected. Question 3 is solved by processing the data of the TNM for each test-node. Question 4 is addressed by the following heuristic procedure:

Step 1: Select the test-node that has the largest number of detected faults.
Step 2: Select next the test-node whose intersection with previously selected nodes will result in the largest number of detected faults.
Step 3: Repeat step 2 till you reach m test-nodes.

If we describe the number of detected faults at each step as a function of the number of picked test-nodes, we get a useful design-for-testability tool to aid the user. All the described information is provided while running Option 3 of the CFD software. The input consists of the system description, list of input variables, fault variables and initial (maximal) test-node list.

Example

Suppose only one input, front wheel velocity, was used as an input. The fault list consists of the following variables:

```
f.5
e.6
e.11
e.16
f.4
e.k2
f.c2
f.3
f.15
f.12
```

The initial test-node list contains the following variables:

```
e.j2
e.m2
e.ii
f.k2
e.m5
e.k2
f.k3
e.c2
e.c4
```

All this input data is provided by the user; see details in APPENDIX F3.

Results:

Below are the results, as provided by the software.

```
TEST-NODE REPORT
****************

   LIST OF UNNECESSARY TEST-NODES
   ******************************
                 TEST-NODE #10 (DEFINED BY e.c4  )
```

```
TEST-NODE EQUIVALENCE SETS
**************************

   TEST-NODE EQUIVALENCE SET #1 CONTAINS:
              TEST-NODE #4 (DEFINED BY f.k2  )
              TEST-NODE #9 (DEFINED BY e.c2  )


DETECTABILITY OF TEST-NODES
***************************

-TOTAL NUMBER OF FAULTS: 10

-NUMBER OF EFFECTIVE TEST-NODES: 8
 * UNNECESSARY TEST-NODES WERE DELETED FROM THE LIST
 * AN EQUIVALENCE SET IS MARKED BY ITS FIRST T.N. VARIABLE
```

Table 5.6: Detectability of test-nodes

TEST-NODE NUMBER	TEST-NODE VARIABLE	NUMBER OF DETECTED FAULTS	NUMBER OF UNIQUELY DETECTED FAULTS
1	e.j2	3	0
2	e.m2	3	0
3	e.ii	3	0
4	f.k2	4	0
5	e.m5	4	0
6	e.k2	5	0
7	f.k3	3	0
8	e.c2	4	0

REDUCTION OF TEST-NODES
****

Table 5.7: Cumulative detectability of test-nodes

NUMBER OF T.N.	INCLUDING T.N. VARIABLES	TOTAL NUMBER OF D.F.	TOTAL NUMBER OF U.T.F.
1	e.k2	5	0
2	e.m5	7	0
3	e.ii	8	1
4	e.j2	9	2
5	e.m2	10	3
6	f.k2	10	4
7	f.k3	10	6
8	e.c2	10	6

ABBREVIATIONS:

T.N.	- TEST-NODES
D.F.	- DETECTED FAULTS
U.D.F.	- UNIQUELY DETECTED FAULTS

Discussion:

Observing the results the designer can conclude the following:

(1) e.c4 is useless as output for detecting the faults defined by the fault list.

(2) Either f.k2 or e.c2 should be chosen as outputs. There is no need to choose both.

(3) e.k2 is the "best" test-node variable as far as maximum detected faults are concerned (refer to Table 5.6).

Table 5.7, whose information is abstracted graphically in Fig. 5.5, provides a design tool for a limited number of test nodes. For example, if only four test-nodes are allowed, one can see from the graph that nine faults will be detected, two of them uniquely. From Table 5.7 we can find which test nodes should be chosen. For this example the first four variables, i.e., e.k2, e.m5, e.ii and e.j2, are the optimal test-node variables.

In this option, the algorithm has been developed to detect maximum faults, ignoring uniqueness considerations. Other algorithms may be developed taking into account other design considerations, as specified by the designer of the system.

5.3 Summary

The chapter described two, out of many, families of design problems, that may be addressed by the CFD approach. These are the fault detection properties and the test-node detectability characteristics. The procedures needed to solve those problems are imbedded in CFDIC software. Figure 5.6 summarizes these design options of CFDIC software. Further design problems are left for future work (refer to Chapter 6).

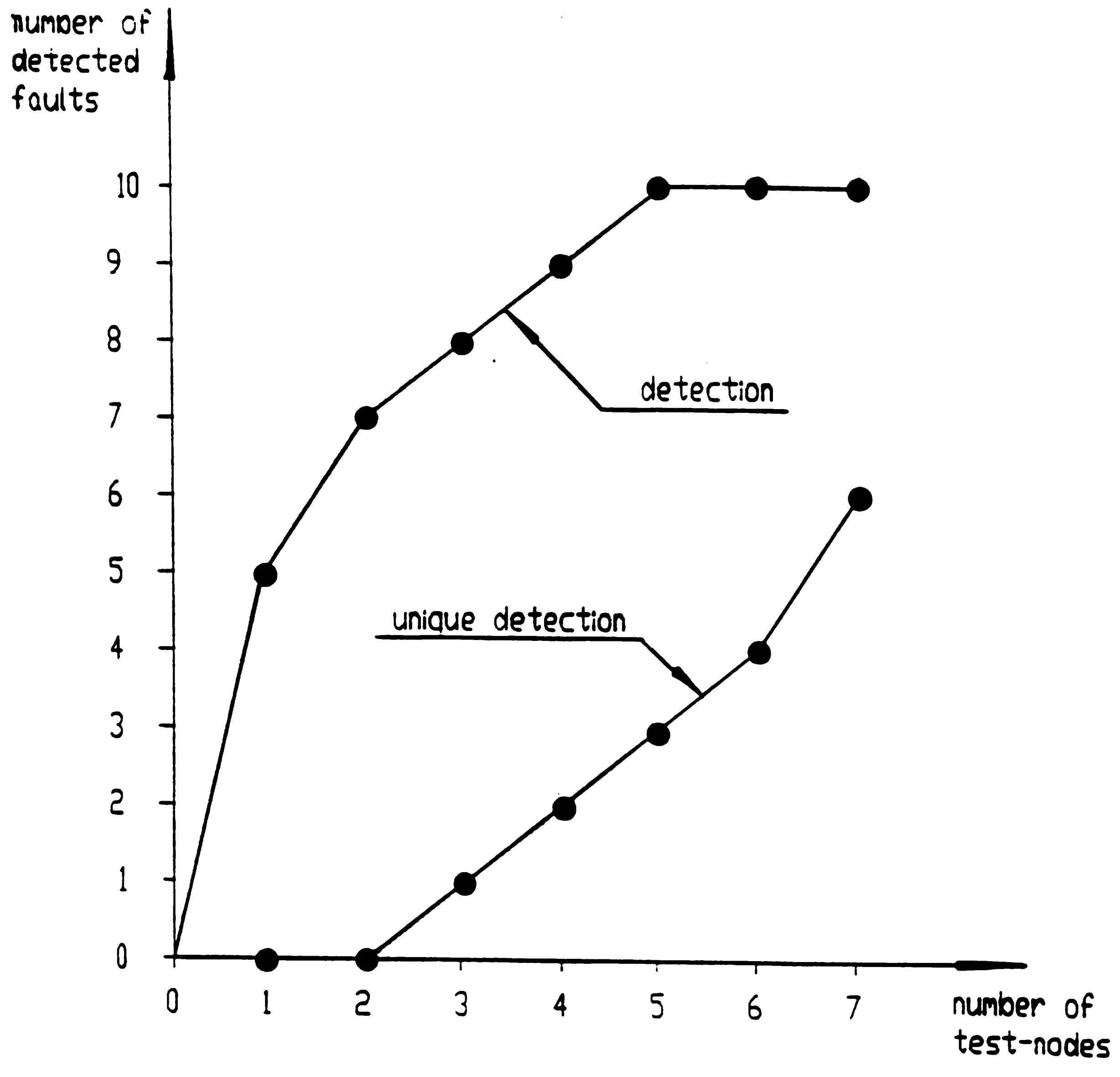
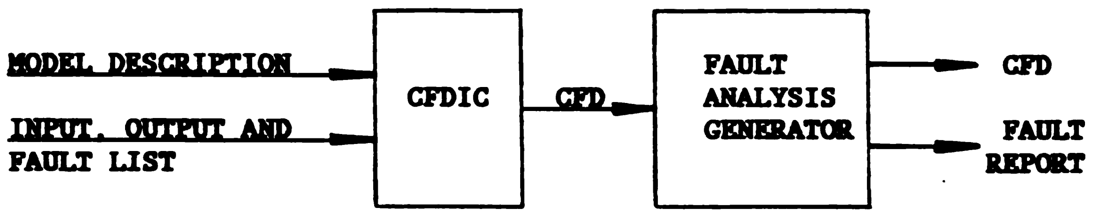
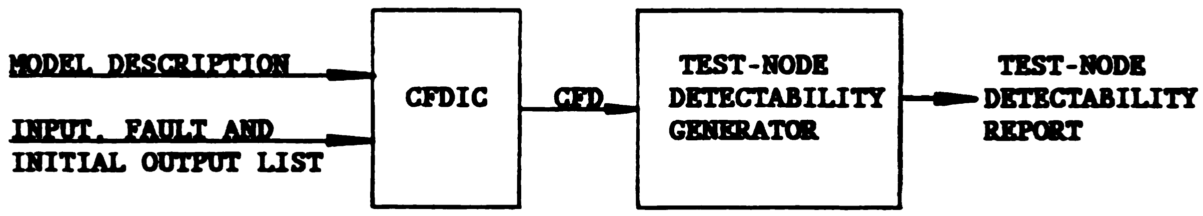


Figure 5.5: Number of detected faults as a function of the number of test-nodes



Option 1: Fault detection properties of system design


MODEL DESCRIPTION
INPUT, FAULT AND INITIAL OUTPUT LIST
CFDIC
CFD
TEST-NODE DETECTABILITY GENERATOR
TEST-NODE DETECTABILITY REPORT

Option 2: Test-node detectability characteristics

Fig. 5.6: Options of CFDIC software

6. SUMMARY AND FUTURE WORK

6.1 Summary

A new approach to constructing the catastrophic fault dictionary for engineering systems was introduced and developed. The CFD is derived directly from the topological structure of the system and the resulting fault dictionary is a three-dimensional boolean array.

The procedure of deriving the CFD and addressing design problems contains three main steps. Please refer to Figure 6.1. Every step has a well defined input, output and associated mathematical tool. In step 1 bond graph methods convert the catastrophic fault diagnosis problem into a graphical one. In step 2 Graph algorithms are used to generate the CFD from the graphical presentation. The CFD, as an output of step 2 and an input of step 3, is then processed by boolean algebra methods to address testability-design engineering problems.

Comparison of CFD to a conventional fault dictionary is summarized in Table 6.1. The major advantage of the proposed approach is that it is derived directly from the model structure. The binary structure of the CFD is attractive due to the limited size of storage needed and the simple look up (when the CFD is compared to experimental data). It also allows one to detect multifault without increasing the dictionary size due to its superposition property. The CFD has been demonstrated to be a useful tool for testability design.

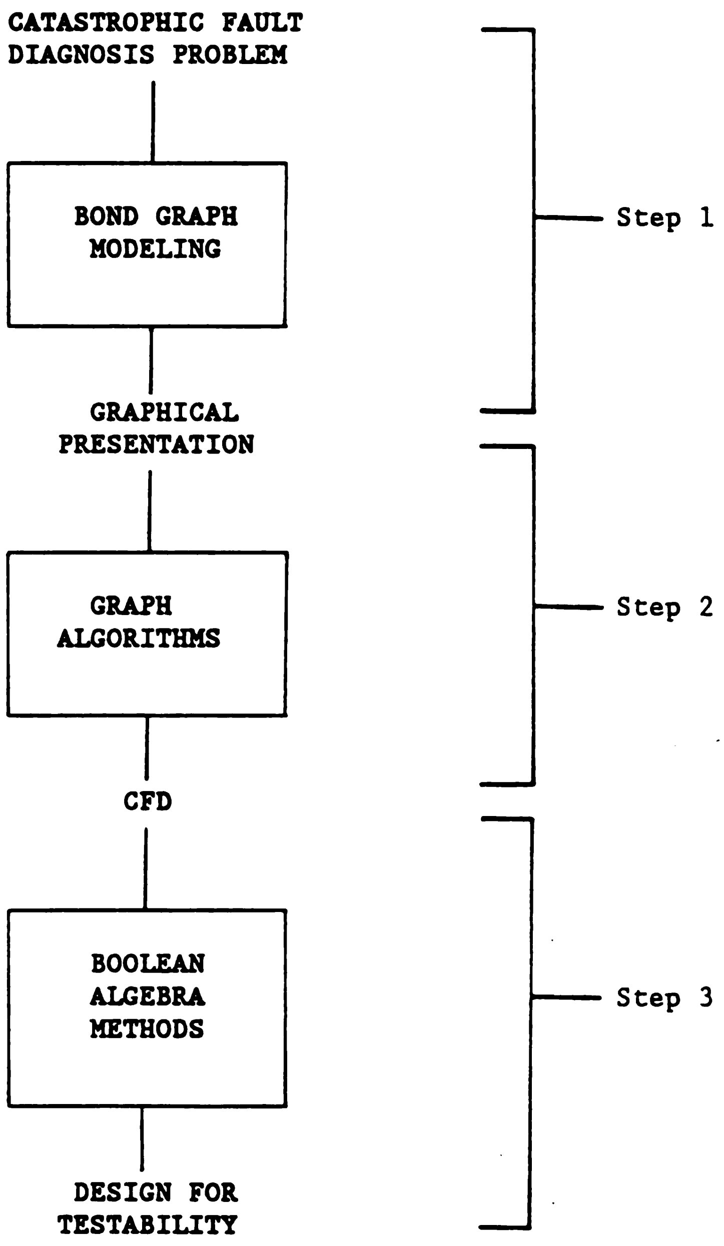


Figure 6.1: CFD procedure

Table 6.1: Comparison between conventional fault dictionary and CFD

Feature	Conventional FD	CFD
FD structure	three dimensional array	three dimensional array
FD entries	outputs, real numbers	structural sensitivities of outputs with respect to inputs, zeros or ones
Type of detected faults	catastrophic and/or soft, usually single	catastrophic, single and multiple
Ambiguity sets	defined by the designer	defined automatically by the computer
Advantages	1. Straight forward 2. Applicable for analog and digital systems 3. Appropriate for hard and soft failures	1. Derived directly from the bond graph model 2. Most general, handles various types of systems 3. easy look up - due to boolean structure 4. based on system structure, parameter change does not change the dictionary 5. Handles nonlinearities 6. efficient in memory size 7. handles multifaults easily
Disadvantages	1. limited by memory size 2. calculations needed to detect faults 3. limited to piecewise linear functions (current version)	1. handles catastrophic faults only 2. efficiency depend on model structure

The disadvantages of this method also lie in the structural procedure of constructing the CFD. At this stage of development faults are limited to catastrophic ones. The practical problem of identifying parameters that are drifted from nominal is not addressed.

6.2 Future work

Although the task of constructing a catastrophic fault dictionary has been completed, many additional areas of worthy research were found while performing this work. Described below are several topics considered as important for future studies.

6.2.1 Extension of the structural approach

6.2.1.1 Non catastrophic faults

The current approach is limited to catastrophic faults only. Although those are critical in engineering systems, non-catastrophic failures are important too. It is suggested to extend the family of detectable faults by modeling non-catastrophic failures as catastrophic. Lin (1985) modeled a fixed change in a parameter value (resistance in an analog circuit) by a "fault switch" which is similar to our fault source. Since soft failures are actually changes in system parameters, they can be theoretically modeled as catastrophic failures. Figure 6.2 (from Lin, 1985) shows open circuit, shorted circuit and fixed change in a parameter value, modeled by switches. In practice, modeling of every soft failure this way is not reasonable. Nevertheless, some non-catastrophic faults may fall into this category. With some complication of the bond graph, which can be handled by the software, various types of non-catastrophic failures may be detected by the structural approach.

Further investigation of those faults, their appearance in real systems and their diagnosis techniques is recommended.

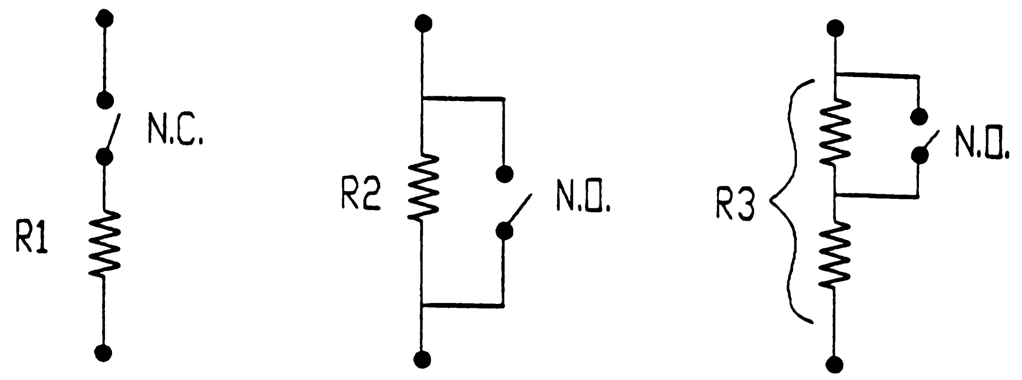


Figure 6.2: Catastrophic failures represented by switches

6.2.1.2 Multifaults

The superposition property, as described in section 3.7.3, has not seen use in this study. A further development of the software is needed to include this feature in the CFD. This property is very attractive to fault detection since there is no need to increase the memory size for including this feature.

6.2.2 Development of conventional fault dictionary using bond graph methods

As mentioned in the literature review, a conventional fault dictionary is constructed by running the nominal and faulty systems, obtaining the test nodes' output and comparing them. A program that accepts bond graph models and simulates their dynamic response is a logical basis for development an automated FD set up procedure. The ENPORT software is a suitable computational tool (Rosenberg 1987), in the process of designing an automated procedure for setting a FD.

A procedural diagram for the construction of a fault dictionary is shown in Figure 6.3. Once the bond graph model is built, one can define the list of failures (component failures and connection failures) which is translated into a list of faulty junctions. After inputs and output locations are defined, the computer runs the nominal and the set up of faulted simulations in order to construct the FD.

So far most of the fault dictionaries developed (Elcherif, 1983; Lin, 1985; Schreiber, 1979) dealt with nonlinear analog circuits, where the nonlinearities were assumed to be piecewise linear functions and the inputs were constant. Hochwald (1979) suggested to expand this dc approach to be able to cover also dynamic failures. This can be done by including small signal periodic stimuli. Under the assumption that both nominal and faulty models are stable at the point of interest, the proposed approach is to use two types of inputs : a constant input (CI) which will respond as a constant steady state output, and a small signal sinusoidal input, which will assure a periodic response (due to linearization) in order to

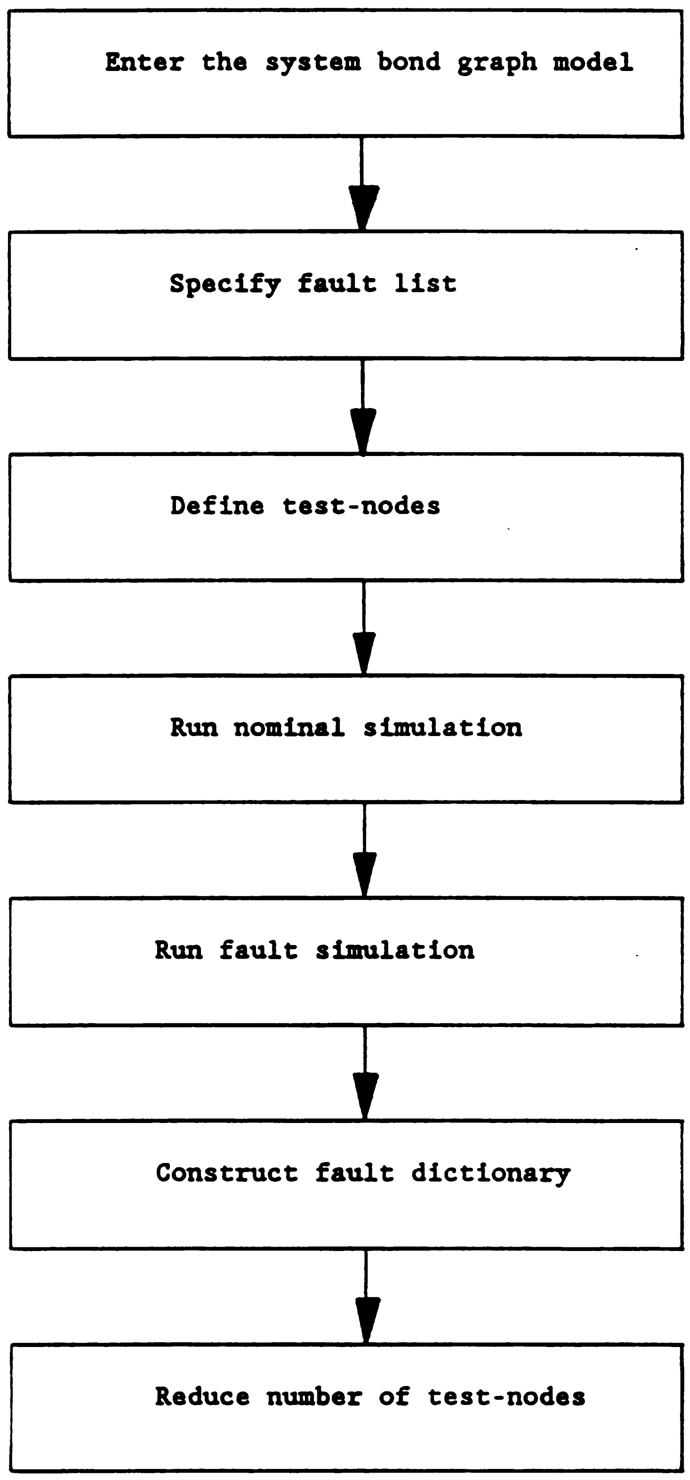

Enter the system bond graph model
Specify fault list
Define test-nodes
Run nominal simulation
Run fault simulation
Construct fault dictionary
Reduce number of test-nodes

Figure 6.3: Automated structural fault dictionary set up

detect dynamic failures. No special assumptions should be made about the nonlinearities of the system.

Since modeling of catastrophic failures can be done in bond graph language and bond graph software can handle nonlinearities in a general system, a fault dictionary based on this technique will be suitable for a variety of dynamic systems. It should be noted that at present the CI fault dictionary is the only one that is used in practice, while others have not progressed beyond the feasibility study stage (Ozawa, 1988).

A detailed diagram for automated construction of a fault dictionary is given in Figure 6.4. It contains five major subroutines. Some of them are straightforward; others need some development.

6.2.3 Further investigation of CFD as a design tool

The CFD has been found to be very attractive for testability-design. So far we have dealt with simplified design problems, in which all faults were equal in their importance and every test-node has the same price (i.e., complexity in installation). In practice this is not the case. The current approach may be extended to solve engineering problems by processing the data of the CFD array in a more complicated way. A complete design problem defined by the user may be solved by the software. By that we mean, for example, that the user defines the constraint of this design, such as: number of faults to be detected, importance of specific faults and price or weight for each test-node. The software should address this problem and provide the best design under the constraints. The binary

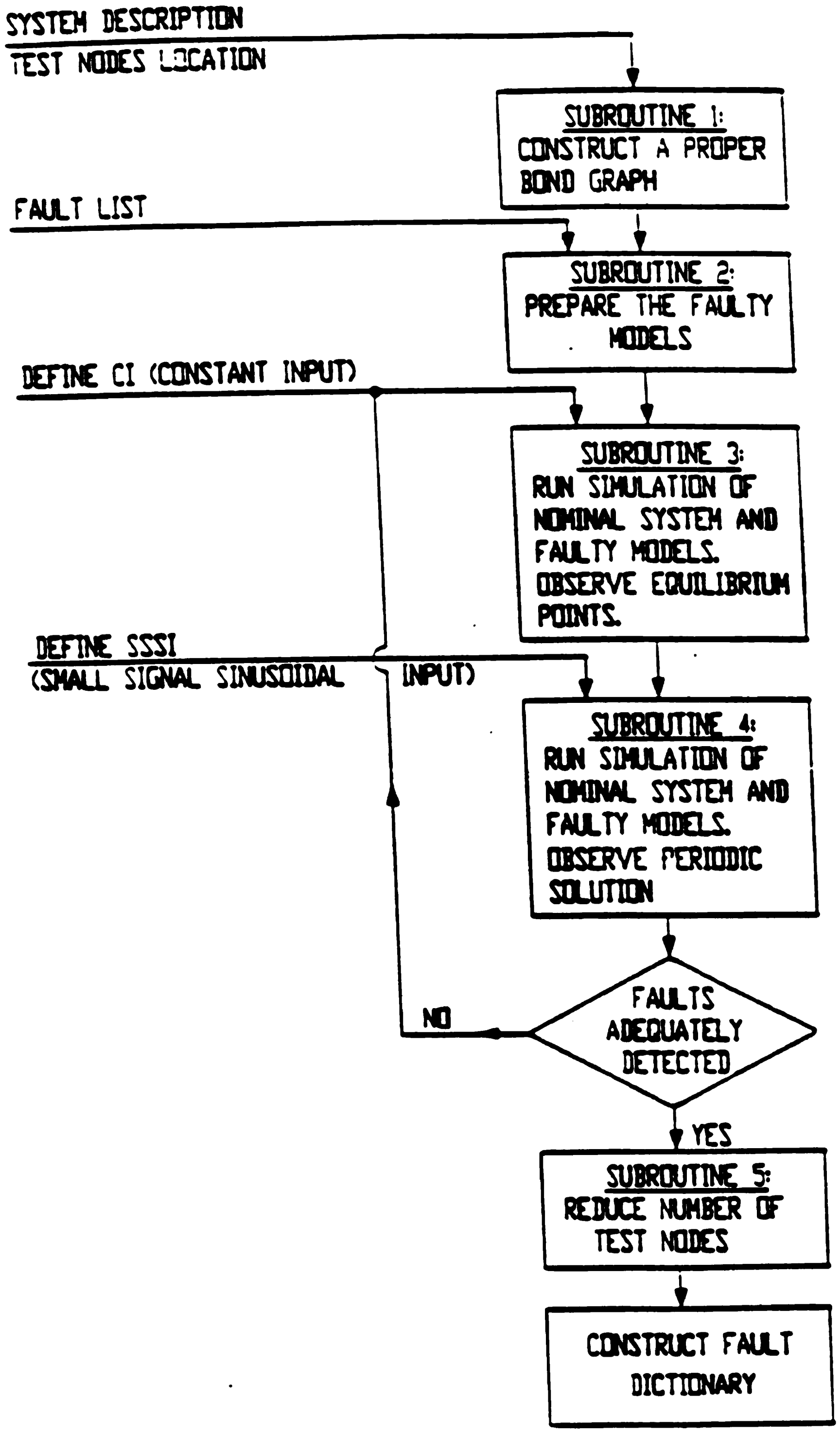

SYSTEM DESCRIPTION
TEST NODES LOCATION
SUBROUTINE 1: CONSTRUCT A PROPER BOND GRAPH
FAULT LIST
SUBROUTINE 2: PREPARE THE FAULTY MODELS
DEFINE CI (CONSTANT INPUT)
SUBROUTINE 3: RUN SIMULATION OF NOMINAL SYSTEM AND FAULTY MODELS. OBSERVE EQUILIBRIUM POINTS.
DEFINE SSSI (SMALL SIGNAL SINUSOIDAL INPUT)
SUBROUTINE 4: RUN SIMULATION OF NOMINAL SYSTEM AND FAULTY MODELS. OBSERVE PERIODIC SOLUTION
FAULTS ADEQUATELY DETECTED
NO
YES
SUBROUTINE 5: REDUCE NUMBER OF TEST NODES
CONSTRUCT FAULT DICTIONARY

Figure 6.4: Detailed flow diagram of a conventional fault dictionary

data base of the CFD array is attractive for solving such problems due to the easy processing of this type of data.

6.2.4 Development of SAT approach using bond graph techniques

As an alternative approach to the fault dictionary approach, which was widely discussed in this work, a number of researchers proposed the *Simulation-After-Test (SAT)* algorithms (Ozawa, 1988; Wu,1984; Salama, 1982). This method, which is mostly used in analog circuits, are useful when hard or soft faults need to be located up to the level a specific replaceable subsystem. In this approach, as mentioned in the literature review, the system under test is decomposed into subnetworks using nodes at which outputs have been measured.

It is recommended to extend the SAT approach to various types of systems by using bond graph modeling. As shown in the fault-dictionary approach, output are usually associated with simple junctions. Thus the constitutive equations for those junctions can be used to check the consistency of the system. An efficient algorithm may be developed in order to isolate faulty replaceable units. The bond graph model can also be used for design purposes, i.e., based on the topology of the system, optimal test-nodes could be chosen to detect specific faulty units.

Example: The nominal circuit and its bond graph model are shown in Figure 6.5a and Figure 6.5b respectively. Nodes of decomposition are A, B, and E. Then each subcircuit consists of one element (Figure 6.5c). The faulty circuit is shown in Figure 6.5d. Resistor r_3 is faulty and is now 3Ω

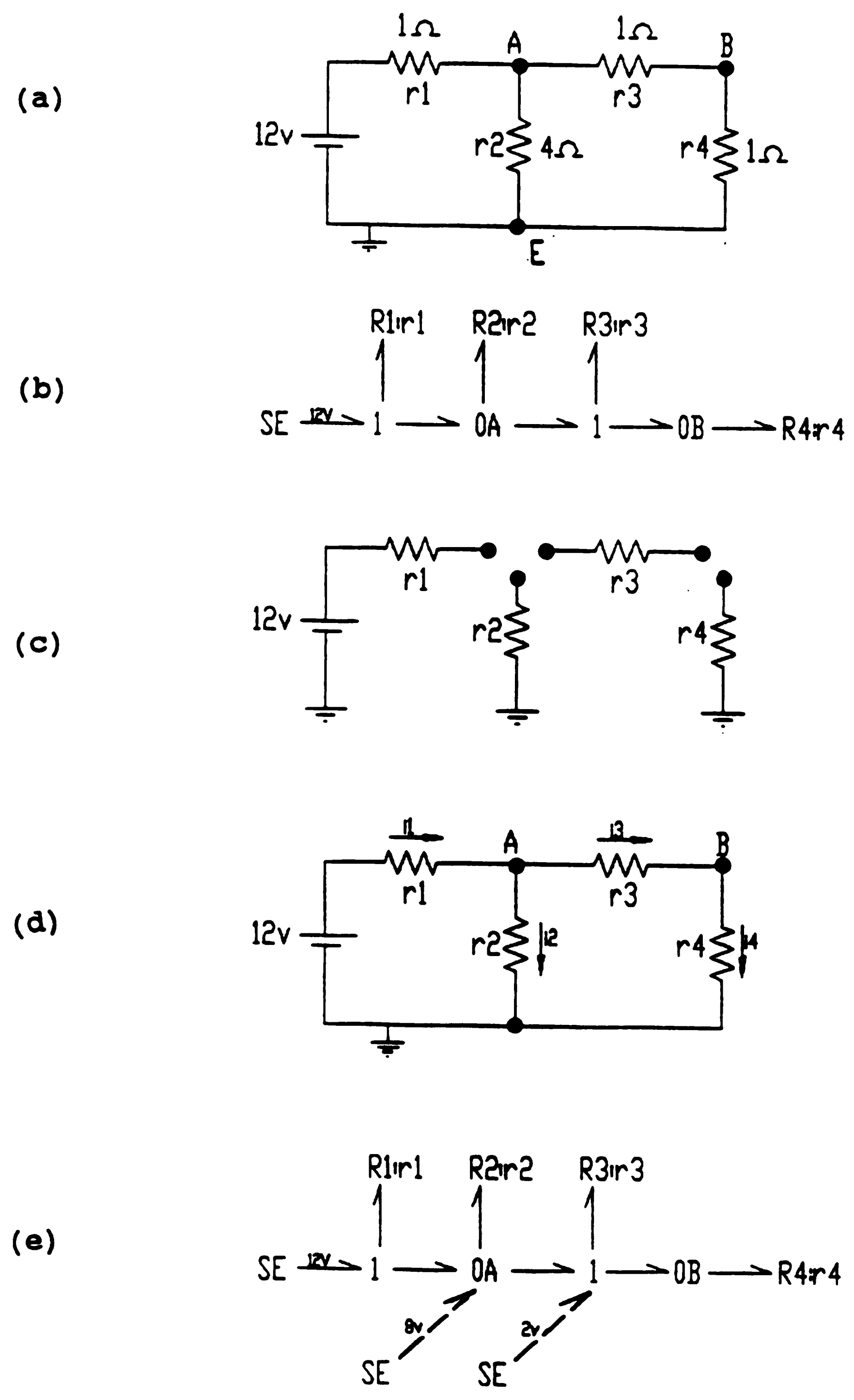


Figure 6.5: Circuit fault detection by SAT approach

instead of its nominal value of 1Ω. The measured node voltages will be v_a=8V, v_b=2V. Using these voltages and nominal value elements, we get i_1=4A, i_2=2A, i_3=6A, and i_4=2A.

The experimental data is then constrained on the *nominal* system by addition of sources of effort at points A and B (Figure 6.5e). The currents (flow variables) of these "artificial sources" are then observed. If the system is fault free, all those flows would be zero, since KCL is satisfied at each 0-junction. However, if a subsystem is faulty, KCL is not satisfied on its connections and a nonzero flow will be observed in the corresponding "artificial source". A nonzero current is associated with junctions 0A and 0B, thus the subsystem which lies between those junctions is faulty. For this case it is the resistor r_3.

6.2.5 Failure detection of shop replaceable units

The development of sophisticated systems has increased both the importance of Shop-Replaceable-Units (SRU) and the demand of more efficient methods for detecting failures associated with those units. The development of macro-models (Vlach, 1983; Rosenberg, 1986) enables us to improve modeling of such systems. Two types of failures can occur in a SRU: a macro failure or a connection failure. The former is defined by any failure that occurs inside the SRU while the latter describes a failure that occurs in the connection between SRUs or between SRUs and other components. Faults inside an SRU do not have to be detected uniquely, which can result in the reduction of the number of test nodes.

In our opinion, failures inside a SRU can be handled better by the SAT approach while connection failures should be detected better by the fault dictionary (SBT) approach. Both approaches (or a combination of them) should be investigated in order to find efficient methods to detect SRU failures.

6.2.6 Digital systems fault diagnosis

Although bond graphs were originally developed for physical dynamic systems (i.e., systems in which there is a power transmission within the system), the current approach of a boolean array as a fault dictionary is similar in principle to fault diagnosis approaches used in digital systems.

As a result of the development of LSI and VLSI, many researches are seeking for new approaches for detecting (in an efficient way) manufacturing faults. Common type of faults in digital systems are fixed faults . These faults remain in the system from the time they occur until they are replaced (Vishnubhotla, 1980). The most common ones occur when one or more wires are stuck at either 0 or 1 value.

A survey done on this subject (Williams and Parker, 1983) describes different diagnosis techniques and their applications for design-for - testability. Among the approaches mentioned there, as well as in other papers, two are close to our approach. These are the *digital fault dictionary* and the *cause-effect analysis*. The former approach, mentioned in early papers (Kautz, 1968; Brown, 1975) is similar to the conventional fault dictionary used for analog circuits. In more recent papers an attempt is made to handle large scale systems by improving the simulation process (Markas, 1990) or by generating the dictionary automatically from the structural description of the system (Sugasaki, 1989), which has the same goal as our research. The *cause-effect analysis*, proposed by Breuer in 1976 and followed by Abramovici (1980), does not require a fault dictionary and it is not based on comparing the obtained response with the expected response. Effect-cause analysis directly processes the actual response of the system-under-test (SUT) to the applied test (the effect) to determine the possible fault situations (the causes) which can generate that response.

Several similarities can be found between fault diagnosis in digital systems and the current approach.

(1) Both approaches focus on catastrophic faults where in digital systems those faults are defined as stuck-at faults.

(2) Fault dictionary is defined by a boolean array.

(3) Attempt is made to generate the dictionary on system structure, mainly by using the graphical representation of the system, and observing input-output paths.

(4) Detectable, undetectable faults and ambiguity sets are defined in the same way.

The similarities between the fault diagnosis in digital system and the CFD approach suggest further investigation in combining the two methods.

7. BIBLIOGRAPHY

Abramovici, M., 1980, *Fault diagnosis in logic circuits based on effect-cause analysis*, Ph.D Thesis, University of Southern California.

Bandler, J. W. and Salama, A. E., 1985, *Fault diagnosis of analog circuits*, IEEE Proc., Vol. 73, No. 8, pp. 1279-1325.

Basseville, M., 1987, *Detecting changes in signals and systems - a survey*, Proceeding of the and IFAC Workshop on Adaptive Systems in Control and Signal Processing,Lund, Sweden.

Breuer, M. A. and Chang, S. and Su, S. Y. H., 1976, *Identification of multiple stuck-type faults in combinatorical networks*, IEEE Trans. on Comp., Vol. C-25, No. 1, pp. 44-54.

Brown, F. M., 1975, *Test vector generation for internode and input diode short faults*, 2nd USA-Japan Computer Conference Proceedings, August 26-28.

Brown, F. T., 1972, *Direct application of the loop rule to bond graphs*, ASME Journal of Dynamic Systems, Measurement, and Control, Vol. 94, No. 3, pp 253-261.

Carre, B., 1979, *Graphs and networks*, Clarndon Press, Oxford.

Chartrand, G. and Lesniak, L., 1986, *Graphs and diagraphs*, Wadsworth and Brooks, Monterey, Calif.

Chen, C. T., 1984, *Linear system theory and design*, Holt, Rinehart and Winston, New York.

Christofides, N., 1975, *Graph theory - an algorithm approach*, Academic Press, Inc. New York.

Cikanek, H. A., 1986, *Space shuttle main engine failure detection*, IEEE Cont. Syst. Mag., June, pp. 13-18.

de Benito, C. D., 1988, *On-board, real-time failure detection and diagnosis of automotive systems*, Proceeding of the 1988 American Control Conference.

de Benito, C. D., 1990, *Control of an active suspension system subject to random component failures*, Journal of Dynamic Systems, Measurement, and Control, Vol. 112, pp 95-99.

Duhamel, P and Rault, J. C., 1979, *Automatic test generation techniques for analog circuits and systems: A review*, IEEE Trans. Circuits Syst., Vol. CAS-26, pp. 411-440.

Elcherif, Y. S. and Lin, P. M., 1983, *Fault diagnosis of nonlinear analog circuits, Volume V, HAFDIC: A program for generating a hard fault dictionary*, School of Electrical Engineering, Purdue Univ., Tech. Rep. to office of Naval Research.

Even, S., 1979, *Graph algorithms*, Computer Science Press, Rockville, MD.

Gertler, J. J., 1988, *Survey of model-based detection and isolation in complex plants*, IEEE Control Syst. Mag., Vol. 8, No. 6, pp 3-11.

Gibbons, A., 1985, *Algorithmic graph theory*, Cambridge University Press, Cambridge.

Goodstein, R. L., 1964, *Boolean algebra*, Pergamon Press, London

Harrary, F., 1972, *Graph theory*, Addison-Wesley, MA.

Hochwald, W. and Bastain, J. D., 1979, *A DC approach for analogue fault dictionary determination*, IEEE Trans. Circuits Syst., Vol. CAS-26, pp.523-529.

Iserman, R., 1984, *Process fault detection based on modeling and estimation methods - a survey*, Automatica, Vol. 20, No. 4, pp. 387-404.

Jagodnik, J. E. and Wolfson, M. S., 1979, *Systematic fault simulation in an analog circuit simulator*, IEEE Trans. Circuits Syst., Vol. CAS-26, NO. 7.

Kaurz, W. H., 1968, *Fault testing and diagnosis in combinatorical digital circuits*, IEEE Trans. on computers, April.

Lin, P. M. and Elcherif, Y. S., 1982, *Fault diagnosis of nonlinear analog circuits, Volume I: DC diagnosis of hard failures*, School of Electrical Engineering, Purdue Univ., Tech. Rep. to office of Naval Research.

Lin, P. M. and Elcherif, Y. S., 1985, *Analogue circuit fault dictionary - new approaches and implementations*, Int. J. Circuit Theory Appi., pp. 149-172.

Ozawa, T., 1988, *Analog methods for computer-aided circuit analysis and diagnosis*, Marcel Dekker, Inc., New York.

Pahwa, A. and Rohrer, R. A., 1982, *Band faults: Efficient approximations to fault bands for the simulation before fault diagnosis of linear circuits*, IEEE Trans. Circuits Syst., Vol.CAS-29, pp. 82-88.

Markas, T. and Royals, M. and Kanapoulus, N., 1990, *On distributed fault simulation*, Computer, Vol. 23, No.1, pp 40 - 52.

Mironovski, L. A., 1980, *Functional diagnosis of dynamic systems - a urvey*, Automn Remote Control, 41, 1122 - 1143.

Paynter, H. M., 1961, *Analysis and design of engineering systems*, M.I.T., Press, Cambridge, MA.

Rapisarda, L. and Decarlo, R. A., 1983, *Analog multifrequency fault diagnosis*, IEEE Trans. Circuits Syst., Vol. CAS-30, No. 4.

Rosenberg, R. C. and Karnopp, D. C., 1972, *A definition of the bond graph language*, Trans. ASME, JDSMC, Vol. 94, No. 3, pp. 179-182.

Rosenberg, R. C., 1979a, *Essential gyrators and reciprocity in junction structure*, JFI, Vol. 308, No. 3, pp. 343-352.

Rosenberg, R. C. and Andry, A. N., Jr., 1979b, *A controllability test for linear systems using a graphical Technique*, IFACSymposium on Computer Aided Design of Control Systems, Zurich, pp. 143-147.

Rosenberg, R. C. and Karnopp, D. C., 1983, *Introduction to physical system dynamics*, McGraw Hill Inc., New York.

Rosenberg, R. C. and Zalewiski, Z., 1986, *Macro modeling of engineering sytems*, ASME Paper 86-WA/DSC-12, Winter Annual Meeting.

Rosenberg, R. C., 1987a, *Exploiting bond graph causality in physical system models*, Trans. ASME, JDSMC, Vol. 109, No. 4, pp. 378-383.

Rosenberg, R. C., 1987b, *ENPORT-7 Reference manual*, Rosencode Assoc., Inc., Lansing, MI.

Salama, A. E. and Starzyk, J. A., 1984, *A unified decomposition approach for fault location in large analog circuits*, IEEE Trans. Circuits Syst., Vol. CAS-31, No. 7, pp. 609-621.

Schreiber, H. H., 1979, *Fault dictionary based upon stimulus design*, IEEE Trans. Circuits Syst., Vol. CAS-26, pp. 529-537.

Seshu, S. and Waxman R., 1966, *Fault isolation in conventional linear systems - a feasibility study*, IEEE Trans. Reliability, Vol. R-15, pp. 11-16.

Sussman, G. J. and Stalman, R. M., 1975, *Heuristic techniques in computer-aided circuit analysis*, IEEE Trans. Circuits Syst., Vol. CAS-22, pp. 857-869.

Sugasaki, I. and Tanaka, H. and Kawai, M., 1989, *Fault dictionary generation to improve system level diagnostic*, NEC Res. Dev., No. 93, pp. 114-120.

Vishnubhotla, S. R. and Altan O. D., 1980, *A structured based procedure to build test sequences for the diagnosis of synchronous sequential circuits*, Proceedings of the Midwest Symposium on Circuits and Systems, August, pp. 165-169

Vlach, J. and Singhal, K., 1983, *Computer methods for circuit analysis and design*, Van Nostrand-Reinhold, New York.

Williams, T. W., 1983, *Design for testability - a survey*, Proceeding of the IEEE, Vol. 71, No. 1.

Willsky, A. S., 1976, *A survey of design methods for failure detection systems*, Automatica, Vol. 12, pp. 601-611.

Willsky, A. S., 1980, *Failure detection in dynamic systems*. AGARD No. 109.

Wu, C. C. and Nakajima, K., 1982. *Analog fault diagnosis with failure bounds*. IEEE Trans. Circuits Syst., Vol. CAS-29, No. 5.

A P P E N D I C S

APPENDIX A: Theorems and proofs

Theorem 2.1: Every catastrophic failure in a multiport system can be modeled by addition of either a fault source of flow or a fault source of effort to the proper bond graph of the system.

Proof: (by construction) By the PBG definition every effort or flow variable is associated with a simple junction. On the other hand, catastrophic faults are defined by zero flow or zero effort, thus zero flow or effort can be constrained by adding a FS to its associated junction.

Theorem 3.1: Sij is identically zero if and only if there exists a catastrophic failure associated with a simple junction on P_{ij}.

Proof:

=> (if) Suppose there exists a catastrophic fault in the system. We define two cases:

Case 1: A CFF associated with a 1-junction on P_{ij}.

Case 2: A CEF associated with a 0-junction on P_{ij}.

Proof of case 1: Please refer to Figure A.1. Let U1 be an input-junction and Y1 an output-junction, where 1A is a 1-junction associated with the fault. 1A is on a unique path from U1 to Y1. Based on Theorem 2.1, modeling of the CFF associated with 1A is done by adding a zero source of flow to this junction, as shown in Figure A.1a. The causality strokes are added to the graph in Figure A.1b. From causality considerations (Rosenberg and Karnopp, 1983), assuming the directions as shown in the figure, we get

$$f_1 = f_2 = \ldots = f_n = f_s = 0 \qquad (1)$$

and

$$e_s = e_1 + e_2 + \dots + e_n \qquad (2)$$

Since e_s is the output of eq. 2 and it only appears in this equation, we can say that it is adjusted such that eq. 2 will always hold. This implies that e_n does not depend on e_1. Since there exists only one *causal path* between Ui and Yj, the output associated with Yj does not depend on the input associated with Ui, thus Sij is identically zero.

Proof of case 2: Using the same arguments for a zero faulty junction 0A (refer to Figure A.2) we obtain

$$e_1 = e_2 = \dots = e_n = e_s = 0 \qquad (3)$$

and

$$f_s = f_1 + f_2 + \dots + f_n \qquad (4)$$

which implies, in the same way, that Sij is identically zero.

<= (only if): Let us assume that the only change that may occur in the nominal system is a catastrophic fault. Since there exists a zero entry in the SSM, and the only change allowed is a CF, then a FS has been added to some SJ of the system bond graph model. Define two cases:

Case 1: A FS has been added to a SJ on P_{ij};

Case 2: A FS has been added to a SJ outside P_{ij}.

If case 1 holds then Sij is identically zero by the first part of the proof. If case 2 holds then there exist a causal path between Ui and Yj, which implies that Sij is not zero. Since S_{ij} is zero, only case 1 holds, i.e., there exists a CF on P_{ij}.

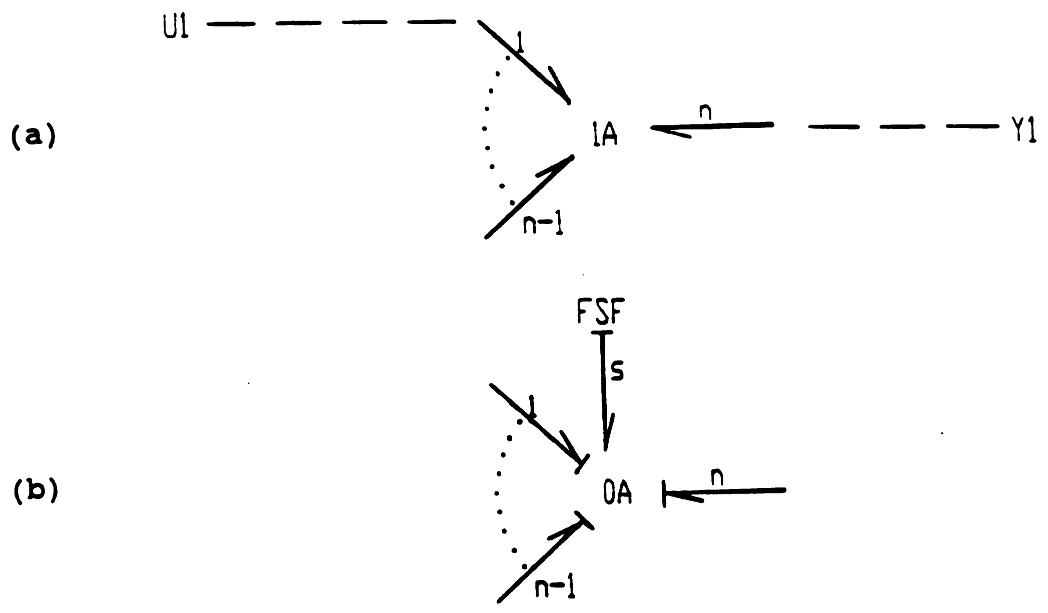


Figure A.1: A catastrophic flow fault modeling

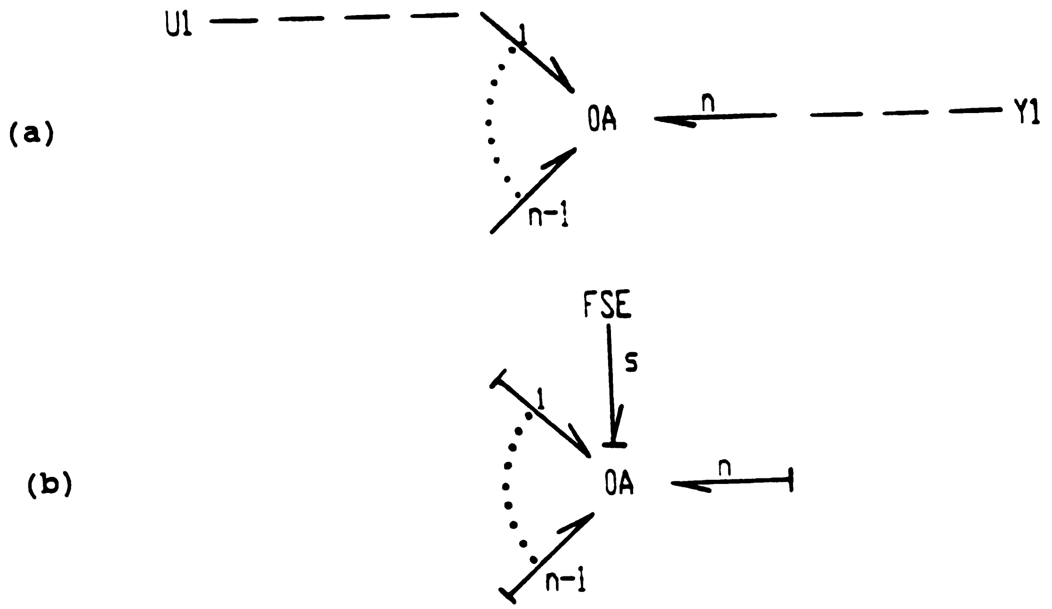


Figure A.2: A catastrophic effort fault modeling

APPENDIX B:

Construction of a Modified-Block-Graph

The construction of the MBG of a connected graph G is done by the following steps:

Step 1: Expand the original graph by duplicating its cut nodes.

Step 2: Define all blocks of more than two nodes.

Step 3: Reduce each block into one node.

Step 4: Delete unneeded duplicate cut nodes.

The procedure is demonstrated on the graph shown in Figure B.1.

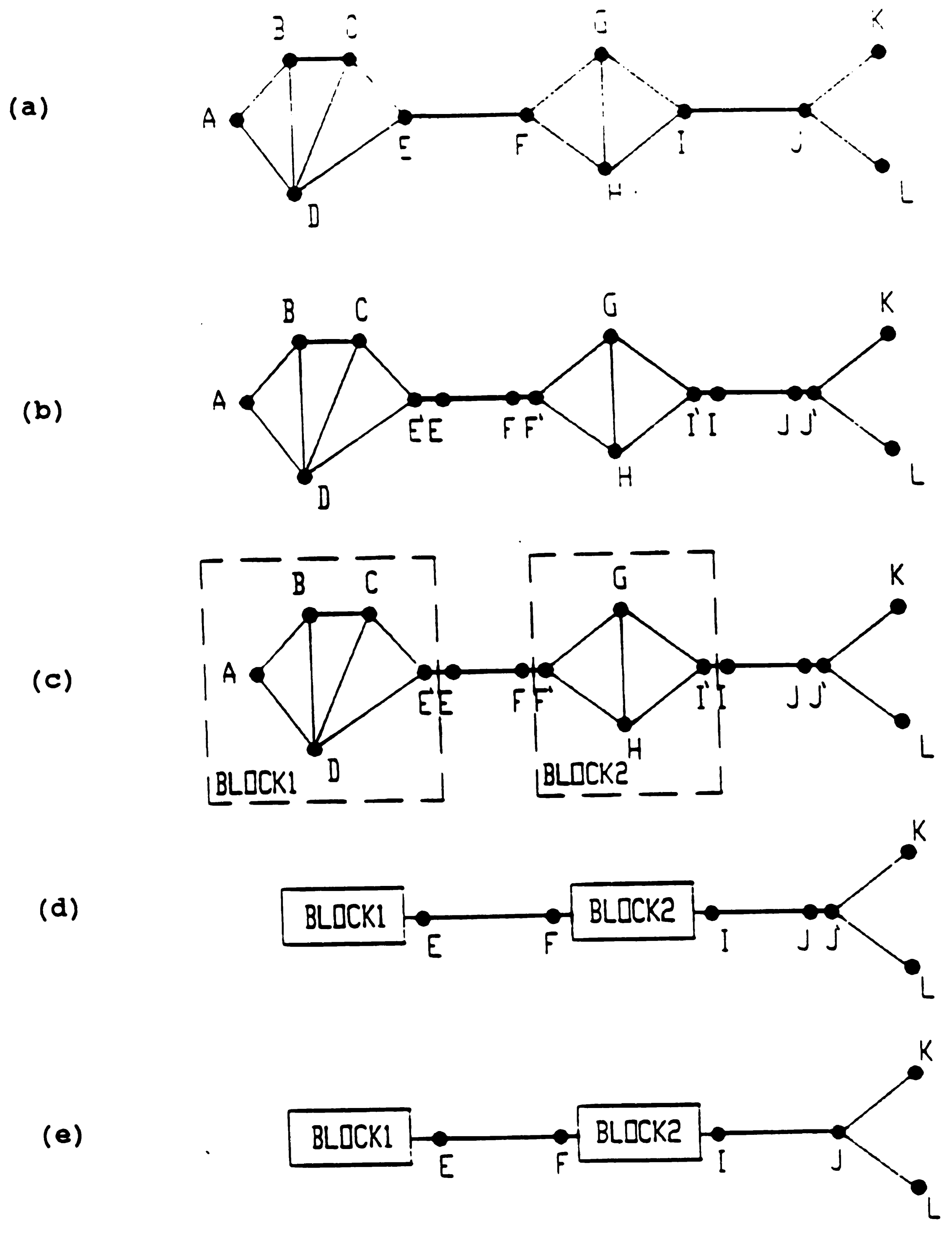

(a)
A B C D E F G H I J K L
(b)
A B C D E' E F F' G H I' I J J' K L
(c)
BLOCK1 BLOCK2
A B C D E' E F F' G H I' I J J' K L
(d)
BLOCK1 E F BLOCK2 I J J' K L
(e)
BLOCK1 E F BLOCK2 I J K L

Figure B.1: Modified-Block-Graph: (a) Original graph; (b) Duplication of cut nodes; (c) Definition of blocks; (d) Reduction of blocks to nodes; (e) MBG completed

APPENDIX C

Derivation of the nominal Structural-Sensitivity-Matrix from system structure

The bond graph represents the system equations in a structured way. Since the Structural-Sensitivity-Matrix (SSM) is concerned only with the influence of the inputs on outputs, the bond graph augmented with causality is a useful tool for the determination of the SSM. The SSM of a nominal system can be derived from the causally augmented bond graph by a series of steps, described next.

First, the bond graph is converted into a *Structural-Influence-Graph (SIG)* that represents the input-output relationships in the system. Then graph algorithms are applied to the SIG to determine the entries of the SSM of the nominal system. Section C.1 defines the SIG and the procedure for constructing it. In Section C.2 the connection between the SIG and the SSM is described. The graph algorithm presented in Section C.3 provides the directed paths of the SIG. Section C.4 provides a detailed example of the whole procedure.

C.1 The Structural-Influence-Graph

The following procedure is used to construct the SIG from a given bond graph. The bond graph requirements are as defined in section 3.1. Please refer to Figure C.1.

Step 1: Define the bond graph model and assign causality using the Sequential Causal Assignment Procedure, SCAP (Rosenberg and Karnopp, 1983).

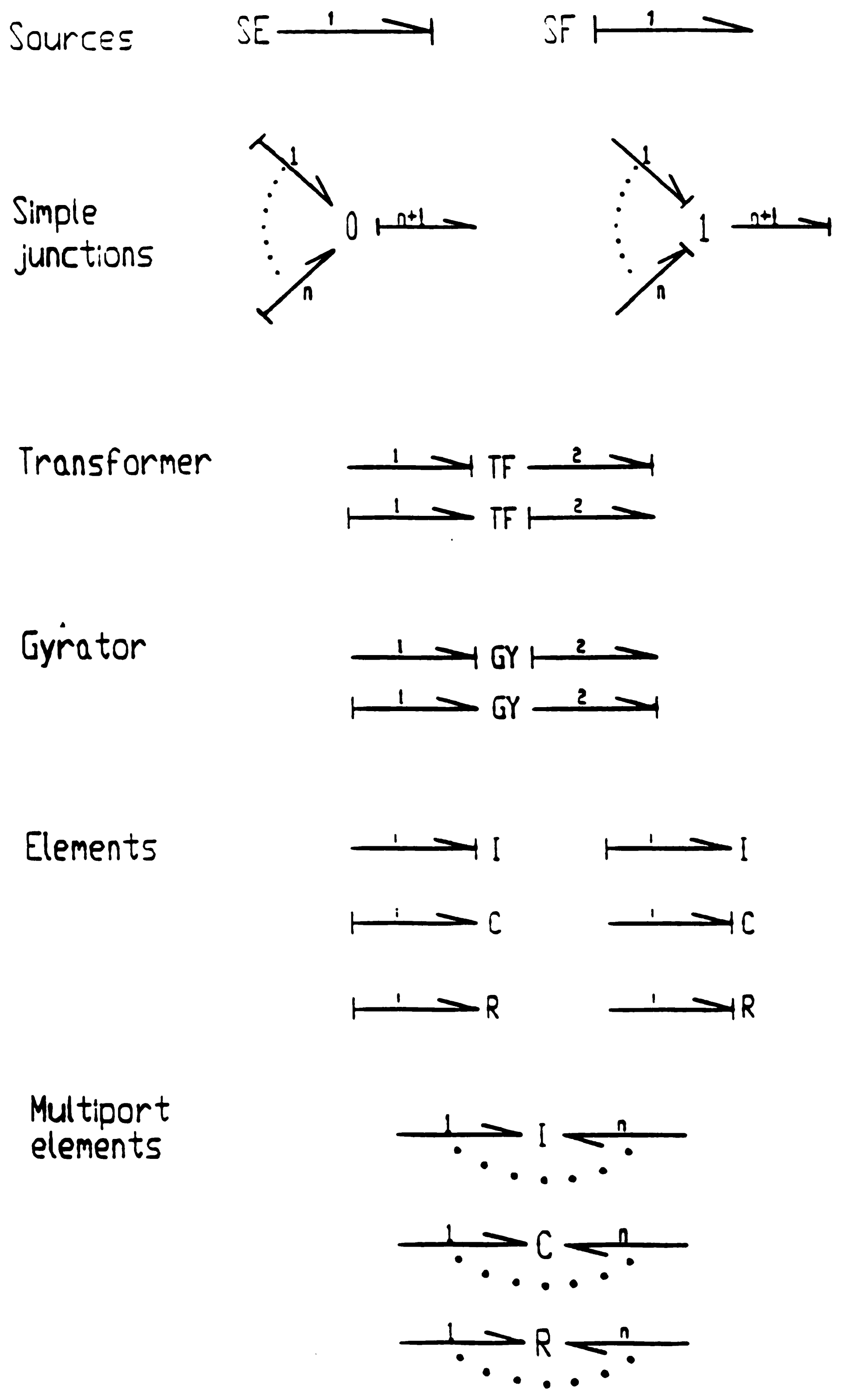


Figure C.1: Junction and element structures

Step 2: Write the input-output equations for each node of the graph. For C and I nodes ignore the state variables and use the condensed equation form indicated below. The equations for the standard bond graph nodes are:

SE: $e_i = \phi(t)$

SF: $f_i = \phi(t)$

0: $f_{n+1} = f_1+f_2+...+f_n$
$e_1 = e_{n+1}$
...
...
$e_n = e_{n+1}$

1: $e_{n+1} = e_1+e_2+...+e_n$
$f_1 = f_{n+1}$
...
...
$f_n = f_{n+1}$

TF:	$f_1 = m_1 * f_2$ $e_2 = m_1 * e_1$	or	$f_2 = m_2 * f_1$ $e_1 = m_2 * e_2$
GY:	$f_1 = r_1 * e_2$ $f_2 = r_1 * e_1$	or	$e_2 = r_2 * f_1$ $e_1 = r_2 * f_2$
I:	$f_i = \phi(e_i)$	or	$e_i = \phi(f_i)$
I(multiport):	$f_i = \phi_i(e_1,...,e_n)$ $i=1,...,n$	or	$e_i = \phi_i(f_1,...,f_n)$
C:	$e_i = \phi(f_i)$	or	$f_i = \phi(e_i)$
C(multiport):	$e_i = \phi_i(f_1,...,f_n)$ $i=1,...,n$	or	$f_i = \phi_i(e_1,...,e_n)$
R:	$e_i = \phi(f_i)$	or	$f_i = \phi(e_i)$
R(multiport):	$e_i = \phi_i(f_1,...,f_n)$ $i=1,...,n$	or	$f_i = \phi_i(e_1,...,e_n)$

where ϕ represents a general function. Note that the equations contain only effort and flow variables (and time for the source equations). Equations for C and I nodes are not written in the conventional way (using the state). The modification indicates our need to keep track of the efforts and flows only. Since our concern is with which inputs define which outputs, no explicit equations are needed. It is sufficient to use the functional forms as written above.

Step 3: Construct the SIG from the system equations. In the SIG every equation is represented by a node. The node is labeled by its output variable (effort or flow) while the edges are labeled arbitrarily. Each edge arises from an input variable to an equation.

Step 4: Augment the SIG by adding nodes (and corresponding edges) that define the given outputs.

The SIG is a directed graph.

Remarks:

A SIG input-node is defined by a zero in-degree node, and a SIG output node is defined by a zero out-degree node. The input-nodes are defined directly by the bond graph, but some of them may be of no interest to the user (for example, weight). On the other hand, outputs (test-node locations) are specified by the user and added to the SIG as shown in step 4. Some outputs that already exist in the graph may be of no interest.

C.2 Determination of the SSM from the SIG

The SIG contains each effort and flow of the bond graph as a node. Let $U_1 \ldots U_m$ be the input variables of the system and $Y_1 \ldots Y_n$ the output variables. Since inputs and outputs were restricted to efforts and flows, every input and output appear in the SIG as a node. The existence of a directed path from a specific input to a specific output is an indication of an influence (or nonzero sensitivity) of that input on that output. With the aid of standard graph algorithms, we can determine the relationship between inputs and outputs. There are three possible cases:

(a) There is no directed path from U_i to Y_j;

(b) There is a unique directed path from U_i to Y_j;

(c) There are more than one directed path from U_i to Y_j.

It can be shown mathematically that an addition of cycles along a path do not change the three basic cases (i.e., a unique directed path with a cycle is treated as a unique path, and a multi-path which contains cycles is treated as a multi-path).

Let us treat every case separately.

Case (a). Clearly $S_{ij}=0$.

Case (b). Provided ϕ along P_{ij} obey the proper conditions defined below, then $S_{ij}=1$.

<u>Proper conditions for 0:</u>

(1) Continuous function for all its input variables;

(2) Single-valued function for all its input variables;

(3) $\frac{\partial \phi}{\partial u_i}$ not identically zero for each input variables.

Case (c). In addition to the previous restrictions on 0 (Case b) the following assumption should be made: For the part of the path which are multiple (including cycles) there is no cancellation of input/output effect due to functional symmetry. Under these assumption case (c) is treated as case (b), i.e., $S_{ij}=1$.

The case of multipath is more complicated since there may be cases of cancellation of sensitivity. Here, explicit system equations are needed to determine the SSM entries. For this study we assume that there is no cancellation of sensitivity due to the multipath. In this work single-path and multi-path are treated identically.

Our next goal is to determine if there exists a directed path from each input to each output in the given SIG. Graph algorithms, in particular the Depth-First-Search algorithm, is used for this task.

C.3 The Depth-First-Search algorithm for finding the reachable nodes in a given digraph

The *Depth-First-Search (DFS)* is a well known systematic method of labeling vertices of a graph (Even, 1979; Gibbons,1985). It is used for a variety of graph theory problems. In our application we use the DFS labeling in directed graph to find directed paths between two given vertices. Here we use the fact that when a digraph is subjected to the algorithm, the edges which are Tree-type form a spanning out-forest of the graph. That forest contains an out-tree (T^*) rooted at U_i, so that the only reachable outputs (nodes) are the ones that are included in this tree.

The following algorithm constructs T^* from a given digraph G, therefore it provides the essential information about reachability of different outputs.

Step 1: Label the graph using the DFS algorithm staring in the node U_i. In the process of labeling define the edge type for each edge as either a tree edge or a non-tree edge.

Step 2: Delete all non-tree edges from the graph.

Step 3: If the resulting graph is connected go to Step 4; else delete all the components that do not contain U_i.

Step 4: Stop, the resulting graph (T^*) is an out-tree rooted at U_i.

The existence of a directed path from U^i to a specific output (defined by a node) is determined by Theorem C.1.

Theorem C.1: An output Y_j is reachable from U_i if and only if it is in T^*.

By reachable we mean that there exists at least one directed path from U_i to Y_i. Under the assumption of no cancellation, single path and multi path can be treated in the same way.

The procedure of constructing the reachable-tree (T^*) is illustrated by the following example.

Example: The digraph shown in Figure C.2a is labeled by the DFS algorithm, starting from node A. The corresponding labels (defined as Depth-First-Index, DFI) are:

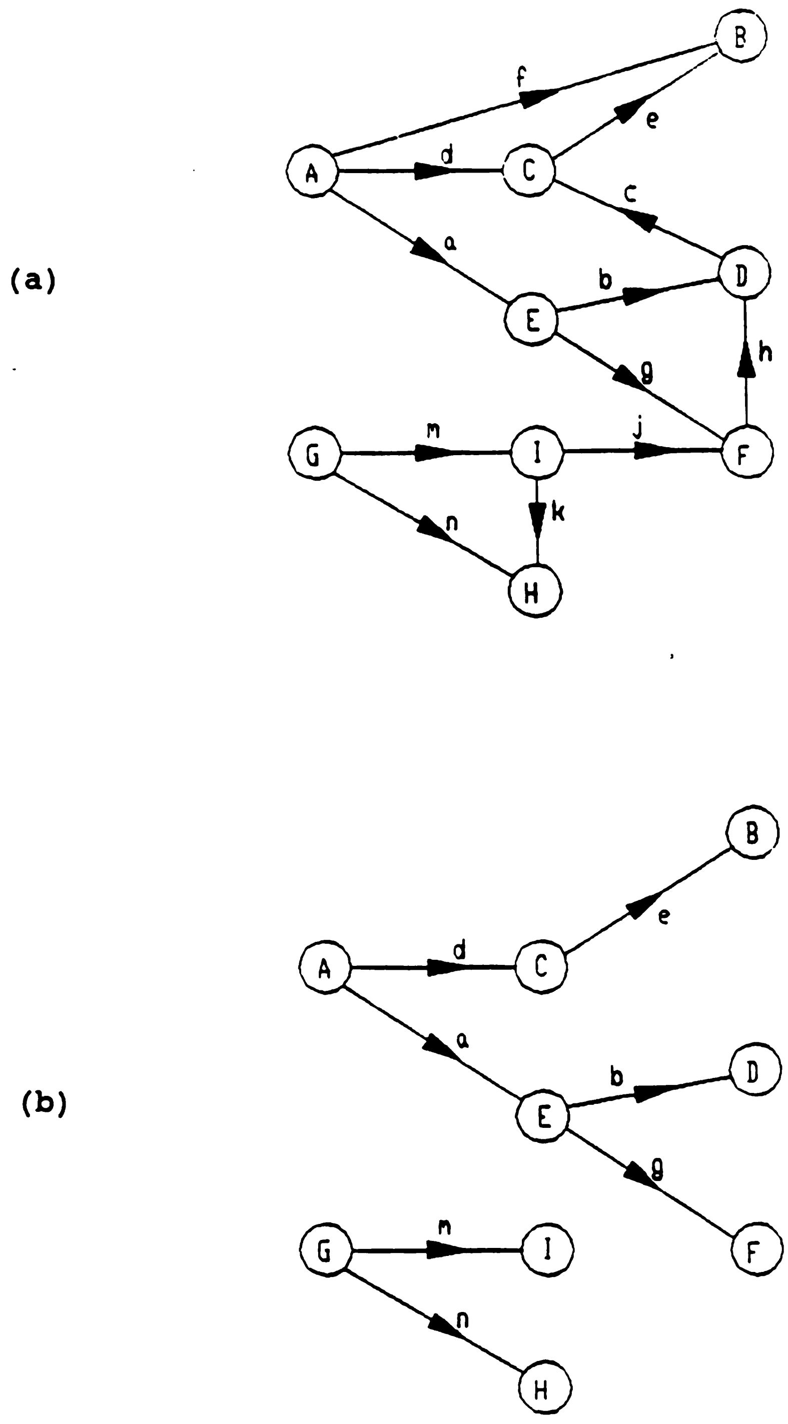


Figure C.2: An example illustrating an application of the DFS

Node	DFI
A	1
B	3
C	2
D	5
E	4
F	6
G	7
H	8
I	9

The edge-types are:

Edge	Type
a	T
b	T
c	NT
d	T
e	T
f	NT
g	T
h	NT
j	NT
k	NT
m	T
n	T

T - Tree edge
NT - Non-tree edge

After deleting all the non-tree edges we are left with a out-forest which includes two out-trees, as shown in Figure C.2b. The out-tree that contains node A is T^*. Thus the reachable nodes are B, C, D, E and F.

The procedure illustrated by this example does not distinguish between unique-path and multi-path. If further investigation is needed, all directed paths from the source to the target should be found. Algorithms that find all those paths exist in basic graph theory references (e.g., an algorithm based on Breadth-First-Search in Carre, 1979).

C.4 Example

A simple mechanical system, shown in Figure C.3a, consists of two masses, m1 and m2, connected by two springs, k1 and k2. The system bond graph model is provided in Figure C.3b. The system contains two inputs: a moving support, associated with SF1, and a constraint of velocity on the mass m2 (associated with SF2). For simplicity, gravity is ignored. The observed outputs are the velocity of m2, and the force in the spring k1. In bond graph term we define the inputs as f_1 and f_4, and the outputs as f_8 and e_2.

Sixteen equation are derived from the system bond graph.

SF1: $f_1 = \phi_1(t)$

0A: $f_2 = f_1 - f_3$
$e_3 = e_2$
$e_1 = e_2$

1A: $e_4 = e_5 + e_6 - e_3$
$f_3 = f_4$
$f_5 = f_4$
$f_6 = f_4$

SF2: $f_4 = \phi_2(t)$

I1: $e_5 = \phi_3(f_5)$

0B: $f_7 = f_6 - f_8$
$e_6 = e_7$
$e_8 = e_7$

C1: $e_2 = \phi_4(f_2)$

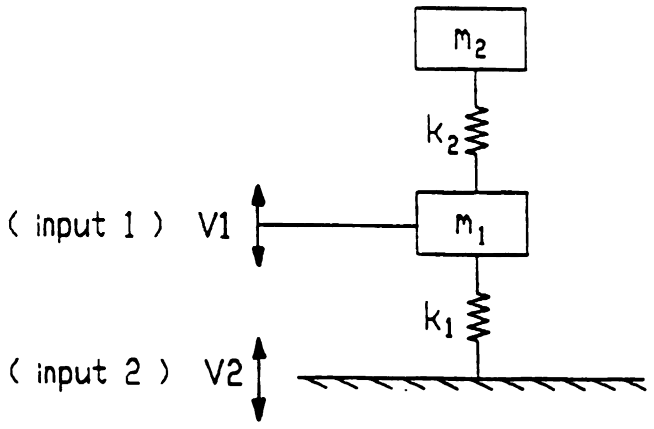
C2: $e_7 = \phi_5(f_7)$

I2: $f_8 = \phi_6(e_8)$

ϕ_i defined general functions, including integrals and derivatives.

The SIG, constructed from these equations is shown in Figure C.4.

(a)



(b)

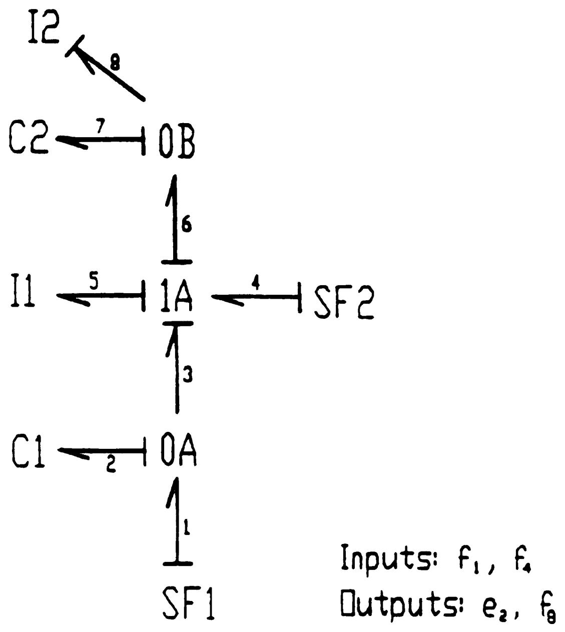


Figure C.3: A mechanical system with two velocity inputs: (a) schematic diagram; (b) bond graph model.

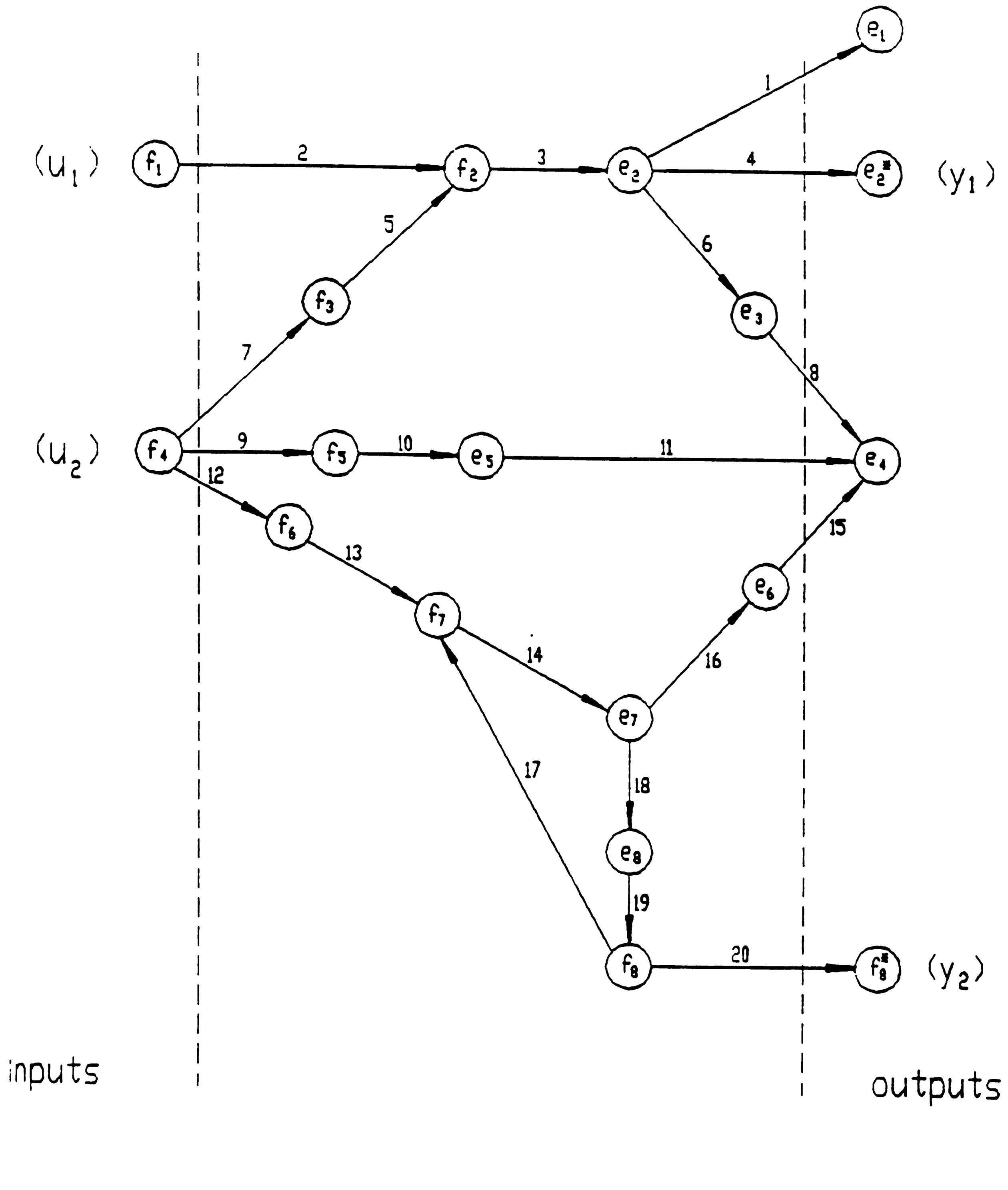


Figure C.4: The Structural-Influence-Graph of the mechanical system of Figure C.3

Observing the graph we get:

(1) There exists a unique directed path from f_1 to e_2;

(2) There is no direct path from f_1 to f_8;

(3) There exists a unique directed path from f_4 to e_2.

(4) There exists a directed path (including a cycle) from f_4 to f_8;

The corresponding SSM is, therefore:

$$SSM = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

where $U = [\ u_1,\ u_2\ ] = [\ f_1,\ f_4\ ]$

and $Y = [\ y_1,\ y_2\ ] = [\ e_2,\ f_8\ ]$

APPENDIX D

USER MANUAL FOR THE CFDIC PROGRAM

Written by Tamar Yarom

Department of Mechanical Engineering

Michigan State University

E. Lansing, Michigan 48824

June 20, 1990

1. Introduction

This manual provides the information necessary to use program CFDIC (Catastrophic Fault Dictionary). The program is used to derive the catastrophic fault dictionary for a known system. The physical system is described using bond graph with linear or nonlinear constitutive elements. This program only derives the CFD for systems which satisfy two conditions; (1) their bond graph is a connected graph, (2) systems which are completely sensitive (i.e., every output of the system is a function of every input).

The program checks connectedness before starting any computations; a fault message is provided to the user when the system bond graph is not connected. The second condition is not checked by the software, since causality is needed for checking it (see details in APPENDIX C). It is assumed that the system is completely sensitive. The current version of this program at Michigan State is run on a personal computer using FORTRAN-77 compiler. This computer program was developed as a part of the doctoral work of the author.

The author assumes that the reader is familiar with the bond graph method and terminology.

2. Installation notes

This program was written in FORTRAN 77. To install this program, compile it and correct any syntax errors, link it and correct unsatisfied external references, and run it.

3. Bond graph requirements

3.1 The bond graph model contains only bonds (no signals allowed)

3.2 The bond graph model is a connected graph.

3.3. The standard bond graph elements are:

C:	generalized capacitance,
I:	generalized inertance,
R:	generalized resistance,
SE:	effort source,
SF:	flow source,
0:	zero junction,
1:	one junction,
TF:	transformer, and
GY:	gyrator.

C, I, and R elements can have any number of ports. Constitutive equations for the C, I, and R elements, linear or nonlinear, are functions only of the local port variables (and their appropriate integrals, for C and I). Source (SE and SF) are function of time only. TF and GY elements have constant parameters.

4. How to use the program

The normal processing procedure for Program CFDIC is as follows:

4.1 As the first step, the user must create a bond graph model of the particular physical system of interest.

4.2 The system data is provided to the program by a file called SYS.DAT which contains two lists: (1) node list, which contains nodes name and type, (2) incident list, which contains bonds name and type, and their connection to the nodes. This file can be created directly (by the software) when using ENPORT software, or directly by the user (practical only for small systems). The format of SYS.DAT should be exactly as defined in the example since the program is reading the data according to that format.

4.3 The input, output and fault variables are provided by a file name VAR.DAT which contains these variables in a list.

4.4 The program provides several running options. The options are specified by the user at the beginning of VAR.DAT file. The options are:

Option 0: Derivation of CFD, information is stored in an array, D.

Option 1: Fault report. In addition to the CFD (stored the same as in Option 0) the software provides information about each fault, if it is detected or undetected, and if detected uniquely or not. Option 1 has a default option as the following: if the fault list is not specified by the user, the software provides the complete list of the detected faults and treated it as a fault list.

Option 2: Test-node report. This option provides a design-for-testability tool. It reports about unnecessary test nodes, equivalence sets of test-nodes and detectability tables for each and cumulative test-nodes.

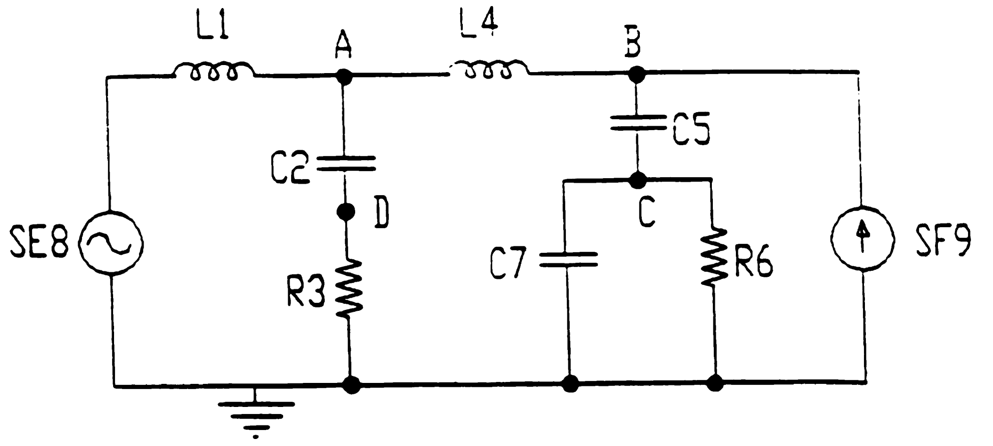
4.5 Output: The output, for each option is written into a file name A.DAT.

5. Example

Consider the circuit shown in Figure D.1a It contains two inputs, a voltage source (SE) and a current source (SF). The bond graph of the circuit is defined in Figure D.1b. The system file (SYS.DAT) is:

```
HEADING

  FILE
    ELECSYS.ENP: ELECTRICAL CIRCUIT

SYSTEM GRAPH DESCRIPTION

  NODE       TYPE
    SF9      EFG
    0B       E0G
    1BC      E1G
    0C       E0G
    R6       ERG
    C7       ECG
    C5       ECG
    1B       E1G
    I4       EIG
    0A       E0G
    1SA      E1G
    0D       E0G
    C2       ECG
    R3       ERG
    1A       E1G
    I1       EIG
    SE       EEG

  CONNECTOR  TYPE  FROM     TO
    1        BE    1SA      I1
    2        BE    1A       C2
    3        BE    1A       0D
    4        BE    1B       I4
    5        BE    1BC      C5
    6        BE    0C       R6
    7        BE    0C       C7
    8        BE    SE8      1SA
    9        BE    SF9      0B
    10       BE    0B       1BC
    11       BE    1BC      0C
    12       BE    1B       0B
    13       BE    0A       1B
    14       BE    0A       1A
    15       BE    1SA      0A
    16       BE    0D       R3
```

(a)



(b)

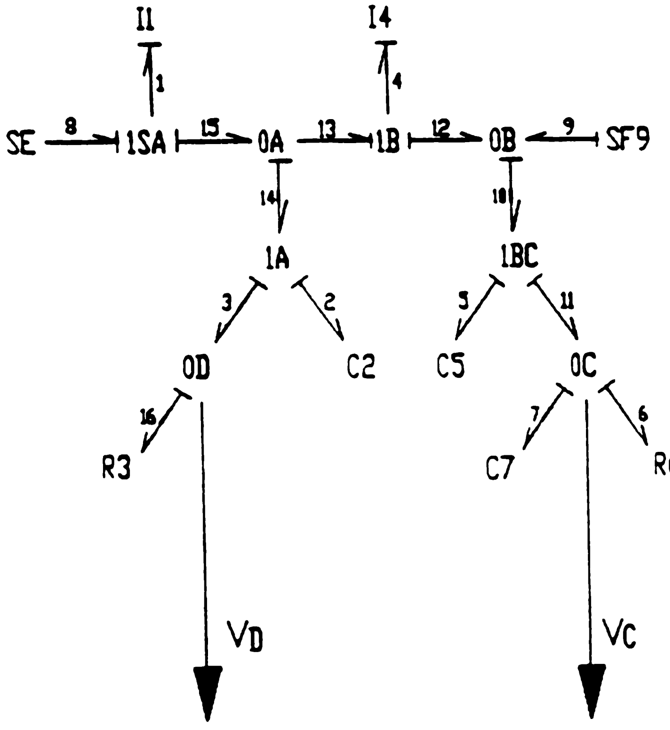


Figure D.1: Electrical circuit: (a) schematic diagram; (b) bond graph model.

System file is common to all running options, while the variable file has to be specified for each option.

Option 0

Suppose test nodes are located at points C and D, measuring the potential of those points. Fault list is defined by the following prospective faults:

Fault	Definition	Fault variable (=0)
F1	L4 open	f.4
F2	R3 open	f.3
F3	R3 shorted	e.3
F4	R6 open	f.6
F5	Point A shorted to ground	e.15
F6	Current source disconnected	f.9

Input data file for option 0 is:

```
OPTION
0

INPUT VARIABLES
e.8
f.9

OUTPUT VARIABLES
e.7
e.3

FAULT VARIABLES
f.4
f.3
e.3
f.6
e.15
f.9
```

The print out results (beside the CFD stored in the array, D) from file A.DAT is:

```
FAULT DICTIONARY, DEFINED BY PAGES
**********************************

 FOR FAULT DEFINED BY f.4   =0:
                                  0  1
                                  1  0

 FOR FAULT DEFINED BY f.3   =0:
                                  1  0
                                  1  0

 FOR FAULT DEFINED BY e.3   =0:
                                  1  0
                                  1  0

 FOR FAULT DEFINED BY f.6   =0:
                                  1  1
                                  1  1

 FOR FAULT DEFINED BY e.15  =0:
                                  0  0
                                  1  0

 FOR FAULT DEFINED BY f.9  =0:
                                  1  1
                                  0  0
```

Fault pages print out is limited to two inputs and two outputs.

Option 1

Variable data file (VAR.DAT) is similar to the previous file:

```
OPTION
1

INPUT VARIABLES
e.8
f.9

OUTPUT VARIABLES
e.7
e.3

FAULT VARIABLES
f.4
f.3
e.3
f.6
e.15
f.9
```

The results file (A.DAT):

```
FAULT REPORT
************

--------------------------------------------------------------------
  FAULT               DEFINED BY           DIAGNOSIS

--------------------------------------------------------------------
   F 1                f.4   =0             DETECTABLE, UNIQUELY

   F 2                f.3   =0             DETECTABLE, AMBIGUITY SET # 1

   F 3                e.6   =0             DETECTABLE, AMBIGUITY SET # 1

   F 4                f.6   =0             UNDETECTABLE

   F 5                e.15  =0             DETECTABLE, UNIQUELY

   F 6                f.9   =0             DETECTABLE, UNIQUELY

--------------------------------------------------------------------
  TOTAL DETECTION PERCENTAGE:  83.3%

  UNIQUE DETECTION PERCENTAGE:  50.0%

  AMBIGUITY SETS
  **************

     AMBIGUITY SET # 1 CONTAINS:
                            FAULT # 2
                            FAULT # 3
```

Option 1 (default option)

In the default option, fault variables are missing in the VAR.DAT file, thus the software provides a list of all detected faults and their report.

Input file VAR.DAT:

```
OPTION
1

INPUT VARIABLES
e.8

OUTPUT VARIABLES
e.7
e.3
```

The results file (A.DAT):

```
LIST OF FAULTS DETECTED BY
THE GIVEN INPUTS AND OUTPUTS
****************************

----------------------------------------------------------------------
 FAULT      DEFINED            ASSOCIATED        DIAGNOSIS
              BY                WITH S.J.
----------------------------------------------------------------------

 F 1        e.11   -0             0C            DETECTABLE, A.S. # 1

 F 2        f.11   -0             1BC           DETECTABLE, A.S. # 1

 F 3        e.10   -0             0B            DETECTABLE, A.S. # 1

 F 4        f.12   -0             1B            DETECTABLE, A.S. # 1

 F 5        e.13   -0             0A            DETECTABLE, A.S. # 2

 F 6        f.15   -0             1SA           DETECTABLE, A.S. # 2

 F 7        e.3    -0             0D            DETECTABLE, A.S. # 3

 F 8        f.3    -0             1A            DETECTABLE, A.S. # 3
----------------------------------------------------------------------
ABBREVIATIONS:

   S.J. - SIMPLE JUNCTION
   A.S. - AMBIGUITY SET

   AMBIGUITY SETS
   **************
      AMBIGUITY SET # 1 CONTAINS:
                              FAULT # 1
                              FAULT # 2
                              FAULT # 3
                              FAULT # 4

      AMBIGUITY SET # 2 CONTAINS:
                              FAULT # 5
                              FAULT # 6

      AMBIGUITY SET # 3 CONTAINS:
                              FAULT # 7
                              FAULT # 8
```

Option 2

Variables data file (VAR.DAT):

```
OPTION
2

INPUT VARIABLES
e.8

OUTPUT VARIABLES
e.7
e.3
f.10
f.3
e.1

FAULT VARIABLES
f.4
f.3
e.3
f.6
e.15
f.9
```

Results file (A.DAT):

```
TEST-NODES REPORT
*****************

   LIST OF UNNECESSARY TEST-NODES
   ******************************
                 TEST-NODE # 5 (DEFINED BY e.1  )

   TEST-NODES EQUIVALENCE SETS
   ***************************


      TEST-NODE EQUIVALENCE SET # 1 CONTAINS:
                 TEST-NODE # 1 (DEFINED BY e.7  )
                 TEST-NODE # 2 (DEFINED BY f.10 )


   DETECTABILITY OF TEST-NODES
   ***************************

   -TOTAL NUMBER OF FAULTS:  6
   -NUMBER OF EFFECTIVE TEST-NODES: 3
    * UNNECESSARY TEST-NODES WERE DELETED FROM THE LIST
    * AN EQUIVALENCE SET IS MARKED BY ITS FIRST T.N. VARIABLE
```

TEST-NODE NUMBER	TEST-NODE VARIABLE	NUMBER OF DETECTED FAULTS	NUMBER OF UNIQUELY DETECTED FAULTS
1	e.7	2	0
2	e.3	3	0
3	f.3	2	0

REDUCTION OF TEST-NODES
****

NUMBER OF T.N.	INCLUDING T.N. VARIABLES	TOTAL NUMBER OF D.F.	TOTAL NUMBER OF U.D.F.
1	e.3	3	0
2	e.7	4	2
3	f.3	4	4

ABBREVIATIONS:

T.N.	- TEST NODES
D.F.	- DETECTED FAULTS
U.D.F.	- UNIQUELY DETECTED FAULTS

The information of the last table (reduction of test-nodes) is described graphically in Figure D.2. This graph is not provided by the current software, but can be easily added to it.

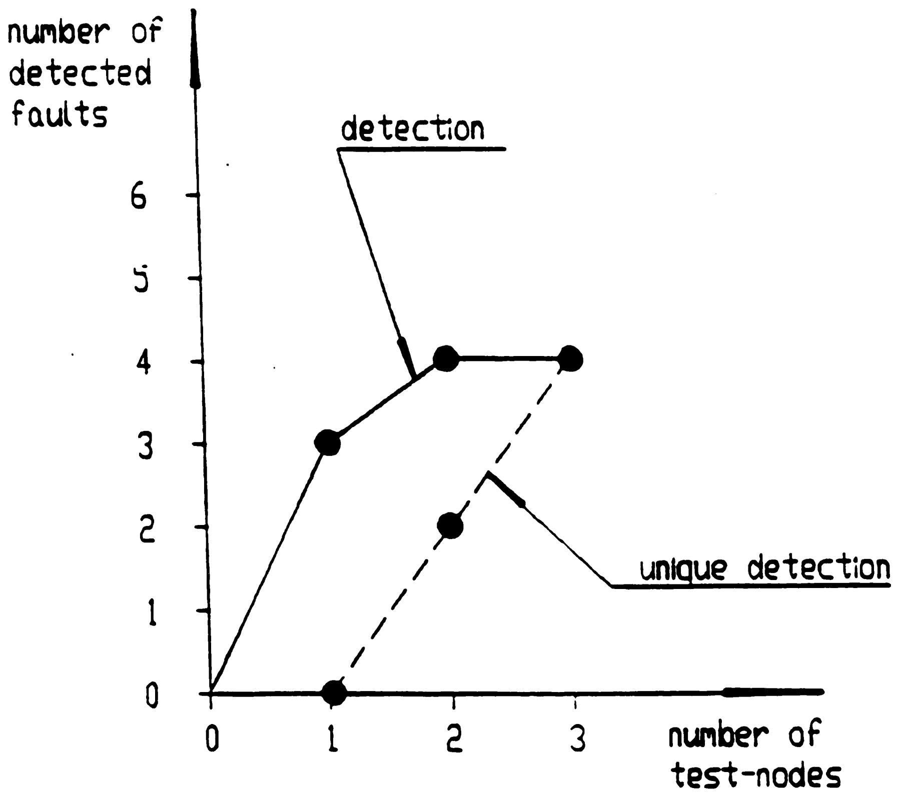


Figure D.2: Number of detected faults as a function of number of test-nodes

APPENDIX E: Implementation of the CFDIC PROGRAM

E.1 Implementation notes

These implementation notes attend to assist a user of CFDIC, or one who wishes to develop this software further. It includes efficiency matters, limitations of the program and other relevant information concern the software.

1. **Input-junctions**: Based on the graphical presentation of the FDD problem (section 2.5), an input-junction should be a simple junction associated with an input variable. Two types of sources are allowed in bond graphs, a source of effort, SE, and a source of flow, SF. Hence, an input-junction is either a 0-junction adjacent to SE or a 1-junction adjacent to SF.

From efficiency point of view, we can define the input-junctions as the source-ports, SE or SF, without changing the basic principles of paths between inputs and outputs. This is true since the sources are always leaves (nodes of 1-degree), thus a path from SE or SF to any output contains the theoretical path with addition of the source port, which is never associated with a fault. Source ports are easy to find (by the software) from the input variable data (provided by the user), and no extra junctions need to be added to the graph.

2. **Partial proper bond graph**: Following the procedure shown in Fig. 3.2, the first step, after reading in the bond graph, is to convert it into the proper bond graph. This process is associated with addition of many identity-junctions, that may be defined later as out put junction or fault-junctions. Practically, many identity-junctions are later unused,

while increasing the memory size for no reason. In order to avoid this situation, Knowing the output variables and fault variables, which are provided by the user, identity-junctions are only added to the graph when needed, i.e., when they are defined as output-junctions or fault-junctions and they do not exist in the original graph (input-junctions are defined as the sources themselves).

The missing identity junctions do not change the paths form input to outputs, since, by definition, they are two-degree junctions. Two-degree junctions, added to a graph can only lengthen existing paths without changing their topology.

3. **Software limitation**: The current version of CFDIC, used by a PC is limited to the following: 100 nodes
    100 bonds
    20 inputs
    20 test-nodes
    20 faults

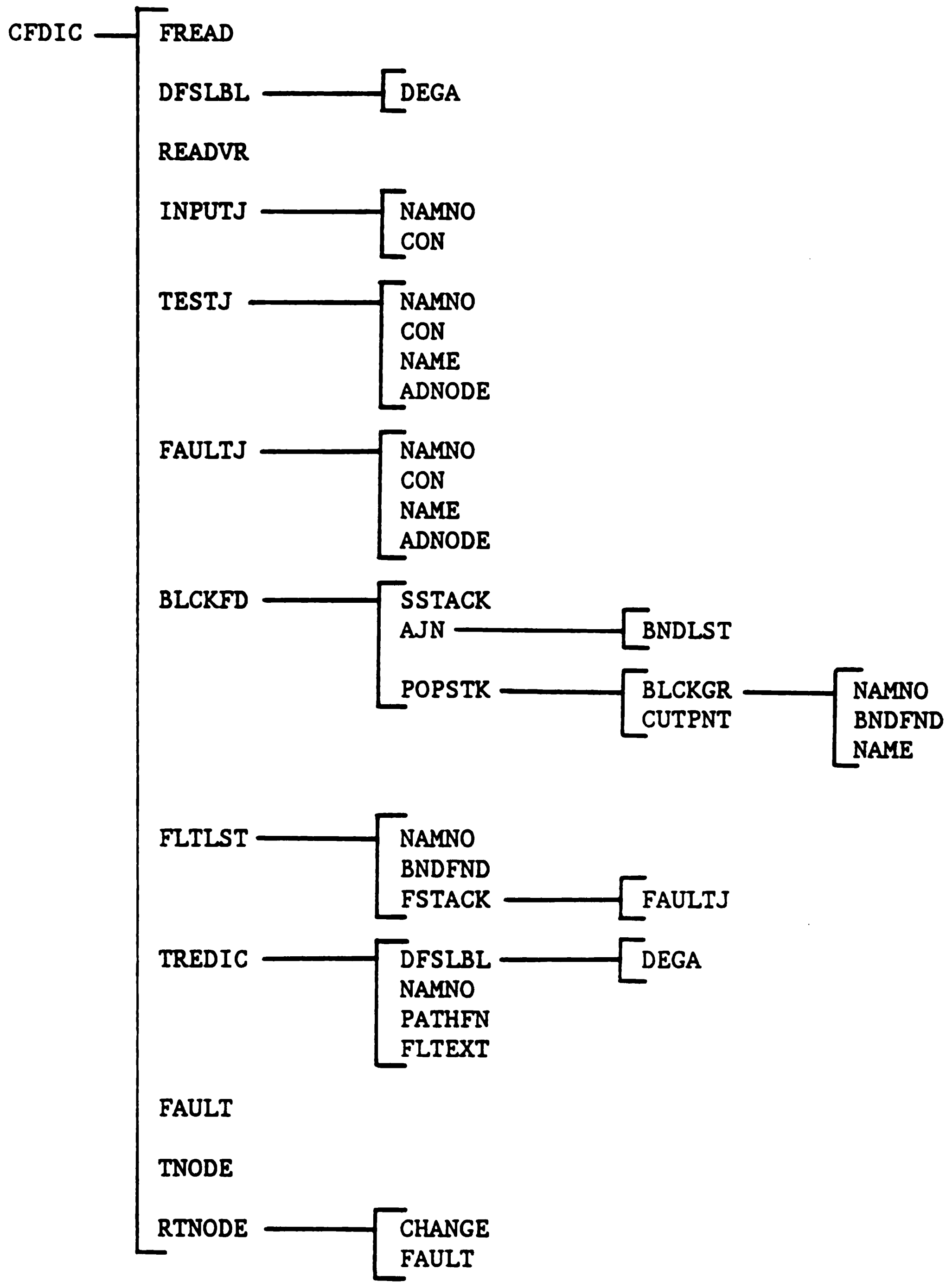
4. The bond graph is assumed to have no signals. A slight improvement of the reader would ignore signals when reading the input file.

5. Basic subroutines were written independently, such that they could be used for various applications. For example, the depth-first-search labeling (subroutine DFSLBL) is used for three purposes: checks connectedness of the graph, finds paths, and defines the blocks.

6. Unnecessary information is destroyed. After a block is defined and is changed into a block-node, the contents of that block is destroyed. If this software will extended to detect faults inside a block, this information should be kept in a separate file.

7. In the current software the MBG description, i.e., its node list and its incident list, are stored in the place as the original graph. Hence, after the MBG has been defined, including the corresponding input, output and fault junctions, the original graph no longer exists and the software relates to the MBG as the basic bond graph. This procedure should be change if the software is used also for other applications.

E.2 Calling tree structure



E.3 Listing

```
      PROGRAM MAIN
        CALL CFDIC
      END

C------------------------ CFDIC ----------------------------------------
      SUBROUTINE CFDIC
C
C---DESCRIPTION: CFDIC  (CATASTROPHIC FAULT DICTIONARY)  PROGRAM   PROVIDES,
C                BASICALLY, THE CFD FOR A GIVEN SYSTEM. THE SYSTEM IS DEFINED
C                BY ITS BOND GRAPH MODEL.INPUTS AND TEST -NODE  LOCATIONS ARE
C                SPECIFIED BY THE USER, ALSO FAULT-LIST IS  PROVIDED  BY  THE
C                USER. THE CFD IS STORED IN AN ARRAY (D). DESIGN-FOR-
C                TESTABILITY OPTIONS ARE INCLUDED IN THIS SUBROUTINE (SEE
C                DETAILS IN INPUT/OUTPUT).
C
C---INPUTS: SYSTEM DESCRIPTION, FILE: SYS.DAT, DEFINED BY TWO LISTS:
C             1. NODE-LIST, CONTAINING NODE NAMES AND TYPES;
C             2. INCIDENT-LIST, CONTAINING BOND NAMES AND THEIR CONNECTIONS.
C
C           INPUT, OUTPUT AND FAULT VARIABLES, PROVIDED IN THREE LISTS IN
C                FILE: VAR.DAT.
C
C           INDX - SPECIFY RUNNING OPTION, SEE OUTPUT.
C
C---OUTPUTS: FOR INDX=0: * D - FAULT DICTIONARY ARRAY
C
C             FOR INDX=1: * D - FAULT DICTIONARY ARRAY
C                         * FAULT REPORT
C                default option: if fault list is not specified by the user
C                                the software provides the complete list of
C                                detected faults and the fault report.
C
C             FOR INDX=2: * TEST-NODE DETECTABILITY REPORT
C
C---LIMITATIONS: 100 NODES
C                100 BONDS
C                20 INPUTS
C                20 TEST-NODES
C                20 FAULTS
C
C---INTERNAL VARIABLES:
C
C   L     number of inputs
C   M     number of test-nodes
C   N     number of prospective faults
C   UV    input variables
C   TV    test-node variables
C   FV    fault variables
C   U     input ports
C   T     test-node junctions (SJ)
C   F     fault junctions (SJ)
C   DFI   Depth-First-Index, defined for each node
C   FA    "father" of each node, defined as the node number visited before
C   CUTP  list of cut points of the graph
C   D     catastrophic fault dictionary array
C
C---DECLARATIONS:
        CHARACTER*10 UV(20),TV(20),FV(20)
        CHARACTER*8 ELNAM(100),BDNAM(100),STACK(100),BLOCK(100)
        CHARACTER*8 U(20),T(20),F(20,2),CUTP(100),NODEB(100),BONDB(100)
        CHARACTER*8 NOTE(100),PATH(100)
        CHARACTER*4 NBXTP(100),NBXTPB(100),BDTP(100)
        CHARACTER*3 OPT(20)
        INTEGER IELSBD(100,2),IELSBB(100,2),IBDPTR(100),IBDSEL(100)
        INTEGER D(20,20,20),DFI(100),FA(100),ICUT(100),CO
        COMMON INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL,NBXTP,BDTP
        COMMON/A/STACK,IO,DFI,FA
        COMMON/BLOCK/INODE,IBOND,NODEB,BONDB,IELSBB
        COMMON/BLGRPH/IBLCK,IEXTB,NEXT
        COMMON/C/CUTP,ICUT,IC,CO,I11
        COMMON/DD/D
        COMMON/INDEX/INDX
        COMMON/FNOTES/NOTE
        COMMON/V/ UV,TV,FV,L,M,N,U,T,F,OPT
C-----------------------
        OPEN(8,FILE='A.DAT',STATUS='NEW')
C---Reads and stores system data
        WRITE(8,106)
 1      CALL FREAD
C
C---Checks if the graph is connected
 2      ICON=0
        CALL DFSLBL(ELNAM(1))
        DO 10 I=1,INELS
           IF (DFI(I).EQ.0) THEN
              WRITE (8,107)
              GOTO 99
           ENDIF
 10     CONTINUE
C
C---Reads input, output, and fault-list variables
C        IF (INDX.EQ.2) INDX=3
 3      CALL READVR
C
C---Finds input, output , and fault-list junctions associated with
C   the input, output, and fault-list variables
 4      WRITE (8,100)
        DO 15 I=1,L
           CALL INPUTJ(UV(I),U(I))
           WRITE (8,101) UV(I)
 15     CONTINUE
        IF (INDX.EQ.2) WRITE (8,110)
        IF (INDX.NE.2) WRITE (8,102)
        NEXT=0
        IEXTB=0
        DO 20 J=1,M
           CALL TESTNJ(TV(J),T(J))
           WRITE (8,101) TV(J)
 20     CONTINUE
        DO 85 I=1,INELS
           NOTE(I)='        '
 85     CONTINUE
C
        IF (INDX.EQ.11) GOTO 5
        WRITE (8,103)
        DO 25 K=1,N
           CALL FAULTJ(FV(K),F(K,1),F(K,2),OPT(K))
           WRITE (8,101) FV(K)
 25     CONTINUE
C
```

```
C---Checks if the graph is a tree
 5      IF ((INELS-INBDS).EQ.1) GOTO 8
C
C---Determines the blocks of the graph
  6      CALL BLCKFD(1)
C
C---Finds cut nodes of the graph
        I=0
        CO=0
 11     I=I+1
        IF (I.EQ.IC+1) GOTO 12
        IF (ICUT(I).EQ.1) THEN
           CO=CO+1
           CUTP(CO)=CUTP(I)
        ENDIF
        GOTO 11
C
C--Creates block-graph
C
C---Initializes
 12     INODE=0
        IBOND=0
        IBLCK=0
        DO 30 I=1,INELS
           NODEB(I)='         '
 30     CONTINUE
        DO 35 I=1,INBDS
           BONDB(I)='         '
           IELSBB(I,1)=0
           IELSBB(I,2)=0
 35     CONTINUE
C
C---Collects data for the block graph
C
C---Finds all nodes with degree one and put them on block graph
C   node list. add bonds that are adjacent to these nodes to
C   block graph.
        DO 70 I=1,INELS
           NBXTPB(I)='MBG '
           NN=IBDPTR(I+1)-IBDPTR(I)
           IF (NN.EQ.1) THEN
              INODE=INODE+1
              NODEB(INODE)=ELNAM(I)
              NBXTPB(INODE)=NBXTP(I)
           ENDIF
 70     CONTINUE
        I11=INODE
C
C---Adds all cut nodes to block-graph node list
        DO 90 I=INODE+1,INODE+CO
           NODEB(I)=CUTP(I-INODE)
           CALL NAMNO(CUTP(I-INODE),'N',JO)
           NBXTPB(I)=NBXTP(JO)
 90     CONTINUE
        INODE=INODE+CO
C
C---Adds block-graph nodes, bond list and incident list
        CALL BLCKFD(2)
C
C---Completes block-graph data base
```

```
       INELS-INODE
       INBDS-IBOND
       DO 45 I-1,INELS
          ELNAM(I)-NODEB(I)
          NBXTP(I)-NBXTPB(I)
 45    CONTINUE
       DO 55 I-1,IBOND
          BDNAM(I)-BONDB(I)
          IELSBD(I,1)-IELSBB(I,1)
          IELSBD(I,2)-IELSBB(I,2)
 55    CONTINUE
C
       IBDPTR(1)-1
       LBD-0
       DO 75 I-1,INELS
          IBDPTR(I+1)-IBDPTR(I)
          DO 65 J-1,INBDS
             IF (IELSBD(J,1).EQ.I.OR.IELSBD(J,2).EQ.I) THEN
                IBDPTR(I+1)-IBDPTR(I+1)+1
                LBD-LBD+1
                IBDSEL(LBD)-J
             ENDIF
 65       CONTINUE
 75    CONTINUE

C
       WRITE(8,109)
 8     IF (INDX.NE.11) GOTO 9
C
C---find the detected faults (only for default of option 1)
       WRITE (8,104)
       WRITE (8,105)
       DO 40 I-1,L
          CALL DFSLBL(U(I))
          DO 50 J-1,M
             CALL NAMNO(T(J),'N',J0)
             CALL PATHFND(J0,PATH,K0)
             CALL FLTLST(PATH,K0)
 50       CONTINUE
 40    CONTINUE
C
       DO 95 K-1,N
          CALL FAULTJ(FV(K),F(K,1),F(K,2),OPT(K))
 95    CONTINUE
C
C---Finds the fault dictionary for a tree graph
 9     IF (INDX.EQ.0) WRITE (8,123)
       CALL TREDIC
C
C---Process Faults/Test-Nodes Report
       IF (INDX.EQ.0) GOTO 99
       IF (INDX.EQ.1.OR.INDX.EQ.11) CALL FAULT(D,0,ND,NI,INDX)
       IF (INDX.EQ.2) THEN
          CALL TNODE
          CALL RTNODE
       ENDIF
C
 100   FORMAT(//,'   INPUT VARIABLES',/,3X,15('*'))
 101   FORMAT(6X,A10)
 102   FORMAT(/,'   OUTPUT VARIABLES',/,3X,16('*'))
```

```
103     FORMAT(/,'   FAULT VARIABLES',/,3X,15('*'))
104     FORMAT(/,' LIST OF FAULTS, DETECTED BY')
105     FORMAT(' THE GIVEN INPUTS AND TEST NODES',/,1X,31('*'))
106     FORMAT(//,'1.  D A T A',/,20('-'),//)
107     FORMAT(/,' THE GRAPH IS A DISCONNECTED GRAPH')
109     FORMAT(///,'2. R E S U L T S',/,20('-'),/)
110     FORMAT(/,'   INITIAL TEST-NODE VARIABLES',/,3X,27('*'))
112     FORMAT(10X,A8)
121     FORMAT(1X,A8,5X,A8,5X,A8)
122     FORMAT(1X,A10,12X,A8)
123     FORMAT(/,'   FAULT DICTIONARY, DEFINED BY PAGES',/,3X,34('*'))
124     FORMAT(/,' FAULT INFORMATION',/,18('*'))
 99     CLOSE(8)
        RETURN
        END
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>> ADNODE <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
        SUBROUTINE ADNODE(VAR,JCN)
C
C---DESCRIPTION: THE SUBROUTINE ADDS AN IDENTITY JUNCTION TO
C                THE MODEL WHEN NECESSARY.
C
C---INPUT: VAR - EFFORT OR FLOW VARIABLE
C
C---OUTPUT: JCN - JUNCTION ADDED TO THE GRAPH AND THE CORRESPONDING BONDS
C
C---DECLARATIONS:
        CHARACTER*10 VAR
        CHARACTER*8 ELNAM(100),BDNAM(100),BOND,NNAM1,NNAM2,JCN
        CHARACTER*4 NBXTP(100)
        CHARACTER TYP1,TYP2,V
        INTEGER IELSBD(100,2),IBDPTR(100),IBDSEL(100)
        COMMON INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL,NBXTP
C******************************
        JCN='        '
        V=VAR(1:1)
        BOND=VAR(3:10)
        CALL NAMNO(BOND,'B',I)
        CALL CON(BOND,NNAM1,NNAM2,TYP1,TYP2)
C
        INELS=INELS+1
        CALL NAME('O',ELNAM(INELS))
        JCN=ELNAM(INELS)
        INBDS=INBDS+1
        CALL NAME('B',BDNAM(INBDS))
        IELSBD(I,2)=INELS
        IELSBD(INBDS,1)=INELS
        CALL NAMNO(NNAM2,'N',K)
        IELSBD(INBDS,2)=K
        IBDPTR(1)=1
        LBD=0
        DO 20 II=1,INELS
           IBDPTR(II+1)=IBDPTR(II)
           DO 10 J=1,INBDS
              IF (IELSBD(J,1).EQ.II.OR.IELSBD(J,2).EQ.II) THEN
                 IBDPTR(II+1)=IBDPTR(II+1)+1
                 LBD=LBD+1
                 IBDSEL(LBD)=J
              ENDIF
 10        CONTINUE
 20     CONTINUE
C
        IF (V.EQ.'e') NBXTP(INELS)=' 0  '
        IF (V.EQ.'f') NBXTP(INELS)=' 1  '
 99     RETURN
        END
C
C>>>>>>>>>>>>>>>>>>>>>>> AJN <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
        SUBROUTINE AJN(J0,J1)
C
C---DESCRIPTION: AJN PROVIDES A "FREE" NODE THAT IS ADJACENT
C                TO NODE #J0.
C
C---INPUTS: J0 - THE SOURCE NODE NUMBER
C
C---OUTPUTS: J1 - NUMBER OF AN ADJACENT NODE. IF J1=0 => THERE
C                IS NO ADJACENT NODE AVAILABLE
C
C---DECLARATIONS:
        CHARACTER*8 ELNAM(100), BDNAM(100)
        INTEGER IELSBD(100,2),IBDPTR(100),IBDSEL(100)
        COMMON INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL
C******************************
        J1=0
        N=IBDPTR(J0+1)-IBDPTR(J0)
        DO 10 I=IBDPTR(J0),IBDPTR(J0)+N-1
           J=IBDSEL(I)
           CALL BNDLST(BDNAM(J),INDEX)
           IF (INDEX.EQ.0) GOTO 10
           IF (IELSBD(J,1).EQ.J0) J1=IELSBD(J,2)
           IF (IELSBD(J,1).NE.J0) J1=IELSBD(J,1)
           IF (J1.NE.0) GOTO 88
 10     CONTINUE
 88     RETURN
        END
C
C>>>>>>>>>>>>>>>>>>>>>>> BLCKFD <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
        SUBROUTINE BLCKFD(INDEX)
C
C---DESCRIPTION: THIS SUBROUTINE FINDS THE BLOCKS OF A GIVEN
C               GRAPH. THE BLOCKS ARE DEFINED BY NODE LIST.
C               EACH BLOCK IS SENT OUTSIDE THE SUBROUTINE AS
C               SOON AS IT IS FOUND. THE METHOD IS BASED ON
C               DEPTH-FIRST-SEARCH ALGORITHM. THE SUBROUTINE
C               IS USED TO FIND CUT-NODES (OPTION 1) AND TO
C               DEFINE THE BLOCK-GRAPH (OPTION 2).
C
C---INPUTS: graph description, indirectly
C           INDEX - DEFINES THE OPTION:
C                   INDEX=1  => FINDS CUT-NODES
C                   INDEX=2  => FINDS BLOCK-GRAPH
C
C---OUTPUT: DEFINITION OF EVERY BLOCK BY A LIST OF NODES
C           DFI (DEPTH FIRST SEARCH) - a list of labels for every
C                node in the graph
C           FA - A LIST OF "FATHER" NODE NUMBER FOR EACH NODE
C
```

```
C---DECLARATIONS:
        integer IBDPTR(100),IELSBD(100,2),IBDSEL(100)
        INTEGER DFI(100),DF,FA(100),P(100),ICUT(100),CO
        character*8 BDNAM(100),ELNAM(100),STACK(100),BLOCK(100)
        CHARACTER*8 BDNAML(100),CUTP(100)
        common INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL,NBXTP
        COMMON/A/ STACK,IO,DFI,FA
        COMMON/B/BDNAML
        COMMON/C/ CUTP,ICUT,IC,CO,I11
        COMMON/P/IND
C**************************
        IND=INDEX
C---Put all edges on the unused list
 1      DO 84 I=1,INBDS
 84        BDNAML(I)=BDNAM(I)
C
C---Empty the stack and initialize
        DO 83 I=1,INELS
           STACK(I)='        '
           IF (INDEX.EQ.1) CUTP(I)='        '
           DFI(I)=0
           ICUT(I)=0
 83     CONTINUE
        IO=0
        IC=0
        ID=0
        JO=1
C
 2      ID=ID+1
        DFI(JO)=ID
        P(JO)=ID
        CALL SSTACK(ELNAM(JO))
C
 3      CALL AJN(JO,J1)
        IF (J1.EQ.0) GOTO 5
C
 4      IF (DFI(J1).NE.0) THEN
           P(JO)=MIN(P(JO),DFI(J1))
           GOTO 3
        ENDIF
        IF (DFI(J1).EQ.0) THEN
           FA(J1)=JO
           JO=J1
           GOTO 2
        ENDIF
C
 5      IF (DFI(FA(JO)).EQ.1) GOTO 9
C
 6      IF (P(JO).LT.DFI(FA(JO))) THEN
           P(FA(JO))=MIN(P(FA(JO)),P(JO))
           GOTO 8
        ENDIF
C
 7      CALL POPSTK(ELNAM(JO),ELNAM(FA(JO)),KO,BLOCK)
 8      JO=FA(JO)
        GOTO 3
 9      CALL POPSTK(ELNAM(JO),ELNAM(FA(JO)),KO,BLOCK)
 10     CALL AJN(1,J1)
        IF (J1.EQ.0) GOTO 99
        IF (J1.NE.0) THEN
```

```
          J0=1
          GOTO 4
       ENDIF
 99    RETURN
       END
C
C>>>>>>>>>>>>>>>>>>>>>> BLCKGR <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
       SUBROUTINE BLCKGR(K0,BLOCK)
C
C---DESCRIPTION: THE SUBROUTINE DOES THE FOLLOWING:
C                 1. PROVIDES THE BLOCK-GRAPH NODE LIST
C                 2. PROVIDES THE BLOCK-GRAPH BOND LIST
C                 3. PROVIDES THE BLOCK GRAPH INCIDENT LIST
C                 4. UP-TO-DATES THE OUTPUT JUNCTIONS THAT WILL
C                    FIT THE BLOCK GRAPH
C
C---INPUT: 1. GRAPH DESCRIPTION (COMON)
C          2. INPUT, OUTPUT AND FAULTY JUNCTION LISTS (COMMON/V/)
C          3. CUT POINTS LIST (COMMON/C/)
C          4. K0 - NUMBER OF NODES IN THE CURRENT BLOCK
C          5. BLOCK - LIST OF NODES IN THE CURRENT BLOCK
C
C---OUTPUT: 1. BLOCK-GRAPH NODE LIST
C           2. BLOCK-GRAPH BOND LIST
C           3. BLOCK-GRAPH INCIDENT LIST
C           4. UP-TO-DATE OUTPUT JUNCTIONS FOR THE BLOCK-GRAPH
C              - ALL OUTPUTS ARE STORED IN COMMON BLOCKS
C---DECLARATIONS:
       INTEGER IBDPTR(100),IELSBD(100,2),IELSBB(100,2),IBDSEL(100)
       INTEGER ICUT(100),C0
       CHARACTER*10 UV(20),TV(20),FV(20)
       CHARACTER*8 ELNAM(100),NODEB(100),BDNAM(100),BONDB(100)
       CHARACTER*8 BOND,BLOCK(100),CUTP(100)
       CHARACTER*8 U(20),T(20),F(20,2)
       CHARACTER*3 OPT(20)
       COMMON INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL
       COMMON/BLOCK/INODE,IBOND,NODEB,BONDB,IELSBB
       COMMON/C/CUTP,ICUT,IC,C0,I11
       COMMON/V/UV,TV,FV,L,M,N,U,T,F,OPT
C******************************
       IF (K0.EQ.2) THEN
          IBOND=IBOND+1
          CALL NAMNO(BLOCK(1),'N',J0)
          CALL NAMNO(BLOCK(2),'N',J1)
          CALL BNDFND(J0,J1,BOND)
          BONDB(IBOND)=BOND
          DO 10 I=1,INODE
             IF (NODEB(I).EQ.BLOCK(1)) J0=I
             IF (NODEB(I).EQ.BLOCK(2)) J1=I
 10       CONTINUE
          IELSBB(IBOND,1)=J0
          IELSBB(IBOND,2)=J1
       ELSE
          INODE=INODE+1
          CALL NAME('N',NODEB(INODE))
          DO 20 IB=1,K0
             JJ=0
             DO 30 I=1,C0
                IF (BLOCK(IB).EQ.CUTP(I)) JJ=I
 30           CONTINUE
              IF (JJ.NE.0) THEN
                 IBOND=IBOND+1
                 CALL NAME('B',BONDB(IBOND))
                 IELSBB(IBOND,1)=INODE
                 IELSBB(IBOND,2)=JJ+I11
              ENDIF
 20        CONTINUE
C
C---Up to dates output junctions
           J=0
 5         JJ=0
           J=J+1
           IF (J.GT.M) GOTO 6
           IF (T(J)(1:4).EQ.'BLCK') GOTO 5
           DO 40 I=1,CO
              IF (T(J).EQ.CUTP(I)) JJ=1
 40        CONTINUE
           IF (JJ.EQ.1) GOTO 5
           DO 50 IB=1,KO
              IF (T(J).EQ.BLOCK(IB)) THEN
                 T(J)=NODEB(INODE)
                 GOTO 5
              ENDIF
 50        CONTINUE
  6     ENDIF
  99    RETURN
        END
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> BNDFND <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
        SUBROUTINE BNDFND(J0,J1,BOND)
C
C---DESCRIPTION: THIS SUBROUTINE FINDS THE BOND WHICH CONNNECTS
C                TWO GIVEN NODES.
C
C---INPUT: J0 - FIRST NODE NUMBER
C          J1 - SECOND NODE NUMBER
C
C---OUTPUT: BOND - BOND NAME WHICH CONNECTS NODE #J0 TO NODE #J1.
C                  IF BOND REMAINS BLANK, J0 OR J1 OR BOTH DONT
C                  EXIST IN THE GRAPH
C
C---DECLARATIONS:
        CHARACTER*8 ELNAM(100),BDNAM(100),BOND
        INTEGER IELSBD(100,2),IBDPTR(100),IBDSEL(100)
        COMMON INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL
C*****************************
        BOND='        '
        K=0
        DO 10 I=1,INBDS
           IF (IELSBD(I,1).EQ.J0.AND.IELSBD(I,2).EQ.J1) K=I
           IF (IELSBD(I,1).EQ.J1.AND.IELSBD(I,2).EQ.J0) K=I
 10     CONTINUE
        IF (K.NE.0) BOND=BDNAM(K)
        RETURN
        END
C
C<>>>>>>>>>>>>>>>>>>>>>>>>>>>> BNDLST <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
```

```
      SUBROUTINE BNDLST(BOND.INDEX)
C
C---DESCRIPTION: BNDLST CHECKS IF "BOND" IS AN UNUSED EDGE.
C
C---INPUTS: BOND - BOND NAME
C          BDNAML - UP TO DATE LIST OF UNUSED EDGES (STORED
C                   AND UP-TO-DATE INSIDE THE SUBROUTINE)
C---OUTPUTS: INDEX - IF "BOND" IS AN UNUSED EDGE THEN INDEX-1,
C                   OTHERWISE INDEX-0
C
C---DECLARATIONS:
      CHARACTER*8 ELNAM(100), BDNAM(100), BDNAML(100),BOND
      INTEGER IELSBD(100,2),IBDPTR(100),IBDSEL(100)
      COMMON INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL
      COMMON/B/BDNAML
C*****************************
      INDEX-0
      DO 10 I-1,INBDS
         IF (BDNAML(I).EQ.BOND) THEN
            INDEX-1
            BDNAML(I)-'        '
         ENDIF
 10   CONTINUE
      RETURN
      END
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>> CHANGE <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE CHANGE(D,J0,J1)
C
C---DESCRIPTION: THE SUBROUTINE CHANGES ROWS IN CFD, TEST-NODE #J0
C                IS REPLACED BY TEST-NODE #J1 AND VISA VERSA.
C
C---DECLARATIONS:
      CHARACTER*10 UV(20),TV(20),FV(20),TTV
      CHARACTER*8 U(20),T(20),F(20,2)
      CHARACTER*3 OPT(20)
      INTEGER D(20,20,20),DD(20,20)
      COMMON/V/UV,TV,FV,L,M,N,U,T,F,OPT
C*****************************
      DO 10 I-1,L
         DO 10 K-1,N
            DD(I,K)-D(I,J1,K)
 10   CONTINUE
      TTV-TV(J1)
      DO 20 I-1,L
         DO 20 K-1,N
            D(I,J1,K)-D(I,J0,K)
            D(I,J0,K)-DD(I,K)
 20   CONTINUE
      TV(J1)-TV(J0)
      TV(J0)-TTV
      RETURN
      END
C
C>>>>>>>>>>>>>>>>>>>>>>>> CON <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE CON(BDNAME,NNAM1,NNAM2,TYP1,TYP2)
C
C---DESCRIPTION: THE SUBROUTINE FINDS THE ADJACENT NODES AND
```

```
C               THEIR TYPE. FOR A GIVEN BOND NAME.
C
C---INPUT:  BDNAME - BOND NAME
C
C---OUTPUTS: NNAM1 - NODE NAME, ADJACENT TO THE BOND
C            NNAM2 - NODE NAME, ADJACENT TO THE BOND
C            TYP1  - TYPE OF NNAM1
C            TYP2  - TYPE OF NNAM2
C
C---DECLARATIONS:
       CHARACTER*8 BDNAME, NNAM1, NNAM2, ELNAM(100),BDNAM(100)
       CHARACTER*4 NBXTP(100), CLASS1, CLASS2
       CHARACTER TYP1,TYP2
       INTEGER IELSBD(100,2), IBDPTR(100), IBDSEL(100)
       COMMON INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL,NBXTP
C****************************
       CALL NAMNO(BDNAME,'B',I)
       IF (I.EQ.0) THEN
          WRITE(*,100) BDNAME
          GOTO 99
       ENDIF
       N1=IELSBD(I,1)
       N2=IELSBD(I,2)
       NNAM1=ELNAM(N1)
       NNAM2=ELNAM(N2)
       CLASS1=NBXTP(N1)
       CLASS2=NBXTP(N2)
       TYP1=CLASS1(2:2)
       TYP2=CLASS2(2:2)
 100   FORMAT(/,'  THE BOND NAME: ',A8,' DOES NOT EXIST ON THE GRAPH')
 99    RETURN
       END
C
C>>>>>>>>>>>>>>>>>>>>>>>>> CUTPNT <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
       SUBROUTINE CUTPNT(K0,BLOCK)
C
C---DESCRIPTION: THE SUBROUTINE DEFINES THE LIST OF CUT NODES OF THE
C               GRAPH. THIS IS DONE BY COLLECTING BLOCK DATA. ANY
C               NODE THAT APPEAR IN MORE THAN ONE BLOCK IS A CUT
C               NODE. THE LIST OF CUT NODES AND THEIR NUMBER IS
C               PROVIDED. THE BLOCKS ARE PROVIDED ONE BY ONE TO THE
C               SUBROUTINE WHICH COLLECTS THEIR DATA.
C
C---INPUTS: BLOCK - THE NODE LIST OF THE CURRENT BLOCK
C           K0 - NUMBER OF NODES IN THE CURRENT BLOCK
C
C---OUTPUTS: CUTP - A LIST OF THE CUT-NODES OF THE GRAPH (IN COMMON/C/)
C            C0 - NUMBER OF CUT-NODES
C
C---DECLARATIONS:
       CHARACTER*8 BLOCK(100),CUTP(100)
       INTEGER ICUT(100),C0
       COMMON INELS
       COMMON/C/ CUTP,ICUT,IC,C0,I11
C******************************
C
       IF (IC.EQ.0) THEN
          DO 10 I=1,K0
             CUTP(I)=BLOCK(I)
```

```
10        CONTINUE
          IC=K0
       ELSE
       DO 20 I=1,K0
          DO 30 J=1,IC
             IF (BLOCK(I).EQ.CUTP(J)) THEN
                ICUT(J)=1
                GOTO 20
             ENDIF
30        CONTINUE
          IC=IC+1
          CUTP(IC)=BLOCK(I)
20     CONTINUE
       ENDIF
99     RETURN
       END
C
C>>>>>>>>>>>>>>>>>>>>>>>> DEGA <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
       SUBROUTINE DEGA(J0,DFI,DF,J1)
C
C---DESCRIPTION: DEGA PROVIDES THE NEXT NODE (IN NODE NUMBER) FOR
C                THE DEPTH-FIRST-SEARCH, AND NOTIFY ABOUT ADDITIONAL
C                NODES. IF THE PATH IS LOCKED, SPECIAL MESSAGE IS
C                PROVIDED.
C
C---INPUTS: J0 - THE SOURCE NODE NUMBER
C          DFI - THE CURRENT DEPTH-FIRST-INDEX FOR ALL NODES
C
C---OUTPUTS: J1 - NEXT NODE ON PATH (IF THERE IS NO NODE LEFT
C                 J1=0.
C            DF - A MESSAGE ABOUT J0, IF ALL PATHS OF J0 ARE DONE -
C                 DF=0; OTHERWISE J0=1
C
C---DECLARATIONS:
       CHARACTER*8 ELNAM(100), BDNAM(100)
       INTEGER IELSBD(100,2),IBDPTR(100), IBDSEL(100),DFI(100),DF
       COMMON INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL
C*****************************
       IN=0
       J1=0
       DF=0
       N=IBDPTR(J0+1)-IBDPTR(J0)
       DO 10 I=IBDPTR(J0),IBDPTR(J0)+N-1
          J=IBDSEL(I)
          IF (IELSBD(J,1).EQ.J0) K=IELSBD(J,2)
          IF (IELSBD(J,1).NE.J0) K=IELSBD(J,1)
          IF (IN.EQ.1) GOTO 2
          IF (DFI(K).EQ.0) THEN
             J1=K
             IN=1
             GOTO 10
          ENDIF
2         IF (DFI(K).EQ.0) THEN
             DF=1
             GOTO 99
          ENDIF
10     CONTINUE
99     RETURN
       END
```

```
C
C>>>>>>>>>>>>>>>>>>>>>>>> DFSLBL <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE DFSLBL(S)
C
C---DESCRIPTION: THIS SUBROUTINE LABELS THE GRAPH FROM NODE
C                S AS ITS SOURCE. DFI(S)=1. LABELING IS DONE
C                BY DFS METHOD. FOR EVERY NODE, THE "FATHER"
C                IS DEFINED.
C
C---INPUTS: graph description, indirectly
C           S - the source element of the graph (node name)
C
C---OUTPUT: DFI (DEPTH FIRST SEARCH) - a list of labels for every
C                node in the graph (IN COMMON/A/)
C           FA - A LIST OF "FATHER" NODE NUMBER FOR EACH NODE
C                (IN COMMON/A/)
C
C---DECLARATIONS:
      integer IBDPTR(100),IELSBD(100,2),IBDSEL(100)
      INTEGER DFI(100),DF,FA(100)
      character*8 BDNAM(100),ELNAM(100),STACK(100),S
      CHARACTER WL(100)*8, V, V1
      common INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL,NBXTP
      COMMON/A/ STACK,IO,DFI,FA
C******************************
      IWL=0
      ID=1
      J0=0
      DO 10 I=1,INELS
         DFI(I)=0
         FA(I)=0
         IF (ELNAM(I).EQ.S) J0=I
 10   CONTINUE
      IF (J0.EQ.0) THEN
         WRITE (*,100) S
         GOTO 99
      ENDIF
C
      DFI(J0)=1
 1    CALL DEGA(J0,DFI,DF,J1)
      IF (DF.EQ.1) THEN
         IWL=IWL+1
         WL(IWL)=J0
      ENDIF
      IF (J1.NE.0) THEN
         ID=ID+1
         FA(J1)=J0
         DFI(J1)=ID
         J0=J1
         GOTO 1
      ENDIF
      IF (J1.EQ.0) THEN
         IF (IWL.EQ.0) GOTO 99
         J0=WL(IWL)
         IWL=IWL-1
         GOTO 1
      ENDIF
 100  FORMAT(' NODE ',A8,'DOES NOT EXIST ON THE GRAPH',/)
 99   RETURN
```

```
      END
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>> FAULT <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE FAULT(D,M,ND,NI,INDEX)
C
C---DESCRIPTION: THE SUBROUTINE PROVIDES INFORMATION ABOUT THE
C                FAULTS IN THE FAULT LIST.
C
C---INPUTS: D - CFD ARRAY
C
C---OUTPUTS: ND - NUMBER OF DETECTED FAULTS
C            NI - NUMBER OF UNIQUELY DETECTED FAULTS
C            M - NUMBER OF TEST-NODES TAKEN INTO ACCOUNT (IF M=0
C                ALL TEST-NODES IN THE LIST ARE COUNTED)
C            INDEX - DEFINES IF THE INFORMATION SHOULD BE PRINTED
C                    INDEX=1 => PRINTED INFORMATION, ELSE NO
C            - DATA ABOUT DETECTIBILITY OF EACH FAULT IN THE
C              FAULT LIST.
C            - AMBIGUITY SETS' CONTENTS.
C
C---DECLARATIONS:
      CHARACTER*10 UV(20),TV(20),FV(20)
      CHARACTER*8 U(20),T(20),F(20,2)
      CHARACTER*3 OPT(20)
      INTEGER D(20,20,20),DD(20,20),AM(100)
      COMMON/V/UV,TV,FV,L,MO,N,U,T,F,OPT
C******************************
      IF (M.EQ.0) M=MO
      IF (INDEX.EQ.1) THEN
         WRITE(8,100)
         WRITE(8,101)
         WRITE(8,102)
         WRITE(8,101)
      ENDIF
      IF (INDEX.EQ.11) THEN
         WRITE(8,101)
         WRITE(8,112)
         WRITE(8,115)
         WRITE(8,101)
      ENDIF
      ISET=1
      DO 90 K=1,N
         AM(K)=0
 90   CONTINUE
 1    DO 10 K0=1,N
         IF (AM(K0).NE.0) GOTO 10
         IN=0
         DO 20 I=1,L
            DO 20 J=1,M
               IF (D(I,J,K0).EQ.0) GOTO 2
 20      CONTINUE
         AM(K0)=111
         GOTO 10
 2       DO 30 K=K0+1,N
            IF (AM(K).NE.0) GOTO 30
            DO 40 I=1,L
               DO 40 J=1,M
                  DD(I,J)=D(I,J,K)-D(I,J,K0)
                  IF (ABS(DD(I,J)).EQ.1) GOTO 30
 40             CONTINUE
                IN=1
                AM(K0)=ISET
                AM(K)=ISET
 30          CONTINUE
             IF (IN.EQ.1) ISET=ISET+1
 10       CONTINUE
          DO 50 K=1,N
             IF (AM(K).EQ.0) AM(K)=999
             IF (AM(K).EQ.111.AND.INDEX.NE.0) WRITE(8,103) K,FV(K)
             IF (AM(K).EQ.999) THEN
                IF (INDEX.EQ.1) WRITE(8,104) K,FV(K)
                IF (INDEX.EQ.11) WRITE(8,113) K,FV(K),F(K,1)
             ENDIF
             IF (AM(K).LT.100) THEN
                IF (INDEX.NE.0) WRITE(8,111)
                IF (INDEX.EQ.1) WRITE(8,105) K,FV(K),AM(K)
                IF (INDEX.EQ.11) THEN
                   IF (F(K,1)(1:4).EQ.'EXTN') F(K,1)='*       '
                   WRITE(8,114) K,FV(K),F(K,1),AM(K)
                ENDIF
             ENDIF
 50       CONTINUE
          IF (INDEX.NE.0) WRITE(8,101)
          ND=0
          NI=0
          DO 60 K=1,N
             IF(AM(K).EQ.999) NI=NI+1
             IF(AM(K).EQ.111) ND=ND+1
 60       CONTINUE
          ND=N-ND
          IF (INDEX.EQ.1) THEN
             WRITE(8,106) ND*100./N
             WRITE(8,107) NI*100./N
          ENDIF
          IF (INDEX.EQ.11) THEN
             WRITE(8,116)
             WRITE(8,117)
             WRITE(8,118)
             WRITE(8,119)
          ENDIF
C
          IF (INDEX.NE.0) WRITE(8,108)
          DO 70 K0=1,N
             IF (AM(K0).LT.100) THEN
                IF (INDEX.NE.0) THEN
                   WRITE(8,109) AM(K0)
                   WRITE(8,110) K0
                ENDIF
                DO 80 K=K0+1,N
                   IF(AM(K).EQ.AM(K0)) THEN
                      IF (INDEX.NE.0) WRITE(8,110) K
                      AM(K)=222
                   ENDIF
 80             CONTINUE
             ENDIF
 70       CONTINUE
C
 100      FORMAT(/,'   FAULT REPORT',/,'    ************'/)
 101      FORMAT(1X,66('_'))
```

```
102     FORMAT(/,3X,'  FAULT',9X,'DEFINED BY',8X,'DIAGNOSIS',/)
103     FORMAT(/,7X,'F',I2,9X,A6,'=0',8X,'UNDETECTABLE')
104     FORMAT(/,7X,'F',I2,9X,A6,'=0',8X,'DETECTABLE, UNIQUELY')
105     FORMAT(7X,'F',I2,9X,A6,'=0',8X,'DETECTABLE, AMBIGUITY SET #',I2)
106     FORMAT(/,'   TOTAL DETECTION PERCENTAGE: ',F5.1,'%')
107     FORMAT(/,'   UNIQUE DETECTION PERCENTAGE: ',F5.1,'%')
108     FORMAT(//,'   AMBIGUITY SETS',/,3X,14('*'))
109     FORMAT(/,'      AMBIGUITY SET #',I2,' CONTAINS:')
110     FORMAT(30X,'FAULT #',I2)
111     FORMAT('   ')
112     FORMAT(/,' FAULT    DEFINED       ASSOCIATED        DIAGNOSIS')
115     FORMAT(2X,'           BY            WITH S.J.'/)
113     FORMAT(/,3X,'F',I2,5X,A6,'=0',9X,A8,7X,'DETECTED, UNIQUELY')
114     FORMAT(3X,'F',I2,5X,A6,'=0',9X,A8,7X,'DETECTABLE, A.S. #',I2)
116     FORMAT(1X,'* THE S.J. ASSOCIATED WITH THIS FAULT DOES NOT')
117     FORMAT(3X,'APPEAR IN THE ORIGINAL GRAPH')
118     FORMAT(/,' ABBREVATIONS:',/,'  S.J. - SIMPLE JUNCTION')
119     FORMAT(2X,'A.S. - AMBIGUITY SET')
 99     RETURN
        END
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> FAULTJ <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
        SUBROUTINE FAULTJ(VAR,JCN1,JCN2,OPT)
C
C---DESCRIPTION: THIS SUBROUTINE DEFINES THE FAULT JUNCTION
C                FOR A GIVEN VARIABLE.
C
C---INPUT: VAR - VARIABLE NAME, ONLY FLOW OR EFFORT ALLOWED, WRITTEN
C                AS: e.BOND-NAME OR f.BOND-NAME
C
C---OUTPUT: JCN1 - THE FIRST (OR ONLY) FAULT-LIST JUNCTION NAME
C           JCN2 - THE SECOND FAULT-LIST JUNCTION NAME
C           OPT  - INDICATES THE EXISTANCE OF TWO ADJACENT
C                  ZERO-JUNCTIONS OR TWO ADJACENT 1-JUNCTIONS
C
C---DECLARATIONS:
        CHARACTER VAR*10,JCN1*8,JCN2*8
        CHARACTER*8 ELNAM(100),BDNAM(100),BOND,NNAM1,NNAM2,NOTE(100)
        CHARACTER*4 NBXTP(100)
        CHARACTER*3 OPT
        CHARACTER TYP1,TYP2,V
        INTEGER IELSBD(100,2),IBDPTR(100),IBDSEL(100)
        COMMON INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL,NBXTP
        COMMON/FNOTES/NOTE
C******************************
C---initializes
        OPT='   '
        JCN1='        '
        JCN2='        '
        V=VAR(1:1)
        BOND=VAR(3:10)
C
C---Checks whether the bond appear on the graph
        CALL NAMNO(BOND,'B',I)
        IF (I.EQ.0) THEN
           WRITE (8,100) BOND
           GOTO 99
        ENDIF
C
```

```
        IF (V.EQ.'e'.OR.V.EQ.'f') GOTO 1
        WRITE (8,101) VAR
        GOTO 99
 1      CALL CON(BOND,NNAM1,NNAM2,TYP1,TYP2)
        IF (V.EQ.'e') THEN
           IF (TYP1.EQ.'0') THEN
              JCN1=NNAM1
              CALL NAMNO(NNAM1,'N',II)
              IF (NOTE(II).EQ.'        ') GOTO 8
              JCN2=NOTE(II)
              OPT='OR '
 8            IF (TYP2.EQ.'0') THEN
                 OPT='OR '
                 JCN2=NNAM2
              ENDIF
              GOTO 99
           ENDIF
           IF (TYP2.EQ.'0') THEN
              JCN1=NNAM2
              GOTO 99
           ENDIF
           CALL ADNODE(VAR,JCN1)
           CALL NAMNO(NNAM1,'N',I0)
           NOTE(I0)=NNAM2
           GOTO 99
 5      ELSEIF (V.EQ.'f') THEN
           IF (TYP1.EQ.'1') THEN
              JCN1=NNAM1
              CALL NAMNO(NNAM1,'N',II)
              IF (NOTE(II).EQ.'        ') GOTO 9
              JCN2=NOTE(II)
              OPT='OR '
 9            IF (TYP2.EQ.'1') THEN
                 OPT='OR '
                 JCN2=NNAM2
              ENDIF
              GOTO 99
           ENDIF
           IF (TYP2.EQ.'1') THEN
              JCN1=NNAM2
              GOTO 99
           ENDIF
           CALL ADNODE(VAR,JCN1)
           CALL NAMNO(NNAM1,'N',I0)
           NOTE(I0)=NNAM2
           GOTO 99
        ELSE
           WRITE (8,101) VAR
        ENDIF
 100    FORMAT(/,' BOND NAMED: ',A8,' DOES NOT EXIST ON THE GRAPH')
 101    FORMAT(/,2X,A10,' IS NOT AN EXCEPTABLE FAULT VARIABLE')
 99     RETURN
        END
C
C>>>>>>>>>>>>>>>>>>>>>>>> FLTEXT <<<<<<<<<<<<<<<<<<<<<<<<<<<
C
        SUBROUTINE FLTEXT(PATH,K0,F1,F2,OPT,IN)
C
C---DESCRIPTION: FLTEXT (FauLT-EXisT) finds out if the fault F
C                (defined by an element(s) name) exists on the
```

```
C                 path defined by PATH.
C
C---INPUTS: PATH - a list of elements
C           INELS - number of elements in the graph (common)
C           F1 - the first element name which defines a fault
C           F2 - the second element which defines a fault, if
C                the fault is defined by one element - F2 will
C                be empty
C           KO - NUMBER OF NODES IN THE PATH
C           OPT - a special indication of existance of two
C                 adjacent junctions or junctions as faulty junctions.
C
C---OUTPUT: in - index which defines the existance of F on PAth
C                if IN=0 then the fault (F1, F2) is on PATH
C                if IN=1 then the fault (F1, F2) is not on PATH
C
C---DECLARATIONS:
       character*8 PATH(100),F1,F2
       CHARACTER*3 OPT
C*****************************
C
       IN=1
       IN1=1
       IN2=1
       IF (OPT.EQ.'   ') THEN
          DO 10 I=1,KO
             IF (F1.EQ.PATH(I)) IN=0
 10       CONTINUE
       ELSE
          DO 20 I=1,KO
             IF (F1.EQ.PATH(I)) IN1=0
             IF (F2.EQ.PATH(I)) IN2=0
 20       CONTINUE
          IF (OPT.EQ.'AND') THEN
             IF (IN1.EQ.0.AND.IN2.EQ.0) IN=0
          ENDIF
          IF (OPT.EQ.'OR ') THEN
             IF (IN1.EQ.0.OR.IN2.EQ.0) IN=0
          ENDIF
       ENDIF
       RETURN
       END
C
C>>>>>>>>>>>>>>>>>>>>>>>>>> FLTLST <<<<<<<<<<<<<<<<<<<<<<<<<<
C
       SUBROUTINE FLTLST(PATH,KO)
C
C---DESCRIPTION: THE SUBROUTINE COLLECTS THE DETECTABLE FAULTS
C                THAT ARE ASSICIATED WITH A GIVEN PATH (DEFINED
C                BY NODE LIST) AND STACK THEM.
C
C---INPUTS: SYSTEM DESCRIPTION (IN COMMON)
C           PATH - A PATH DEFINED BY NODE LIST
C           KO - PATH LENGTH (NUMBER OF NODES)
C
C---OUTPUTS: FAULT/S ASSOCIATED WITH A GIVEN PATH
C            (COLLECTED BY SUBROUTINE FSTACK)
C
C---DECLARATIONS:
       INTEGER IBDPTR(100),IELSBD(100,2),IBDSEL(100)
```

```
      CHARACTER*10 VAR1,VAR2
      CHARACTER*8 BDNAM(100),ELNAM(100),PATH(100),A,B,BOND
      CHARACTER*4 NBXTP(100)
      INTEGER IN(100)
      COMMON INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL,NBXTP
C*******************************
      INDEX=0
      DO 10 K=1,K0-1
         IN(K)=0
         A=PATH(K)
         B=PATH(K+1)
         CALL NAMNO(A,'N',I0)
         CALL NAMNO(B,'N',J0)
         CALL BNDFND(I0,J0,BOND)
C
         IF (K.EQ.K0-1) THEN
            VAR1(3:10)=BOND
            IF (NBXTP(J0)(2:2).EQ.'E') VAR1(1:2)='f.'
            IF (NBXTP(J0)(2:2).EQ.'F') VAR1(1:2)='e.'
            CALL FSTACK(VAR1)
            GOTO 10
         ENDIF
C
         IF (K.EQ.1.AND.BOND(1:3).EQ.'EXT') THEN
            CALL NAMNO(A,'N',J0)
            NN=IBDPTR(J0+1)-IBDPTR(J0)
            DO 20 I=IBDPTR(J0),IBDPTR(J0)+NN-1
               BOND=BDNAM(IBDSEL(I))
               IF(BOND(1:3).EQ.'EXT') GOTO 20
               VAR1(3:10)=BOND
               IF (NBXTP(I0)(2:2).EQ.'0') VAR1(1:2)='e.'
               IF (NBXTP(I0)(2:2).EQ.'1') VAR1(1:2)='f.'
               CALL FSTACK(VAR1)
               GOTO 10
 20         CONTINUE
         ENDIF
         IF (BOND(1:3).EQ.'EXT') THEN
            IN(K)=1
            IF (K.GT.1.AND.IN(K-1).EQ.1) THEN
              IF (NBXTP(I0)(2:2).EQ.'0') WRITE(8,100) A
              IF (NBXTP(I0)(2:2).EQ.'1') WRITE(8,100) A
            ENDIF
            GOTO 10
         ENDIF
         IF (NBXTP(I0)(2:2).EQ.'T'.OR.NBXTP(I0)(2:2).EQ.'G') THEN
            INDEX=1
            GOTO 10
         ENDIF
         VAR1(1:2)='e.'
         VAR1(3:10)=BOND
         VAR2(1:2)='f.'
         VAR2(3:10)=BOND
         IF (INDEX.EQ.1) THEN
            IF (NBXTP(I0)(2:2).EQ.'0') CALL FSTACK(VAR2)
            IF (NBXTP(I0)(2:2).EQ.'1') CALL FSTACK(VAR1)
            INDEX=0
            GOTO 10
         ENDIF
         CALL FSTACK(VAR1)
         CALL FSTACK(VAR2)
 10     CONTINUE
 100    FORMAT ('      VARIABLE ASSOCIATED WITH JUNCTION: ',A6)
        RETURN
        END
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> END FLTLST <<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
CFILE:FREAD ------------ FREAD ----------------------------------
C
C--- PURPOSE: Reads the header segment and graph data segment from
C             ENPORT file, and adds node and connector data to the
C             work space

        SUBROUTINE FREAD
        CHARACTER*4 STRING*80,H*5,NB*1,NBXTP(100),CLASS,BDTP(100),BTP
        CHARACTER*8 NDNAM,ELNAM(100),BDNAM(100),BNAM,FRN,TON,NNN(100,2)
        INTEGER IELSBD(100,2), IBDPTR(100), IBDSEL(100)
        COMMON INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL,NBXTP,BDTP
C
        OPEN (14,FILE='SYS.DAT',STATUS='OLD')
 5      READ(14,100) STRING
        H=STRING(1:5)
        IF (H.NE.' HEAD') GOTO 5
 6      READ (14,100) STRING
        H=STRING(1:5)
        IF (H.NE.'   FI') GOTO 6
        READ(14,100) STRING
C
C---Writes file name on the screen
        WRITE(8,101) STRING
7       READ(14,100) STRING
        H=STRING(1:5)
        IF (H.NE.'   NO') GOTO 7
C--- Initializes
        INELS=0
        INBDS=0
        NEL=0
        NBD=0
C---Adds node data to the workspace, as read from file
  8     READ(14,100) STRING
        READ (STRING,102) NDNAM,CLASS
        IF (NDNAM.EQ.'        ') GOTO 9
        INELS=INELS+1
        IF (CLASS(1:1).EQ.'M') NEL=NEL+1
        IF (INELS.LT.100) THEN
             ELNAM(INELS)=NDNAM
             NBXTP(INELS)=CLASS
             GOTO 8
        ENDIF
C
 9      READ (14,100) STRING
        H=STRING(1:5)
        IF (H.NE.'   CO') GOTO 9
C
C---Adds connector data to workspace, as read from file
 11     READ(14,100) STRING
C
C---Get connector name, type, from-node, to-node
        READ (STRING,103) BNAM,BTP,FRN,TON
C
C---See if this is a continuation line for current connector
```

```
      IF (BNAM.EQ.'        ') GOTO 10
      INBDS=INBDS+1
      IF (BTP(1:1).EQ.'B') NBD=NBD+1
      IF (INBDS.LT.100) THEN
         BDNAM(INBDS)=BNAM
         BDTP(INBDS)=BTP
         NNN(INBDS,1)=FRN
         NNN(INBDS,2)=TON
         GOTO 11
      ENDIF
 10   DO 30 I=1,INBDS
         CALL NAMNO(NNN(I,1),'N',NO)
         IELSBD(I,1)=NO
         CALL NAMNO(NNN(I,2),'N',NO)
         IELSBD(I,2)=NO
 30   CONTINUE
C--- Complete the GREDBK data base
      IBDPTR(1)=1
      LBD=0
      DO 90 I=1,INELS
         IBDPTR(I+1)=IBDPTR(I)
         DO 80 J=1,INBDS
            IF (IELSBD(J,1).EQ.I.OR.IELSBD(J,2).EQ.I) THEN
               IBDPTR(I+1)=IBDPTR(I+1)+1
               LBD=LBD+1
               IBDSEL(LBD)=J
            ENDIF
 80      CONTINUE
 90   CONTINUE
 99   CLOSE(14)
 100  FORMAT(A80)
 101  FORMAT('    FILE NAME:',A40)
 102  FORMAT(5X,A8,1X,A4))
 103  FORMAT(5X,A8,1X,A4,2X,A8,1X,A8,2X)
      RETURN
      END
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> FSTACK <<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE FSTACK(FVAR)
C
C---DESCRIPTION: THE SUBROUTINE ADDS FAULT VARIABLES TO THE FAULT LIST.
C                FAULT ARE ADDED TO THE TOP OF THE LIST ONLY IF THEY
C                DO NOT EXIST THERE ALREADY. NUMBER OF FAULTS IN THE
C                FAULT LIST IS UP-TO-DATED.
C
C---INPUTS: FVAR - FAULT VARIABLE TO BE ADDED TO THE FAULT LIST
C           FV - OLD FAULT LIST (STORED IN COMMON/V/)
C
C---OUTPUTS: FV - UP-TO-DATE FAULT LIST (STORED IN COMMON/V/)
C            N - NUMBER OF FAULTS IN THE FAULT LIST (STORED IN COMMON/V/)
C
C---DECLARATIONS:
      CHARACTER*10 UV(20);TV(20),FV(20),FVAR
      CHARACTER*8 U(20),T(20),F(20,2),FVAR1,FVAR2
      CHARACTER*8 ELNAM(100),BDNAM(100)
      CHARACTER*4 NBXTP(100)
      CHARACTER*3 OPT(20),OPTVAR
      INTEGER IELSBD(100,2),IBDPTR(100),IBDSEL(100)
      COMMON INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL,NBXTP
      COMMON/V/ UV,TV,FV,L,M,N,U,T,F,OPT
C**************************
      IF (FVAR(3:10).EQ.'        ') GOTO 99
      INDEX=0
      IF (N.EQ.0) GOTO 5
      CALL FAULTJ(FVAR,FVAR1,FVAR2,OPTVAR)
      DO 10 I=1,N
         IF (FV(I).EQ.FVAR) THEN
            INDEX=1
            GOTO 10
         ENDIF
         CALL FAULTJ(FV(I),F(I,1),F(I,2),OPT(I))
         IF (F(I,1).EQ.FVAR1.AND.F(I,2).EQ.FVAR2) INDEX=1
 10   CONTINUE
 5    IF (INDEX.EQ.0) THEN
         N=N+1
         FV(N)=FVAR
      ENDIF
 99   RETURN
      END
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> INPUTJ <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE INPUTJ(VAR,JCN)
C
C---DESCRIPTION: THIS SUBROUTINE DEFINES THE INPUT JUNCTION FOR A
C                GIVEN INPUT VARIABLE.
C
C---INPUT: VAR - VARIABLE NAME, ONLY FLOW OR EFFORT ALLOWED, WRITTEN
C                AS: e.BOND-NAME OR f.BOND-NAME
C
C---OUTPUT: JCN - THE INPUT JUNCTION NAME (TYPE SE OR SF)
C
C---DECLARATIONS:
      CHARACTER*10 VAR
      CHARACTER*8 ELNAM(100),BDNAM(100),BOND,NNAM1,NNAM2,JCN
      CHARACTER*4 NBXTP(100)
      CHARACTER TYP1,TYP2,V
      INTEGER IELSBD(100,2),IBDPTR(100),IBDSEL(100)
      COMMON INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL,NBXTP
C*****************************
      JCN='        '
      V=VAR(1:1)
      BOND=VAR(3:10)
      CALL NAMNO(BOND,'B',I)
      IF (I.EQ.0) THEN
         WRITE (8,100) BOND
         GOTO 99
      ENDIF
      CALL CON(BOND,NNAM1,NNAM2,TYP1,TYP2)
      IF (V.EQ.'e') THEN
         IF (TYP1.EQ.'E') THEN
            JCN=NNAM1
            GOTO 99
         ENDIF
         IF (TYP2.EQ.'E') THEN
            JCN=NNAM2
            GOTO 99
         ENDIF
      ENDIF
      IF (V.EQ.'f') THEN
         IF (TYP1.EQ.'F') THEN
            JCN=NNAM1
            GOTO 99
         ENDIF
         IF (TYP2.EQ.'F') THEN
            JCN=NNAM2
            GOTO 99
         ENDIF
      ENDIF
         WRITE(8,101) VAR
100   FORMAT(/,' BOND NAMED: ',A8,' DOES NOT EXIST ON THE GRAPH')
101   FORMAT(/,2X,A10,' IS NOT AN EXCEPTABLE INPUT VARIABLE')
99    RETURN
      END
C
C>>>>>>>>>>>>>>>>>>>>>>>> NAME <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE NAME(TYPE,STRING)
C
C---DESCRIPTION: THE SUBROUTINE NAMES A BLOCK, AN EXTRA NODE OR AN
C                EXTRA BOND
C
C---DECLARATIONS:
      CHARACTER TYPE, STRING*8
      COMMON/BLGRPH/IBLCK,IEXTB,NEXT
C****************************
      IF (TYPE.EQ.'N') THEN
         STRING(1:4)='BLCK'
         IBLCK=IBLCK+1
         WRITE(STRING(5:6),'(I2.2)') IBLCK
      ENDIF
      IF (TYPE.EQ.'B') THEN
         STRING(1:4)='EXTB'
         IEXTB=IEXTB+1
         WRITE(STRING(5:6),'(I2.2)') IEXTB
      ENDIF
      IF (TYPE.EQ.'O') THEN
         STRING(1:4)='EXTN'
         NEXT=NEXT+1
         WRITE(STRING(5:6),'(I2.2)') NEXT
      ENDIF
      RETURN
      END
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> NAMNO <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE NAMNO(NAME,NB,NO)
C
C---DESCRIPTION: THE SUBROUTINE PROVIDES THE NODE/BOND NUMBER
C                (IN THE NODE/BOND LIST) FOR A GIVEN NAME
C
C---INPUTS: NAME - NODE/BOND NAME
C           NB - DEFINED THE NEEDED INFORMATION:
C                  NB='N'  -> NODE NUMBER REQUIRED
C                  NB='B'  -> BOND NUMBER REQUIRED
C
C---OUTPUT: NO - NODE/BOND NUMBER. IF NO=0, THE NODE/BOND IS
C                NOT ON THE LIST.
C
```

```
C---DECLARATIONS:
       CHARACTER*8 ELNAM(100), BDNAM(100), NAME, NB*1
       COMMON INELS, INBDS, ELNAM, BDNAM
C***************************
       NO=0
       IF (NB.EQ.'B') GOTO 2
       DO 10 I=1,INELS
          IF (ELNAM(I).EQ.NAME) NO=I
 10    CONTINUE
       IF (NB.EQ.'N') GOTO 99
 2     DO 20 I=1,INBDS
          IF (BDNAM(I).EQ.NAME) NO=I
 20    CONTINUE
 99    RETURN
       END
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>> PATHFN <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
       SUBROUTINE PATHFN(J0,PATH,K0)
C
C---DESCRIPTION: PATHFN (PATH-FiNd) finds the path from node #j0
C                back to the source node ,after all nodes has been
C                labeled. The path is defined by a list of node names.
C
C---INPUTS: j0 - "sink" node number
C           DFI - LABELING OF THE GRAPH (IN COMMON/A/)
C           FA - LIST OF "FATHER" FOR EACH NODE (IN COMMON/A/)
C
C---OUTPUTS: PATH - PATH DEFINED BY NODE NAMES (ARRAY)
C             K0 - NUMBER OF NODES IN THE PATH
C
C---DECLARATIONS:
       INTEGER IBDPTR(100), IELSBD(100,2), IBDSEL(100)
       INTEGER DFI(100),FA(100)
       CHARACTER*8 BDNAM(100), ELNAM(100), PATH(100),STACK(100)
       COMMON INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL,NBXTP
       COMMON/A/ STACK,IO,DFI,FA
C***************************
       I=1
       PATH(1)=ELNAM(J0)
       J=J0
       K0=1
 5     IF (DFI(J).EQ.1) GOTO 99
          I=I+1
          PATH(I)=ELNAM(FA(J))
          K0=K0+1
          J=FA(J)
       GOTO 5
 99    RETURN
       END
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> POPSTK <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
       SUBROUTINE POPSTK(NODE,NODE1,K0,BLOCK)
C
C---DESCRIPTION: THIS SUBROUTINE POPS AND OUTPUT THE STACK UP TO
C                AND INCLUDING "NODE". THE POPED STACK IS A BLOCK
C                AND IS THE OUTPUT OF THE SUBROUTINE. THE SUBROUTINE
C                ALSO REARRANGED THE REMAINING STACK.
C
```

```
C---INPUT: NODE - NODE NAME. THE LAST ONE IN THE DEFINED BLOCK
C          NODE1 - THE NAME OF THE "FATHER" OF "NODE"
C          STACK - OLD STACK (STORED IN COMMON/A/)
C          IO - # OF NODES IN OLD STACK (STORED IN COMMON/A/)
C
C---OUTPUT: BLOCK - LIST OF NODES THAT ARE IN THE SAME BLOCK
C           KO - # OF NODES IN THE BLOCK
C           STACK - UP TO DATE STACK (STORED IN COMMON/A/)
C           IO - # OF NODES IN UP TO DATE STACK
C
C---DECLARATIONS:
      CHARACTER*8 STACK(100),BLOCK(100),NODE,NODE1
      INTEGER DFI(100),FA(100)
      COMMON/A/ STACK,IO,DFI,FA
      COMMON/P/INDEX
C******************************
      K=0
      DO 10 I=1,IO
         IF (STACK(I).EQ.NODE) K=I
 10   CONTINUE
      IF (K.EQ.0) GOTO 99
      DO 20 I=1,IO-K+1
         BLOCK(I)=STACK(I+K-1)
 20   CONTINUE
      KO=IO-K+2
      BLOCK(KO)=NODE1
      IO=K-1
      IF (INDEX.EQ.1) CALL CUTPNT(KO,BLOCK)
      IF (INDEX.EQ.2) CALL BLCKGR(KO,BLOCK)
 99   RETURN
      END
C
CFILE:READVR-------------- READVR --------------------------------
C
C---PURPOSE: READS THE INPUT, OUTPUT AND FAULT-LIST VARIABLES.
C            COUNTS NUMBER OF INPUTS, OUTPUTS AND FAULT VARIABLES.
C
CFOFH:READVR-----------------------------------------------------
      SUBROUTINE READVR
      CHARACTER*10 STRING*80,H*10,UV(20),TV(20),FV(20)
      CHARACTER*8 U(20),T(20),F(20,2)
      CHARACTER*3 OPT(20)
      COMMON/INDEX/INDX
      COMMON/V/UV,TV,FV,L,M,N,U,T,F,OPT
C
      OPEN(15,FILE='VAR.DAT',STATUS='OLD')
      L=0
      M=0
      N=0
C
 10   READ(15,100) STRING
      IF (STRING(1:6).NE.'OPTION') GOTO 10
      READ(15,101) INDX
 1    READ(15,100) STRING
      H=STRING(1:10)
      IF (H.NE.'INPUT VARI') GOTO 1
 2    READ(15,100) STRING
      H=STRING(1:10)
      IF (H.EQ.'          ') GOTO 3
      L=L+1
      UV(L)=H
      GOTO 2
C
 3    READ(15,100) STRING
      H=STRING(1:10)
      IF (H.NE.'OUTPUT VAR') GOTO 3
 4    READ(15,100) STRING
      H=STRING(1:10)
      IF (H.EQ.'          ') GOTO 8
      M=M+1
      TV(M)=H
      GOTO 4
C
 8    READ(15,100) STRING
      H=STRING(1:10)
      IF (H.NE.'          ') GOTO 9
      INDX=11
      GOTO 7
C
 5    READ(15,100) STRING
      H=STRING(1:10)
 9    IF (H.NE.'FAULT VARI') GOTO 5
 6    READ(15,100) STRING
      H=STRING(1:10)
      IF (H.EQ.'          ') GOTO 7
      N=N+1
      FV(N)=H
      GOTO 6
C
 7    CLOSE(15)
 100  FORMAT(A80)
 101  FORMAT(I1)
      RETURN
      END
C
C>>>>>>>>>>>>>>>>>>>>>>>>>> RTNODE <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE RTNODE
C
C---DESCRIPTION: RTNODE (REDUCTION OF TEST-NODES) PROVIDES THE USER
C                INFORMATION ABOUT THE DETECTIBILITY OF EACH TEST-NODE.
C                A DESIGN TOOL FOR CHOOSING TEST-NODES IS ALSO PROVIDED.
C
C---INPUTS: D - CFD ARRAY (STORED IN COMMON/DD/)
C
C---OUTPUTS: 1. LIST OF UNNECESARY TEST-NODES
C            2. TEST-NODE EQUIVALENT SETS
C            3. DETECTIBILITY OF TEST-NODES (TWO TABLES)
C
C---DECLARATIONS:
      CHARACTER*10 UV(20),TV(20),FV(20)
      CHARACTER*8 U(20),T(20),F(20,2)
      CHARACTER*3 OPT(20)
      INTEGER D(20,20,20)
      COMMON/DD/D
      COMMON/V/UV,TV,FV,L,M,N,U,T,F,OPT
C*************************
      WRITE(8,100)
      WRITE(8,105) N
      WRITE(8,106) M
      WRITE(8,114)
      WRITE(8,115)
      WRITE(8,101)
      WRITE(8,102)
      WRITE(8,103)
      WRITE(8,101)
      DO 10 J=1,M
         CALL CHANGE(D,J,1)
         CALL FAULT(D,1,ND,NI,0)
         WRITE(8,104) J,TV(1),ND,NI
         CALL CHANGE(D,J,1)
10    CONTINUE
      WRITE(8,101)
      WRITE(8,110)
      WRITE(8,101)
      WRITE(8,107)
      WRITE(8,108)
      WRITE(8,101)
      DO 40 J0=0,M-1
         NDMAX=0
         DO 50 J=J0+1,M
            CALL CHANGE(D,J,J0+1)
            CALL FAULT(D,J0+1,ND,NI,0)
            IF (ND.GT.NDMAX) THEN
               NDMAX=ND
               NI0=NI
            ELSE
               CALL CHANGE(D,J,J0+1)
            ENDIF
50       CONTINUE
         WRITE(8,109) J0+1,TV(J0+1),NDMAX,NI0
40    CONTINUE
      WRITE(8,101)
      WRITE(8,111)
      WRITE(8,112)
      WRITE(8,113)
100   FORMAT(///,3X,'DETECTIBILITY OF TEST-NODES',/,3X,27('*'))
101   FORMAT(/,1X,75('_'))
102   FORMAT(/,2X,2(4X,'TEST-NODE'),1X,2(7X,'NUMBER OF '),'UNIQUELY')
103   FORMAT(6X,'NUMBER',7X,'VARIABLE  ',2(6X,'DETECTED FAULTS'))
104   FORMAT(/,7X,I2,12X,A6,13X,I2,17X,I2)
105   FORMAT(/,'    -TOTAL NUMBER OF FAULTS: ',I2)
106   FORMAT(/,'    - NUMBER OF EFFECTIVE TEST-NODES: ',I2)
107   FORMAT(/,3X,'NUMBER OF',8X,'INCLUDING ',2(7X,'TOTAL NUMBER OF'))
108   FORMAT(6X,'T.N.',8X,'T.N. VARIABLES',11X,'D.F.',17X,'U.T.F.')
109   FORMAT(/,7X,I2,13X,A6,14X,I2,18X,I2)
110   FORMAT(////,3X,'REDUCTION OF TEST-NODES',/,3X,23('*'))
111   FORMAT(/,3X,'ABBREVIATIONS:',/,4X,'T.N.    - TEST NODES')
112   FORMAT(4X,'D.F.    - DETECTED FAULTS')
113   FORMAT(4X,'U.D.F. - UNIQUELY DETECTED FAULTS')
114   FORMAT('   *UNNECESSARY TEST-NODES WERE DELETED FROM THE LIST')
115   FORMAT('   *T.N. EQUIVALENCE SETS ARE REPRESENTED BY ONE T.N.')
99    RETURN
      END
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> SSTACK <<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE SSTACK(NODE)
C
C---DESCRIPTION: THIS SUBROUTINE ADDS NODES TO THE TOP OF THE STACK.
```

```
C               ONLY IF THIS NODE DOES NOT EXIST THERE ALREADY.
C
C---INPUT: NODE - NODE NAME TO BE ADDED TO THE STACK
C          STACK - OLD STACK (STORED IN COMMON/A/)
C
C---OUTPUT: STACK - UP-TO-DATE STACK (STORED IN COMMON/A/)
C           IO - # OF NODES IN THE UP TO DATE STACK (STORED IN COMMON/A/)
C
C---DECLARATIONS:
        CHARACTER*8 STACK(100),NODE
        INTEGER DFI(100),FA(100)
        COMMON INELS,INBDS
        COMMON/A/ STACK,IO,DFI,FA
C*****************************
        INDEX=0
        IF (IO.EQ.0) GOTO 5
        DO 10 I=1,IO
           IF (STACK(I).EQ.NODE) INDEX=1
 10     CONTINUE
 5      IF (INDEX.EQ.0) THEN
           IO=IO+1
           STACK(IO)=NODE
        ENDIF
 99     RETURN
        END
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> TESTNJ <<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
        SUBROUTINE TESTNJ(VAR,JCN)
C
C---DESCRIPTION: THIS SUBROUTINE DEFINES THE TEST-NODE (OUTPUT)
C                JUNCTION FOR A GIVEN VARIABLE.
C
C---INPUT: VAR - VARIABLE NAME, ONLY FLOW OR EFFORT ALLOWED, WRITTEN
C                AS: e.BOND-NAME OR f.BOND-NAME
C
C---OUTPUT: JCN - THE OUTPUT JUNCTION NAME (0-JUNCTION OR 1-JUNCTION)
C
C---DECLARATIONS:
        CHARACTER*10 VAR
        CHARACTER*8 ELNAM(100),BDNAM(100),BOND,NNAM1,NNAM2,JCN
        CHARACTER*4 NBXTP(100)
        CHARACTER TYP1,TYP2,V
        INTEGER IELSBD(100,2),IBDPTR(100),IBDSEL(100)
        COMMON INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL,NBXTP
C****************************
        JCN='        '
        V=VAR(1:1)
        BOND=VAR(3:10)
        CALL NAMNO(BOND,'B',I)
        IF (I.EQ.0) THEN
           WRITE (8,100) BOND
           GOTO 99
        ENDIF
        IF (V.EQ.'e'.OR.V.EQ.'f') GOTO 1
        WRITE (8,105) VAR
        GOTO 99
  1     CALL CON(BOND,NNAM1,NNAM2,TYP1,TYP2)
        IF (V.EQ.'e') THEN
           IF (TYP1.EQ.'0') THEN
```

```
               JCN=NNAM1
               GOTO 99
            ENDIF
            IF (TYP2.EQ.'0') THEN
               JCN=NNAM2
               GOTO 99
            ENDIF
         ENDIF
         IF (V.EQ.'f') THEN
            IF (TYP1.EQ.'1') THEN
               JCN=NNAM1
               GOTO 99
            ENDIF
            IF (TYP2.EQ.'1') THEN
               JCN=NNAM2
               GOTO 99
            ENDIF
         ENDIF
C
         CALL ADNODE(VAR,JCN)
C
 100     FORMAT(/,' BOND NAMED: ',A8, 'DOES NOT EXIST ON THE GRAPH')
 105     FORMAT(/,2X,A10,' IS NOT AN EXCEPTABLE OUTPUT VARIABLE')
 101     FORMAT(/,' THERE IS NO 0-JUNCTION ASSOCIATED WITH ',A10)
 102     FORMAT(' AN EXTRA 0-JUNCTION WAS ADDED BETWEEN ',A8,' AND ',A8)
 106     FORMAT(' JUNCTION NAME:',A8)
 103     FORMAT(/,' THERE IS NO 1-JUNCTION ASSOCIATED WITH ',A10)
 104     FORMAT(' AN EXTRA 1-JUNCTION WAS ADDED BETWEEN ',A8,' AND ',A8)
 107     FORMAT(' JUNCTION NAME:',A8)
 99      RETURN
         END
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>> TNODE <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
         SUBROUTINE TNODE
C
C---DESCRIPTION: THE SUBROUTINE PROVIDES INFORMATION ABOUT THE
C                DETECTABILITY OF THE INITIAL TEST-NODES LIST.
C
C---INPUTS: D - CFD ARRAY, STORED IN COMMON/DD/
C
C---OUTPUTS: 1. LIST OF UNNECESSARY TEST NODES
C            2. EQUIVALENT SETS OF TEST NODES
C
C---DECLARATIONS:
         CHARACTER*10 UV(20),TV(20),FV(20)
         CHARACTER*8 U(20),T(20),F(20,2)
         CHARACTER*3 OPT(20)
         INTEGER D(20,20,20),DD(20,20),AT(100)
         COMMON/DD/D
         COMMON/V/UV,TV,FV,L,M,N,U,T,F,OPT
C***************************
         WRITE(8,100)
         ISET=1
         DO 90 J=1,M
            AT(J)=0
 90      CONTINUE
 1       DO 10 JO=1,M
            IF (AT(JO).NE.0) GOTO 10
            IN=0
```

```
          DO 20 I=1,L
             DO 20 K=1,N
                IF (D(I,J0,K).EQ.0) GOTO 2
 20       CONTINUE
          AT(J0)=111
          GOTO 10
 2        DO 30 J=J0+1,M
             IF (AT(J).NE.0) GOTO 30
             DO 40 I=1,L
                DO 40 K=1,N
                   DD(I,K)=D(I,J,K)-D(I,J0,K)
                   IF (ABS(DD(I,K)).EQ.1) GOTO 30
 40          CONTINUE
             IN=1
             AT(J0)=ISET
             AT(J)=ISET
 30       CONTINUE
          IF (IN.EQ.1) ISET=ISET+1
 10    CONTINUE
C
       IN=0
       DO 50 J0=1,M
          IF (AT(J0).EQ.0) AT(J0)=999
          IF (AT(J0).EQ.111) THEN
             IN=IN+1
             IF(IN.EQ.1) WRITE(8,101)
             WRITE(8,104) J0,TV(J0)
          ENDIF
 50    CONTINUE
C
       IN=0
       DO 80 J0=1,M
          IF (AT(J0).LT.100) THEN
             IN=IN+1
             IF (IN.EQ.1) WRITE(8,102)
             WRITE(8,103) AT(J0)
             WRITE(8,104) J0,TV(J0)
             DO 70 J=J0+1,M
                IF (AT(J).EQ.AT(J0)) THEN
                   WRITE(8,104) J,TV(J)
                   AT(J)=222
                ENDIF
 70          CONTINUE
          ENDIF
 80    CONTINUE
C
 96    DO 95 J0=1,M
       IF (AT(J0).EQ.111.OR.AT(J0).EQ.222) THEN
          DO 88 J=J0,M-1
             T(J)=T(J+1)
             TV(J)=TV(J+1)
             AT(J)=AT(J+1)
             DO 88 I=1,L
                DO 88 K=1,N
                   D(I,J,K)=D(I,J+1,K)
 88       CONTINUE
          M=M-1
          GOTO 96
       ENDIF
 95    CONTINUE
 100     FORMAT(///,3X,'TEST-NODES REPORT',/,3X,17('*'))
 101     FORMAT(//,6X,'LIST OF UNNECESSARY TEST-NODES',/,6X,30('*'))
 102     FORMAT(//,6X,'TEST-NODES EQUIVALENCE SETS',/,6X,27('*'))
 103     FORMAT(/,9X,'TEST-NODE EQUIVALENCE SET #',I2,' CONTAINS:')
 104     FORMAT(20X,'TEST-NODE #',I2,' (DEFINED BY ',A6,')')
 99      RETURN
         END
C
C>>>>>>>>>>>>>>>>>>>>>>>>> TREDIC <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
         SUBROUTINE TREDIC
C
C---DESCRIPTION: THE SUBROUTINE FINDS THE FAULT DICTIONARY (STORED
C                IN ARRAY D) FOR A TREE GRAPH. THE PROCEDURE CONTAINS
C                THE FOLLOWING STEPS:
C                 1. LABEL THE GRAPH FOR EACH INPUT USING THE DFS
C                    ALGORITHM
C                 2. FIND THE PATH FOR EACH INPUT-OUTPUT SET
C                 3. CHECK WHETHER THE FAULTY JUNCTIONS (ONE BY ONE)
C                    EXIST ON THAT PATH AND PUT THE RELEVANT RESULTS
C                    IN THE FAULT DICTIONARY ARRAY, D.
C
C---INPUTS: SYSTEM DESCRIPTION (STORED IN COMMON).
C           INPUT, OUTPUT AND FAULTY JUNCTIONS (STORED IN COMMON/V/)
C
C---OUTPUT: D - CFD ARRAY (STORED IN COMMON/DD/)
C
C---DECLARATIONS:
         CHARACTER*10 UV(20),TV(20),FV(20)
         CHARACTER*8 ELNAM(100),BDNAM(100),PATH(100),STACK(100)
         CHARACTER*8 U(20),T(20),F(20,2)
         CHARACTER*4 NBXTP(100),BDTP(100)
         CHARACTER*3 OPT(20)
         INTEGER IELSBD(100,2),IBDPTR(100),IBDSEL(100)
         INTEGER D(20,20,20),DFI(100),FA(100)
         COMMON INELS,INBDS,ELNAM,BDNAM,IBDPTR,IELSBD,IBDSEL,NBXTP,BDTP
         COMMON/A/STACK,IO,DFI,FA
         COMMON/DD/D
         COMMON/INDEX/ INDX
         COMMON/V/ UV,TV,FV,L,M,N,U,T,F,OPT
C***************************
         DO 40 I=1,L
            CALL DFSLBL(U(I))
            DO 50 J=1,M
               CALL NAMNO(T(J),'N',JO)
               CALL PATHFN(JO,PATH,KO)
                DO 60 K=1,N
                  CALL FLTEXT(PATH,KO,F(K,1),F(K,2),OPT(K),D(I,J,K))
 60             CONTINUE
 50         CONTINUE
 40      CONTINUE
         IF (INDX.GE.1) GOTO 99
         DO 70 K=1,N
            WRITE (8,105) FV(K)
            WRITE (8,106) D(1,1,K),D(1,2,K)
            WRITE (8,106) D(2,1,K),D(2,2,K)
 70      CONTINUE
 105     FORMAT(/,'    FOR FAULT DEFINED BY ',A5,'=0:')
 106     FORMAT(38X,2(I3))
 99      RETURN
         END
```

APPENDIX F: Input files for five-degree-of-freedom suspension system

F.1: System description

```
HEADING

  FILE
    CAR1.ENP

  TITLE
     CAR1.ENP: FIVE-DEGREE-OF-FREEDOM VEHICLE MODEL

SYSTEM GRAPH DESCRIPTION

  NODE        TYPE     XLOC       YLOC
    SF1       MFG       300.       400.
    SF2       MFG       500.       400.
    SE1       MEG       500.       300.
    SE4       MEG       100.      -100.
    SE5       MEG       100.       100.
    0F1       M0G       700.       500.
    1F1       M1G       700.       400.
    RC2       MRG       700.       600.
    CK2       MCG       500.       600.
    0B1       M0G       800.       500.
    1B1       M1G       900.       500.
    1M1       M1G       800.         0.
    RC3       MRG       150.       600.
    CK3       MCG         0.       200.
    TFF       MTG      -200.       100.
    1R        M1G        50.      -100.
    IJ2       MIG       400.      -200.
    TBF       MTG       100.      -100.
    1T        M1G       -50.         0.
    IM2       MIG       800.      -200.
    0D        M0G      -800.      -300.
    1D        M1G      -550.      -300.
    RC1       MRG         0.         0.
    CK1       MCG       250.       300.
    IM1       MIG       100.       100.
    TFD       MTG       200.       400.
    1Z4       M1G       100.       200.
    IM4       MIG         0.        50.
    1Z5       M1G       400.       600.
    IM5       MIG       200.       100.
    0F        M0G       300.       300.
    1F        M1G       200.      -400.
    CK4       MCG       -50.      -600.
    RC4       MRG      -900.      -900.
    0B        M0G       200.       750.
    1B        M1G       700.       700.
    CK5       MCG       500.      -600.
    RC5       MRG       800.       600.
```

CONNECTOR	TYPE	FROM	TO	VERTICES
1	BM	OF	1Z4	
2	BM	OB	1Z5	
3	BM	1Z4	OF1	
4	BM	1Z5	OB1	
5	BM	OF1	1F1	
6	BM	OB1	1B1	
7	BM	OF1	TFF	
8	BR	TFF	1R	
9	BR	1R	TBF	
10	BM	TBF	OB1	
11	BM	OF1	1T	
12	BR	1R	TFD	
13	BM	TFD	OD	
14	BM	1T	OD	
15	BM	OB1	1T	
16	BM	OD	1D	
17	BM	OD	1M1	
s1	BM	SF1	OF	
s2	BM	SF2	OB	
f	BR	OF	1F	
ff	BM	OB	1B	
k4	BM	1F	CK4	
c4	BM	1F	RC4	
k5	BM	1B	CK5	
c5	BM	1B	RC5	
m4	BM	1Z4	IM4	
m5	BM	1Z5	IM5	
k2	BM	1F1	CK2	
c2	BM	1F1	RC2	
k3	BM	1B1	CK3	
c3	BM	1B1	RC3	
j2	BR	1R	IJ2	
m2	BM	1T	IM2	
ii	BM	1M1	IM1	
k1	BM	1D	CK1	
c1	BM	1D	RC1	
mg1	BM	SE1	1M1	
mg4	BM	SE4	1Z4	
mg5	BM	SE5	1Z5	

F.2: Variable-list for option 1

```
OPTION
1

INPUT VARIABLES
f.s1
f.s2

OUTPUT VARIABLES
e.m4
e.m2
e.ii
e.k2

FAULT VARIABLES
f.c2
e.3
e.k2
e.2
f.k5
```

Variable-list for option 1 (default option)

```
OPTION
1

INPUT VARIABLES
f.s1

OUTPUT VARIABLES
e.j2
e.m2
e.ii
```

F.3: Variable-list for option 2

```
OPTION
2

INPUT VARIABLES
f.s1

OUTPUT VARIABLES
e.j2
e.m2
e.ii
f.k2
e.m5
e.k2
f.k3
e.c2
e.c4

FAULT VARIABLES
f.5
e.6
e.11
e.16
f.4
e.k2
f.c2
f.3
f.15
f.12
```