# CAT: A Computer Code for the Automated Construction of Fault Trees

NP-705
Research Project 297-1

Interim Report, March 1978

Prepared by

School of Engineering and Applied Science
UNIVERSITY OF CALIFORNIA
Chemical, Nuclear and Thermal Engineering Department
Los Angeles, California 90024

Principal Investigators
G. E. Apostolakis
S. L. Salem
J. S. Wu



Prepared for

Electric Power Research Institute
3412 Hillview Avenue
Palo Alto, California 94304

EPRI Project Manager
Boyer B. Chu
Nuclear Power Division

---

LEGAL NOTICE

# FOREWORD

Recent attempts have been made to develop an automated algorithm for constructing the logic models of fault trees. In a previous report, NP-288, a decision table method was introduced and applied to model component behavior. This report is an extension of report NP-288. It presents the application aspect of the decision table method for fault tree construction. Several nuclear subsystems are analyzed to demonstrate various usages of the CAT computer code. The report is also intended to serve as a CAT Code Users Manual; the code may be obtained from Electric Power Software Center.

System reliability analysis has been increasingly recognized as an integral part of design safety evaluations for nuclear power generation plants. Fault and event tree analysis has been extensively applied to quantify the systems and subsystems reliability both by the industry and regulatory agencies. The analysis generally involves the construction and evaluation of system logic models which describe various interconnections among components and their operation requirements. Several computer codes have been developed for the numerical evaluation of a given logic model. Model construction has still remained a manual task which usually contributes the bulk of time to probabilistic system analysis. The objectives of this automated fault tree construction could perhaps speed up the entire reliability analysis process. Several other concepts have been examined for computerized fault tree construction; it appears that the CAT code approach could provide a more adequate modeling capability to nuclear systems and subsystems.

Boyer B. Chu
Project Manager
Nuclear Safety & Analysis Dept.

## ABSTRACT

This report presents a computer code, CAT (Computer Automated Tree), which applies decision table methods to model the components behavior for systematic construction of fault trees. The decision tables for some commonly encountered mechanical and electrical components are developed; two nuclear subsystems, a Containment Spray Recirculation System and a Consequence Limiting Central Systems, are analyzed to demonstrate the applications of CAT code.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

## SUMMARY

The CAT (Computer Automated Tree) methodology is a systematic approach to the construction of fault trees, based upon the use of decision tables. This approach consists of a scheme which utilizes these decision tables to model component behavior, a method of describing the specific system configuration including initial system states, and a means of defining a top event (or events) of interest. Given these inputs, the decision table models are used for the appropriate components within the system, and are combined and edited to form a completed fault tree for the TOP event desired. This fault tree may then be analyzed, either by hand, or by any of several fault tree analysis codes, in order to obtain the desired reliability (availability) information for the TOP event.

The current approach has several important features which make it especially useful in the analysis of nuclear systems, as well as of other, general types of systems. The decision table methodology is capable of modeling complex components of essentially any type, including mechanical, electrical and hydraulic. It can incorporate models for human interactions, environmental influences and provides a number of ways of treating common cause effects. Furthermore, the specific approach developed here also allows the analysis of systems containing feedback loops, such as may be found in many types of control circuits in use in nuclear plants.

This report documents a slightly newer version of the CAT code than presented by Salem, Apostolakis and Okrent in EPRI NP-288. This version has been updated by the incorporation of a new subroutine

('OUTPUT'), which produces the fault tree in a punched output format directly compatible with the PREP-KITT codes. This output will provide the complete data deck required as input for the PREP code, if desired. However, the input requirements for this version are identical with those of the previous one if the new output is not desired. Thus, the same data decks may be utilized with either version of the CAT code. If the new output option is to be utilized, however, additional failure and repair data must be provided.

EPRI NP-288 has presented the CAT methodology from a somewhat theoretical point of view. In the following chapters, those aspects most important to the user of the code itself will be emphasized. Actual details of the code itself will be included in the appendices.

In order to provide a general introduction to the CAT methodology, Chapter 2 begins by presenting some basic concepts and terminology used in this approach. The general organization of the input deck, and the specific requirements for all input data are then presented. Finally, the output produced by the code is briefly described.

Chapter 3 discusses the methods of developing the decision tables to be used by the code. Each method is illustrated by an example in order to familiarize the user with some of the techniques useful in decision table modeling. Also, the way in which such decision tables are used by the code in constructing fault trees is described. In combination with Chapter 2, this chapter completes the discussion of input required for operating the CAT code.

In Chapter 4, the use of the CAT code is illustrated by two applications: a Containment Spray Recirculation System, and a

Consequence Limiting Control System. These systems have been chosen to provide additional examples, complementing those of References [1,2], and will help to display various features of the code. New features illustrated by these examples include maintenance, TOP event logic which is a function of time, and the use of inhibit conditions.

Finally, Chapter 5 summarizes these results and provides a few concluding suggestions for the user.

This report is supplemented by several appendices, useful in running the CAT code. Appendix A describes the functional organization of CAT, and its specific subroutines. Appendix B provides a number of decision tables, many of them developed during the past year, as examples to guide the user in constructing his own tables. Appendix C presents a simple example system which, together with the sample input and output of Appendices D and E, provides a complete sample case which can be used to check out the operation of the code and to obtain familiarity with it. Finally, Appendix F contains a complete listing of the updated version of CAT for those interested in becoming more familiar with the actual mechanics of the code.

# 1. INTRODUCTION

With the increasingly important role being played by probabilistic risk assessment in the electric power industry, and the interest being expressed both by commercial groups and regulatory bodies, the development and improvement of methods of safety and reliability analysis are currently of great interest in the nuclear field. Fault tree analysis (FTA) is one of these techniques which is especially useful because of its advanced state of development, and acceptance and use by large segments of the technical community. The usefulness of FTA has been greatly augmented by the development and application of a number of computer codes for the numerical analysis of fault trees. However, the rapid analysis of a complex system using such codes has, until recently, had to await the much slower process of constructing the fault tree itself. This construction phase has traditionally been a manual task, relegated to the analyst himself, due to the lack of a systematic fault tree methodology which could be readily programmed on the computer.

In light of the relatively advanced state of fault tree analysis codes, the current effort was devoted to automating this one remaining phase. The benefits of this automation would include the speeding up of the entire fault tree analysis process as well as the freeing of much of the analyst's time for other tasks. Furthermore, an automated fault tree construction code would allow the preparation of additional fault trees for investigating other events of interest, or for evaluating various system modifications. The development and implementation of such an approach has been the goal of this work. The methodology has been described elsewhere [1,2]; the operation and use of the CAT

code, which implements this methodology, will be presented in detail in the following chapters of this report.

In order to see some of the other significant characteristics of such a methodology, a few of the features incorporated into the CAT code itself will be enumerated here. These include:

1) a simple, tabular decision table form for modeling component behavior,
2) capability of multiple state and multiple failure mode decision table models,
3) ability to develop both simple and highly detailed component models to produce fault trees of varying complexity,
4) ease of defining multiple TOP events for the construction of both fault and success trees for various events,
5) provision to define TOP events of any logical complexity,
6) ability to define initial states to specify initial system configurations,
7) simplicity of changing or modifying systems to analyze differences between various configurations,
8) essentially unlimited numbers of components, complexities of systems or sizes of fault trees produced,
9) specification of various levels of editing, and printout,
10) capability of interfacing with the PREP-KITT analysis codes,
11) repeatability of fault trees produced for identical input, and
12) ability to "pre-structure" fault trees by varying TOP event definitions and component models.

This report documents a slightly newer version of the CAT code than

presented by Salem, Apostolakis and Okrent [1,2]. This version has been updated by the incorporation of a new subroutine ('OUTPUT'), which produces the fault tree in a punched output format directly compatible with the PREP-KITT codes. This output will provide the complete data deck required as input for the PREP code, if desired. However, the input requirements for this version are identical with those of the previous one if the new output is not desired. Thus, the same data decks may be utilized with either version of the CAT code. If the new output option is to be utilized, however, additional failure and repair data must be provided.

Internally the only changes in the code have been to incorporate a call to the new subroutine, add input and error-checking sections for the new data, delete one array which was no longer needed, and change one variable in common.

In addition to a discussion of the fundamentals of the CAT methodology, this report presents sufficient material to allow the reader to understand and operate the code itself. Thus, Chapter 4 includes two examples of actual reactor systems analyzed by the code, and several appendices have been added to further help the potential user. These appendices contain a discussion of the code itself, with flowcharts, a library of sample decision table models, a complete example case with input and output listings, and a listing of the CAT code itself.

## 2. BASIC OPERATION OF THE CAT CODE

The CAT code has been developed as a systematic method of fault tree construction utilizing the methodology of decision table modeling [1,2]. Sufficient detail has been presented to allow the user to understand the basics of decision table modeling, and to construct fault trees by hand identical to those produced by the CAT code. This chapter will briefly review some essential terminology, and will discuss, in detail, the data required to actually operate CAT. A further discussion of specific subroutines and their functions will be found in Appendix A of this report. Furthermore, examples of input and output will be found in Appendices D and E.

### 2.1 Terminology and Basic Concepts

A number of definitions and fundamental ideas will first be presented. More detailed discussions will be found elsewhere [1,2]. The basic function of the code is to analyze a system, starting from a specific initial state, and to produce a fault tree for the TOP event of interest. This system can be any collection of units, known as "components", which performs some certain defined function or functions under specified conditions. These functions will be defined as the output of the system, and the specified conditions the inputs. Although the systems used here will generally be mechanical in nature, others, such as economic systems, corporate organizations, etc., are all equally valid.

Each system is first broken down into a set of components, which are the smallest elements to be individually modeled. With the CAT methodology, a component may actually consist of an entire system or

subsystem, as long as the features of interest can be accurately modeled by a single decision table. For a complex component or system, however, a number of coupled decision tables may be desirable.

Since there are often a number of similar or identical components within a single system, it is advantageous to employ a single model to describe the behavior of all such similar components. This introduces the concept of a component type. A component type is a unique description of the operational and failure modes of a component, represented by a single decision table. Any components which can be described by a single model will be considered as a single type, even though they may represent different physical components; thus, simple models for a fuse, resistor and circuit breaker might be represented by the same decision table. On the other hand, similar components with different characteristics (e.g., valves with different modes of operation), would be considered different types, each with a unique decision table representation. In either case, the models should be independent of the system itself in order to allow the development of libraries of component types to be used in systems of general interest.

Given a set of components, modeled by various component types, a system is then described in terms of the interconnections between these components. Each of these connections shall be defined as a node, or any point at which an output from one component is connected to the inputs of succeeding components. Note that one output may be connected to any number of succeeding inputs at a node; however, only a single output may exist at any one node. If outputs are to be connected together in parallel, they should be connected to the inputs

of a gate (junction), modeled by a decision table which represents the logical state formed by various combinations of the outputs to be connected.

At each node of a system, a system state may be defined to describe system conditions or signal states at that point. This state may have been set as an initial condition, or defined by later fault tree development, and must be consistent with all component inputs and outputs connected to that node. That is, this state must satisfy these two conditions:

1) it must exist as a valid output state in the decision table for the component output connected to that node, and
2) it must exist as a valid input state, or be allowed by don't care entries, in all decision tables of components whose inputs are connected to that node.

Table 1 contains a number of system states as examples of typical input, output and system states. These will be used in many of the models developed in Appendix B.

Table 1. REPRESENTATIVE SYSTEM STATES

| STATE | DEFINITION |
|---|---|
| -1 | "Don't care" (signal state irrelevant or undefined) |
| 0 | no signal, or signal too low |
| 1 | normal signal |
| 2 | overload (signal too high) |
| 3 | low signal (used if separate states are desired for "no" and "low" signals) |
| 101 | ground (zero) or short to ground |
| 102 | floating (open, undefined) |

An internal mode represents an internal function of a component, and is represented by an "internal" column in a decision table. Since there may be several internal functions, or sub-components, within a complex component, a decision table may have several internal columns, each labeled by a specific sub-component function. For example, the motor-operated valve in Appendix B has the internal modes "position", "mechanical", "slip-clutch" and "relay". For each internal mode, or column, a component state may be defined to describe the internal state. The simplest states are simply "good" and "failed". However, since the mode of failure may determine the output state, it is often necessary to provide several failure states, such as "failed open" and "failed closed". Finally, it is important to consider all combinations of failure states of the various failure modes (columns) of a multiple-column component.

In order to utilize consistent decision table models, Table 2 was developed to systematically categorize failure states. This numbering scheme was then used to define a representative number of basic failure states, as shown in Table 3. Although use of such schemes is highly recommended, any similar categorization may be employed by the user.

Boundary conditions may now be described as system or component states which have been predefined as existing or not existing "initial" or "boundary" conditions within the system. These states exist initially, and continue to exist throughout the construction of the fault tree. In general, they will be used to determine the initial system configuration or operating conditions, and may be used to

TABLE 2. GENERAL FAILURE STATE CATEGORIES

| STATE | DEFINITION |
|---|---|
| -1 | don't care (internal mode irrelevant) |
| 0 | good |
| 1-1000 | general faults |
| 1001-2000 | electrical (shorts, surges, etc.) |
| 2001-3000 | mechanical |
| 3001-4000 | fluid (leak, rupture, plugged, etc.) |
| 4001-5000 | electronic (logic errors, etc.) |
| 5001-6000 | human |
| 6001-7000 | environmental (temperature, pressure, stress, etc.) |

TABLE 3. DECISION TABLE FAILURE STATES

| STATE | DEFINITION |
|---|---|
| 1 | failed open (fails to close; fails to transmit signal) |
| 2 | fails closed (shorted; fails to open; welded shut) |
| 3 | internal failure (general, undefined) |
| 4 | fails to start (fails to actuate or change position) |
| 5 | fails to operate properly (fails during operation) |
| 6 | operates prematurely (starts without signal to start) |
| 1001 | short to ground |
| 1002 | short to power |
| 1003 | power surge (power supply failure mode) |
| 3001 | leak |
| 3002 | rupture |
| 3003 | plugged |
| 3004 | stuck |
| 5001 | calibration error |
| 5002 | design error |
| 5003 | general operator error |

specify certain components as being failed at the start. Furthermore, they may be used to qualify the TOP event description.

Finally, the TOP event is that event which defines the failed (or successful) state of the system for which the fault tree is to be constructed. The basic requirement for the TOP event is that it be definable in terms of system states at specific nodes, which then serve as starting points for the fault tree construction process. Furthermore, if more than one event (or state) is used in the TOP event definition, these must have some logical relationship to each other in order to structure the tree beneath. For example, the TOP event "failure of systems A and B" might be represented by simultaneous states of zero at two nodes, logically connected by an AND gate. The resulting fault tree would then have a top AND gate connecting these two events.

## 2.2 Input Data

### 2.2.1 Outline of Input Data Organization

The input to the CAT code consists of the following information:

1) Program Control Data,
2) Decision Table Models,
3) System Configuration,
4) TOP Event definition,
5) Initial or Boundary Conditions,
6) Failure and repair data (for use with PREP-KITT output option).

The first of these inputs, as described below, consists of program dimensions used to define the sizes of the component library and system

configuration, and flags to control the printing and editing options to be used. The decision table models comprise the bulk of the input, and will be described in detail in Chapter 3. The system configuration, initial conditions and TOP event definition will all be described later in this chapter.

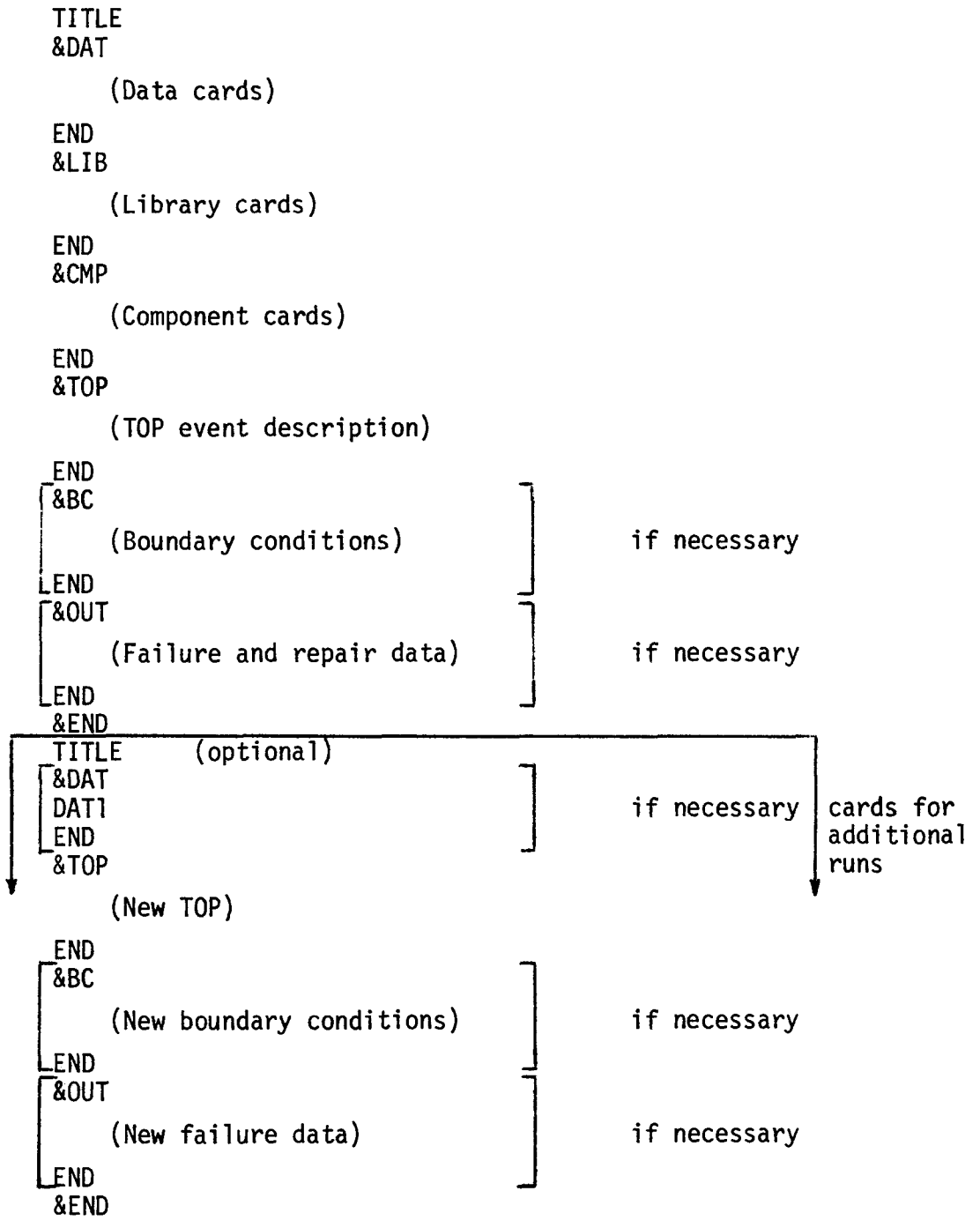
Output from the code consists of two parts. First is the printed output of all input data, cross-references and the fault tree itself. Secondly, if desired, is a punched deck (or tape or disk file) consisting of the fault tree and failure data in a format compatible with the PREP-KITT codes. This is produced by a separate subroutine, 'OUTPUT', which can be modified or replaced to interface with any code of the user's option. This routine will be discussed later in this chapter, and in the appendices.

### 2.2.2 Data Deck Setup and General Input Considerations

The input data deck is arranged as shown in Table 4. Each set of data will be described individually; general information is provided below.

Each card (except the failure and repair data) begins with a four character code which identifies to the computer what type of information is to follow. This also facilitates later runs in which the input must be modified on certain cards. On all except the 'DAT' cards, the four character field can be followed by up to six columns of information. Thus, the 'ROW' cards can be numbered in column 5, the 'LIBR' codes can be followed by the appropriate type number, etc. (see sample input data). This information is for the user's convenience and is not read by the code. It is important to note that, on

TABLE 4. INPUT DATA DECK FOR CAT



certain cards, the four column code begins with a blank column.

The major sections begin with '&---' cards and terminate with 'END' cards. These codes allow the program to search for the next set of data if terminal errors occur in any section. The '&DAT' section contains the basic parameters for the job. This consists of four cards, format (A4, I1, 12I5). However, cards 'DAT1' and 'DAT4' may be omitted if default values of all parameters so skipped are desired.

The second section, beginning with the '&LIB' card, contains the library of component type decision tables. Each table begins with a 'LIBR' card which contains the basic information for that table. There follows a 'MOD' card which lists the names of that component's internal failure mechanisms. The decision table itself is input on a set of 'ROW' cards, concluded by an 'END' card.

The components themselves are described by the '&CMP' section, one card per component. On them are given the component name, type number and input/output node numbers.

The '&TOP' segment begins with a 'TTOP' card defining the size of the TOP event decision table and the node numbers referring to the columns of the table. This is followed by the 'TOP' cards, on which the rows of the table are input.

If any boundary conditions are to be included, they follow the '&BC' card. Both internal component modes and system nodes may be initialized by use of cards coded 'INT' and 'EXT' respectively.

Should fault tree output be desired for use by the PREP-KITT codes, a section labeled '&OUT' follows. The first two cards contain

control data to be used by PREP, followed by failure and repair data. Since the input formats are similar to those used by PREP, the initial four letter codes are not used on these cards.

The last group of input cards must be followed by both its own 'END' card and a final '&END' card. These signify the end of the data section and the end of the job. Following the '&END' card, second and succeeding jobs may begin. Although each job of a multiple run uses the same library and system description, new boundary conditions and parameters from the 'DAT1' card may be defined, as well as the required redefinition of the TOP event. Furthermore, new failure and repair data must be provided, if necessary.

### 2.2.3 Program Control Data

The CAT input deck begins with a title card (20A4), followed by the program control data, in the '&DAT' section. This data section consists of four sets of program flags and dimensioning information, and is contained on cards labled ' DAT1' - ' DAT4'. These cards have the formats (A4, I1, 12I5), and the four character code field 'DATn' must be preceeded by a blank. This group of cards must be preceeded by an '&DAT' card and followed by an 'END ' card. The data on these cards are as follows:

```
 Title Card
&DAT
 DAT1      IJOB     IPRINT     KOUT     IEDIT      IOT
 DAT2      NLIB      LNROW   MAXINT    MXNROW
 DAT3     NNCMP      NNODE
 DAT4      MROW
END
```

The parameters on these cards are described below. Note that the parameters on cards 'DAT1' and 'DAT4' have default values. In the

event that all parameters on the 'DAT1' card are to be set to the default values, the 'DAT1' card may be omitted. Similarly, if MROW = 1, the 'DAT4' card may be eliminated. However, if the DAT1 card is used, all values must be defined, since a blank location will be read as 'zero.'

The following is a description of the program control parameters used for CAT.

IJOB

IJOB is the identification number of the first job of a particular sequence, and is incremented by 1 for each succeeding job. This value is merely for the convenience of the user.

Default = 1.

IPRINT

This parameter determines the amount of printout from the fault tree construction and editing phases. Any integer from -1 to 4 may be specified, with each increasing value producing additional output. For an absolute minimum of output, code IPRINT = -1; however, a value of IPRINT of 0 or larger is suggested to provide the most useful information. Values of 1 or 2 allow the complete construction and editing phases to be followed step by step. IPRINT = 3 or 4 produces printouts of certain intermediate arrays during editing. A value of 4 includes the maximal number of arrays. This last value is not suggested, since large amounts of output will be produced. The most useful printouts will be obtained by setting IPRINT = 0, 2 or 3, as needed.

Default = 0.

KOUT

The KOUT parameter determines whether output is to be produced for use by PREP-KITT. KOUT = 0 for no output, and KOUT = 1 for output to be

produced on output device (unit) IOT. Note that KOUT = 1 requires additional CAT input in input section '&OUT'.

Default = 0.

IEDIT

This parameter is used to omit certain editing phases. Values below 98 produce full editing. IEDIT = 98 will bypass the search for transfers within the tree. IEDIT = 99 skips the intermediate editing stage. A value of IEDIT of 100 or above will omit both these editing sections.

Default = 0.

IOT

IOT is the unit number for fault tree and data output produced for PREP-KITT when KOUT = 1. This can be punched, or written onto disc or tape, depending upon the specification of IOT, and the user's installation. This output is also printed, along with other editing information, as part of the CAT output. If IOT is left blank for the first job of a multiple job run, it is set equal to the default value. If it is left blank in a subsequent job, it is set equal to IOT of the previous job in that run.

Default = 10.

NLIB

NLIB defines the number of component types to be input into the library section. A number which is erroneous may produce one of two effects. If the number is too large a warning will be produced, but the program will continue. If NLIB is too small, the extra component types will be skipped. However, the program will continue unless the system itself requires one of the component types so bypassed.

LNROW

LNROW should be set to the maximum number of columns of any of the decision tables, including that of the TOP event.

MAXINT

Set the value of MAXINT to the largest number of internal failure mode columns of any component table. This value is used to determine the number of failure mode names to be read by the program.

MXNROW

This variable is the total number of rows of all decision tables to be input. It need be only an estimate, and is used in determining whether sufficient space has been allocated for the total decision table library.

NNCMP

NNCMP defines the number of components in the system, as input by block 3 of the program. This value must be exact.

NNODE

NNODE is the largest node number used in the system flowchart. If NNODE is greater than the largest node number, its value will be correctly redefined later in the program. If modifications in the system being analyzed will change the numbering of nodes in later cases, NNODE may be set to the largest value anticipated and not changed for any of the runs.

MROW

MROW is set to the largest number of rows in any of the TOP event decision tables for one group of jobs. If any TOP event table exceeds MROW, that tree will be terminated and the next job begun.

Default = 1.

### 2.2.4 Library Data

The second section of program data is the decision table library input group. This consists of one set of cards for each component type (there will be NLIB sets in all). Each set is input in the following order:

```
LIBR      NAME1   NTYPE    NIN    NINT    NOUT    NROW (A4, 6X, A8, 2X, 12I5)
 MOD      NAME2   NAME3    ...                          (A4, 6X, 7(A8, 2X))
 ROW1               i1     i2      i3     ...          (A4, 16X, 12I5)
  .
  .
 ROWn               n1     n2      n3     ...
END
```

The 'LIBR' card contains the basic information for the component type. NAME1 is the 8 character name of the component type, NTYPE is a unique 5 digit type number, and NIN, NINT and NOUT are the numbers of inputs, internal failure mechanisms and outputs of the component. That is, NIN + NINT + NOUT is the length of each row of the decision table which is to follow. Furthermore, NIN + NOUT is the number of nodes assigned to that component type. Finally, NROW is the number of rows of the decision table.

The 'MOD' card lists the 8 character names of the component internal mechanisms. There should be exactly NINT of these. Note that, even if NINT = 0, this card is required.

Finally, the decision table itself will be input on the 'ROW' cards. There will be 'NROW' of these, one for each row of the decision table. The columns must be arranged in the order: inputs - internals - outputs, and must have exactly NIN + NINT + NOUT entries. A "don't care" state will be indicated by a '-1' in the appropriate column. Note that each component type group must end with its own 'END' card.

As pointed out previously, any library entries in excess of the 'NLIB' groups specified on 'DAT2' will be ignored. However, as it is not necessary that every library type be used in the system itself, the program

will not terminate unless one of the extra types is specifically required. Furthermore, if several different models will be used for one component in various runs, it may be convenient to include all models in the library using different type numbers. In each run, only the specific component type desired will be used.

2.2.5 Component Cards

Following the '&CMP' card is the group describing the system itself. This consists of a total of NNCMP cards, one per component. The format is (A4, 6X, A8, 2X, 1215), and the input is as follows:

COM NAME ITYPE NODE1 NODE2 NODE3 ...

Notice that the code 'COM' begins in column 2, and that the following six column field has been used, in the sample inputs, to number the component cards. This is solely for the convenience of the user, since the six columns following 'COM' are not read by the code. 'NAME' is the 8 character name of the component, and must be unique. In setting later boundary conditions, this will be used to identify the specified components. ITYPE is the component type number of the decision table to be used for this component. Finally NODE1, NODE2, etc., are the input/output node numbers of the component. Referring to component type ITYPE, there must be a total of NIN + NOUT node numbers, the first NIN of which will be inputs, with the final NOUT as outputs.

The requirements for the node numbers are that:

1) All output node numbers be unique. This refers to multiple outputs of a single component, and to all outputs of other components.

2) No component output may be connected directly to an input of the same component. However, an output and input may be connected to each other through any other component, including a simple "piece of wire."

3) All inputs must be connected to valid output nodes from other components; that is, no component may be left with undefined inputs. However, any output may be left unconnected. If a particular input node is not going to be used, it may be connected to a "dummy" component. For example, if it is desired to set a boundary condition at the input of a component such as a sensor, a dummy component must still be connected to that node. A simple dummy component that is of often used would have the following decision table inserted into the library:

| ROW | Internal | Output |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 1 | 1 |

Note that this component type has no inputs and only one output. A 'COM' card would then be set up for the dummy component of this type, whose single output node would be connected to the input node in question. Finally, the boundary state could be defined at this node, as in Section 2.2.7. With a boundary condition defined at that node, the dummy decision table would never be used, and its exact form is irrelevant. However, the NIN and NOUT parameters on the "LIBR' card for the dummy decision table (0 and 1 in this example) must agree with the number of nodes defined on the dummy 'COM' card.

### 2.2.6 TOP Event Definition

The TOP event is input in much the same way as the decision tables for the component types. After the '&TOP' card, the basic data for the TOP

is input on the 'TTOP' card, followed by the decision table itself, on the 'TOP' cards.

```
TTOP    NAME    NROW    NIN    NODE1    NODE2   ...   (A4, 6X, A8, 2X, 12I5)
 TOP1            I1      I2      ...                  (A4, 16X, 12I5)
 TOP2            J1      J2      ...
 TOPn
END
```

The 'TTOP' card contains an 8 character identification ('NAME'), for the TOP, followed by the number of rows and number of columns of the decision table (NROW and NIN). It is important to remember that NROW must be less than or equal to MROW given by the 'DAT4' card. Finally, the system nodes at which the TOP event decision table is defined (one per column) are input, up to a maximum of 10. The decision table is input on the 'TOP' cards. Note that the code 'TOP' begins in column 2, and that the numbers directly following are for the user's convenience, and are not read by the code.

2.2.7 Boundary Conditions

This data group, if required, contains the specification of the boundary conditions. If this group is needed, it begins with an '&BC' card and is followed, in any order, by 'INT' and 'EXT' cards defining the boundary (initial) values of internal component modes and external system states respectively. The form of the data is thus:

```
&BC
 INT    NAME    MODE1  MODE2   ...                          (A4, 6X, A8, 2X, 12I5)
 EXT            NODE1  STATE1  ...   NODE6   STATE6       (A4, 16X, 12I5)
END
```

Each 'INT' card specifies the predefined states for one component's internal modes. NAME is the 8 character identification of the component as given by NAME on the appropriate 'COM' card. Then one value is defined for each internal mechanism (column) of the component, including a -1 for

any mode which is not to be set as a boundary condition. Note that no state should be left as a blank, even if only one state is to be defined. Any blanks will be read as zeros, and thus a "good" state will be set, rather than being left undefined (i.e., set equal to -1).

The 'EXT' cards set the boundary conditions at system nodes. Each card can define up to 6 node conditions, by first specifying the appropriate node, followed by the state to be set. Notice that any number of 'EXT' cards may be used and may be intermixed with the 'INT' cards. This data group must be followed by an 'END' card.

2.2.8 Failure and Repair Data

If output is desired to be used with the PREP-KITT codes, one final data group is required. This data begins with an '&OUT' card, and contains the control information and failure and repair data. This information, combined with the fault tree produced by CAT, is output to a card punch (or disc or tape file) in a format suitable for input to PREP-KITT. The input in this data group is as follows:

```
&OUT
      NG     MIN    MAX       IDEX1  IDEX2  NPROB                    (6I10)
      MC  NREJEC    NTR        IREN           TAA              (4I10,F20.3)
NAME1          λ1     τ1 INT1 STATE1NAME2       λ2    τ2 INT2 STATE2

  .
  .                                                 2(A8,2X,F10.6,F10.3,215)
  .
NAMEi          λi     τi INTi STATEiNAMEj       λj    τj INTj STATEj
END
```

The first two cards contain the control information for PREP, in the same format as used by that code. A description of these variables may be found in the PREP-KITT manual [3]. (Note that NPROB is an extra

variable, used only by the UCLA version of PREP, and may be omitted for other versions.) Since NG, the number of gates, is supplied by CAT, it may be left blank.

The remaining cards contain the failure and repair data for the components, supplied one or two sets to a card, at the user's option. Since each component may have several failure modes, with several states for each mode,several sets of data may be required for each component. Each set contains the following information:

| Data | Format | Description |
|---|---|---|
| Name | A8, 2X | 8 character name of component, same as used on 'COM' card. |
| $\lambda$ | F10.6 | Failure rate (per $10^6$ hour). Note: $\lambda \geq 10^{-9}$, or $\lambda < 0$ |
| $\tau$ | F10.6 | Repair time (hours) $10^6 > \tau \geq 10^{-3}$ |
| INT | I5 | Internal failure mode (column of decision table) |
| STATE | I5 | Failure state |

Note that each $\lambda$ and $\tau$ represent the data for one specific failure state of one internal failure mode (column), where 'INT' is the column number and 'STATE' is the failure state. (For a component with only one internal column in its decision table, INT will always = 1.) Then, for each column, one set of data is required for each state which appears in that column in the decision table, unless it is known that a specific state will not appear in the final decision table. The use of a component as an inhibit condition is also allowed as an input option to PREP by setting $\lambda \leq 0$, and $\tau$ as a number between 0 and 1 (see reference 3, page 31 for definitions).

As an example of the above, the following two rows of a decision table

will be used to represent a simple system:

| Input | Maintenance | Internal Failure | Output |
|---|---|---|---|
| - | - | 5 | 0 |
| - | 101 | - | 0 |

Internal failure 5 represents a failure to run, and maintenance = 101 means system unavailable due to maintenance. Assuming a failure rate of $5 \times 10^{-4}$ ($500 \times 10^{-6}$), 24 hour repair time, and 1% maintenance unavailability, the input data for 'SYSTEM-A' would be:

| | | | | |
|---|---|---|---|---|
| SYSTEM-A | 500.0 | 24.0 | 2 | 5 |
| SYSTEM-A | -1.0 | 0.01 | 1 | 101 |

These sets may be input in any order and intermixed with other components. However, all data in this input data group must obey the following FORTRAN rules:

1) all names must be left justified, and

2) all integers must be right justified.

Finally, the last data card must be followed by an 'END' card.

2.2.9 Multiple Jobs

The data for the first job terminates with an '&END' card, and may be followed by further jobs. Each job must utilize the same system and library, but may define a new TOP and boundary conditions, as well as new failure data. We will see, however, that changes in a system may often be made simply by appropriate changes in boundary conditions.

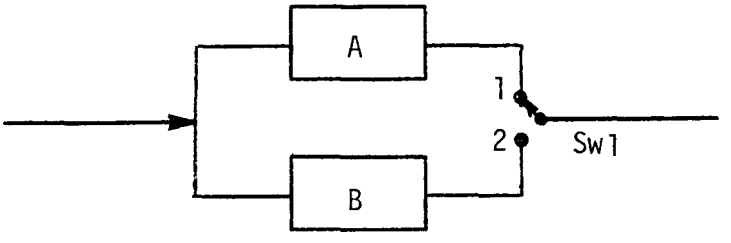
The new job may begin, if desired, with a new title (20A4). Furthermore, the parameters of the DAT1 card, may be redefined, again as an optional feature. This would be done with the following setup.

```
&DAT
 DAT1      IJOB      IPRINT     KOUT     IEDIT     IOT
END
```

If the DAT1 card is included, all values must be defined. If DAT1 is not used, neither the &DAT nor END cards are needed.

The TOP definition is the only required data for the new run, and is input exactly as before. For the new run, all boundary conditions will be reset to undefined states. All boundary conditions must be set, as before, by the '&BC' section following the TOP event definition. If KOUT is set to 1 (or left from the previous run), new PREP-KITT data is required, even if identical to the previous run. This is input in the '&OUT' section, as described in section 2.2.8. Finally, an '&END' card terminates this new run, which may be followed by further jobs.

Although the system may not be redefined in succeeding jobs, one may use switches, set by different boundary conditions, to switch in new subsystems, different component models, etc., at any points in the system. Consider the block diagram below:




By defining switch 1 to be in position 1 or 2 in successive runs, components (or systems) A and B are alternately switched into the overall system. Other capabilities include the possibility that A and B are simply different models for the same component. Both would be defined as part of the original system on 'COM' cards, and the appropriate one

would be chosen by the switch position.

## 2.3 CAT Output

The output from the CAT code consists of five phases illustrated in more detail in the sample output in Appendix E. First is a listing of all input data, excepting the PREP-KITT data (if any). This includes extensive error checking and diagnostics to pinpoint input errors, inconsistencies or missing data.

Second is a set of cross-references, listing input and output nodes, and specific component names for all inputs to each component.

Next is the step-by-step construction of the fault tree. The output of this phase is controlled by the IPRINT flag on the 'DAT1' card. This is followed by the final fault tree printout itself.

Finally, if KOUT = 1, CAT produces the PREP-KITT output. In addition to the punched (or tape or disc) output, is an identical printed output of the tree. This includes the '*' cards required by the PREP code. This printed output also includes error messages to indicate extra or missing data. Note that the fault tree output will be produced in all cases where KOUT = 1, even if the failure data is missing. However, in this case, the output will not contain the required failure data for PREP.

Since PREP can only accept a single failure mode for each component, CAT must construct a unique name for each component-failure mode-failure state set. This is listed in a final table, along with the original component name, failure mode and state. This name is simply an eight digit code consisting of the internal mode number (4 digits) and the failure state (4 digits). Using the example in Section 2.2.8, assume that the internal (computer generated) mode number for maintenance for SYSTEM-A was 123, and for internal failure was 124 (corresponding to

columns 1 and 2 of the table). Then, the code for: SYSTEM-A, internal failure, state 5 would be:

01240005,

and the code for: SYSTEM-A, maintenance, state 101, would be:

01230101.

Notice that, in this case, the "component" 01230101 would represent an inhibit condition (see Section 2.2.8) and 01240005 would represent a primary component failure.

# 3. DEVELOPMENT OF DECISION TABLES

## 3.1 Introduction

The development of accurate decision table models is a central requirement in the current approach. However, since the components and systems of interest are so diverse, and models of various levels of sophistication are desirable, an effective method of constructing such tables is needed. Two general methods for developing decision tables of components will be described in the following sections, and a number of decision tables which have been studied and used previously will be outlined in Appendix B and will serve as a reference to the users.

Two ways of generating decision tables will be described in this report. The first approach (inductive) consists of systematically constructing the decision table by enumerating all possible combinations of input states and internal modes, and then finding the appropriate output state for each combination. This is a typical method of constructing decision tables which assures a complete, though complex table. The second approach begins by considering all possible output states and tracing back to all possible input states. This deductive method is similar to that used by CAT itself in constructing fault trees.

The advantage of the first method, as pointed out, is the assurance of completeness, at the expense of complexity. For example, a component with 2 inputs, each with 3 states, and 3 internal modes, each with 3 states, would result in a table with 6 columns and $3^5 = 243$ rows before reduction. The reduction itself, although tedious, is a process amenable to computer implementation. The second method, although not as straightforward, has the advantage of allowing one to immediately

concentrate on the output states of most interest. It has the serious drawback, however, of allowing the possible oversight of some important features unless a careful check of completeness is made.

## 3.2 Inductive Method of Decision Table Development

The first step in this method is to enumerate all combinations of input states and internal modes of the component. Then, the output state(s) for each combination are determined essentially by a failure modes and effects analysis. Finally, a decision table reduction method may be utilized to produce a compact table, suitable for use with the code. A step by step description of the process of generating decision tables by this method is given in following paragraphs. As an example a decision table for a pump is developed.

Step 1. First, an investigation must be made of the physical characteristics and design considerations of the component, in order to determine all of its possible input states and internal modes. For a pump, there are two inputs, one internal mode and one output:

Input 1: Main flow (pressure) input

0 - no pressure in or pressure too low

1 - normal pressure

Input 2: Power input

0 - no power in

1 - power in

Internal Mode: Condition of pump

0 - pump in good condition

4 - pump fails to start

5 - pump fails to run normally

Output: Main flow (pressure) output

0 - no pressure out or pressure out too low

1 - pressure out

Note that in other analyses, these states may differ, depending upon the specific nature of the pump, and the depth of analysis desired. For example, the state "fails to run normally" could further be broken down into specific failures such as impeller failure, shaft failure, etc., if specific data on these are available, and it is desired to separate out these failures. This could be especially useful in trying to isolate potential common mode failures, etc.

Step 2. An initial decision table is then constructed by listing these combinations of input states and internal modes, along with the output state which results. For the pump as example, there are 2 x 2 x 3 = 12 possible rows:

TABLE 5. ORIGINAL DECISION TABLE OF PUMP.

| Row | Input 1 Main flow | Input 2 Power | Internal Mode | Output |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 4 | 0 |
| 3 | 0 | 0 | 5 | 0 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 4 | 0 |
| 6 | 0 | 1 | 5 | 0 |
| 7 | 1 | 0 | 0 | 0 |
| 8 | 1 | 0 | 4 | 0 |
| 9 | 1 | 0 | 5 | 0 |
| 10 | 1 | 1 | 0 | 1 |
| 11 | 1 | 1 | 4 | 0 |
| 12 | 1 | 1 | 5 | 0 |

Table 5 is the original decision table of the pump. Row 1 of Table 5 shows that if there is no fluid input, no power input, and the pump is good, there will be no flow at the output. Row 10 shows that if the flow (pressure) input is high enough, the power is on, and the pump is good, there will be flow at the output side of the pump. The other rows can be understood in a similar manner.

Step 3. The decision table is now ready to be reduced. Although the decision table developed in the previous steps can be used as the input data for CAT code, it is very lengthy. A modification can be made by introducing "don't care" states into the table, in order to make it simpler, as well as to save computer time and memory in the process of constructing the fault tree.

The basic rule for reducing decision tables is as follows. If several rows have identical output, input, and internal states except for one input or internal mode, and if this exception includes all possible states which can occur, then these rows can be combined into a single row with a "don't care" state. For example, in Table 5, rows 1, 2 and 3 all have 0 as output, 0 main flow input, and 0 power input. Furthermore, the remaining column, the internal mode, includes all three states possible (0, 4 and 5). This implies that, regardless of the internal state, the output will be 0 as long as the flow input is at low pressure and no power is present at input 2. Thus, rows 1, 2 and 3 can be combined into a single row with a 'don't care' state, i.e.,

0 0 -1 0

Similarly, rows 4, 5 and 6 in Table 5 can be reduced to

0 1 -1 0,

and rows 7, 8 and 9 can be reduced to

1 0 -1 0.

Finally, similar reduction can be done to rows 2, 5, rows 3, 6, rows 8, 11, rows 9, 12, and a reduced decision table of the pump can be obtained as shown in Table 6.

TABLE 6. REDUCED DECISION TABLE OF PUMP.

| Row | Input 1 Main Flow | Input 2 Power | Internal Mode | Output |
|---|---|---|---|---|
| 1 | 0 | 0 | -1 | 0 |
| 2 | 0 | 1 | -1 | 0 |
| 3 | 1 | 0 | -1 | 0 |
| 4 | 1 | 1 | 0 | 1 |
| 5 | 1 | -1 | 4 | 0 |
| 6 | 1 | -1 | 5 | 0 |
| 7 | 0 | -1 | 4 | 0 |
| 8 | 0 | -1 | 5 | 0 |

If we look at rows 1 and 2 in Table 6, both of them have 0 as output, 0 as flow input and -1 as the internal mode; furthermore, the power input includes both 0 and 1, the only possible states of that input. In this case rows 1 and 2 of Table 2 can be combined again into:

0 -1 -1 0.

Similarly rows 1 and 3 in Table 6 can be combined into:

-1 0 -1 0.

Similar reductions are done to rows 5, 7, and rows 6 and 8. The decision table of the pump is further reduced as Table 7.

TABLE 7. FINAL DECISION TABLE OF PUMP.

| Row | Input 1 Main Flow | Input 2 Power | Internal Mode | Output |
|---|---|---|---|---|
| 1 | 0 | -1 | -1 | 0 |
| 2 | -1 | 0 | -1 | 0 |
| 3 | 1 | 1 | 0 | 1 |
| 4 | -1 | -1 | 4 | 0 |
| 5 | -1 | -1 | 5 | 0 |

Table 7 is much simpler than the initial decision table as shown in Table 5. A discussion about this reduced form of decision tables has been worked out in reference [1] which indicates the reduced form is equivalent to the original decision table in the sense of probability considerations in the construction of fault trees, but with the advantage of being simpler than the original form.

## 3.3 Deductive Method of Decision Table Development

One should bear in mind that the purpose of the decision tables is to supply information for constructing the fault trees of various systems. Each system fault tree starts from the TOP event and is then traced back to the primary events. In this way, the decision table is actually used in the reverse direction. For instance, in the pump example, the information desired will be of the following type: "What causes can produce 'no pressure or pressure too low' at the output?", or "What events are required to obtain 'normal pressure' at its output?" From such investigations, we can obtain a mini-fault/success tree [4] for each possible output state. By collecting these trees for all possible outcomes, one obtains the decision table for the component. A pump is again used as an example in this case.

Step 1. First, the investigator must become familiar with the physical characteristics and design purpose of the component. From these, the analyst then finds different source reasons for malfunctions (this is essentially a FMEA process). The analyst should also collect the available failure history of each component in actual industry experience.

For the case of a pump, the only situation in which there will be output is when the pump is good, there is power input to the pump, and the flow input has 'pressure in'.

The internal mode of the pump can either be 'good', 'fails to start' or 'fails to run', each with a different probability; the total, however, will sum up to one.

Step 2. Define all the possible states of inputs, internal modes and outputs of the component. For a pump, there are two inputs, one internal mode and one output as described previously.

Step 3. Construct the mini-fault/success tree. As an example, Figures 1 and 2 show the mini-fault/success trees for a pump.

Step 4. Construct the decision table by using the mini-fault/success trees, treating blank spaces as 'don't care' states. Table 8 shows the decision table for a pump obtained in this way. Notice the equivalence between Tables 7 and 8.

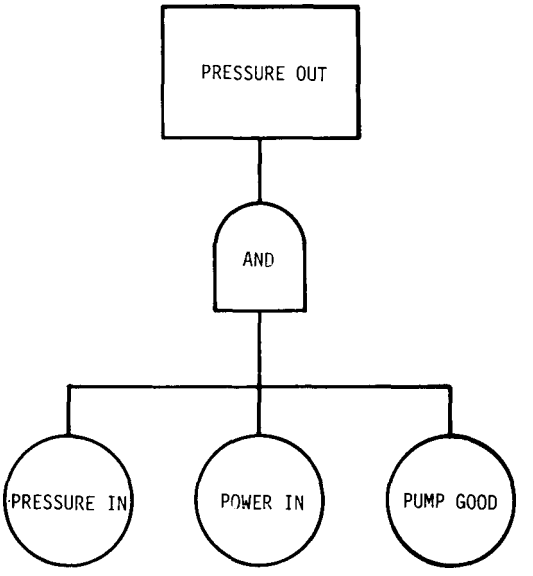



FIG. 1 MINI-FAULT/SUCCESS TREE I FOR PUMP

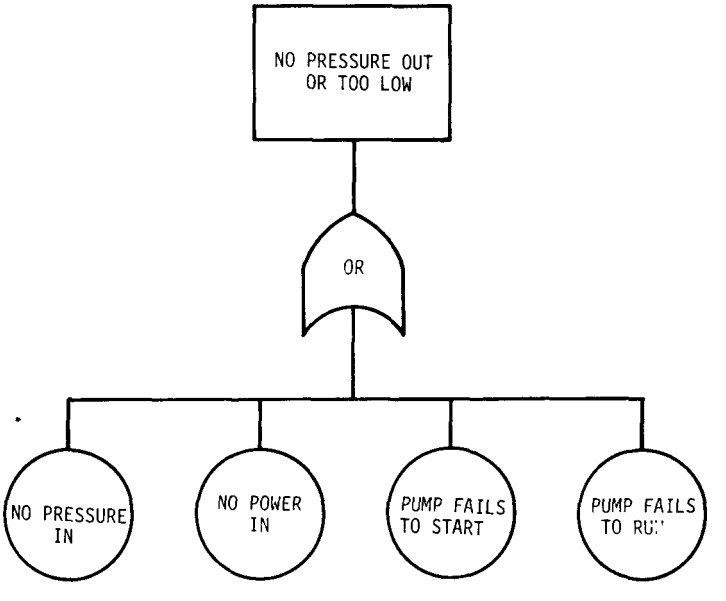



FIG. 2 MINI-FAULT/SUCCESS TREE II FOR PUMP

TABLE 8. DECISION TABLE FOR PUMP BY DEDUCTIVE METHOD.

| Row | Main Flow | Input 2 Power | Internal Mode | Output |
|---|---|---|---|---|
| 1 | 0 | -1 | -1 | 0 |
| 2 | -1 | 0 | -1 | 0 |
| 3 | -1 | -1 | 4 | 0 |
| 4 | -1 | -1 | 5 | 0 |
| 5 | 1 | 1 | 0 | 1 |

Step 5. Check the completeness of decision tables. Make sure each mode of operation of the component has been included in the decision table unless it is impossible or negligible.

## 3.4 The Use of Decision Tables in the Construction of Fault Trees

In order to construct fault trees for general systems, information is required both to describe the system itself, and the operation of the specific components within the system. The decision table methodology has been used to describe the operation of the specific components of the system. The use of such decision tables in constructing fault trees will now be illustrated referring to Table 8 of Section 3.3. The event "no output from pump" will be used as an event to be analyzed. This might be the TOP event of a tree, or some intermediate event which would be required to produce a zero input to a succeeding component.

Given the desired output state, a search is made for rows with the correct state, in this case Rows 1 through 4 of Table 8. Since any one of these rows has the correct output, they are connected by an OR gate, each row being a single input (see the mini-fault tree used for the pump).

Since in all rows, two of the three signals are of the "don't care" type, each row is replaced by a single event. Thus, Row 2 is replaced by the event "no power," which must be developed further with the use of another decision table. Row 4 is replaced by the event "pump fails to start", which is a primary failure and thus becomes a direct primary input event.

If the desired output state were "normal output from pump", then a search of the rows reveals that only row 5 gives the correct output. Here, there are three states defined, all of which must be true for the output state to be 1. The result, then, is an AND gate with the three appropriate inputs. In this case, one input represents a primary event (pump good), and thus terminates that branch. However, the other inputs to the AND gate are component input states, and must therefore, be traced backwards to the previous components and developed further.

## 4. APPLICATIONS

Using the preceding methodology, the CAT Code was developed, and used to construct a number of fault trees for various systems. As each system illustrates different features of this approach, two new examples will be used to complement the pressure tank system, and reactor Residual Heat Removal System, described in Reference [1]. These new systems are a Containment Spray Recirculation System and a Consequence Limiting Control System.

### 4.1 Containment Spray Recirculation System (CSRS)

#### 4.1.1 Description of CSRS

The Containment Spray Recirculation System (CSRS) is described in WASH-1400 [5]. For purposes of illustration, this system has been simplified slightly and its flow diagram is shown in Figure 3. The intended function of the CSRS is the recirculation of the containment sump water through the heat exchangers of the Containment Heat Removal System to spray headers inside the containment, thus removing energy and fission products from the containment in the event of a LOCA.

The following are important features of the system:

1. The system is comprised of four trains. During the first twenty-four hours following a LOCA, the logic of the system for successful operation is two-out-of-four; it becomes one-out-of-four after that period.
2. Each train consists of a pump, a heat exchanger and a spray header. Two of the pumps are inside the containment.
3. All valves in the two trains which have the pumps outside the containment are normally open.

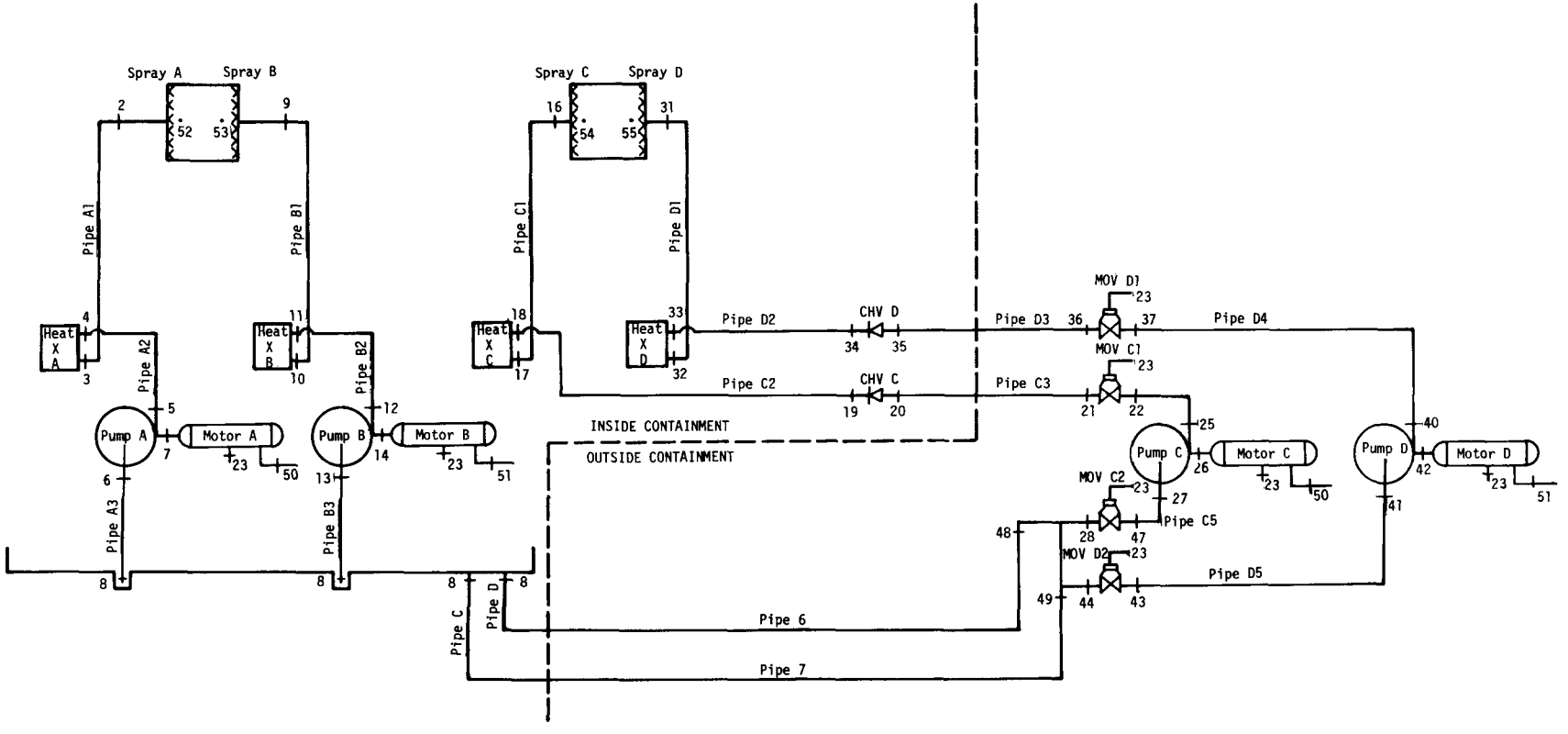



Figure 3. Simplified Flow Diagram for the Containment Spray Recirculation System

4. The CSRS is actuated by a signal from the Consequence Limiting Control System which turns on the pumps and also opens the motor-operated valves if they had been inadvertently left closed after maintenance.
5. The electrical power supply is common to one inside and one outside train. The other pair of trains also have a common electrical power supply to the motors of their pumps.
6. A train can be disabled due to maintenance. However, only one leg is allowed to be down for maintenance at any time.

4.1.2 TOP Event and Preliminary Considerations

In order to construct the fault tree for this system, the decision tables and TOP event definition had to be developed. Many of the tables had been used for previous examples [1,2]. However, new models were developed for the spray headers, heat exchangers, and pump motor (see Appendix B). Furthermore, the model used for the motor-operated valve [1] was modified by deleting a slip clutch failure mode.

The model for the pump motor was included in order to separate the failures of the pump and pump motor into two components. Thus, the motor itself received power from the power supply, and a signal to turn on or off. The "power" for the pump (Table 8) was then actually the mechanical coupling from the motor.

The TOP event for this system was defined as failure of the recirculation system to supply adequate spray cooling to the containment following a LOCA. This requires operation of at least two of the four legs for the first twenty-four hours, and one-out-of-four legs thereafter. Thus, the logic of the TOP event changes after twenty-four hours, and

this situation must be developed explicitly. This was done using inhibit gates modeled by the following decision table (Table 9) as part of the TOP event:

TABLE 9. LOGIC MODEL FOR INHIBIT CONDITION.

| Row | Leg A | Leg B | Leg C | Leg D | Inhibit Condition | System Output |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | -1 | 1 | 0 |
| 2 | 0 | 0 | -1 | 0 | 1 | 0 |
| 3 | 0 | -1 | 0 | 0 | 1 | 0 |
| 4 | -1 | 0 | 0 | 0 | 1 | 0 |
| 5 | 0 | 0 | 0 | 0 | 2 | 0 |

In this table, zeroes represent the condition "no flow" from the appropriate leg, or "insufficient flow" as a system output. The inhibit condition is treated as an internal mode, with state 1 representing the condition $0 < t < 24$ and state 2 as $t > 24$. Only those rows leading to the system state "insufficient flow" are shown; that is, only those combinations in Table 2 will allow system failure. Thus, it is seen that for $0 < t < 24$, either three or four leg failures will lead to TOP failure, while for $t \geq 24$, four failures are required.

An additional complication in this system is the inclusion of maintenance. Any single one of the four legs is permitted to be under repair for a period of up to twenty-four hours. Thus, should the system fail while one leg is under maintenance, during the first

twenty-four hours, only two subsequent failures would be needed, or three subsequent failures thereafter. Since only one leg may be under maintenance at a time, this situation represents an "exclusive OR" type of logic, and is modeled analogously to the inhibit conditions in Table 9. The TOP event is defined as "insufficient output with no leg under maintanance OR insufficient output with one leg under maintenance"; "one leg under maintenance" is further divided into "leg A under maintenance", "leg B under maintenance", etc. Since each of these maintenance conditions is treated as if it were an inhibit condition, the "exclusive" nature of the TOP OR gate is retained without explicitly requiring such a gate. Thus, the resulting tree can be analyzed by any of the existing computer codes (such as PREP-KITT). This structure is seen in the upper level structure of the completed fault tree, Figure 4.

Finally, as an example of the input used in the CAT Code, Table 10 reproduces the component node-numbering table for this system. Referring to this table and Figure 3, and using heat exchanger A as an example (component 3 in the table), the input defines this component as "type 2" (i.e., using decision Table number 2), with input node 4 and output node 3. This output node is connected to the input of the pipe (component 2), whose output (node 2) is ultimately connected to spray header A (component 1). Components 47-52 represent the inhibit gate models as shown in Table 9 for the TOP structure, including both the maintenance modeling and time switching logic.

4.1.3 Discussion of Fault Tree

The previously described input to CAT was used to produce the

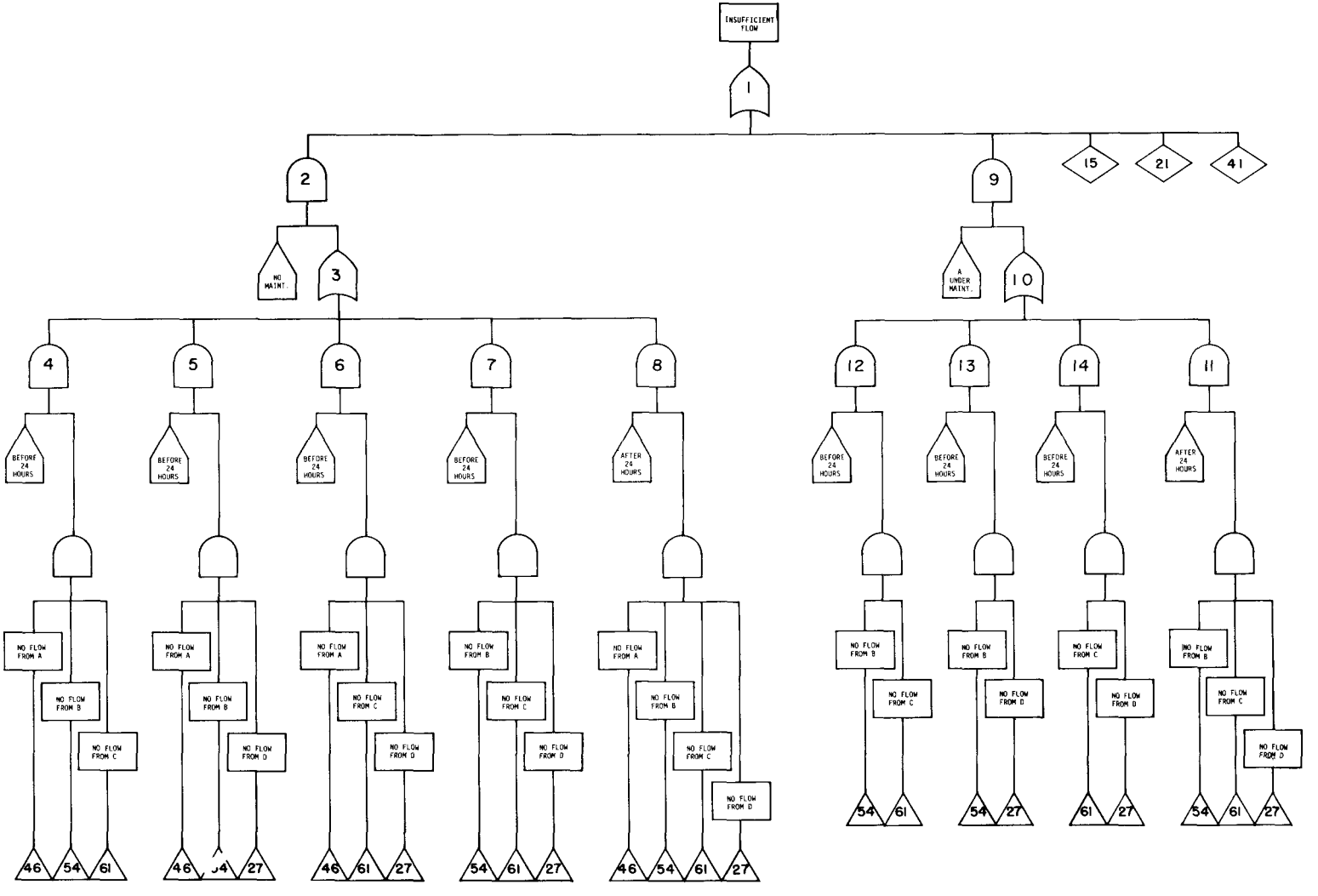



Figure 4. Upper Level Structure of Fault Tree for CSRS

Table 10. Component Index Input Printout for CSRS

COM CARD FOUND. DATA VALIDATION CONTINUING.

CONTAINMENT SPRAY RECIRCULATION SYSTEM

COMPONENT INDEX INPUT PRINTOUT

COMPONENT CARD PRINTOUT

| INDEX | CODE | NAME | TYPE | INPUT/OUTPUT NODES | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | COM1 | SPRAYA | 1 | 2 | 52 | | | | |
| 2 | COM2 | PIPEA1 | 3900 | 3 | 2 | | | | |
| 3 | COM3 | HTEXA | 2 | 4 | 3 | | | | |
| 4 | COM4 | PIPEA2 | 3900 | 5 | 4 | | | | |
| 5 | COM5 | PUMPA | 4300 | 6 | 7 | 5 | | | |
| 6 | COM6 | PIPEA3 | 3900 | 8 | 6 | | | | |
| 7 | COM7 | WATER | 3 | 8 | | | | | |
| 8 | COM8 | MOTORA | 4 | 50 | 23 | 7 | | | |
| 9 | COM9 | SPRAYB | 1 | 9 | 53 | | | | |
| 10 | COM10 | PIPEB1 | 3900 | 10 | 9 | | | | |
| 11 | COM11 | HTEXB | 2 | 11 | 10 | | | | |
| 12 | COM12 | PIPEB2 | 3900 | 12 | 11 | | | | |
| 13 | COM13 | PUMPB | 4300 | 13 | 14 | 12 | | | |
| 14 | COM14 | PIPEB3 | 3900 | 8 | 13 | | | | |
| 15 | COM15 | MOTORB | 4 | 51 | 23 | 14 | | | |
| 16 | COM16 | SPRAYC | 1 | 16 | 54 | | | | |
| 17 | COM17 | PIPEC1 | 3900 | 17 | 16 | | | | |
| 18 | COM18 | HTEXC | 2 | 18 | 17 | | | | |
| 19 | COM19 | PIPEC2 | 3900 | 19 | 18 | | | | |
| 20 | COM20 | CHVC | 5 | 20 | 19 | | | | |
| 21 | COM21 | PIPEC3 | 3900 | 21 | 20 | | | | |
| 22 | COM22 | MOVC1 | 6810 | 22 | 23 | 21 | | | |
| 23 | COM23 | CONT SIG | 3 | 23 | | | | | |
| 24 | COM25 | PIPEC4 | 3900 | 25 | 22 | | | | |
| 25 | COM26 | PUMPC | 4300 | 27 | 26 | 25 | | | |
| 26 | COM27 | MOTORC | 4 | 50 | 23 | 26 | | | |
| 27 | COM28 | MOVC2 | 6810 | 28 | 23 | 47 | | | |
| 28 | COM31 | SPRAYD | 1 | 31 | 55 | | | | |
| 29 | COM32 | PIPED1 | 3900 | 32 | 31 | | | | |
| 30 | COM33 | HTEXD | 2 | 33 | 32 | | | | |
| 31 | COM34 | PIPED2 | 3900 | 34 | 33 | | | | |
| 32 | COM35 | CHVD | 5 | 35 | 34 | | | | |
| 33 | COM36 | PIPED3 | 3900 | 36 | 35 | | | | |
| 34 | COM37 | MOVD1 | 6810 | 37 | 23 | 36 | | | |
| 35 | COM40 | PIPED4 | 3900 | 40 | 37 | | | | |
| 36 | COM41 | PUMPD | 4300 | 41 | 42 | 40 | | | |
| 37 | COM42 | MOTORD | 4 | 51 | 23 | 42 | | | |
| 38 | COM43 | PIPED5 | 3900 | 43 | 41 | | | | |
| 39 | COM44 | MOVD2 | 6810 | 44 | 23 | 43 | | | |
| 40 | COM47 | PIPEC5 | 3900 | 47 | 27 | | | | |
| 41 | COM48 | OR1 | 6 | 48 | 49 | 28 | | | |
| 42 | COM49 | OR2 | 6 | 48 | 49 | 44 | | | |
| 43 | COM50 | PIPE6 | 3900 | 8 | 48 | | | | |
| 44 | COM51 | PIPE7 | 3900 | 8 | 49 | | | | |
| 45 | COM52 | POWER1 | 3 | 50 | | | | | |
| 46 | COM53 | POWER2 | 3 | 51 | | | | | |
| 47 | COM54 | TOPC | 101 | 52 | 53 | 54 | 55 | 15 | |
| 48 | COM55 | TOPD1 | 102 | 53 | 54 | 55 | 56 | | |
| 49 | COM56 | TOPD2 | 102 | 52 | 54 | 55 | 57 | | |
| 50 | COM57 | TOPD3 | 102 | 52 | 53 | 55 | 58 | | |
| 51 | COM58 | TOPD4 | 102 | 52 | 53 | 54 | 59 | | |
| 52 | COM59 | TOTAL | 103 | 15 | 56 | 57 | 58 | 59 | 1 |

END

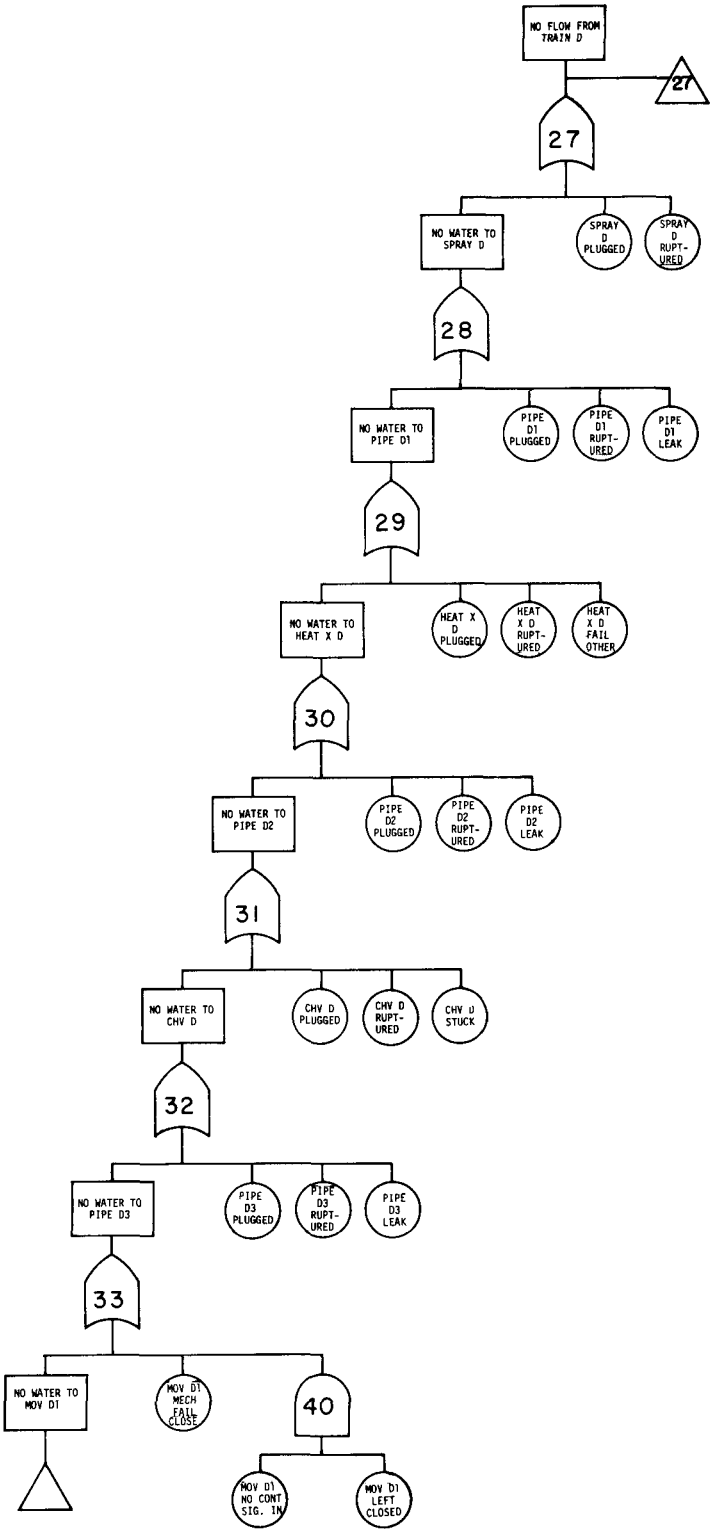



Figure 5. Development of Lower Events for CSRS Fault Tree (page 1)

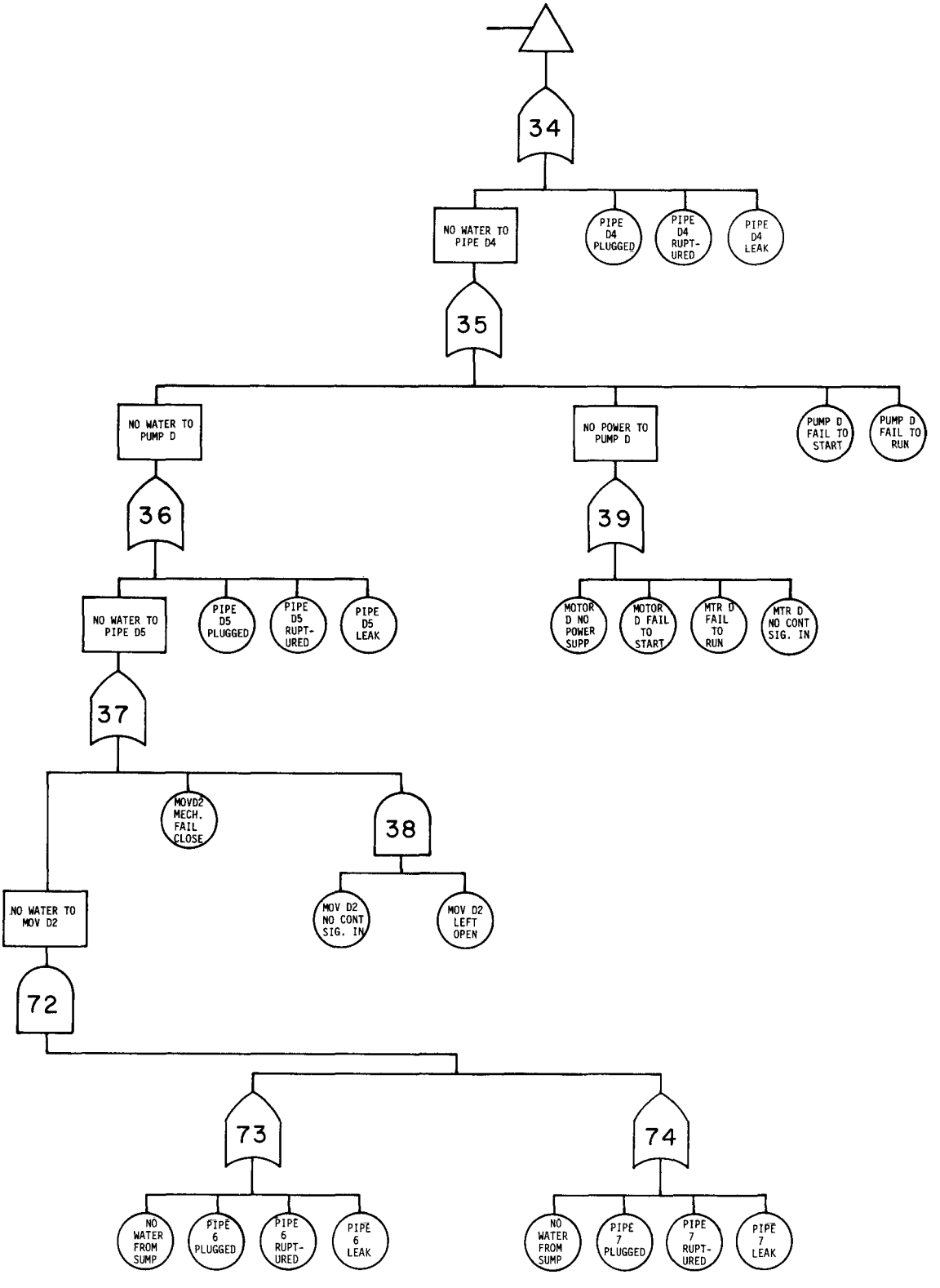



Figure 5. Development of Lower Events for CSRS Fault Tree (page 2)

fault tree discussed here. The program was run on the IBM 360/91 computer at UCLA. A core storage of 150K bytes was used which would be sufficient for 800 gates in the tree before editing (a core storage of 114K bytes would be sufficient for 200 gates). The code produced 656 gates which were reduced to 77 in the final tree, mainly through the use of transfers. A total of 6.5 CPU seconds was used.

The upper-level structure of the tree is shown in Figure 4. The branch of the tree beneath gate 2 assumes no maintenance, while the branches beneath 9, 15, 21, and 41 assume that one train is disabled for maintenance. Since these four branches are similar, only the branch beneath gate 9 is developed further in the figure.

The houses at the next level of the tree show whether the tree is developed assuming system failure when all trains fail (after 24 hours following a LOCA), or when any three out of the four trains fail (before 24 hours). The branches of the tree beneath this level are produced in a straightforward manner by the CAT code, utilizing Table 9 for the modeling of inhibit gates. For illustrative purposes, the development of the event "no flow from D" is shown in Figure 5.

## 4.2 Consequence Limiting Control System (CLCS)

### 4.2.1 Description of CLCS

The Consequence Limiting Control System is designed to measure the containment pressure and, if specific pressure levels are exceeded, initiate operation of equipment designed to control the containment environment [5].

A simplified block diagram of a Consequence Limiting Control System is shown in Figure 6, and a more detailed system diagram is shown in Figure 7. A general description of the system is given as follows:

1. The system is designed to detect out-of-tolerance conditions within the containment by measuring containment pressure, and to initiate operation of equipment and systems designed to limit and counteract these conditions.

2. There are two output signals that will activate the safety devices. A containment pressure rise to 1.5 psig produces signals which initiate the HI containment pressure phase and the containment vacuum pumps are tripped; certain containment isolation valves are closed and back-up signals are sent to the safety injection control system. A further rise of containment pressure to 10.3 psig will initiate the HI HI containment pressure phase of the CLCS. This will start the containment spray injection system and the containment spray recirculation system, close the remaining containment isolation valves, initiate start-up of two diesel generators and activate the appropriate motor-operated circulating and service water valves to divert service water to the containment spray heat exchanger.

3. The CLCS is made up of four logic trains (two for HI and two for HI HI), and four measurement channels. Each logic train trips when three-out-of-four measurement channels sense a trip pressure signal. Each measurement channel contains its own transducer to sense the pressure, and a comparator to detect different pressure levels. Upon a

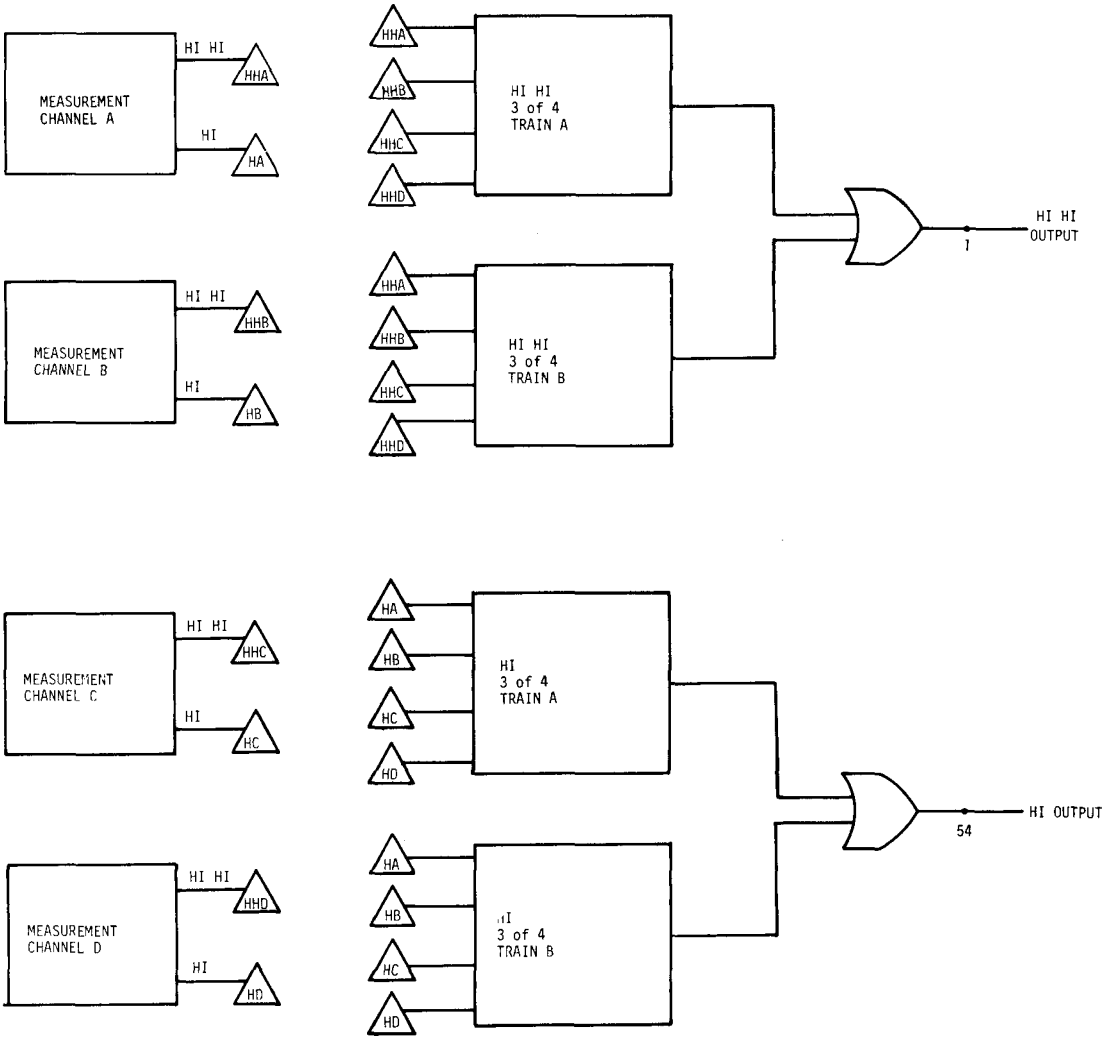



Figure 6. CLCS Simplified Diagram (5)

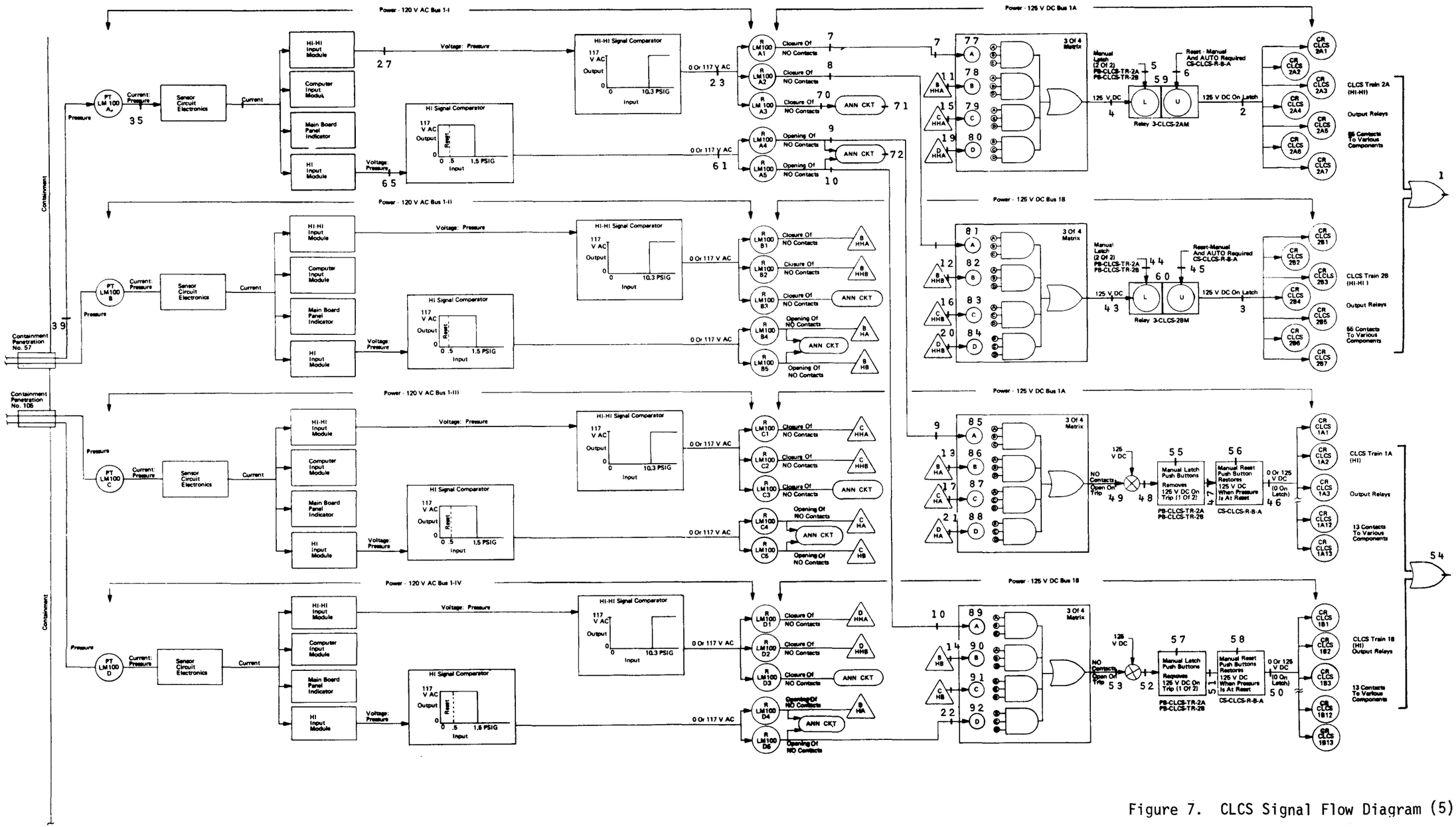



Figure 7. CLCS Signal Flow Diagram (5)

pressure of 1.5 psig, each measurement channel will send a HI containment pressure signal, and at 10.3 psig, each channel will send a HI HI signal. There are two logic trains for each pressure level, in a one-out-of-two configuration, to provide redundancy in initiating the equipment.

4. The operator can initiate either of the two sets of trains manually.

5. When containment pressure is reduced to -0.5 psig, following an increased pressure above either or both set points, the CLCS circuits signal the operator that the reset point has been reached, permitting him to reset the logic trains. This reset function occurs for all four logic trains at the same time. The equipment can be returned to the pre-accident configuration only after the reset signal has been received by the operator.

6. The HI HI signal trains were designed to be harder to trip than the HI signal trains, in order to minimize the chance of inadvertent spray actuation by a HI HI signal. The logic channels and sensors are designed to trip on loss of power to a HI signal state, but are prevented from tripping to the HI HI state. Also, manual trip by the operator requires the operation of two push buttons for a HI HI signal, but only one button for a HI signal.

7. The measurement channels can be tested monthly, one at a time. The system logic trains trip when two-out-of-three measurement channels reach certain pressure levels under test conditions.

8. The logic trains are also testable. While in the test mode, a logic train will be automatically pulled out-of-test and returned to normal operation if the train not being tested trips. Thus, testing does not negate the function of any portion of the CLCS during its testing.

4.2.2 Top Event and Preliminary Considerations

Four different failure modes of the Consequence Limiting Control System have been considered as TOP events in the fault tree construction:

1. Containment pressure between 1.5 psig and 10.3 psig, but no Hi signal is sent by the CLCS .

2. Containment pressure above 10.3 psig, but no HI HI signal is sent by the CLCS.

3. Containment pressure normal, but HI HI signal is sent because of a malfunction of the CLCS.

4. Containment pressure normal, but HI signal is sent by a malfunction of the CLCS.

In this example, the TOP event "containment pressure normal, but Hi signal sent by malfunction of CLCS" has been further investigated in detail. The upper level logic of this TOP event has been developed as required by the CAT Code. It includes the maintenance of the measurement channels and also shows the logic of the maintenance scheme, which allows testing of only one at a time. Table 11 shows the decision table of this TOP event.

TABLE 11. TOP EVENT DECISION TABLE FOR: "CONTAINMENT PRESSURE NORMAL, BUT HI SIGNAL SENT BY MALFUNCTION OF CLCS".

| Row | Node 1 | Node 54 | Node 73 | Node 74 | Node 75 | Node 76 |
|---|---|---|---|---|---|---|
| 1 | -1 | 1 | 0 | 0 | 0 | 0 |
| 2 | -1 | 1 | 1 | 0 | 0 | 0 |
| 3 | -1 | 1 | 0 | 1 | 0 | 0 |
| 4 | -1 | 1 | 0 | 0 | 1 | 0 |
| 5 | -1 | 1 | 0 | 0 | 0 | 1 |

Node 1 in this table is the HI HI signal output node, and node 54 is the HI signal output node (see Figure 7 ). Nodes 73, 74, 75, and 76 are those nodes which indicate maintenance of measurement channel A, B, C, or D, respectively; that is, a '1' under one of these nodes indicates that channel is under maintenance. The only TOP event we present here is the one with HI pressure signal output; thus, no attention is given to the HI HI signal channels, and a '-1' is assigned to node 1 in all cases; however, for other TOP events, node 1 would be set to zero or one, depending on the specific event. Row 1 represents the situation when no channel is under maintenance. Thus, all four maintenance columns have been set to '0'. The '1' at node 73 in row 2 indicates that measurement channel A is under maintenance. Rows 3, 4 and 5 similarly represent maintenance of channels B, C, and D.

Under this TOP event, the only boundary condition necessary is normal pressure in the containment. Other TOP events would be accompanied by different boundry conditions; however, the component library of decision tables and the system description would remain the same throughout the analysis. This characteristic points out one convenience of using

such a methodology for constructing fault trees for multiple TOP events of a system.

Components in the consequence limiting control system include pressure transducers, comparators, relays, circuit breakers, annunciators, "OR" logic gates, etc. Furthermore, the Operator has been treated as an additional component of the system in this example. The decision table of an operator is given as Table 12. The input to the operator is the information transmitted to him in order to allow him to take action. In this example, we assume that the basic information he receives is from the annunciator. An input signal of '0' indicates normal pressure signal, '1' is a HI pressure signal and '2' is a HI HI pressure signal. The internal mode is separated into two states. A zero means that the operator takes the correct action. '5003' for the internal state of the operator represents an incorrect action by the operator. The output of the operator in this example is connected to the four trains of the CLCS system; a '0' represents no action by the operator, '1' means that the operator has initiated the HI signal trains, and '2' means the HI HI signal trains have been triggered by the operator. The assumption has been made here that the operator takes the same action on both trains in the same set, e.g., if the operator has pushed the button of the HI signal train, he will push the button of the other HI signal train at the same time; however, this table could be very simply expanded (by the addition of more rows) to include other possible combinations of train actuation.

TABLE 12. DECISION TABLE OF OPERATOR

| Row | Input | Operator | To HI Signal Train Out 1 | Out 2 | To HI HI Signal Out 3 | Out 4 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 1 | 1 | 0 | 0 |
| 3 | 2 | 0 | 0 | 0 | 2 | 2 |
| 4 | 0 | 5003 | 1 | 1 | -1 | -1 |
| 5 | 0 | 5003 | -1 | -1 | 2 | 2 |
| 6 | 1 | 5003 | 0 | 0 | -1 | -1 |
| 7 | 2 | 5003 | -1 | -1 | 0 | 0 |
| 8 | 1 | 5003 | -1 | -1 | 2 | 2 |
| 9 | 2 | 5003 | 1 | 1 | -1 | -1 |

With this basic information, the decision table of the operator, as shown in Table 12, can be interpreted as follows: Row 1 to row 3 represent correct operator actions, e.g., if no signal is sent to the operator, he will not send a signal to any of the four trains. Rows 4 through 9 are those cases in which the operator 'fails', (those cases in which the operator does not take the correct actions).

Finally, Table 13 shows the component node-numbering table for the consequence limiting control system as input to the CAT code. Since the four measurement channels are composed of the same set of components, only channel A has been further developed in the system fault tree. It is assumed the other three measurement channels have the same fault tree construction as channel A. Referring to this table and Figure 7, and using the HI-HI signal comparator in measurement channel A as an example (component 12 in the table), the input defines this component

as "type 017" (i.e., using Decision Table number 17), with input node 27 and output node 23. This output node is connected to the input of the relays (components 5, 6, and 71). Several of the components are considered as "dummy" components, (e.g., components 28-31 and 36-52), as discussed in Section 2.2.5. This is due to the requirement that each component input node be connected to a valid output node from some component. Component 2, for example, requires an input at node 6 to represent the reset signal for the CLCS. Since no detailed treatment is given this signal, and it is not required for evaluation of the TOP event of interest, the output from a dummy component (Component 28) is connected to this node. This provides the required output at node 6; furthermore, since this dummy component type (type 009) has no inputs, component 28 requires no further connections. Notice that all dummy components are of type 009 and are treated in the same manner.

### 4.2.3 Discussion of Fault Tree

Figure 8 shows the fault tree of the Consequence Limiting Control System constructed by the CAT code for the TOP event "containment pressure normal but HI signal sent by malfunction of CLCS". The inhibit gates below gates 2, 12, 24, 38, and 54 represent the five maintenance possibilities. The branch below gate 72 represents a HI signal sent by the operator, which may be caused by human error or by a malfunction of the system. This branch can be seen beneath a number of gates (as transfer symbol 72).

One of the features of this fault tree can be seen by comparing the two different subtrees under gates 7 and 17, which represent the same event "0 VDC at node 49". Gate 7, under the inhibit gate "no measurement

Table 13. Component Index Input Printout for CLCS

CONSEQUENCE LIMITING CONTROL SYSTEM

COMPONENT INDEX INPUT PRINTOUT

| COMPONENT INDEX | CARD CODE | PRINTOUT NAME | TYPE | INPUT/OUTPUT NODES | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | COM1 | OR2A | C14 | 2 | 3 | 1 | | |
| 2 | COM2 | SR2A | 016 | 59 | 6 | 2 | | |
| 3 | COM3 | MANN2A | 014 | 4 | 5 | 59 | | |
| 4 | COM4 | 3/4 2A | C15 | 77 | 78 | 79 | 80 | 4 |
| 5 | COM5 | RA1 | 004 | 23 | 7 | | | |
| 6 | COM6 | RA2 | 004 | 23 | 8 | | | |
| 7 | COM7 | RA4 | C04 | 61 | 9 | | | |
| 8 | COM8 | RA5 | 004 | 61 | 10 | | | |
| 9 | COM9 | 3/4 2B | 015 | 81 | 82 | 83 | 84 | 43 |
| 10 | COM10 | 3/4 1A | C03 | 85 | 86 | 87 | 88 | 49 |
| 11 | COM11 | 3/4 1B | 003 | 89 | 90 | 91 | 92 | 53 |
| 12 | COM12 | COMPAHH | C17 | 27 | 23 | | | |
| 13 | COM13 | COMPAHI | 005 | 65 | 61 | | | |
| 14 | COM14 | TRANSAHH | C18 | 31 | 27 | | | |
| 15 | COM15 | TRANSAHI | C06 | 31 | 65 | | | |
| 16 | COM16 | SENCKTA | 007 | 35 | 31 | | | |
| 17 | COM17 | PTA | 008 | 39 | 35 | | | |
| 18 | COM18 | SENA | C09 | 39 | | | | |
| 19 | COM19 | SR2B | 016 | 50 | 45 | 3 | | |
| 20 | COM20 | MANN2B | C14 | 43 | 44 | 60 | | |
| 21 | COM21 | OR1A | C01 | 46 | 50 | 54 | | |
| 22 | COM22 | SR1A | 012 | 47 | 56 | 46 | | |
| 23 | COM23 | SR1B | 012 | 51 | 58 | 50 | | |
| 24 | COM24 | MANN1A | 011 | 48 | 55 | 47 | | |
| 25 | COM25 | MANN1B | 011 | 52 | 57 | 51 | | |
| 26 | COM26 | BCK1A | C13 | 49 | 45 | | | |
| 27 | COM27 | BCK 1B | C13 | 53 | 52 | | | |
| 28 | COM28 | DUM1R | 009 | 6 | | | | |
| 29 | COM29 | DUM2R | C09 | 45 | | | | |
| 30 | COM30 | DUM3R | 009 | 56 | | | | |
| 31 | COM31 | DUM4R | C09 | 58 | | | | |
| 32 | COM32 | OPERATOR | C02 | 69 | 55 | 57 | 5 | 44 |
| 33 | COM33 | ANNC HII | 023 | 70 | 71 | | | |
| 34 | COM36 | DUM11 | C09 | 11 | | | | |
| 35 | COM37 | DUM15 | C09 | 15 | | | | |
| 36 | COM38 | DUM19 | 009 | 19 | | | | |
| 37 | COM39 | DUM12 | 009 | 12 | | | | |
| 38 | COM40 | DUM16 | 009 | 16 | | | | |
| 39 | COM41 | DUM20 | 009 | 20 | | | | |
| 40 | COM42 | DUM13 | 009 | 13 | | | | |
| 41 | COM43 | DUM17 | 009 | 17 | | | | |
| 42 | COM44 | DUM21 | 009 | 21 | | | | |
| 43 | COM45 | DUM14 | 009 | 14 | | | | |
| 44 | COM46 | DUM18 | 009 | 18 | | | | |
| 45 | COM47 | DUM22 | 009 | 22 | | | | |
| 46 | COM49 | DUM49 | 009 | 73 | | | | |
| 47 | COM50 | DUM50 | 009 | 74 | | | | |
| 48 | COM51 | DUM51 | 009 | 75 | | | | |
| 49 | COM52 | DUM52 | 009 | 76 | | | | |
| 50 | COM53 | TEST6 | 021 | 7 | 73 | 77 | | |
| 51 | COM54 | TEST7 | 021 | 8 | 73 | 81 | | |
| 52 | COM55 | TEST8 | 021 | 11 | 74 | 78 | | |
| 53 | COM56 | TEST9 | 021 | 12 | 74 | 82 | | |
| 54 | COM57 | TEST10 | 021 | 15 | 75 | 79 | | |
| 55 | COM58 | TEST11 | 021 | 16 | 75 | 83 | | |
| 56 | COM59 | TEST12 | 021 | 19 | 76 | 80 | | |
| 57 | COM60 | TEST13 | 021 | 20 | 76 | 84 | | |
| 58 | COM61 | TEST14 | 022 | 9 | 73 | 85 | | |
| 59 | COM62 | TEST15 | 022 | 13 | 74 | 86 | | |
| 60 | COM63 | TEST16 | 022 | 17 | 75 | 87 | | |
| 61 | COM64 | TEST17 | 022 | 21 | 76 | 68 | | |
| 62 | COM65 | TEST18 | 022 | 10 | 73 | 89 | | |
| 63 | COM66 | TEST19 | 022 | 14 | 74 | 90 | | |
| 64 | COM67 | TEST20 | 022 | 18 | 75 | 91 | | |
| 65 | COM68 | TEST21 | 022 | 22 | 76 | 92 | | |
| 66 | COM69 | ANNCHIHI | 010 | 9 | 10 | 72 | | |
| 67 | COM70 | AND ANC | C24 | 71 | 72 | 69 | | |
| 68 | COM71 | RA3 | 004 | 23 | 70 | | | |

END

channel under maintenance," will send a 'HI' signal whenever three-out-of-four measurement channels send a 'HI' signal to it. As shown in Figure 8, gates 18, 30, 44 and 60, under gate 7, represent a 'HI' signal sent by any three of the four measurement channels. Gate 30, for example, receives 'HI' signals from channels A, C and D, where only channel A has been further developed (see transfer 76). On the other hand, gate 17 under the inhibit gate "measurement channel A under maintenance", will send a HI signal if any two of measurement channels B, C and D send a 'HI' signal to it. Since channel A is already under maintenance, the boundary condition "signal = 1 at node 73" has already been set by row 2 of the TOP event Table 10. Thus, the three-out-of-four logic becomes two-out-of-three.

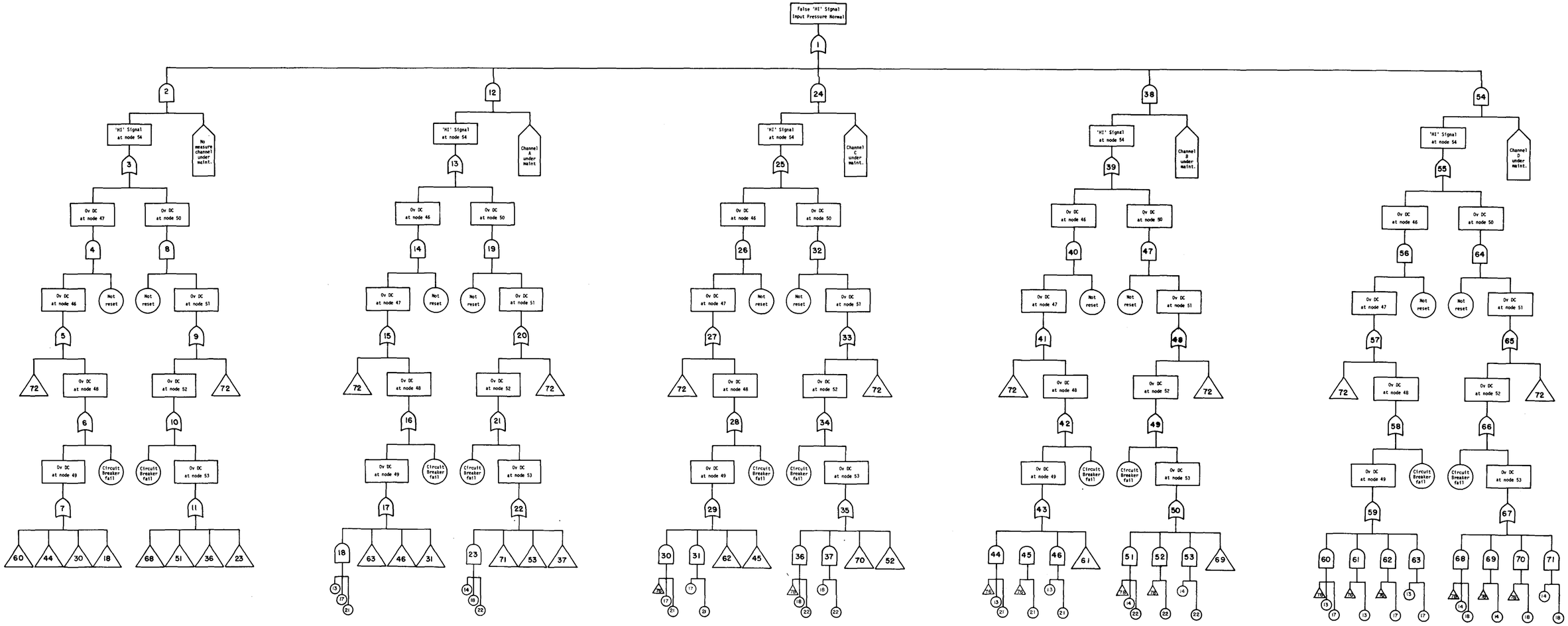



Figure 8. Fault Tree of Consequence Limiting Control System (page 1)

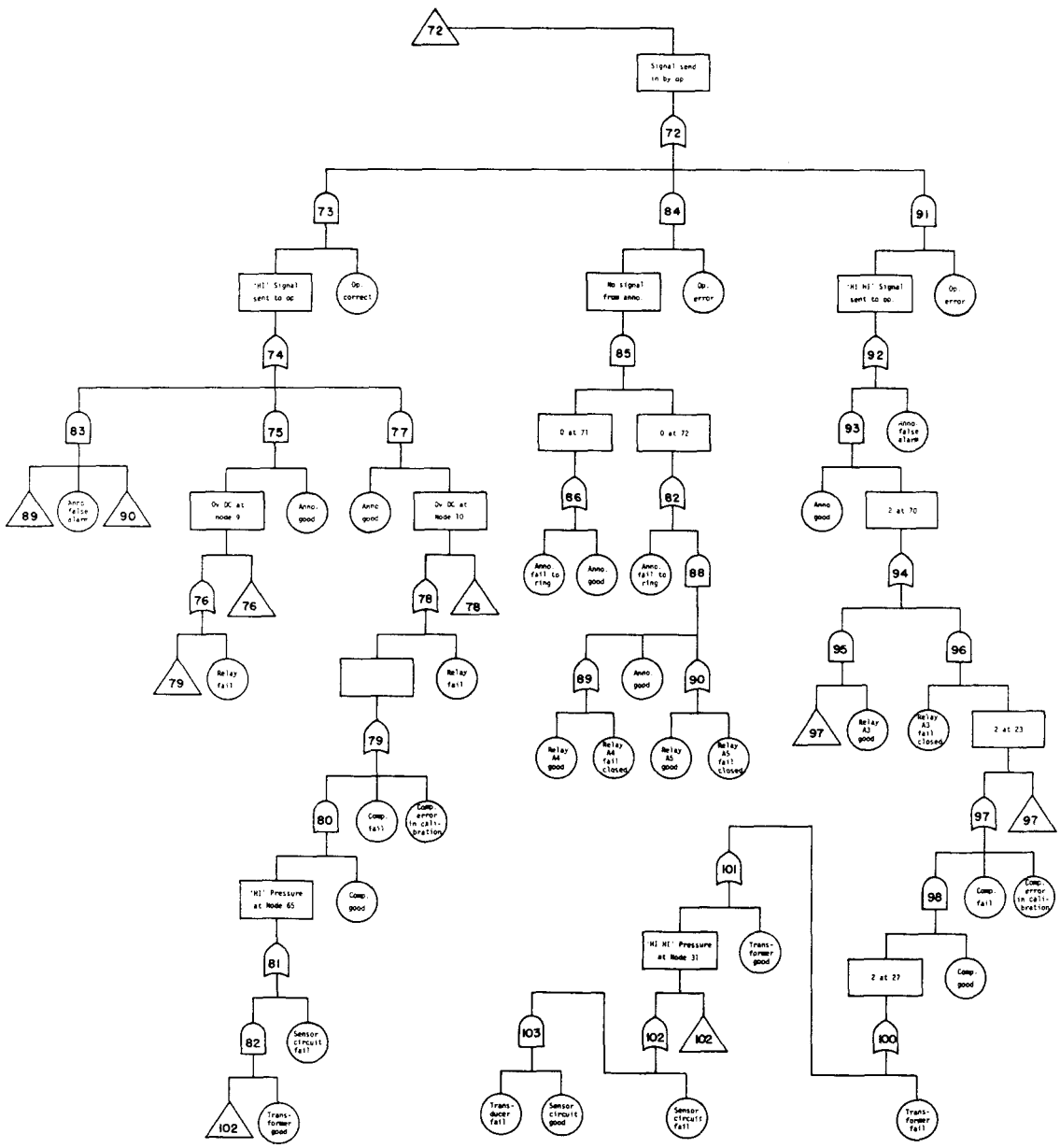



Figure 8. Fault Tree of Consequence Limiting Control System (page 2)

## 5. CONCLUSIONS

This report has presented a discussion of all of the basic information needed by a user of the CAT code. The applications included here, in combination with those of ref. [1], will serve as examples to guide the user in setting up a system, and developing his own component models. Further details of the code, the modeling of decision tables, and a complete sample case with input and output will be found in the appendices as additional aids.

The simple, tabular decision table form for modeling component behavior allows the user to develop both simple and highly detailed component models to produce fault trees of varying complexity. This is the stage of the analysis in which much care must be exercised and, in fact, this represents a major area in which errors may be introduced via faulty or incomplete tables. Furthermore, the user must always bear in mind that failure rates for many specific failure modes of components are generally not available and must use judgment in order to avoid the development of too detailed decision tables, which will prohibit a probabilistic analysis of the fault trees so produced.

The flexibility of the CAT methodology in defining TOP events of any logocal complexity was demonstrated via specific examples. Thus, the modeling of inhibit gates showed a general technique for dealing with mutually exclusive events, such as are encountered in test and maintenance schemes of redundant systems. This approach was also shown to be useful in situations where the basic configuration of the system changes as a function of time. This is especially important in nuclear reactor applications where decay-heat levels, and therefore

cooling requirements, are reduced with time. Another example of the flexibility of the code is the option of producing the fault tree in a punched output format directly compatible with the PREP-KITT codes.

One final point must be emphasized to the prospective user of CAT. Although the computer automation of fault tree construction can greatly speed up this phase of FTA, and readily allow the construction and investigation of very detailed fault trees, it should not be viewed as a replacement for the analyst's efforts. The insight and understanding provided by the analyst himself in carefully setting up the system and component decision tables, as well as in choosing the appropriate TOP events, cannot be supplied by the computer. Furthermore, the fault tree so produced can only be as accurate and complete as the input provided by the analyst. If the CAT code is viewed in this light, it can be seen as a useful tool in assisting the fault tree analyst, and will become even more effective when combined with care and foresight in its use.

## REFERENCES


1. Salem, S. L., G. E. Apostolakis and D. Okrent, A Computer-Oriented Approach to Fault-Tree Construction, EPRI NP-288, Palo Alto, November 1976.

2. Salem, S. L., G. E. Apostolakis and D. Okrent, "A New Methodology for the Computer-Aided Construction of Fault Trees," Annals of Nuclear Energy, 4 (1977) 417-433.

3. Vesely, W. E., and R. E. Narum, PREP and KITT: Computer Codes for the Automatic Evaluation of a Fault Tree, Idaho Nuclear Corporation, Idaho Falls, Idaho, IN-1349, 1970.

4. Fussell, J. B., "A Formal Methodology for Fault Tree Construction," Nucl. Sci. and Engr., 52 (1973) 421-432.

5. U.S. Nuclear Regulatory Commission, Reactor Safety Study, WASH-1400 (NUREG-75/014), October 1975.

# APPENDIX A

## CODE STRUCTURE AND SUBROUTINE FUNCTIONS

### A.1 Code Structure

The basic operation of the CAT code can be broken down into the following functions:

1) Input data, check for errors, and set up basic array structure.

2) Construct and edit fault tree.

3) Print input edit, fault tree construction and editing phases, and final tree.

4) Punch output (if desired) for use with PREP-KITT codes.

The code consists of a main program and eight subroutines, organized as shown by the subroutine calling sequence in Figure A-1. Notice that the major operations of the code are actually performed by subroutine DRIVER. The functional organization of the code is illustrated by the flowchart in Figure A-2, which is essentially a flowchart of this subroutine. The function of each of these routines will be discussed in the remainder of this appendix. In addition, flowcharts for the major functions of each subroutine will be provided in order to illustrate the basic structure and operation of each subroutine. These flowcharts will contain program statement numbers for use when referring to the actual FORTRAN program itself. Although these flowcharts are not intended to illustrate the specific programming details of each routine, they can be used, along with the comment cards contained in the program itself, to assist the user in following the FORTRAN programming, if desired.

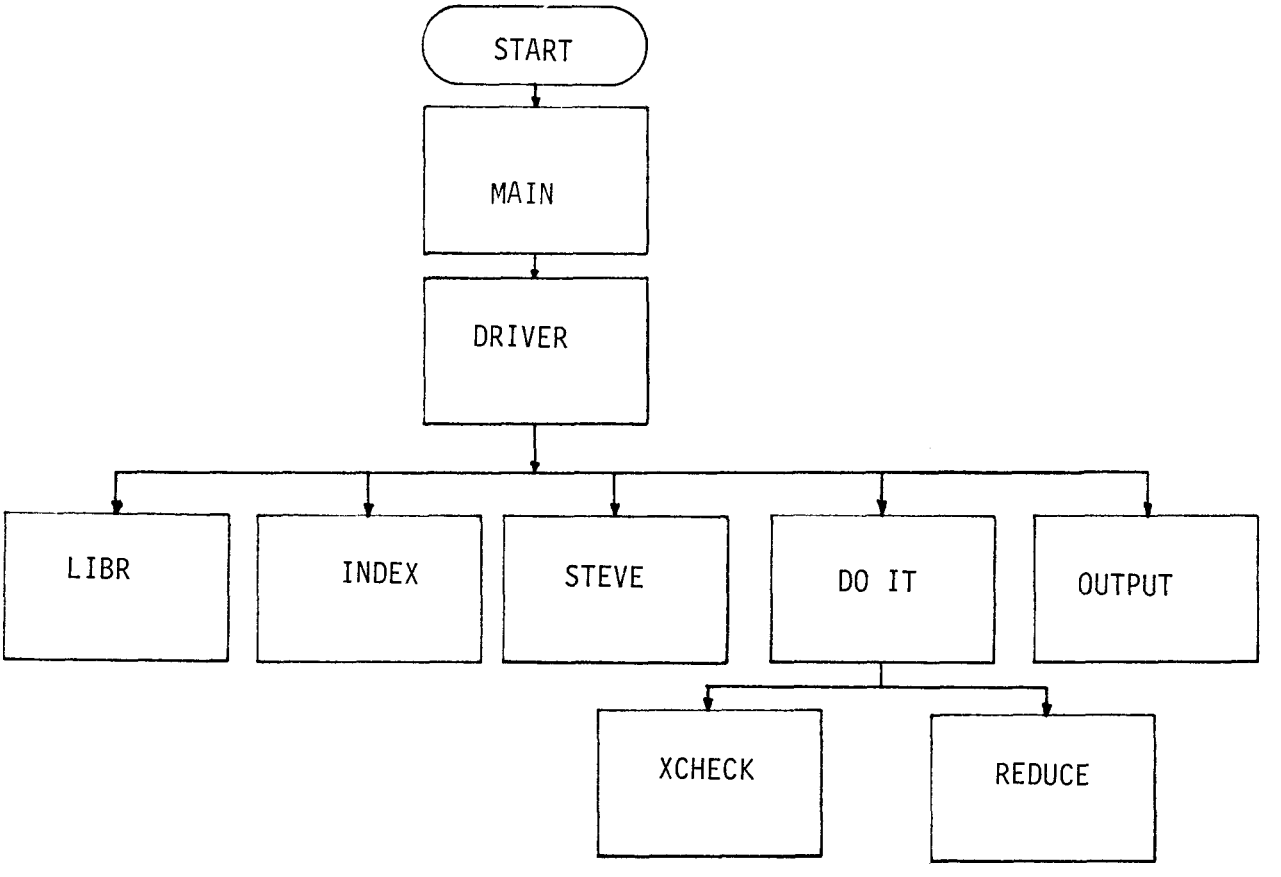



Figure A-1. Subroutine Calling Sequence

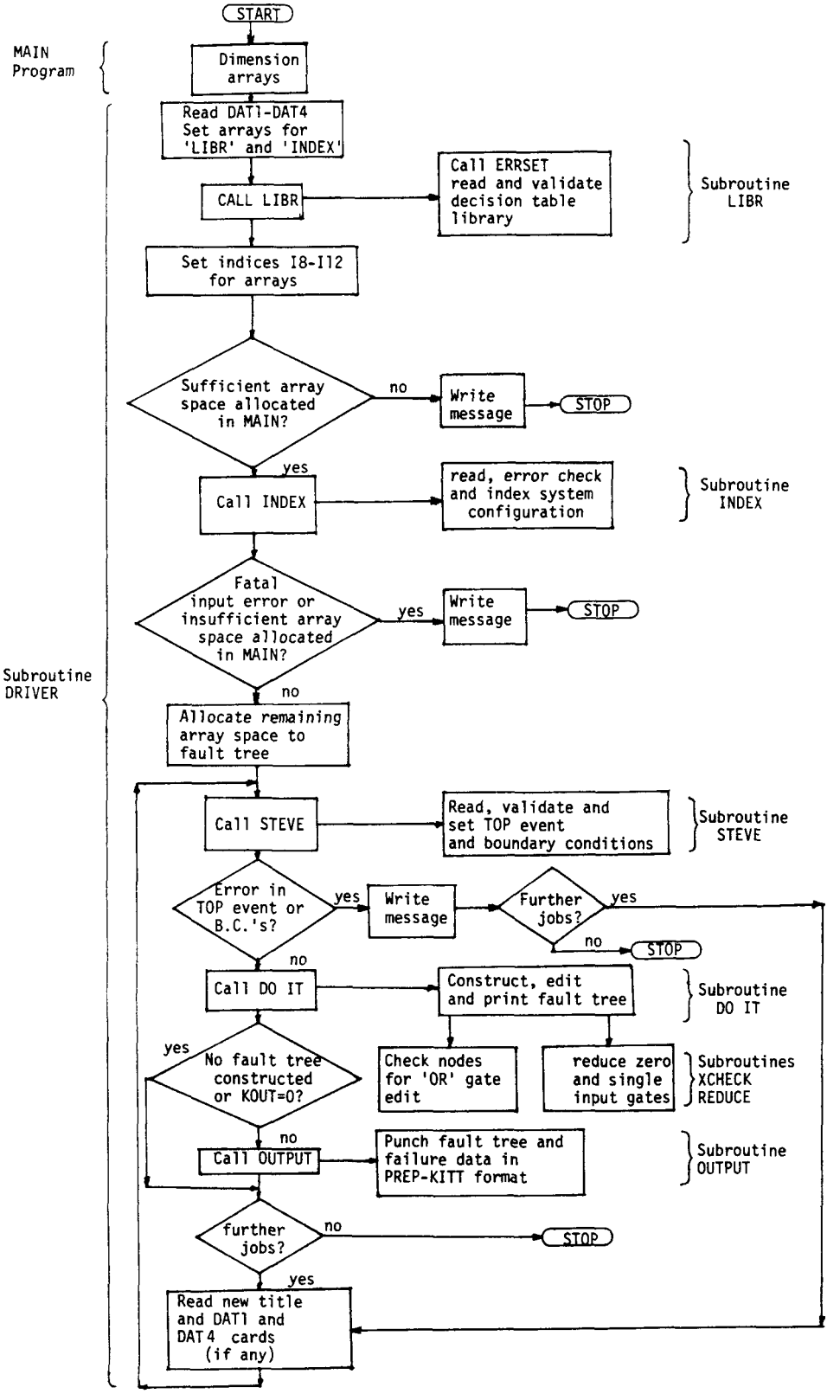



Figure A-2. Flowchart for CAT Code

## A.2 System and Component Node Organization

Before discussing the program structure itself, it will be useful to describe how the CAT code utilizes the system node numbering scheme (Section 2.1) and integrates the component internal modes into this scheme. As can be seen from Sections 2.1 and 2.2.7, states and boundary conditions can be described analogously for both system nodes and component internal modes. This suggests an identical method for treating both within the code.

The approach employed within CAT is to combine both system nodes and component internal modes into a single numbering scheme, and to define program variables which can then represent both of these. The first step is to determine the largest system node used by the system (as input on the 'COM' cards). This is defined by the variable MNODE. The first internal mode (internal column of the decision table) of the first component is then assigned the node number MNODE +1. If there are NNCMP components, with a maximum of MXINT2 internal columns per component (as determined by subroutine LIBR) then there will be NNCMP*MXINT2 nodes required to index all internal modes, producing a total of MNODE + NNCMP*MXINT2 nodes in all. This number is defined as the variable NODES. Notice that each component will be assigned exactly MXINT2 internal nodes, even though it may not require this many. For components with fewer internal columns, only the first of these nodes will actually be used, with the remainder left blank. This simplifies the numbering scheme, by allowing the jth internal column of the ith component to be assigned the node number:

$$\text{MNODE} + (i-1)*\text{MXINT2} + j.$$

Referring to the sample case in Appendix C, MNODE = 11 and MXINT2 = 2. As seen from the seventh page of the output (Appendix E), the single internal column of component 1 is given node number 12, with node number 13 not used; the single internal column of component 2 is then numbered 14, etc.

With all inputs, internal modes and outputs of each component now numbered, the array INODE (j,i) is used to store the j node numbers of the ith component. For component 1 of the sample case, which has no inputs, one internal node (12) and only one output (node 1),

INODE (j,1) = (12, 1).

For component 6, which has two inputs (nodes 5 and 6), two internal nodes (22 and 23), and one output (node 7),

INODE (j,6) = (5,6,22,23,7).

Finally, the array X is used to indicate the state existing at any system or internal node. Each node is represented by two entries: X(1,i) is set to the state existing at node i, and X(2,i) indicates the gate beneath which this state was set. All entries are initialized to -1, which indicates no defined state in existence. Boundary conditions are indicated by X(2,i) = 0. Whenever a branch of an OR gate is completed, all states set by events beneath that branch must be reset to -1 (see Sections 3.3.2 and 3.4 of reference A-1). Since higher numbered gates are always input into lower numbered gates, this corresponds to resetting only those nodes for which X(2,i) is greater than or equal to the number of the current OR gate. Thus, boundary conditions will never be reset, nor will the events set by higher order gates (gates above the OR gate in question).

## A.3 MAIN Program and Program Dimensioning

The MAIN program has only two functions: to set up the program dimensions and to call subroutine DRIVER, which effectively controls the rest of the code operation. The CAT code has two basic arrays:

1) array MAT (Dimension = NSIZE):
   an integer array which contains all program integer array elements, and
2) array NAME (Dimension = LSIZE):
   a double precision array which contains all alphanumeric names within the program.

These two arrays will be split up into 15 and 3 arrays respectively in subroutine DRIVER and, by means of variable dimensions, these sub-arrays can be redimensioned within each job. However, the two arrays within the MAIN program itself must contain fixed dimensions. This is done by the following three cards in MAIN:

```
DIMENSION MAT (nnnn)
DOUBLE PRECISION NAME (ℓℓℓℓ)
DATA LSIZE, NSIZE/ℓℓℓℓ, nnnn/
```

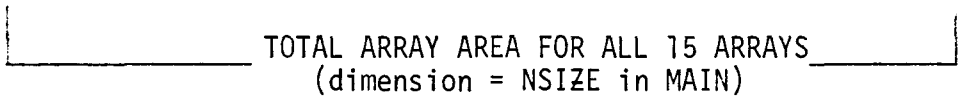
where 'nnnn' is the (fixed) dimension of MAT, and 'ℓℓℓℓ' is the (fixed) dimension of NAME. These two dimensions must be set by the user before the code is compiled (nnnn = 3000 and ℓℓℓℓ = 200 are sufficient for small systems and fault trees). If any systems or fault trees are encountered which are too large for these dimensions, suitable warnings will be printed out by the code. The user need only change the three cards shown above and recompile the MAIN program (6 FORTRAN statements) in order to run any size job. Note that nnnn = 20,000 will accomodate the largest fault tree shown in this report. However,

the only limits to system and fault tree sizes are those of the user's own computer installation itself.

## A.4 Subroutine DRIVER and Sub-Array Allocation

Subroutine DRIVER reads the program control data (see Section 2.2.3), sets the dimensions of the sub-arrays, and calls the remaining subroutines. Its flowchart is basically that shown in Figure A-2.

Once the program control data have been read in, the dimensions for the first seven integer arrays and the three alphanumeric arrays are determined. In order to conserve space, all major program arrays (15 in all), are combined in the one large fixed-dimension array of the MAIN program with the dimensions of the individual arrays expanding at times to fill all available space, and contracting later to the exact size necessary as determined by the program. This also allows the program to determine at any point whether sufficient space has been allocated by the main program. As an example, the figure below represents the complete array area required by subroutine LIBR, which uses only the first 7 of the 15 total arrays:




The exact dimensions of the first 6 arrays are known and are thus fixed. However, the size of array 7, which stores the component type library, is only an estimate, and thus the program allows this array to expand and fill the space which will be required for arrays 8-15 later. Once subroutine LIBR has finished, and the component type library is completed, its exact size will then be known, and array 7 will be appropriately redimensioned downward, thus allowing maximal room for the remaining arrays.

At this point, arrays 8-11 can now be dimensioned, as needed for

input of the system diagram by subroutine INDEX. Furthermore, the proper dimension for array 12 is returned from subroutine INDEX. The last three arrays, which will contain the fault tree itself, can now be dimensioned to fill up the remaining space. These three arrays, IGATE, JGATE and GATE require 1, 1 and (5 + 2*XX) entries per completed gate, where XX is the average number of inputs per gate (XX = 3 in subroutine DRIVER). Thus each gate will require 13 storage spaces, and the three arrays are appropriately dimensioned to use up the total remaining array space.

In a similar manner the three alphanumeric arrays (double precision) are dimensioned to partition the available array space. Table A-1 contains a listing and definition of the 15 integer arrays and dimensions while A-2 contains a similar listing for the alphanumeric arrays.

At this time all library and system data have been read in and checked for errors. If any fatal errors have been detected, (JERR $\neq$ 0), or if no space was left to allocate for arrays IGATE, JGATE and GATE, the program will terminate. If no errors have been found, subroutine DRIVER will call subroutine STEVE, which inputs the TOP event and boundary conditions.

In the event that STEVE detects a fatal error in the TOP or boundary conditions (IERR < 0), this specific tree cannot be constructed. However, if the library and system data were correct (JERR = 0), DRIVER will proceed to any subsequent job input, and search for a new TOP event and boundary conditions.

Whenever a valid TOP event and boundary conditions are found, subroutine DO IT is called to construct the actual fault tree. Should a

TABLE A-1 INTEGER ARRAYS

| No. | Name | Dimension | Description |
|---|---|---|---|
| 1 | NTYPE | NLIB* | The 'type number of the ith component type. In the sample case (Appendix C), NTYPE(1) = 101. |
| 2 | IROW | NLIB | Location of first row of ith component type decision table in array JROW. IROW(1) = 1. If table 1 has M rows, IROW(2) = M+1, etc. Note: IROW(NLIB) = location of first row of decision table for TOP event. |
| 3 | NINT | NLIB | Number of internal failure columns for ith component type. |
| 4 | NIN | NLIB | Number of inputs for ith component type. |
| 5 | NOUT | NLIB | Number of outputs for ith component type. |
| 6 | NROW | NLIB | Number of rows in ith component type decision table. |
| 7 | JROW | LNROW, MXNROW | Two-dimensional array which contains the rows of the decision tables. There are 'MXNROW' rows, each with $\leq$ 'LNROW' entries. Location of first row of ith table is given by IROW(i). Thus the first entry in the ith table is JROW (1,IROW(i)). |
| 8 | NCMP | NNODE | Number of component whose output is connected to node "i". In the sample problem, NCMP(2) = 4 (the output of the 4th component is connected to node 2). |
| 9 | MOUT | NNODE | For components with multiple outputs, MOUT(i) specifies which output of component NCMP(i) is connected to node i. |

*NLIB = 1 + NLIB from DAT2 card, to allow TOP event decision table to be stored following the last component decision table.

Table A-1 (Continued)

| No. | Name | Dimension | Description |
|---|---|---|---|
| 10 | ITYPE | NNCMP** | Component type of ith component. ITYPE (NNCMP) = "type" number of TOP event decision table (last table in library). |
| 11 | INODE | LNRP1, NNCMP<br><br>(LNRP1 = LNROW + 1) | Two dimensional array which contain the node numbers for each input, internal column and output of each component. (See Appendix A.2) |
| 12 | X | (2, NODES) | Two dimensional array which contains the system states existing at all nodes. x(1,i) = state at ith node; x(2,i) = gate beneath which state was set. (See Appendix A.2). |
| 13 | IGATE | MGATE | Location in array GATE of start of entries for ith gate. IGATE(1) = 1. |
| 14 | JGATE | MGATE | Gate into which ith gate is input. JGATE(1) = 0; JGATE(2) = 1; in sample case, JGATE(3) = 2, JGATE(4) = 2, etc. (JGATE is redefined during final editing). |
| 15 | GATE | NGSIZE*** | Array GATE contains all entries for fault tree. The location of the first entry for the ith gate is given by IGATE(i). The entries for the ith gate are defined and illustrated in Appendix E, and are briefly: gate type (1 = AND, 2 = OR), number of gates input, number of primary inputs, event signal state and node number developed by gate, and pairs of inputs for each input. |

** NNCMP = 1 + NNCMP from DAT 3 card to allow for TOP event.

*** NGSIZE is total remaining space of integer array 'MAT'.

TABLE A-2 ALPHANUMERIC ARRAYS

| No. | Name | Dimension | Description |
|---|---|---|---|
| 1 | NAME | NLIB | NAME(i) is the name of ith component type; NAME(NLIB) is 'TOPEVENT'. |
| 2 | MODNAM | (MAXINT, NLIB) | MODNAM(j,i) is the name of the jth internal mode (column) of the ith component type. |
| 3 | CMPNAM | NNCMP | CMPNAM(i) is the name of the ith component; NAME(NNCMP) is the name of the TOP event from 'TTOP' card. |

complete tree be produced (IERR = 0), subroutine OUTPUT is called if PREP-KITT output is desired. Otherwise, DRIVER will produce any appropriate messages, and search for any input data for a subsequent job. Note that there are a number of reasons that may preclude construction of a complete fault tree by subroutine DO IT. These include defining a TOP event which cannot occur or is "sure to occur" under the stated boundary conditions, or allocating insufficient array space for completion of the tree.

## A.5 Subroutine LIBR

After the program control data is read by DRIVER, subroutine LIBR is called to set up the component library. A simplified flowchart of this subroutine is shown in Figure A-3. Since this subroutine has its own extensive error checking routine, LIBR first overrides the IBM IHC215I error message with a call to the IBM routine 'ERRSET'. This call will suppress the "illegal character" message which results from reading input with the wrong format (such as when an extra card has been inserted). Note that the parameters in this call must be changed in order to run on a CDC computer.

The remaining functions of this subroutine, as shown by the flowchart, are as follows. Following the '&LIB' card, loop 200 is performed once for each of the 'NLIB' decision tables. For each table, the 'LIBR' card, containing the name, type, numbers of inputs, internal columns, outputs and rows of the table, is first read. If this card is missing, appropriate action is taken to search for the next valid card. When a valid 'LIBR' card is encountered, the data contained on it are then validated. This is followed by the 'MOD' card, containing the names of the internal failure columns. Next, the row cards are read by loop 120 followed by an 'END' card, if present.

As indicated by Figure A-3, a large number of error checks are performed by this subroutine. Table A-3 lists the diagnostics produced by subroutine LIBR, along with the FORMAT number and probable cause.

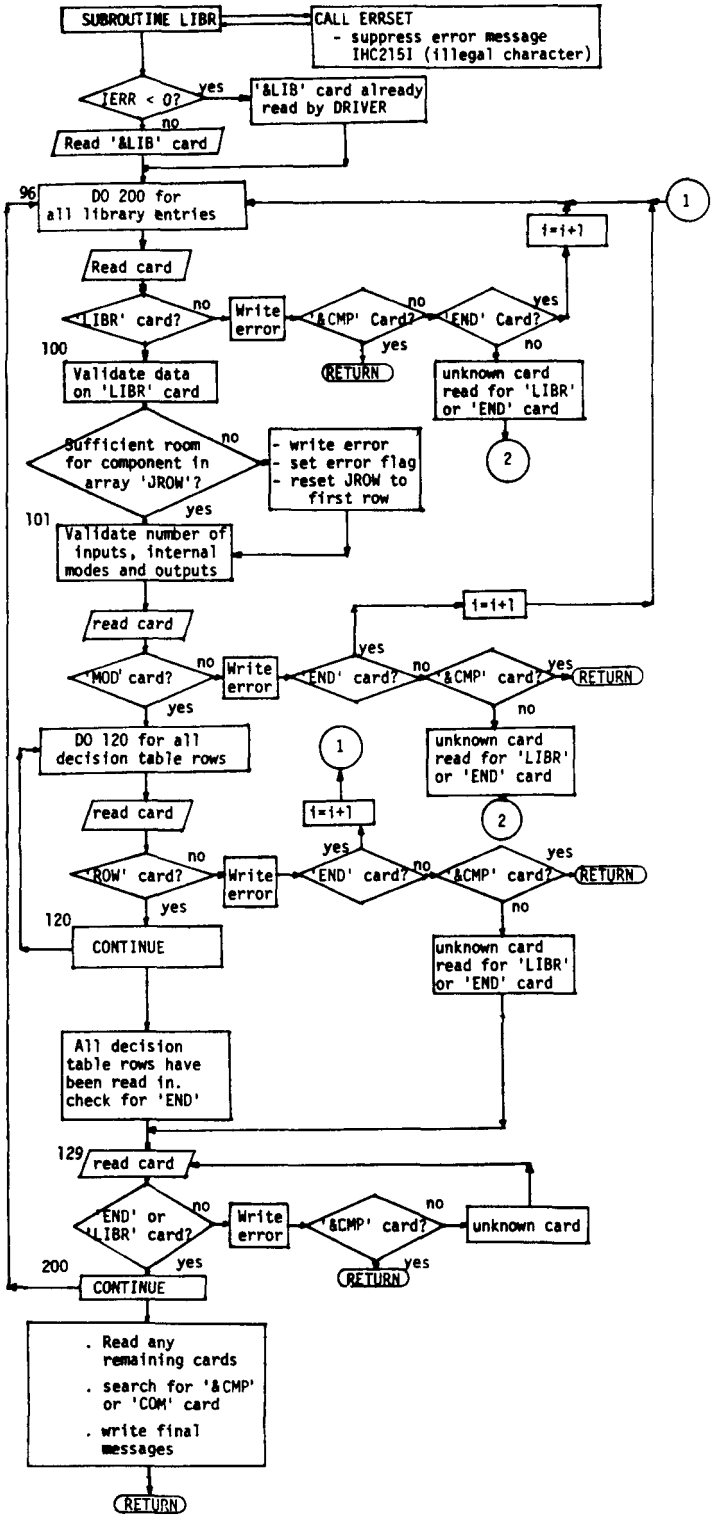



Figure A-3. Flowchart for Subroutine LIBR

TABLE A-3. DIAGNOSTICS PRODUCED BY SUBROUTINE LIBR

| MESSAGE | FORMAT STATEMENT | PROBABLE CAUSE |
|---|---|---|
| CARD MISSING OR MISPUNCHED | 1003 | Card missing, out of order, extra card, or four letter code mispunched. |
| ONLY -- COMPONENT TYPES INPUT | 1005 | Component library incomplete, NLIB incorrect, or error in previous library components. |
| NO VALID HEADER CARDS FOUND | 1011 | Missing system data and TOP event input. |
| NUMBER OF COMPONENT ROWS EXCEEDS NUMBER ALLOCATED | 1012 | Dimension NSIZE in MAIN program too small. |
| COMPONENT -- HAS -- INTERNAL MODES. THIS EXCEEDS THE 'MAXINT' MODES ALLOWED. | 1013 | Component has more internal columns than specified by 'MAXINT' on 'DAT2' card; 'MAXINT' or 'NINT' in error. |
| COMPONENT -- HAS -- INTERNAL MODES + INPUTS + OUTPUTS. THIS EXCEEDS THE 'LNROW' ALLOWED. | 1014 | Rows for current decision table too long; error in 'LNROW' on card 'DAT2', or in 'NIN', 'NINT' or 'NOUT' on 'LIBR' card. |
| EXTRA COMPONENT TYPES INPUT. | 1016 | Error in 'NLIB' on card 'DAT2' extra decision tables in library, or extra 'LIBR' or 'END' cards found. |

## A.6 Subroutine INDEX

Subroutine INDEX reads the system flow chart as input in the '&CMP' section. Its flowchart is shown in Figure A-4, and its basic operation is as follows. After reading the '&CMP' card (or immediately, if '&CMP' is missing), INDEX reads and validates each 'COM' card, one at a time. Block 100 validates the input which is listed there, and Table A-4 lists the error messages which are produced. If any error is detected, an error flag is set for that component (component i), by setting: INODE (LNRP1, i) = -2.

If the component has no errors, the correct node numbers are stored in array INODE (see Section A.2). Finally, extra or missing 'COM' cards are indicated by further error messages (Table A-4), and indexing and cross-referencing tables are printed out.

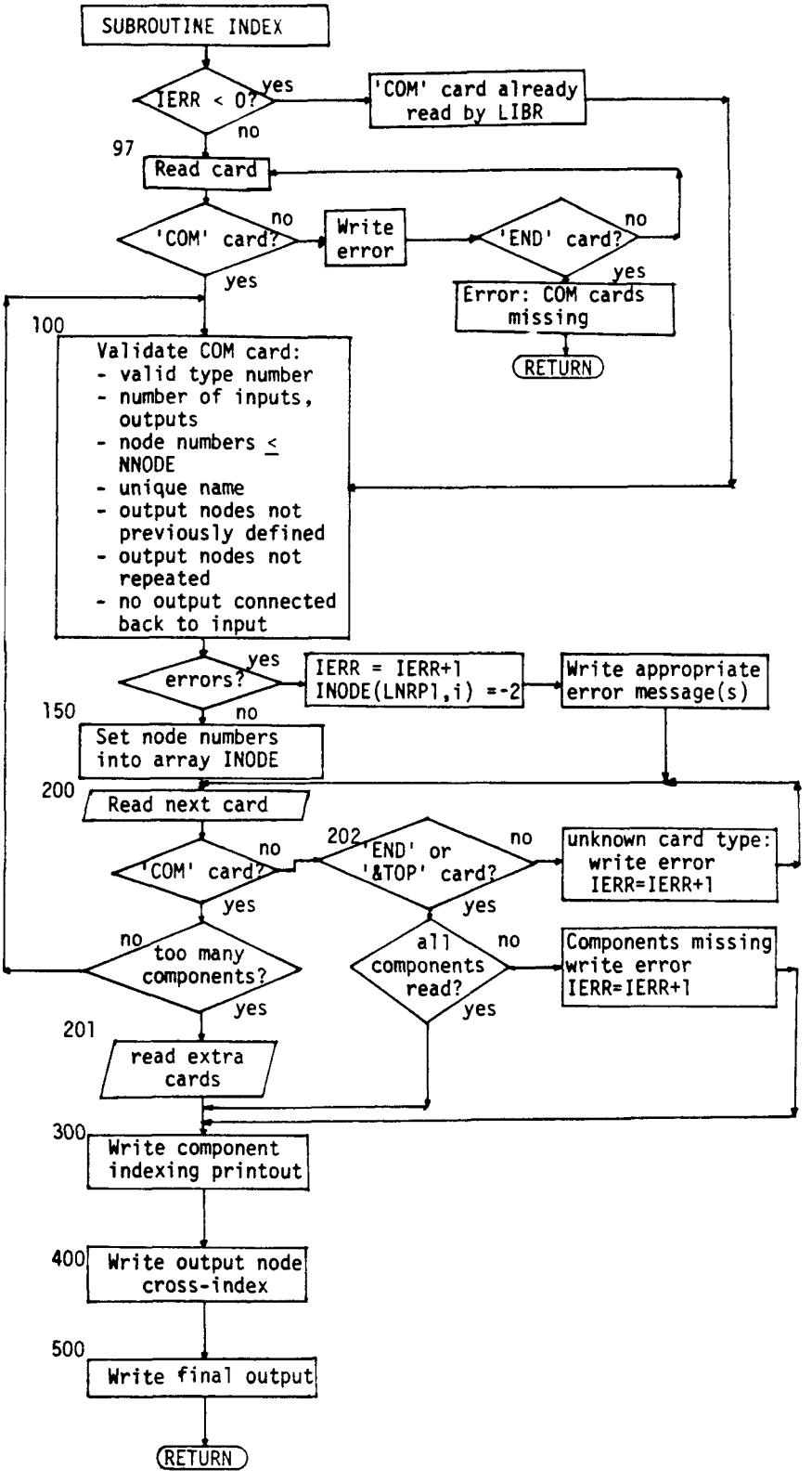



Figure A-4. Flowchart for Subroutine INDEX.

TABLE A-4. DIAGNOSTICS PRODUCED BY SUBROUTINE LIBR

| MESSAGE | FORMAT STATEMENT | PROBABLE CAUSE |
|---|---|---|
| CARD MISSING OR MISPUNCHED | 1002 | Card missing, out of order, extra card, or four letter code mispunched. |
| NO COMPONENT TYPE -- FOUND IN LIBRARY | 1005; 1022 | TYPE number on 'COM' card in error; TYPE number in component library in error; other error in component library or library incomplete. |
| COMPONENT NAME "--" HAS PREVIOUSLY BEEN USED BY COMPONENT --. | 1007 | Duplicate component name, or name mispunched. |
| TOO FEW NODES, OR NON-POSITIVE NODES. | 1008 | Current component has too few nodes, or a blank or negative entry was found; an input or output node has been left undefined; wrong decision table (wrong TYPE number) chosen. |
| TOO MANY NODES. | 1009 | Current component has too many node numbers defined on 'COM' card, or wrong decision table (wrong TYPE number) chosen. |
| ONE OR MORE OUTPUTS IDENTICAL WITH ONE OR MORE INPUTS | 1010 | An input and output for the current component have the same node number: no component may have an output directly connected to one of its inputs. |
| OUTPUT NODES NOT UNIQUE | 1011 | Two or more outputs of one component have same node number: output nodes must be unique. |
| OUTPUT NODE -- HAS ALREADY BEEN ASSIGNED TO COMPONENT -- | 1012 | Two or more components have the same output node number, all output nodes must be unique. |
| MORE THAN THE -- COMPONENTS SPECIFIED HAVE BEEN INPUT | 1013 | Extra components included, or NNCMP on 'DATS' card in error. |

Table A-4. (Continued)

| MESSAGE | FORMAT STATEMENT | PROBABLE CAUSE |
|---|---|---|
| NODE TOO LARGE. MAXIMUM NODE ALLOWED = "NNODE" | 1016 | A node number for current component exceeds 'NNODE' on 'DAT3' card; node number or 'NNODE' in error. |
| ONLY -- COMPONENTS INPUT. -- COMPONENTS EXPECTED. | 1017 | 'COM' card missing or mispunched; 'NNCMP' on 'DAT3' card in error. |
| END CARD FOUND WHERE COMPONENT DATA EXPECTED. | 1026 | Component cards missing, or 'END' card following '&CMP' card. |
| --INPUT NODES REFERENCE UNDEFINED OUTPUT NODES | 1028 | One or more input nodes have no component input into them (see Section 2.2.5); error in node numbering; other error in a component which should have had the node in question as an output node. |

## A.7 Subroutine STEVE

Subroutine STEVE has three main functions, as shown in the simplified flowchart of Figure A-5. These are:

1) number internal nodes,

2) set the TOP event, and

3) set boundary conditions.

First, loop 102 numbers the internal component columns, as discussed in Section A.2. Then, before setting the TOP event and boundary conditions, loop 201 initializes array X (see section A.2). Then, the input is searched for a 'TTOP' card to input the TOP event. Should '&END' be found, or later errors occur, this subroutine will terminate, and subroutine DRIVER will attempt to find a subsequently valid '&TOP' or 'TTOP' card for a succeeding job.

Once a 'TTOP' card has been found, the input parameters shown in block 302 will be validated and, if no errors are found, subroutine STEVE will then read and input the row cards for the TOP event.

If a valid TOP event is found, the final step is to input the boundary conditions, if any. Since both internal and external boundary conditions (component columns and system nodes) may be set, either of two loops, 320 and 330, is used, determined by the code 'INT' or 'EXT' on the card being read. The subroutine then concludes by printing any final messages before returning.

Although the flowchart in Figure A-5 does not present a detailed picture of all the error checks performed, Table A-5 can be consulted for a list of the error messages which may be produced along with their causes. Furthermore, in order to more easily trace the succession of

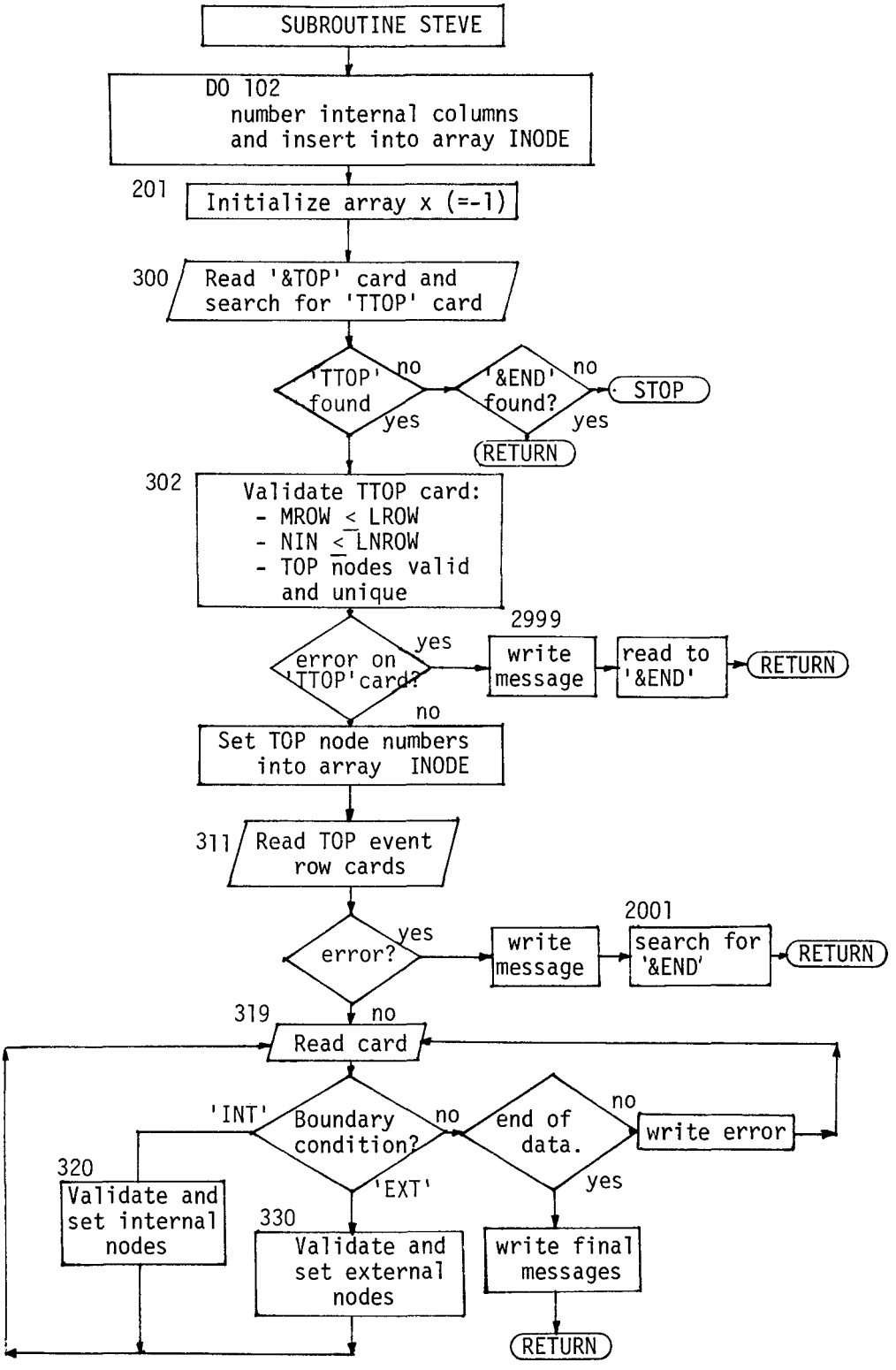



FIGURE A-5. Flowchart for subroutine STEVE.

program statements executed when an error is encountered, the following states of the program flag 'ISET' are set at various stages of the subroutine to indicate the point at which an error occurred:

| | |
|---|---|
| ISET = 0 | Beginning of subroutine |
| ISET = 1 | '&TOP' or 'TTOP' card has been read |
| ISET = 2 | 'TTOP' card has been read and validated |
| ISET = 3 | TOP event has been validated |
| ISET = 4 | '&BC' has been read |
| ISET = 5 | First 'INT' or 'EXT' card has been found. |

TABLE A-5. DIAGNOSTICS PRODUCED BY SUBROUTINE STEVE

| MESSAGE | FORMAT STATEMENT | PROBABLE CAUSE |
|---|---|---|
| CARD MISSING OR MISPUNCHED | 1003 | Card missing, out of order, extra card or four letter code mispunched |
| -- CARD EXPECTED: DATA CARDS MISSING OR MISPUNCHED | 1008 | End of file read where data expected: '&TOP', 'TTOP' or 'TOP' cards missing. |
| NUMBER OF ROWS OF TOP EVENT EXCEEDS SPACE ALLOCATED | 1009 | 'MROW' on 'DAT4' card or 'NROW' on 'TTOP' card in error. 'MROW' must be $\geq$ 'NROW' for all jobs. |
| NUMBER OF NODES OF TOP EVENT EXCEEDS SPACE ALLOCATED | 1010 | 'LNROW' on 'DAT2' card or 'NIN' on 'TTOP' card in error: 'LNROW' must be $\geq$ 'NIN' for all jobs. |
| NODE -- IS NOT UNIQUE | 1011 | Duplicate node numbers on 'TTOP' card. |
| NODE -- IS NOT BETWEEN 1 AND MNODE | 1012 | Non-positive node number, or node number larger than maximum node. |
| NODE -- HAS NOT BEEN DEFINED | 1013 | Node number refers to non-existant node; error on 'TTOP' card or error on previous 'COM' card. |
| END CARD FOUND WHERE "--" CARD EXPECTED | 1017 | Cards missing, out of order, previous error, or extraneous 'END' card. |
| MORE THAN THE -- 'TOP' CARDS SPECIFIED HAVE BEEN INPUT | 1018 | Too many TOP event row cards; error in 'NROW'; also check 'MROW' on 'DAT4' card. |
| COMPONENT "--" DOES NOT EXIST | 1020 | Attempt to set internal boundary condition for nonexistant component, or one with no internal columns; error in name on 'INT' or error on previous 'COM' card. |
| NODE -- IS A TOP EVENT AS WELL AS A BOUNDARY CONDITION. BOUNDARY CONDITION WILL BE IGNORED. | 1021 | Attempt to set a boundary condition at a node defined as the TOP event; error on 'TTOP' or 'EXT' card. |

TABLE A-5. DIAGNOSTICS PRODUCED BY SUBROUTINE STEVE (Continued)

| MESSAGE | FORMAT STATEMENT | PROBABLE CAUSE |
|---|---|---|
| '&END', '&OUT' OR END OF FILE FOUND | 1023, 1025, 1026 | 'END' or data cards missing or mispunched; extra or misplaced '&END' or '&OUT' card. |
| UNEXPECTED '&OUT' READ (KOUT = 0) | 1027 | Unexpected PREP data input; erroneous '&OUT' in data; KOUT omitted or in error on 'DAT1' card. |
| '&END' OR END OF FILE REACHED WITHOUT '&OUT' CARD. REQUIRED PREP DATA MISSING (KOUT = 1). | 1028 | '&OUT' missing or mispunched; PREP data missing; KOUT in error on 'DAT1' card. |

## A.8 Subroutine DO IT

The actual construction of the fault tree is done by subroutine DO IT, along with two subsidiary subroutines, XCHECK and REDUCE. DO IT employs a top-down construction algorithm, which begins with the event(s) defined by the TOP event and constructs the fault tree in a downward direction, by searching through the system for events which may lead to the TOP.

### A.8.1 Gate Construction

The construction methodology is identical to that developed in Chapter 3 of Reference 1, and illustrated in Section 3.4 of the current report. The basic features of the CAT methodology, as implemented in subroutine DO IT, include the following:

1) The construction and editing of the fault tree are broken down into three phases: construction and preliminary editing, intermediate editing, and final editing.
2) The preliminary fault tree construction stage will result in an alternating series of AND and OR gates; however, many gates will later be eliminated by editing, often resulting in a series of identical gate types.

The reason for this alternating sequence stems from the method of utilizing decision tables for the construction of the fault tree. When, at any point in the fault tree, a decision table is being searched for rows which match the necessary conditions, an OR gate will be produced with each matched row as an input. Then, when one of these inputs (a specific row of the table) is being further developed, it will lead to an AND gate, with each entry in the row as an input.

Finally, should any of these entries require further development, this will lead to another decision table with those rows matching the proper state conditions forming an OR gate. Thus, the OR-AND-OR nature of the fault tree construction is a specific result of the nature of the decision tables, in which any "TRUE" row results in a "TRUE" result (OR logic), while each "TRUE" row requires all entries to be "TRUE" (AND logic).

The development of the TOP event and the OR and AND gate construction and preliminary editing phases of subroutine DO IT are illustrated by the flowcharts in Figures A-6 and A-7. Since the TOP event is basically a decision table used to start the fault tree construction process, the initial phases of DO IT merely determine the nature of the TOP gate, and send it to the proper location for further development (Loop 200 for an OR gate, loop 300 for an AND gate). Note that a TOP OR gate automatically implies a multiple input TOP gate (i.e., the TOP event decision table has multiple rows). However, a TOP AND gate might consist of only a single entry, of the form: "state j at node i." This single input gate will be immediately replaced by an OR gate, whose entries are the rows of the decision table of the component whose output is connected to node i, and whose output states are j.

Once the proper gate type for the TOP event has been determined, the construction phase itself begins, and sections 200, 300 and 400 of DO IT will be executed repetitively for each gate. For each OR gate, loop 200 will be performed as follows (Figure A-6). First, the index 'JDEX' will be set to the number of the gate above the current OR gate ("INDEX" is the number of the current OR gate itself).

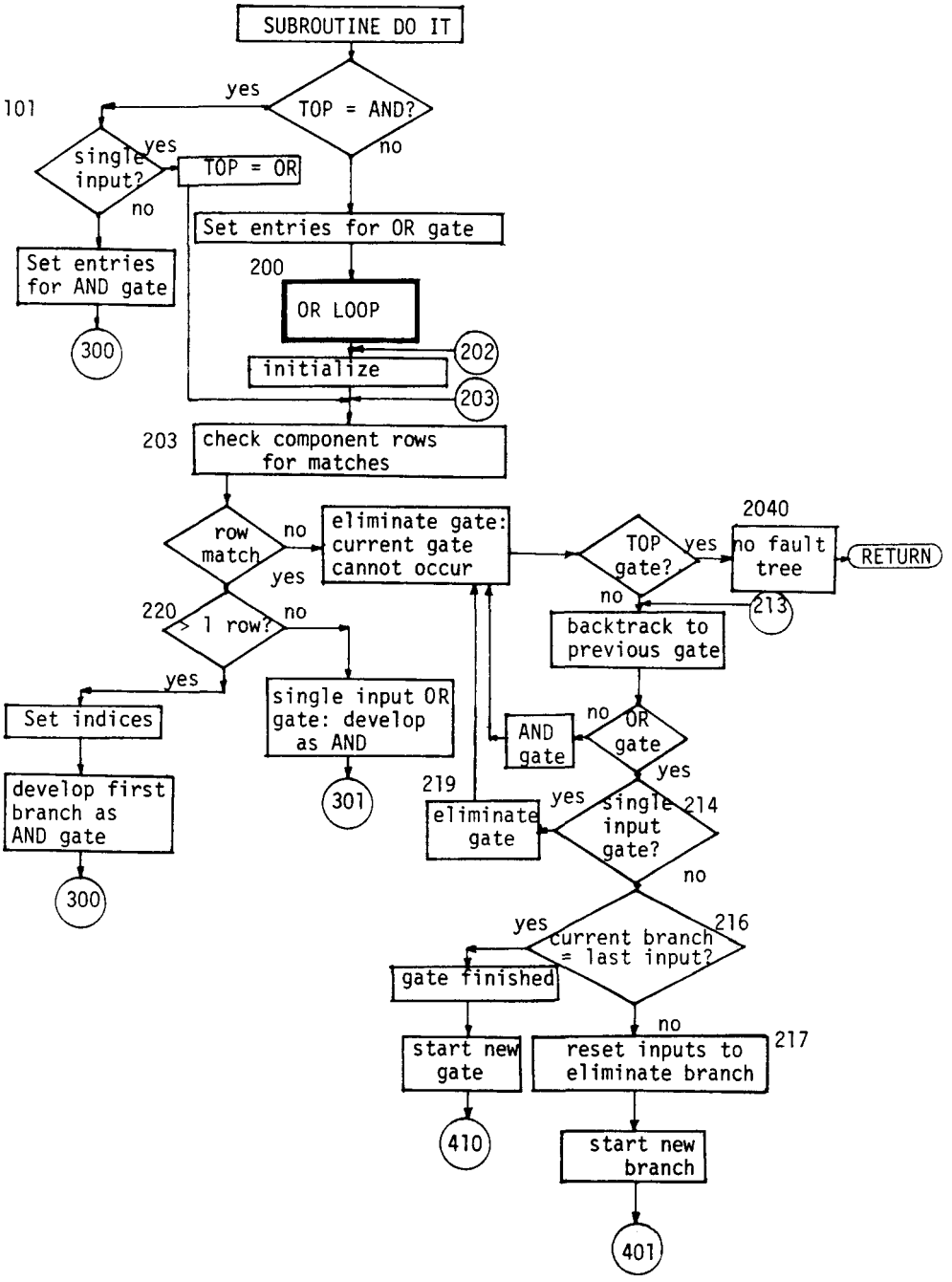



Figure A-6. Flowchart for TOP Event and OR Gate Algorithm of Subroutine DO IT

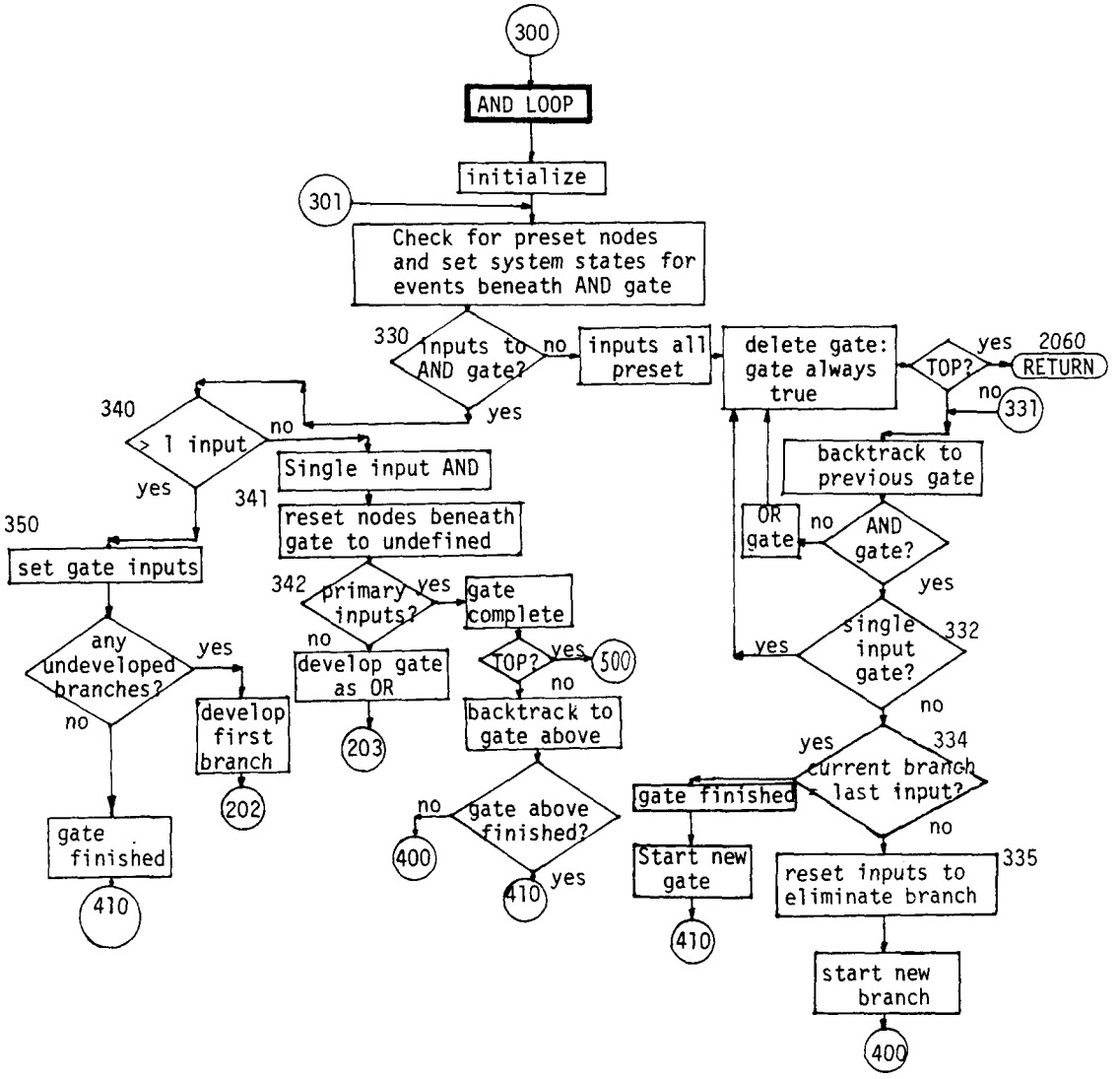



Figure A-7. AND Gate Construction Flowchart of Subroutine DO IT

Then the component whose output is connected to node "JNODE" (as stored in location "KDEX" of gate "JDEX") will be located, and its rows searched for those with the proper output state, "IMODE." Those rows having the correct output state will also be searched, and any whose entries do not contradict any system or component states in existence, will be input into the OR gate. Thus, each row input into the final OR gate will match all current system states in existence.

Three situations may be encountered at this stage:

1) Several rows match the necessary conditions: in this case, the current OR gate has multiple inputs, and the first input will be developed as an AND gate, numbered "INDEX + 1," in section 301.

2) Exactly one row matches: the current OR gate has only one input, and thus will be deleted and replaced by its single input. That is, its single input row will be developed in section 300 as an AND gate, numbered "INDEX," which replaces the single input OR gate, which was numbered "INDEX."

3) No rows match the necessary conditions: that is, the OR gate cannot occur and must be eliminated. If this is the TOP gate, then the TOP can never occur under the stated boundary conditions, and no fault tree can be constructed. If there is an AND gate or a single input OR gate above, this, too, cannot occur, and must be deleted, along with any AND or single OR gate it may be input into. If this leads back to the TOP, then no fault tree can be constructed. However, if a multiple input OR gate exists anywhere above the current OR gate, then only the single branch containing the current

gate need be deleted as a "cannot occur" branch. If this is the final branch of the gate, then the gate is finished, and is sent to location 410 for intermediate editing. If further branches remain to be constructed, the OR gate is set to location 401 to reset its indices and begin development of the next branch.

Thus it is seen that development of an OR gate can lead to development of lower gates (cases 1 and 2), or to removal of gates and deletion of branches of gates above (case 3).

Several of these concepts are also utilized in loop 300 (Figure A-7) for constructing AND gates. Loop 300 is entered each time a row, input into an OR gate, is to be developed. As such, it has already been checked for contradictions, and the first step of the AND loop is to check for any row entries which match existing conditions. Since entries which duplicate existing system states are "sure to occur" (they have "already" occurred), they are treated as TRUE inputs into the AND gate which need neither be shown nor developed. However, should any entry in the row refer to a node ("KNODE") at which no state has yet been defined, the following will occur:

1) X(1,KNODE) will be set to the state required by the row entry, and
2) X(2, KNODE) will be set to "INDEX" (the number of the current AND gate).

This is done to assure that all events developed beneath this gate are mutually compatible with (i.e., do not contradict) all other events below, as required for an AND gate. Since all succeeding events occurring at this node will be checked against array X, this compatibility is assured.

Once all entries in the row have been checked against system and component states, those entries which represent previously undefined events will remain as inputs into the gate. As in the OR gate construction, three possibilities now exist:

1) Several new events have been found (i.e., states at several nodes have been found): thus, a multiple input AND gate exists. If all of these inputs are primary events, the gate is complete, and location 410 will be executed (post-gate intermediate editing). If one or more undeveloped inputs exist, they will be developed as OR gates (section 202).

2) A single input has been found: the gate is not a true AND, and the single entry set into array X must be reset to -1 ("undefined"). If the single input is an undeveloped event, it will be sent to location 203 to be developed as an OR gate, replacing the current AND gate. If the single input is a primary input, it will be directly inserted into the gate above. If the gate above is thus completed, it is sent to location 410 for intermediate editing; if further branches remain, location 400 is entered to reset indices and begin the next branch.

3) No previously undefined states have been found: thus the AND gate is automatically TRUE and will be deleted. If this is the TOP gate, the fault tree has no entries (is always true under the stated boundary conditions). Otherwise, section 331 will delete any OR, or single input AND gates above, up to the lowest multiple input AND gate, as "sure to occur." If no such multiple input AND gate is found above the current AND, the top event is always TRUE.

However, if such a multiple input AND occurs above the current gate, only the current branch is deleted, and the next branch can then be begun (location 400). If there are no further branches, then that AND is finished, and loop 410 is entered for intermediate editing.

A.8.2 Intermediate Editing

This completes the construction and preliminary editing phases of subroutine DO IT. The Intermediate Editing phase is then entered, as is illustrated by Figure A-8. This loop is entered (locations 400 or 401) any time a branch of a gate is completed. If further branches remain to be completed, section 401 sets indices for the next branch and sends it to the proper (AND/OR) loop to construct the next branch. Notice that if the current gate is an OR, and the last branch completed was an AND gate, system nodes will have been set which must be reset (set to "undefined") before the next branch of the OR is constructed. These nodes will be indicated by:

X(2, NODE) > JDEX,

(where "JDEX" is the number of the current OR gate), since all gates below the current OR will have larger indices.

If the current gate has been completed, and more than one input exists, the intermediate editing phase for the appropriate gate type will begin. (If only one input remains, see section 485).

The intermediate editing algorithm for AND gates (reference A-2) is required because the removal of single input OR gates may lead to contradictory events beneath an AND gate. Such a situation is illustrated by the following figure:

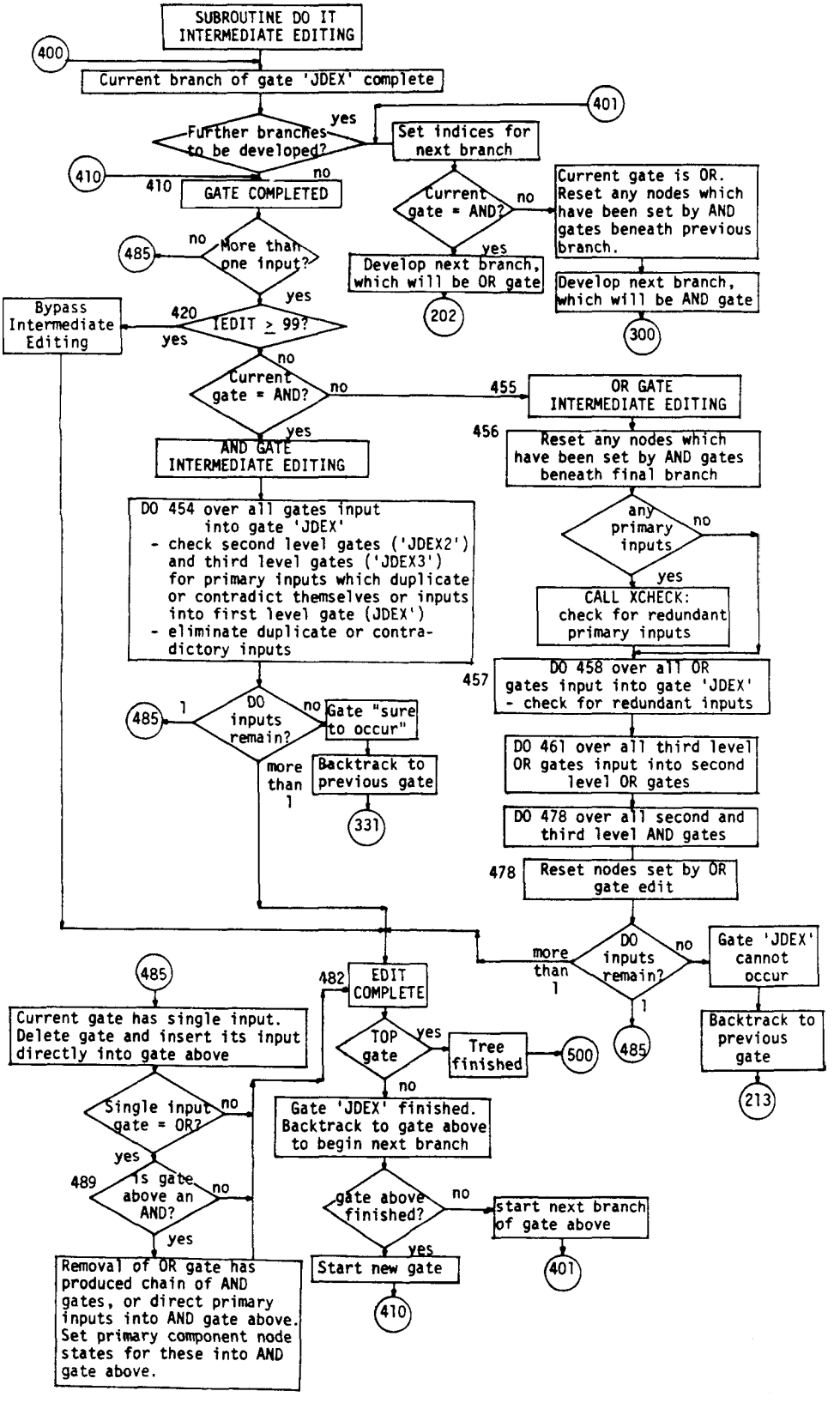
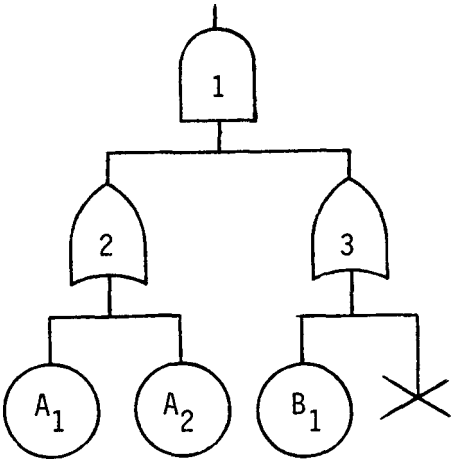



Figure A-8. Flowchart for Intermediate Editing Loop of Subroutine DO IT




Since primary events $A_i$ and $B_i$ are beneath separate OR gates, these primary inputs do not affect one another. However, once branch 2 of OR gate 3 has been eliminated, gate 3 itself can be removed and primary event $B_1$ becomes input into AND gate 1. In the resulting tree, event $B_1$ is forced to occur for gate 1 to be true, and thus both $A_1$ and $B_1$ should be compatible with it. That is, if the original tree is represented by:

$$(A_1 \cup A_2) \cap (B_1 \cup B_2) \neq \phi,$$

then $A_1B_2 \neq \phi$ or $A_2B_2 \neq \phi$ would satisfy it, even if $A_1B_1 = \phi$ and $A_2B_1 = \phi$. However, the elimination of $B_2$ leaves:

$$(A_1 \cup A_2) \cap B_1 \neq \phi.$$

In this case, if $A_1B_1 = \phi$, or $A_2A_1 = \phi$ then $A_1$ or $A_2$ may be removed without affecting the tree. The effect of this elimination is a more compact tree, although the previous form would still be correct. If, however, $A_1B_1 = \phi$ and $A_2B_1 = \phi$, then

$$(A_1 \cup A_2) \cap B_1 = \phi,$$

and the entire subtree 1 should be eliminated.

A third situation arises if, in the original tree, $A_1$ or $A_2$ was identical with $B_1$. In this case, the reduced tree would become:

$$(A_1 \cup A_2) \cap B_1 \equiv B_1.$$

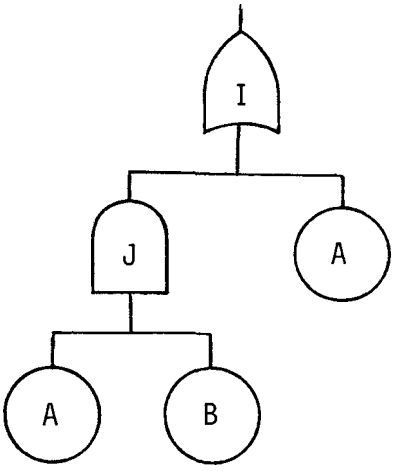
Thus gate 1 becomes a single input gate, and is replaced by the input $B_1$ itself.

In Subroutine DO IT, this type of editing is performed by loop 454 each time an AND gate is completed. For any AND gate "JDEX", all second level gates (gates directly input into JDEX) and third level gates (input into the second level gates) are searched for such redundant or contradictory events (all primary inputs to all three levels of gates are checked against each other). Note that this may lead to zero or single input gates beneath the current AND gate, which will be edited out by calls to subroutine REDUCE.

When the AND gate edit is completed, and multiple inputs remain, loop 482 is entered to begin the next gate. If only a single input remains, section 485 is entered to delete the gate. Finally, if no inputs remain, the AND gate either cannot occur (go to 213 to edit gates above) or is "sure to occur " (go to 331 to edit gates).

Although intermediate editing is not required for OR gates, it can be useful in simplifying some fault trees. The basis for the OR gate intermediate editing is the concept of minimal cut sets, as follows. If an event occurs as a direct input to an OR gate, and to an AND gate below, the AND gate is redundant and can be deleted, along with any AND gates above, up to the lowest OR. This can be seen by referring

to the figure below and using the definition of minimal cut sets [A-3]. The cut sets for gate I are (A) and (A,B). However, since (A) is minimal, (A,B) can be eliminated and gate I replaced by the single event A.




The more general situation is shown in Figure A-9. This tree represents the upper three levels of a typical sub-tree beneath an OR gate, and illustrates the level of editing actually performed in section 455 of DO IT.

First notice that all events $A_i$, B and C are equivalent, lying beneath direct OR gates. That is, any one of these events represents a minimal cut set for gate 1. Thus each of these must be checked against the primary inputs to the AND gates 4, 5 and 7, as well as against the inputs to OR gate 6, whose events are not minimal, since they lie beneath AND gate 5. This cross checking is performed at each gate level by subroutine XCHECK.

Additionally, it is seen that primary events D, E and G are equivalent, since each lies beneath an AND or chain of AND's and is input to the top OR or chain of OR's. Thus, should any of the events $A_i$, B

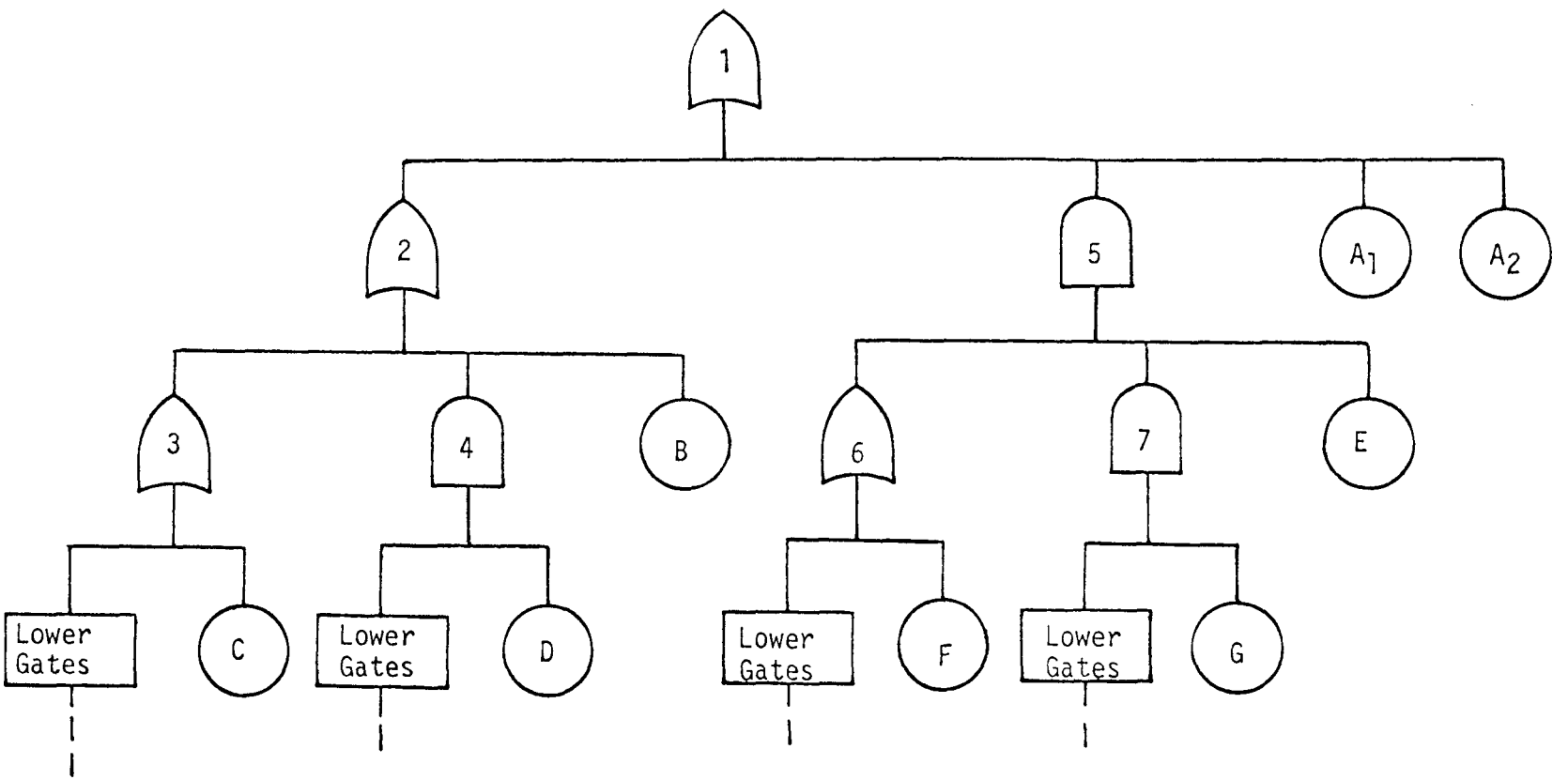



Figure A-9. Sample Tree for Intermediate Editing

or C be identical to D, E or G, the appropriate AND would be deleted and, in the case of event G, both gates 7 and 5 would be eliminated. Event F presents a different situation. Each individual event beneath gate 6 couples with gate 7 and event E to form a different cut set. If event F is itself minimal, only that cut set containing F need be removed from beneath gate 5. Thus event F would be deleted from gate 6, leaving any other branches intact. If this leaves gate 6 with a single input, then that gate would be eliminated and the lower events would input into gate 5. However, this could result in a direct primary input, or AND gate input to gate 5, requiring further editing which could eventually lead to the removal of gate 5 itself. Specifically, in DO IT, section 458 checks for events B identical to events A, loop 461 checks events C against A and B, and loop 478 checks events D, E, F and G. In order to do this, section 456 resets any nodes set by AND gates below the current gate, and sets nodes for each event found by the successive loops. Thus, for example, should loop 458 find an event B which matches an event A already set, that event will be deleted as redundant. However, any events B not already found will then be set in array X, to be checked by later loops for C, D, E, F and G.

Following the OR gate edit, the number of inputs remaining is checked, as in the AND edit, and similar actions are taken. Finally, when either type gate is completed (step 482), it is checked to see if the TOP gate. If so, the tree is finished (step 500). If a further gate remains above, it is sent to location 401 if other branches remain, or to location 410 if that gate itself has been completed.

If in the process of the intermediate editing, a single input

gate is produced, section 485 is also executed. The first step is to delete the gate and insert its single input into the gate above. Then, if the gate deleted is an OR, and the gate above is an AND, nodes must be set by the AND gate above in the event any AND gates or primary events are directly input into it. That is, since any events directly input into an AND (or into a chain of AND's) must be set into array X to assure no contradictory events beneath, sections 490-4999 check all direct sequences of AND gates, down to 5 succeeding levels, for primary events which must be set for the AND gate above.

A.8.3 Final Editing

The flowchart for the final editing phases of DO IT is shown in Figure A-10. This consists primarily of writing fault tree output and producing gate transfers. First, if the TOP gate has only a single input, it is replaced by the gate input into it (or is left as is if the input is a primary event). Then, following a preliminary gate printout, a search is made for transfers (if IEDIT has been properly set).

This transfer search proceeds from the bottom up using the following algorithm:

1) Search for gates with only primary inputs into them (these are the lowest level gates). If any gates have identical inputs, remove the duplicate gate and replace by a transfer (this is done by using the same gate number for both gates).
2) Search for gates with only primary inputs and gates input which have already been checked for transfers. Note that since higher numbered gates are always input into lower

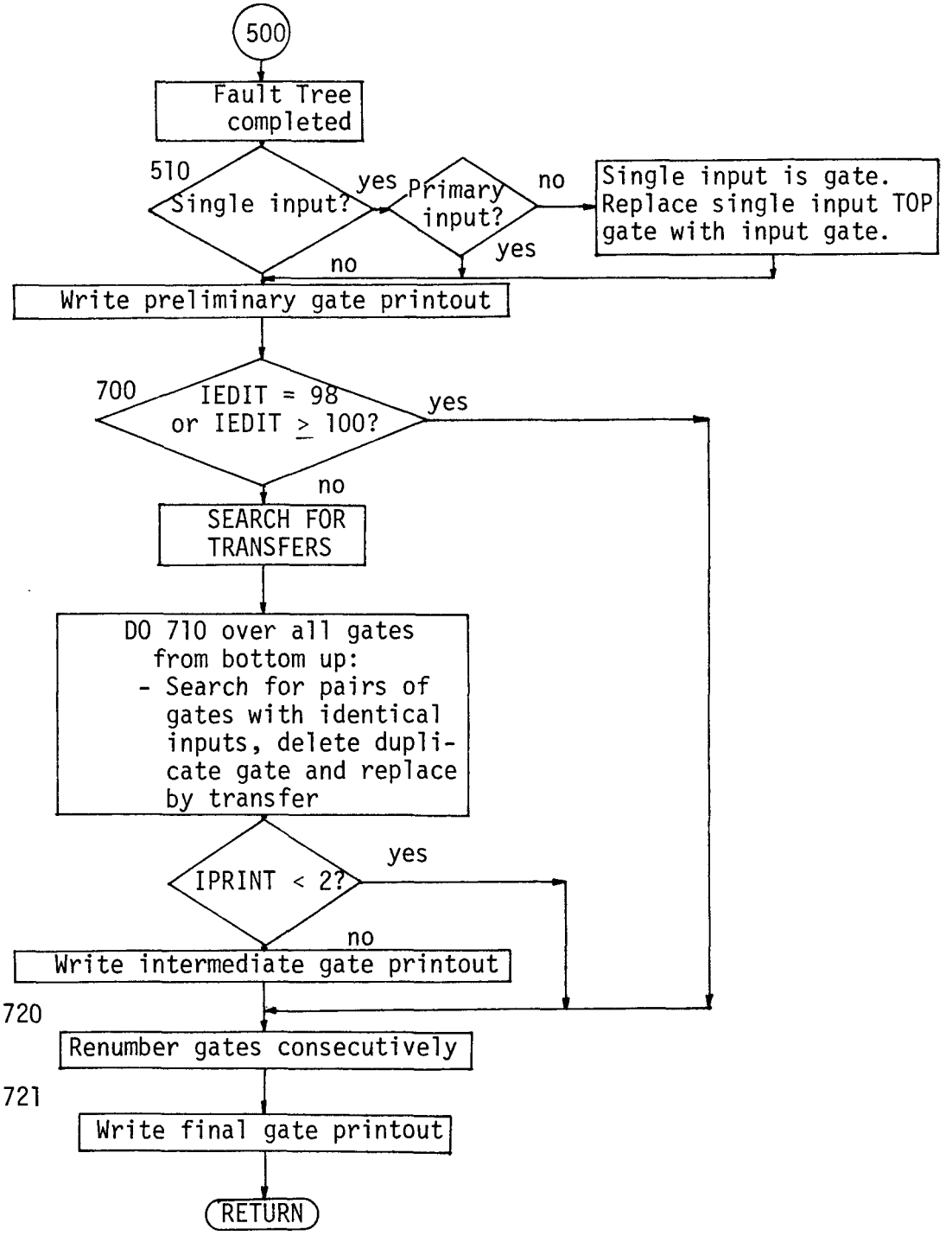



Figure A-10. Flowchart for Final Editing Phase of Subroutine DO IT

numbered gates, by starting the search from the highest numbered gate, it is guaranteed that all inputs to a higher level gate will have been checked by the time that higher level gate is reached. Then, for each gate, check to see if its type (AND/OR) and numbers of gate and primary inputs coincide with any other gate. If so, and if all inputs are identical, the gates are duplicates, so a transfer can be produced.

Finally, after transfers are checked, the gates are renumbered consecutively to fill in any gaps left by the editing phases, and final printout is produced.

A.8.4 Error Messages from DO IT

In addition to a large number of messages produced during the fault tree construction phases, several error messages may be produced by subroutine DO IT. Most are self-explanatory, and concern fault trees which "cannot occur", are "sure to occur," or are too large for the current dimensions of array MAT (Section A.3). One additional message, produced by FORMAT 1, has the following form:

```
* * * * * * * * * * * * * * * *
* ERROR NUMBER  nnn           *
* * * * * * * * * * * * * * * *
```

This message will not occur with the present version of the code unless certain programming modifications are made which produce internal code errors within DO IT. In these cases, this message will serve as a debugging aid, pointing to one of 13 locations where the error was detected. The number "nnn" refers to a program statement number either just before or after this message was printed (a double row of asterisks

in the program indicates the location of each of these). If such a message ever occurs, the program logic should be checked at that point to determine what caused the message, and any program modifications which affect the variables in question can then be searched for errors.

## A.9 Subroutine XCHECK

As pointed out in Section A.8.2, subroutine XCHECK is called by DO IT in the intermediate OR gate edit, in order to check and set nodes to eliminate redundant events beneath OR gates. The input flag ISTART is set by DO IT as follows:

ISTART = 1 check and set nodes for check of direct primary inputs (A, B and C of figure A-9).

ISTART = 0 check, but do not set nodes for OR-AND-OR gates (inputs F).

ISTART = -1 check, but do not set nodes for OR-OR-AND/OR-AND-AND gates (inputs D, E, G).

Internal variable ISET is then set to ISTART.

This subroutine then cross checks all primary inputs to the current gate with any primary component states already set into array X. If no preset state is found, the current input is not redundant, and the next input is checked (after setting the current state into X if ISET = 1). If the current state has already been preset into array X, it is redundant and the current input will be deleted. Note that if ISET = -1, the current event inputs an AND gate, and subroutine XCHECK returns to DO IT where the entire gate will be deleted (Figure A-11).

Finally, if a state different from the current state has been preset at the same component node "JNODE" in array X, this means that multiple states of the same component exist beneath the OR gate (a valid situation). In this case, array location X(2, JNODE) is set to the negative location in array GATE where a further state for that node may have been set (if X(2, JNODE) $\geq$ -1, no further states exist).

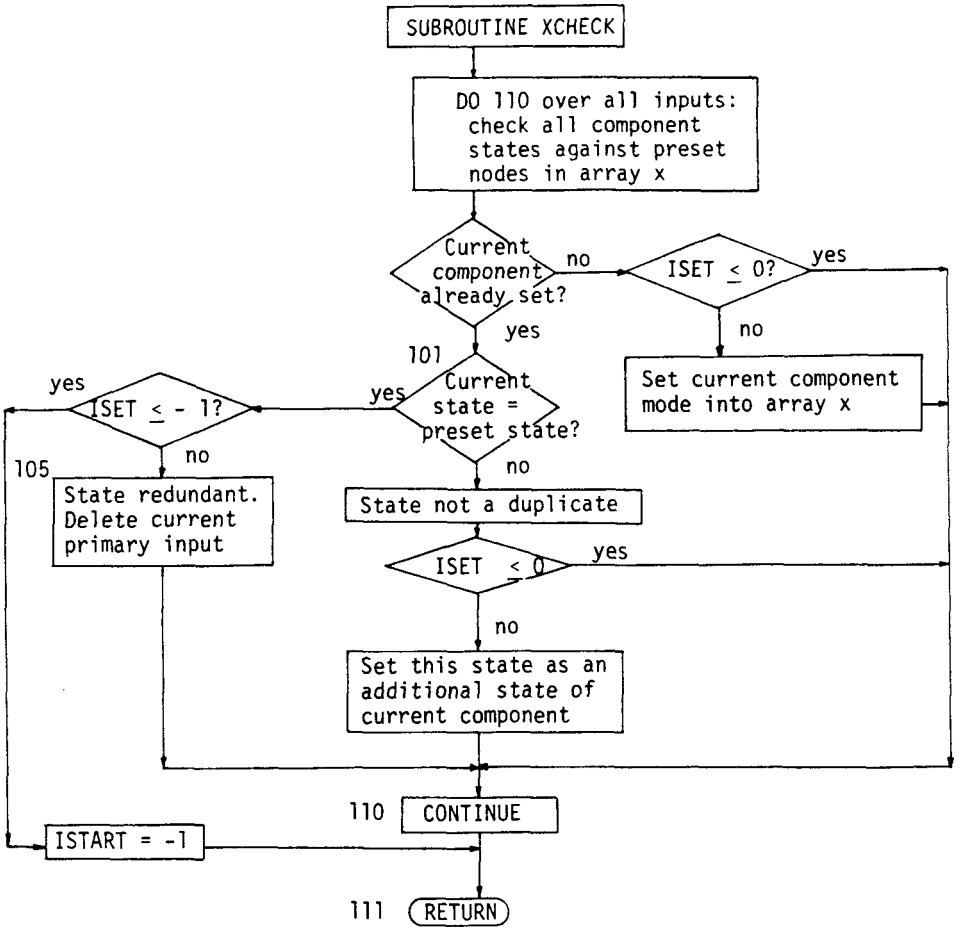



Figure A-11. Flowchart for Subroutine XCHECK.

This location in array GATE may point to additional locations where further states have been set. Thus several different component states may co-exist under the same OR gate, and each must be compared with the current state. If a redundancy exists between the current state and any of these, the current input is treated as before. However, if it is different from each of the others, a new location is set (if ISET = 1) by setting the last location checked in array GATE equal to the negative location of the current state, thus pointing to the new additional mode of node JNODE. This, then leads to a chain of indices, each pointing to the location of the next state of node JNODE. (Note that the final location is indicated by a positive number which may later be set to a negative pointer if further states are found).

## A.10 Subroutine REDUCE

Subroutine REDUCE is called by DO IT in order to delete zero and single input gates. Its flowchart is shown in Figure A-12. LLDEX is first set by DO IT to the number of inputs (0 or 1) of gate "JDEX2," which is input into gate "JDEX." If JDEX2 has no inputs, that input location in gate JDEX is eliminated, the number of gates input is reduced by one, and the empty location is filled by moving up subsequent inputs.

If LLDEX = 1, the single input into gate JDEX2 is input directly into JDEX. Note that if the single input is a primary input, this means that a gate input into gate JDEX is being replaced by a primary input. Thus, entries must be switched to assure that the new primary input in JDEX follows all gate inputs.

A special situation may arise if this subroutine was called from the OR gate intermediate edit region of DO IT, where negative gate entries serve as pointers (section A.9). In this case, moving entries to fill deleted inputs may move a pointer. Thus, inputs so moved must be checked, and the pointer changed to indicate the new location.

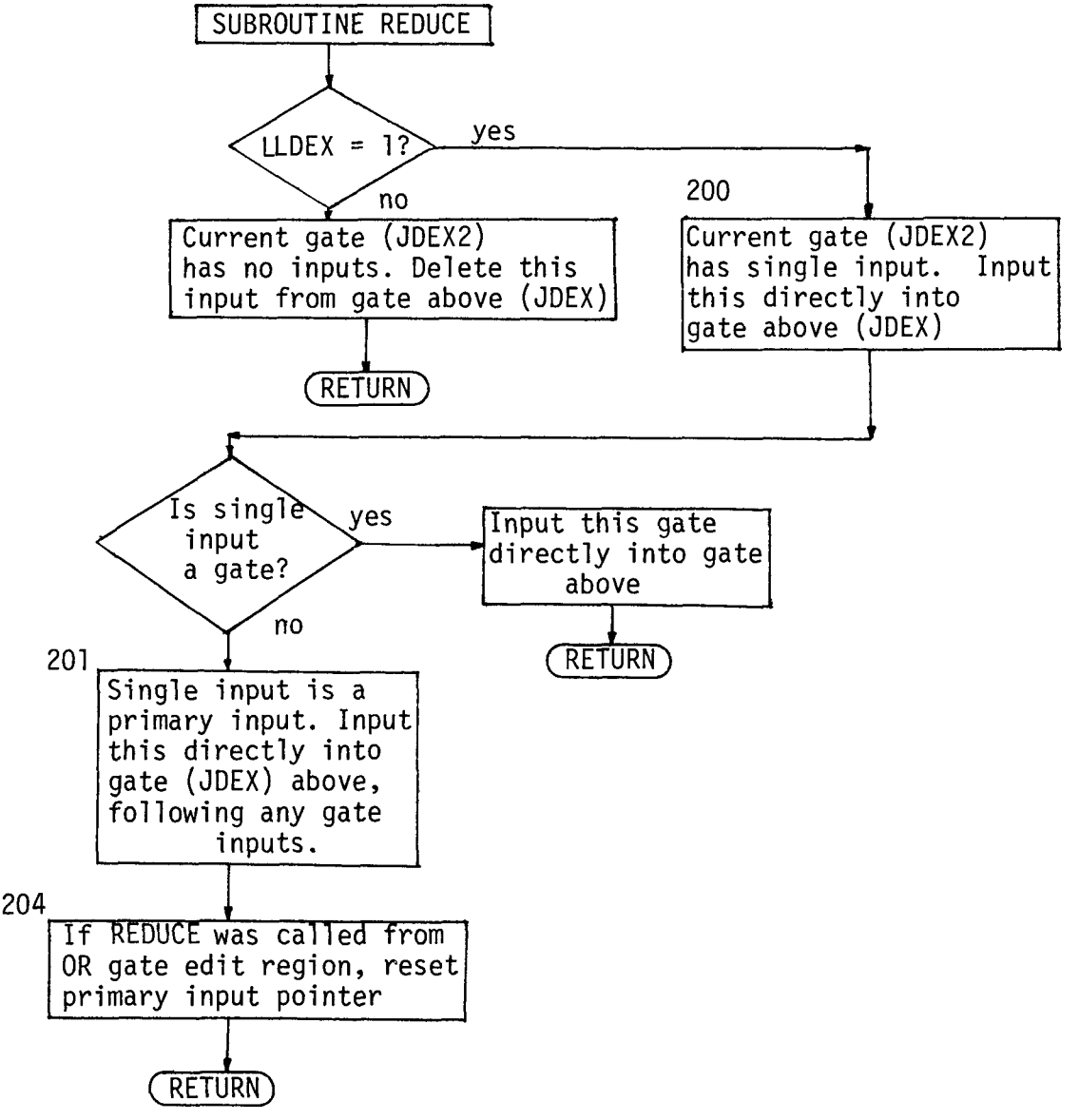



Figure A-12. Flowchart for Subroutine REDUCE.

## A.11 Subroutine OUTPUT

The final routine to be called by DRIVER is subroutine OUTPUT. Its function is to produce the fault tree along with failure data and control information, required as input by the PREP code. By using the CAT input parameter "IOT" to specify the device number to which the output is to be sent, either punched cards, tape or disk output may be produced, in the input format required by PREP. In addition, this identical output, along with further information in the form of cross references, and error messages, is printed by this subroutine following the output from the fault tree construction and editing phases. A simplified flowchart of subroutine OUTPUT is shown in Figure A-13, and is described below. It should be noted that, if a fault tree analysis code other than PREP is desired to be used with the CAT code, the basic structure of this subroutine may still be used in many cases, with suitable changes in input/output formats, naming or numbering conventions, order of output, etc.

Since the PREP input data required is of three types, subroutine OUTPUT has been programmed in three sections. These correspond to statement numbers 100-150, 200-400 and 500-590 in the program listing, and produce the output for the "DATA," "TREE" and "RATES" input sections for the PREP code, respectively. Additional pre- and post-processing is done in other sections of the subroutine. For example, integer array "IDUM" and alphanumeric arrays "INAME" and "JNAME" are initially cleared in sections 10-30, along with setting certain flags and writing output headers.

The first output produced by the code is the control data for

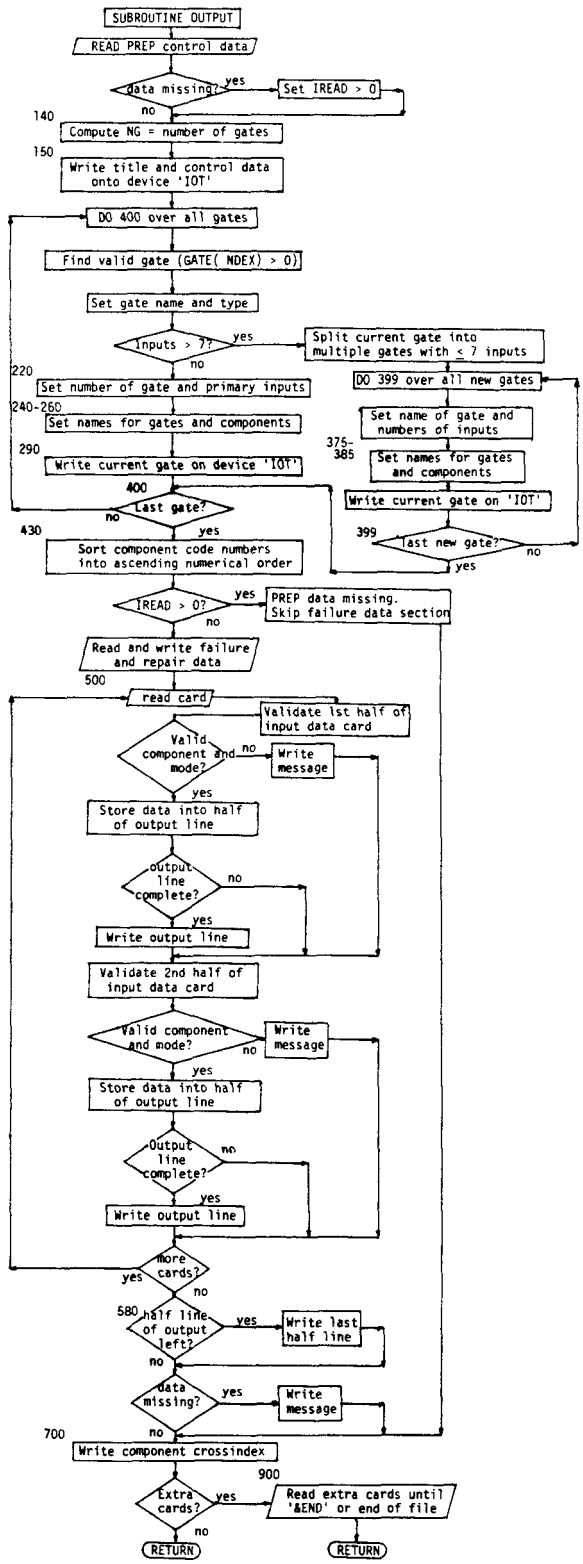



Figure A-13. Flowchart for Subroutine OUTPUT

PREP. This is read between statements 100-110, and stored in array IDUM and as variable TAA. Should one or both of the two control data cards be missing from the '&OUT' section of the CAT input data, the PREP "DATA" output would still be produced, but with zeroes in place of any missing data. In this case, error messages would also be produced and the flag, "IREAD" would also be set, to suppress further attempts to read or crosscheck the missing data. Note that if an "&END" or end of file has already been read by subroutine STEVE, flag "IREAD" will be set immediately, and the initial read statements in OUTPUT will also be bypassed. Since the value of "NG" (the number of gates in the fault tree) is not known beforehand, its value need not be input to CAT. This number will be set in loop 150, and all control data will then be written onto device "IOT."

Loop 400 is executed once for each gate, in order to write out the fault tree itself in the required PREP input format. This is done by storing gate and component names letter-by-letter (and digit-by-digit) in arrays INAME and JNAME. INAME stores the current gate name as eight characters, in the form "GATEnnnn" (except for the first gate, which must have the name "TOP"). This is followed by the gate type, AND or OR, (stored in NKIND), the numbers of gates and primary components input (INGATE, IPRIME), and the names of the gates input, followed by the primary components (stored as eight character groups in array JNAME).

The majority of loop 400 is concerned with generating the required names, character-by-character. For components, as discussed in Section 2.3, this consists of combining the four digit "internal node number" with the four digit component state into one eight digit

number which serves as the name of that component failure state. Gate names are simply the word "GATE" plus the gate number itself (up to 9999 gates can be accomodated with the eight character name). However, should CAT produce a gate with more than the seven inputs allowed by PREP, loop 399 is entered, to split the single large gate into several smaller gates. This is done by creating seven (or fewer) new gates of the same type, and inputting them into the large gate above, each of the newer gates taking up to seven of the original inputs. For an original gate named "GATEnnnn," these new gates would be named "GAT1nnnn," "GAT2nnnn," up to "GAT7nnnn" if necessary. Note that if the TOP gate has more than seven inputs, the additional gates would be named "GAT10001," "GAT20001," etc., even though the original gate would retain the name "TOP."

Following the fault tree printout, subroutine OUTPUT will then produce the failure data output. The first step is to arrange the newly created component state numbers into numerical order, in order to have a rapid means of checking the failure data for either extra, or missing components or failure states. This is done in sections 410-460. Since the fault tree is no longer needed, array GATE is used for this table: locations 1-NCOUNT store the component state numbers, and locations (NCOUNT + 1) - (2*NCOUNT) will be set to 1 when the appropriate input data are later validated. Then the failure data can be read in (statement 500), checked against this table, and output to PREP if this component and state actually appear in the fault tree. Again, flag IREAD is used to bypass the input operations if data are missing.

Since each input and output data card contains data for two components, each half of these cards is treated essentially independentally, permitting one half of a CAT input card to be written into the opposite half of the PREP data card. This is done utilizing two flags with the following states:

ILOOP = 1 currently validating first half of input card

ILOOP = 2 currently card validating second half of input card

JLOOP = 1 first half of current output card available

JLOOP = 2 first half of current output card filled, second half available.

The program then proceeds as follows. Both halves of an input card are read (statement 500), ILOOP is set to 1, and the first component is validated. If valid, the component data is stored in the first half of the output locations if JLOOP = 1, and into the second half if JLOOP = 2. If JLOOP = 2, the output locations are now full, and a complete output card is written onto 'IOT.' In either case, both ILOOP and JLOOP are set to the opposite states, and the validation of the second half of the input card begins. After repeating the above sequence, one full input card has been validated, and the next input card is read.

The validation itself consists of the following. First, the current half-card is checked to see if blank (although PREP requires both halves of each card to contain data, CAT will accept cards with either one or two sets of data). If input is present, array "CMPNAM" is searched to see if that component exists. If it does, its index number "NDEX" is used, along with the internal column number "INT" and failure state "STATE" (as input on the data card) to generate

the required eight digit reference number (see Section A.2). This number is checked to see if it appears in the ordered table (the first half of "GATE"), generated in 410-460. If so, then the current data are stored in the correct output locations, and a flag is set in the second half of "GATE" to indicate that data for this state has now been input. However, if the component name, as input, does not exist, the specific failure state reference number does not appear in the fault tree or data for that number has already been input, an appropriate message is written, and no values are stored.

When no failure data cards remain (as indicated by "END," "&END" or "end-of-file,") statements 580 and following will write any remaining half of an output data card onto device "IOT." This completes the PREP output phases, and loop 620 of subroutine OUTPUT then searches the second half of array "GATE" to make sure that data has been read for all component failure states. If data is missing, each component and state without failure data will be listed. Then, a cross reference of the eight digit component failure state numbers, with their corresponding input names, node numbers and failure states, is produced in section 700. Finally, section 900 is executed, depending on the state of the flag IREAD, to read or bypass any remaining input. If an "&END" or end-of-file was already read (IREAD = 0, 2, 3, 5 or 7), no data remains. However, if "END," or an unknown card type was last read, the subroutine will search for an "&END" or end-of-file before returning.

# REFERENCES


A-1. Salem, S. L., G. E. Apostolakis and D. Okrent, A Computer-Oriented Approach to Fault-Tree Construction, EPRI Report NP-288, Palo Alto, Ca., November 1976.

A-2. Salem, S. L., G. E. Apostolakis and D. Okrent, "A New Methodology for the Computer-Aided Construction of Fault Trees," Annals of Nucl. Energy, vol. 4, pp. 417-433, 1977.

A-3. Vesely, W. E., "A Time-Dependent Methodology for Fault Tree Analysis," Nucl. Engr. and Design, vol. 13, No. 2, pp. 337-360, August 1970.

# APPENDIX B. DECISION TABLE MODELS

In order to make effective use of the CAT code, it is desirable for the user to develop a basic library of component models for the types of systems he expects to analyze most frequently. Chapter 3 of this report has discussed two methods by which decision table models may be readily constructed. As a further aid to the user, this appendix will present a number of decision tables which may serve as examples for subsequent decision table development.

The decision tables which follow are not intended to be completely detailed component models. Rather, they are intended to represent the basic feature of a number of varied components. Thus, quite often several failure modes have been combined into one when their effect on the output is identical. Furthermore, several of the simpler decision tables may be used to model a wide range of components either unchanged, or by means of a few simple additions or deletions.

Two special modeling considerations have been illustrated in these examples. First, a method of treating common cause failures has been introduced in the three-state amplifier model. This approach can be useful for producing decision tables which treat such common cause mechanisms as calibration errors, operator errors, design errors, design deficiencies, etc. Secondly, the manner of including external power sources into component models has been shown in the general pump table.

If the user desires to develop more complex decision table models, or to add to these samples, the recent IEEE Std 500-1977[B-1] may serve as a good reference for the failure modes and failure rates of

electrical components used in nuclear systems. In addition, Appendix III of the Reactor Safety Study [B-2] contains failure modes and rates for many types of nuclear components.

The sample tables presented here include the following component models:

1. Amplifiers/sensors
2. Annunciator
3. Batteries/power supplies
4. Cables/resistors
5. Circuit breakers/contacts (normally closed)
6. Contacts (normally open)
7. Fuse/circuit breaker
8. Pipe/fitting
9. Pump/general active component
10. Signal source
11. Switch
12. Valve (motor-operated)

AMPLIFIERS/SENSORS

These amplifiers model a three state "unity gain", or isolation type amplifier, and a very simple, two state inverting amplifier. Note that the first model could be easily extended to represent an amplifier with positive gain. Notice also that this model is a very simple extension of the decision table for cables and resistors which, themselves, can be considered unity gain amplifiers. The only change has been the addition of the states 5001, 5010 and 5011 (calibration errors) to the amplifier model. This amplifier model could also be used to represent a sensor circuit which produces an output proportional to the measured variable.

THREE STATE AMPLIFIER

| Row | Input | Internal Mode | Output |
|---|---|---|---|
| 1 | 0 | -1 | 0 |
| 2 | -1 | 1 | 0 |
| 3 | -1 | 1001 | 0 |
| 4 | -1 | 5010 | 0 |
| 5 | 1 | 0 | 1 |
| 6 | -1 | 5001 | 1 |
| 7 | -1 | 1002 | 1 |
| 8 | 2 | 0 | 2 |
| 9 | -1 | 5011 | 2 |
| 10 | -1 | 1002 | 2 |

Using the above model for several identical amplifiers in a system would produce a fault tree with independent calibration errors for each amplifier. It is, however, possible to include common mode calibration errors via an additional external input. If a common "signal" were sent to the calibration inputs of several amplifiers using the table below, all amplifiers would simultaneously fail due to this common mode error. Furthermore, both independent and common mode failures could be included in this model by adding rows 4, 6 and 9 from the previous model (with '0' for the second input column).

THREE STATE AMPLIFIER WITH COMMON MODE
CALIBRATION ERROR

| Row | Signal Input | Calibration Input | Internal Mode | Output |
|---|---|---|---|---|
| 1 | 0 | 0 | -1 | 0 |
| 2 | -1 | 0 | 1 | 0 |
| 3 | -1 | 0 | 1001 | 0 |
| 4 | 1 | 0 | 0 | 1 |
| 5 | -1 | 0 | 1002 | 1 |
| 6 | 2 | 0 | 0 | 2 |
| 7 | -1 | 0 | 1002 | 2 |
| 8 | -1 | 1 | -1 | 0 |
| 9 | -1 | 2 | -1 | 1 |
| 10 | -1 | 3 | -1 | 2 |

The decision table for the calibration input "component" would have the following form:

| Row | Internal Mode | Output |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 5010 | 1 |
| 3 | 5001 | 2 |
| 4 | 5011 | 3 |

A single "component" using this model could be defined as input to all amplifiers; alternatively, several different components could be used, each as input to a specific group of amplifiers which would be calibrated at the same time or by the same person. Finally, such a model would be useful for comparator circuits, sensor circuits, trip logic, etc.

The above models could be also extended by the addition of a

power supply input, whose failure would produce the zero output state.

A final amplifier is the following highly reduced model for a two state invertor:

INVERTOR AMPLIFIER

| Row | Input | Internal Mode | Output |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 2 | -1 | 1 | 0 |
| 3 | 0 | 0 | 1 |
| 4 | -1 | 1002 | 1 |

This model is useful for representing sensor circuits which produce an output when the measured parameter falls below a preset value. In this case, calibration errors could be included, as before. Furthermore, this can be useful for driving such circuits as the annunciator which is described below.

ANNUNCIATOR

The function of this communicator is to produce an alarm (output) on receipt of a signal. If an alarm on a zero signal is desired, an inverting amplifier (such as described previously) may be inserted before the annunciator.

| Row | Input | Internal Mode | Output |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 2 | -1 | 4 | 0 |
| 3 | 1 | 0 | 1 |
| 4 | -1 | 6 | 1 |

BATTERIES/POWER SUPPLIES

This model provides a signal (voltage) source to those components which explicitly include a power supply input. Since a single power supply may be used by many components, this may provide a major common cause failure mechanism.

| Row | Internal Mode | Output |
|---|---|---|
| 1 | 3 | 0 |
| 2 | 0 | 1 |
| 3 | 1003 | 2 |

CABLES/RESISTORS

This decision table can be used to represent any general type of transmission device (cable, resistor, connector, etc.). It is very similar to the previously described unity gain amplifier with the omission of the calibration errors. It is also similar to the decision table for a pipe, with the addition of "short circuit" failure modes. If the cable or resistor cannot withstand an overload, the fuse model should also be consulted.

| Row | Input | Internal Mode | Output |
|---|---|---|---|
| 1 | 0 | -1 | 0 |
| 2 | -1 | 1 | 0 |
| 3 | -1 | 1001 | 0 |
| 4 | 1 | 0 | 1 |
| 5 | -1 | 1002 | 1 |
| 6 | 2 | 0 | 2 |
| 7 | -1 | 1002 | 2 |

CIRCUIT BREAKERS/CONTACTS (Normally closed)

This model represents a circuit breaker which trips on an external signal. Note that this is identical in operation to normally closed contacts which open on a control signal. A protective circuit breaker (trips on overload) will be described later as the fuse model.

| Row | Voltage Input | Control Input | Internal Mode | Output |
|---|---|---|---|---|
| 1 | 0 | -1 | -1 | 0 |
| 2 | -1 | 1 | 0 | 0 |
| 3 | -1 | -1 | 1 | 0 |
| 4 | -1 | -1 | 6 | 0 |
| 5 | 1 | 0 | 0 | 1 |
| 6 | 1 | -1 | 2 | 1 |
| 7 | 2 | 0 | 0 | 2 |
| 8 | 2 | -1 | 2 | 2 |

CONTACTS (NORMALLY OPEN)

The function of this component is to close on an appropriate signal. This model is essentially the inverse of the previous one.

| Row | Voltage Input | Control Input | Internal Mode | Output |
|---|---|---|---|---|
| 1 | 0 | -1 | -1 | 0 |
| 2 | -1 | 0 | 0 | 0 |
| 3 | -1 | -1 | 1 | 0 |
| 4 | 1 | 1 | 0 | 1 |
| 5 | 1 | -1 | 2 | 1 |
| 6 | 1 | -1 | 6 | 1 |
| 7 | 2 | 1 | 0 | 2 |
| 8 | 2 | -1 | 2 | 2 |
| 9 | 2 | -1 | 6 | 2 |

FUSE/CIRCUIT BREAKER

This table represents a component (such as a fuse, fusible resistor, or circuit breaker) which is designed to fail on an overload.

| Row | Input | Internal Mode | Output |
|---|---|---|---|
| 1 | 0 | -1 | 0 |
| 2 | -1 | 1 | 0 |
| 3 | 2 | 0 | 0 |
| 4 | 1 | 0 | 1 |
| 5 | 1 | 2 | 1 |
| 6 | 2 | 2 | 2 |

PIPE/FITTING

The pipe decision table is a mechanical equivalent to the electrical cable model. Note, however, that there is no equivalent to an electrical short in this model.

| Row | Input | Internal Mode | Output |
|---|---|---|---|
| 1 | 0 | -1 | 0 |
| 2 | -1 | 3001 | 0 |
| 3 | -1 | 3002 | 0 |
| 4 | -1 | 3003 | 0 |
| 5 | 1 | 0 | 1 |

PUMP/GENERAL ACTIVE COMPONENT

This is the pump model developed in Section 3.2. A large number of components can be represented by a model such as this, which requires both a source of power and a signal (fluid) input in order to operate (produce an output). Note that this can also be used to represent a two state amplifier with external power supply.

| Row | Fluid Input | Power Input | Internal Mode | Output |
|---|---|---|---|---|
| 1 | 0 | -1 | -1 | 0 |
| 2 | -1 | 0 | -1 | 0 |
| 3 | -1 | -1 | 4 | 0 |
| 4 | -1 | -1 | 5 | 0 |
| 5 | 1 | 1 | 0 | 1 |

SIGNAL SOURCE

Since CAT requires all component input nodes to be connected to a preceding component output node, this signal source may be connected to any component input (Section 2.2.5). Since this model has no inputs, any backtracking by the code will terminate here. A signal source may be used for the following purposes:

1) To connect to component inputs such as amplifier or sensor inputs. Often the state at such a node will be preset by a boundary condition.
2) To connect to component inputs which will not be used.

| Row | Internal | Output |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 1 | 1 |
| 3 | 2 | 2 |
| 4 | 3 | 3 |
| ⋮ | ⋮ | ⋮ |
| | | etc. |

SWITCH

This is the model for the ON-OFF switch developed in Section 3.5 of reference B-3. It has been generalized to a three-state table.

This switch has two internal modes: Position (1 = off, 2 = on), and mechanical(0= good, 1 = failed open, 2 = failed closed). Note that the position column could be treated as an external input, and would then behave as a relay.

| Row | Input | Position | Mechanical | Output |
|---|---|---|---|---|
| 1 | 0 | -1 | -1 | 0 |
| 2 | -1 | 1 | 0 | 0 |
| 3 | -1 | -1 | 1 | 0 |
| 4 | 1 | -1 | 2 | 1 |
| 5 | 1 | 2 | 0 | 1 |
| 6 | 2 | -1 | 2 | 2 |
| 7 | 2 | 2 | 0 | 2 |

VALVE (MOTOR-OPERATED)

This decision table models a motor-operated isolation valve with slip clutch. This valve is designed to remain closed (isolate) as long as its input is under high pressure (input = 1). A discussion of this valve model has been presented in section 4.2.1 of reference B-3. The first input is the fluid input, followed by the control signal and maintenance override inputs (the latter used to bypass the externally interlocked control signal). The four internal modes include an initial position (1 = left open, 2 = left closed), mechanical state, slip clutch and relay failure states, followed by the output. Note that the slip clutch is used to prevent the valve from opening while under pressure, and that the relay is designed to "lock in" on a control signal and assure that the valve opens and closes fully.

| Row | Fluid Input | Control Input | Maint. Input | Position | Mech. | Clutch | Relay | Output |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 | 0 |
| 2 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | 1 |
| 3 | 1 | -1 | -1 | 1 | -1 | -1 | -1 | 1 |
| 4 | 1 | 1 | -1 | -1 | -1 | 1 | -1 | 1 |
| 5 | 1 | -1 | 1 | -1 | -1 | 1 | -1 | 1 |
| 6 | -1 | -1 | -1 | 2 | 0 | 0 | -1 | 0 |
| 7 | -1 | 0 | 0 | 2 | 0 | -1 | 0 | 0 |
| 8 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

# REFERENCES


B-1 IEEE Std 500-1977, "IEEE Guide to the Collection and Presentation of Electrical, Electronic and Sensing Component Reliability Data for Nuclear-Power Generating Stations".

B-2 U.S. Nuclear Regulatory Commission, Reactor Safety Study, WASH-1400 (NUREG-75/014, October 1975.

B-3 Salem, S. L., G. E. Apostolakis and D. Okrent, A Computer-Oriented Approach to Fault-Tree Construction, EPRI NP-288, Palo Alto, November 1976.

# APPENDIX C

## SAMPLE CASE

A simple example system has been developed in order to illustrate the use of the CAT code. This example is an extension of the system used in Section 3.5 of reference C-1 to demonstrate manual fault tree construction techniques.

This system, shown in Figure C-1, is a simple electrical circuit with an operator in a feedback loop. Signals 1 and 2 are simple two state inputs, defined to be in existence as boundary (initial) conditions. These signals will feed through switches 1 and 2 into a terminal where, if either signal is present, it will appear at node 5. If the contacts of the relay switch are closed, the signal will be transmitted to node 7. If no signal is present at this node, an invertor will send a signal, via an annunciator, to the operator, who will check to assure that switches 1 and 2 are closed (or will close them, if necessary).

The input signals will be described by decision table C-1, listed as type 101 in the sample input. This table indicates that an output will be produced if the signal is "good;" (note, however that a signal will <u>always</u> exist due to the boundary conditions defined).

TABLE C-1. SIGNAL (101)

| Row | Internal | Output |
|---|---|---|
| 1 | 0 | 1 |
| 2 | 1 | 0 |

Switches 1 and 2 will be two position (on/off) switches similar to the table in Appendix B but with two changes:

1) input/output states of 2 have been deleted, leaving 7 rows.

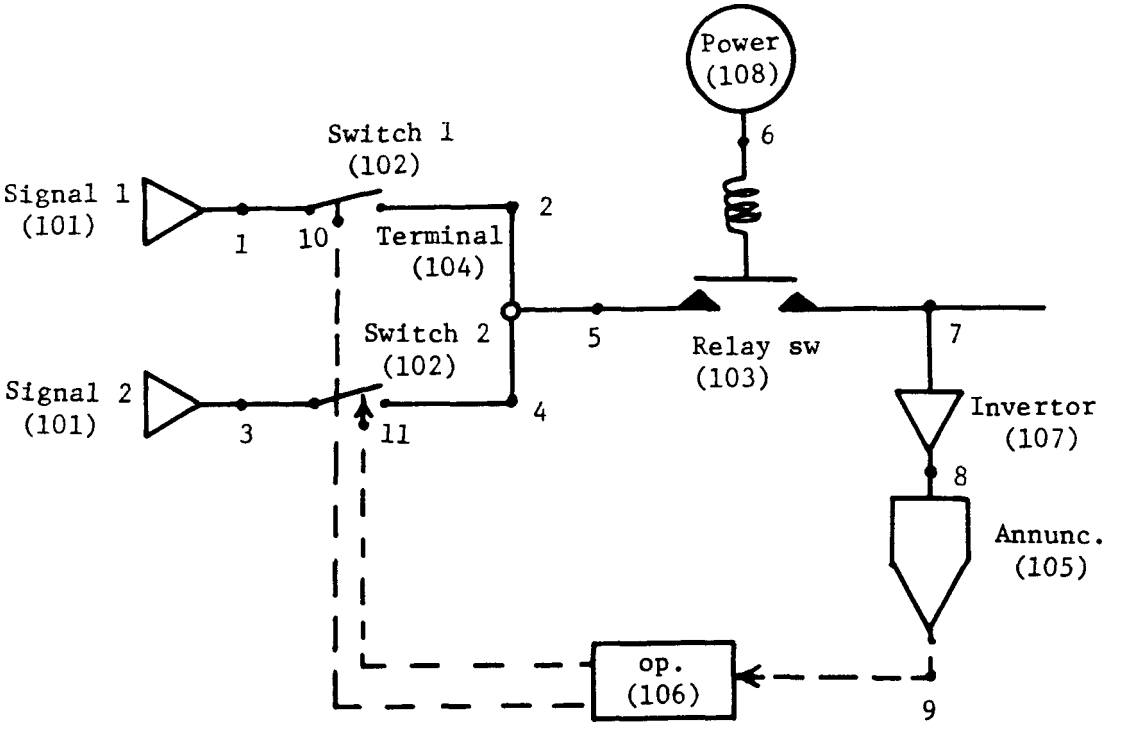



Figure C-1. Sample System

2) the internal "position" column has been changed to an external operator input. The states of this column have been changed to be:

0: operator leaves switch open

1: operator closes switch.

This switch is defined as type 102, and is shown in Table C-2.

TABLE C-2. ON/OFF SWITCH (102)

| Row | Signal Input | Operator Input | Internal | Output |
|---|---|---|---|---|
| 1 | 0 | -1 | -1 | 0 |
| 2 | -1 | 0 | 0 | 0 |
| 3 | -1 | -1 | 1 | 0 |
| 4 | 1 | -1 | 2 | 1 |
| 5 | 1 | 1 | 0 | 1 |

The relay switch is identical to that developed in section 4.3.2 of reference C-1 and operates as follows. The relay contacts are normally open unless held closed by a signal input to the coil. If power is input to the coil, and if the coil and contacts are good, the contacts will close and transmit a signal. The contacts will also remain closed if they have failed shorted (state 2), or if the coil has shorted (state 2). However, if no power is present, or if the coil or contacts fail open, no signal will be transmitted. This relay switch is modeled by Table C-3 (type 103).

TABLE C-3. RELAY SWITCH (103)

| Row No. | Signal Input | Coil Input | Internal Coil | Internal Contacts | Output Signal |
|---|---|---|---|---|---|
| 1 | 1 | -1 | -1 | 2 | 1 |
| 2 | 1 | 1 | 0 | 0 | 1 |
| 3 | 1 | -1 | 2 | 0 | 1 |
| 4 | 0 | -1 | -1 | -1 | 0 |
| 5 | -1 | -1 | -1 | 1 | 0 |
| 6 | -1 | 0 | -1 | 0 | 0 |
| 7 | -1 | -1 | 1 | 0 | 0 |

As described previously, the junction (nodes 2, 4 and 5) acts as a simple OR gate defined by Table C-4 (type 104).

TABLE C-4. JUNCTION (OR gate, type 104)

| Row | Input 1 | Input 2 | Output |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 2 | 1 | -1 | 1 |
| 3 | -1 | 1 | 1 |

The Annunciator model is that shown in Appendix B, and is defined as type 105 in this example. Its function is to produce a signal in the presence of an input. Failure states are 4 (fails to operate on demand) and 6 (operates spuriously).

The operator is modeled very simply in Table C-5 (type 106) receiving an input from the annunciator, and sending outputs to (opening or closing) switches 1 and 2. This model assumes complete common mode action by the operator: if he takes correct action, he will close both switches. Operator error (state 5003) will imply leaving both

switches open (row 2). However, for this example, it will be assumed that, even if the operator has left the switches open, a signal from the annunciator will cause him to close them correctly (row 3). A more detailed model might include other operator failure modes to represent closing only one switch, or leaving switches open even in the presence of an alarm.

TABLE C-5. OPERATOR (106)

| Row No. | Annun. Input | Operator Mode | Output 1 | Output 2 |
|---|---|---|---|---|
| 1 | -1 | 0 | 1 | 1 |
| 2 | 0 | 5003 | 0 | 0 |
| 3 | 1 | 5003 | 1 | 1 |

The invertor (type 107) is modeled by the Invertor Amplifier in Appendix B. This model produces a positive signal when no input is present, and can fail in either of two modes: 1 (fails open), or 1002 (shorts to power-always produces an output).

Finally, the power supply model (type 108) is identical to the battery model in Appendix B, with the failure modes 1 (fails open) and 1003 (power surge). Note that, even though this model has an output state of 2, not used by the relay coil input, this row has been left in the table to show that unused states in a decision table can be left without any difficulty.

The preceding decision tables were used to construct a fault tree for the TOP event "No output at node 7," under the stated boundary conditions "signals present at nodes 1 and 3." This tree is shown in Figure C-2. The explicit manner in which these decision tables were

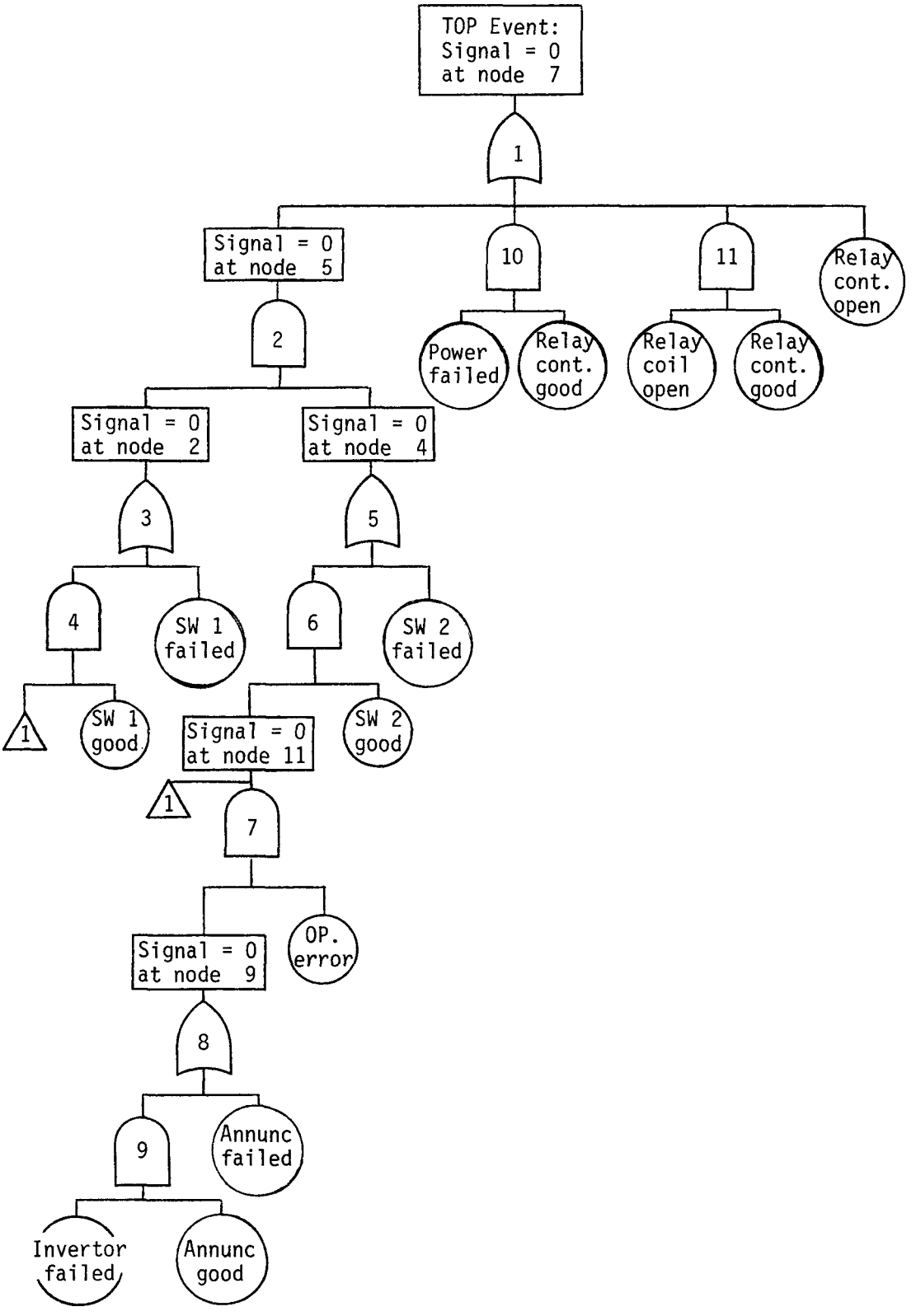



Figure C-2. Fault Tree for Sample System

used in the fault tree construction process can be seen, for example, by considering the structure of gate 1.

The construction of gate 1 begins by first locating node 7, at which the TOP event is defined, as the output of the Relay Switch, type 103. Thus, the TOP event becomes "output = 0 from Relay Switch" which, as seen from Table C-3 (type 103) corresponds to rows 4, 5, 6 and 7. Thus gate 1 becomes an OR gate with these four rows as inputs. Specifically, these four rows correspond to the following events:

row 4: no input to Relay Switch (i.e., signal = 0 at node 5),

row 5: relay contacts failed open,

row 6: no power input AND contacts good,

row 7: coil failed open AND contacts good.

Thus, row 4 requires further backtracking, leading to the events beneath gate 2, and rows 6 and 7 become AND gates 10 and 11 when these rows are eventually developed. (The reason rows 5 through 7 are rearranged in the final tree is that CAT structures all gates so that gate inputs precede primary inputs.)

The explicit appearance of good components in the fault tree results from the specific decision tables used. Referring to the above example, row 7 requires the coil to fail open and contacts to be good in order to assure no output at node 7. That is, coil failure will not lead to signal loss if the contacts have failed shorted. However, the probability of contacts failing shorted is generally negligibly small compared to the probability of being good so that, in the interests of simplifying fault trees, analysts generally leave such states out. This is also useful in computer analyses since roundoff errors may greatly reduce the

accuracy of numbers such as (1-Q) where Q is a small number. Furthermore, fault tree analysis codes such as KITT may not be able to correctly treat such good states (which are examples of NOT gates), except when using fixed probabilities.

Although these good states may be removed at this stage of the analysis, it is generally simpler to eliminate them from the decision tables before constructing the tree. This can be done by replacing '0' states by '-1' in the internal columns; however, some caution should be used to insure that no potentially significant errors be introduced. Furthermore, it must then be remembered that success trees cannot be constructed using these tables.

In this manner, a set of decision tables was derived by removing good states from the originally developed tables. For example, Table C-6 is the table which results for the Relay Switch (type 103). This set of tables was used with the identical system configuration, TOP event and boundary conditions as before, to produce the fault tree in Figure C-3. This is the example which has been used for the sample input and output of Appendices D and E. These appendices can be consulted to see the form of the new decision tables. In addition, the output in Appendix E may be used to trace the full construction of the fault tree, as was done above for the upper gate of Figure C-2.

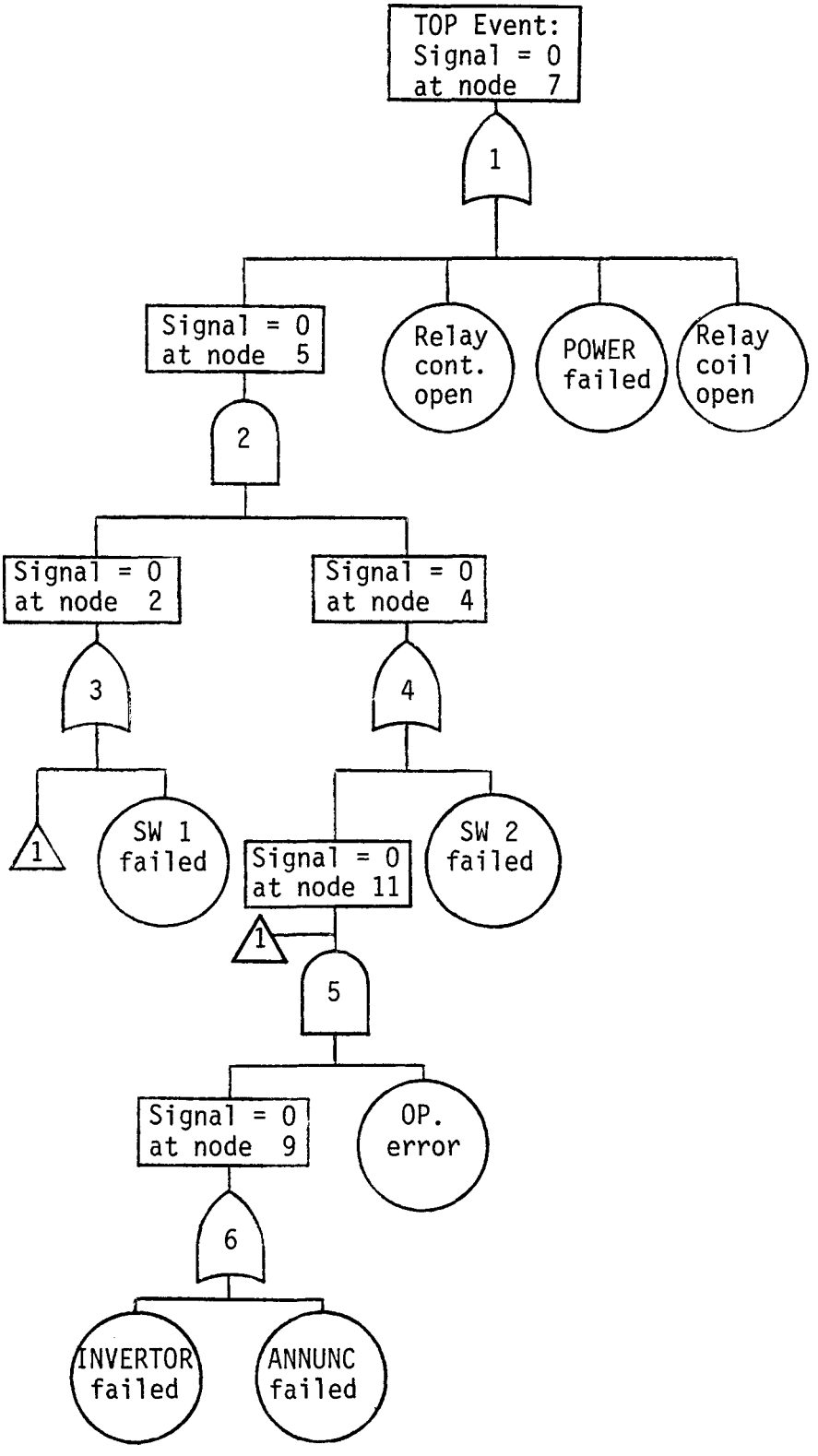



Figure C-3. Fault Tree for Sample System with Good States Removed

TABLE C-6. RELAY SWITCH WITH GOOD STATES REMOVED

| Row No. | Signal Input | Coil Input | Internal Coil | Internal Contacts | Output Signal |
|---|---|---|---|---|---|
| 1 | 1 | -1 | -1 | 2 | 1 |
| 2 | 1 | 1 | -1 | -1 | 1 |
| 3 | 1 | -1 | 2 | -1 | 1 |
| 4 | 0 | -1 | -1 | -1 | 0 |
| 5 | -1 | -1 | -1 | 1 | 0 |
| 6 | -1 | 0 | -1 | -1 | 0 |
| 7 | -1 | -1 | 1 | -1 | 0 |

A comparison of Figures C-2 and C-3 shows them to be identical, save for the deletion of the good states. Figure C-3 is significantly smaller and easier to analyze, either by hand or by computer. Furthermore, it corresponds more closely to manually constructed fault trees used in safety/reliability analyses.

The final step in this analysis was to use the computer code PREP to find the minimal cut sets for the fault tree produced by CAT. As shown by the PREP output in Appendix E, these cut sets were:

1: (Power failed)

2: (Relay coil failed open)

3: (Relay contacts failed open)

4: (Switch 2 failed open, switch 1 failed open)

5: (Annunciator failed, operator error)

6: (Invertor failed, operator error).

That these are, in fact, the complete minimal cut sets can be seen by a careful evaluation of Figure C-3.

## REFERENCES

C-1 Salem, S. L., G. E. Apostolakis and D. Okrent, A Computer-Oriented Approach to Fault-Tree Construction, EPRI-288, Palo Alto, November 1976.

# APPENDIX D

## SAMPLE INPUT

This appendix contains the complete input used to produce the sample fault tree (Figure C-3) in Appendix C. All cards follow the organization and formats as described in Section 2.

The input begins with a title card, followed by the control data in the '&DAT' section. The 'DAT1' card specifies IJOB = 1, IPRINT = 2 (intermediate level of printout), KOUT = 1 (PREP output desired), IEDIT = 0 (full editing desired), and IOT = 10 (write PREP output on I/O device 10). 'DAT2' specifies NLIB = 8 (eight decision tables in library), LNROW = 5 (length of longest decision table row), MAXINT = 2 (maximum number of internal columns in any table), and MXNROW = 40 (approximate number of rows in total decision table library). The 'DAT3' card specifies NNCMP = 10 (10 components in system) and NNODE = 11 (highest node number in system). Finally, 'DAT4' indicates MROW = 1 (1 row in the TOP event table). Notice that the '&DAT' section is the only one in which the numbers in column 5 (e.g., 'DAT1', 'DAT2', etc.) are required for identification.

The next section, beginning with the '&LIB' card which follows the END card, contains the decision table library itself. Referring to the models discussed in Appendix C, the first library entry ('LIBR0101') identifies component type 101, named "SIGNALON," which has 0 inputs, 1 internal column, 1 output and 2 rows. The single internal column is named "SIGNAL" on the 'MODE' card. This is followed by the two 'ROW' cards which contain the entries for the decision table itself, terminating with an 'END' card. All other decision tables are input similarly.

Note that component type 104, which does not have an internal column, still requires a 'MODE' card, which has been left blank. Furthermore, the numbers in columns 5-8 of the 'LIBR' and 'ROW' cards are for user identification and are not required. Finally, the specific library type numbers, and order of entry, are completely arbitrary; the library itself may contain any number of component types in any order, including some which may not be required by the specific systems being analyzed.

The '&CMP' section specifies the system components, types, and node numbers as shown in Figure C-1 of Appendix C. For example, component 1, named "SIGNAL 1" is defined as type 101, having a single node, numbered 1. Using the library as input in the '&LIB' section, type 101 is seen to have no inputs and a single output; thus this output has been assigned to node 1. Referring to component 4, "SWITCH 1" type 102 (2 inputs and 1 output), node 1 is seen to be an input in this case, along with input node 10 and output node 2.

The following data, beginning with '&TOP' define the TOP event. The 'TTOP' card specifies the event name "SAMPLE 1," the number of rows (1) and nodes (1) of the TOP event decision table, and the node number itself (7). The single row of this table is input on the card labeled 'TOP1' which indicates a state of zero to be defined at node 7 as the TOP event.

Next, the '&BC' data section includes the boundary conditions as discussed in Appendix C. Since the boundary conditions are defined at system nodes, an 'EXT' card is used. This card specifies two sets of boundary conditions: (1,1) - node 1, state = 1, and (3,1) - node 3, state = 1.

Finally, the PREP data are input following the '&OUT' card. The first two cards include the PREP control information, as discussed in Section 2.2.8. For this specific job, 1000 Monte Carlo simulations are specified. Note that NG (input arbitrarily as 1) will be replaced by the actual number of gates produced by CAT.

The final PREP inputs are the failure and repair rates. Although only those components and failure modes are included which actually appear in the fault tree, all components and modes could have been included, and only those actually needed would have been used. Should any data have been omitted, the full fault tree and all but the missing data, would be output on I/O device 'IOT.' Thus, any missing data could be included at a later time. An example of this input can be seen by referring to the relay switch, which has two internal failure columns (1 = coil and 2 = contacts), and a failure state of 1 (failed open) specified for each. Note that a failure rate of $1.0 \times 10^{-6}$ $hr^{-1}$ has been assigned to the coil, and a rate of $0.1 \times 10^{-6}$ to the contacts, while a repair time of 2.0 hours has been used for both. Note also that no failure rates have been included for failure states of 2 (failed shorted). Additional examples are the separate input data shown for SWITCH 1 and SWITCH 2. This illustrates that, although the types of two components may be identical, the failure data are specified separately, and may, in fact, be different.

```
SAMPLE CASE FOR CAT CODE
&DAT
 DAT1     1     2     1     0    10
 DAT2     8     5     2    40
 DAT3    10    11
 DAT4     1
END
&LIB
LIBR0101  SIGNALCN      101     0     1     1     2
 MODE     SIGNAL
 ROW1                     0     1
 ROW2                     1     0
END
LIBR0102  SW-TYPE1      102     2     1     1     5
 MODE     INTERNAL
 ROW1                     0    -1    -1     0
 ROW2                    -1     0    -1     0
 ROW3                    -1    -1     1     0
 ROW4                     1    -1     2     1
 ROW5                     1     1    -1     1
END
LIBR0103  SW-TYPE2      103     2     2     1     7
 MODE     COIL       CONTACT
 ROW1                     1    -1    -1     2     1
 ROW2                     1     1    -1    -1     1
 ROW3                     1    -1     2    -1     1
 ROW4                     0    -1    -1    -1     0
 ROW5                    -1    -1    -1     1     0
 ROW6                    -1     0    -1    -1     0
 ROW7                    -1    -1     1    -1     0
END
LIBR0104  JUNCTION      104     2     0     1     3
 MODE
 ROW1                     0     0     0
 ROW2                     1    -1     1
 ROW3                    -1     1     1
END
LIBR0105  ANNUNC.       105     1     1     1     4
 MODE     INTERNAL
 ROW1                     0    -1     0
 ROW2                    -1     4     0
 ROW3                     1    -1     1
 ROW4                    -1     6     1
END
LIBR0106  OPERATOR      106     1     1     2     3
 MODE     OP ERROR
 ROW1                    -1    -1     1     1
 ROW2                     0  5003     0     0
 ROW3                     1  5003     1     1
END
LIBR0107  INVERTOR      107     1     1     1     4
 MODE     INTERNAL
 ROW1                     1    -1     0
 ROW2                    -1     1     0
 ROW3                     0    -1     1
 ROW4                    -1  1002     1
END
LIBR0108  POWER         108     0     1     1     3
```

Figure D-1. Data Deck for Sample Case (page 1)

```
 MODE     INTERNAL
 ROW1                  -1    1
 ROW2                   3    0
 ROW3                1003    2
END
&CMP
 COM1     SIGNAL 1    101    1
 COM2     SIGNAL 2    101    3
 COM3     TERMINAL    104    2    4    5
 COM4     SWITCH 1    102    1   10    2
 COM5     SWITCH 2    102    3   11    4
 COM6     RELAY SW    103    5    6    7
 COM7     POWER       108    6
 COM8     INVERTOR    107    7    8
 COM9     CP.         106    9   10   11
 COM10    ANNUNC.     105    8    9
END
&TOP
TTOP      SAMPLE 1      1    1    7
 TOP1                   0
END
&BC
 EXT                    1    1    3    1
END
&OUT
         1          0          0          0          0
         1          1       100C          2                0.0
SWITCH 1          1.0        1.0    1    1
SWITCH 2          1.0        1.0    1    1
RELAY SW          1.0        2.0    1    1
RELAY SW          0.1        2.0    2    1POWER              3.0        5.0    1    3
INVERTOR          1.0        2.0    1    1OP.                0.0      0.001    1 5003
ANNUNC.           1.0        1.0    1    4
END
&END
```

Figure D-1. Data Deck for Sample Case (page 2)

# APPENDIX E

## SAMPLE OUTPUT

The following 18 pages contain the complete output for the sample fault tree of Appendix C. This printout was produced with IPRINT = 2 so that an intermediate amount of printout during the fault tree construction was produced. In addition, 4 sample pages from the PREP run made using the CAT output have been included.

The first 4 pages represent essentially a direct printout of the input, with certain headings and messages added. These pages can be compared with Section 2 (input description) and the sample input in Appendix D for reference. This output was produced directly by the sample input shown in that appendix.

The fifth page, "Component Indexing Printout," is a cross-reference of the components which were input on page 4. Following the index number and name of each component is the type number of the corresponding decision table, as input for that component (see column 3), and the type name itself. Furthermore, for each input node of a specific component, the component which is input to that node has been looked up and printed out. In the case of a component with no inputs, the statement "This component has no inputs" is printed instead. For example, component number 4 (switch 1) has two input nodes, numbered 1 and 10 on page 4. Here the components whose outputs are connected to these nodes have also been printed: 'SIGNAL 1' and 'OP.'. This is an aid to error-checking to the user, as well as a quick reference when modifying a system.

A further cross reference follows on page 6 ("Output Node Cross-Index"). Each node which has been defined is referenced, along with

the index and name of the component whose output has been assigned to that node. Finally, for use with components with multiple outputs, the specific output connected to the node of interest is listed. Here the only multiple-output component is the operator, assigned to nodes 10 (output 1) and 11 (output 2). The message at the bottom of the page tells the amount of storage already used by the library and system inputs. The amount of space remaining and the number of gates it will store are computed here, and the appropriate arrays are so dimensioned within the program.

The final cross reference is produced on the page labeled "Internal Node Index." This is a listing of the internal nodes, as numbered by the program, and is used to interpret the numerical inputs to gates produced in the remaining output (see Appendix A.2). Notice that, for this case, 11 nodes were defined in the system itself. Thus, number 12 was assigned by CAT as the first of the internal component failure mechanisms. As an example, the numbers 22 and 23 refer to the two failure mechanisms of component 6, the relay switch.

The last page of input information is the definition of the TOP event and specification of boundary conditions. This page is essentially a listing of the input cards with additional information. In the event of errors, appropriate messages would be produced here. As noted in Appendix D, the TOP event is defined as a state of zero at a single node, node 7. This, the TOP event decision table has a single row, defining state '0' at this node.

Pages 9-13 ("CAT Gate Printout Section") represent the construction and preliminary and intermediate editing phases of the actual fault tree

construction for IPRINT = 2. As an example, the first two blocks of information on the first of these pages represent the preliminary development of gates 1 and 2; an initial negative value within a set of parentheses indicates an input which has not yet been developed.

In order to understand the undeveloped inputs, a distinction must be made between AND and OR gates. Each input to a gate consists of a pair of numbers. The inputs to AND gates consist of a node number, followed by the signal state to be traced at this point. Thus gate 1, as defined by the TOP event, will investigate a signal state of 0 at node 7. The preliminary inputs to OR gates contain a component index number, followed by the specific row to be developed. Since the TOP event of the sample case is a single input gate (no signal at node 7) this can be immediately traced to the appropriate component (component 6), and the TOP event thus becomes: "no output from component 6." Since the decision table for component 6 (RELAY SW, type 103) has four rows with zero outputs (rows 4-7), the TOP event can be replaced by an OR gate, with rows 4-7 of component 6 as undeveloped inputs (see Section 3.4). Thus the preliminary printout of gate 1 shows inputs (-6,4), (-6,5), (-6,6), (-6,7), corresponding to undeveloped inputs from rows 4, 5, 6 and 7 of component 6.

In addition to gate construction, editing phases are included here, as indicated by asterisks. Thus, for example, the first input to gate 1, consisting of row 4 of decision table type 103, is seen to have only a single entry other than -1. This entry is signal = 0 at node 5. Since single input gates can be eliminated and directly replaced by their input in the gate above, gate 2 (an AND gate) is replaced by the input "signal = 0 at node 5" as the first entry in gate 1.

Backtracking further, node 5 is the output from component 3 (type 104). Since only row 1 of type 104 has a zero output, this becomes a single input OR gate, which is then eliminated, and its input inserted into gate 1. Thus "signal = 0 at node 5" is replaced by an AND gate with inputs: "signal = 0 at node 2 AND signal = 0 at node 4." This is shown by "GATE 2 TYPE = AND" with the inputs: (-2,0), (-4,0), indicating undeveloped entries of states 0 at nodes 2 and 4.

As each branch of a gate is completed, an intermediate printout of that gate appears. For example, on the second page of the fault tree construction printout, branch 1 of gate 5 is completed, followed by the printout:

GATE 5: -2 1 1 9 0 (26, 1), (-10, 2),

The initial '-2' indicates an OR gate (-1 = AND), and the negative sign implies that the gate has not yet been completed. The four succeeding values are the numbers of gates and primary events input, followed by the event being developed by gate 2. In this case it is "signal = 0 at node 9" and is the event which will be placed in the rectangle above the OR gate in the final fault tree. Finally, the two inputs to gate 2 are printed. The first has been completed (26, 1), and is the primary input "invertor failed;" that is, input 26 is the internal state of the invertor, (see Internal Node Index), and state 1 is the failed state. Notice that the second input to gate 2, (-10, 2), has not yet been developed and so is the same as before.

Skipping to the final page of the construction phase it is seen that gate 9 is the last to be constructed. It, in fact, is a single input gate which is reduced to a direct primary input, leaving only 8

gates. This leads back to gate 1 which, when completed, signals the completion of the fault tree.

The next three pages are printouts of the completed fault tree. The first shows the fault tree following the preliminary and intermediate editing. The second page reproduces the tree after transfers have been removed. In the sample case, it is seen that gates 4 and 5 have been eliminated and replaced by gates 7 and 8 (which are identical to the replaced gates). Finally, the third page lists the final tree after the gates have been renumbered consecutively to fill in the missing gap.

The gate listings on these three pages consist of the following information, which is similar to the intermediate information produced during gate construction: first, the gate type, (1 = AND, 2 = OR); second and third, the number of gates and primary events input to the gate; fourth and fifth, the event developed by the gate. This event can be of two types. For numbers below 10,000, the first is a node number and the second is a signal state to be evaluated at that node. For a number over 10,000, the first minus 10,000 is the component number, and the second is the row number of the corresponding component type being evaluated. (This does not occur in the sample case.)

Finally, the inputs to the gate are of two kinds. First are the succeeding gates input, as indicated by a -1 in the second position. Then come the primary inputs: first the internal component node (see Internal Node Index), followed by the failure state itself.

Since KOUT = 1, this output is followed by the PREP output printout and cross index (see Section 2.3). The first of these two pages is an exact printout of the data to be written or punched on I/O device

"IOT." This includes three sets of data:

1) * DATA, followed by the control data for PREP [E-1];

2) * TREE, followed by the fault tree just constructed;

3) * RATES, followed by failure and repair data as required by PREP.

This data is described in Sections 2.2.8 and 2.3, and reference E-1.

Two special considerations must be discussed here. First, PREP can accept a maximum of only seven inputs per gate. Should the CAT code produce a gate with more than seven inputs, this gate will be split into two smaller gates here, and input into a single gate above. Thus, should "GATE00nn' have too many inputs, gates numbered 'GAT100nn' and 'GAT200nn' would be created, each containing half the total gates, and each input into 'GATE00nn'.

A second consideration is that the PREP code allows only a single failure mode for each component. As discussed in Section 2.3, a unique component name is constructed for each failure state of each internal column of each component which occurs in the fault tree. This "component" is then incorporated into the tree, wherever needed, and is included in a crossindex on the next page. Thus, failure state 1 (state "0001") of component 4 (internal column node number 18) becomes component "00180001," and the failure and repair data for component 4, internal column 1, state 1, is included in the '* RATES' section. If data is included in the '&OUT' section of the CAT input for components or states not included in the final fault tree, a message will be printed and the data will not be output to PREP. If data is missing, a similar warning will be printed.

The final four pages show some of the PREP output actually produced

using the CAT output. The first page shows the control data, and the second the fault tree as input by CAT. This is followed by a listing of the component names and failure rates, and finally the minimal cut sets found by the code. For the sample case there were six minimal cut sets, as expected.

# REFERENCES

E-1 Vesely, W. E. and R. E. Narum, "PREP and KITT: Computer Codes for the Automatic Evaluation of a Fault Tree," Idaho Nuclear Corporation, Idaho Falls, Idaho, IN-1349, 1970.

```
-------------------------------------------------------------
- PROGRAM CAT,  VERSION OF 10/75
- PROGRAM FOR THE AUTOMATED CONSTRUCTION OF FAULT TREES.
-
- SAMPLE CASE FOR CAT CODE
-------------------------------------------------------------


*** DATA VALIDATION SECTION ***


&DAT
 DAT1     1     2     1     0    10
 DAT2     8     5     2    40
 DAT3    10    11
 DAT4     1
END

NUMBER OF LIBRARY ROWS ESTIMATED (MXNROW) =    40 ROWS.
ALLOCATED SPACE REMAINING        (MXROW2) =  1989 ROWS.
```

```
*** SAMPLE CASE FOR CAT CODE                                          ***

    LIBRARY INPUT PRINTOUT

&LIB


LIBRARY DATA PRINTOUT FOR COMPONENT   1
          NAME        TYPE   MIN  MINI  NOUT  NBCH
LIBRC101  SIGNALCN     101     0     1     1     2
 MODE     SIGNAL
 ROW1                    0     1
 RCW2                    1     0
END



LIBRARY DATA PRINTOUT FOR COMPONENT   2
          NAME        TYPE   MIN  MINI  NOUT  NBCH
LIBR0102  SR-TYPE1     102     2     1     1     5
 MODE     INTERNAL
 ROW1                    0    -1    -1     0
 ROW2                   -1     0    -1     0
 ROW3                   -1    -1     1     0
 RCW4                    1    -1     2     1
 ROW5                    1     1    -1     1
END



LIBRARY DATA PRINTOUT FOR COMPONENT   3
          NAME        TYPE   MIN  MINI  NOUT  NBCH
LIBR0103  SR-TYPE2     103     2     2     1     7
 MODE     COIL       CONTACT
 RCW1                    1    -1    -1     2     1
 ROW2                    1     1    -1    -1     1
 ROW3                    1    -1     2    -1     1
 ROW4                    0    -1    -1    -1     0
 ROW5                   -1    -1    -1     1     0
 ROW6                   -1     0    -1    -1     0
 ROW7                   -1    -1     1    -1     0
END



LIBRARY DATA PRINTOUT FOR COMPONENT   4
          NAME        TYPE   MIN  MINI  NOUT  NBCH
LIBR0104  JUNCTICN     104     2     0     1     3
 MODE
 ROW1                    0     0     0
 RCW2                    1    -1     1
 ROW3                   -1     1     1
END
```

```
LIBRARY DATA PRINTOUT FOR COMPONENT   5
          NAME        TYPE   NIN NINT NOUT NROW
LIBR0105  ANNUNC.      105     1    1    1    4
 MODE     INTERNAL
 ROW1                    0    -1    0
 ROW2                   -1     4    0
 ROW3                    1    -1    1
 ROW4                   -1     6    1
END
```

```
LIBRARY DATA PRINTOUT FOR COMPONENT   6
          NAME        TYPE   NIN NINT NOUT NROW
LIBR0106  OPERATOR     106     1    1    2    3
 MODE     OF ERROR
 ROW1                   -1    -1    1    1
 ROW2                    0  5003    0    0
 ROW3                    1  5003    1    1
END
```

```
LIBRARY DATA PRINTOUT FOR COMPONENT   7
          NAME        TYPE   NIN NINT NOUT NROW
LIBR0107  INVERTOR     107     1    1    1    4
 MODE     INTERNAL
 ROW1                    1    -1    0
 ROW2                   -1     1    0
 ROW3                    0    -1    1
 ROW4                   -1  1002    1
END
```

```
LIBRARY DATA PRINTOUT FOR COMPONENT   8
          NAME        TYPE   NIN NINT NOUT NROW
LIBR01C8  POWER        108     0    1    1    3
 MODE     INTERNAL
 ROW1                   -1     1
 ROW2                    3     0
 ROW3                 1003     2
END
```

SCRP CARD FOUND.  DATA VALIDATION CONTINUING.

SAMPLE CASE FOR CAT CODE

COMPONENT INDEX INPUT PRINTOUT

COMPONENT CARD PRINTOUT

| INDEX | CODE | NAME | TYPE | INPUT/OUTPUT NODES | | |
|---|---|---|---|---|---|---|
| 1 | COM1 | SIGNAL 1 | 101 | 1 | | |
| 2 | COM2 | SIGNAL 2 | 101 | 3 | | |
| 3 | COM3 | TERMINAL | 104 | 2 | 4 | 5 |
| 4 | COM4 | SWITCH 1 | 102 | 1 | 10 | 2 |
| 5 | COM5 | SWITCH 2 | 102 | 3 | 11 | 4 |
| 6 | COM6 | RELAY SW | 103 | 5 | 6 | 7 |
| 7 | COM7 | POWER | 108 | 6 | | |
| 8 | COM8 | INVERTOR | 107 | 7 | 8 | |
| 9 | COM9 | CP. | 106 | 9 | 10 | 11 |
| 10 | COM10 | ANNUNC. | 105 | 8 | 9 | |

END

COMPONENT INDEXING PRINTOUT FOR:
SAMPLE CASE FOR CAT CODE

| INDEX | NAME | TYPE# | TYPE | INPUTS FROM: NODE/NAME |
|---|---|---|---|---|
| 1 | SIGNAL 1 | 101 | SIGNALON | THIS COMPONENT HAS NO INPUTS |
| 2 | SIGNAL 2 | 101 | SIGNALON | THIS COMPONENT HAS NO INPUTS |
| 3 | TERMINAL | 104 | JUNCTION | 2/SWITCH 1 4/SWITCH 2 |
| 4 | SWITCH 1 | 102 | SW-TYPE1 | 1/SIGNAL 1 10/OP. |
| 5 | SWITCH 2 | 102 | SW-TYPE1 | 3/SIGNAL 2 11/OP. |
| 6 | RELAY SW | 103 | SW-TYPE2 | 5/TERMINAL 6/POWER |
| 7 | POWER | 108 | POWER | THIS COMPONENT HAS NO INPUTS |
| 8 | INVERTOR | 107 | INVERTOR | 7/RELAY SW |
| 9 | OP. | 106 | OPERATOR | 9/ANNUNC. |
| 10 | ANNUNC. | 105 | ANNUNC. | 8/INVERTOR |

OUTPUT NODE CROSS-INDEX FOR:
SAMPLE CASE FOR CAT CODE

| OUTPUT NODE | COMPONENT: INDEX | NAME | OUTPUT NO. |
|---|---|---|---|
| 1 | 1 | SIGNAL 1 | 1 |
| 2 | 4 | SWITCH 1 | 1 |
| 3 | 2 | SIGNAL 2 | 1 |
| 4 | 5 | SWITCH 2 | 1 |
| 5 | 3 | TERMINAL | 1 |
| 6 | 7 | POWER | 1 |
| 7 | 6 | RELAY SW | 1 |
| 8 | 8 | INVERTOR | 1 |
| 9 | 10 | ANNUNC. | 1 |
| 10 | 9 | OR. | 1 |
| 11 | 9 | OR. | 2 |



**********
* STORAGE SPACE SUMMARY
*  ARRAY 'MAT' HAS USED  375 WORDS.
*  REMAINING  9625 WORDS WILL ACCOMMODATE  740 GATES WITH AN AVERAGE OF 3.0 INPUTS PER GATE.
**********

INTERNAL NODE INDEX FOR:
SAMPLE CASE FOR CAT CODE

| INDEX | NAME | TYPE# | TYPE | INTERNAL NODES | |
|---|---|---|---|---|---|
| 1 | SIGNAL 1 | 101 | SIGNALON | 12: SIGNAL | |
| 2 | SIGNAL 2 | 101 | SIGNALON | 14: SIGNAL | |
| 3 | TERMINAL | 104 | JUNCTION | | |
| 4 | SWITCH 1 | 102 | SW-TYPE1 | 18: INTERNAL | |
| 5 | SWITCH 2 | 102 | SW-TYPE1 | 20: INTERNAL | |
| 6 | RELAY SW | 103 | SW-TYPE2 | 22: COIL | 23: CONTACT |
| 7 | POWER | 108 | POWER | 24: INTERNAL | |
| 8 | INVERTOR | 107 | INVERTOR | 26: INTERNAL | |
| 9 | OP. | 106 | OPERATOR | 28: OP ERROR | |
| 10 | ANNUNC. | 105 | ANNUNC. | 30: INTERNAL | |

```
*****************************************************************
* PROGRAM CAT,  VERSION CP 10/75
* TOP EVENT AND BOUNDARY CONDITION PRINTOUT FOR JOB    1
*
* SAMPLE CASE FOR CAT CODE
*****************************************************************


&TOP

TOP EVENT FOR SAMPLE CASE FOR CAT CODE


EVENT:  SAMPLE 1
NUMBER OF ROWS  =  1
NUMBER OF NODES =  1
TOP EVENT NODES =

NODE   COMPONENT     OUTPUT

   7     6   RELAY SW   1


TABLE FOR TOP EVENT:

TTOP      NODE(S):      7
 TOP1                   0
END



&BC

BOUNDARY CONDITIONS FOR SAMPLE CASE FOR CAT CODE


 EXT                   1    1    3    1
END
```

```
-------------------------------------------------------------------------------------------------------------------------------
-                                                                                                                             -
- SAMPLE CASE FOR CAT CODE                                                                                                    -
-                                                                                                                             -
- CAT  GATE PRINTOUT SECTION.                                                                                                 -
-                                                                                                                             -
-------------------------------------------------------------------------------------------------------------------------------

*** PRELIMINARY CATB PRINTOUT ***


GATE   1   TYPE = OR
           NUMBER OF GATES INPUT = 4 NUMBER OF PRIMARY INPUTS = 0
           EVENT:  TOP EVENT
           INPUTS: (    -6,     4),(    -6,     5),(    -6,     6),(    -6,     7),(

    GATE   2: MODE     7 PRESET TO MODE =     0 BY INITIAL CONDITIONS.

  * GATE   2 TYPE = AND HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   1.

  * GATE   2 TYPE = OR  HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   1.

GATE   2   TYPE = OR
           NUMBER OF GATES INPUT = 1 NUMBER OF PRIMARY INPUTS = 0
           EVENT:  SIGNAL =     0 AT MODE     5
           INPUTS: (    -3,     1),(

    GATE   2: MODE     5 PRESET TO MODE =     0 BY GATE   1.

GATE   2   TYPE = AND
           NUMBER OF GATES INPUT = 2 NUMBER OF PRIMARY INPUTS = 0
           EVENT:  SIGNAL =     0 AT MODE     5
           INPUTS: (    -2,     0),(    -4,     0),(

    GATE   3: COMPONENT 'SWITCH 1' TYPE     2, HON  1 MODE     1: MODE =     0 CONTRADICTS MODE =     1, SET BY INITIAL CONDITIONS.

GATE   3   TYPE = OR
           NUMBER OF GATES INPUT = 2 NUMBER OF PRIMARY INPUTS = 0
           EVENT:  SIGNAL =     0 AT MODE     2
           INPUTS: (    -4,     2),(    -4,     3),(

    GATE   4: MODE     2 PRESET TO MODE =     0 BY GATE   2.

  * GATE   4 TYPE = AND HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   3.

  * GATE   4 TYPE = OR  HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   3.

GATE   4   TYPE = OR
           NUMBER OF GATES INPUT = 1 NUMBER OF PRIMARY INPUTS = 0
           EVENT:  SIGNAL =     0 AT MODE    10
           INPUTS: (    -9,     2),(

    GATE   4: MODE    10 PRESET TO MODE =     0 BY GATE   3.

GATE   4   TYPE = AND
           NUMBER OF GATES INPUT = 1 NUMBER OF PRIMARY INPUTS = 1
           EVENT:  SIGNAL =     0 AT MODE    10
           INPUTS: (    -9,     0),(    28,  5003),(

GATE   5   TYPE = OR
           NUMBER OF GATES INPUT = 2 NUMBER OF PRIMARY INPUTS = 0
           EVENT:  SIGNAL =     0 AT MODE     9
           INPUTS: (   -10,     1),(   -10,     2),(
```

```
    GATE   6: MODE     9 PRESET TO MODE =     0 BY GATE   4.

  * GATE   6 TYPE = AND HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   5.

    GATE   6: COMPLEMENT 'INVERTOR' TYPE     7, SUM  1 MODE     7: MODE =     1 CONTRADICTS MODE =     0, SET BY INITIAL CONDITIONS.

  * GATE   6 TYPE = OR  HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   5.

GATE   6   TYPE = OR
           NUMBER OF GATES INPUT = 1 NUMBER OF PRIMARY INPUTS = 0
           EVENT:  SIGNAL =     0 AT NODE     8
           INPUTS: (     -8,     2),(

    GATE   6: MODE     8 PRESET TO MODE =     0 BY GATE   5.

  * GATE   6 TYPE = AND HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   5.
    GATE   5: -2  1  1     9     0 (     26,     1),(    -10,     2),(

    GATE   6: MODE     9 PRESET TO MODE =     0 BY GATE   4.

  * GATE   6 TYPE = AND HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   5.
    GATE   5: -2  0  2     9     0 (     26,     1),(     30,     4),(


*** GATE   5 COMPLETED. PRELIMINARY GATE EDIT FOLLOWS ***

    GATE   5:  2  0  2     9     0 (     26,     1),(     30,     4),(
    GATE   5:  2  0  2     9     0 (     26,     1),(     30,     4),(
    GATE   5:  2  0  2     9     0 (     26,     1),(     30,     4),(

*** PRELIMINARY EDIT OF GATE   5 COMPLETED ***



*** GATE   4 COMPLETED. PRELIMINARY GATE EDIT FOLLOWS ***

    GATE   4:  1  1  1    10     0 (      5,    -1),(     28,  5003),(
    GATE   4:  1  1  1    10     0 (      5,    -1),(     28,  5003),(
    GATE   5:  2  0  2     9     0 (     26,     1),(     30,     4),(
    GATE   4:  1  1  1    10     0 (      5,    -1),(     28,  5003),(

*** PRELIMINARY EDIT OF GATE   4 COMPLETED ***


    GATE   6: MODE     2 PRESET TO MODE =     0 BY GATE   2.

  * GATE   6 TYPE = AND HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   3.
    GATE   3: -2  1  1     2     0 (      4,    -1),(     18,     1),(


*** GATE   3 COMPLETED. PRELIMINARY GATE EDIT FOLLOWS ***

    GATE   3:  2  1  1     2     0 (      4,    -1),(     18,     1),(
    GATE   3:  2  1  1     2     0 (      4,    -1),(     18,     1),(
    GATE   5:  2  0  2     9     0 (     26,     1),(     30,     4),(
    GATE   3:  2  1  1     2     0 (      4,    -1),(     18,     1),(
    GATE   3:  2  1  1     2     0 (      4,    -1),(     18,     1),(

*** PRELIMINARY EDIT OF GATE   3 COMPLETED ***
```

```
    GATE   6: COMPONENT 'SWITCH 2' TYPE     2, ROW  1 NODE     3: NODE =    0 CONTRADICTS NODE =    1, SET BY INITIAL CONDITIONS.

GATE   6    TYPE = OR
            NUMBER OF GATES INPUT = 2 NUMBER OF PRIMARY INPUTS = 0
            EVENT:  SIGNAL =      0 AT NODE      4
            INPUTS: (     -5,      2),(     -5,      3),(

    GATE   7: NODE     4 PRESET TO NODE =    0 BY GATE   2.

  * GATE   7 TYPE = AND HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   6.

  * GATE   7 TYPE = OR  HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   6.

GATE   7    TYPE = OR
            NUMBER OF GATES INPUT = 1 NUMBER OF PRIMARY INPUTS = 0
            EVENT:  SIGNAL =      0 AT NODE     11
            INPUTS: (     -9,      2),(

    GATE   7: NODE    11 PRESET TO NODE =    0 BY GATE   6.

GATE   7    TYPE = AND
            NUMBER OF GATES INPUT = 1 NUMBER OF PRIMARY INPUTS = 1
            EVENT:  SIGNAL =      0 AT NODE     11
            INPUTS: (     -9,      0),(     28,   5003),(

GATE   8    TYPE = OR
            NUMBER OF GATES INPUT = 2 NUMBER OF PRIMARY INPUTS = 0
            EVENT:  SIGNAL =      0 AT NODE      9
            INPUTS: (    -10,      1),(    -10,      2),(

    GATE   9: NODE     9 PRESET TO NODE =    0 BY GATE   7.

  * GATE   9 TYPE = AND HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   8.

    GATE   9: COMPONENT 'INVERTOR' TYPE     7, ROW  1 NODE     7: NODE =    1 CONTRADICTS NODE =    0, SET BY INITIAL CONDITIONS.

  * GATE   9 TYPE = OR  HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   8.

GATE   9    TYPE = OR
            NUMBER OF GATES INPUT = 1 NUMBER OF PRIMARY INPUTS = 0
            EVENT:  SIGNAL =      0 AT NODE      8
            INPUTS: (     -8,      2),(

    GATE   9: NODE     8 PRESET TO NODE =    0 BY GATE   8.

  * GATE   9 TYPE = AND HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   8.
    GATE   8: -2  1  1      9      0 (     26,      1),(    -10,      2),(

    GATE   9: NODE     9 PRESET TO NODE =    0 BY GATE   7.

  * GATE   9 TYPE = AND HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   8.
    GATE   8: -2  0  2      9      0 (     26,      1),(     30,      4),(


*** GATE   8 COMPLETED. PRELIMINARY GATE EDIT FOLLOWS ***

    GATE   8:  2  0  2      9      0 (     26,      1),(     30,      4),(
    GATE   8:  2  0  2      9      0 (     26,      1),(     30,      4),(
    GATE   8:  2  0  2      9      0 (     26,      1),(     30,      4),(

*** PRELIMINARY EDIT OF GATE   8 COMPLETED ***
```

```
*** GATE   7 COMPLETED. PRELIMINARY GATE EDIT FOLLOWS ***

    GATE   7:  1  1  1    11     0 (     8,    -1),(    28,  5003),(
    GATE   7:  1  1  1    11     0 (     8,    -1),(    28,  5003),(
    GATE   8:  2  0  2     9     0 (    26,     1),(    30,     4),(
    GATE   7:  1  1  1    11     0 (     8,    -1),(    28,  5003),(

*** PRELIMINARY EDIT OF GATE   7 COMPLETED ***


    GATE   9: MODE     4 PRESET TO MODE =    0 BY GATE   2.

  * GATE   9 TYPE = AND HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   6.
    GATE   6: -2  1  1     4     0 (     7,    -1),(    20,     1),(


*** GATE   6 COMPLETED. PRELIMINARY GATE EDIT FOLLOWS ***

    GATE   6:  2  1  1     4     0 (     7,    -1),(    20,     1),(
    GATE   6:  2  1  1     4     0 (     7,    -1),(    20,     1),(
    GATE   8:  2  0  2     9     0 (    26,     1),(    30,     4),(
    GATE   6:  2  1  1     4     0 (     7,    -1),(    20,     1),(
    GATE   6:  2  1  1     4     0 (     7,    -1),(    20,     1),(

*** PRELIMINARY EDIT OF GATE   6 COMPLETED ***


*** GATE   2 COMPLETED. PRELIMINARY GATE EDIT FOLLOWS ***

    GATE   2:  1  2  0     5     0 (     3,    -1),(     6,    -1),(
    GATE   4:  1  1  1    10     0 (     5,    -1),(    28,  5003),(
    GATE   3:  2  1  1     2     0 (     4,    -1),(    18,     1),(
    GATE   7:  1  1  1    11     0 (     8,    -1),(    28,  5003),(
    GATE   6:  2  1  1     4     0 (     7,    -1),(    20,     1),(
    GATE   2:  1  2  0     5     0 (     3,    -1),(     6,    -1),(

*** PRELIMINARY EDIT OF GATE   2 COMPLETED ***


    GATE   9: MODE     7 PRESET TO MODE =    0 BY INITIAL CONDITIONS.

  * GATE   9 TYPE = AND HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   1.
    GATE   1: -2  3  1     7     0 (     2,    -1),(    23,     1),(    -6,     6),(    -6,     7),(

    GATE   9: MODE     7 PRESET TO MODE =    0 BY INITIAL CONDITIONS.

  * GATE   9 TYPE = AND HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   1.

  * GATE   9 TYPE = OR  HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE   1.
```

```
GATE   9   TYPE = OR
           NUMBER OF GATES INPUT = 1 NUMBER OF PRIMARY INPUTS = 0
           EVENT:  SIGNAL =      0 AT NODE      6
           INPUTS: (     -7,      2),(

    GATE   9: NODE      6 PRESET TO MODE =     0 BY GATE   1.

  * GATE   9 TYPE = AND HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE    1.
    GATE   1: -2  2  2      7      0 (      2,     -1),(     23,      1),(     24,      3),(     -6,      7),(

    GATE   9: NODE      7 PRESET TO MODE =     0 BY INITIAL CONDITIONS.

  * GATE   9 TYPE = AND HAS SINGLE INPUT.  GATE BEING ELIMINATED AND INPUT DIRECTLY INTO GATE    1.
    GATE   1: -2  1  3      7      0 (      2,     -1),(     23,      1),(     24,      3),(     22,      1),(


*** GATE   1 COMPLETED. PRELIMINARY GATE EDIT FOLLOWS ***


    GATE   1:   2  1  3      7      0 (      2,     -1),(     23,      1),(     24,      3),(     22,      1),(
    GATE   1:   2  1  3      7      0 (      2,     -1),(     23,      1),(     24,      3),(     22,      1),(
    GATE   3:   2  1  1      2      0 (      4,     -1),(     18,      1),(
    GATE   6:   2  1  1      4      0 (      7,     -1),(     20,      1),(
    GATE   1:   2  1  3      7      0 (      2,     -1),(     23,      1),(     24,      3),(     22,      1),(
    GATE   1:   2  1  3      7      0 (      2,     -1),(     23,      1),(     24,      3),(     22,      1),(

*** PRELIMINARY EDIT OF GATE   1 COMPLETED ***



**** 'TOP' HAS BEEN COMPLETED ****
```

```
------------------------------------------------------------------------------------------------------------
- EBCGRAM CAT,  VERSICN CF 10/75
- PROGRAM FOR THE AUTOMATED CONSTRUCTICN OF FAULT TREES.
- OUTPUT BEGICN
-
- SAMPLE CASE ECR CAT CCIE
------------------------------------------------------------------------------------------------------------


*** GATE PRINTOUT SECTICN ***

    GATE   1:  2  1  3     7     0 (     2,    -1),(    23,     1),(    24,     3),(    22,     1),(
    GATE   2:  1  2  0     5     0 (     3,    -1),(     6,    -1),(
    GATE   3:  2  1  1     2     0 (     4,    -1),(    18,     1),(
    GATE   4:  1  1  1    10     0 (     5,    -1),(    28,  5003),(
    GATE   5:  2  0  2     9     0 (    26,     1),(    30,     4),(
    GATE   6:  2  1  1     4     0 (     7,    -1),(    20,     1),(
    GATE   7:  1  1  1    11     0 (     8,    -1),(    28,  5003),(
    GATE   8:  2  0  2     9     0 (    26,     1),(    30,     4),(
```

*** GATE TRANSFERS COMPLETED ***

INTERMEDIATE PRINTOUT FOR SAMPLE CASE FOR CAT CODE

```
GATE   1:  2  1  3     7     0 (     2,    -1),(    23,     1),(    24,     3),(    22,     1),(
GATE   2:  1  2  0     5     0 (     3,    -1),(     6,    -1),(
GATE   3:  2  1  1     2     0 (     7,    -1),(    18,     1),(
GATE   6:  2  1  1     4     0 (     7,    -1),(    20,     1),(
GATE   7:  1  1  1    11     0 (     8,    -1),(    28,  5003),(
GATE   8:  2  0  2     9     0 (    26,     1),(    30,     4),(
```

```
------------------------------------------------------------------------------------------------------------
- PROGRAM CAT,  VERSION CE 10/75
- PROGRAM FOR THE AUTOMATED CONSTRUCTION OF FAULT TREES.
- OUTPUT REGION: FINAL GATE PRINTOUT
-
- SAMPLE CASE FOR CAT CODE
------------------------------------------------------------------------------------------------------------


*** FINAL GATE PRINTOUT SECTION ***


    GATE   1:  2  1  3     7     0 (     2,    -1),(    23,     1),(    24,     3),(    22,     1),(
    GATE   2:  1  2  0     5     0 (     3,    -1),(     4,    -1),(
    GATE   3:  2  1  1     2     0 (     5,    -1),(    18,     1),(
    GATE   4:  2  1  1     4     0 (     5,    -1),(    20,     1),(
    GATE   5:  1  1  1    11     0 (     6,    -1),(    28,  5003),(
    GATE   6:  2  0  2     9     0 (    26,     1),(    30,     4),(
```

```
------------------------------------------------------------------------
- PROGRAM CAT, OUTPUT REGION
- OUTPUT TO I/O DEVICE 10 IN FORMAT FOR PREP-KITT CODES
-
- SAMPLE CASE FOR CAT CODE
------------------------------------------------------------------------


SAMPLE CASE FOR CAT CODE

     *** FAULT TREE CONSTRUCTED BY CAT, VERSION OF MAY  1977 ***
* DATA
         6          0          0          0          0          0
         1          1       1000          2                  0.0
END
* TREE
TOP       OR    1 3 GATE0002 00230001 00240003 00220001
GATE0002  AND   2 0 GATE0003 GATE0004
GATE0003  OR    1 1 GATE0005 00180001
GATE0004  OR    1 1 GATE0005 00200001
GATE0005  AND   1 1 GATE0006 00285003
GATE0006  OR    0 2 00260001 00300004
END
* RATES
00180001        1.000      1.000 00200001        1.000      1.000
00220001        1.000      2.000 00230001        0.100      2.000
00240003        3.000      5.000 00260001        1.000      2.000
00285003        0.0        0.001 00300004        1.000      1.000
END
```

```
- PROGRAM CAT, SUBROUTINE OUTPUT, VERSION OF MAY 1977
- CROSS-INDEX OF COMPONENT NAMES USED FOR PREP/KITT INPUT
-
- SAMPLE CASE FOR CAT CODE
```

| PREP NAME | COMPONENT INDEX | COMPONENT NAME | INTERNAL: COLUMN: | INTERNAL: NUMBER: | INTERNAL NAME | FAILURE STATE |
|---|---|---|---|---|---|---|
| 00180001 | 4 | SWITCH 1 | 1: | 18: | INTERNAL | 1 |
| 00200001 | 5 | SWITCH 2 | 1: | 20: | INTERNAL | 1 |
| 00220001 | 6 | RELAY SW | 1: | 22: | COIL | 1 |
| 00230001 | 6 | RELAY SW | 2: | 23: | CONTACT | 1 |
| 00240003 | 7 | POWER | 1: | 24: | INTERNAL | 3 |
| 00260001 | 8 | INVERTER | 1: | 26: | INTERNAL | 1 |
| 00285003 | 9 | OP. | 1: | 28: | OP ERROR | 5003 |
| 00300004 | 10 | ANNUNC. | 1: | 30: | INTERNAL | 4 |

```
**********************************************************************
*TREBIL FAULT TREE BUILDING PROGRAM
**********************************************************************

SAMPLE CASE FOR CAT CODE


      *** FAULT TREE CONSTRUCTED BY CAT, VERSION OF MAY  1977 ***

NUMBER OF GATES,NG------------------------     6

COMBO STARTING VALUE,MIN------------------     0

COMBO ENDING VALUE,MAX--------------------     0

CUT SET - PATH SET SWITCH,IDEX1-----------     0

PRINT - DISC  SWITCH,IDEX2----------------     0

MONTE CARLO STARTER,MCS-------------------     1

NO. OF RANDOM NUMBERS TO REJECT,NREJEC-----    1

NO. OF MONTE CARLO TRIALS,NTR------------- 1000

MIXING PARAMETER SWITCH,IREN--------------     2

MONTE CARLO MIXING PARAMETER,TAA----------    0.0
```

```
**********************************************************
*TREBIL FAULT TREE BUILDING PROGRAM
**********************************************************


SAMPLE CASE FOR CAT CODE


NAME      TYPE     INPUTS----
TOP       OR    1  3 GATE0002 00230001 00240003 00220001
GATE0002  AND   2  C GATE0003 GATE0004
GATE0003  OR    1  1 GATE0005 00180001
GATE0004  OR    1  1 GATE0005 00200001
GATE0005  AND   1  1 GATE0006 00285003
GATE0006  OR    C  2 00260001 00300004
END             C  C
```

**********
*TREBIL FAULT TREE BUILDING PROGRAM *
**********

SAMPLE CASE FCF CAT CODE

COMPONENT INDICES, NAMES, AND FAILURE RATES (PER HOUR) -

| TREE INDEX | COMPONENT NAME | LAMBDA(FAILURE INTENSITY/HR.) | TAU |
|---|---|---|---|
| 1 | 00260001 | 1.00000D-06 | 2.00000D+00 |
| 2 | 00300004 | 1.00000D-06 | 1.00000D+00 |
| 3 | 00285003 | 0.0 | 1.00000D-03 |
| 4 | 00200001 | 1.00000D-06 | 1.00000D+00 |
| 5 | 00180001 | 1.00000D-06 | 1.00000D+00 |
| 6 | 00230001 | 1.00000D-07 | 2.00000D+00 |
| 7 | 00240003 | 3.00000D-06 | 5.00000D+00 |
| 8 | 00220001 | 1.00000D-06 | 2.00000D+00 |

**********
MINIMAL SETS FOR THIS TREE
**********

MINIMAL CUT SET 1 00240003

MINIMAL CUT SET 2 00220001

MINIMAL CUT SET 3 00230001

MINIMAL CUT SET 4 00200001 00180001

MINIMAL CUT SET 5 00300004 00285003

MINIMAL CUT SET 6 00260001 00285003

********** END OF OUTPUT FROM MINSET **********

# APPENDIX F

## Program List for CAT

The following is the complete listing of the FORTRAN source deck for the CAT code, including all comment cards. This listing contains the current version (as of December, 1977) and incorporates the following changes to the version described in reference 1:

1) Addition of subroutine OUTPUT, and call statement in subroutine DRIVER, to produce PREP-KITT output; this necessitates additional input in new '&OUT' section of data input if KOUT = 1;
2) deletion of array 'AMAT' and dimension 'MSIZE' from MAIN program and subroutine DRIVER;
3) replacement of paramater 'IGOOD' by 'KOUT' and addition of 'IOT' on DAT1 card and in COMMON;
4) addition of parameter 'NGATE' in calling sequence for'DO IT';
5) addition of error checking statements for new input in '&OUT' section. This requires changes in DRIVER, LIBR, INDEX and STEVE to recognize and differentiate between input sequences for KOUT = 0 and KOUT = 1. Also, changes in DO IT and DRIVER have been made to bypass PREP-KITT input and output if no fault tree is produced by CAT (e.g., if TOP event cannot occur in the fault tree desired).

```
C*************************************************************************
C                                                                        *
C       PROGRAM CAT                                                      *
C         PROGRAM FOR THE AUTOMATED CONSTRUCTION OF FAULT TREES          *
C       VERSION OF MAY,1977                                              *
C         PROGRAMMED BY S. L. SALEM, 825-2792                            *
C         CHEMICAL, NUCLEAR, AND THERMAL ENGINEERING DEPARTMENT          *
C         5532 BOELTER HALL                                              *
C         SCHOOL OF ENGINEERING AND APPLIED SCIENCE                      *
C         UNIVERSITY OF CALIFORNIA, LOS ANGELES                          *
C         LOS ANGELES, CALIFORNIA  90024                                 *
C                                                                        *
C*************************************************************************
C
C         MAIN ROUTINE (DUMMY)
C         ROUTINE TO SET MAIN ARRAY DIMENSIONS AND CALL DRIVER
      DIMENSION  MAT(10000)
      DOUBLE PRECISION NAME( 200)
      DATA LSIZE,NSIZE/ 200,10000/
      CALL DRIVER (NAME,MAT,LSIZE,NSIZE)
      STOP
      END
      SUBROUTINE DRIVER (NAME,MAT,LSIZE,NSIZE)
C
C       MAIN DRIVER ROUTINE OF PROGRAM CAT
C         ROUTINE TO READ MAIN PARAMETERS, SET UP PROGRAM DIMENSIONS
C         AND CALL PROGRAM SUBROUTINES
C         VARIABLES I1 - I15 ARE STARTING INDICES OF SUBARRAYS
C           OF MAIN INTEGER ARRAY 'MAT'
C         VARIABLES J1 - J3 ARE STARTING INDICES FOR DOUBLE PRECISION.
C
      DIMENSION MAT(NSIZE)
      DOUBLE PRECISION NAME(LSIZE),NNAME
      COMMON TITLE(20),XXXX(20),IERR,IEDIT,IDUM(12),NNAME,IPRINT,KOUT
      DATA  LDAT,MDAT,LEND,LLIB/'&DAT',' DAT','END ','&LIB'/
      DATA  LTOP,MEND,XOUT/'&TOP','&END','&OUT'/
      DATA  IJOB,NFOW/1,1/
      IPRINT = 0
      IEDIT  = 0
      KOUT   = 0
      ICT    = 10
      INGATE = 11
      IERR = 0
      JERR = 0
      READ (5,1000) TITLE
      WRITE (6,1001) TITLE
      READ (5,1002) XXXX,NCODE,I,IDUM
      IF (NCODE .EQ. LDAT) GO TO 90
   89 WRITE (6,1003) LDAT
      WRITE (6,1004) XXXX
C
C       INPUT ERROR DETECTED.  PROGRAM WILL ATTEMPT TO CONTINUE.
```

```
      IF (NCODE .NE. MDAT) IERR = 1
      IF (NCODE .EQ. LEND) GO TO 600
      IF (NCODE .EQ. MDAT) GO TO 93
      GO TO 91
   90 WRITE (6,1004) XXXX
C
C       READ INPUT IN GENERAL FORMAT.
   91 READ (5,1002) XXXX,NCODE,I,IDUM
      IF (NCODE.EQ.MDAT .OR. NCODE.EQ.LEND) GO TO 92
      WRITE (6,1003) MDAT
      WRITE (6,1004) XXXX
      IERR = IERR + 1
      GO TO 2000
   92 WRITE (6,1004) XXXX
      IF (NCODE .EQ. LEND) GO TO 600
   93 GO TO (100,200,300,400,500),I
  100 CONTINUE
      IF (IDUM(1) .GT. 0) IJOB = IDUM(1)
      IPRINT = IDUM(2)
      KOUT   = IDUM(3)
      IEDIT  = IDUM(4)
      IF (IDUM(5) .GT. 0)  IOT = IDUM(5)
      GO TO 91
  200 CONTINUE
C
C       PROGRAM SETS PARAMETERS OF ARRAYS FOR 'LIER' ROUTINE
      NLIB   = IDUM(1)
      LNRCW  = IDUM(2)
      MAXINT = IDUM(3)
      MXNRCW = IDUM(4)
      LNRP1  = LNRCW + 1
      GO TO 91
  300 CONTINUE
C
C       PROGRAM SETS PARAMETERS OF ARRAYS FOR INDEX ROUTINE
      NNAME = IDUM(1)
      NNODE = IDUM(2)
      GO TO 91
  400 CONTINUE
      IF (IDUM(1) .GT. 0) NROW = IDUM(1)
      GO TO 91
  500 CONTINUE
  600 CONTINUE
C
C       ALL PARAMETERS HAVE BEEN INPUT.
C       COMPUTE INDICES AND DIMENSIONS FOR ARRAYS.
C       INDICES HAVE ADDITIONAL INCREMENTS OF 1 TO ALLOW FOR
C       INSERTION OF TOP EVENT.
      I1 = 1
      I2 = I1 + NLIB + 1
      I3 = I2 + NLIB + 1
      I4 = I3 + NLIB + 1
```

```
      I5 = I4 + NLIB + 1
      I6 = I5 + NLIB + 1
      I7 = I6 + NLIB + 1
      J1 = 1
      J2 = J1 + NLIB + 1
      J3 = J2 + MAXINT*(NLIB + 1)
C
C       COMPUTE REMAINING ARRAY SPACE
C       AND ALLOCATE ENTIRELY TO ARRAY 'JROW'
      MXROW2 = (NSIZE - I7 +1)/LNROW
      IF (MXROW2 .LE. 0) GO TO 2200
      IF (J3 .GT. LSIZE) GO TO 2350
      WRITE (6,1007) MXNROW,MXROW2
      IF ((10*MXROW2) .LT. (9*MXNROW))  WRITE (6,1008)
      MXNROW = MXROW2
C
C       ALL INDICES FOR LIBR ROUTINE HAVE BEEN COMPUTED.
C       BEGIN EXECUTION OF LIBR ROUTINE.
C
      CALL LIBR(MAT(I1),MAT(I2),MAT(I3),MAT(I4),MAT(I5),MAT(I6),
     1          MAT(I7),NAME(J1),NAME(J2),NLIB,LNROW,MXNROW,MAXINT,
     2          MXINT2,MXROW2)
      JERR = IERR
      IF (IERR .LT. 0) JERR = -IERR - 1
C
C       LIBR HAS RETURNED: MXINT2 = MAX NO. INTERNAL MODES PER TYPE
C                          MXROW2 = TOTAL NUMBER OF LIBRARY ROWS.
C       NOTE: IERR LESS THAN 0 IMPLIES 'CCM' CARD HAS ALREADY BEEN
C                  READ BY SUBROUTINE 'LIBR' (RUN WILL CONTINUE).
C             IERR GREATER THAN 0 IMPLIES FATAL DATA ERROR DETECTED
C                  BY SUBROUTINE 'LIBR' (RUN WILL ABORT).
C       NOW SET IROW(NLIB + 1)
      MAT(I3 - 1) = MXROW2 + 1
      MXROW2 = MXROW2 + NROW
      I8  = I7  + LNROW*MXROW2
      I9  = I8  + NNODE
      I10 = I9  + NNODE
      I11 = I10 + NNCMP + 1
      I12 = I11 + LNRP1*(NNCMP + 1)
      IF (I12 .GT. (NSIZE + 1)) GO TO 2300
      IF ((J3 + NNCMP) .GT. LSIZE) GO TO 2350
  700 CONTINUE
      CALL INDEX  (MAT(I1),MAT(I3),MAT(I4),MAT(I5),NAME(J1),MAT(I8),
     1             MAT(I9),MAT(I10),MAT(I11),NAME(J3),NLIB,NNCMP,
     2             NNODE,LNRP1)
      JERR = JERR + IERR
      IERR = 0
      MNODE = IDUM(1)
      NODES = MNODE + MXINT2*NNCMP
      NLIB  = NLIB + 1
      NNCMP = NNCMP + 1
      I13   = I12 + 2*NODES
```

```
C
C       COMPUTE REMAINING ARRAY SPACE AND ALLOCATE TO GATES
      IF (NSIZE .LT. (I13 + 2*INGATE + 3)) GO TO 2400
      MGATE  = (NSIZE - I13 + 1)/(INGATE + 2)
      NGSIZE = NSIZE - I13 - 2*MGATE +1
      XX     = (1.0*INGATE - 5.0)/2.0
      I14    = I13 - 1
      I15    = NSIZE - I14
      WRITE (6,1010) I14,I15,MGATE,XX
      I14    = I13 + MGATE
      I15    = I14 + MGATE
      IF (JERR .NE. 0) GO TO 998
  800 IDUM(1) = IJOB
      CALL STEVE (MAT(I1),MAT(I2),MAT(I3),MAT(I4),MAT(I5),MAT(I6),
     1      MAT(I7),MAT(I8),MAT(I9),MAT(I10),MAT(I11),
     2      MAT(I12),NAME(J1),NAME(J2),NAME(J3),
     3     NLIB,LNROW,MXROW2,NNODE,MNODE,NNCMP,LNRP1,MXINT2,NODES,MAXINT)
C
C           SUBROUTINE 'STEVE' RETURNS IERR LESS THAN 0 FOR FATAL
C             ERRORS, AND IERR GREATER THAN 0 FOR LESS SEVERE ERRORS.
C             JOB WILL ABORT FOR IERR LESS THAN 0, BUT SUBSEQUENT JOBS
C             WILL BE RUN IF NO 'LIBR' OR 'INDEX' ERRORS WERE FOUND.
      IF (IERR .LT. 0) GO TO 810
      IERR = 0
      CALL DO IT(MAT(I1),MAT(I2),MAT(I3),MAT(I4),MAT(I5),MAT(I6),MAT(I7)
     1,           MAT(I8),MAT(I9),MAT(I10),MAT(I11),MAT(I12),MAT(I13),
     2           MAT(I14),MAT(I15),NAME(J3),NLIB,LNROW,MXROW2,NNODE,
     3           MNODE,NNCMP,LNRP1,MGATE,MXINT2,NGSIZE,NODES,NGATE)
      IF (IERR .NE. 0  .OR.  KOUT .EQ. 0) GO TO 810
C
C       SUBROUTINE 'DO IT' RETURNS IERR = 1 IF NO FAULT TREE
C         OR ONLY PARTIAL FAULT TREE WAS PRODUCED.
C
C       SUBROUTINE 'OUTPUT' WILL WRITE PREP INPUT ONTO
C         DATASET 'IOT' IF IERR = 0 AND KOUT = 1.
C
      CALL OUTPUT (MAT(I10),MAT(I13),MAT(I14),MAT(I15),NAME(J2),
     1             NAME(J3),NLIB,MNODE,NNCMP,MGATE,MXINT2,MAXINT,
     2             NGSIZE,NGATE,IOT)
      GO TO 998
  810 CONTINUE
C
C       FAULT TREE NOT PRODUCED BY SUBROUTINE 'DO IT'.
C       READ PAST PREP-KITT DATA (IF PRESENT).
C       XXXX CONTAINS LAST CARD READ BY SUBROUTINE 'STEVE'
C         ('&END', '&CUT', OR END-OF-FILE).
C
      IF (KOUT .EQ. 0  .OR.  XXXX(1) .NE. XOUT) GO TO 998
  820 READ (5,1002,END=999) XXXX,NCODE
      WRITE (6,1004) XXXX
      IF (NCODE .NE. MEND) GO TO 820
  998 IF (IPRINT .LT. 0  .OR.
```

```
     1      (IERR .EQ. 0  .AND.  IPRINT .LT. 3)) GO TO 990
       WRITE (6,9000) MAT
       WRITE (6,9001) NAME
  990  WRITE (6,9002)
       IF (JERR .NE. 0) GO TO 999
       IERR = 0
C
C         NOW READ INPUT FOR NEXT RUN
  899  IJOB = IJOB + 1
  900  READ (5,1002,END=999) XXXX,NCODE
       ISET = 1
       IF (NCODE .EQ. LDAT) ISET = 2
       IF (NCODE .EQ. LTOP) ISET = 3
       IF (NCODE .EQ. MEND .OR. NCODE .EQ. LEND) GO TO 901
       GO TO (910,920,920),ISET
  901  WRITE (6,1004) XXXX
       GO TO 900
  910  DO 911 I = 1,20
  911    TITLE(I) = XXXX(I)
  920  WRITE (6,1011) TITLE,IJOB
       IF (ISET .EQ. 3) GO TO 800
  921  READ (5,1002,END=999) XXXX,NCODE,I,IDUM
       WRITE (6,1004) XXXX
       IF (NCODE .EQ. LTOP) GO TO 800
       IF (NCODE .EQ. LDAT .OR. NCODE .EQ. LEND) GO TO 921
       IF (NCODE .EQ. MEND) GO TO 900
       IF (NCODE .EQ. MDAT) GO TO 930
       IF (ISET .EQ. 0) GO TO 921
       WRITE (6,1003) LDAT
       WRITE (6,1004) XXXX
       ISET = 0
       GO TO 921
  930  CONTINUE
C
C        READ NEW PARAMETERS FROM 'DAT' CARD.
       IPRINT = IDUM(2)
       KOUT   = IDUM(3)
       IEDIT  = IDUM(4)
       IF (IDUM(5) .GT. 0)  IOT = IDUM(5)
       ISET = 1
  931  READ (5,1002,END=940) XXXX,NCODE
       WRITE (6,1004) XXXX
       IF (NCODE .EQ. LTOP .OR. NCODE .EQ. LEND) GO TO 800
       IF (NCODE .EQ. MEND) GO TO 950
       IF (ISET .EQ. 0) GO TO 931
       WRITE (6,1003) LEND
       WRITE (6,1004) XXXX
       ISET = 0
       GO TO 931
  940  WRITE (6,1012)
       RETURN
  950  WRITE (6,1013)
```

```
      GO TO 899
  999 RETURN
C
C       ERROR REGION.
 2000 CONTINUE
      IF (NCODE .EQ. LLIB) GO TO 2100
      RETURN
 2100 IERR = -IERR
      GO TO 600
 2200 ISPACE = I7 + LNROW*MXNRCW
      WRITE (6,1005) ISPACE,NSIZE
      GO TO 2350
 2300 ISPACE = I12 - 1
      WRITE (6,1005) ISPACE,NSIZE
 2350 J3 = J3 + NNCME
      IF (J3 .GT. LSIZE) WRITE(6,1006) J3,LSIZE
      JERR = JERR + 1
      GO TO 998
 2400 ISPACE = I13 - 1
      WRITE (6,1009) NSIZE,ISPACE
      JERR = JERR + 1
      GO TO 998
C
C       FORMAT REGION
 1000 FORMAT (20A4)
 1001 FORMAT(1H1,65('--')/,' - PROGRAM CAT,  VERSION OF 10/75',T131,'-'/
     1,       ' - PROGRAM FOR THE AUTOMATED CONSTRUCTION OF FAULT TREES.'
     2,        T131,'-' /,' -',T131,'-'/ ,' - ',20A4,T131,'-' /,1X,
     3         65('--')///,1X,'*** DATA VALIDATION SECTION ***'//)
 1002 FORMAT (20A4,T1,A4,I1,12I5)
 1003 FORMAT (1H0,'*** INPUT ERROR ***'/,1X,A4,' CARD MISSING OR MISPUNC
     1HED'/,7X,'CARD IN ERROR IS:'/)
 1004 FORMAT (1X,20A4)
 1005 FORMAT (1H0,'*** ERROR ***'/5X,'ARRAY SPACE REQUIRED EXCEEDS SPACE
     1 ALLOCATED.'/5X,'ARRAY MAT REQUIRES ESTIMATED',I6,' WORDS FOR ROUT
     2INE LIBR, PLUS ADDITIONAL SPACE FOR REST OF JOB.'/5X,'ONLY',I6,
     3 ' WORDS ALLOCATED.'/' *** JOB TERMINATING ***')
 1006 FORMAT (1H0,'*** ERROR ***'/5X,'ARRAY SPACE REQUIRED EXCEEDS SPACE
     1 ALLOCATED.'/5X,'ARRAY ''NAME'' REQUIRES',I6,' DOUBLE PRECISION WO
     2RDS.'/5X,'ONLY',I6,' WORDS ALLOCATED.'/
     3 ' *** JOB TERMINATING ***')
 1007 FORMAT (1H0,'NUMBER OF LIBRARY ROWS ESTIMATED (MXNROW) =',I5,
     1  ' ROWS.'/' ALLOCATED SPACE REMAINING        (MXROW2) =',I5,
     2  ' ROWS.')
 1008 FORMAT (1H0,'*** WARNING ***'/5X,'REMAINING SPACE MAY BE INSUFFICI
     1ENT FOR JOB.  JOB CONTINUING.')
 1009 FORMAT (1H0,'*** ERROR ***'/5X,'INSUFFICIENT ARRAY SPACE REMAINING
     1 FOR GATE STORAGE.'/5X,'ARRAY SPACE ALLOCATED =',I6,' WORDS.'/
     25X,'ARRAY SPACE REQUIRED  =',I6,' PLUS 13 WORDS FOR EACH GATE GENE
     3RATED.'/' *** JOB TERMINATING ***')
 1010 FORMAT (////,1X,129('*')/' * STORAGE SPACE SUMMARY',T130,'*'/
     1       ' *   ARRAY ''MAT'' HAS USED',I6,' WORDS.',T130,'*'/
```

```
     2       ' *   REMAINING',I6,' WORDS WILL ACCOMMODATE',I6,' GATES WI
     3TH AN AVERAGE OF',F4.1,' INPUTS PER GATE.',T130,'*'/1X,129('*')/)
 1011 FORMAT(1H1,65('--')/,' - PROGRAM CAT,  VERSION OF 10/75',T131,'-'/
     1,       ' - PROGRAM FOR THE AUTOMATED CONSTRUCTION OF FAULT TREES.'
     2,        T131,'-' /,' -',T131,'-'/ ,' - ',20A4,T131,'-' /,1X,
     3         65('--')///,1X,'*** JOB',I5,' ***'//)
 1012 FORMAT ('0*** END OF FILE REACHED WITH NO TOP EVENT DEFINED ***'/
     1        ' *** JOB TERMINATING ***'/1H1)
 1013 FORMAT ('0*** INPUT ERROR ***'/5X,'''&END'' CARD REACHED WITH NO T
     1OP EVENT DEFINED.'/5X,'DATA FOR NEXT JOB WILL BE READ.'/1H1)
C
C
 9000 FORMAT (1H1,'ARRAY MAT:'//20(1X,I5))
 9001 FORMAT (1H1,'ARRAY NAME:'//(1X,10(A8,2X)))
 9002 FORMAT (1H1)
      END
      SUBROUTINE LIBR (NTYPE,IROW,NINT,NIN,NOUT,NROW,JROW,NAME,MODNAM,
     1                 NLIB,LNROW,MXNROW,MAXINT,MXINT2,MXROW2)
C
C       SUBROUTINE LIBR
C       SUBROUTINE TO READ LIBRARY AND VALIDATE ENTRIES
C         NTYPE IS TYPE NUMBER OF COMPONENT TYPE I
C         IROW POINTS TO BEGINNING ROW OF COMPONENT TYPE I
C
      DIMENSION NTYPE(NLIB),IROW(NLIB),NINT(NLIB),NIN(NLIB),NOUT(NLIB),
     1          NROW(NLIB),JROW(LNROW,MXNROW)
      REAL*8 NAME(NLIB),MODNAM(MAXINT,NLIB),NNAME
      COMMON TITLE(20),XXXX(20),IERR,IEDIT,IDUM(12),NNAME,IPRINT,KOUT
      DATA LIB,LROW,LMOD,LEND,LLIB,LCMP,LCOM/'LIBR',' ROW',
     1    ' MOD','END ','&LIB','&CMP',' COM'/
C
C       FIRST SUPPRESS ERROR MESSAGES FOR IHC215I (ILLEGAL CHAR.)
      CALL ERRSET(215,0,-1,0,1)
      IROW(1) = 1
      ISET = 0
      JSTOP = 1
      MXINT2 = 1
C      MXINT2 WILL BE SET TO MAX NO. OF INTERNAL MODES USED
      WRITE(6,1000) TITLE
      IF (IERR .LT. 0) GO TO 94
C
C       READ FIRST CARD, WHICH SHOULD BE '&LIB' CARD.
C       IF '&LIB' CARD IS MISSING, CORRECTIVE ACTION WILL BE ATTEMPTED.
      READ (5,1001) XXXX,NCODE,NAME(1),NTYPE(1),NIN(1),NINT(1),
     1              NOUT(1),NROW(1)
      IF (NCODE .EQ. LLIB) GO TO 95
      WRITE (6,1003) LLIB
      WRITE (6,1002) XXXX
      IERR = IERR + 1
      IF (NCODE .NE. LIB) GO TO 96
      IERR = IERR - 1
      ISET = 1
```

```
      WRITE (6,1010) LIB
C
C       IF '&LIB' CARD IS MISSING AND 'LIBR' CARD HAS BEEN READ,
C       SKIP TO DATA VALIDATION SECTION WITHOUT READING NEXT CARD.
      GO TO 96
C
C       '&LIB' CARD HAS ALREADY BEEN READ BY DRIVER PROGRAM,
C       AND IERR HAS BEEN SET NEGATIVE.
C       ROUTINE RESETS IERR AND CONTINUES.
   94 IERR = -IERR - 1
   95 WRITE (6,1002) XXXX
   96 CONTINUE
      DO 200 I=1,NLIB
        IF (ISET .EQ. 1) GO TO 97
        READ(5,1001) XXXX,NCODE,NAME(I),NTYPE(I),NIN(I),NINT(I),
     1    NOUT(I),NROW(I)
   97   ISET = 0
        IF (NCODE .EQ. LIB) GO TO 100
        IF (NCODE .EQ. LOMP)  GO TO 3998
   98   WRITE (6,1003) LIB
C
C         ERROR FOUND
C         ROUTINE LOOPS TO FIND NEXT VALID COMPONENT HEADER CARD.
C         IF NONE FOUND, ROUTINE TERMINATES.
        IERR = IERR + 1
   99   WRITE (6,1002) XXXX
        IF (NCODE .EQ. LEND) GO TO 130
        ISET = 2
        GO TO 129
C
  100   WRITE (6,1015) I,XXXX
C
C         VALID 'LIBR' CARD FOUND FOR COMPONENT I.
C         ROUTINE NOW VALIDATES INPUT DATA FOR COMPONENT I.
        NROW = NROW(I)
        IF(I .GT. 1) IROW(I) = IROW(I-1) + NROW(I-1)
        IF (IROW(I) .LT. I) IROW(I) = I
        IF (IROW(I) + NROW .LE. MXNROW + 1) GO TO 101
        WRITE(6,1012) NTYPE(I)
C
C         IF ALLOCATED NUMBER OF ROWS IN ARRAY JROW IS EXCEEDED,
C         SET ERROR FLAG AND OVERLAP EXTRA ROWS ONTO ROW IROW = 1.
        IROW(I) = 1
        IERR = IERR + 1
  101   NNINT = NINT(I)
        MXINT2 = MAXO(MXINT2,NNINT)
        NTOT = NNINT + NIN(I) + NOUT(I)
        IF (NNINT .LE. MAXINT) GO TO 102
        WRITE (6,1013) NTYPE(I),NNINT,MAXINT
        IERR = IERR + 1
  102   IF (NTOT .LE. LNROW) GO TO 103
        WRITE (6,1014) NTYPE(I),NTOT,LNROW
```

```
        IERR = IERR + 1
  103   CONTINUE
C
C     NOW READ IN NAMES OF INTERNAL FAILURE MODES.
        READ(5,1004) XXXX,NCODE,(MODNAM(J,I),J=1,MAXINT)
        IF (NCODE .EQ. LMOD) GO TO 110
  109   WRITE(6,1003) LMOD
C
C         ERROR FOUND ON 'MOD' CARD.
        WRITE (6,1002) XXXX
        IERR = IERR + 1
        IF (NCODE .EQ. LEND) GO TO 130
        IF (NCODE .EQ. LCMP) GO TO 3999
        ISET = 2
        GO TO 129
  110   WRITE(6,1002) XXXX
C
C     NOW READ IN COMPONENT TABLE ROWS.
C     JSTART AND JSTOP MARK BEGINNING AND END OF COMPONENT I IN ARRAYS.
  111   JSTART = IROW(I)
        JSTOP = JSTART + NROW - 1
        DO 120 J=JSTART,JSTOP
          READ(5,1001) XXXX,NCODE,NNAME,(JROW(K,J),K=1,LNROW)
          IF (NCODE .EQ. LROW) GO TO 119
  118     WRITE (6,1003) LROW
C
C           ERROR FOUND ON 'ROW' CARD.
          WRITE (6,1002) XXXX
          IERR = IERR + 1
          IF (NCODE .EQ. LEND) GO TO 130
          IF (NCODE .EQ. LCMP) GO TO 3999
          IF (IERR .GE. 100) GO TO 2000
          ISET = 2
          GO TO 129
  119     WRITE (6,1002) XXXX
  120     CONTINUE
C
C           CHECK FOR END CARD AFTER EACH COMPONENT.
C           ROUTINE WILL TAKE CORRECTIVE ACTION IF END CARD IS OMITTED.
  129   IF (I .EQ. NLIB) GO TO 121
        READ (5,1001)          XXXX,NCODE,NAME(I+1),NTYPE(I+1),NIN(I+1),
     1                        NINT(I+1),NOUT(I+1),NROW(I+1)
        IF (NCODE .NE. LIB) GO TO 122
        IF (ISET .NE. 2) WRITE (6,1003) LEND
        WRITE (6,1002) XXXX
        ISET = 1
        GO TO 200
  121   CONTINUE
        READ (5,1006) NCODE,XXXX
  122   IF (NCODE .EQ. LEND) GO TO 124
  123   IF (ISET .EQ. 2) GO TO 124
        WRITE (6,1003) LEND
```

```
          ISET = 2
          IERR = IERR + 1
  124     WRITE (6,1002) XXXX
          IF (NCODE.EQ.LCMP .AND. I.LT.NLIB) GO TO 3998
          IF (NCODE .EQ. LCMP) GO TO 3999
  130     IF (NCODE.NE.LIB .AND. NCODE.NE.LEND) GO TO 129
          WRITE (6,1007)
  200     CONTINUE
C
C       LIBRARY HAS BEEN READ IN AND VALIDATED.
C         FINAL ERROR MESSAGES FOLLOW.
C         FIRST ROUTINE SETS MAX NUMBER OF INTERNAL MODES,
C         AND NUMBER OF ROWS ACTUALLY USED.
C         THEN ROUTINE CHECKS TO MAKE SURE NO EXTRA CARDS REMAIN
        MXINT2 = MINO(MXINT2,MAXINT)
        READ (5,1001,END=9999) XXXX,NCODE,NNAME,IDUM
        IF (NCODE.NE.LCMP .AND. NCODE.NE.LCOM) WRITE (6,1016) NLIB
  201   IF (NCODE .EQ. LCMP) GO TO 210
        IF (NCODE .NE. LCOM) GO TO 202
        WRITE (6,1010) LCOM
        IF (IERR .NE. 0) WRITE (6,1008) IERR
        IERR = -IERR - 1
        MXROW2 = JSTOP
        RETURN
  202   WRITE (6,1002) XXXX
        READ (5,1001,END=9999) XXXX,NCODE,NNAME,IDUM
        GO TO 201
  210   WRITE (6,1010) LCMP
        IF (IERR .NE. 0) WRITE (6,1008) IERR
        MXROW2 = JSTOP
        RETURN
 2000   WRITE (6,1009) IERR
        GO TO 202
 3998   I = I - 1
        WRITE (6,1005) I,NLIB
 3999   WRITE (6,1008) IERR
        WRITE (6,1010) LCMP
        MXROW2 = JSTOP
        RETURN
 9999   WRITE (6,1011)
        IERR = 2
        RETURN
C
C         FORMAT REGION.
 1000   FORMAT (1H1,'*** ',20A4,' ***'//,5X,'LIBRARY INPUT PRINTOUT'//)
 1001   FORMAT (20A4,T1,A4,6X,A8,2X,12I5)
 1002   FORMAT (1X,20A4)
 1003   FORMAT (1H0,'*** INPUT ERROR ***'/,1X,A4,' CARD MISSING OR MISPUNC
     1HED'/,7X,'CARD IN ERROR IS:'/)
 1004   FORMAT (20A4,T1,A4,6X,7(A8,2X))
 1005   FORMAT (1H0,'*** WARNING ***'/,5X,'ONLY',I4,' COMPONENT TYPES INPU
     1T.',I5,' COMPONENT TYPES EXPECTED')
```

```
 1006 FORMAT (A4,T1,20A4)
 1007 FORMAT (//)
 1008 FORMAT (1H0,'LIBRARY ROUTINE TERMINATING. NUMBER OF INPUT ERRORS =
     1',I4,/,1X,'VALIDATION OF REMAINING DATA WILL BE ATTEMPTED.'/)
 1009 FORMAT (1H0,'LIBRARY ROUTINE TERMINATING ABNORMALLY DUE TO',I4,
     1       ' OR MORE INPUT ERRORS',/,1X,'VALIDATION OF REMAINING INPUT
     2 WILL BE ATTEMPTED.'/)
 1010 FORMAT (1H0,A4,' CARD FOUND.  DATA VALIDATION CONTINUING.'//)
 1011 FORMAT (1H0,'*** END OF INPUT DATA STREAM ***'/,1X,'NO VALID HEADE
     1R CARD FOUND'/,1X,'JOB TERMINATING'//,1X,'*** END OF PROGRAM ***')
 1012 FORMAT (1H0,'*** INPUT ERROR ***'/,5X,'NUMBER OF COMPONENT ROWS EX
     1CEEDS NUMBER ALLOCATED.'/,5X,'ROW INDEX BEING RESET TO 1 FOR COMPO
     2NENT',I6//)
 1013 FORMAT (1H0,'*** INPUT ERROR ***'/,5X,'COMPONENT',I6,' HAS',I2,
     1 ' INTERNAL MODES.'/,5X,'THIS EXCEEDS THE',I2,' MODES ALLOWED.'/)
 1014 FORMAT (1H0,'*** INPUT ERROR ***'/,5X,'COMPONENT',I6,' HAS',I3,
     1 ' INTERNAL MODES + INPUTS + OUTPUTS.'/,5X,'THIS EXCEEDS THE',I3,
     2 ' ALLOWED.'/)
 1015 FORMAT (1H0,//,' LIBRARY DATA PRINTOUT FOR COMPONENT',I4/
     1,       11X,'NAME',7X,'TYPE',1X,' NIN NINT NOUT NROW'/,1X,20A4)
 1016 FORMAT (1H0,'*** WARNING ***'/,5X,'EXTRA COMPONENT TYPES INPUT.
     1ONLY',I5,' COMPONENT TYPES EXPECTED'/,5X,'EXTRA CARDS FOLLOW'/)
      END
      SUBROUTINE INDEX (NTYPE,NINT,NIN,NOUT,NAME,NCMP,MOUT,ITYPE,INODE,
     1                  CMPNAM,NLIB,NNCMP,NNODE,LNRP1)
C
C       SUBROUTINE TO READ AND INDEX COMPONENT FLOW CHART
C
      DIMENSION NTYPE(NLIB),NINT(NLIB),NIN(NLIB),NOUT(NLIB),NCMP(NNODE),
     1          MOUT(NNODE),ITYPE(NNCMP),INODE(LNRP1,NNCMP),
     2          IER6(12),IER7(12)
      DOUBLE PRECISION NNAME,CMPNAM(NNCMP),NAME(NLIB),MNAME(12),IERR
      COMMON TITLE(20),XXXX(20),IERR,IEDIT,IDUM(12),NNAME,IPRINT,KOUT
      DATA        LCOM,LEND,LTOP,LERR/
     1             ' COM','END ','&TOP','***ERROR'/
      DATA   IER6/12*0/
      DO 10 IZERO=1,NNODE
   10   NCMP(IZERO) = 0
      I = 0
      WRITE (6,1000) TITLE
C         READ FIRST CARD, WHICH SHOULD BE ' COM' CARD.
C         IF ' COM' CARD HAS ALREADY BEEN READ BY LIBR ROUTINE,
C         IDUM ARRAY CONTAINS COMPONENT 1, SO READ IS SKIPPED.
      IF (IERR .LT. 0) GO TO 98
      IERR = 0
   97 READ (5,1001) XXXX,NCODE,NNAME,IDUM
      IF (NCODE .EQ. LCOM) GO TO 99
      IF (NCODE .EQ. LEND) GO TO 1999
      WRITE (6,1002) LCOM
      WRITE (6,1014) XXXX
      IERR = IERR + 1
      GO TO 97
```

```
   98 IERR = 0
   99 WRITE (6,1003)
  100 I = I + 1
      ISET = 0
      WRITE (6,1004) I,XXXX
      CMPNAM(I) = NNAME
      INCDE(LNRP1,I) = 0
C
C       COMPONENT DATA IS NOW VALIDATED AND STORED IN ARRAYS
      JDUM = IDUM(1)
      DO 110 J = 1,NLIB
        IF (JDUM .NE. NTYPE(J)) GO TO 110
        ITYPE(I) = J
        GO TO 120
  110   CONTINUE
C         ERROR ENCOUNTERED: NO COMPONENT TYPE 'I' IN LIBRARY
      WRITE (6,1005) JDUM,NNAME
      IERR = IERR + 1
      ITYPE(I) = JDUM
      INCDE(LNRP1,I) = -1
      GO TO 200
  120 IN = NIN(J)
      ICUT = NOUT(J)
      ITOT = IN + ICUT
      ITOTP1 = ITOT + 1
C       SIGN, NUMBER AND VALIDITY OF NODES NOW CHECKED.
C       FIRST, CHECK FOR CORRECT NUMBER AND SIGN OF INPUTS/OUTPUTS.
      IER1 = 0
      IER2 = 0
      IER3 = 0
      IER4 = 0
      IER5 = 0
      IER8 = 0
      IER9 = 0
      DO 130 IT = 2,12
        JDUM = IDUM(IT)
        IF (IT. LE. ITOTP1) GO TO 129
        IF (JDUM .EQ. 0) GO TO 131
C         ERROR: COMPONENT HAS TOO MANY NODES
        IER2 = 1
        IF (JDUM .GT. NNODE) IER9 = 1
        IF (JDUM .LT. 0) IER1 = 1
        GO TO 131
  129   IF (JDUM .LE. 0) IER1 = 1
        IF (JDUM .GT. NNODE) IER9 = 1
  130   CONTINUE
  131 CONTINUE
      IF (I .EQ. 1) GO TO 133
C
C       NOW CHECK FOR UNIQUENESS OF COMPONENT NAME
      IM1 = I - 1
      DO 132 IS=1,IM1
```

```
        IF (NNAME .NE. CMPNAM(IS)) GO TO 132
        IER3 = IS
        GO TO 133
  132 CONTINUE
  133 IF (IER1 + IER2 + IER 9 .NE. 0) GO TO 140
C
C       NOW CHECK THAT ALL OUTPUTS HAVE UNIQUE NODE NUMBERS
C       IST IS INDEX OF FIRST OUTPUT NODE
      IST = IN + 2
      DO 137 IS=IST,ITOTP1
        NTEST = IDUM(IS)
        IF (NCMP(NTEST) .EQ. 0) GO TO 134
C         ERROR: OUTPUT NODE HAS BEEN PREVIOUSLY DEFINED
        IER6(IS) = NTEST
        IER7(IS) = NCMP(NTEST)
        IER8 = 1
        GO TO 135
C
C         NOW SET COMPONENT INDEX AND OUTPUT NUMBER
C         CORRESPONDING TO NODE NUMBER
  134   NCMP(NTEST) = I
        NOUT(NTEST) = IS - (IN + 1)
  135   IF (IS .LE. 2) GO TO 137
C         NOW CHECK THAT EACH OUTPUT IS DIFFERENT
C         FROM ALL OTHER INPUTS/OUTPUTS
        ISM1 = IS - 1
        DO 136 IT = 2,ISM1
          IF (IDUM(IT) .NE. NTEST) GO TO 136
          IF (IT .LT. IST) IER4 = 1
          IF (IT .GE. IST) IER5 = 1
  136     CONTINUE
  137   CONTINUE
  140 IERTOT = IER1 + IER2 + IER3 + IER4 + IER5 + IER8 + IER9
      IF (IERTOT .EQ. 0) GO TO 150
      IERR = IERR + 1
C
C       INPUT CARD 'I' HAS ONE OR MORE ERRORS. WRITE MESSAGE(S).
      WRITE (6,1006) I
      INODE(INRP1,I) = -2
      IF (IER3 .NE. 0) WRITE(6,1007) NNAME,IER3
      IF (IER9 .NE. 0) WRITE(6,1016) NNODE
      IF (IER1 .NE. 0) WRITE(6,1008) IDUM(1),ITOT
      IF (IER2 .NE. 0) WRITE(6,1009) IDUM(1),ITOT
      IF (IER4 .NE. 0) WRITE(6,1010)
      IF (IER5 .NE. 0) WRITE(6,1011)
      IF (IER8 .EQ. 0) GO TO 150
      DO 141 IS = 2,12
        IF (IER6(IS) .EQ. 0) GO TO 141
        WRITE (6,1012) IER6(IS),IER7(IS)
        IER6(IS) = 0
  141   CONTINUE
      GO TO 160
```

```
  150 CONTINUE
C       CARD NOW READ AND VALIDATED.  SET NODE NUMBERS.
      IF (IN .EQ. 0) GO TO 152
      DO 151 IS = 1,IN
  151   INODE(IS,I) = IDUM(IS+1)
  152 IS2 = NINT(J)
      IN1 = IN + 1
      DO 153 IS = IN1,ITOT
  153   INODE(IS+IS2,I) = IDUM(IS+1)
  160 CONTINUE
      IF (IERTOT .NE. 0) WRITE (6,1029)
  200 READ (5,1001,END=9999) XXXX,NCODE,NNAME,IDUM
      IF (NCODE .NE. LCOM) GO TO 202
      IF (ISET .EQ. 1) WRITE (6,1029)
      IF ( I .LT. NNCMP) GO TO 100
C
C       ERROR ENCOUNTERED: TOO MANY COMPONENTS
      WRITE (6,1013) NNCMP
      IERR = IERR + 1
  201 WRITE (6,1014) XXXX
      READ (5,1001,END=9999) XXXX,NCODE,NNAME,IDUM
      IF (NCODE.NE.LEND .AND. NCODE.NE.LTOP) GO TO 201
      WRITE (6,1015) NCODE
      GO TO 300
  202 IF (NCODE.EQ. LEND .OR. NCODE.EQ.LTOP) GO TO 203
C
C       ERROR ENCOUNTERED: UNKNOWN CARD TYPE
      LCODE = LEND
      IF (I .LT. NNCMP) LCODE = LCOM
      IF (ISET .EQ. 0) WRITE (6,1002) ICODE
      WRITE (6,1014) XXXX
      IF (ISET .EQ. 0) IERR = IERR + 1
      ISET = 1
      GO TO 200
  203 IF (I .EQ. NNCMP) GO TO 204
C
C       ERROR ENCOUNTERED: TOO FEW COMPONENTS
      WRITE (6,1017) I,NNCMP
      IERR = IERR + 1
      ISET = I
      GO TO 300
  204 WRITE (6,1014) XXXX
      GO TO 300
C
C       ALL COMPONENTS HAVE BEEN INPUT AND VALIDATED
C       INPUT EDIT FOLLOWS
  300 CONTINUE
      IEND = NNCMP
      IF (ISET .GT. 1) IEND = ISET
      ISET = 0
      WRITE (6,1019) TITLE
      DO 400 I = 1,IEND
```

```
        IF (INODE(LNEE1,I) .EQ. -1) GO TO 330
        JTYPE = ITYPE(I)
        IF (INODE(LNEE1,I) .NE. 0) GO TO 340
        IN = NIN(JTYPE)
        IF (IN .EQ. 0) GO TO 320
        DO 310 J=1,IN
C
C           FIND COMPONENT NUMBERS/NAMES FOR EACH INPUT NODE
          JDUM     = INODE(J,I)
          MCMP     = NCMP(JDUM)
          IDUM(J) = JDUM
          IF (MCMP .LE. 0) GO TO 309
          MNAME(J) = CMPNAM(MCMP)
          GO TO 310
  309     MNAME(J) = IERR
          IERR = IERR + 1
          ISET = ISET + 1
  310     CONTINUE
C
C     INDEX EDIT COMPLETE FOR COMPONENT I: WRITE CORRECT OUTPUT FORMAT
        WRITE (6,1020) I,CMPNAM(I),NTYPE(JTYPE),NAME(JTYPE),
     1                 (IDUM(J),MNAME(J),J=1,IN)
        GO TO 400
  320   WRITE (6,1021) I,CMPNAM(I),NTYPE(JTYPE),NAME(JTYPE)
        GO TO 400
  330   WRITE (6,1022) I,CMPNAM(I),ITYPE(I)
        GO TO 400
  340   WRITE (6,1023) I,CMPNAM(I),NTYPE(JTYPE),NAME(JTYPE)
  400   CONTINUE
        IF (ISET .NE. 0) WRITE (6,1028) ISET
C
C         NOW DO OUTPUT NODE EDIT
      WRITE (6,1024) TITLE
      DO 500 I=1,NNODE
        IF (NCMP(I) .EQ. 0) GO TO 500
        JCMP = NCMP(I)
        MNODE = I
        WRITE (6,1025) I,JCMP,CMPNAM(JCMP),MOUT(I)
  500   CONTINUE
        IDUM(1) = MNODE
C
C       EDIT CONCLUDED.  WRITE FINAL OUTPUT
  600 IF (IERR .NE. 0) WRITE (6,1018) IERR
      RETURN
 1999 WRITE (6,1026)
      IERR = 1
      WRITE (6,1018) IERR
      RETURN
 9999 IERR = 9999
      WRITE (6,1027)
      RETURN
C
```

```
C       FORMAT REGION
 1000 FORMAT (1H1,20A4//,1X,'COMPONENT INDEX INPUT PRINTOUT'//)
 1001 FORMAT (20A4,I1,A4,6X,A8,2X,12I5)
 1002 FORMAT (1H0,'*** INPUT ERROR ***'/,7X,A4,' CARD MISSING OR MISPUNC
     1HED.  CARD IN ERROR IS:')
 1003 FORMAT (//,1H0,'COMPONENT CARD PRINTOUT',//,1X,'INDEX',2X,'CODE',5X
     1,      'NAME',7X,'TYPE',2X,'INPUT/OUTPUT NODES'/)
 1004 FORMAT (1X,I5,1X,20A4)
 1005 FORMAT (1H ,'*** INPUT ERROR ***'/,5X,'NO COMPONENT TYPE',I6,
     1       ' FOUND IN LIBRARY.',2X,A8,' IN ERROR, OR LIBRARY INCOMPLET
     2E.'/)
 1006 FORMAT (1H ,'*** INPUT ERROR ***'/,5X,'COMPONENT',I4,' HAS ERRORS
     1IN INPUT AS FOLLOWS:')
 1007 FORMAT (7X,'COMPONENT NAME "',A8,'" HAS PREVIOUSLY BEEN USED BY CO
     1MPONENT',I4)
 1008 FORMAT (7X,'TOO FEW NODES, OR NON-POSITIVE NODES. COMPONENT TYPE',
     1       I6,' REQUIRES',I3,' POSITIVE NODES')
 1009 FORMAT (7X,'TOO MANY NODES. COMPONENT TYPE',I6,' REQUIRES',
     1       I3,' NODES')
 1010 FORMAT(7X,'ONE OR MORE OUTPUTS IDENTICAL WITH ONE OR MORE INPUTS')
 1011 FORMAT (7X,'OUTPUT NODES NOT UNIQUE')
 1012 FORMAT (7X,'OUTPUT NODE',I5,' HAS ALREADY BEEN ASSIGNED TO COMPONE
     1NT',I5)
 1013 FORMAT (1H0,'*** INPUT ERROR ***'/,5X,'MORE THAN THE',I5,' COMPONE
     1NTS SPECIFIED HAVE BEEN INPUT. EXTRA CARDS FOLLOW:')
 1014 FORMAT (7X,20A4)
 1015 FORMAT (1H0,A4,' CARD FOUND. VALIDATION AND FINAL EDIT FOLLOW.')
 1016 FORMAT (7X,'NODE TOO LARGE. MAXIMUM NODE ALLOWED =',I5)
 1017 FORMAT (1H0,'*** INPUT ERROR ***'/,5X,'ONLY',I4,' COMPONENTS INPUT
     1.',I5,' COMPONENTS EXPECTED.')
 1018 FORMAT (1H0,'INDEX ROUTINE TERMINATING. NUMBER OF INPUT ERRORS =',
     1  I4/,1X,'VALIDATION OF REMAINING DATA WILL BE ATTEMPTED.'/)
 1019 FORMAT (1H1,'COMPONENT INDEXING PRINTOUT FOR:'/,1X,20A4//,1X,
     1       'INDEX',2X,'NAME',6X,'TYPE#',2X,'TYPE',6X,'INPUTS FROM: ',
     2       'NODE/NAME'/)
 1020 FORMAT (1X,I5,2X,A8,2X,I5,2X,A8,5X,(5(I5,'/',A8,2X)/))
 1021 FORMAT (1X,I5,2X,A8,2X,I5,2X,A8,5X,'THIS COMPONENT HAS NO INPUTS')
 1022 FORMAT (1X,I5,2X,A8,2X,I5,2X,'****',10X,'*** ERROR: NO SUCH COMPONE
     1NT TYPE FOUND IN LIBRARY')
 1023 FORMAT (1X,I5,2X,A8,2X,I5,2X,A8,5X,'*** ERROR: INPUT HAS ERRORS SP
     1ECIFIED PREVIOUSLY')
 1024 FORMAT (1H1,'OUTPUT NODE CROSS-INDEX FOR:'/,1X,20A4//,1X,
     1       'OUTPUT NODE',2X,'COMPONENT:',1X,'INDEX',2X,'NAME',3X,
     2        'OUTPUT NO.'/)
 1025 FORMAT (1X,I7,16X,I5,2X,A8,I5)
 1026 FORMAT (1H0,'*** INPUT ERROR ***'/,1X,'END CARD FOUND WHERE COMPON
     1ENT DATA EXPECTED')
 1027 FORMAT (1H0,'*** END OF INPUT DATA STREAM ***'/,1X,'NO VALID HEADE
     1R CARD FOUND'/,1X,'JOB TERMINATING'//,1X,'*** END OF PROGRAM ***')
 1028 FORMAT (///' *** ERROR:',I5,' INPUT NODES REFERENCE UNDEFINED OUTP
     1UT NODES.'/,1X,'PRINTOUT SPECIFIES NODE NO./***ERROR'/)
 1029 FORMAT (1X)
```

```
      END
      SUBROUTINE STEVE (NTYPE,IROW,NINT,NIN,NOUT,NROW,JROW,NCMP,MOUT,
     1                  ITYPE,INODE,X,NAME,MODNAM,CMPNAM,
     2 NIIB,LNROW,MXROW2,NNODE,MNODE,NNCMP,LNRP1,MXINT2,NODES,MAXINT)
C*********************************************************************
C                                                                     *
C       SUBROUTINE TO 'SET TOP EVENT VALUES, ETC.'                    *
C                     S   T   E     V       E                         *
C       SUBROUTINE TO SET UP INTERNAL NODE NUMBERS, SET ARRAY X,      *
C       AND SET TOP EVENT AND SYSTEM BOUNDARY VALUES.                 *
C                                                                     *
C*********************************************************************
      COMMON TITLE(20),XXXX(20),IERR,IEDIT,IDUM(12),NNAME,IPRINT,KOUT
      INTEGER X(2,NODES)
      DIMENSION NINT(NLIB),NIN(NLIB),NOUT(NLIB),JROW(LNROW,MXROW2),
     1          INODE(LNRP1,NNCMP),ITYPE(NNCMP),NTYPE(NLIB),IROW(NLIB),
     2          NROW(NLIB),NCMP(NNODE),MOUT(NNODE)
      DOUBLE PRECISION NNAME,NAME(NLIB),MODNAM(MAXINT,NLIB),
     1                 CMPNAM(NNCMP),NAME1
      DATA  LTOP,MTOP,NTOP,LEND,MEND,NEC,NNT,NEXT,NAME1,LOUT/'&TOP',
     1 'TTOP',' TOP','END','&END','&BC',' INT',' EXT','TOPEVENT','&OUT'/
      WRITE (6,1000) TITLE
      NAME(NLIB) = NAME1
C
C       FIRST SET UP NODE NUMBERS FOR INTERNAL NODES.
      NNCM1 = NNCMP - 1
      DO 102 I = 1,NNCM1
        IT   = ITYPE(I)
        INT  = NINT(IT)
        IF (INT .EQ. 0) GO TO 101
C
C         COMPONENT I HAS INTERNAL MODES WHICH MUST HAVE NODE NUMBERS.
        IN   = NIN(IT)
        NODE = MNODE + (I-1)*MXINT2
        IS   = IN + 1
        ITOT = IN + INT
        DO 100 J = IS,ITOT
  100     INODE(J,I) = NODE + J - IN
        WRITE (6,1001) I,CMPNAM(I),NTYPE(IT),NAME(IT),
     1         (INODE(IN+J,I),MODNAM(J,IT),J=1,INT)
        GO TO 102
  101   WRITE (6,1001) I,CMPNAM(I),NTYPE(IT),NAME(IT)
  102   CONTINUE
C
C         NOW INITIALIZE X ARRAY
      DO 201 I = 1,NODES
        X(1,I) = -1
  201   X(2,I) = -1
C*********************************************************************
C                                                                     *
C       SECTION TO READ AND VALIDATE TOP EVENT AND BOUNDARY CONDITIONS *
C       FIRST SET UP LIBRARY ENTRIES FOR TOP (I = NLIB), THEN VALIDATE *
```

```
C                                                                              *
C*******************************************************************************
      WRITE (6,1006) IDUM(1),TITLE
      ISET = 0
      NTYPE(NLIB)  = 0
      ITYPE(NNCME) = NLIE
      NIN1(NLIB)   = 0
      NOUT(NLIB)   = 0
C
C       READ FIRST CARD, WHICH SHOULD BE '&TOP' CARD.
C       IF CARD IS MISSING, SEARCH FOR 'TTOP' CARD.
  300 READ (5,1002,END=996) XXXX,NCODE,NNAME,IDUM
      IF (NCODE .EQ. LTOP .OR. NCODE .EQ. MTOP) GO TO 301
      IF (NCODE .EQ. MEND) GO TO 990
      IF (NCODE .EQ. LEND) NCODE = LTOP
      IF (NCODE .EQ. LEND) GO TO 1999
      WRITE (6,1003) LTOP
      WRITE (6,1004) XXXX
      IERR = IERR + 1
      GO TO 2000
  301 WRITE (6,1007) NCODE,TITLE
      ISET = 1
      IF (NCODE .EQ. MTOP) GO TO 302
      READ (5,1002,END=997) XXXX,NCODE,NNAME,IDUM
      IF (NCODE .EQ. MTOP) GO TO 302
      IF (NCODE .EQ. MEND) GO TO 991
      IF (NCODE .EQ. LEND) NCODE = MTOP
      IF (NCODE .EQ. LEND) GO TO 1999
      WRITE (6,1003) MTOP
      WRITE (6,1004) XXXX
      IERR = IERR + 1
      GO TO 2000
  302 CONTINUE
C
C       TOP EVENT CARD 'TTOP' HAS BEEN READ.
C       WRITE AND VALIDATE DATA.
      CMENAM(NNCME) = NNAME
      MROW        = IDUM(1)
      NIN(NLIB)   = IDUM(2)
      NROW(NLIB)  = MROW
      LROW        = MXROW2 - IROW(NLIB) + 1
      IF (MROW .LE. LROW) GO TO 303
      IERR = -1
      WRITE (6,1009) LROW,MROW,XXXX
  303 IF (NIN(NLIB) .LE. INROW) GO TO 304
      IERR = -1
      WRITE (6,1010) INROW,NIN(NLIB),XXXX
      NIN(NLIE) = LNROW
  304 CONTINUE
C
C       NOW CHECK THAT ALL NODES HAVE BEEN DEFINED AND ARE UNIQUE.
      IN = NIN(NLIE)
```

```
      DO 309 I = 1,IN
        JDUM = IDUM(I + 2)
        IF (JDUM .GT. MNODE  .OR.  JDUM .LE. 0) GO TO 307
        IF (NCMP(JDUM) .EQ. 0) GO TO 308
C
C         VALID OUTPUT NODE FOUND. NOW CHECK FOR UNIQUENESS.
        IF (I .EQ. 1) GO TO 309
        IM1 = I - 1
        DO 305 J = 1,IM1
          IF (JDUM .EQ. IDUM(J + 2)) GO TO 306
  305     CONTINUE
        GO TO 309
  306   WRITE (6,1005) XXXX
        WRITE (6,1011) JDUM
        IERR = -1
        GO TO 309
  307   WRITE (6,1005) XXXX
        WRITE (6,1012) JDUM,MNODE
        IERR = -1
        GO TO 309
  308   WRITE (6,1005) XXXX
        WRITE (6,1013) JDUM
        IERR = -1
  309   CONTINUE
      IF (IERR .LT. 0) GO TO 2999
C
C       'TTOP' CARD HAS BEEN VALIDATED.  ENTER INTO INODE AND WRITE.
      ISET = 2
      WRITE (6,1014) NNAME,MROW,IN
      DO 310 I = 1,IN
        JDUM = IDUM(I + 2)
        INODE(I,NNCMP) = JDUM
        ICOMP = NCMP(JDUM)
  310   WRITE (6,1015) JDUM,ICOMP,CMPNAM(ICOMP),MOUT(JDUM)
C
C         NOW INPUT TOP EVENT ROWS FROM ' TOP' CARDS.
      INP2 = IN + 2
      WRITE (6,1016) (IDUM(I),I=3,INP2)
      I = 0
  311   READ (5,1002,END=998) XXXX,NCODE,NNAME,IDUM
        IF (NCODE .EQ. NTOP) GO TO 313
        IF (NCODE .EQ. MEND) GO TO 992
        IF (NCODE .EQ. LEND) NCODE = NTOP
        IF (NCODE .EQ. LEND) GO TO 1999
        WRITE (6,1003) NTOP
        WRITE (6,1004) XXXX
        GO TO 2000
  313   I = I + 1
C
C         VALID ' TOP' CARD FOUND.
        WRITE (6,1005) XXXX
        IROW = IROW(NLIB) + I - 1
```

```
      DO 314 J = 1,IN
  314   JROW(J,IRCW) = IDUM(J)
      IF (I .LT. MRCW) GO TO 311
C
C       TOP EVENT ROWS HAVE BEEN READ.  SEARCH FOR END CARD
C       AND SET BOUNDARY CONDITIONS (IF ANY).
C       MAKE SURE NO EXTRA ' TOP' CARDS REMAIN.
      ISET = 3
      READ (5,1002,END=2201) XXXX,NCODE,NNAME,IDUM
      IF (NCODE .EQ. LEND) GO TO 316
      IF (NCODE .EQ. NBC ) GO TO 318
      IF (NCODE .EQ. MEND  .OR.  NCODE .EQ. LOUT)  GO TO 2201
      IF (NCODE .EQ. NNT  .OR.  NCODE .EQ. NEXT) GO TO 317
      IF (NCODE .NE. NTOP) GO TO 315
C
C       EXTRA ' TOP' CARDS FOUND.  ABORT JOB.
      IERR = -1
      WRITE (6,1018) MROW,XXXX
      GO TO 2000
  315 WRITE (6,1003) LEND
      WRITE (6,1004) XXXX
      IF (IERR .GE. 0) IERR = IERR + 1
      GO TO 2000
  316 WRITE (6,1005) LEND
      READ (5,1002,END=2203) XXXX,NCODE,NNAME,IDUM
      IF (NCODE .EQ. NBC)  GO TO 318
      IF (NCODE .EQ. LEND  .OR. NCODE .EQ. MEND) WRITE (6,1004) XXXX
      IF (NCODE .EQ. LEND) GO TO 2000
      IF (NCODE .EQ. MEND  .OR.  NCODE .EQ. LOUT)  GO TO 2203
      WRITE (6,1003) NBC
      WRITE (6,1004) XXXX
      IF (NCODE .EQ. NNT  .OR.  NCODE .EQ. NEXT) GO TO 317
      IF (IERR .GE. 0) IERR = IERR + 1
      GO TO 2000
  317 CONTINUE
C
C       '&EC' CARD OMITTED.
      WRITE (6,1003) NBC
      WRITE (6,1004) XXXX
      WRITE (6,1019) TITLE
      IF (NCODE .EQ. NNT) GO TO 320
      GO TO 330
  318 CONTINUE
C
C       '&EC' CARD FOUND.  SEARCH FOR VALID BOUNDARY CONDITIONS.
      WRITE (6,1019) TITLE
      ISET = 4
  319 READ (5,1002,END=2201) XXXX,NCODE,NNAME,IDUM
      IF (NCODE .EQ. LEND) GO TO 2300
      IF (NCODE .EQ. MEND  .OR.  NCODE .EQ. LOUT)  GO TO 2201
      IF (NCODE .EQ. NNT ) GO TO 320
      IF (NCODE .EQ. NEXT) GO TO 330
```

```
      WRITE (6,1003) NEXT
      WRITE (6,1004) XXXX
      IF (IERR .GT. 0) IERR = IERR + 1
      GO TO 2200
  320 ISET = 5
      WRITE (6,1005) XXXX
C
C       INTERNAL NODES TO BE SET.  FIRST FIND COMPONENT.
      DO 321 J = 1,NNCMP
        IF (CMPNAM(J) .EQ. NNAME) GO TO 322
  321   CONTINUE
C
C         NO NAME MATCH FROM ' INT' CARD.
      IERR = -1
  329 WRITE (6,1020) NNAME
      GO TO 319
  322 CONTINUE
C
C       VALID NAME FOUND.  SET NODES.
      INT = NINT(J)
      IF (INT .EQ. 0) GO TO 329
      NODE = MNODE + (J - 1)*MXINT2
      DO 323 K = 1,INT
        IF (IDUM(K) .LT. 0) GO TO 323
        KNODE = NODE + K
        X(1,KNODE) = IDUM(K)
        X(2,KNODE) = 0
  323   CONTINUE
      GO TO 319
  330 ISET = 5
      WRITE (6,1005) XXXX
C
C       EXTERNAL NODES TO BE SET.  CHECK EACH NODE FOR VALIDITY.
      DO 335 J = 1,6
        JJ   = 2*J - 1
        JDUM = IDUM(JJ)
        IF (JDUM .EQ. 0) GO TO 336
        IF (JDUM  .GT. MNODE  .OR.  JDUM .LE. 0) GO TO 333
        IF (NCMP(JDUM) .EQ. 0) GO TO 334
C
C         VALID NODE FOUND.  CHECK AGAINST TOP EVENT.
        IN = NIN(NLIB)
        DO 331 K = 1,IN
          IF (INODE(K,NNCMP) .EQ. JDUM) GO TO 332
  331     CONTINUE
C
C           NODE VALID.  SET MODE.
        X(1,JDUM) = IDUM(JJ + 1)
        X(2,JDUM) = 0
        GO TO 335
  332   CONTINUE
C
```

```
C          BOUNDARY CONDITION IS ALSO TOP EVENT.
C          NEGLECT BOUNDARY CONDITION.
           WRITE (6,1021) JDUM
           GO TO 335
  333      WRITE (6,1012) JDUM,MNODE
           IERR = -1
           GO TO 335
  334      WRITE (6,1013) JDUM
           IERR = -1
  335      CONTINUE
  336 CONTINUE
      GO TO 319
C
C          FINAL REGION.
C          ERRORS CHECKED, EXTRA CARDS READ, ETC.
  990 NCODE = LTOP
      GO TO 995
  991 NCODE = MTOP
      GO TO 995
  992 NCODE = NTOP
  995 WRITE (6,1025) NCODE
      IERR = -1
      RETURN
  996 NCODE = LTOP
      GO TO 999
  997 NCODE = MTOP
      GO TO 999
  998 NCODE = NTOP
  999 WRITE (6,1008) NCODE,XXXX
      IERR = -2
      RETURN
C
C          PROGRAM ENTERS THIS REGION WHEN ISET = 0,1,2 OR 3.
 1999 WRITE (6,1017) NCODE
      IF (IERR .GE. 0) IERR = IERR + 1
 2000 JSET = ISET + 1
      IF (NCODE .EQ. LOUT  .AND.  ISET .EQ. 3)  GO TO 2299
 2001 READ (5,1002,END=2298) XXXX,NCODE,NNAME,IDUM
      GO TO (2009,2019,2029,2039),JSET
 2009 IF (NCODE .EQ. LTOP) GO TO 301
      IF (NCODE .EQ. MTOP) GO TO 301
      GO TO 2002
 2019 IF (NCODE .EQ. MTOP) GO TO 302
      GO TO 2002
 2029 IF (NCODE .EQ. NTOP) GO TO 313
      GO TO 2002
 2039 IF (NCODE .EQ. NBC) GO TO 318
      IF (NCODE .EQ. NNT  .OR.  NCODE .EQ. NEXT) GO TO 317
      IF (NCODE .EQ. ICUT)  GO TO 2299
 2002 WRITE (6,1005) XXXX
      IF (NCODE .NE. MEND) GO TO 2001
C
```

```
C       '&END', '&OUT' OR END OF FILE READ.
 2297 GO TO (990,991,992,2299),JSET
 2298 GO TO (996,997,998,2299),JSET
 2299 WRITE (6,1026)
      WRITE (6,1022)
      IF (NCODE .EQ. LOUT) GO TO 2301
      IF (IERR .LT. 0) WRITE (6,1024)
      RETURN
C
C       PROGRAM ENTERS THIS REGION WHEN ISET = 3 OR ISET = 4.
 2200 READ (5,1002,END=2201) XXXX,NCODE,NNAME,IDUM
      IF (NCODE .EQ. NNT) GO TO 320
      IF (NCODE .EQ. NEXT) GO TO 330
      IF (NCODE .EQ. LEND) GO TO 2300
      IF (NCODE .NE. MEND  .AND.  NCODE .NE. LOUT)  GO TO 2202
C
C       '&END', '&OUT' OR END OF FILE READ (ISET = 3 OR 4)
 2201 IF (ISET .EQ. 4) WRITE (6,1022)
      WRITE (6,1023)
 2203 IF (ISET .EQ. 3) WRITE (6,1022)
      IF (NCODE .EQ. LOUT)  GO TO 2301
      IF (IERR .LT. 0) WRITE (6,1024)
      RETURN
 2202 WRITE (6,1005) XXXX
      IF (IERR .GE. 0) IERR = IERR + 1
      GO TO 2200
C
C       END CARD ENCOUNTERED, NO CARDS MISSING (TRANSFER FROM 319)
C       SEARCH FOR '&END',&OUT OR END OF FILE.
 2300 WRITE (6,1005) XXXX
      READ (5,1002,END=2301) XXXX,NCODE,NNAME,IDUM
      IF (NCODE .EQ. MEND  .OR.  NCODE .EQ. LOUT)  GO TO 2301
      IF (IERR .GE. 0) IERR = IERR + 1
      IF (NCODE .NE. NNT  .AND. NCODE .NE. NEXT) GO TO 2300
      WRITE (6,1017) NCODE
      IF (NCODE .EQ. NNT) GO TO 320
      GO TO 330
 2301 IF (ISET .EQ. 4) WRITE (6,1022)
      ISET = 6
      IF (IERR .LT. 0  .AND.  NCODE .EQ. LOUT)  GO TO 2300
      IF (KOUT .EQ. 1)  GO TO 2310
      IF (NCODE .EQ. LOUT)  GO TO 2302
      IF (IERR .LT. 0)  WRITE (6,1024)
      RETURN
 2302 WRITE (6,1027)  XXXX
      IERR = IERR + 1
 2303 READ (5,1002,END=2304) XXXX,NCODE
      WRITE (6,1005) XXXX
      IF (NCODE .NE. MEND) GO TO 2303
 2304 RETURN
 2310 IF (NCODE .NE. LOUT) GO TO 2311
      RETURN
```

```
 2311 WRITE (6,1028)
      IF (IERR .GE. 0) IERR = IERR + 1
      IF (IERR .LT. 0) WRITE (6,1024)
      RETURN
 2999 WRITE (6,1024)
 2901 READ (5,1002,END=2902) XXXX,NCODE
      WRITE (6,1005) XXXX
      IF (NCODE .NE. MEND) GO TO 2901
 2902 RETURN
C*********************************************************************
C                                                                     *
C       FORMAT REGION                                                 *
C                                                                     *
C*********************************************************************
 1000 FORMAT (1H1,'INTERNAL NODE INDEX FOR:'/1X,20A4//,1X,
     1    'INDEX',2X,'NAME',6X,'TYPE#',2X,'TYPE',6X,'INTERNAL NODES'/)
 1001 FORMAT (1X,I5,2X,A8,I7,2X,A8,5X,5(I5,': ',A8,2X)/
     1                              (38X,5(I5,': ',A8,2X)))
 1002 FORMAT (20A4,T1,A4,6X,A8,2X,12I5)
 1003 FORMAT ('0*** INPUT ERROR ***'/7X,A4,' CARD MISSING OR MISPUNCHED.
     1  CARD IN ERROR IS:')
 1004 FORMAT (7X,20A4)
 1005 FORMAT (1X,20A4)
 1006 FORMAT (1H1,65('**')/' * PROGRAM CAT,  VERSION OF 10/75',T131,'*'/
     1       ' * TOP EVENT AND BOUNDARY CONDITION PRINTOUT FOR JOB',I5,
     2       T131,'*'/
     3       ' *',T131,'*'/' * ',20A4,T131,'*'/1X,65('**')//)
 1007 FORMAT (1X,A4//' TOP EVENT FOR ',20A4/)
 1008 FORMAT ('0*** END OF INPUT DATA STREAM ***'/7X,A4,' CARD EXPECTED:
     1  DATA CARDS MISSING OR MISPUNCHED. LAST CARD READ WAS:'/1X,20A4/
     2  ' JOB TERMINATING'//' *** END OF PROGRAM ***'//)
 1009 FORMAT ('0*** ERROR ',60('**')/' * NUMBER OF ROWS OF TOP EVENT EXC
     1EEDS SPACE ALLOCATED.',T131,'*'/' *',I4,' ROWS ALLOCATED (''NROW''
     2 FROM DAT4 CARD).',T131,
     3 '*'/' *',I4,' ROWS SPECIFIED ON ''TTOP'' CARD.',T131,'*'/
     4 ' * ''TTOP'' CARD READS: ',20A4,T131,'*'/1X,65('**')//)
 1010 FORMAT ('0*** ERROR ',60('**')/' * NUMBER OF NODES OF TOP EVENT EX
     1CEEDS SPACE ALLOCATED.',T131,'*'/' *',I4,' NODES ALLOCATED (''LNRO
     2W'' FROM DAT2 CARD).',T131,'*'/' *',I4,' NODES SPECIFIED ON ''TTOP
     3'' CARD.',T131,'*'/' * ''TTOP'' CARD READS: ',20A4,T131,'*'/
     4  1X,65('**')//)
 1011 FORMAT (' *** INPUT ERROR ***'/1X,'NODE',I6,' IS NOT UNIQUE.'/)
 1012 FORMAT (' *** INPUT ERROR ***'/1X,'NODE',I6,' IS NOT BETWEEN 1 AND
     2',I6,' (MNODE).'/)
 1013 FORMAT (' *** INPUT ERROR ***'/1X,'NODE',I6,' HAS NOT BEEN DEFINED
     2.'/)
 1014 FORMAT ('0EVENT:  ',A8/' NUMBER OF ROWS  =',I3/
     1        ' NUMBER OF NODES =',I3/' TOP EVENT NODES ='//
     2        ' NODE',3X,'COMPONENT',5X,'OUTPUT'/)
 1015 FORMAT (I5,I6,3X,A8,I4)
 1016 FORMAT (//'0TABLE FOR TOP EVENT:'//' TTOP',6X,'NODE(S):',2X,10I5/)
 1017 FORMAT ('0*** INPUT ERROR ***'/' END CARD FOUND WHERE ''',A4,''' C
```

```
     1ARI EXPECTED.'/' DATA ERROR, PREVIOUS MISPUNCH, OR CARD MISSING OR
     2 OUT OF ORDER.'/' CORRECTIVE ACTION WILL BE ATTEMPTED.'/)
 1018 FORMAT ('0*** INPUT ERROR ***'/' MORE THAN THE',I4,' '' TOP'' CARD
     1S SPECIFIED HAVE BEEN INPUT.  EXTRA CARDS FOLLOW.'/' PROGRAM TERMI
     1NATING; VALIDATION OF REMAINING CARDS WILL BE ATTEMPTED.'//
     2  1X,20A4)
 1019 FORMAT (//'0&BC'//' BOUNDARY CONDITIONS FOR ',20A4//)
 1020 FORMAT (' *** INPUT ERROR ***'/' COMPONENT ''',A8,''' DOES NOT EXI
     1ST OR HAS NO INTERNAL NODES.'/)
 1021 FORMAT (' *** WARNING ***'/' NODE',I6,' IS A TOP EVENT AS WELL AS
     1A BOUNDARY CONDITION.  BOUNDARY CONDITION WILL BE IGNORED.'/)
 1022 FORMAT (/38H0NO BOUNDARY CONDITIONS HAVE BEEN SET.//)
 1023 FORMAT (16H *** WARNING ***/52H &END, &OUT OR END OF FILE REACHED
     1WITH NO END CARD./28H DATA MAY HAVE BEEN OMITTED./
     2  23H0*** JOB CONTINUING ***/)
 1024 FORMAT (/47H0*** JOB TERMINATING DUE TO PREVIOUS ERRORS ***/)
 1025 FORMAT (/20H0*** INPUT ERROR ***/1X,27H &END OR &OUT FOUND WHERE '
     1         ,A4,16H' CARD EXPECTED./1X,24H *** JOB TERMINATING ***/)
 1026 FORMAT (16H0*** WARNING ***/46HUNEXPECTED &END, &OUT OR END OF FI
     1LE REACHED./28H DATA MAY HAVE BEEN OMITTED./
     2  23H0*** JOB CONTINUING ***/)
 1027 FORMAT (16H0*** WARNING ***/33H UNEXPECTED &OUT READ (KOUT = 0)./
     1        38H EXTRA FAILURE DATA WILL BE NEGLECTED./
     2        20H EXTRA CARDS FOLLOW://1X,20A4)
 1028 FORMAT (16H0*** WARNING ***/47H &END OR END OF FILE REACHED WITHOU
     1T &OUT CARD./39H REQUIRED PREP DATA MISSING (KOUT = 1)./
     1        48H FAULT TREE WILL BE OUTPUT WITHOUT FAILURE DATA./)
      END
      SUBROUTINE DC IT (NTYPE,IROW,NINT,NIN,NOUT,NROW,JROW,NCMP,MOUT,
     1                  ITYPE,INODE,X,IGATE,JGATE,GATE,CMPNAM,
     2                  NLIB,LNROW,MXROW2,NNODE,MNODE,NNCMP,LNRP1,MGATE,
     3                  MXINT2,NGSIZE,NODES,NGATE)
C
C        SUBROUTINE TO GENERATE FAULT TREE
      DIMENSION NTYPE(NLIB),IROW(NLIB),NINT(NLIB),NIN(NLIB),NOUT(NLIB),
     1          NROW(NLIB),JROW(LNROW,MXROW2),NCMP(NNODE),MOUT(NNODE),
     2          ITYPE(NNCMP),INODE(LNRP1,NNCMP),IGATE(MGATE),
     3          JGATE(MGATE),KIND(2)
      COMMON TITLE(20),XXXX(20),IERR,IEDIT,IDUM(12),NNAME,JPRINT,KOUT
      DOUBLE PRECISION NNAME,CMPNAM(NNCMP)
      INTEGER X(2,NODES),GATE(NGSIZE)
      DATA KIND(1),KIND(2)/'AND ','OR  '/
      DO 98 I = 1,NGSIZE
   98   GATE(I) = 0
      DO 99 I = 1,MGATE
        IGATE(I) = 0
   99   JGATE(I) = 0
C
C         IDUM1   = NUMBER OF ROWS OF TABLE
C         IDUM2   = NUMBER OF COLUMNS (NODES)
      IDUM1  = NROW(NLIB)
      IDUM2  = NIN(NLIB)
```

```
      IPRINT = JPRINT
      IF (IPRINT .GT. 0) WRITE (6,1000) TITLE
C
C       BEGIN TO GENERATE TREE WITH TOP EVENT.
C       IF IDUM1   = IDUM2   = 1, TOP GATE MUST BE DEVELOPED FURTHER.
C
C       IF IDUM1   .GT. 1, TOP GATE IS OR GATE.
C
      IGATE(1) = 1
      JGATE(1) = 0
      INDEX    = 1
      JDEX     = 0
      GATE(3)  = 0
      GATE(4)  = 0
      GATE(5)  = 0
      IROWI    = IROW(NLIB)
      IF (IDUM1 .EQ. 1) GO TO 101
C
C       TOP GATE IS OR GATE WITH IDUM1   INPUTS
C       NOW FILL IN ENTRIES OF TOP GATE.
      GATE(1)  = -2
      GATE(2)  = IDUM1
      DO 100 JDUM = 1,IDUM1
        JJDUM = 2*JDUM + 4
        GATE(JJDUM)   = -NNCMP
  100   GATE(JJDUM+1) = JDUM
      IGATE(2) = 2*IDUM1 + 6
      GO TO 110
  101 IF (IDUM2 .EQ. 1) GO TO 104
C
C       TOP GATE IS AND WITH IDUM2 INPUTS.
      GATE(1) = -1
      GATE(2) = IDUM2
      DO 103 JDUM = 1,IDUM2
        JJDUM = 2*JDUM + 4
        JNODE = INODE(JDUM,NNCMP)
        IMODE = JROW(JDUM,IROWI)
  102   X(1,JNODE)   = IMODE
        X(2,JNODE)   = 1
        GATE(JJDUM) =-JNODE
  103   GATE(JJDUM+1) = IMODE
      IGATE(2) = 2*IDUM2 + 6
      GO TO 110
  104 CONTINUE
C
C       TOP GATE REQUIRES FURTHER DEVELOPMENT AS OR GATE.
C       SET TOP NODE AND FIND FIRST COMPONENT TO CHECK.
      JNODE = INODE(1,NNCMP)
      IMODE = JROW(1,IROWI)
  105 X(1,JNODE) = IMODE
      X(2,JNODE) = 0
C
```

```
C       GO TO OR LOOP TO CONSTRUCT TOP GATE.
        GO TO 203
  110   CONTINUE
        JGATE(2) = 1
        INDEX    = 2
        JDEX     =1
C       TOP GATE HAS BEEN GENERATED.
C       PRINT OUTPUT, IF DESIRED, AND ENTER PROPER GATE LOOP.
        KK    = IGATE(2) - 1
        IKIND = -GATE(1)
        IF (IPRINT .GT. 0) WRITE (6,1003)  KIND(IKIND),GATE(2),GATE(3),
       1                                   (GATE(II),II=6,KK)
        GATE(KK + 4) = 0
        LDEX = 1
        KDEX = 6
        GO TO (202,300),IKIND
C*******************************************************************************
C                                                                               *
C       BEGINNING OF OR LOOP.                                                   *
C       THIS LOOP IS ENTERED EACH TIME                                          *
C       A BRANCH OF AN AND GATE IS BEGUN.                                       *
C       FIRST LOCATE PROPER BRANCH TO SEARCH.                                   *
C                                                                               *
C*******************************************************************************
  200   CONTINUE
        JDEX = JGATE(INDEX)
        KDEX = IGATE(JDEX) + 3
        LLDEX = GATE(KDEX - 2) + GATE(KDEX - 1)
        DO 201 J = 1,LLDEX
          KDEX = KDEX + 2
          IF (GATE(KDEX) .LT. 0) GO TO 202
  201   CONTINUE
        GO TO 2030
  202   CONTINUE
C
C       KDEX HAS BEEN SET TO LOCATION OF BRANCH TO BE DEVELOPED.
C       GATE(KDEX) CONTAINS NODE TO BE CHECKED.
        JNODE = -GATE(KDEX)
        IMODE = GATE(KDEX + 1)
        GATE(KDEX) = INDEX
        GATE(KDEX + 1) = -1
C
C       SINGLE-INPUT GATES TRANSFER TO THIS POINT.
C       TO GENERATE 'OR' GATES.
C       FIRST CHECK FOR ARRAY SPACE LEFT.
C       THEN CHECK COMPONENT ROWS FOR MATCHES.
  203   CONTINUE
        ICOMP = NOMP(JNODE)
        IOUT  = MOUT(JNODE)
        JTYPE = ITYPE(ICOMP)
        IDEX = IGATE(INDEX)
        LIROW = IDEX + 4 + 2*NROW(JTYPE)
```

```
      IF (ILROW .GT. NGSIZE) GO TO 920
      IPLUS = NIN(JTYPE) + NINT(JTYPE)
      ISPOT = IPLUS + IOUT
      ITOT  = IPLUS + NOUT(JTYPE)
      ISET  = 0
      IDEXP1= IDEX + 1
      GATE(IDEX)   = -2
      GATE(IDEXP1) = 0
      GATE(IDEX+2) = 0
C
C       NOW SEARCH FOR VALID OUTPUT MODE.
      IRCWI = IRCW(JTYPE)
      NRCWI = NRCW(JTYPE) + IRCWI - 1
      DO 210 IR = IRCWI,NRCWI
        IF (JROW(ISPOT,IR) .NE. IMODE) GO TO 210
        ISET = 1
C
C         OUTPUT MATCH.  NOW CHECK FOR COMPLETE MATCH.
        DO 205 IS = 1,ITOT
          KNODE = INODE(IS,ICOMP)
          JMODE = X(1,KNODE)
          IF (JMODE .EQ. -1) GO TO 205
          IF ((JRCW(IS,IR) .NE.  -1)  .AND.
     1        (JRCW(IS,IR) .NE.  JMODE)) GO TO 206
  205     CONTINUE
C
C           COMPLETE ROW MATCH.  SET GATE INPUT
        GATE(IDEXP1) = GATE(IDEXP1) + 1
        IIDEX = 2*GATE(IDEXP1) + 3 + IDEX
        GATE(IIDEX) = -ICOMP
        GATE(IIDEX + 1) = IR + 1 - IRCWI
        GO TO 210
C
C         NODE CONTRADICTION IN ROW IR.  WRITE OUTPUT
  206   IF (IPRINT .LT. 1) GO TO 210
        NGATE = X(2,KNODE)
        KROW  = IR + 1 - IROWI
        IF (KNODE .GT. MNODE) GO TO 207
        IF (NGATE .GT. 0) WRITE (6,1010) INDEX,CMPNAM(ICOMP),JTYPE,
     1                    KROW,KNODE,JROW(IS,IR),JMODE,NGATE
        IF (NGATE .EQ. 0) WRITE (6,1011) INDEX,CMPNAM(ICOMP),JTYPE,
     1                    KROW,KNODE,JROW(IS,IR),JMODE
        GO TO 210
  207   KNODE = IS - NINT(JTYPE)
        IF (NGATE .GT. 0) WRITE (6,1012) INDEX,CMPNAM(ICOMP),JTYPE,
     1                    KROW,KNODE,JROW(IS,IR),JMODE,NGATE
        IF (NGATE .EQ. 0) WRITE (6,1013) INDEX,CMPNAM(ICOMP),JTYPE,
     1                    KROW,KNODE,JROW(IS,IR),JMODE
  210   CONTINUE
      IF (ISET .EQ. 1) GO TO 211
C
C       NO OUTPUT MODE MATCH FOUND.
```

```
C       PRINT MESSAGE AND DELETE GATE.
      IF (IPRINT.GT.0) WRITE(6,1004)IMODE,JNODE,CMPNAM(ICCMP),JTYPE,IOUT
      GO TO 212
  211 IF (GATE(IDEXP1) .NE. 0) GO TO 220
C*********************************************************************
C                                                                     *
C       NO COMPLETE FLOW MATCH FOUND.                                 *
C       DELETE GATE AND EDIT PREVIOUS GATES.                          *
C                                                                     *
C*********************************************************************
      IF (IPRINT .GT. 0) WRITE (6,1005) CMPNAM(ICCMP),JTYPE
  212 IF (INDEX .EQ. 1) GO TO 2020
      IF (IPRINT .GT. 0) WRITE (6,1006) INDEX
C
C       GATE IS DELETED SIMPLY BY NOT INCREMENTING INDEX.
C       BACKTRACK TO PREVIOUS GATE.  DELETE IF 'AND',
C       DELETE INPUT IF 'OR'.
  213 IF (JDEX .EQ. 0) GO TO 2040
      NDEX  = IGATE(JDEX)
      IKIND = -GATE(NDEX)
      IF (IKIND .EQ. 2) GO TO 214
C
C       PREVIOUS GATE IS 'AND' AND WILL BE DELETED.
      IF (IPRINT .GT. 0) WRITE (6,1007) INDEX,KIND(1),JDEX,JDEX
      INDEX = JDEX
      JDEX  = JGATE(INDEX)
      GO TO 213
C
C       PREVIOUS GATE IS 'OR'. ELIMINATE BRANCH.
  214 LLDEX = GATE(NDEX + 1) + GATE(NDEX + 2)
      IF (LLDEX .EQ. 1) GO TO 219
      KDEX = NDEX + 3
      DO 215 K = 1,LLDEX
        KDEX = KDEX + 2
        IF (GATE(KDEX).EQ.INDEX  .AND.  GATE(KDEX+1).EQ.-1) GO TO 216
  215   CONTINUE
C
C       KDEX IS LOCATION OF BRANCH TO BE ELIMINATED.
C       MOVE UP SUCCEEDING ENTRIES (IF ANY).
C       OTHERWISE, GATE IS FINISHED.
      IB = 215
      PRINT 1,IB
C*********************************************************************
C*********************************************************************
  216 LDEX = (KDEX - NDEX - 3)/2
      KK   = NDEX + 4 + 2*LLDEX
      IF (IPRINT .GT. 0) WRITE (6,1008)INDEX,KIND(IKIND),JDEX,LDEX,LLDEX
      GATE(NDEX + 1) = GATE(NDEX + 1) - 1
      IF (LDEX .LT. LLDEX) GO TO 217
C
C       GATE HAS NO FURTHER ENTRIES.  GATE COMPLETED.
      IF (IPRINT .GT. 1) WRITE (6,1020) JDEX,(GATE(II),II=NDEX,KK)
```

```
      GO TO 410
  217 LLDEX1 = LDEX - 1
      DO 218 K = LDEX,LLDEX1
        GATE(KDEX)   = GATE(KDEX + 2)
        GATE(KDEX+1) = GATE(KDEX + 3)
  218   KDEX = KDEX + 2
      IF (IPRINT .GT. 1) WRITE (6,1020) JDEX,(GATE(II),II=NDEX,KK)
      GO TO 401
C
C       PREVIOUS GATE IS SINGLE INPUT 'OR' AND WILL BE ELIMINATED.
  219 IF (IPRINT .GT. 0) WRITE (6,1009) INDEX,KIND(IKIND),JDEX,JDEX
      INDEX = JDEX
      JDEX  = JGATE(INDEX)
      GO TO 213
  220 CONTINUE
C*****************************************************************************
C                                                                             *
C       ROW MATCH FOUND (TRANSFER FROM STATEMENT 211).                        *
C       NOW PROGRAM GENERATES 'OR' GATE INPUTS.                               *
C                                                                             *
C*****************************************************************************
      IF (GATE(IDEXP1) .GT. 1) GO TO 221
C
C       'OR' GATE HAS ONLY ONE ROW, THUS IS NOT TRUE GATE.
C       RETURN TO PREVIOUS GATE AND DEVELOP CURRENT BRANCH
C       AS AN 'AND' GATE.  LDEX,KDEX,ICOMP,GATE(KDEX)
C       HAVE ALREADY BEEN SET.
      IF (IPRINT .GT. 0) WRITE (6,1019) INDEX,KIND(2),JDEX
      MROW = GATE(IIDEX + 1)
  221 CONTINUE
C
C       'OR' GATE HAS MULTIPLE ROWS.
C       SET GATE PARAMETERS AND INDICES FOR FIRST ROW.
      IF (GATE(IDEX+3).LT.10000 .AND. GATE(IDEX+3).NE.0) GO TO 222
      GATE(IDEX+3) = JNODE
      GATE(IDEX+4) = IMODE
  222 KK = IDEX + 4 + 2*GATE(IDEXP1)
      KG = IDEX + 5
      IF (IPRINT .GT. 0  .AND.  JDEX .GT. 0) WRITE (6,1002) INDEX,
     1   KIND(2),GATE(IDEXP1),GATE(IDEX+2),IMODE,JNODE,
     2   (GATE(II),II=KG,KK)
      IF (IPRINT .GT. 0  .AND.  JDEX .EQ. 0) WRITE (6,1003) KIND(2),
     1   GATE(2),GATE(3),(GATE(II),II=6,KK)
      IF (GATE(IDEXP1) .EQ. 1) GO TO 301
      JDEX  = INDEX
      INDEX = INDEX + 1
      IF (INDEX .GT. MGATE) GO TO 900
      IGATE(INDEX) = KK + 1
      JGATE(INDEX) = JDEX
      GATE(KK + 4) = 0
      LDEX = 1
C*****************************************************************************
```

```
C                                                                       *
C       BEGINNING OF 'AND' LOOP.                                        *
C       THIS LOOP IS ENTERED EACH TIME                                  *
C       A BRANCH OF AN 'OR' GATE IS BEGUN.                              *
C       PRESET VARIABLES:                                               *
C         INDEX         = GATE NUMBER OF CURRENT 'AND' GATE             *
C         JGATE(INDEX)  = INDEX  OF GATE ABOVE CURRENT GATE             *
C         IGATE(INDEX)  = STARTING POINT OF CURRENT GATE IN ARRAY 'GATE'*
C         IDEX          = BRANCH OF GATE 'JGATE' BEING EVALUATED        *
C                                                                       *
C       VARIABLES TO BE SET:                                            *
C         LROW          = ROW IN TYPE LIBRARY TO BE SEARCHED            *
C                                                                       *
C***********************************************************************
  300 CONTINUE
      JDEX  = JGATE(INDEX)
      NDEX  = IGATE(JDEX)
      KDEX  = NDEX + 3 + 2*LDEX
      ICOMP = -GATE(KDEX)
      MROW  =  GATE(KDEX + 1)
      GATE(KDEX)    = INDEX
      GATE(KDEX+1) = -1
  301 JTYPE = ITYPE(ICOMP)
      LROW  = IROW(JTYPE) + MROW - 1
      INPT = NIN(JTYPE)
      INT  = NINT(JTYPE)
      JOUT = NOUT(JTYPE)
C
C       FIRST CHECK FOR ARRAY SPACE LEFT.
      LLROW = IGATE(INDEX) + 4 + 2*(INPT + INT)
      IF (LLROW .GT. NGSIZE) GO TO 920
C
C       NODES WERE CHECKED FOR CONTRADICTIONS WITH PRESET VALUES
C       IN LOOP 205.  NOW CHECK FOR OTHER PRESET VALUES, SET NODES,
C       AND GENERATE 'AND' GATE.
      IDEX = IGATE(INDEX)
      GATE(IDEX)     = -1
      GATE(IDEX + 1) =  0
      GATE(IDEX + 2) =  0
C
C       CHECK COMPONENT INPUTS FIRST.
      IF (INPT .EQ. 0) GO TO 311
      DO 310 IN = 1,INPT
        KMODE = JROW(IN,LROW)
        IF (KMODE .EQ. -1) GO TO 310
        KNODE = INODE(IN,ICOMP)
        JMODE = X(1,KNODE)
        IF (JMODE .NE. -1)  GO TO 309
C
C         NODE NOT PRESET, SO SET NODE AND GATE INPUTS.
        X(1,KNODE) = KMODE
        X(2,KNODE) = INDEX
```

```
        GATE(IDEX + 1) = GATE(IDEX + 1) + 1
        IIDEX = IDEX + 3 + 2*GATE(IDEX + 1)
        GATE(IIDEX)   = -KNODE
        GATE(IIDEX+1) =  KMODE
        GO TO 310
  309   CONTINUE
C
C         NODE PRESET. DO NOT SET GATE INPUT.
      IB = 309
      IF (JMODE .NE. KMODE) PRINT 1,IE
C*******************************************************************************
C*******************************************************************************
        IF (IPRINT .LT. 1) GO TO 310
        NGATE = X(2,KNODE)
        IF (NGATE .NE. 0) WRITE (6,1014) INDEX,KNODE,JMODE,NGATE
        IF (NGATE .EQ. 0) WRITE (6,1015) INDEX,KNODE,JMODE
  310   CONTINUE
C
C         NOW CHECK INTERNAL NODES.
  311 IF (INT .EQ. 0) GO TO 321
      INETP1 = INET + 1
      IPLUS  = INET + INT
      DO 320 IN = INETP1,IPLUS
        KMODE = JBCW(IN,LBCW)
        IF (KMODE .EQ. -1) GO TO 320
        KNODE = INODE(IN,ICOME)
        JMODE = X(1,KNODE)
        IF (JMODE .NE. -1) GO TO 319
C
C         NODE NOT PRESET.  SET NODE.
        X(1,KNODE) = KMODE
        X(2,KNODE) = INDEX
C
C         THIS IS A PRIMARY INPUT.  SET GATE INPUTS.
        GATE(IDEX + 2) = GATE(IDEX + 2) + 1
        IIDEX = IDEX + 3 + 2*(GATE(IDEX+1) + GATE(IDEX+2))
        GATE(IIDEX)   = KNODE
        GATE(IIDEX+1) = KMODE
        GO TO 320
  319   CONTINUE
C
C         NODE PRESET.  DO NOT SET GATE INPUT.
      IB = 319
      IF (JMODE .NE. KMODE) PRINT 1,IE
C*******************************************************************************
C*******************************************************************************
        IF (IPRINT .LT. 1) GO TO 320
        NGATE = X(2,KNODE)
        IF (NGATE .NE. 0) WRITE (6,1016) INDEX,KNODE,JMODE,NGATE
        IF (NGATE .EQ. 0) WRITE (6,1017) INDEX,KNODE,JMODE
  320   CONTINUE
C
```

```
C          NOW CHECK OUTPUTS AND SET NODES. (DO NOT SET GATE INPUTS.)
  321 IPLUS = INPT + INT + 1
      ITOT  = IPLUS + JOUT - 1
      IF (ICOMP .EQ. NNCMP) GO TO 330
      DO 329 IN = IPLUS,ITOT
        KMODE = JROW(IN,LROW)
        IF (KMODE .EQ. -1) GO TO 329
        KNODE = INODE(IN,ICOMP)
        JMODE = X(1,KNODE)
        IF (JMODE .NE. -1) GO TO 328
        X(1,KNODE) = KMODE
        X(2,KNODE) = INDEX
        GO TO 329
  328   CONTINUE
      IB = 328
      IF (JMODE .NE. JROW(IN,LROW)) PRINT 1,IB
C*****************************************************************************
C*****************************************************************************
        IF (IPRINT .LT. 1) GO TO 329
        NGATE = X(2,KNODE)
        IF (NGATE .NE. 0) WRITE (6,1014) INDEX,KNODE,JMODE,NGATE
        IF (NGATE .EQ. 0) WRITE (6,1015) INDEX,KNODE,JMODE
  329   CONTINUE
C
C          ALL NODES CHECKED.  DETERMINE TOTAL INPUTS.
  330 LLDEX = GATE(IDEX + 1) + GATE(IDEX + 2)
      IF (LLDEX .GT. 0) GO TO 340
C*****************************************************************************
C                                                                             *
C       ALL INPUTS HAVE BEEN PRESET: GATE IS ALWAYS TRUE.                     *
C       DELETE GATE AND EDIT PREVIOUS GATES.                                  *
C       DELETE PREVIOUS 'OR' GATES AND BRANCH OF LAST PREVIOUS 'AND'.         *
C                                                                             *
C*****************************************************************************
      IF (IPRINT .GT. 0) WRITE (6,1018) CMPNAM(ICOMP),JTYPE,MROW,INDEX
  331 IF (JDEX .EQ. 0) GO TO 2060
      NDEX  = IGATE(JDEX)
      IKIND = -GATE(NDEX)
      IF (IKIND .EQ. 1) GO TO 332
C
C       PREVIOUS GATE IS 'OR' AND WILL BE DELETED.
      IF (IPRINT .GT. 0) WRITE (6,1007) INDEX,KIND(IKIND),JDEX,JDEX
      INDEX = JDEX
      JDEX  = JGATE(INDEX)
      GO TO 331
C
C       PREVIOUS GATE IS 'AND'. LOCATE AND ELIMINATE CURRENT BRANCH.
  332 LLDEX = GATE(NDEX + 1) + GATE(NDEX + 2)
      IF (LLDEX .EQ. 1) GO TO 337
      KDEX = NDEX + 3
      DO 333 K = 1,LLDEX
        KDEX = KDEX + 2
```

```
        IF (GATE(KDEX) .EQ. INDEX .AND. GATE(KDEX+1) .EQ. -1) GO TO 334
  333   CONTINUE
C
C         KDEX HAS BEEN SET TO LOCATION OF BRANCH TO BE ELIMINATED.
C         MOVE UP SUCCEEDING ENTRIES (IF ANY).
C         OTHERWISE, GATE IS FINISHED.
      IE = 333
      PRINT 1,IB
C*********************************************************************
C*********************************************************************
  334 LDEX = (KDEX - NDEX - 3)/2
      KK   = NDEX + 4 + 2*LLDEX
      IF (IPRINT .GT. 0) WRITE (6,1008)INDEX,KIND(IKIND),JDEX,LDEX,LLDEX
      GATE(NDEX+1) = GATE(NDEX + 1) - 1
      IF (LDEX .LT. ILDEX) GO TO 335
C
C       GATE HAS NO FURTHER ENTRIES TO BE MOVED.  GATE COMPLETED.
      IF (IPRINT .GT. 1) WRITE (6,1020) JDEX,(GATE(II),II=NDEX,KK)
      GO TO 410
  335 LLDEX1 = LLDEX - 1
      DO 336 K = LDEX,LLDEX1
        GATE(KDEX)   = GATE(KDEX + 2)
        GATE(KDEX+1) = GATE(KDEX + 3)
  336   KDEX = KDEX + 2
      IF (IPRINT .GT. 1) WRITE (6,1020) JDEX,(GATE(II),II=NDEX,KK)
      GO TO 400
C
C       PREVIOUS GATE IS SINGLE INPUT 'AND' AND WILL BE ELIMINATED.
  337 IF (IPRINT .GT. 0) WRITE (6,1009) INDEX,KIND(IKIND),JDEX,JDEX
      INDEX = JDEX
      JDEX  = JGATE(INDEX)
      GO TO 331
  340 CONTINUE
C*********************************************************************
C       'AND' GATE INPUTS HAVE BEEN FOUND (TRANSFER FROM 330).      *
C       FIRST, CHECK TO SEE IF GATE HAS MORE THAN ONE INPUT.        *
C       IF NOT, GATE IS NOT TRUE GATE: RETURN TO PREVIOUS GATE      *
C       AND DEVELOP CURRENT BRANCH.                                 *
C                                                                   *
C*********************************************************************
      IF (LLDEX .GT. 1) GO TO 350
C
C       GATE HAS ONLY ONE INPUT.  DELETE GATE AND RESET NODES.
C       FIRST LOCATE AND SET NODES BACK INTO PREVIOUS GATE.
      IF (IPRINT .GT. 0) WRITE (6,1019) INDEX,KIND(1),JDEX
      KNODE = IABS(GATE(IDEX+5))
      IE = 340
      IF (X(2,KNODE) .NE. INDEX) PRINT 1,IB
C*********************************************************************
C*********************************************************************
      X(2,KNODE) = JDEX
C
```

```
C       NOW SEARCH OUTPUT NODES, IF MORE THAN ONE EXIST.
        IF (JOUT .LT. 2) GO TO 342
        DO 341 IN = IPLUS,ITOT
          KNODE = INCDE(IN,ICOMP)
  341     IF (X(2,KNODE) .EQ. INDEX) X(2,KNODE) = JDEX
C
C         ALL NODES HAVE BEEN RESET.  NOW RESET GATE INPUTS.
  342 IF (GATE(IDEX+1) .EQ. 1) GO TO 343
C
C       CURRENT GATE HAS ONLY PRIMARY INPUT.
C       INSERT ENTRIES OF CURRENT GATE INTO PREVIOUS GATE.
        IF (JDEX .EQ. 0) GO TO 500
        NDEX = IGATE(JDEX)
        GATE(NDEX+1) = GATE(NDEX + 1) - 1
        GATE(NDEX+2) = GATE(NDEX + 2) + 1
        GATE(KDEX)   = GATE(IDEX + 5)
        GATE(KDEX+1) = GATE(IDEX + 6)
        GATE(IDEX+3) = 0
        KK = NDEX + 4 + 2*(GATE(NDEX+1) + GATE(NDEX+2))
        IF (IPRINT .GT. 1) WRITE (6,1020) JDEX,(GATE(II),II=NDEX,KK)
C
C       NOW RETURN TO PREVIOUS GATE, JDEX, SENDING KDEX, LDEX, INDEX,
C       AND DEVELOP NEW BRANCH.  FIRST CHECK IF GATE JDEX IS FINISHED.
        IF (LDEX .EQ. (GATE(NDEX+1) + GATE(NDEX+2))) GO TO 410
        LDEX = LDEX + 1
        GO TO 400
  343 CONTINUE
C
C       CURRENT GATE HAS SINGLE GATE INPUT.
C       SET PARAMETERS AND DEVELOP AS 'OR' (LDEX & KDEX HAVE BEEN SET).
        JNODE = -GATE(IDEX + 5)
        IMODE =  GATE(IDEX + 6)
        IF (GATE(IDEX+3).LT.10000 .AND. GATE(IDEX+3) .NE. 0) GO TO 203
        GATE(IDEX+3) = JNODE
        GATE(IDEX+4) = IMODE
        GO TO 203
  350 CONTINUE
C
C       'AND GATE HAS MULTIPLE INPUTS. IF ANY UNDEVELOPED INPUTS EXIST,
C       BEGIN TO DEVELOP FIRST BRANCH; OTHERWISE, GATE IS FINISHED.
        KK = IDEX + 4 + 2*LIDEX
        KG = IDEX + 5
        IF (GATE(IDEX+3).LT.10000 .AND. GATE(IDEX+3).NE.0) GO TO 351
        GATE(IDEX+3) = 10000 + ICOMP
        GATE(IDEX+4) = MROW
        IF (IPRINT .GT. 0) WRITE (6,1001) INDEX,KIND(1),GATE(IDEX+1),
       1   GATE(IDEX+2),MROW,ICOMP,(GATE(II),II=KG,KK)
        GO TO 352
  351 IF (IPRINT .GT. 0) WRITE (6,1002) INDEX,KIND(1),GATE(IDEX+1),
      1   GATE(IDEX+2),GATE(IDEX+4),GATE(IDEX+3),(GATE(II),II=KG,KK)
  352 CONTINUE
        JDEX = INDEX
```

```
      INDEX = INDEX + 1
      IF (INDEX .GT. MGATE) GO TO 900
      IGATE(INDEX) = KK + 1
      JGATE(INDEX) = JDEX
      GATE(KK + 4) = 0
      KDEX = IDEX + 5
      LDEX = 1
      IF (GATE(IDEX+1) .NE. 0) GO TO 202
C
C       GATE HAS NO UNDEVELOPED BRANCHES.
      NDEX = IDEX
      GO TO 410
  400 CONTINUE
C*************************************************************************
C                                                                        *
C       GATE ENTERS THIS REGION EACH TIME A BRANCH IS COMPLETED,         *
C         (FROM 336, 342).                                               *
C       JDEX  = INDEX OF CURRENT GATE.                                   *
C       INDEX = INDEX OF NEXT GATE TO BE GENERATED.                      *
C       NDEX  = IGATE(JDEX)                                              *
C       IDEX  = INDEX OF NEXT BRANCH.                                    *
C                                                                        *
C       FIRST FIND IF ANY UNDEVELOPED BRANCHES REMAIN.                   *
C                                                                        *
C*************************************************************************
      KDEX = NDEX + 3 + 2*LDEX
      IF (GATE(KDEX) .LT. 0) GO TO 401
C
C       NO UNDEVELOPED BRANCHES REMAIN: GATE FINISHED.
      GO TO 410
  401 CONTINUE
C
C       GATE HAS FURTHER BRANCHES (FROM 218/400/484.
C       LDEX HAS BEEN SET TO NEXT UNDEVELOPED BRANCH
C         (FROM 216/334/342/484).
      IKIND = -GATE(NDEX)
      JGATE(INDEX) = JDEX
      IDEX = IGATE(INDEX)
      GATE(IDEX + 3) = 0
      IF (IKIND .EQ. 1) GO TO 202
C
C       GATE IS 'OR'.  RESET NODES, THEN BEGIN NEW BRANCH (300).
      DO 402 I = 1,NODES
        IF (X(2,I) .LT. JDEX) GO TO 402
        X(1,I) = -1
        X(2,I) = -1
  402   CONTINUE
      GO TO 300
  410 CONTINUE
C*************************************************************************
C       GATE ENTERS THIS REGION WHEN FINISHED...                         *
C       ...FROM STEPS 216,334,342,352,400,483.                           *
```

```
C*********************************************************************
      IF (IPRINT .GE. 1) WRITE (6,1022) JDEX
      GATE(NDEX) = -GATE(NDEX)
      LLDEX      = GATE(NDEX + 1) + GATE(NDEX + 2)
      KK         = NDEX + 2*LLDEX + 4
      IF (IPRINT .GT. 1) WRITE (6,1020) JDEX,(GATE(II),II=NDEX,KK)
      IF (LLDEX .EQ. 1) GO TO 485
C
C       GATE HAS MULTIPLE INPUTS.  CHECK FOR REDUNDANCIES/CONTRADICTIONS
C       FIRST ARRANGE GATE TO INCLUDE GATE ENTRIES FIRST.
      INGATE = GATE(NDEX + 1)
      IF (INGATE.EQ.0  .OR.  GATE(NDEX+2).EQ.0) GO TO 420
      ISPOT = NDEX + 5
      ND    = NDEX + 4
      IG    = NDEX + 3 + 2*INGATE
      DO 412 I = 1,LLDEX
        ND = ND + 2
        IF (GATE(ND) .NE. -1) GO TO 412
C
C         GATE INPUT FOUND.
        IF (ISPOT .EQ. (ND-1)) GO TO 411
        GATE(ND) = GATE(ISPOT + 1)
        GATE(ISPOT+1) = -1
        ISTORE       = GATE(ND-1)
        GATE(ND-1)   = GATE(ISPOT)
        GATE(ISPOT) = ISTORE
  411   ISPOT = ISPOT + 2
        IF (ISPOT .GT. IG) GO TO 413
  412   CONTINUE
  413 IF (IPRINT .GT. 1) WRITE (6,1020) JDEX,(GATE(II),II=NDEX,KK)
C
C       GATE ENTRIES NOW ARRANGED.
C       NOW CHECK 'AND' GATES FIRST.
  420    IF (IEDIT.GE.99  .OR.
     1       (INGATE.EQ.0  .AND.  GATE(NDEX).EQ.1))  GO TO 482
         IF (GATE(NDEX) .EQ. 2) GO TO 455
C*********************************************************************
C                                                                     *
C       'AND' GATE POST GATE EDIT REGION                              *
C         CHECK FOR RESET OR CONTRADICTORY GATE ENTRIES                *
C         VS. 'AND' GATE NODES (PRIMARY EVENTS) SET.                  *
C         ROUTINE CHECKS GATES TO 3 LEVELS (JDEX, JDEX2, JDEX3).      *
C                                                                     *
C*********************************************************************
      ISPOT = NDEX + 3
      DO 454 I = 1,INGATE
        ISPOT  = ISPOT + 2
        JDEX2  = GATE(ISPOT)
        NDEX2  = IGATE(JDEX2)
        JNGATE = GATE(NDEX2 + 1)
        JPRIME = GATE(NDEX2 + 2)
        JLDEX  = JNGATE + JPRIME
```

```
      IF (GATE(NDEX2) .EQ. 2) GO TO 433
      IF (JNGATE .EQ. 0) GO TO 454
C
C       CHECK INPUTS TO SECOND LEVEL 'AND' GATE JDEX2.
C       SEARCH ONLY FOR THIRD LEVEL 'OR' GATES WITH PRIMARY INPUTS.
      JSPOT = NDEX2 + 3
      DO 429 J = 1,JNGATE
        JSPOT  = JSPOT + 2
        JDEX3  = GATE(JSPOT)
        NDEX3  = IGATE(JDEX3)
        IF (GATE(NDEX3) .EQ. 1) GO TO 429
        KPRIME = GATE(NDEX3 + 2)
        IF (KPRIME .EQ. 0)      GO TO 429
C
C         THIRD LEVEL GATE JDEX3 HAS KPRIME PRIMARY INPUTS.
C         CHECK TO SEE IF PRESET OR CONTRADICTORY.
        KSPOT = NDEX3 + 3 + 2*GATE(NDEX3 + 1)
        DO 424 K = 1,KPRIME
          KSPOT = KSPOT + 2
          KNODE = GATE(KSPOT)
          JMODE = X(1,KNODE)
          IF (JMODE .EQ. -1) GO TO 424
C
C           NODE HAS BEEN PREVIOUSLY SET.
          KMODE = GATE(KSPOT + 1)
          IF (IPRINT .GT. 0) WRITE(6,1016)JDEX3,KNODE,JMODE,X(2,KNODE)
          IF (KMODE .EQ. JMODE) GO TO 422
C
C           PRIMARY INPUT CONTRADICTS PRESET NODE.
C           DELETE PRIMARY INPUT TO GATE JDEX3 AND MOVE UP REST.
          IF (IPRINT .GT. 0) WRITE (6,1023) K,JDEX3
          GATE(NDEX3 + 2) = GATE(NDEX3 + 2) - 1
          IF (K .EQ. KPRIME) GO TO 424
          KPM1 = KPRIME - 1
          KKSPOT = KSPOT
          DO 421 K2 = K,KPM1
            GATE(KKSPOT)   = GATE(KKSPOT + 2)
            GATE(KKSPOT+1) = GATE(KKSPOT + 3)
  421       KKSPOT = KKSPOT + 2
          KSPOT = KSPOT - 2
          GO TO 424
  422     CONTINUE
C
C           PRIMARY INPUT HAS BEEN PRESET; THUS 'OR' GATE JDEX3
C           IS SURE TO OCCUR, AND WILL BE REMOVED.
C           REMOVE INPUT TO PRECEEDING 'AND' GATE JDEX2.
          IF (IPRINT .GT. 0) WRITE(6,1006) JDEX3
          IF (IPRINT .GT. 0) WRITE(6,1008) JDEX3,KIND(1),JDEX2,J,JLDEX
          GATE(NDEX3)   = -99
          JGATE(JDEX3)  = -99
          GATE(NDEX2+1) = GATE(NDEX2 + 1) - 1
          JM1 = JNGATE + JPRIME - 1
```

```
            IF (J .GT. JM1) GO TO 429
            JJSPOT = JSPOT
            DO 423 K2 = J,JM1
              GATE(JJSPOT)   = GATE(JJSPOT + 2)
              GATE(JJSPOT+1) = GATE(JJSPOT + 3)
  423         JJSPOT = JJSPOT + 2
            JSPOT = JSPOT - 2
            GO TO 429
  424       CONTINUE
C             CHECK OF PRIMARY INPUTS TO THIRD LEVEL GATE CONCLUDED.
C              IF NO INPUTS REMAIN, GATE JDEX CANNOT OCCUR, SO
C              ELIMINATE JDEX. IF ONE INPUT REMAINS, INSERT INTO JDEX2.
C
          KLDEX = GATE(NDEX3 + 1) + GATE(NDEX3 + 2)
          IL = NDEX3 + 4 + 2*KLDEX
          IF (IPRINT .GT. 1) WRITE (6,1020) JDEX3,(GATE(II),II=NDEX3,LL)
          IF (KLDEX .GE. 2) GO TO 429
          IF (KLDEX .EQ. 1) GO TO 425
          GATE(NDEX3)  = -99
          JGATE(JDEX3) = -99
          IF (IPRINT .LT. 1) GO TO 449
            WRITE (6,1024) JDEX3,JDEX3
            WRITE (6,1007) JDEX3,KIND(1),JDEX2,JDEX2
          GO TO 448
  425     CONTINUE
C
C           ONLY ONE INPUT TO GATE JDEX3 REMAINS.  INSERT DIRECTLY
C           INTO GATE JDEX2 AND REARRANGE ENTRIES, IF NECESSARY.
          CALL REDUCE (1,JDEX3,NDEX3,JDEX2,NDEX2,IPRINT,J,JNGATE,JSPOT,
     1                 GATE,JGATE,NGSIZE,MGATE)
 4000     CONTINUE
C
C           NOW SET PRIMARY INPUT FROM JDEX, OR INPUTS FROM LOWER
C           'AND' GATES INTO ARRAY 'X'.
          IF (GATE(NDEX3 + 1) .GT. 0) GO TO 4001
C
C           SET PRIMARY INPUT INTO ARRAY 'X'.
          KNODE = GATE(NDEX3 + 5)
          X(1,KNODE) = GATE(NDEX3 + 6)
          X(2,KNODE) = JDEX2
          JPRIME = JPRIME + 1
          GO TO 429
 4001     CONTINUE
C
C           SEARCH FOR LOWER LEVEL 'AND' GATES AND SET PRIMARY INPUTS.
          JDEX3 = GATE(NDEX3 + 5)
          NDEX3 = IGATE(JDEX3)
          IF (GATE(NDEX3) .EQ. 2) GO TO 429
          KPRIME = GATE(NDEX3 + 2)
          IF (KPRIME .EQ. 0) GO TO 429
          KSPOT = NDEX3 + 3 + 2*GATE(NDEX3 + 1)
          DO 4003 K = 1,KPRIME
```

```
            KSPOT = KSPOT + 2
            KNODE = GATE(KSPOT)
            JMODE = X(1,KNODE)
            IF (JMODE .EQ. -1) GO TO 4002
C
C             NODE HAS BEEN PREVIOUSLY SET; CHECK FOR AGREEMENT.
            IF (KMODE .EQ. JMODE) GO TO 4003
C
C             PRIMARY INPUT CONTRADICTS PRESET NODE AND CANNOT OCCUR.
C             DELETE JDEX3, JDEX2 AND JDEX AND RETURN.
            GATE(NDEX3)  = -99
            JGATE(JDEX3) = -99
            IF (IPRINT .LT. 1) GO TO 449
            WRITE (6,1016) JDEX3,KNODE,JMODE,X(2,KNODE)
            WRITE (6,1006) JDEX3
            WRITE (6,1007) JDEX3,KIND(1),JDEX2,JDEX2
            GO TO 448
 4002       CONTINUE
C
C             NODE HAS NOT BEEN PRESET.  SET NODE.
            X(1,KNODE) = GATE(KSPOT + 1)
            X(2,KNODE) = JDEX3
 4003       CONTINUE
  429   CONTINUE
C
C             GATE JDEX2 CHECK COMPLETED.
C             IF NO INPUTS REMAIN, DELETE INPUT TO GATE JDEX.
C             IF ONE INPUT REMAINS, INSERT DIRECTLY INTO GATE JDEX.
        JLDEX = GATE(NDEX2 + 1) + GATE(NDEX2 + 2)
        JJ    = NDEX2 + 4 + 2*JLDEX
        IF (IPRINT .GT. 1) WRITE (6,1020) JDEX2,(GATE(II),II=NDEX2,JJ)
        IF (JLDEX .GT. 1) GO TO 454
        IF (JLDEX .EQ. 1) GO TO 450
C
C           GATE JDEX2 HAS NO INPUTS.  DELETE INPUT TO JDEX.
        IF (IPRINT .GT. 0) WRITE (6,1024) JDEX2,JDEX2
  430   IF (IPRINT .GT. 0) WRITE (6,1008) JDEX2,KIND(1),JDEX,I,IIDEX
        GATE(NDEX + 1) = GATE(NDEX + 1) - 1
        GATE(NDEX2)    = -99
        JGATE(JDEX2)   = -99
        JM1 = INGATE + GATE(NDEX + 2) - 1
        IF (I .GT. JM1) GO TO 432
        IISPOT = ISPOT
        DO 431 K2 = I,JM1
          GATE(IISPOT)   = GATE(IISPOT + 2)
          GATE(IISPOT+1) = GATE(IISPOT + 3)
  431     IISPOT = IISPOT + 2
  432   ISPOT = ISPOT - 2
        IF (IPRINT .GT. 1) WRITE (6,1020) JDEX,(GATE(II),II=NDEX,KK)
        GO TO 454
  433   CONTINUE
C
```

```
C         GATE JDEX2 IS 'OF' GATE (TRANSFER FROM AFTER 420).
C         FIRST CHECK FOR DIRECT PRIMARY INPUTS.
        IF (JPRIME .EQ. 0) GO TO 436
        JSPCT = NDEX2 + 3 + 2*JNGATE
        DO 435 J = 1,JPRIME
          JSPOT = JSPCT + 2
          JNODE = GATE(JSPOT)
          JMODE = X(1,JNODE)
          IF (JMODE .EQ. -1) GO TO 435
C
C           NODE HAS BEEN PREVIOUSLY SET
          KMODE = GATE(JSPOT + 1)
          IF (IPRINT .GT. 0) WRITE (6,1016) JDEX2,JNODE,JMODE,X(2,JNODE)
          IF (IPRINT .GT. 0  .AND.  KMODE .EQ. JMODE)
     1        WRITE (6,1006) JDEX2
          IF (KMODE .EQ. JMODE) GO TO 430
C
C           PRIMARY INPUT CONTRADICTS PRESET NODE.
C           DELETE PRIMARY INPUT TO GATE JDEX2 AND MOVE UP REST.
          IF (IPRINT .GT. 0) WRITE (6,1023) J,JDEX2
          GATE(NDEX2 + 2) = GATE(NDEX2 + 2) - 1
          IF (J .EQ. JPRIME) GO TO 435
          JM1 = JPRIME - 1
          JJSPCT = JSPCT
          DO 434 J2 = J,JM1
            GATE(JJSPCT)   = GATE(JJSPCT + 2)
            GATE(JJSPOT+1) = GATE(JJSPCT + 3)
  434       JJSPCT = JJSPCT + 2
          JSPOT = JSPOT - 2
  435     CONTINUE
C
C           CHECK OF PRIMARY INPUTS TO GATE JDEX2 COMPLETE.
C           IF GATE INPUTS REMAIN, CONTINUE CHECK.  OTHERWISE,
C           CHECK NUMBER OF INPUTS TO GATE AND PROCEED.
        IF (JNGATE .EQ. 0) GO TO 447
  436   CONTINUE
C
C         GATE JDEX2 HAS 'JNGATE' GATE INPUTS TO BE CHECKED
        JSPCT = NDEX2 + 3
        DO 446 J = 1,JNGATE
          JSPOT  = JSPOT + 2
          JDEX3  = GATE(JSPOT)
          NDEX3  = IGATE(JDEX3)
          KPRIME = GATE(NDEX3 + 2)
          IF (KPRIME .EQ. 0) GO TO 446
C
C           GATE JDEX3 HAS KPRIME PRIMARY INPUTS.
C           CHECK FOR PRESET OR CONTRADICTORY CONDITIONS.
          KSPOT = NDEX3 + 3 + 2*GATE(NDEX3 + 1)
          DO 439 K = 1,KPRIME
            KSPOT = KSPOT + 2
            KNODE = GATE(KSPOT)
```

```
            JMODE = X(1,KNODE)
            IF (JMODE .EQ. -1) GO TO 439
C
C             NODE 'KNODE' HAS BEEN PREVIOUSLY SET.
            KMODE = GATE(KSPOT + 1)
            IF (IPRINT .GT. 0)WRITE(6,1016) JDEX3,KNODE,JMODE,X(2,KNODE)
            IF ((GATE(NDEX3).EQ.1  .AND.  KMODE.EQ.JMODE)  .OR.
     1           (GATE(NDEX3).EQ.2  .AND.  KMODE.NE.JMODE))  GO TO 437
            IF (IPRINT .GT. 0) WRITE (6,1006) JDEX3
            IF (GATE(NDEX3).EQ.1 .AND. KMODE.NE.JMODE) GO TO 440
C
C             GATES JDEX3 AND JDEX2 MUST BE DELETED.
            GATE(NDEX3)  = -99
            JGATE(JDEX3) = -99
            IF (IPRINT .GT. 0) WRITE (6,1007) JDEX3,KIND(2),JDEX2,JDEX2
            GO TO 430
  437       CONTINUE
C
C             DELETE PRIMARY INPUT TO GATE KDEX3
            IF (IPRINT .GT. 0) WRITE (6,1023) K,JDEX3
            GATE(NDEX3 + 2) = GATE(NDEX3 + 2) - 1
            IF (K .EQ. KPRIME)  GO TO 439
            KPM1   = KPRIME - 1
            KKSPOT = KSPOT
            DO 438 K2 = K,KPM1
              GATE(KKSPOT)   = GATE(KKSPOT + 2)
              GATE(KKSPOT+1) = GATE(KKSPOT + 3)
  438         KKSPOT = KKSPOT + 2
            KSPOT = KSPOT - 2
  439       CONTINUE
C
C             CHECK OF PRIMARY INPUTS TO GATE JDEX3 COMPLETED.
C               CHECK NUMBER OF INPUTS REMAINING.
          KLDEX = GATE(NDEX3 + 1) + GATE(NDEX3 + 2)
          II    = NDEX3 + 4 + 2*KLDEX
          IF (IPRINT .GT. 1) WRITE (6,1020) JDEX3,(GATE(II),II=NDEX3,LL)
          IF (KLDEX .GT. 1) GO TO 446
          IF (KLDEX .EQ. 1) GO TO 442
          IF (IPRINT .GT. 0) WRITE (6,1024) JDEX3,JDEX3
          IF (GATE(NDEX3) .EQ. 2) GO TO 440
C
C           DELETE 'AND' GATE JDEX3 AND 'OR' GATE JDEX2
          IF (IPRINT .GT. 0) WRITE (6,1007) JDEX3,KIND(2),JDEX2,JDEX2
          GATE(NDEX3)  = -99
          JGATE(JDEX3) = -99
          GO TO 430
  440     CONTINUE
C
C           DELETE GATE JDEX3 AND INPUT TO JDEX2.
          IF (IPRINT .GT. 0) WRITE (6,1008) JDEX3,KIND(2),JDEX2,J,JLDEX
          GATE(NDEX3)  = -99
          JGATE(JDEX3) = -99
```

```
          GATE(NDEX2 + 1) = GATE(NDEX2 + 1) - 1
          JM1 = JNGATE + GATE(NDEX2 + 2) - 1
          IF (J .GT. JM1) GO TO 446
          JJSPCT = JSPCT
          DO 441 K2 = J,JM1
            GATE(JJSPCT)   = GATE(JJSPCT + 2)
            GATE(JJSPCT+1) = GATE(JJSPCT + 3)
  441       JJSPCT = JJSPCT + 2
          JSPCT = JSPCT - 2
          GO TO 446
  442     CONTINUE
C
C           GATE JDEX3 HAS SINGLE INPUT.  DELETE JDEX3
C           AND INSERT DIRECTLY INTO JDEX2.
          CALL REDUCE (1,JDEX3,NDEX3,JDEX2,NDEX2,IPRINT,J,JNGATE,JSPCT,
     1                 GATE,JGATE,NGSIZE,MGATE)
  446     CONTINUE
C*************************************************************************
C                                                                        *
C           GATE JDEX2 CHECK COMPLETED (FROM 446 OR TRANSFER FROM 435).*
C           IF NO INPUTS REMAIN, DELETE GATE JDEX.                      *
C           IF ONE INPUT REMAINS, INSERT DIRECTLY INTO JDEX.            *
C                                                                        *
C*************************************************************************
  447   JLDEX = GATE(NDEX2 + 1) + GATE(NDEX2 + 2)
        JJ    = NDEX2 + 4 + 2*JLDEX
        IF (IPRINT .GT. 1) WRITE (6,1020) JDEX2,(GATE(II),II=NDEX2,JJ)
        IF (JLDEX .GT. 1) GO TO 454
        IF (JLDEX .EQ. 1) GO TO 450
C
C         GATE JDEX2 HAS NO INPUTS (FROM 447 OR TRANSFER FROM 424).
C         DELETE GATE JDEX AND RETURN TO PREVIOUS GATE.
        IF (IPRINT .LT. 1) GO TO 449
          WRITE (6,1024) JDEX2,JDEX2
  448     WRITE (6,1007) JDEX2,KIND(1),JDEX,JDEX
  449   INDEX = JDEX
        JDEX  = JGATE(INDEX)
        GO TO 213
  450   CONTINUE
C
C         GATE JDEX2 HAS SINGLE INPUT (TRANSFER FROM 429 OR 447).
C         DELETE JDEX2 AND INSERT DIRECTLY INTO JDEX.
        IKIND = GATE(NDEX2)
        CALL REDUCE (1,JDEX2,NDEX2,JDEX,NDEX,IPRINT,I,INGATE,ISPCT,
     1               GATE,JGATE,NGSIZE,MGATE)
 4100   CONTINUE
C
C         IF GATE JDEX2 WAS 'OR' GATE, SET PRIMARY INPUT FROM GATE,
C         OR FROM LOWER 'AND' GATES INTO ARRAY 'X'.
        IF (IKIND .EQ. 1) GO TO 454
        IF (GATE(NDEX2 + 1) .GT. 0) GO TO 4101
C
```

```
C         JDEX2 HAD PRIMARY INPUT.  SET INTO ARRAY 'X'.
          KNODE = GATE(NDEX2 + 5)
          X(1,KNODE) = GATE(NDEX2 + 6)
          X(2,KNODE) = JDEX
          GO TO 454
 4101     CONTINUE
C
C         SEARCH FOR LOWER LEVEL 'AND' GATES AND SET PRIMARY INPUTS.
          JDEX2 = GATE(NDEX2 + 5)
          NDEX2 = IGATE(JDEX2)
          IF (GATE(NDEX2) .EQ. 2) GO TO 454
C
C         FIRST SET PRIMARY INPUTS.
          JPRIME = GATE(NDEX2 + 2)
          JNGATE = GATE(NDEX2 + 1)
          IF (JPRIME .EQ. 0) GO TO 4103
          JSPOT = NDEX2 + 3 + 2*JNGATE
          DO 4102 J = 1,JPRIME
            JSPOT = JSPOT + 2
            KNODE = GATE(JSPOT)
            JMODE = X(1,KNODE)
            IF (JMODE .EQ. -1) GO TO 4110
C
C             NODE HAS BEEN PREVIOUSLY SET. CHECK FOR AGREEMENT.
            KMODE = GATE(JSPOT + 1)
            IF (KMODE .EQ. JMODE) GO TO 4102
C
C             PRIMARY INPUT CONTRADICTORY. DELETE JDEX2 AND JDEX.
            GATE(NDEX2)  = -99
            JGATE(JDEX2) = -99
            IF (IPRINT .LT. 1) GO TO 449
            WRITE (6,1016) JDEX2,KNODE,JMODE,X(2,KNODE)
            WRITE (6,1006) JDEX2
            GO TO 448
 4110       CONTINUE
C
C             NODE HAS NOT BEEN PREVIOUSLY SET. SET NODE.
            X(1,KNODE) = GATE(JSPOT + 1)
            X(2,KNODE) = JDEX2
 4102       CONTINUE
 4103     IF (JNGATE .EQ. 0) GO TO 454
C
C         NOW CHECK AND SET 'AND' GATE INPUTS TO GATE JDEX2.
          JSPOT = NDEX2 + 3
          DO 4106 J = 1,JNGATE
            JSPOT = JSPOT + 2
            JDEX3 = GATE(JSPOT)
            NDEX3 = IGATE(JDEX3)
            IF (GATE(NDEX3) .EQ. 2) GO TO 4106
            KPRIME = GATE(NDEX3 + 2)
            IF (KPRIME .EQ. 0) GO TO 4106
            KSPOT = NDEX3 + 3 + 2*GATE(NDEX3 + 1)
```

```
          DO 4105 K = 1,KPRIME
            KSPOT = KSPOT + 2
            KNODE = GATE(KSPOT)
            JNODE = X(1,KNODE)
            IF (JNODE .EQ. -1) GO TO 4104
C
C             NODE HAS BEEN PREVIOUSLY SET.  CHECK FOR AGREEMENT.
            KNODE = GATE(KSPOT + 1)
            IF (KNODE .EQ. JNODE) GO TO 4105
C
C             PRIMARY INPUT CONTRADICTORY, THUS CANNOT OCCUR.
C             DELETE GATES JDEX3, JDEX2 AND JDEX.
            GATE(NDEX3)   = -99
            JGATE(JDEX3) = -99
            IF (IPRINT .LT. 1) GO TO 449
            WRITE (6,1016) JDEX3,KNODE,JNODE,X(2,KNODE)
            WRITE (6,1006) JDEX3
            WRITE (6,1007) JDEX3,KIND(1),JDEX2,JDEX2
            GO TO 448
 4104       CONTINUE
C
C             NODE HAS NOT PREVIOUSLY BEEN SET.  SET NODE.
            X(1,KNODE) = GATE(KSPOT + 1)
            X(2,KNODE) = JDEX3
 4105       CONTINUE
 4106     CONTINUE
  454   CONTINUE
C*************************************************************************
C                                                                        *
C          EDIT OF 'AND' GATE JDEX COMPLETE  (LOOP FROM 420).            *
C          CHECK FOR NUMBER OF INPUTS REMAINING.                         *
C          IF NO INPUTS REMAIN, DELETE GATE AND RETURN TO 331.           *
C                                                                        *
C*************************************************************************
      LLDEX = GATE(NDEX + 1) + GATE(NDEX + 2)
      JJ    = NDEX + 4 + 2*LLDEX
      IF (IPRINT .GT. 3) WRITE (6,1026) GATE
      IF (IPRINT .GT. 1) WRITE (6,1020) JDEX,(GATE(II),II=NDEX,JJ)
      IF (LLDEX .GT. 1) GO TO 482
      IF (LLDEX .EQ. 1) GO TO 485
C
C       GATE JDEX HAS NO INPUTS AND THUS IS "SURE TO OCCUR."
C       DELETE GATE AND RETURN TO PREVIOUS GATE. (THIS SHOULDN'T OCCUR)
      IF (IPRINT .GT. 0) WRITE (6,1024) JDEX,JDEX
      INDEX = JDEX
      JDEX = JGATE(INDEX)
      GO TO 331
  455 CONTINUE
C*************************************************************************
C                                                                        *
C       'OR' GATE POST GATE EDIT BEGINS.                                 *
C         CHECK FOR REDUNDANT PRIMARY EVENTS BENEATH 'OR' GATES.         *
```

```
C          FIRST RESET NODES, THEN CHECK PRIMARY INPUTS.                *
C                                                                       *
C***********************************************************************
      DO 456 I = 1,NODES
        IF (X(2,I) .LT. JDEX) GO TO 456
        X(1,I) = -1
        X(2,I) = -1
  456   CONTINUE
      IPRIME = GATE(NDEX + 2)
      IF (IPRIME .EQ. 0) GO TO 457
      ISPOT = NDEX + 5 + 2*INGATE
      JNODE = GATE(ISPOT)
      X(1,JNODE) = GATE(ISPOT + 1)
      X(2,JNODE) = -2
      IF (IPRIME .EQ. 1) GO TO 457
C
C       INITIAL NODE SET.  NOW CHECK PRIMARY INPUTS TO GATE JDEX.
      CALL XCHECK (2,JDEX,NDEX,IPRINT,X,GATE,NODES,NGSIZE)
      IF (INGATE .EQ. 0) GO TO 479
  457 CONTINUE
C
C       NOW CHECK 'OR' GATE INPUTS TO GATE JDEX.
      ISPOT = NDEX + 3
      DO 458 I = 1,INGATE
        ISPOT  = ISPOT + 2
        JDEX2  = GATE(ISPOT)
        NDEX2  = IGATE(JDEX2)
        JPRIME = GATE(NDEX2 + 2)
        IF (GATE(NDEX2) .EQ. 1  .OR.  JPRIME .EQ. 0)  GO TO 458
C
C         'OR' GATE WITH PRIMARY INPUTS FOUND.  CHECK INPUTS.
        CALL XCHECK (1,JDEX2,NDEX2,IPRINT,X,GATE,NODES,NGSIZE)
  458   CONTINUE
C
C         NOW CHECK THIRD LEVEL 'OR' GATES FOR INPUTS.
      ISPOT = NDEX + 3
      DO 461 I = 1,INGATE
        ISPOT  = ISPOT + 2
 4580   JDEX2  = GATE(ISPOT)
        NDEX2  = IGATE(JDEX2)
        JNGATE = GATE(NDEX2 + 1)
        IF (GATE(NDEX2).EQ.1  .OR.  JNGATE.EQ.0) GO TO 461
        JSPOT = NDEX2 + 3
        DO 460 J = 1,JNGATE
          JSPOT = JSPOT + 2
  459     JDEX3  = GATE(JSPOT)
          NDEX3  = IGATE(JDEX3)
          KPRIME = GATE(NDEX3 + 2)
          IF (GATE(NDEX3).EQ.1  .OR.  KPRIME.EQ.0)  GO TO 460
C
C           THIRD LEVEL 'OR' GATE HAS PRIMARY INPUTS TO BE CHECKED.
          CALL XCHECK (1,JDEX3,NDEX3,IPRINT,X,GATE,NODES,NGSIZE)
```

```
          KLDEX = GATE(NDEX3 + 1) + GATE(NDEX3 + 2)
          KK    = NDEX3 + 4 + 2*KLDEX
          IF (IPRINT .GT. 1) WRITE(6,1020) JDEX3,(GATE(II),II=NDEX3,KK)
          IF (KLDEX .GT. 1) GO TO 460
C
C           GATE JDEX3 HAS ZERO OR ONE INPUT. REDUCE INPUT TO JDEX3.
          CALL REDUCE (KLDEX,JDEX3,NDEX3,JDEX2,NDEX2,IPRINT,J,JNGATE,
     1                 JSPOT,GATE,JGATE,NGSIZE,MGATE)
          IF (KLDEX.EQ.1  .AND.  GATE(NDEX3+1).EQ.1) GO TO 459
  460     CONTINUE
        JLDEX = GATE(NDEX2+1) + GATE(NDEX2 + 2)
        IF (JLDEX .GT. 1) GO TO 461
C
C         GATE JDEX2 HAS ZERO OR ONE INPUT. REDUCE INPUT TO JDEX.
        CALL REDUCE (JLDEX,JDEX2,NDEX2,JDEX,NDEX,IPRINT,I,INGATE,
     1               ISPOT,GATE,JGATE,NGSIZE,MGATE)
       IF (GATE(NDEX2+1) .EQ. 1) GO TO 4580
  461   CONTINUE
C
C         NOW CHECK SECOND AND THIRD LEVEL 'AND' GATES.
      ISPOT = NDEX + 3
      DO 478 I = 1,INGATE
        ISPOT  = ISPOT + 2
  462   JDEX2  = GATE(ISPOT)
        NDEX2  = IGATE(JDEX2)
        JNGATE = GATE(NDEX2 + 1)
        IF (GATE(NDEX2) .EQ. 1) GO TO 468
C
C         GATE JDEX2 IS 'OR'. CHECK FOR THIRD LEVEL 'AND' GATES.
       IF (JNGATE .EQ. 0) GO TO 467
        JSPOT = NDEX2 + 3
        DO 466 J = 1,JNGATE
          JSPOT  = JSPOT + 2
          JDEX3  = GATE(JSPOT)
          NDEX3  = IGATE(JDEX3)
          KPRIME = GATE(NDEX3 + 2)
          IF (GATE(NDEX3) .EQ. 2  .OR.  KPRIME .EQ. 0) GO TO 466
C
C           THIRD LEVEL 'AND' GATE HAS PRIMARY INPUTS.
          ISTART = -1
          CALL XCHECK (ISTART,JDEX3,NDEX3,IPRINT,X,GATE,NODES,NGSIZE)
          IF (ISTART .NE. -1) GO TO 466
C
C           GATE JDEX3 HAS PRESET INPUT AND WILL BE DELETED.
          GATE(NDEX2 + 1) = GATE(NDEX2 + 1) - 1
          GATE(NDEX3)  = -99
          JGATE(JDEX3) = -99
          IF (IPRINT .GT. 0) WRITE (6,1008) JDEX3,KIND(2),JDEX2,J,JNGATE
          JJSPOT = JSPOT
          IF (J .GE. JNGATE) GO TO 464
          JM1 = JNGATE - 1
          DO 463 J2 = J,JM1
```

```
            GATE(JJSPCT)    = GATE(JJSPCT + 2)
            GATE(JJSPCT+1) = GATE(JJSPCT + 3)
  463       JJSPCT = JJSPCT + 2
  464     JPRIME = GATE(NDEX2 + 2)
          IF (JPRIME .EQ. 0) GO TO 465
          JSPOT2 = JJSPCT + 2*JPRIME
          GATE(JJSPCT)    = GATE(JSPOT2)
          GATE(JJSPOT+1) = GATE(JSPOT2 + 1)
          IF (GATE(JSPOT2) .LT. 0) GATE(JSPOT2) = -JJSPOT
  465     JSPOT = JSPCT - 2
          JJ = NDEX2 + 4 + 2*(GATE(NDEX2+1) + JPRIME)
          IF (IPRINT .GT. 1) WRITE (6,1020) JDEX2,(GATE(II),II=NDEX2,JJ)
  466     CONTINUE
  467   JLDEX = GATE(NDEX2+1) + GATE(NDEX2 + 2)
        IF (JLDEX .GT. 1) GO TO 478
C
C         GATE JDEX2 HAS ZERO OR ONE INPUT. REDUCE INPUT TO JDEX.
        CALL REDUCE (JLDEX,JDEX2,NDEX2,JDEX,NDEX,IPRINT,I,INGATE,
     1               ISPOT,GATE,JGATE,NGSIZE,MGATE)
        IF (JLDEX .EQ. 1  .AND.  GATE(NDEX2+1) .EQ. 1) GO TO 462
        GO TO 478
  468   CONTINUE
C
C         CHECK SECOND LEVEL 'AND' GATES.
        JPRIME = GATE(NDEX2 + 2)
        IF (JPRIME .EQ. 0) GO TO 472
C
C         CHECK PRIMARY INPUTS TO GATE JDEX2.
        ISTART = -1
        CALL XCHECK (ISTART,JDEX2,NDEX2,IPRINT,X,GATE,NODES,NGSIZE)
        IF (ISTART .NE. -1) GO TO 472
C
C         GATE JDEX2 HAS PRESET INPUT AND WILL BE ELIMINATED.
        GATE(NDEX+1) = GATE(NDEX+1) - 1
        GATE(NDEX2)  = -99
        JGATE(JDEX2) = -99
        IF (IPRINT .GT. 0) WRITE (6,1008) JDEX2,KIND(2),JDEX,I,INGATE
        IISPOT = ISPCT
        IF (I .GE. INGATE) GO TO 470
        JM1 = INGATE - 1
        DO 469 I2 = I,JM1
          GATE(IISPCT)    = GATE(IISPOT + 2)
          GATE(IISPOT+1) = GATE(IISPOT + 3)
  469     IISPCT = IISPOT + 2
  470   IPRIME = GATE(NDEX + 2)
        IF (IPRIME .EQ. 0) GO TO 471
        ISPOT2 = IISPOT + 2*IPRIME
        GATE(IISPCT)    = GATE(ISPCT2)
        GATE(IISPOT+1) = GATE(ISPOT2+1)
        IF (GATE(ISPOT2) .LT. 0) GATE(ISPOT2) = -IISPOT
  471   ISPOT = ISPOT - 2
        JJ = NDEX + 4 + 2*(GATE(NDEX+1) + IPRIME)
```

```
      IF (IPRINT .GT. 1) WRITE (6,1020) JDEX,(GATE(II),II=NDEX,JJ)
      GO TO 478
  472 CONTINUE
C
C       NOW CHECK THIRD LEVEL GATES.
      IF (JNGATE .EQ. 0) GO TO 478
      JSPOT = NDEX2 + 3
      DO 477 J = 1,JNGATE
        JSPOT  = JSPOT + 2
  473   JDEX3  = GATE(JSPOT)
        NDEX3  = IGATE(JDEX3)
        KPRIME = GATE(NDEX3 + 2)
        IF (KPRIME .EQ. 0) GO TO 477
C
C         THIRD LEVEL GATE HAS PRIMARY INPUTS TO BE CHECKED.
        ISTART = 0
        IF (GATE(NDEX3) .EQ. 1) ISTART = -1
        CALL XCHECK (ISTART,JDEX3,NDEX3,IPRINT,X,GATE,NODES,NGSIZE)
        IF (GATE(NDEX3) .EQ. 1) GO TO 475
C
C         THIRD LEVEL GATE IS 'OR' GATE AND IS FINISHED.
C         CHECK FOR NUMBER OF INPUTS.
        KLDEX = GATE (NDEX3 + 1) + GATE(NDEX3 + 2)
        KK    = NDEX3 + 4 + 2*KLDEX
        IF (IPRINT .GT. 1) WRITE (6,1020) JDEX3,(GATE(II),II=NDEX3,KK)
        IF (KLDEX .GT. 1) GO TO 477
C
C         GATE JDEX3 HAS ZERO OR ONE INPUT. REDUCE INPUT TO JDEX2.
        IF (KLDEX .EQ. 1) GO TO 474
        IF (IPRINT .GT. 0) WRITE (6,1024) JDEX3,JDEX3
        GO TO 476
  474   CALL REDUCE (1,JDEX3,NDEX3,JDEX2,NDEX2,IPRINT,J,JNGATE,
     1               JSPOT,GATE,JGATE,NGSIZE,NGATE)
        IF (GATE(NDEX3+1) .GT. 0) GO TO 473
        GO TO 477
  475   IF (ISTART .NE. -1) GO TO 477
C
C         GATES JDEX3 AND JDEX2 WILL BE DELETED.
  476   GATE(NDEX3)  = -99
        JGATE(JDEX3) = -99
        CALL REDUCE (0,JDEX2,NDEX2,JDEX,NDEX,0,I,INGATE,ISPOT,
     1               GATE,JGATE,NGSIZE,NGATE)
        IF (IPRINT .LE. 0) GO TO 478
        WRITE (6,1007) JDEX3,KIND(1),JDEX2,JDEX2
        WRITE (6,1008) JDEX2,KIND(2),JDEX,I,INGATE
        JJ = NDEX + 4 + 2*(GATE(NDEX+1) + GATE(NDEX+2))
        IF (IPRINT .GT. 1) WRITE (6,1020) JDEX,(GATE(II),II=NDEX,JJ)
        GO TO 478
  477   CONTINUE
  478 CONTINUE
C*************************************************************************
C                                                                        *
```

```
C          'OR' GATE EDIT COMPLETE (CONTINUATION, OR TRANSFER FROM 456).*
C          FIRST RESET NODES AND GATE ENTRIES SET BY GATE EDIT.         *
C          THEN CHECK NUMBER OF INPUTS TO GATE JDEX AND PROCEED.        *
C                                                                       *
C***********************************************************************
  479 CONTINUE
      LLDEX = GATE(NDEX + 1) + GATE(NDEX + 2)
      JJ    = NDEX + 4 + 2*LLDEX
      IF (IPRINT .GT. 1) WRITE (6,1020) JDEX,(GATE(II),II=NDEX,JJ)
      IF (IPRINT .GT. 2) WRITE (6,1027) JDEX,X
      IF (IPRINT .GT. 2) WRITE (6,1025) JDEX,GATE
      DO 481 I = 1,NODES
        ISPOT = -X(2,I)
        IF (ISPOT .LE. 1) GO TO 481
        X(1,I) = -1
        X(2,I) = -1
        IF (ISPOT .EQ. 2) GO TO 481
C
C       CHAIN OF NODES HAS BEEN SET, STARTING AT LOCATION ISPOT.
  480   IISPOT = -GATE(ISPOT)
        GATE(ISPOT) = I
        ISPOT = IISPOT
        IF (ISPOT .GT. 0) GO TO 480
  481   CONTINUE
      IF (IPRINT .GT. 2) WRITE (6,1026) GATE
      IF (IPRINT .GT. 1) WRITE (6,1020) JDEX,(GATE(II),II=NDEX,JJ)
      IF (LLDEX .GT. 1) GO TO 482
      IF (LLDEX .EQ. 1) GO TO 485
C
C       GATE 'JDEX' HAS NO INPUTS; CANNOT OCCUR (SHOULDN'T OCCUR).
      IF (IPRINT .GT. 0) WRITE (6,1024) JDEX,JDEX
      INDEX = JDEX
      JDEX = JGATE(INDEX)
      GO TO 213
  482 CONTINUE
C***********************************************************************
C                                                                       *
C       'AND'/'OR' EDIT COMPLETE (FROM 420/454/481/487/488/489/490).    *
C       BACKTRACK TO PREVIOUS GATE AND BEGIN NEW BRANCH.                *
C                                                                       *
C***********************************************************************
      IF (IPRINT .GT. 0) WRITE (6,1021) JDEX
      JDEX  = JGATE(JDEX)
      IF (JDEX .EQ. 0) GO TO 500
      NDEX  = IGATE(JDEX)
      IKIND = -GATE(NDEX)
C
C       NOW FIND NEXT BRANCH OF PREVIOUS GATE; OR, IF FINISHED,
C       SEND PREVIOUS GATE TO BE EDITED.
      KDEX  = NDEX + 3
      LLDEX = GATE(NDEX + 1) + GATE(NDEX + 2)
      DO 483 I = 1,LLDEX
```

```
        KDEX = KDEX + 2
        IF (GATE(KDEX) .LT. 0) GO TO 484
  483   CONTINUE
C
C         NO UNDEVELOPED BRANCHES REMAIN. GATE 'JDEX' FINISHED.
      GO TO 410
  484 LDEX = (KDEX - NDEX - 3)/2
      GO TO 401
  485 CONTINUE
C*************************************************************************
C                                                                        *
C       SINGLE INPUT GATES TRANSFER TO THIS REGION (FROM 410/454/481)    *
C       GATE WILL BE ELIMINATED AND ITS INPUT INSERTED INTO              *
C       PRECEEDING GATE.  IN ADDITION, IF GATE IS AN 'OR' AND INPUTS     *
C       AN 'AND', RESET NODES FROM 'AND' GATES BELOW.                    *
C         JDEX  = CURRENT GATE                                           *
C         IDEX  = PREVIOUS GATE                                          *
C         INDEX = NEXT GATE TO BE GENERATED.                             *
C         JDEX2,JDEX3,JDEX4,JDEX5 = SUCCEEDING GATES                     *
C         NDEX  = IGATE(JDEX)                                            *
C                                                                        *
C*************************************************************************
      IDEX = JGATE(JDEX)
      IF (IDEX .EQ. 0) GO TO 510
      MDEX = IGATE(IDEX)
C
C       NOW SEARCH PREVIOUS GATE TO FIND CURRENT BRANCH.
      JSPOT = MDEX + 5
      KLDEX = GATE(MDEX + 1) + GATE(MDEX + 2)
      KSPOT = MDEX + 3 + 2*KLDEX
      DO 486 ISPOT = JSPOT,KSPOT,2
        IF (GATE(ISPOT).EQ.JDEX .AND. GATE(ISPOT+1).EQ.-1) GO TO 487
  486   CONTINUE
      IE = 486
      PRINT 1,IE
C*************************************************************************
C*************************************************************************
C
C       ISPOT HAS BEEN SET TO CURRENT LOCATION IN PREVIOUS GATE.
  487 IKIND = GATE(NDEX)
      KK = KSPOT + 1
      GATE(NDEX) = -99
      IF (IPRINT .GT. 0) WRITE (6,1019) JDEX,KIND(IKIND),IDEX
      IF (IKIND .EQ. 2) GO TO 489
C
C       CURRENT GATE IS 'AND'. INSERT INTO PREVIOUS GATE AND BACKTRACK.
      IF (GATE(NDEX + 1) .EQ. 0) GO TO 488
      GATE(ISPOT) = GATE(NDEX + 5)
      IISPOT = GATE(ISPOT)
      JGATE(IISPOT) = IDEX
      IF (IPRINT .GT. 1) WRITE (6,1020) IDEX,(GATE(II),II=MDEX,KK)
      GO TO 482
```

```
  488 CONTINUE
C
C       INPUT IS PRIMARY INPUT.
      GATE(MDEX + 1) = GATE(MDEX + 1) - 1
      GATE(MDEX + 2) = GATE(MDEX + 2) + 1
      GATE(ISPOT)    = GATE(NDEX + 5)
      GATE(ISPOT +1) = GATE(NDEX + 6)
      INDEX = JDEX + 1
      IGATE(INDEX) = IGATE(JDEX) + 1
      IF (IPRINT .GT. 1) WRITE (6,1020) IDEX,(GATE(II),II=MDEX,KK)
      GO TO 482
  489 CONTINUE
C
C       CURRENT GATE IS 'OR'. IF PRECEEDING GATE 'IDEX' IS 'AND',
C       SET NODES FROM SUCCEEDING 'AND' GATES.
      JKIND = -GATE(MDEX)
      IF (GATE(NDEX + 1) .EQ. 1) GO TO 490
      GATE(MDEX + 1) = GATE(MDEX + 1) - 1
      GATE(MDEX + 2) = GATE(MDEX + 2) + 1
      GATE(ISPOT)    = GATE(NDEX + 5)
      GATE(ISPOT+1)  = GATE(NDEX + 6)
      IF (IPRINT .GT. 1) WRITE (6,1020) IDEX,(GATE(II),II=MDEX,KK)
      IF (JKIND .EQ. 2) GO TO 482
      JNODE = GATE(ISPOT)
      IB = 489
      IF (X(2,JNODE) .GE. 0  .AND.  X(2,JNODE) .LT. JDEX) PRINT 1,IB
C*************************************************************************
C*************************************************************************
      X(1,JNODE) = GATE(ISPOT + 1)
      X(2,JNODE) = IDEX
      GO TO 482
  490 CONTINUE
C
C       INPUT IS GATE INPUT.  CHECK AND SET NODES IF 'AND'.
      GATE(ISPOT) = GATE(NDEX + 5)
      IISPOT = GATE(ISPOT)
      JGATE(IISPOT) = IDEX
      IF (IPRINT .GT. 1) WRITE (6,1020) IDEX,(GATE(II),II=MDEX,KK)
      IF (JKIND .EQ. 2) GO TO 482
      JDEX2 = GATE(ISPOT)
      NDEX2 = IGATE(JDEX2)
      IF (GATE(NDEX2) .EQ. 2) GO TO 482
      INGATE = GATE(NDEX2 + 1)
      IPRIME = GATE(NDEX2 + 2)
      IF (IPRIME .EQ. 0) GO TO 492
C
C       SET NODES FOR PRIMARY INPUTS.
      ISPOT = NDEX2 + 3 + 2*INGATE
      DO 491 I = 1,IPRIME
        ISPOT = ISPOT + 2
        JNODE = GATE(ISPOT)
        IB = 491
```

```
        IF (X(2,JNODE).GE.0 .AND. X(2,JNODE).LT.JDEX) PRINT 1,IB
C*******************************************************************
C*******************************************************************
        X(1,JNODE) = GATE(ISPOT + 1)
  491   X(2,JNODE) = JDEX2
      IF (INGATE .EQ. 0) GO TO 482
  492 ISPOT = NDEX2 + 3
C
C       NOW SEARCH SECOND LEVEL GATE JDEX2 FOR THIRD LEVEL 'AND' GATES.
      DO 4999 I = 1,INGATE
        ISPOT  = ISPOT + 2
        JDEX3  = GATE(ISPOT)
        NDEX3  = IGATE(JDEX3)
        IF (GATE(NDEX3) .EQ. 2) GO TO 4999
        JNGATE = GATE(NDEX3 + 1)
        JPRIME = GATE(NDEX3 + 2)
        IF (JPRIME .EQ. 0) GO TO 494
C
C         GATE 'JDEX3' HAS PRIMARY INPUTS.  SET NODES.
        JSPOT = NDEX3 + 3 + 2*JNGATE
        DO 493 J = 1,JPRIME
          JSPOT = JSPOT + 2
          JNODE = GATE(JSPOT)
          IB = 493
          IF (X(2,JNODE).GE.0 .AND. X(2,JNODE).LT.JDEX) PRINT 1,IB
C*******************************************************************
C*******************************************************************
          X(1,JNODE) = GATE(JSPOT + 1)
  493     X(2,JNODE) = JDEX3
        IF (JNGATE .EQ. 0) GO TO 4999
  494   JSPOT = NDEX3 + 3
C
C         NOW SEARCH THIRD LEVEL GATE JDEX3 FOR FOURTH LEVEL 'AND' GATES
        DO 499 J = 1,JNGATE
          JSPOT = JSPOT + 2
          JDEX4 = GATE(JSPOT)
          NDEX4 = IGATE(JDEX4)
          IF (GATE(NDEX4) .EQ. 2) GO TO 499
          KNGATE = GATE(NDEX4 + 1)
          KPRIME = GATE(NDEX4 + 2)
          IF (KPRIME .EQ. 0) GO TO 496
C
C           GATE 'JDEX4' HAS PRIMARY INPUTS.  SET NODES.
          KSPOT = NDEX4 + 3 + 2*KNGATE
          DO 495 K = 1,KPRIME
            KSPOT = KSPOT + 2
            JNODE = GATE(KSPOT)
            IB = 495
            IF (X(2,JNODE).GE.0 .AND. X(2,JNODE).LT.JDEX) PRINT 1,IB
C*******************************************************************
C*******************************************************************
            X(1,JNODE) = GATE(KSPOT + 1)
```

```
  495       X(2,JNCDE) = JDEX4
          IF (KNGATE .EQ. 0) GO TO 495
  496     KSPOT = NDEX4 + 3
C
C           NOW SEARCH FOURTH LEVEL GATE JDEX4 FOR 'AND' GATES.
          DO 498 K = 1,KNGATE
            KSPOT = KSPOT + 2
            JDEX5 = GATE(KSPOT)
            NDEX5 = IGATE(JDEX5)
            IF (GATE(NDEX5) .EQ. 2) GO TO 498
            LNGATE = GATE(NDEX5 + 1)
            LPRIME = GATE(NDEX5 + 2)
            IF (LPRIME .EQ. 0) GO TO 498
C
C             GATE 'JDEX5' HAS PRIMARY INPUTS. SET NODES.
            LSPOT = NDEX5 + 3 + 2*LNGATE
            DO 497 L = 1,LPRIME
              LSPOT = LSPOT + 2
              JNODE = GATE(LSPOT)
              IB = 497
              IF (X(2,JNODE).GE.0 .AND. X(2,JNODE).LT.JDEX) PRINT 1,IB
C*************************************************************************
C*************************************************************************
              X(1,JNCDE) = GATE(LSPOT + 1)
  497         X(2,JNCDE) = JDEX5
  498       CONTINUE
  499     CONTINUE
 4999   CONTINUE
      GO TO 482
  500 CONTINUE
C*************************************************************************
C                                                                        *
C       'TOP' GATE HAS BEEN COMPLETED.  TREE IS FINISHED.                *
C       BEGIN TO PRINT RESULTS AND DO FINAL EDITING, ETC.                *
C                                                                        *
C*************************************************************************
      WRITE (6,1028) TITLE
      DO 501 I = 1,MGATE
        IF (IGATE(I) .EQ. 0) GO TO 600
        NDEX = IGATE(I)
        IF (GATE(NDEX) .LE. 0) GO TO 501
        NGATE = I
        KK = NDEX + 4 + 2*(GATE(NDEX+1) + GATE(NDEX+2))
        WRITE (6,1020) I,(GATE(II),II=NDEX,KK)
  501   CONTINUE
      GO TO 600
  510 CONTINUE
C
C       TOP GATE HAS SINGLE INPUT (TRANSFER FROM 485).
      IF (GATE(NDEX+1) .EQ. 0) GO TO 500
C
C       TOP GATE HAS SUBTREE GATE INPUT.  SET IT AS TOP GATE.
```

```
      GATE(NDEX) = -99
      JDEX = GATE(NDEX + 5)
      JGATE(JDEX) = 0
      GO TO 500
  600 CONTINUE
  700 CONTINUE
C*********************************************************************
C                                                                     *
C       ...FINAL GATE REGION...                                       *
C       TRANSFERS ARE ELIMINATED AND EXTRA GATES DELETED.             *
C       GATES ARE RENUMBERED CONSECUTIVELY.                           *
C                                                                     *
C       BEGIN BY SEARCHING FOR TRANSFERS.                             *
C       FOR EACH GATE CHECKED, SET GATE(NDEX) = GATE(NDEX) + 10.      *
C                                                                     *
C*********************************************************************
      IF (IEDIT .EQ. 98  .OR.  IEDIT .GE. 100) GO TO 720
      IF (NGATE .LT. 4) GO TO 799
      NG2 = NGATE - 2
      NG3 = NGATE - 3
      DO 710 I1 = 1,NG3
        I = NGATE + 1 - I1
        NDEX = IGATE(I)
        IKIND = GATE(NDEX)
        IF (IKIND .LE. 0  .OR.  IKIND .GE. 3) GO TO 710
        INGATE = GATE(NDEX + 1)
        IPRIME = GATE(NDEX + 2)
C
C         VALID GATE 'I' FOUND.  NOW SEARCH FOR MATCHING GATES.
        DO 709 J1 = I1,NG2
          J = NGATE - J1
          NDEX2 = IGATE(J)
          JKIND = GATE(NDEX2)
          IF (JKIND .NE. IKIND) GO TO 709
          JNGATE = GATE(NDEX2 + 1)
          JPRIME = GATE(NDEX2 + 2)
          IF (JPRIME .NE. IPRIME  .OR.  JNGATE .NE. INGATE) GO TO 709
C
C           GATE MATCH.  SEARCH FOR INPUT MATCHES.
          IF (IPRIME .EQ. 0) GO TO 703
          K1 = NDEX  + 5 + 2*INGATE
          K2 = NDEX  + 3 + 2*(INGATE + IPRIME)
          L1 = NDEX2 + 5 + 2*JNGATE
          L2 = NDEX2 + 3 + 2*(JNGATE + JPRIME)
          DO 702 K = K1,K2,2
            KNODE = GATE(K)
            KMODE = GATE(K + 1)
            DO 701 L = L1,L2,2
              IF (GATE(L).EQ.KNODE .AND. GATE(L+1).EQ.KMODE) GO TO 702
  701         CONTINUE
            GO TO 709
  702       CONTINUE
```

```
C
C             COMPLETE MATCH OF PRIMARY INPUTS FOUND.
C             NOW SEARCH FOR MATCHING GATE INPUTS.
            IF (INGATE .EQ. 0) GO TO 706
  703       K1 = NDEX  + 5
            K2 = NDEX  + 3 + 2*INGATE
            I1 = NDEX2 + 5
            L2 = NDEX2 + 3 + 2*JNGATE
            DO 705 K = K1,K2,2
              KGATE = GATE(K)
              DO 704 L = L1,L2,2
                IF (GATE(L).EQ.KGATE) GO TO 705
  704           CONTINUE
              GO TO 709
  705         CONTINUE
C
C               COMPLETE MATCH FOUND.  REMOVE GATE 'J'.
C               SEARCH PRECEEDING GATE AND REPLACE INPUT.
  706       GATE(NDEX2) = -1
            JDEX = JGATE(J)
            IDEX = IGATE(JDEX)
            K1   = IDEX + 5
            K2   = IDEX + 3 + 2*GATE(IDEX + 1)
            DO 707 K = K1,K2,2
              IF (GATE(K) .EQ. J  .AND.  GATE(K+1) .LT. 0) GO TO 708
  707         CONTINUE
            IE = 707
            PRINT 1,IE
C*******************************************************************************
C*******************************************************************************
  708       CONTINUE
C
C             REPLACE DUPLICATE GATE 'J' IN GATE JDEX WITH GATE 'I'.
            GATE(K) = I
  709       CONTINUE
C
C             GATE 'J' SEARCH COMPLETED.  SET GATE 'I' TO FINISHED.
          GATE(NDEX) = GATE(NDEX) + 10
  710     CONTINUE
C
C           GATE CHECK COMPLETED.  NOW RESET GATE TYPES.
        DO 711 I = 1,NGATE
          NDEX = IGATE(I)
  711     IF (GATE(NDEX) .GT. 3) GATE(NDEX) = GATE(NDEX) - 10
        IF (IPRINT .LT. 2) GO TO 720
C
C         WRITE GATE PRINTOUT.
        WRITE (6,1029) TITLE
        DO 712 I = 1,NGATE
          NDEX = IGATE(I)
          IF (GATE(NDEX) .LE. 0) GO TO 712
          KK = NDEX + 4 + 2*(GATE(NDEX+1) + GATE(NDEX+2))
```

```
        WRITE (6,1020) I,(GATE(II),II=NDEX,KK)
  712   CONTINUE
  720 CONTINUE
C
C         NOW RENUMBER GATES CONSECUTIVELY.  SET JGATE(I) = NEW INDEX.
      WRITE (6,1030) TITLE
      INDEX = 1
      DO 721 I = 1,NGATE
        NDEX = IGATE(I)
        IF (GATE(NDEX) .LE. 0) GO TO 721
        JGATE(I) = INDEX
        INDEX = INDEX + 1
  721   CONTINUE
C
C         NOW RENUMBER GATE INPUTS.
      DO 724 I = 1,NGATE
        NDEX = IGATE(I)
        IF (GATE(NDEX) .LE. 0) GO TO 724
        INGATE = GATE(NDEX + 1)
        IF (INGATE .EQ. 0) GO TO 723
        J1 = NDEX + 5
        J2 = NDEX + 3 + 2*INGATE
        DO 722 J = J1,J2,2
          JDEX = GATE(J)
  722     GATE(J) = JGATE(JDEX)
  723   KK = NDEX + 4 + 2*(INGATE + GATE(NDEX+2))
        WRITE (6,1020) JGATE(I),(GATE(II),II=NDEX,KK)
  724   CONTINUE
  799 RETURN
  900 CONTINUE
C
C       FAULT TREE TOO LARGE.  WRITE OUTPUT AND PARTIAL TREE.
      WRITE (6,1031) MGATE
      GO TO 930
  920 CONTINUE
      WRITE (6,1032) NGSIZE
  930 INDEX = INDEX - 2
      WRITE (6,1030) TITLE
      DO 931 I = 1,INDEX
        NDEX = IGATE(I)
        IF (GATE(NDEX) .LT. -2) GO TO 931
        KK = IGATE(I + 1) - 1
        WRITE (6,1020) I,(GATE(II),II=NDEX,KK)
  931   CONTINUE
      I = INDEX + 1
      NDEX = IGATE(I)
      KK = NDEX + 4 + 2*(GATE(NDEX+1) + GATE(NDEX+2))
      WRITE (6,1020) I,(GATE(II),II=NDEX,KK)
      IERR = 1
      RETURN
 2020 WRITE (6,1033)
      IERR = 1
```

```
      RETURN
 2030 IE = 2030
      WRITE (6,1) IB
      IERR = 1
      RETURN
 2040 WRITE (6,1034)
      IERR = 1
      RETURN
 2060 WRITE (6,1035)
      IERR = 1
      RETURN
C*****************************************************************************
C                                                                             *
C       FORMAT REGION                                                         *
C                                                                             *
C*****************************************************************************
    1 FORMAT (1H0,65('**')/' * ERROR NUMBER',I5,T131,'*'/1X,65('**')/)
 1000 FORMAT (1H1,65('--')/' -',T131,'-'/' - ',20A4,T131,'-'/' -',T131,
     1        '-'/' - CAT  GATE PRINTOUT SECTION.', T131,'-'/' -',T131,
     2        '-'/1X,65('--')//1X,'*** PRELIMINARY GATE PRINTOUT ***'//)
 1001 FORMAT (1H0,'GATE',I4,3X,'TYPE = ',A3/12X,'NUMBER OF GATES INPUT =
     1',I2,' NUMBER OF PRIMARY INPUTS =',I2/12X,'EVENT:  EVALUATE ROW',
     2  I6,' OF COMPONENT',I5/12X,'INPUTS: ',7('(',I6,',',I6,'),')/
     3  (20X,7('(',I6,',',I6,'),')))
 1002 FORMAT (1H0,'GATE',I4,3X,'TYPE = ',A3/12X,'NUMBER OF GATES INPUT =
     1',I2,' NUMBER OF PRIMARY INPUTS =',I2/12X,'EVENT:  SIGNAL =',I6,
     2  ' AT NODE',I6        /12X,'INPUTS: ',7('(',I6,',',I6,'),')/
     3  (20X,7('(',I6,',',I6,'),')))
 1003 FORMAT ('0GATE   1',  3X,'TYPE = ',A3/12X,'NUMBER OF GATES INPUT =
     1',I2,' NUMBER OF PRIMARY INPUTS =',I2/12X,'EVENT:  TOP EVENT'/
     2                        12X,'INPUTS: ',7('(',I6,',',I6,'),')/
     3  (20X,7('(',I6,',',I6,'),')))
 1004 FORMAT (1H0,'**  NO OUTPUT MODE =',I5,' EXISTS FOR NODE',I5,
     1       ' (COMPONENT ''',A8,''' TYPE',I6,', OUTPUT NO.',I2,').')
 1005 FORMAT (1H0,'**  NO COMPLETE ROW MATCH FOUND FOR COMPONENT ''',
     1       A8,''' TYPE',I6,','/5X,'DUE TO CONTRADICTING INITIAL OR SOU
     2NDARY CONDITIONS.')
 1006 FORMAT (4X,'*  GATE',I4,'  BEING ELIMINATED AND PRECEEDING GATES E
     1DITED.')
 1007 FORMAT (4X,'*  GATE',I4,' INPUTS ''',A3,''' GATE',I4,';  GATE',I4,
     1       ' BEING ELIMINATED.')
 1008 FORMAT (7X,'GATE',I4,' INPUTS ''',A3,''' GATE',I4,';  BRANCH',I3,
     1       ' OF',I3,' BRANCHES BEING ELIMINATED.')
 1009 FORMAT (4X,'*  GATE',I4,' INPUTS SINGLE INPUT ''',A3,''' GATE',
     1    I4,';  GATE',I4,' BEING ELIMINATED.')
 1010 FORMAT (/5X,'GATE',I4,': COMPONENT ''',A8,''' TYPE',I6,', ROW',I3,
     1       ' NODE',    I6,': MODE =',I5,' CONTRADICTS MODE =',I5,
     2       ', SET BY GATE',I4,'.')
 1011 FORMAT (/5X,'GATE',I4,': COMPONENT ''',A8,''' TYPE',I6,', ROW',I3,
     1       ' NODE',    I6,': MODE =',I5,' CONTRADICTS MODE =',I5,
     2       ', SET BY INITIAL CONDITIONS.')
 1012 FORMAT (/5X,'GATE',I4,': COMPONENT ''',A8,''' TYPE',I6,', ROW',I3,
```

```
     1       ' INTERNAL',I6,': MODE =',I5,' CONTRADICTS MODE =',I5,
     2       ', SET BY GATE',I4,'.')
1013 FORMAT (/5X,'GATE',I4,': COMPONENT ''',A8,''' TYPE',I6,', ROW',I3,
     1       ' INTERNAL',I6,': MODE =',I5,' CONTRADICTS MODE =',I5,
     2       ', SET BY INITIAL COND.')
1014 FORMAT (/5X,'GATE',I4, ': NODE',I6,' PRESET TO MODE =',I5,
     1       ' BY GATE',I4,'.')
1015 FORMAT (/5X,'GATE',I4, ': NODE',I6,' PRESET TO MODE =',I5,
     1       ' BY INITIAL CONDITIONS.')
1016 FORMAT (1H0,4X,'GATE',I4,': INTERNAL',I6,' PRESET TO MODE =',I5,
     1       ' BY GATE',I4,'.')
1017 FORMAT (1H0,4X,'GATE',I4,': INTERNAL',I6,' PRESET TO MODE =',I5,
     1       ' BY INITIAL CONDITIONS.')
1018 FORMAT ('0 ** COMPONENT ''',A8,'''  TYPE',I6,', ROW',I4,': ALL NOD
     1ES HAVE BEEN PRESET.'/4X,'*  GATE',I4,' BEING ELIMINATED AND PRECE
     2EDING GATES EDITED.')
1019 FORMAT (1H0, '  * GATE',I4,' TYPE = ',A3,' HAS SINGLE INPUT.  GATE
     1 BEING ELIMINATED AND INPUT DIRECTLY INTO GATE',I4,'.')
1020 FORMAT (5X,'GATE',I4,':',3I3,2I6,1X,6('(',I6,',',I6,'),')/
     1       (36X,6('(',I6,',',I6,'),')))
1021 FORMAT ('0*** PRELIMINARY EDIT OF GATE',I4,' COMPLETED ***'//)
1022 FORMAT (//'0*** GATE',I4,' COMPLETED. PRELIMINARY GATE EDIT FOLLOW
     1S ***'//)
1023 FORMAT (7X,'PRIMARY INPUT',I2,' OF GATE',I4,' BEING DELETED.')
1024 FORMAT ('0 ** ALL INPUTS TO GATE',I4,' HAVE BEEN ELIMINATED.'/
     14X,'*  GATE',I4,' BEING ELIMINATED AND PRECEEDING GATES EDITED.')
1025 FORMAT (1H1,'*** OF GATE EDIT OF GATE',I4,' COMPLETED ***'/
     1       ' *** PRINTOUT OF ARRAY ''GATE'' BEFORE RESETTING NODES ***
     2'//    (5X,10I6))
1026 FORMAT (//'0*** PRINTOUT OF ARRAY ''GATE'' AFTER RESETTING NODES *
     1**'     //(5X,10I6))
1027 FORMAT (//,'0*** OF GATE EDIT OF GATE',I4,' COMPLETED ***'/
     1       ' *** PRINTOUT OF ARRAY ''X'' BEFORE RESETTING NODES ***'//
     2       (5X,10I6))
1028 FORMAT (//'0**** ''TOP'' HAS BEEN COMPLETED ****'/
     1       1H1,65('--')/,' - PROGRAM CAT,  VERSION OF 10/75',T131,'-'/
     2,      ' - PROGRAM FOR THE AUTOMATED CONSTRUCTION OF FAULT TREES.'
     3,       T131,'-' /' - OUTPUT REGION'
     4,       T131,'-' /,' -',T131,'-'/ ,' - ',20A4,T131,'-' /,1X,
     5        65('--')///,1X,'*** GATE PRINTOUT SECTION ***'//)
1029 FORMAT (1H1,'*** GATE TRANSFERS COMPLETED ***'//
     1       1X,'INTERMEDIATE PRINTOUT FOR ',20A4//)
1030 FORMAT (1H1,65('--')/' - PROGRAM CAT,  VERSION OF 10/75',T131,'-'/
     1,      ' - PROGRAM FOR THE AUTOMATED CONSTRUCTION OF FAULT TREES.'
     2,       T131,'-' /' - OUTPUT REGION: FINAL GATE PRINTOUT'
     3,       T131,'-' /,' -',T131,'-'/ ,' - ',20A4,T131,'-' /,1X,
     4        65('--')///,1X,'*** FINAL GATE PRINTOUT SECTION ***'//)
1031 FORMAT (1H1,65('**')/5(' *',T131,'*'/),' *', 5X,'FAULT TREE TOO LA
     1RGE FOR PROGRAM DIMENSIONS.',T131,'*'/' *',T131,'*'/
     2     ' *', 5X,'NUMBER OF GATES REQUIRED EXCEEDS THE',I5,' GATES AL
     3LOCATED.',T131,'*'/' *',T131,'*'/   ' *', 5X,'PARTIAL GATE PRINTOU
     4T FOLLOWS.',T131,'*'/ 5(' *',T131,'*'/),1X,65('**'))
```

```
 1032 FORMAT (1H1,65('**')/5(' *',T131,'*'/),' *', 5X,'FAULT TREE TOO LA
     1RGE FOR PROGRAM DIMENSIONS.',T131,'*'/' *',T131,'*'/
     2   ' *', 5X,'ARRAY ''GATE'' REQUIRES MORE THAN THE',I6,' SPACES AL
     3LOCATED.',T131,'*'/' *',T131,'*'/   ' *', 5X,'PARTIAL GATE PRINTOU
     4T FOLLOWS.',T131,'*'/ 5(' *',T131,'*'/),1X,65('**'))
 1033 FORMAT (1H1,65('**')/5(' *',T131,'*'/),' *', 5X,'TOP EVENT FOR FAU
     1LT TREE HAS NO ROW MATCH.',  T131,'*'/' *',T131,'*'/
     2     ' *', 5X,'TOP EVENT CANNOT OCCUR.',
     3          T131,'*'/' *',T131,'*'/   ' *', 5X,'*** PROGRAM TERMINAT
     4ING ***',   T131,'*'/ 5(' *',T131,'*'/),1X,65('**'))
 1034 FORMAT (1H1,65('**')/5(' *',T131,'*'/),' *', 5X,'TOP EVENT FOR FAU
     1LT TREE HAS BEEN ELIMINATED - TOP EVENT CANNOT OCCUR -',T131,'*'/
     2                  ' *',T131,'*'/   ' *', 5X,'*** PROGRAM TERMINAT
     3ING ***',   T131,'*'/ 5(' *',T131,'*'/),1X,65('**'))
 1035 FORMAT (1H1,65('**')/5(' *',T131,'*'/),' *', 5X,'TOP EVENT FOR FAU
     1LT TREE HAS BEEN ELIMINATED - TOP EVENT ''SURE TO OCCUR'' -',
     2          T131,'*'/' *',T131,'*'/   ' *', 5X,'*** PROGRAM TERMINAT
     3ING ***',   T131,'*'/ 5(' *',T131,'*'/),1X,65('**'))
      END
      SUBROUTINE XCHECK (ISTART,JDEX,NDEX,IPRINT,X,GATE,NODES,NGSIZE)
C
C          SUBROUTINE TO CHECK AND SET NODES FOR 'OR' GATE EDIT ROUTINE.
C          ISTART SETS FLAG; IF ISTART IS NON-POSITIVE, NO NODES ARE SET
C          ISTART = 0:  CHECK OR-AND-OR GATE
C          ISTART =-1:  CHECK OR-OR-AND, OR-AND-AND GATES
      INTEGER X(2,NODES),GATE(NGSIZE)
  100 ISET = ISTART
      IF (ISTART .LE. 0) ISTART = 1
      NPRIME = GATE(NDEX + 2)
      ISPOT = NDEX + 1 + 2*(GATE(NDEX+1) + ISTART)
      DO 110 I = ISTART,NPRIME
        ISPOT = ISPOT + 2
        JNODE = GATE(ISPOT)
        KNODE = GATE(ISPOT + 1)
        JMODE = X(1,JNODE)
        IF (JMODE .NE. -1) GO TO 101
C
C         NODE HAS NOT PREVIOUSLY BEEN SET.
        IF (ISET .LE. 0) GO TO 110
        X(1,JNODE) = KMODE
        X(2,JNODE) = -2
        GO TO 110
  101   CONTINUE
C
C         NODE HAS BEEN PRESET.  CHECK FOR AGREEMENT.
        IF (JMODE .EQ. KMODE) GO TO 105
C
C         NODE DISAGREES.  CHECK FOR FURTHER MODES PRESET.
C         IF NONE FOUND, SET ADDITIONAL MODE.
        IF (X(2,JNODE) .LT. -2) GO TO 102
        IF (ISET .GT. 0) X(2,JNODE) = -ISPOT
        GO TO 110
```

```
  102    IISPOT = -X(2,JNODE)
  103    IF (KMODE .EQ. GATE(IISPOT + 1)) GO TO 105
C
C          MODE NOT EQUAL TO PRESET MODE. CHECK FOR FURTHER PRESET MODES.
         IF (GATE(IISPOT) .GE. 0) GO TO 104
         IISPOT = -GATE(IISPOT)
         GO TO 103
  104    IF (ISET .GT. 0) GATE(IISPOT) = -ISPOT
C
C          ALL MODES CHECKED; NO AGREEMENT.  SET NEW MODE LOCATION.
         GO TO 110
  105    CONTINUE
C
C          MODE AGREES WITH PRESET VALUE.
C          DELETE PRIMARY INPUT ISPOT.
         IF (ISET .LE. -1) GO TO 111
         GATE(NDEX + 2) = GATE(NDEX + 2) - 1
         IF (IPRINT .GT. 0) WRITE (6,1000) JDEX,JNODE,KMODE,I,NPRIME
         IF (I .EQ. NPRIME) GO TO 110
         IM1 = NPRIME - 1
         IISPOT = ISPOT
         DO 106 I2 = I,IM1
           GATE(IISPOT)   = GATE(IISPOT + 2)
           GATE(IISPOT+1) = GATE(IISPOT + 3)
  106      IISPOT = IISPOT + 2
         ISPOT = ISPOT - 2
  110    CONTINUE
       RETURN
  111  ISTART = -1
       IF (IPRINT .GT. 0) WRITE (6,1001) JDEX,JNODE,KMODE,JDEX
       RETURN
 1000  FORMAT ('0    GATE',I4,': INTERNAL NODE',I6,' PRESET TO MODE =',
      1   I5/7X,'PRIMARY INPUT',I2,' OF',I2,' INPUTS BEING DELETED.')
 1001  FORMAT ('0    GATE',I4,': INTERNAL NODE',I6,' PRESET TO MODE =',
      1     I5/4X,'*  GATE',I4,' BEING ELIMINATED AND PRECEEDING GATES ED
      2ITED.')
       END
       SUBROUTINE REDUCE(LLDEX,JDEX2,NDEX2,JDEX,NDEX,IPRINT,J,JNGATE,
      1                  JSPOT,GATE,JGATE,NGSIZE,MGATE)
C
C        SUBROUTINE TO ELIMINATE ZERO AND SINGLE INPUT GATES.
C        GATE JDEX2 IS TO BE ELIMINATED.
C        GATE JDEX  WILL HAVE CORRESPONDING INPUT CHANGED OR DELETED.
       INTEGER   GATE(NGSIZE)
       DIMENSION JGATE(MGATE),KIND(2)
       DATA      KIND/'AND ','OR  '/
       IKIND = GATE(NDEX2)
       GATE(NDEX2)  = -99
       JGATE(JDEX2) = -99
       IF (LLDEX .EQ. 1) GO TO 200
C
C        GATE JDEX2 HAS NO INPUTS REMAINING.
```

```
C       DELETE GATE JDEX2 AND INPUT TO JDEX.
        JKIND = GATE(NDEX)
        JJ = JNGATE + GATE(NDEX + 2)
        IF (IPRINT .GT. 0) WRITE (6,1000) JDEX2,JDEX2,JDEX2,KIND(JKIND),
       1                    JDEX,J,JJ
        GATE(NDEX + 1) = GATE(NDEX + 1) - 1
        JJSPOT = JSPOT
        IF (J .GE. JNGATE) GO TO 101
        JM1 = JNGATE - 1
        DO 100 J2 = J,JM1
          GATE(JJSPOT)   = GATE(JJSPOT + 2)
          GATE(JJSPOT+1) = GATE(JJSPOT + 3)
  100   JJSPOT = JJSPOT + 2
  101 JPRIME = GATE(NDEX + 2)
        IF (JPRIME .EQ. 0) GO TO 102
        JSPOT2 = JJSPOT + 2*JPRIME
        GATE(JJSPOT)   = GATE(JSPOT2)
        GATE(JJSPOT+1) = GATE(JSPOT2 + 1)
        IF (GATE(JSPOT2) .LT. 0) GATE(JSPOT2) = -JJSPOT
  102 JSPOT = JSPOT - 2
        JJ = NDEX + 4 + 2*(GATE(NDEX+1) + GATE(NDEX+2))
        IF (IPRINT .GT. 1) WRITE (6,1001) JDEX,(GATE(II),II=NDEX,JJ)
        RETURN
  200 CONTINUE
C
C       GATE JDEX2 HAS SINGLE INPUT.
C       DELETE JDEX2 AND INSERT DIRECTLY INTO GATE JDEX.
        IF (IPRINT .GT. 0) WRITE (6,1002) JDEX2,KIND(IKIND),JDEX
        IF (GATE(NDEX2 + 1) .EQ. 0) GO TO 201
C
C       INPUT TO JDEX2 IS GATE INPUT.  SET INTO GATE JDEX.
        GATE(JSPOT) = GATE(NDEX2 + 5)
        JJSPOT = GATE(JSPOT)
        JGATE(JJSPOT) = JDEX
        RETURN
  201 CONTINUE
C
C       INPUT TO GATE JDEX2 IS PRIMARY INPUT.
C       MOVE UP REMAINING GATE INPUTS AND INSERT PRIMARY INPUT.
        GATE(NDEX + 1) = GATE(NDEX + 1) - 1
        GATE(NDEX + 2) = GATE(NDEX + 2) + 1
        JJSPOT = JSPOT
        IF (J .EQ. JNGATE) GO TO 203
        JM1 = JNGATE - 1
        DO 202 J2 = J,JM1
          GATE(JJSPOT)   = GATE(JJSPOT + 2)
          GATE(JJSPOT+1) = GATE(JJSPOT + 3)
  202   JJSPOT = JJSPOT + 2
  203 GATE(JJSPOT)   = GATE(NDEX2 + 5)
        GATE(JJSPOT+1) = GATE(NDEX2 + 6)
C
C       IN 'OR' GATE EDIT REGION, GATE(NDEX2 + 5) MAY BE POINTER.
```

```
C       IF SO, SET POINTER FOR NEW LOCATION.
  204 IF (GATE(JJSPOT) .LT. 0) GATE(NDEX2 + 5) = -JJSPOT
      JJ = NDEX + 4 + 2*(GATE(NDEX+1) + GATE(NDEX+2))
      IF (IPRINT .GT. 1) WRITE (6,1001) JDEX,(GATE(II),II=NDEX,JJ)
      JSECT = JSECT - 2
      RETURN
 1000 FORMAT ('0 ** ALL INPUTS TO GATE',I4,' HAVE BEEN ELIMINATED.'/
     1     4X,'*  GATE',I4,' BEING ELIMINATED AND PRECEEDING GATES EDITE
     2D.'/7X,'GATE',I4,' INPUTS ''',A3,''' GATE',I4,';  BRANCH',I3,
     3         ' OF',I3,' BRANCHES BEING ELIMINATED.')
 1001 FORMAT (5X,'GATE',I4,':',3I3,2I6,1X,6('(',I6,',',I6,')')/
     1        36X,6('(',I6,',',I6,'))
 1002 FORMAT ('0  * GATE',I4,' TYPE = ',A3,' HAS SINGLE INPUT.  GATE BEI
     1NG ELIMINATED AND INPUT DIRECTLY INTO GATE',I4,'.')
      END
      SUBROUTINE OUTPUT (ITYPE,IGATE,JGATE,GATE,MODNAM,CMPNAM,NLIB,
     1                   MNODE,NNCMP,MGATE,MXINT2,MAXINT,NGSIZE,NGATE,IOT)
C*********************************************************************
C                                                                     *
C         SUBROUTINE OUTPUT, VERSION OF MAY 1977                      *
C       SUBROUTINE TO WRITE OUTPUT IN FORMAT FOR PREP-KITT CODES      *
C       LIMITED TO 9999 GATES WITH MAXIMUM OF 49 INPUTS PER GATE      *
C       LAMBDA LIMITED TO VALUES BETWEEN 0.9999 AND 1.E-9             *
C                                                                     *
C       IOT    = OUTPUT DEVICE NUMBER (PUNCH, TAPE OR DISK)           *
C       INMAX  = MAX NO. OF INPUTS ALLOWED PER GATE ( = 7 FOR PREP)   *
C       IREAD  = O/N (DATA CORRECT/MISSING INPUT FROM DEVICE 'NINP')  *
C                                                                     *
C*********************************************************************
      DOUBLE PRECISION CMPNAM(NNCMP),NAME1,NAME2,NNAME,MNAME(2),NAMEX,
     1                 LLEND,MMEND,MODNAM(MAXINT,NLIB),MBLANK
      DIMENSION ALAMDA(2),BLAMDA(2),ATAU(2),BTAU(2),NUMBER(10),
     1          ITYPE(NNCMP),IGATE(MGATE),JGATE(MGATE),INAME(8),
     2          JNAME(56),KNAME(8),KIND(2)
      INTEGER   GATE(NGSIZE)
      LOGICAL   NOTOP
      COMMON TITLE(20),XXXX(20),IERR,IEDIT,IDUM(12),NNAME,JPRINT,KOUT
      DATA   KIND(1),KIND(2),MBLANK,LG,LA,LT,LE,LO,LP,LEND,MEND,LLEND,
     1       MMEND,XEND,IOUT,XOUT,NAMEX,NUMBER,MBLANK/
     2       'AND','OR',' ','G','A','T','E','O','P','END','&END','END',
     3       '&END','&END','&OUT','&OUT','END FILE','0','1','2','3','4',
     4       '5','6','7','8','9',' '/
      DO 10 I = 1,12
   10   IDUM(I) = 0
      DO 20 I = 4,8
   20   INAME(I) = MBLANK
      DO 30 I = 1,56
   30   JNAME(I) = MBLANK
      NINP  = 5
      INMAX = 7
      IREAD = 0
      KSECT = 1
```

```
      TAA   = 0.0
      WRITE (6,1000) ICT,TITLE
 1000 FORMAT (1H1,65(2H--)/29H - PROGRAM CAT, OUTPUT REGION,T131,1H-/
     1        23H - OUTPUT TO I/O DEVICE,I3,
     2        30H IN FORMAT FOR PREP-KITT CODES,T131,1H-/2H -,T131,1H-/
     3        3H - ,20A4,T131,1H-/1X,65(2H--)/)
C*********************************************************************
C                                                                     *
C       CHECK FOR PRESENCE OF '&OUT', '&END' OR END OF FILE           *
C       READ IN SUBROUTINE 'STEVE'                                    *
C       IF DATA IS MISSING OUTPUT WILL STILL BE PRODUCED,              *
C       BUT WITH NO CROSS-CHECKING                                    *
C                                                                     *
C*********************************************************************
      XCODE = XXXX(1)
      IF (XCODE .EQ. XOUT) GO TO 100
      NNAME = NAMEX
      IF (XCODE .EQ. XEND) NNAME = MMEND
      WRITE (6,1001) NNAME
 1001 FORMAT ('0*** PREP DATA MISSING ***'/7X,A8,' READ BY SUBROUTINE ST
     1EVE.'/  ' FAULT TREE WILL BE OUTPUT WITHOUT FAILURE DATA.'//)
      IREAD = 3
      IF (XCODE .EQ. XEND) IREAD = 2
      GO TO 140
  100 CONTINUE
C
C       READ PREP CONTROL DATA
      READ (NINP,1002,END=130) XXXX,NCODE,(IDUM(I),I=1,6)
 1002 FORMAT (20A4,T1,A4,I6,5I10)
      IF (NCODE .NE. NBLANK) GO TO 110
      READ (NINP,1003,END=130) XXXX,NCODE,(IDUM(I),I=7,10),TAA
 1003 FORMAT (20A4,T1,A4,I6,3I10,F20.3)
      IF (NCODE .NE. NBLANK) GO TO 110
      GO TO 140
  110 CONTINUE
      IF (NCODE .EQ. LOUT) GO TO 100
C
C       INPUT ERROR READ.  SUPPRESS FURTHER DATA CHECKING
      IF (NCODE .EQ. LEND .OR. NCODE .EQ. MEND) GO TO 120
      WRITE (6,1004) XXXX
 1004 FORMAT ('0*** INPUT ERROR ***'/7X,'DATA CARD MISSING OR MISPUNCHED
     1.  CARD IN ERROR IS:'/7X,20A4/' FAULT TREE WILL BE OUTPUT WITHOUT
     2FAILURE DATA.'//)
      IREAD = 4
      GO TO 140
  120 WRITE (6,1005) NCODE
 1005 FORMAT ('0*** WARNING ***'/1X,A4,' CARD FOUND WHERE DATA EXPECTED.
     1'/' INPUT EDIT TERMINATING.  FAULT TREE WILL BE OUTPUT WITHOUT FAI
     2LURE DATA.'//)
      IREAD = 1
      IF (NCODE .EQ. MEND) IREAD = 2
      GO TO 140
```

```
  130 WRITE (6,1006)
 1006 FORMAT ('0*** WARNING ***'/' END OF FILE REACHED'/' INPUT EDIT TER
     1MINATING.  FAULT TREE WILL BE OUTPUT WITHOUT FAILURE DATA.'//)
      IREAD = 3
  140 CONTINUE
C
C       COMPUTE NUMBER OF GATES FOR FREE INPUT
      NG = 0
      DO 150 I = 1,NGATE
        NDEX = IGATE(I)
        IF (GATE(NDEX) .LE. 0) GO TO 150
        NG = NG + 1
        INTOT = GATE(NDEX+1) + GATE(NDEX+2)
        IF (INTOT .LE. INMAX) GO TO 150
        NG = NG + (INTOT - 1)/INMAX + 1
  150 CONTINUE
      IDUM(1) = NG
      WRITE (IOT,1008) TITLE,(IDUM(I),I=1,10),TAA
 1008 FORMAT (20A4//5X,59H*** FAULT TREE CONSTRUCTED BY CAT, VERSION OF
     1MAY  1977 ***/6H* DATA/6I10/4I10,F20.3/3HEND/6H* TREE)
      WRITE (6,1009) TITLE,(IDUM(I),I=1,10),TAA
 1009 FORMAT (1H0,20A4//6X,59H*** FAULT TREE CONSTRUCTED BY CAT, VERSION
     1 OF MAY  1977 ***/7H * DATA/1X,6I10/1X,4I10,F20.3/4H END/
     2        7H * TREE)
C*****************************************************************************
C                                                                             *
C       NOW PRODUCE FAULT TREE IN FREE FORMAT                                 *
C       KSPOT = NEXT AVAILABLE STORAGE SPOT FOR COMPONENT NUMBER              *
C                                                                             *
C*****************************************************************************
      NOTOP = .FALSE.
      INAME(1) = LT
      INAME(2) = IC
      INAME(3) = LE
      INTOT = 0
      DO 400 I = 1,NGATE
        NDEX = IGATE(I)
        JTYPE = GATE(NDEX)
        IF (JTYPE. LE. 0) GO TO 400
        NG = JGATE(I)
        NKIND = KIND(JTYPE)
        IF (.NOT. NOTOP) GO TO 220
        NUM1 = NG/1000
        NUM2 = (NG - 1000*NUM1)/100
        NUM3 = (NG - 100*(NG/100))/10
        NUM4 =  NG - 10*(NG/10)
        INAME(5) = NUMBER(NUM1+1)
        INAME(6) = NUMBER(NUM2+1)
        INAME(7) = NUMBER(NUM3+1)
        INAME(8) = NUMBER(NUM4+1)
  220   INGATE = GATE(NDEX+1)
        IPRIME = GATE(NDEX+2)
```

```
      NRESET = 8*INTCT
      INTOT  = INGATE + IPRIME
      IF (INTOT .GT. INMAX) GO TO 300
C
C       SET NAMES FOR INPUT GATES
C
      ISPOT = NDEX + 3
      JSPOT = 1
      IF (INGATE .EQ. 0) GO TO 250
      DO 240 KGATE = 1,INGATE
        ISPOT = ISPOT + 2
        NG = GATE(ISPOT)
        JNAME(JSPOT)   = LG
        JNAME(JSPOT+1) = LA
        JNAME(JSPOT+2) = LT
        JNAME(JSPOT+3) = LE
        NUM1 = NG/1000
        NUM2 = (NG - 1000*NUM1)/100
        NUM3 = (NG - 100*(NG/100))/10
        NUM4 =  NG - 10*(NG/10)
        JNAME(JSPOT+4) = NUMBER(NUM1+1)
        JNAME(JSPOT+5) = NUMBER(NUM2+1)
        JNAME(JSPOT+6) = NUMBER(NUM3+1)
        JNAME(JSPOT+7) = NUMBER(NUM4+1)
  240   JSPOT = JSPOT + 8
      IF (IPRIME .EQ. 0) GO TO 270
  250 CONTINUE
C
C     SET NAMES FOR COMPONENTS
C
      DO 260 KGATE = 1,IPRIME
        ISPOT = ISPOT + 2
        DO 255 II = 1,2
          NC = GATE(ISPOT + II - 1)
          NUM1 = NC/1000
          NUM2 = (NC - 1000*NUM1)/100
          NUM3 = (NC - 100*(NC/100))/10
          NUM4 =  NC - 10*(NC/10)
          JNAME(JSPOT)   = NUMBER(NUM1+1)
          JNAME(JSPOT+1) = NUMBER(NUM2+1)
          JNAME(JSPOT+2) = NUMBER(NUM3+1)
          JNAME(JSPOT+3) = NUMBER(NUM4+1)
  255     JSPOT = JSPOT + 4
C
C           NOW SET COMPONENT NUMBER INTO TABLE FOR LATER REFERENCE
C
        GATE(KSPOT) = 10000*GATE(ISPOT) + GATE(ISPOT+1)
  260   KSPOT = KSPOT + 1
  270 CONTINUE
C
C         NOW BLANK OUT ANY NAMES REMAINING FROM PREVIOUS RECORD
C
```

```
        IF (NRESET .LT. JSPOT) GO TO 290
        DO 280 JJSPOT = JSPOT,NRESET
  280    JNAME(JJSPOT) = NBLANK
  290   CONTINUE
C
C         NOW WRITE OUTPUT
C
        WRITE (IOT,1010) INAME,NKIND,INGATE,IPRIME,JNAME
 1010   FORMAT (8A1,1X,A4,2I2,7(1X,8A1))
        WRITE (6,1011)  INAME,NKIND,INGATE,IPRIME,JNAME
 1011   FORMAT (1X,8A1,1X,A4,2I2,7(1X,8A1))
        IF (NOTOP) GO TO 400
        INAME(1) = LG
        INAME(2) = LA
        INAME(3) = LT
        INAME(4) = LE
        NOTOP = .TRUE.
        GO TO 400
  300   CONTINUE
C*************************************************************************
C                                                                        *
C         ADD NEW GATES FOR THOSE GATES WHICH HAVE TOO MANY INPUTS       *
C                                                                        *
C*************************************************************************
        JNGATE = (INICT-1)/INMAX  +  1
        JPRIME = 0
        IF (NOTOP) GO TO 310
        NUM5 = NUMBER(1)
        NUM6 = NUM5
        NUM7 = NUM5
        NUM8 = NUMBER(2)
        GO TO 320
  310   NUM5 = INAME(5)
        NUM6 = INAME(6)
        NUM7 = INAME(7)
        NUM8 = INAME(8)
  320   JSPOT = 1
C
C         SET NAMES FOR ADDITIONAL GATES REQUIRED
C
        DO 330 KGATE = 1,JNGATE
          JNAME(JSPOT)   = LG
          JNAME(JSPOT+1) = LA
          JNAME(JSPOT+2) = LT
          JNAME(JSPOT+3) = NUMBER(KGATE+1)
          JNAME(JSPOT+4) = NUM5
          JNAME(JSPOT+5) = NUM6
          JNAME(JSPOT+6) = NUM7
          JNAME(JSPOT+7) = NUM8
  330     JSPOT = JSPOT + 8
C
C           NOW BLANK OUT ANY NAMES REMAINING FROM PREVIOUS RECORD
```

```
C
        IF (NRESET .LT. JSPOT) GO TO 350
        DO 340 JJSPOT = JSPOT,NRESET
  340     JNAME(JJSPOT) = NBLANK
  350   CONTINUE
C
C         WRITE OUTPUT
        WRITE (IOT,1010) INAME,NKIND,JNGATE,JPRIME,JNAME
        WRITE (6,1011)   INAME,NKIND,JNGATE,JPRIME,JNAME
        IF (NOTOP) GO TO 355
          INAME(1) = LG
          INAME(2) = IA
          INAME(3) = LT
          INAME(5) = NUM5
          INAME(6) = NUM6
          INAME(7) = NUM7
          INAME(8) = NUM8
          NOTOP = .TRUE.
  355   CONTINUE
C*************************************************************************
C                                                                        *
C         WRITE ADDITIONAL GATE CARDS.                                   *
C         THE 'INTOT' INPUTS TO THE ORIGINAL GATE WILL BE SPLIT INTO     *
C         'JNGATE' GATES, WITH 'JNTOT' INPUTS INTO THE FIRST 'KNGATE'    *
C         GATES AND 'JNTOT + 1' INPUTS INTO THE LAST 'JNGATE-KNGATE.'    *
C         FIRST BLANK OUT ANY EXTRA POSITIONS.                           *
C                                                                        *
C*************************************************************************
        JNTOT = INTOT/JNGATE
        IF (JNTOT .GE. JNGATE) GO TO 360
        NRESET = 8*JNGATE
        JSPOT = 8*JNTOT + 1
        DO 358 JJSPOT = JSPOT,NRESET
  358     JNAME(JJSPOT) = NBLANK
  360   KNGATE = JNGATE*(JNTOT+1) - INTOT
        ISPOT = NDEX + 3
C
C         LOOP 399 WRITES ONE RECORD FOR EACH EXTRA GATE
C
        DO 399 KGATE = 1,JNGATE
          INAME(4) = NUMBER(KGATE+1)
          IF (KGATE .EQ. (KNGATE+1)) JNTOT = JNTOT + 1
          IF (INGATE .LT. JNTOT) GO TO 365
          JJGATE = JNTOT
          JPRIME = 0
          INGATE = INGATE - JJGATE
          GO TO 370
  365     JJGATE = INGATE
          JPRIME = JNTOT - JJGATE
          INGATE = 0
  370     JSPOT = 1
          IF (JJGATE .EQ. 0) GO TO 380
```

```
C
C           SET NAMES FOR INPUT GATES
C
          DO 375 KKGATE = 1,JJGATE
            ISPOT = ISPOT + 2
            NG = GATE(ISPOT)
            JNAME(JSPOT)   = IG
            JNAME(JSPOT+1) = LA
            JNAME(JSPOT+2) = IT
            JNAME(JSPOT+3) = LE
            NUM1 = NG/1000
            NUM2 = (NG - 1000*NUM1)/100
            NUM3 = (NG - 100*(NG/100))/10
            NUM4 =  NG - 10*(NG/10)
            JNAME(JSPOT+4) = NUMBER(NUM1+1)
            JNAME(JSPOT+5) = NUMBER(NUM2+1)
            JNAME(JSPOT+6) = NUMBER(NUM3+1)
            JNAME(JSPOT+7) = NUMBER(NUM4+1)
  375       JSPOT = JSPOT + 8
          IF (JPRIME .EQ. 0) GO TO 390
  380     CONTINUE
C
C           SET NAMES FOR COMPONENTS
C
          DO 385 KKGATE = 1,JPRIME
            ISPOT = ISPOT + 2
            DO 382 II = 1,2
              NC = GATE(ISPOT + II - 1)
              NUM1 = NC/1000
              NUM2 = (NC - 1000*NUM1)/100
              NUM3 = (NC - 100*(NC/100))/10
              NUM4 =  NC - 10*(NC/10)
              JNAME(JSPOT)   = NUMBER(NUM1+1)
              JNAME(JSPOT+1) = NUMBER(NUM2+1)
              JNAME(JSPOT+2) = NUMBER(NUM3+1)
              JNAME(JSPOT+3) = NUMBER(NUM4+1)
  382         JSPOT = JSPOT + 4
C
C               NOW SET COMPONENTS INTO TABLE FOR LATER REFERENCE
C
            GATE(KSPOT) = 10000*GATE(ISPOT) + GATE(ISPOT+1)
  385       KSPOT = KSPOT + 1
  390     CONTINUE
C
C           WRITE OUTPUT
C
          WRITE (IOT,1010) INAME,NKINI,JJGATE,JPRIME,JNAME
          WRITE (6,1011)   INAME,NKINI,JJGATE,JPRIME,JNAME
  399     CONTINUE
        INTOT = JNTOT
        INAME(4) = LE
  400   CONTINUE
```

```
      WRITE (ICT,1012)
 1012 FORMAT (3HEND)
      WRITE (6,1013)
 1013 FORMAT (4H END)
C*********************************************************************
C                                                                     *
C         NOW SORT COMPONENT NUMBERS INTO ASCENDING NUMERICAL ORDER    *
C         TO BE USED TO CROSSCHECK PREP INPUT.                        *
C         IREAD .GT. 0 MEANS INPUT ERROR WAS DETECTED                 *
C         AND NO CROSSCHECK WILL BE PERFORMED.                        *
C                                                                     *
C*********************************************************************
      NCOUNT = KSECT - 1
      NCNT2 = NCOUNT/2
      DO 430 NDEX = 1,NCNT2
        NDXMAX = NCOUNT + 1 - NDEX
        IGMAX  = GATE(NDEX)
        IGMIN  = IGMAX
        NDEX1  = NDEX + 1
        IDEX   = NDEX
        JDEX   = NDEX
C
C         FIND LOCATIONS OF LARGEST AND SMALLEST ENTRIES.
C         IGMAX & IGMIN ARE ENTRIES, IDEX & JDEX ARE LOCATIONS.
C
        DO 420 NDEX2 = NDEX1,NDXMAX
          IF (GATE(NDEX2) .GE. IGMIN) GO TO 410
          IGMIN = GATE(NDEX2)
          IDEX  = NDEX2
          GO TO 420
  410     IF (GATE(NDEX2) .LE. IGMAX) GO TO 420
          IGMAX = GATE(NDEX2)
          JDEX  = NDEX2
  420     CONTINUE
        IF (IDEX .EQ. JDEX) GO TO 440
C
C         NOW SWITCH LARGEST AND SMALLEST ENTRIES TO ENDS OF ARRAY
C
        GATE(IDEX) = GATE(NDEX)
        GATE(JDEX) = GATE(NDXMAX)
        IF (IDEX .EQ. NDXMAX) GATE(JDEX) = GATE(NDEX)
        IF (JDEX .EQ. NDEX)   GATE(IDEX) = GATE(NDXMAX)
        GATE(NDEX) = IGMIN
  430   GATE(NDXMAX) = IGMAX
  440 CONTINUE
C
C       PROGRAM NOW ELIMINATES DUPLICATED ENTRIES
C
      NCNT2 = 1
      DO 450 NDEX = 2,NCOUNT
        IF (GATE(NDEX) .EQ. GATE(NCNT2)) GO TO 450
        NCNT2 = NCNT2 + 1
```

```
        GATE(NCNT2) = GATE(NDEX)
  450   CONTINUE
        NCCUNT = NCNT2
        NCNT2 = 0
        DO 460 NDEX = 1,NCCUNT
  460   GATE(NCCUNT + NDEX) = 0
        IF (IREAD .NE. 0) GO TO 700
C*************************************************************************
C                                                                        *
C          NOW READ FAILURE AND REPAIR DATA AND CROSSCHECK               *
C          WITH COMPONENT LIST.  WRITE INTO PREP DATASET ONLY            *
C          THAT DATA ACTUALLY REQUIRED.  FLAG ANY EXTRA OR MISSING DATA  *
C                                                                        *
C*************************************************************************
        WRITE (IOT,1014)
 1014   FORMAT (7H* RATES)
        WRITE (6,1015)
 1015   FORMAT (8H * RATES)
        IDONE = 1
  500   READ (NINP,1016,END=580) NAME1,ALAMDA(1),ATAU(1),INT1,MODE1,
       1                         NAME2,ALAMDA(2),ATAU(2),INT2,MODE2
 1016   FORMAT (2(A8,2X,F10.6,F10.3,2I5))
        IF (NAME1 .EQ. ILEND .OR. NAME1 .EQ. MMEND) GO TO 580
        IF (NAME1 .EQ. MBLANK) GO TO 575
        NNAME  = NAME1
        INTRNL = INT1
        MODE   = MODE1
        IDONE  = 1
  505   CONTINUE
C
C         SEARCH ARRAY 'CMPNAM' FOR COMPONENT 'NNAME'
C
        DO 510 NDEX = 1,NNCMP
          IF (NNAME .EQ. CMPNAM(NDEX)) GO TO 520
  510   CONTINUE
  515   WRITE (6,1017) NNAME,MODE,INTRNL
 1017   FORMAT (7X,'COMPONENT ',A8,' OR COMPONENT MODE ',I5,
       1         ' FOR INTERNAL MODE',I5,' NOT FOUND')
        GO TO 570
  520   CONTINUE
        NODE = MNODE + (NDEX-1)*MXINT2 + INTRNL
        NCOMP = 10000*NODE + MODE
C
C         NOW SEARCH ARRAY 'GATE' FOR COMPONENT NODE AND MODE NUMBER
C
        NMIN = 0
        NMAX = NCCUNT + 1
  530   N = (NMIN + NMAX)/2
        IF (NCOMP-GATE(N)) 531,540,533
  531   NMAX = N
  532   IF ((NMAX-NMIN)-1) 515,515,530
  533   NMIN = N
```

```
      GO TO 532
  540 CONTINUE
C
C       COMPONENT INDEX 'N' IN ARRAY 'GATE' FOUND.
C       SET DATA INTO OUTPUT ARRAYS.
C
      IF (GATE(NCOUNT + N) .EQ. 0) GO TO 550
      WRITE (6,1018) NNAME,INTRNL,MODE
 1018 FORMAT ('0*** WARNING ***'/7X,'DUPLICATE DATA INPUT FOR COMPONENT'
     1         ,1X,A8,', INTERNAL NODE',I5,', MODE',I5/
     2         7X,'PREVIOUS DATA WILL BE USED'/)
      GO TO 570
  550 GATE(NCOUNT+N) = 1
      NCNT2 = NCNT2 + 1
      BLAMDA(JLOOP) = ALAMDA(ILOOP)
      BTAU(JLOOP)   = ATAU(ILOOP)
      NUM1 = NODE/1000
      NUM2 = (NODE - 1000*NUM1)/100
      NUM3 = (NODE - 100*(NODE/100))/10
      NUM4 =  NODE - 10*(NODE/10)
      NUM5 =  MODE/1000
      NUM6 = (MODE - 1000*NUM5)/100
      NUM7 = (MODE - 100*(MODE/100))/10
      NUM8 =  MODE - 10*(MODE/10)
      IF (JLOOP .EQ. 2) GO TO 560
      INAME(1) = NUMBER(NUM1+1)
      INAME(2) = NUMBER(NUM2+1)
      INAME(3) = NUMBER(NUM3+1)
      INAME(4) = NUMBER(NUM4+1)
      INAME(5) = NUMBER(NUM5+1)
      INAME(6) = NUMBER(NUM6+1)
      INAME(7) = NUMBER(NUM7+1)
      INAME(8) = NUMBER(NUM8+1)
      JLOOP = 2
      GO TO 570
  560 KNAME(1) = NUMBER(NUM1+1)
      KNAME(2) = NUMBER(NUM2+1)
      KNAME(3) = NUMBER(NUM3+1)
      KNAME(4) = NUMBER(NUM4+1)
      KNAME(5) = NUMBER(NUM5+1)
      KNAME(6) = NUMBER(NUM6+1)
      KNAME(7) = NUMBER(NUM7+1)
      KNAME(8) = NUMBER(NUM8+1)
      WRITE (IOT,1019) INAME,BLAMDA(1),BTAU(1),KNAME,BLAMDA(2),BTAU(2)
 1019 FORMAT (2(8A1,2X,F10.3,F10.3,1X))
      WRITE (6,1020)   INAME,BLAMDA(1),BTAU(1),KNAME,BLAMDA(2),BTAU(2)
 1020 FORMAT (1X,2(8A1,2X,F10.3,F10.3,1X))
      JLOOP = 1
  570 IF (ILOOP .EQ. 2) GO TO 500
  575 ILOOP = 2
      IF (NAME2 .EQ. MBLANK) GO TO 500
      NNAME = NAME2
```

```
      INTERNL = INT2
      MODE   = MODE2
      GO TO 505
C*****************************************************************************
C                                                                             *
C         END OF OUTPUT REGION.                                               *
C         CHECK FOR MISSING DATA AND PRINT CROSSTABLE OF OUTPUT               *
C                                                                             *
C*****************************************************************************
  580 CONTINUE
      IF (JLOOP .EQ. 1) GO TO 590
C
C         HALF CARD REMAINS.
C
      WRITE (IOT,1019) INAME,BLANK(1),ETAU(1)
      WRITE (6,1020)   INAME,BLANK(1),ETAU(1)
  590 WRITE (ICT,1012)
      WRITE   (6,1013)
      IF (NCNT2 .GT. 0) GO TO 600
      IREAD = 5
      IF (NAME1 .EQ. LIEND) IREAD = 6
      IF (NAME1 .EQ. MMEND) IREAD = 7
      IF (IREAD .EQ. 5) NAME1 = NAMEX
      WRITE (6,1021) NAME1
 1021 FORMAT ('0*** WARNING ***'/7X,A8,' READ WITHOUT VALID FAILURE DATA
     1.'/7X,'NO FAILURE DATA WILL BE OUTPUT.'//)
      GO TO 700
  600 CONTINUE
C*****************************************************************************
C                                                                             *
C         CHECK ARRAY 'GATE' FOR MISSING DATA                                 *
C                                                                             *
C*****************************************************************************
      JDEX = 2*NCOUNT
      IDEX = NCOUNT + 1
      ILOOP = 1
      DO 620 NDEX = IDEX,JDEX
        IF (GATE(NDEX) .EQ. 1) GO TO 620
        IF (ILOOP .EQ. 2) GO TO 610
        WRITE (6,1022)
 1022   FORMAT (1H0,65(2H**)/2H *,T131,1H*/
     1    32H * *** INPUT ERROR, DATA MISSING,T131,1H*/
     2    47H * THE FOLLOWING COMPONENTS HAVE NO INPUT DATA:,T131,1H*/
     3    2H *,T131,1H*/2H *,18X,8HINTERNAL,T131,1H*/2H *,5X,9HCOMPONENT
     4,   6X,4HNODE,6X,4HMODE,T131,1H*/2H *,T131,1H*)
        ILOOP = 2
  610   KDEX = GATE(NDEX-NCOUNT)/10000
        IDEX = (KDEX - MNODE - 1)/MXINT2 + 1
        MODE = GATE(NDEX-NCOUNT) - 10000*KDEX
        NODE = KDEX - (LDEX-1)*MXINT2 - MNODE
        WRITE (6,1023) CMPNAM(LDEX),NODE,MODE
 1023   FORMAT (2H *,5X,A8,4X,I6,5X,I6,T131,1H*)
```

```
  620   CONTINUE
      IF (ILOOP .EQ. 1) GO TO 700
      WRITE (6,1024)
 1024 FORMAT (1H ,65(2H**))
  700 CONTINUE
C*****************************************************************************
C                                                                             *
C         WRITE CROSS-INDEX FOR COMPONENTS INDEXED IN THIS FAULT TREE         *
C                                                                             *
C*****************************************************************************
      WRITE (6,1025) TITLE
 1025 FORMAT (1H1,65(2H--)/54H - PROGRAM CAT, SUBROUTINE OUTPUT, VERSION
     1 OF MAY 1977,T131,1H-/58H - CROSS-INDEX OF COMPONENT NAMES USED FO
     2R PREP/KITT INPUT,T131,1H-/2H -,T131,1H-/3H - ,20A4,T131,1H-/1X,
     3       65(2H--)///5H PREP,6X,9HCOMPONENT,2X,9HCOMPONENT,2X,
     4       28HINTERNAL: INTERNAL: INTERNAL,2X,7HFAILURE/5H NAME,10X,
     5       11HINDEX  NAME,9X,7HCOLUMN:,3X,12HNUMBER: NAME,8X,5HSTATE/)
      DO 710 NDEX = 1,NCOUNT
        NCOMP = GATE(NDEX)
        NODE  = NCOMP/10000
        MODE  = NCOMP - 10000*NODE
        IDEX  = (MODE - MNODE - 1)/MXINT2 + 1
        MODE2 = NODE - (IDEX-1)*MXINT2 - MNODE
        IT    = ITYPE(IDEX)
        NUM1  = MODE/1000
        NUM2  = (MODE - 1000*NUM1)/100
        NUM3  = (MODE - 100*(MODE/100))/10
        NUM4  =  MODE - 10*(MODE/10)
        NUM5  =  MODE/1000
        NUM6  = (MODE - 1000*NUM5)/100
        NUM7  = (MODE - 100*(MODE/100))/10
        NUM8  =  MODE - 10*(MODE/10)
        INAME(1) = NUMBER(NUM1+1)
        INAME(2) = NUMBER(NUM2+1)
        INAME(3) = NUMBER(NUM3+1)
        INAME(4) = NUMBER(NUM4+1)
        INAME(5) = NUMBER(NUM5+1)
        INAME(6) = NUMBER(NUM6+1)
        INAME(7) = NUMBER(NUM7+1)
        INAME(8) = NUMBER(NUM8+1)
        WRITE (6,1026) INAME,IDEX,CMPNAM(IDEX),MODE2,NODE,
     1          MODNAM(MODE2,IT),MODE
 1026   FORMAT (1X,8A1,I11,2X,A8,I11,1H:,I9,2H: ,A8,I9)
  710   CONTINUE
      ILOOP = 0
      JREAD = IREEL + 1
      IF (IREAD .EQ. 0 .AND. NAME1 .NE. LLEND) GO TO 999
      GO TO (910,910,999,999,900,999,910,999),JREAD
  900 CONTINUE
C
C       READ REMAINING CARDS IF IREAD .EQ. 0, 1, 4 OR 6
C       UNTIL '&END' OR END OF FILE IS FOUND
```

```
C
      IICCE = 1
      WRITE (6,1027)
 1027 FORMAT (1H1,'*** OUTPUT REGION TERMINATING ***'/
     1        ' EXTRA OR ERRONEOUS CARDS READ:'/)
      WRITE (6,1028) XXXX
 1028 FORMAT (' ***',20A4)
  910 READ (NINP,1002,END=999) XXXX,NCODE
      IF (NCODE .EQ. MEND) GO TO 999
      IF (ILOOP .EQ. 0) WRITE (6,1027)
      IICCE = 1
      WRITE (6,1028) XXXX
      GO TO 910
  999 IERR = IREAD
      RETURN
      END
```