NUREG/CR-4213
SAND83—2675
RG, 1S
Printed May 1985

SETS Reference Manual

R. B. Worrell

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-76DP00789

Prepared for
U. S. NUCLEAR REGULATORY COMMISSION

Available from

Superintendent of Documents
U.S. Government Printing Office
Post Office Box 37082
Washington, D.C. 20013-7982

and

National Technical Information Service
Springfield, VA 22161

NUREG/CR-4213
SAND83-2675
RG,1S

SETS REFERENCE MANUAL

R. B. Worrell

May 1985

Sandia National Laboratories
Albuquerque, New Mexico 87185
Operated by
Sandia Corporation
for the
U.S. Department of Energy

Prepared for
Division of Risk Analysis, Reactor Risk Branch
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555
Under Memorandum of Understanding DOE 40-550-75
NRC FIN NO. A1360

ABSTRACT

The Set Equation Transformation System (SETS) is a computer program used to manipulate Boolean equations symbolically. The capabilities of SETS and how they are invoked by procedure calls in a SETS user program are described in this reference manual.

CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

LIST OF FIGURES

LIST OF FIGURES (Cont.)

LIST OF TABLES

ACKNOWLEDGMENTS

I want to thank Bernie L. Hulme for many helpful discussions concerning the contents of the SETS Reference Manual. Bernie's help was invaluable. He helped with simple details of grammar and phrasing, and he helped resolve problems that arose in describing complex concepts. His suggestions shortened and improved many portions of the manual. But more than anything else, I appreciate the support Bernie provided when I became discouraged. Without his support, I doubt the SETS Reference Manual would have been completed.

I was fortunate to have two people who helped resolve problems in the descriptions of complex concepts. Desmond W. Stack read early descriptions of many concepts and proposed changes for improving them. Desmond provided insight and suggestions from a user's point of view, and he helped simplify many parts of the manual.

Sharon L. Daniel and Bernie served as reviewer and referee, respectively, for the final draft of the SETS Reference Manual. I thank them for performing these functions. Their attention to detail and careful reading of the final draft led to several final improvements in the manual.

SETS REFERENCE MANUAL

1. INTRODUCTION

The Set Equation Transformation System (SETS) is a computer program used to achieve the symbolic manipulation of Boolean equations. Symbolic manipulation changes equations from their original forms into more useful or desirable forms -- particularly by the application of identities. The SETS program is an interpreter which reads, interprets, and executes SETS user programs. The user writes a SETS user program specifying the processing to be achieved and submits it, along with the required data, for execution by SETS. SETS user programs and the capabilities for symbolically manipulating Boolean equations are described in this manual.

1.1 History

The impetus for developing SETS stemmed from the need to study the safety and reliability of complex systems. Originally, the program was called the Eventpoint System Analysis Program (ESAP), and it was designed to analyze system logic through the symbolic manipulation of Boolean equations. A few years after the development of ESAP began, a new method for studying the behavior of complex systems was reported. The new method, fault tree analysis, was soon recognized to be well suited for the kind of processing required in safety and reliability analysis. It was apparent the ESAP algorithms for manipulating Boolean equations could be used to manipulate the Boolean equations of a fault tree. The decision was made to change ESAP to a fault tree analysis program, while retaining the general Boolean algebra capabilities possessed by the program. With this change in emphasis, the program was renamed the Set Equation Transformation System (SETS) (Ref. 1), and its use for fault tree analysis was subsequently described (Ref. 2).

1.2 Applications

Early versions of SETS were being used for system safety analysis when WASH-1400 (Ref. 3) signaled the beginning of the use of fault trees and event trees to study nuclear reactor safety. The capabilities of SETS were ideally suited for both fault tree and event tree analysis. The fault tree analysis capabilities had already been developed for use in safety studies, and the general capability to process Boolean equations provided the basis for doing event tree analysis. Sandia National Laboratories began to conduct reactor safety studies using SETS to achieve the required analytical processing.

The general nature of SETS, i.e., the capability to manipulate Boolean equations regardless of their origin, led to the identification of ways to study other aspects of reactor safety. For example, in vital area analysis (Refs. 4 and 5), fault trees are used to model reactor hardware failures and Boolean equations are used to model reactor hardware locations. These models are then combined to identify the vital areas of the reactor and the reactor site. The effect of common cause events on reactor safety can also be analyzed using the general equation manipulation capabilities of SETS (Refs. 6 and 7).

Applications of Probabilistic Risk Analysis (PRA) to analyze reactor safety, together with applications of vital area analysis and common cause analysis, led to the identification of additional capabilities and improvements for SETS. Over

the last few years, the capabilities of SETS have been extended significantly in these areas. The most recent improvements stemmed from experience gained from the analyses performed as part of the Interim Reliability Evaluation Program (IREP) (Ref. 8). The IREP study illuminated certain needs with respect to extending SETS capabilities, and it also led to the development of improved methods for using SETS, such as building up large reactor fault trees from generic, modular fault trees (Ref. 9). The improvements that emerged from IREP produced a comprehensive method for obtaining and analyzing accident sequences in a PRA (Ref. 10).

The current version of SETS is a flexible and efficient tool for performing PRA, vital area analysis, and common cause analysis on nuclear reactor plants. In addition, the equation manipulation capabilities of SETS can also be used to solve a variety of other problems. The use of SETS to analyze noncoherent fault trees and determine prime implicants of Boolean functions has been described (Refs. 11 and 12). The SETS program can also be used to verify circuit design implementation (Ref. 13); to determine minimum cost fire protection requirements for reactor plants (Refs. 14 and 15); and to obtain solutions to combinatorial optimization problems with Boolean constraints (Refs. 16-20). Also, SETS can be used to determine the susceptibility of a facility to unauthorized access through nullification of sensors in its protection system (Ref. 21).

1.3 Using the Reference Manual

This manual describes SETS user programs and the equation manipulation capabilities available in SETS. The SETS user language, whose final construction is a SETS user program, is described in Section 2. In Section 3, the use of the equation, block and value block files during the execution of SETS user programs is explained. The procedures that can be invoked in a SETS user program are described in section 4. The procedure descriptions are very thorough and they include examples to illustrate various uses of the procedures.

Errors are described in Appendix A. Errors that indicate a malfunction in the SETS program are called SETS errors; they cause execution of a SETS user program to be terminated, and they usually cannot be corrected by the user. SETS user program errors, which indicate mistakes in the SETS user program or its data, are also described in Appendix A; they can usually be corrected by the user.

Appendix B shows all of the parameter forms for every procedure. This appendix may be useful as a quick indicator of procedure capabilities.

Appendix C is a summary description of the algorithm used to factor disjunctive normal form expressions.

The SETS Reference Manual is two manuals in one. Taken as a whole, it is a complete description of SETS user programs and the processing achieved by procedures called in these programs. However, Sections 2 and 3, together with the general material in Section 4 and the synopsis and procedure call parts of each procedure description, constitute an abridged reference manual that may be suitable for many levels of interest. Appendices A and B would probably also be useful as part of an abridged reference manual.

2. SETS USER LANGUAGE

The ultimate construction of the SETS user language is a SETS user program. A SETS user program is comprised of Boolean equation statements and procedure call statements which are built up from basic elements of the language.

Variables, expressions, and equations in the SETS user language are always Boolean, and the word "Boolean" is usually omitted when referring to them.

Spaces (or blanks) have no meaning in the SETS user language; nor do they have meaning in input representations of blocks and value blocks. This means spaces can be used freely to arrange information in SETS user programs, blocks, and value blocks for both appearance and readability.

2.1 Character Set

The character set for the SETS user language is comprised of capital letters (A-Z), decimal digits (0-9), and the following special characters:

. + * / ( ) $ = , - ∅

where ∅ represents a space or blank.

2.2 Basic Elements

The basic elements of the SETS user language are special characters, key words, names, numbers, and comments.

2.2.1 Special Characters

Some special characters have more than one use, but the meaning of multiple-use special characters is obvious from the context in which they occur. For example, the asterisk (*) is an operator in an expression and a separator in the parameter part of a procedure call.

2.2.2 Key Words

Key words are catenations of letters that have a fixed meaning in the language. For example, GENFTEQN and METHOD1 are key words. GENFTEQN is the procedure identifier that begins each call of the Generate Fault Tree Equation procedure; and METHOD1 is a parameter in a call of GENFTEQN that specifies a top-down approach is to be used to find minimal cut sets.

2.2.3 Names

Names identify variables, blocks, value blocks, and SETS user programs. Names are comprised of from 1 to 16 name characters. Name characters are the capital letters (A-Z), decimal digits (0-9), and the minus sign (-) used as a hyphen. A name can begin with any name character. Examples of names are:

A	391	BLK-1-A
MCS-EQNS	FT-3-AND-4	-3267-A

2.2.4 Numbers

Numbers are used to assign nonnegative values to variables and specify truncation values in procedure call statements. Numbers are either basic numbers or basic numbers with exponents. Basic numbers must be positive or zero. A basic number is an integer, a decimal fraction, or an integer followed by a decimal fraction. A basic number has one of the forms

i .f i.f

where i and f represent integer and decimal fraction, respectively. An empty decimal fraction is not allowed; if the decimal point occurs, f cannot be empty. A decimal fraction whose value is zero is not empty. For example, 9.00 has a zero decimal fraction; whereas, 18. has an empty decimal fraction. The number of digits in a basic number, d_i+d_f, is restricted as follows:

$$d_i+d_f \leq 14$$

Exponents can be either positive or negative. Exponents begin with the letter E and have one of the forms

Ej E-j

where j is an integer restricted to the range:

$$-279 \leq j \leq 308.$$

(The number of digits in a basic number and the exponent range are for the implementation of SETS on a CDC CYBER76.) The plus sign (+) cannot occur if the exponent is positive, and the minus sign (-) must occur if the exponent is negative. For example, 7.0E+2 is an invalid number because there is a plus sign in the exponent. Examples of numbers are:

82	.00999	3.00
1.75 E-3	6.009E17	173.0 E5

2.2.5 Comments

Comments can occur in SETS user programs and input representations of blocks and value blocks. A comment begins with key word COMMENT followed by a dollar sign, and it includes the string of characters through the next dollar sign. Comments are read and discarded as the SETS user program, block, or value block is read. In SETS user programs and blocks, a comment can follow a period delimiter (not a decimal point in a number), or another comment. In value blocks, a comment can follow another comment or the dollar sign at the end of a value assignment. Examples of comments are:

```
COMMENT$ EQUATIONS FOR COMMON CAUSE EVENTS.$
COMMENT$ X, Y, AND Z ARE CONSTANT$
```

Comments are not described further in this manual, but a few examples in the manual contain comments to illustrate their use.

2.3 Constants

The constants of the Boolean algebra, one (1) and zero (0), are represented by OMEGA and /OMEGA, respectively.

2.4 Variables

Variable names can be chosen freely except for OMEGA and /OMEGA which are reserved for constants.

There are complement and noncomplement variables. Actually, two variables are complementary or one variable is the complement of another. However, it is helpful to arbitrarily distinguish between complement and noncomplement variables so either of these groups of variables can be referenced without having to identify every variable in the group. Noncomplement variables have the form X, and complement variables have the form /X. Actually, /X is the unary complement operator (NOT) acting on the single variable X, but together the operator and the variable can be interpreted as a complement variable. Example variables are:

PUMP-FAILS /ON -BUS-A /12-RM-3B

2.5 Expressions

Expressions are built up from variables, operators, and parentheses. The operations of AND, OR, and NOT are represented by special characters *, +, and /, respectively. Complement expressions are represented by enclosing the expression in parentheses and preceding the expression with the NOT operator. Expressions are interpreted from left to right, and the following rule of precedence applies to operators of the same level:

first, NOT (/)
second, AND (*)
third, OR (+)

Example expressions are:

/OMEGA

A*/B*C + B*/C + C*D*E*F

X*(/Y*(X+Z*/(Y*W+/Y*/W)+/Z)+Y)+/X

Expressions are often put into disjunctive normal form. A disjunctive normal form is a sum-of-products in which products having complementary variables are deleted, and remaining products do not contain repeated variables. (Products in a sum-of-products form are also called terms.) Disjunctive normal forms are obtained by expanding by the distributive law, P*(Q + R) = P*Q + P*R, and applying P*/P = /OMEGA, P*/OMEGA = /OMEGA, P+/OMEGA = P, and P*P = P to the result.

Disjunctive normal form expressions are often simplified by deleting subsuming terms. (Deleting subsuming terms is the same as applying the identity P + P*Q = P.) A term T_1 subsumes a term T_2, if every variable of T_2 is among the variables of T_1, e.g., if T_1=A*B*C and T_2=A*C, then T_1 subsumes T_2.

2.6 Equations

Equations define equivalence relationships between variables and expressions. The left side of an equation must be a single, noncomplement variable. Because of this restriction, an equation can be referred to by its left-side variable. For example, "the equation for X" refers to the equation that has X as its left-side variable.

The right-side expression of an equation can be any well-formed expression, but there is one restriction. The right-side expression cannot contain either the left-side variable of the equation or its complement. For example, equation

```
X=Y*(Z+X)
```

cannot be used because the right-side expression contains left-side variable X. The restriction only applies to explicit occurrences of the left-side variable or its complement. For example, equations

```
A=B+C
B=D*/(C+A)
```

can be used because the right-side expression of the equation for A does not contain A or /A, and the right-side expression of the equation for B does not contain B or /B. However, substituting either equation into the other, which is frequently done in SETS, produces an equation whose right-side expression contains the left-side variable. Anytime an equation is read or created which has an explicit occurrence of the left-side variable or its complement in the right-side expression, a SETS user program error occurs (see APPENDIX A, Section 3.3, Numbered Errors, error 48). Examples of valid equations are:

```
X=Y*Z

OUTPUT=INPUT1+/INPUT1*INPUT2

A = /(B*(C*(/D+/B)+/C*B)+E)

POWER-ON = OMEGA.
```

2.7 Blocks

A block is a group of one or more equations that can be referenced by a single block name.

A fault tree is a block. When it is necessary to distinguish between fault tree blocks and other blocks, they are referred to as fault tree blocks and equation blocks. Often, fault tree blocks are simply called fault trees.

Only block names are needed in SETS user programs. However, input representations of equation blocks and fault trees must also be described because they can be read by SETS user programs.

2.7.1 Equation Blocks

Equation blocks can be created by SETS user programs or they can be read as input by SETS user programs. The input representation of an equation block is described in Section 4.2.1, Read Block.

2.7.2 Fault Tree Blocks

A fault tree block contains the intermediate event equations of the fault tree. However, a fault tree block also contains tables showing how the fault tree is connected. The fault tree block name and the name of the fault tree are the same.

Fault trees can be created by SETS user programs or they can be read as input by SETS user programs. The input representation of a fault tree is described in Section 4.4.1, Read Fault Tree.

2.8 Value Blocks

A value block is a group of one or more variables and their nonnegative values. In a given value block, each variable has one value. Value blocks specify variable values used in term value computations (see Section 4.1.6, Compute Term Value, and Section 4.1.7, Truncate on Term Value).

Only value block names are needed in SETS user programs. However, input representations of value blocks must also be described because they can be read by SETS user programs. The input representation of a value block is described in Section 4.3.1, Read Value Block.

2.9 SETS User Programs

A SETS user program is an algorithm which manipulates Boolean equations symbolically. A SETS user program consists of a header and one or more statements.

2.9.1 Header

A SETS user program begins with a header of the form:

PROGRAM$ pn.

where pn is the program name chosen by the user.

2.9.2 Statements

There are two kinds of statements in a SETS user program: equation statements and procedure call statements. Each statement is terminated by a period (.).

2.9.2.1 Equation Statements

Equation statements define equations. An equation statement can be any well-formed equation (see Section 2.6, Equations).

2.9.2.2 Procedure Call Statements

Procedure call statements invoke procedures. A procedure call begins with a procedure identifier which is usually followed by a parameter part enclosed in parentheses.

A procedure represents a body of processing required to achieve a particular result. For example, a call of the Print Equation in Disjunctive Normal Form procedure obtains each specified equation, expands it into disjunctive normal form, and prints the terms of the equation in the order of an increasing number of variables per term. The procedures that can be invoked in a SETS user program are described individually in Section 4. All forms of the procedure call are shown and the processing achieved by the procedure is described.

3. WHAT THE USER MUST KNOW TO WRITE SETS USER PROGRAMS

SETS user programs are algorithms which manipulate Boolean equations symbolically. A SETS user program is comprised of statements that are executed in the order they occur; there are no conditional, iterative, or transfer statements in the SETS user language. There are two kinds of statements: equation statements and procedure call statements. The user must know what each statement does, i.e., causes to happen, and how each procedure affects three SETS user program files. Equation statements and the SETS user program files are described in this section; procedures are described in Section 4.

3.1 SETS User Program Files

The equation file, block file, and value block file, are used to store equations, blocks, and value blocks, respectively. The block file and value block file can be saved after the execution of a SETS user program. Saving these files saves blocks and value blocks produced by one SETS user program so they may be used in another SETS user program at a later time. The capability to save equations (a block contains one or more equations) and, to a lesser extent, value blocks for later use is extremely useful. Problems can be solved using a series of manageable SETS user programs instead of all at once with one large SETS user program. Also, the results of a completed analysis can be used to extend the analysis at a later time without having to duplicate work that has already been done.

A distinction is made between using a file and changing a file. Using a file means to obtain a copy of the equation, block, or value block from its respective file; using a file does not change the file. In the Substitute in Equation procedure, for example, obtaining an equation from the equation file to begin the substitution process does not change the equation file. A file is changed when an equation, block, or value block is either entered into or deleted from its respective file.

3.1.1 Equation File

Equations are stored in the equation file. Equations are identified and referenced by their left-side variables. The equation file cannot contain two equations with the same left-side variable. Whenever an equation to be entered into the equation file has the same left-side variable as an equation already in the file, the new equation replaces the existing equation.

The equation file is always empty when the execution of a SETS user program begins, and it is lost when execution is completed. However, equations in the equation file can be saved by forming a block that contains them, adding it to the block file, and saving the block file.

3.1.2 Block File

Equation blocks and fault tree blocks are stored in the block file; they are identified and referenced by their block names. The block file cannot contain two blocks with the same name. A SETS user program error occurs whenever a block to be entered into the block file has the same name as a block already in the file (see APPENDIX A, Section 3.3, Numbered Errors, error 60).

The block file can be saved after a SETS user program has been executed. The block file is saved by placing statements in the job control sequence which assign the block file (TAPE4) to a permanent system file. To use a saved block file, statements are placed in the job control sequence which establish the saved file as TAPE4 before execution of the SETS user program begins.

3.1.3 Value Block File

Value blocks are stored in the value block file. Value blocks are identified and referenced by their value block names. The value block file cannot contain two value blocks with the same name. Whenever a value block to be entered into the value block file has the same name as a value block already in the file, the values from the new value block are added to the existing value block. Variables that occur in both value blocks will have the value from the new value block.

The value block file can be saved after a SETS user program has been executed. The value block file (TAPE8) is saved and used in the same way the block file is saved and used.

3.2 How Statements Affect SETS User Program Files

The equation, block, and value block files change as a SETS user program is executed. The user must know how each statement affects these files and use this knowledge when writing a SETS user program.

3.2.1 Equation Statements

What an equation statement does is simple: the equation is entered into the equation file. The equation is entered into the equation file by a procedure that the user cannot call. The procedure is called Load Temporary Block (LDTMPBLK) and it works like the Load Block procedure described in Section 4.2.6, i.e., it enters the equations from a temporary block into the equation file. (Temporary blocks are not stored in the block file and they are not available to the user.) Executing an equation statement is like loading a block having one equation. The printed output produced by the execution of an equation statement consists of printing the equation and a terminal message indicating LDTMPBLK has been executed. If no error has occurred during execution of the SETS user program, the equation is printed in the form described in Section 4.1.1, Print Equation, and the terminal message has the form

STATEMENT EXECUTION REQUIRED n SECONDS FOR LDTMPBLK

where n represents seconds of CPU time. Once an error has occurred, the equation is not printed and the terminal message has the form:

STATEMENT EXECUTION WAS BYPASSED FOR LDTMPBLK

3.2.2 Procedure Call Statements

Procedures are described in Section 4. At the beginning of the section, a general diagram shows how every procedure interacts with the equation file, block file, and value block file. Furthermore, the procedures are described in four groups and, at the beginning of each group, another diagram shows how the procedures in that

group interact with the equation, block, and value block files. Procedure interaction information presented in Section 4 is summarized here in a way that corresponds to the four procedure groups in Section 4.

3.2.2.1 Equation Procedures

Equation procedures may enter an equation into the equation file, delete an equation from the equation file, or print or write an equation that is in the equation file. Equation procedures never change the block file or the value block file, nor do they use the block file. The Compute Term Value and Truncate on Term Value procedures use value blocks from the value block file to compute term values.

3.2.2.2 Block Procedures

Block procedures, as used here, means equation block procedures since fault tree procedures are treated as a separate group (see Section 3.2.2.4). Block procedures may enter a block into the block file, delete a block from the block file, or print the equations from a block in the block file. Block procedures never use or change the value block file. The Form Block procedure uses the equation file to obtain equations to form a block, and the Load Block procedure changes the equation file by loading, i.e., entering, equations from a block into the equation file.

3.2.2.3 Value Block Procedures

Value block procedures may enter value blocks into the value block file, delete value blocks from the value block file, or print variables and values from a value block in the value block file. Value block procedures never use or change the equation file or block file.

3.2.2.4 Fault Tree Procedures

Fault tree procedures may enter a fault tree block into the block file. (Fault tree blocks are deleted and printed using block procedures.) Fault tree procedures do not use or change the equation file or value block file. (The Generate Fault Tree Equation procedure produces a SETS user program segment which, when it is executed, may use all three files and may change the equation file and block file.)

4. PROCEDURES

Procedures are invoked by procedure call statements. A procedure call begins with a procedure identifier which is usually followed by a parameter part enclosed in parentheses. Parameters specify variables, equations, blocks, and value blocks either as input to be processed or as names of output created by the procedure. Some parameters allow the user to control the processing achieved by a procedure; others, called options, provide a way to modify the results produced by a procedure. It is important to understand the processing achieved by a procedure, and how parameters are used to control processing and modify results. It is very important to understand how each procedure affects the equation file, block file, and value block file.

Some procedures do not change the equation file, block file, or value block file. If a procedure changes the equation, block or value block file, it changes only one of these files and the changes are stated in the procedure description. For example, the Reduce Equation procedure creates an equation and . . . "enters it into the equation file."

Procedures are described in four groups. The procedures in each group are listed below with their identifiers:

1. Equation Procedures

PRTEQN	- Print Equation
PRTEQNDNF	- Print Equation in Disjunctive Normal Form
DLTEQN	- Delete Equation
SUBINEQN	- Substitute in Equation
REDUCEQN	- Reduce Equation
COMTRMVAL	- Compute Term Value
TRNTRMVAL	- Truncate on Term Value
DLTRM	- Delete Term
WRTEQNDNF	- Write Equation in Disjunctive Normal Form

2. Block Procedures

RDBLK	- Read Block
PRTBLK	- Print Block
BLKSTAT	- Block Status
DLTBLK	- Delete Block
FRMBLK	- Form Block
LDBLK	- Load Block

3. Value Block Procedures

RDVALBLK	- Read Value Block
PRTVALBLK	- Print Value Block
DLTVALBLK	- Delete Value Block

4. Fault Tree Procedures

RDFT	- Read Fault Tree
FRMNEWFT	- Form New Fault Tree
GENFTEQN	- Generate Fault Tree Equation

The interactions of the procedures with the equation file, block file, and value block file are depicted in the diagram in Figure 4.1. At the beginning of the descriptions for each group, a diagram showing only the procedures in the group and the files they interact with is presented. Each of these diagrams is a portion of the general diagram in Figure 4.1.

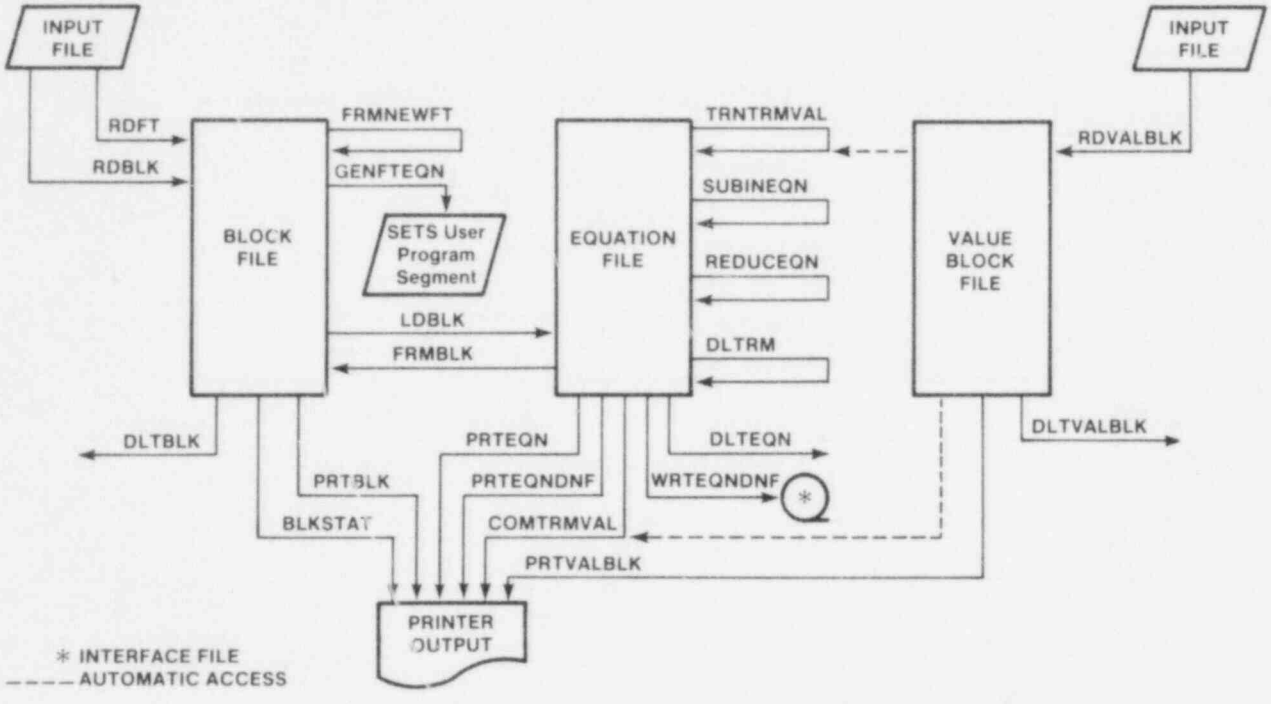


Figure 4.1 Procedure interactions with the equation file, block file, and value block file.

Each procedure description consists of five or six sections. If the procedure has options, the description has six sections; otherwise the section on options is omitted. The six sections in the order they occur in a description are as follows:

Synopsis
Procedure Call
Processing Achieved
Options
Printed Output
Example

A procedure description begins with a synopsis comprised of a one-paragraph description of the procedure and a simple example of its use. The synopsis does not have a heading, but each remaining section in the description does have a heading.

Procedure Call

In this section, every form of the procedure call is shown. Each parameter is identified and its purpose is succinctly stated.

The synopsis and procedure call sections of a procedure description constitute an abridged description of the procedure which may be sufficient for the beginner and also serves as a quick reference for the experienced user. The remaining sections of each procedure description are very thorough.

Notation used to define procedure call statements is as follows:

v_i - variable name

e_i - equation name, i.e., the left-side variable of an equation

t_i - truncation value (for number of variables)

p_i - e_i or e_i/t_i

ft or ft_i - fault tree block name

eb or eb_i - equation block name

b_i - block name (fault tree or equation)

vb_i - value block name

O or O_i - procedure option

c_i - computation type

l_i - truncation value (for computations)

q_i - c_i or c_i/l_i

f_i or f_i' - fault tree event name (intermediate or primary)

ie or ie_i - fault tree intermediate event name

pe or pe_i - fault tree primary event name

Also, a character string enclosed by angle brackets ⟨ ⟩ represents any example of the entity described by the character string. For example, ⟨variable list⟩ represents any list of variable names separated by commas.

Processing Achieved

In this section, concepts must often be defined before the processing achieved by the procedure can be described. When necessary, examples are presented to clarify the concepts. Once the required concepts are defined, the processing of the procedure is described, again with clarifying examples if necessary. Parameters used to control the procedure processing are also described in this section. For

example, a parameter controls whether the Generate Fault Tree Equation procedure uses a top-down or a bottom-up method to obtain minimal cut sets.

Options

In this section, options used to modify the results produced by the procedure are explained. Options are always separated from each other by slashes (/) in procedure calls that have more than one option.

Printed Output

In this section, the printed output produced by the procedure is described. Every procedure produces the standard printed output, and it is described here rather than in each procedure. The standard output consists of a representation of the procedure call and a terminal message indicating execution of the procedure is finished. The terminal message has the form:

STATEMENT EXECUTION REQUIRED n SECONDS FOR <procedure identifier>

where n represents seconds of CPU time. If the only printed output for a procedure is the standard output, a statement indicating this fact appears in this section. Otherwise, printed output in addition to the standard output is described in this section. Printed output in addition to the standard output occurs between the representation of the procedure call and the terminal message.

The terminal message for every procedure executed after an error occurs during execution of a SETS user program is different than the message shown above. Once an error has occurred, the terminal message for procedures that do not read input has the form:

STATEMENT EXECUTION WAS BYPASSED FOR <procedure identifier>

For procedures that read input, the terminal message after an error has occurred has the form:

STATEMENT EXECUTION WAS BYPASSED FOR <procedure identifier>
EXCEPT FOR READING AND CHECKING THE INPUT

Example

In this section, a SETS user program is presented which shows the main uses of the procedure and the printed output produced by the procedure.

4.1 Equation Procedures

Procedures that process equations do not use or change the block file. Interactions between equation procedures and the equation file and value block file are shown in Figure 4.2.

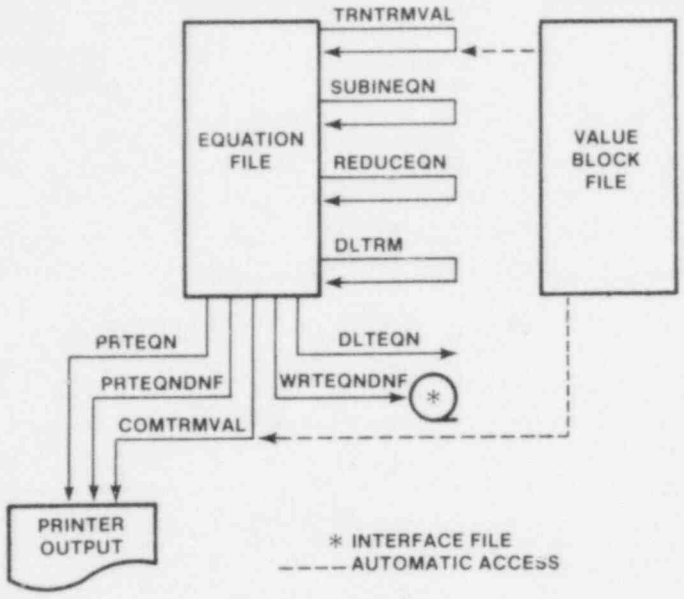


Figure 4.2 Interactions of equation procedures with the equation file and value block file.

Five of the nine procedures used to process equations change the equation file. Procedures SUBINEQN, REDUCEQN, TRNTRMVAL, and DLTRM create a new equation and enter it into the equation file. If the equation file already contains an equation having the same left-side variable as a new equation, the new equation replaces the existing equation. Otherwise, the new equation is added to the equation file. Procedure DLTEQN deletes equations from the equation file.

Procedures PRTEQN and PRTEQNDNF print equations from the equation file. Procedure COMTRMVAL computes term values for equations in the equation file and prints terms and term values. Procedure WRTEQNDNF writes equations from the equation file to an interface file used by other programs (Refs. 22 and 23).

Procedures COMTRMVAL and TRNTRMVAL use value blocks, but they do not change the value block file.

4.1.1 Print Equation

The Print Equation procedure is used to print equations in the equation file. This procedure prints the right-side expression of an equation in a factored form. For example, suppose the equation file contains the equation

A=B*(C+D*(E+F)).

Execution of procedure call statement

PRTEQN (A)

prints the equation for A in the factored form

A = B * (C + D * (E + F)).
1 2 21

Procedure Call

A call of the Print Equation procedure has the form:

PRTEQN (e_1,e_2,...,e_n).

Parameters, e_i, i=1,2,...,n, specify equations to be printed.

Processing Achieved

Does not apply.

Printed Output

If the equation file does not contain an equation for parameter e_i, the message

THERE IS NO EQUATION FOR e_i

is printed and processing of the parameters continues. If the equation file contains an equation for parameter e_i, the equation is printed.

Right-side expressions are printed in the factored form that they have in the equation file. If an expression contains parentheses, an integer is printed below each parenthesis when the equation is printed. These integers are printed to aid in the interpretation of complex equations. The integer printed below a parenthesis is one greater than the number of pairs of parentheses which surround the parenthesis. Integer 1 is printed below the left and the right parenthesis of each outermost pair of parentheses, integer 2 is printed below the left and the right parenthesis of each pair of parentheses surrounded only by an outermost set of parentheses, etc. Paired parentheses have the same number.

In a printed equation, the operations of AND, OR, and NOT are represented by *, +, and /, respectively.

Example

Consider SETS user program:

```
PROGRAM$ PRTEQN-EX.
   EQN-1=/(/X1+X2*OMEGA)*(X3+/X1+X4*
         (X1+X5*(X2+/X3))).
   PRTEQN (X4, X6, EQN-1).
```

The equation file is empty when execution of PRTEQN-EX begins. The first statement in the SETS user program is an equation statement that enters an equation for EQN-1 into the equation file. Then, a PRTEQN call shows the equation file does not contain an equation for either X4 or X6 and prints a factored form of the equation for EQN-1. (The distinction between X4 and X6 is X4 appears as a variable in the equation for EQN-1 and X6 does not appear in an equation.) The printed output produced by these two statements is as follows:

```
          EQN-1 = /( /X1 + X2 * OMEGA ) * ( X3 + /X1 + X4 * ( X1 +
                    1                     1   1                   2

          X5 * ( X2 + /X3 ) ) )
               3          3 2 1

     STATEMENT EXECUTION REQUIRED      .005 SECONDS FOR LDTMPBLK


PRTEQN (X4, X6, EQN-1).

          THERE IS NO EQUATION FOR X4

          THERE IS NO EQUATION FOR X6

          EQN-1 = /( /X1 + X2 * OMEGA ) * ( X3 + /X1 + X4 * ( X1 +
                    1                     1   1                   2

          X5 * ( X2 + /X3 ) ) )
               3          3 2 1

     STATEMENT EXECUTION REQUIRED      .004 SECONDS FOR PRTEQN
```

4.1.2 Print Equation in Disjunctive Normal Form

The Print Equation in Disjunctive Normal Form procedure is used to print equations in the equation file. This procedure prints the right-side expression of an equation in a disjunctive normal form. For example, suppose the equation file contains the equation

A=B*(C+D*(E+F)).

Execution of procedure call statement

PRTEQNDNF (A)

prints the equation for A in the disjunctive normal form:

TERM NUMBER	NUMBER OF VARIABLES	
		A =
1	2	B * C +
2	3	B * D * E +
3	3	B * D * F

Procedure Call

A call of the Print Equation in Disjunctive Normal Form procedure has the form:

PRTEQNDNF (p_1,p_2,...,p_n).

Parameters, p_i, i=1,2,...,n, specify the equations to be printed. Each p_i is either e_i or e_i/t_i which contains left-side variable e_i and truncation value t_i, if t_i is present. Truncation value t_i must be an integer.

Processing Achieved

The right-side expression of an equation in the equation file is in a factored form. DeMorgan's rules must be applied to the expression and it must be expanded by the distributive law to obtain a disjunctive normal form of the right-side expression for printing. Occurrences of the constants OMEGA and /OMEGA must be eliminated by simplification; terms that contain complementary variables must be deleted; and all except one occurrence of a repeated variable in a term must be deleted. Also, if truncation value t_i is specified, terms having more than t_i variables must be identified and deleted. All of this processing is referred to as expansion.

The expansion of an expression to obtain a disjunctive normal form for printing is the same expansion that begins the process of reducing an expression to a simpler form in the Reduce Equation procedure. The processing achieved by expansion is described in detail in Section 4.1.5, Reduce Equation.

Printed Output

If the equation file does not contain an equation for left-side variable e_i contained in parameter p_i, the message

THERE IS NO EQUATION FOR e_i

is printed and processing of the parameters continues. If the equation file contains an equation for e_i, the equation is printed.

There are three parts to the printed output associated with printing a disjunctive normal form of an equation. There is the output associated with changing the factored form of the right-side expression of the equation into a disjunctive normal form; there is a Variable Occurrence Table that contains information about the variables in the printed equation; and there is the printed equation.

The printed output associated with changing the right-side expression into a disjunctive normal form can begin with the warning:

SUBSUMING TERMS MAY OCCUR IN THE
EQUATION FOR e_i

The warning occurs if the equation for e_i was not created by a call of REDUCEQN, TRNTRMVAL, or DLTRM. It means that, even though the right-side expression of the equation may not contain subsuming terms, the absence of subsuming terms in the right-side expression has not been ensured by the execution of one of these procedures in a SETS user program.

The next part of the printed output associated with changing the right-side expression into a disjunctive normal form is the printed output produced by expansion. A detailed description of this output appears in Section 4.1.5, Reduce Equation, and a summary description of it is presented here.

The printed output produced by expansion begins with a message indicating the maximum number of terms that can be generated by expansion. The messages in the remainder of the output show a breakdown of the terms generated with respect to the number of variables per term, the total number of terms generated, and the time required for expansion. For example, expansion of the expression

B*(C+D*(E+F))

produces the printed output

```
THE MAXIMUM NUMBER OF TERMS THAT CAN BE
GENERATED BY EXPANSION IS                3.

TERMS GENERATED BY EXPANSION
        1 TERMS CONTAIN  2 VARIABLES
        2 TERMS CONTAIN  3 VARIABLES
TOTAL TERMS GENERATED        3.
EXPANSION TOOK        .007 SECONDS.
```

Terms that contain complementary variables and terms having more than t_i variables, if t_i is specified, are identified and deleted during expansion. If all of the terms of the expression are deleted, the expression is /OMEGA and the breakdown of terms with respect to the number of variables per term is replaced by the message:

THE EXPRESSION IS /OMEGA

If any terms are deleted because of truncation value t_i, the message

THERE WERE TERMS DELETED BECAUSE OF
THE TRUNCATION VALUE OF t_i

is printed after the message for the total number of terms generated.

A Variable Occurrence Table is printed preceding each equation printed. All and only the variables that occur in the terms of the printed equation are listed in the table. If any terms are deleted during expansion, some variables that occur in the equation for e_i may not appear in the table because they do not occur in the terms produced by expansion. The number of times that each variable occurs in the printed equation is printed with the variable in the table. Since all occurrences of a repeated variable in a term except one have been deleted, the number of times a variable occurs is the number of terms which contain the variable. A message indicating the number of different variables that occur in the terms of the printed equation is printed at the end of the table.

After the Variable Occurrence Table has been printed, the remaining terms of the equation are printed in a disjunctive normal form. Each term is numbered and the terms are printed according to an increasing number of variables per term.

The operations of AND, OR, and NOT are represented in the printed equation by *, +, and /, respectively.

Example

Consider SETS user program:

```
PROGRAM$ PRTEQNDNF-EX.
   EQN-1=/(/X1+X2*OMEGA)*(X3+/X1+X4*
         (X1+X5*(X2+/X3))).
   PRTEQNDNF (X1, EQN-1, EQN-1/3, EQN-1/2).
```

The equation file is empty when execution of PRTEQNDNF-EX begins. The first statement in the SETS user program is an equation statement that enters an equation for EQN-1 into the equation file. Then, a call of PRTEQNDNF shows the equation file does not contain an equation for X1; prints all of the terms of the equation for EQN-1; prints the terms of the equation for EQN-1 that contain 3 or fewer variables; and, shows the equation for EQN-1 does not contain any terms that have 2 or fewer variables. The printed output produced by the equation statement and the printed output produced by the PRTEQNDNF call through the processing of parameter X1 is as follows:

```
EQN-1 = /( /X1 + X2 * OMEGA ) * ( X3 + /X1 + X4 * ( X1 +
          1                  1   1                 2

X5 * ( X2 + /X3 ) ) )
     3          3 2 1

STATEMENT EXECUTION REQUIRED      .004 SECONDS FOR LDTMPBLK


PRTEQNDNF (X1, EQN-1, EQN-1/ 3, EQN-1/ 2).

          THERE IS NO EQUATION FOR X1
```

The printed output produced by the PRTEQNDNF call for parameter EQN-1 is as follows:

```
     SUBSUMING TERMS MAY OCCUR IN THE
     EQUATION FOR EQN-1

     THE MAXIMUM NUMBER OF TERMS THAT CAN BE
     GENERATED BY EXPANSION IS          5.

     TERMS GENERATED BY EXPANSION
          2 TERMS CONTAIN  3 VARIABLES
          1 TERMS CONTAIN  5 VARIABLES
     TOTAL TERMS GENERATED        3.
EXPANSION TOOK      .007 SECONDS.


                          * * * * VARIABLE OCCURRENCE TABLE * * * *

        NONCOMPLEMENT      NUMBER OF        COMPLEMENT       NUMBER OF
          VARIABLE        OCCURRENCES        VARIABLE       OCCURRENCES

            X1                3
                                             /X2                3
            X3                1              /X3                1
            X4                2
            X5                1

     THERE ARE   6 DIFFERENT VARIABLES IN THE
     EQUATION FOR EQN-1


 TERM   NUMBER OF
NUMBER  VARIABLES

                    EQN-1 =

   1        3       X1 * X3 * /X2 +

   2        3       X1 * X4 * /X2 +

   3        5       X1 * X4 * X5 * /X2 * /X3
```

The printed output produced by the PRTEQNDNF call for parameter EQN-1/3 is as follows:

```
          SUBSUMING TERMS MAY OCCUR IN THE
          EQUATION FOR EQN-1

          THE MAXIMUM NUMBER OF TERMS THAT CAN BE
          GENERATED BY EXPANSION IS          5.

          TERMS GENERATED BY EXPANSION
               2 TERMS CONTAIN  3 VARIABLES
          TOTAL TERMS GENERATED        2.

          THERE WERE TERMS DELETED BECAUSE OF
          THE TRUNCATION VALUE OF  3

     EXPANSION TOOK      .005 SECONDS.


                          * * * * VARIABLE OCCURRENCE TABLE * * * *


            NONCOMPLEMENT        NUMBER OF          COMPLEMENT          NUMBER OF
              VARIABLE           OCCURRENCES          VARIABLE          OCCURRENCES

               X1                    2
                                                     /X2                    2
               X3                    1
               X4                    1

          THERE ARE   4 DIFFERENT VARIABLES IN THE
          TRUNCATED EQUATION FOR EQN-1


      TERM   NUMBER OF
     NUMBER  VARIABLES

                        EQN-1/3 =

         1        3     X1 * X3 * /X2 +

         2        3     X1 * X4 * /X2
```

The printed output produced by the PRTEQNDNF call for parameter EQN-1/2 is as follows:

```
          SUBSUMING TERMS MAY OCCUR IN THE
          EQUATION FOR EQN-1

          THE MAXIMUM NUMBER OF TERMS THAT CAN BE
          GENERATED BY EXPANSION IS          5.

          TERMS GENERATED BY EXPANSION
          THE EXPRESSION IS /OMEGA
          TOTAL TERMS GENERATED        1.

          THERE WERE TERMS DELETED BECAUSE OF
          THE TRUNCATION VALUE OF  2

     EXPANSION TOOK      .003 SECONDS.

STATEMENT EXECUTION REQUIRED      .034 SECONDS FOR PRTEQNDNF
```

This page intentionally left blank.

4.1.3 Delete Equation

The Delete Equation procedure is used to delete equations from the equation file. For example, suppose the equation file contains the equation

A=B*(C+D*(E+F)).

Execution of procedure call statement

DLTEQN (A)

deletes the equation for A from the equation file.

Procedure Call

A call of the Delete Equation procedure has one of the forms:

a. DLTEQN.

b. DLTEQN (e_1,e_2,...,e_n).

Parameters, e_i, i=1,2,...,n, specify equations to be deleted from the equation file.

Processing Achieved

The specified equations are deleted from the equation file. If the parameter part of the DLTEQN call is empty (form a), every equation is deleted from the equation file; but if the parameter part is not empty (form b), only the equations for the parameters in the call are deleted from the equation file.

If the equation file does not contain an equation for parameter e_i, the equation file is not changed. The result is the same as if the parameter had not occurred in the DLTEQN call.

Printed Output

A DLTEQN call produces only the standard printed output for a procedure call statement.

Example

Consider SETS user program:

```
PROGRAM$ DLTEQN-EX.
   X1 = X2 + /X2*X3.
   X2 = X3*C1.
   X3 = C2 + C1*C3.
   PRTEQN (X1, X2, X3).
   DLTEQN (X2).
   PRTEQN (X1, X2, X3).
   DLTEQN.
   PRTEQN (X1, X2, X3).
```

The equation file is empty when execution of DLTEQN-EX begins. The first three statements in the SETS user program are equation statements that enter equations for X1, X2, and X3 into the equation file. Then, a PRTEQN call shows these three equations are in the equation file. The printed output produced by the PRTEQN call is as follows:

```
PRTEQN (X1, X2, X3).

          X1 = X2 + /X2 * X3

          X2 = X3 * C1

          X3 = C2 + C1 * C3

     STATEMENT EXECUTION REQUIRED       .005 SECONDS FOR PRTEQN
```

The next statement in the SETS user program is a DLTEQN call which deletes only the equation for X2 from the equation file. Then, a PRTEQN call shows X2 has been deleted from the equation file. The printed output produced by these two procedures is as follows:

```
DLTEQN (X2).

     STATEMENT EXECUTION REQUIRED       .001 SECONDS FOR DLTEQN


PRTEQN (X1, X2, X3).

          X1 = X2 + /X2 * X3

          THERE IS NO EQUATION FOR X2

          X3 = C2 + C1 * C3

     STATEMENT EXECUTION REQUIRED       .004 SECONDS FOR PRTEQN
```

The next statement in the SETS user program is a DLTEQN call without parameters which deletes every equation from the equation file. Then, a PRTEQN call shows all equations have been deleted from the equation file. The printed output produced by these two procedures is as follows:

```
DLTEQN.

     STATEMENT EXECUTION REQUIRED       .001 SECONDS FOR DLTEQN


PRTEQN (X1, X2, X3).

          THERE IS NO EQUATION FOR X1

          THERE IS NO EQUATION FOR X2

          THERE IS NO EQUATION FOR X3

     STATEMENT EXECUTION REQUIRED       .002 SECONDS FOR PRTEQN
```

4.1.4 Substitute in Equation

The Substitute in Equation procedure is used to create a new equation and enter it into the equation file. The right-side expression for the new equation begins as a single variable that occurs as a parameter in the procedure call and is built up by substituting equals for equals using the equations in the equation file. The left-side variable for the new equation occurs as a parameter in the procedure call. For example, suppose the equation file contains the equations

A=B+E
B=C*D
C=D+E.

Execution of procedure call statement

SUBINEQN (A, F)

creates the equation

F=(((D+E)*D)+E)

and enters it into the equation file. The right-side expression for the created equation is built up from variable A by substitutions beginning with the substitution of (B+E) for variable A; continuing with the substitution of (C*D) for variable B; and concluding with the substitution of (D+E) for variable C. The left-side variable for the created equation is F.

Procedure Call

A call of the Substitute in Equation procedure has one of the forms:

a. SUBINEQN (e_1, e_2).

b. SUBINEQN (e_1, e_2* O_1/O_2/.../O_m).

Parameters e_1 and e_2 are required in both forms of the procedure call. Parameter e_1 is the variable that is used to start the substitutions that form the right-side expression for the new equation. Parameter e_2 is the left-side variable for the new equation. Parameters e_1 and e_2 can be the same variable.

Options, O_j, j=1,2,...,m, specify different ways to arrest the substitutions prior to their normal completion. If two or more options occur in a SUBINEQN call, they are separated from each other by slashes (/). The constant OMEGA cannot occur in an option in a SUBINEQN call.

Processing Achieved

If the equation file does not contain an equation for parameter e_1, an equation is not created and the equation file is not changed; but if the equation file contains an equation for e_1, an equation is created and entered into the equation file. Parameter e_2 is the left-side variable for the new equation. The right-side

expression begins as variable e_1 and is built up by substitutions using equations in the equation file. The equation for e_2 is entered into the equation file after the substitutions that form its right-side expression have been completed. Thus, if e_1 and e_2 are the same variable, the equation for e_2 replaces the equation for e_1 in the equation file that was used to start the substitutions.

The right-side expression for the new equation begins as the single variable e_1 which is the first parameter in the SUBINEQN call. Then, every variable in the new expression that has an equation in the equation file is replaced in the new expression by its equivalent right-side expression from the equation file. (Every right-side expression that is substituted into the new expression is first enclosed in parentheses to ensure that the logic of the expression is preserved.) By continuing to make such substitutions for every variable in the new expression, including variables introduced by previous substitutions, the new expression will ultimately contain only variables that do not have an equation in the equation file.

For example, suppose the equation file contains the equations

```
X1=X7*X3
X2=X4*C
X3=X6*(X5+C)
X4=F*(X5+X2)
X5=X7*F
X6=D+E
X7=A+B
```

and that an expression is to be built up by substitutions beginning with the expression

X1. (4-1)

Substitution of the right-side expression of the equation for X1 into Exp. 4-1 produces the expression

(X7*X3). (4-2)

Substitution of the right-side expressions of the equations for X7 and X3 into Exp. 4-2 produces the expression

((A+B)*(X6*(X5+C))). (4-3)

Substitution of the right-side expressions of the equations for X5 and X6 into Exp. 4-3 produces the expression

((A+B)*((D+E)*((X7*F)+C))). (4-4)

Finally, substitution of the right-side expression of the equation for X7 into Exp. 4-4 produces the expression

((A+B)*((D+E)*(((A+B)*F)+C))). (4-5)

The final expression, Exp. 4-5, was built up by substitutions that began with the

substitution for variable X1 and continued until none of the variables in Exp. 4-5 has an equation in the equation file. Thus, the author of a SETS user program must know the contents of the equation file when a SUBINEQN call occurs in order to know the result of the substitutions.

The substitution of the right-side expression of an equation for its left-side variable can initiate an unending sequence of substitutions. Suppose an expression is to be built up by substitutions beginning with the expression

X4. (4-6)

Substitution of the right-side expression of the equation for X4 into Exp. 4-6 produces the expression

(F*(X5+X2)). (4-7)

Substitution of the right-side expressions of the equations for X2 and X5 into Exp. 4-7 produces the expression

(F*((X7*F)+(X4*C))) (4-8)

which contains the variable X4. In this example, substitution for the variable X4 initiates an unending sequence of substitutions.

Equations that would produce an unending sequence of substitutions can remain undetected in the equation file. If a sequence of substitutions including these equations is never initiated, the unending sequence of substitutions will not be discovered. However, if a sequence of substitutions including these equations is initiated, the unending sequence of substitutions will be detected and a SETS user program error will occur (see APPENDIX A, Section 3.3, Numbered Errors, error 48). The first example, which began with the substitution for X1 and produced Exp. 4-5, did not encounter the equations for either X2 or X4 and the unending sequence of substitutions involving these equations was not detected. The second example, which began with the substitution for X4 and produced Exp. 4-8, resulted in the detection of the unending sequence of substitutions and would have caused the SETS user program error to occur.

Options

If the SUBINEQN call contains options (form b), variables in the expression built up by substitutions may not be replaced by their right-side expressions from the equation file, or they may be replaced by OMEGA or /OMEGA instead of their right-side expressions.

The options that can occur in a SUBINEQN call are the stop option, the omega option, and the phi option. Any combination of these options can occur in a SUBINEQN call, and they can occur in any order. Moreover, a SUBINEQN call can contain two or more occurrences of the same kind of option, e.g., two stop options. The effect, however, is the same as if all of the separate options of the same kind had been collected into a single option of that kind.

A variable cannot occur more than once in all of the options in a call of SUBINEQN. The constant OMEGA cannot occur in any option in a call of SUBINEQN.

A stop option has the form:

STOP$ $v_1,v_2,...,v_r$

A stop option identifies variables that are not replaced by their right-side expressions from the equation file as the right-side expression of the new equation is built up by substitutions. Every occurrence of each v_i, i=1,2,...,r, in the expression being built up by substitutions remains in the expression as a variable. If a variable does not have an equation in the equation file, its occurrence in a stop option has no effect. If a variable has an equation in the equation file, its occurrence in a stop option does not affect its equation in the equation file.

An omega option has the form:

OMEGA$ $v_1,v_2,...,v_r$

An omega option in a SUBINEQN call identifies variables that are replaced by the constant OMEGA as the right-side expression of the new equation is built up by substitutions. Every occurrence of each v_i, i=1,2,...,r, in the expression being built up by substitutions is replaced in the expression by the constant OMEGA. If a variable has an equation in the equation file, its occurrence in an omega option does not affect its equation in the equation file.

A phi option has the form:

PHI$ $v_1,v_2,...,v_r$

A phi option in a SUBINEQN call identifies variables that are replaced by the constant /OMEGA as the right-side expression of the new equation is built up by substitutions. Every occurrence of each v_i, i=1,2,...,r in the expression being built up by substitutions is replaced in the expression by the constant /OMEGA. If a variable has an equation in the equation file, its occurrence in a phi option does not affect its equation in the equation file.

Printed Output

If the equation file does not contain an equation for e_1, the message

THERE IS NO EQUATION FOR e_1

is printed, and execution of the SUBINEQN call is terminated. Otherwise, a SUBINEQN call produces only the standard printed output for a procedure call statement.

Example

Consider SETS user program:

```
PROGRAM$ SUBINEQN-EX.
   Z1 = Z2 + /Z2*Z3.
   Z2 = Z5 + K2.
   Z3 = Z5 + Z6*(K1 + K3).
   Z4 = Z3*K3.
   Z5 = K1 + K5.
   Z6 = K2*K4*Z5.
   SUBINEQN (NO-EQN, NEW-EQN).
   PRTEQN (NEW-EQN).
   SUBINEQN (Z1, Z1-SUB).
   PRTEQN (Z1-SUB).
   SUBINEQN (Z1, Z1-STOP* STOP$ Z2, K2, Z6).
   PRTEQN (Z1-STOP).
   PRTEQN (Z1).
   SUBINEQN (Z1, Z1* OMEGA$ Z6/ PHI$ Z2, K1).
   PRTEQN (Z1).
```

The equation file is empty when execution of SUBINEQN-EX begins. The first six statements in the SETS user program are equation statements that enter equations for Z1, Z2, Z3, Z4, Z5, and Z6 into the equation file. Then, a SUBINEQN call shows the equation file does not contain an equation for NO-EQN, and the following PRTEQN call shows that the SUBINEQN call did not create an equation for NEW-EQN. The printed output produced by the two procedure calls is as follows:

```
SUBINEQN (NO-EQN, NEW-EQN).

          THERE IS NO EQUATION FOR NO-EQN

     STATEMENT EXECUTION REQUIRED       .001 SECONDS FOR SUBINEQN


PRTEQN (NEW-EQN).

          THERE IS NO EQUATION FOR NEW-EQN

     STATEMENT EXECUTION REQUIRED       .002 SECONDS FOR PRTEQN
```

The next statement in the SETS user program is a SUBINEQN call that creates an equation for Z1-SUB. This call does not contain any options. The substitutions begin with the substitution for variable Z1 and continue until no further substitutions can be made. After the substitutions are complete, an equation for Z1-SUB is created and added to the equation file. Then, a PRTEQN call prints the equation for Z1-SUB and shows that it has been added to the equation file. The printed output produced by these two procedures is as follows:

```
SUBINEQN (Z1, Z1-SUB).

     STATEMENT EXECUTION REQUIRED       .003 SECONDS FOR SUBINEQN


PRTEQN (Z1-SUB).

          Z1-SUB = ( ( ( K1 + K5 ) + K2 ) + /( ( K1 + K5 ) + K2 ) * ( (
                    1 2 3         3      2    2 3         3      2   2 3

          K1 + K5 ) + ( K2 * K4 * ( K1 + K5 ) ) * ( K1 + K3 ) ) )
                3   3           4         4 3   3          3 2 1

     STATEMENT EXECUTION REQUIRED       .004 SECONDS FOR PRTEQN
```

The next statement in the SETS user program is a SUBINEQN call that creates an equation for Z1-STOP. This call contains a stop option that stops the substitutions for variables Z2, K2, and Z6. Thus, Z2, K2, and Z6 remain as variables in the right-side expression of the new equation, while all other substitutions continue until no further substitutions can be made. (The equation created for Z1-STOP is the same whether or not K2 occurs in the stop option because K2 does not have an equation in the equation file.) After the substitutions are complete, an equation for Z1-STOP is created and added to the equation file. Then, a PRTEQN call prints the equation for Z1-STOP and shows that it has been added to the equation file. The printed output produced by these two procedures is as follows:

```
SUBINEQN (Z1, Z1-STOP* STOP$ Z2, K2, Z6).

     STATEMENT EXECUTION REQUIRED      .003 SECONDS FOR SUBINEQN


PRTEQN (Z1-STOP).

          Z1-STOP = ( Z2 + /Z2 * ( ( K1 + K5 ) + Z6 * ( K1 + K3 ) ) )
                    1            2 3        3        3         3 2 1

     STATEMENT EXECUTION REQUIRED      .003 SECONDS FOR PRTEQN
```

The next statement in the SETS user program is a PRTEQN call that prints the equation for Z1 prior to the SUBINEQN call that redefines this equation. The SUBINEQN call that creates an equation for Z1 contains both an omega and a phi option. The omega option causes occurrences of Z6 to be replaced by OMEGA, and the phi option causes occurrences of Z2 and K1 to be replaced by /OMEGA. All other substitutions continue until no further substitutions can be made. After the substitutions are complete, an equation for Z1 is created which replaces the equation for Z1 in the equation file. Then, a PRTEQN call prints the equation for Z1 and shows the equation for Z1 created by the SUBINEQN call has replaced the original equation for Z1 in the equation file. The printed output produced by these three procedures is as follows:

```
PRTEQN (Z1).

          Z1 = Z2 + /Z2 * Z3

     STATEMENT EXECUTION REQUIRED      .002 SECONDS FOR PRTEQN


SUBINEQN (Z1, Z1* OMEGA$ Z6/ PHI$ Z2, K1).

     STATEMENT EXECUTION REQUIRED      .003 SECONDS FOR SUBINEQN


PRTEQN (Z1).

          Z1 = ( /OMEGA + OMEGA * ( ( /OMEGA + K5 ) + OMEGA * (
               1                  2 3             3           3

          /OMEGA + K3 ) ) )
                   3 2 1

     STATEMENT EXECUTION REQUIRED      .003 SECONDS FOR PRTEQN
```

4.1.5 Reduce Equation

The Reduce Equation procedure is used to create a new equation and enter it into the equation file. A parameter in the procedure call specifies an equation in the equation file that is used to generate the new equation. A copy of the right-side expression from the equation in the equation file is expanded, simplified, and factored to produce the right-side expression for the new equation. The left-side variable for the new equation appears as a parameter in the procedure call. For example, suppose the equation file contains the equation

A=B*(C*(D+E)+B*D+E).

Execution of procedure call statement

REDUCEQN (A, F)

creates the equation

F=B*(D+E)

and enters it into the equation file. A copy of the expression for A is expanded by the distributive law to obtain four terms B*C*D, B*C*E, B*B*D, and B*E. Identity P*P = P is used to delete one occurrence of variable B from term B*B*D and produce a disjunctive normal form of the expression for A,

B*C*D + B*C*E + B*D + B*E.

The expression is simplified by deleting terms B*C*D and B*C*E because they subsume terms B*D and B*E, respectively. Remaining terms, B*D and B*E, are factored to obtain the right-side expression for the new equation. The left-side variable for the new equation is F.

Procedure Call

A call of the Reduce Equation procedure has one of the forms:

a. REDUCEQN (p_1, e_2).

b. REDUCEQN (p_1, e_2* $O_1/O_2/.../O_m$).

Parameters p_1 and e_2 are required in both forms of the procedure call. Parameter p_1 specifies the equation in the equation file that is used to generate the right-side expression for the new equation. Parameter p_1 is either e_1 or e_1/t_1 which contains left-side variable e_1 and truncation value t_1, if t_1 is present. Truncation value t_1 must be an integer. Parameter e_2 is the left-side variable for the new equation. Variable e_1 contained in parameter p_1 can be the same variable as parameter e_2.

Options, O_j, j=1,2,...,m, specify different ways to affect the reduction of an expression. If two or more options occur in a REDUCEQN call, they are separated from each other by slashes (/). The constant OMEGA cannot occur in any option in a REDUCEQN call.

Processing Achieved

If the equation file does not contain an equation for variable e_1 contained in parameter p_1, an equation is not created and the equation file is not changed; but if there is an equation for e_1 in the equation file, a new equation is created and entered into the equation file. Parameter e_2 is the left-side variable for the new equation. The right-side expression for the new equation is produced by reducing a copy of the right-side expression of the equation for e_1. The equation for e_2 is entered into the equation file after reduction of the expression for e_1 has been completed. If e_1 and e_2 are the same variable, the equation for e_2 replaces the equation for e_1 that was used to generate the new equation.

The reduction of a copy of the expression for e_1 is achieved in three steps: Expansion, Simplification, and Factorization. The processing accomplished by each of these steps is summarized as follows:

1. Expansion

   a. Apply DeMorgan's rules to the factored expression.

   b. Apply the identities P+OMEGA = OMEGA, P*OMEGA = P, P*/OMEGA = /OMEGA, and P+/OMEGA = P to the factored expression produced by step 1a.

   c. Expand the factored expression produced by step 1b by the distributive law and apply the identities P*/P = /OMEGA, P*/OMEGA = /OMEGA, P+/OMEGA = P, and P*P = P to obtain a disjunctive normal form of the expression.

2. Simplification

   Apply the identity P + P*Q = P to the disjunctive normal form expression produced by expansion.

3. Factorization

   Factor the disjunctive normal form of the expression produced by simplification to create a factored form of the expression.

In the last part of expansion (step 1c), terms may be deleted from the expression due to truncation value t_1, and variables may be deleted from terms in the expression due to delete complement or delete noncomplement options (see this procedure, Options). If no term or variable is deleted, the disjunctive normal form of the expression produced by step 1c of expansion is equivalent to the factored form of the expression produced by step 1b of expansion, i.e., the expression used to begin step 1c. However, if any terms or variables are deleted, the expression produced by step 1c is usually not equivalent to the expression at the beginning of step 1c. The expression produced by the other parts of expansion (steps 1a and 1b) and by the simplification and factorization steps is always equivalent to the expression at the beginning of that part or step. Equivalence, if it is lost at all, is lost in step 1c of expansion.

The remainder of the description of the processing achieved by the REDUCEQN procedure is divided into three sections corresponding to the three steps in the reduction process. The example expression

B*(/C+D*(E+F*G)+B*E*H)+I*(E*/(/D*(/H+C)*OMEGA)+(/C+C*/I)) (4-9)

is used to show the effect of the processing described in each section.

EXPANSION

In the first part of expansion (step 1a), DeMorgan's rules are applied to a factored expression to produce an equivalent factored expression wherein all remaining complement operators apply to single variables or constants. After the application of DeMorgan's rules, every variable in the expression can be recognized as a complement or a noncomplement variable. This distinction is maintained throughout the remainder of the reduction process, and it is a distinction that is used by some of the options that can occur in a REDUCEQN call.

Consider the effect of step 1a of expansion on Exp. 4-9. DeMorgan's rules are applied to eliminate the one complement operator in the expression that is not applied to a single variable. The subexpression

/(/D*(/H+C)*OMEGA)

is changed by DeMorgan's rules into the subexpression

(D+(H*/C)+/OMEGA)

which produces the factored expression

B*(/C+D*(E+F*G)+B*E*H)+I*(E*(D+(H*/C)+/OMEGA)+(/C+C*/I)). (4-10)

In the second part of expansion (step 1b), the identities P+OMEGA = OMEGA, P*OMEGA = P, P*/OMEGA = /OMEGA, and P+/OMEGA = P are applied repeatedly to the factored expression produced by step 1a until an equivalent factored expression that does not contain any occurrences of the constants OMEGA or /OMEGA is obtained. The identities must be applied repeatedly because two of them, P+OMEGA = OMEGA and P*/OMEGA = /OMEGA, introduce occurrences of the constants OMEGA and /OMEGA which must also be eliminated.

If the expression is reduced to OMEGA or /OMEGA by step 1b, the remainder of the reduction process is bypassed and the right-side expression of the new equation is OMEGA or /OMEGA, respectively.

If the expression is not reduced to OMEGA or /OMEGA by step 1b, the maximum number of terms that can be generated by the last part of expansion (step 1c) is known. This number is determined after the identities of step 1b have been applied, and is printed as part of the printed output for expansion (see this procedure, Printed Output).

Consider the effect of step 1b of expansion on Exp. 4-10. The identity P+/OMEGA = P is applied to remove the constant /OMEGA and produce the factored expression

$$B*(/C+D*(E+F*G)+B*E*H)+I*(E*(D+(H*/C))+(/C+C*/I)). \quad (4-11)$$

In the last part of expansion (step 1c), the distributive law

$$P*(Q+R) = P*Q + P*R$$

is repeatedly applied to the factored expression produced by step 1b to produce a disjunctive form of the expression. Also, terms that contain complementary variables are deleted from the expression using the identities P*/P = /OMEGA, P*/OMEGA = /OMEGA, and P+/OMEGA = P, and all occurrences of a repeated variable in a term except one are deleted using the identity P*P = P to obtain a disjunctive normal form of the expression.

Consider the effect of step 1c of expansion on Exp. 4-11. The distributive law is used to expand the expression into a disjunctive form

$$B*/C + B*D*E + B*D*F*G + B*B*E*H + I*E*D + I*E*H*/C + I*/C + I*C*/I. \quad (4-12)$$

Then, the identities P*/P = /OMEGA, P*/OMEGA = /OMEGA, and P+/OMEGA = P are used to delete the term I*C*/I from Exp. 4-12 and the identity P*P = P is used to delete one occurrence of the variable B from the term B*B*E*H in Exp. 4-12, to produce the disjunctive normal form of the expression

$$B*/C + B*D*E + B*D*F*G + B*E*H + I*E*D + I*E*H*/C + I*/C. \quad (4-13)$$

SIMPLIFICATION

In the simplification step of the reduction process. subsuming terms are deleted from the disjunctive normal form expression produced by expansion using the identity P + P*Q = P.

The deletion of variables or terms in step 1c of expansion cannot cause a subsuming term, which would normally be deleted during the simplification step of the reduction process, to be retained in the reduced expression. The subsuming term may be deleted during expansion instead of simplification, but one way or another it will not survive these two steps of the reduction process.

Consider the effect of simplification on Exp. 4-13 produced by expansion. The term I*E*H*/C, which subsumes the term I*/C, is deleted from Exp. 4-13 using the identity P + P*Q = P to produce the disjunctive normal form of the expression

$$B*/C + B*D*E + B*D*F*G + B*E*H + I*E*D + I*/C. \quad (4-14)$$

FACTORIZATION

A rigorous description of factorization is beyond the scope and purpose of this manual. Furthermore, it is not necessary to know exactly how an expression is factored in order to write and execute SETS user programs. It is enough to know that factorization produces a factored expression which, when expanded by the distributive law, yields the disjunctive normal form that was used to produce the factored expression. A summary description of factorization is presented in APPENDIX C to provide some knowledge of the factoring that occurs during the execution of a REDUCEQN call.

Consider the effect of factorization on Exp. 4-14 produced by simplification. Factorization produces the final factored form of the expression

B*(/C+E*(H+D)+D*F*G)+I*(/C+D*E). (4-15)

Options

If the REDUCEQN call contains options (form b), variables in the expression may be replaced by OMEGA or /OMEGA prior to reduction, excepted from counting toward truncation value t_1, or deleted from terms in the expression after it has been expanded.

Options that can occur in a REDUCEQN call are the omega option, the phi option, the except complement option, the except noncomplement option, the delete complement option, and the delete noncomplement option. Any combination of these options can occur in a REDUCEQN call, and they can occur in any order. Moreover, a REDUCEQN call can contain two or more occurrences of the same kind of option, e.g., two omega options. The effect, however, is the same as if all of the separate options of the same kind are collected into a single option of that kind.

A variable cannot occur more than once in the same kind of option in a REDUCEQN call. Furthermore, a variable that occurs in an omega or a phi option in a REDUCEQN call can occur only once in all of the options in the call. However, a variable that does not occur in an omega or a phi option can occur in all of the other options in a REDUCEQN call. For example, a variable can occur in both an except complement option and an except noncomplement option. The constant OMEGA cannot occur in any option in a REDUCEQN call.

An omega option has the form:

OMEGA$ $v_1,v_2,\ldots,v_r$

An omega option in a REDUCEQN call identifies variables that are replaced by the constant OMEGA before the expression is reduced. Every occurrence of each v_i, i=1,2,...,r, in the expression is replaced by the constant OMEGA prior to the expansion of the expression.

A phi option has the form:

PHI$ $v_1,v_2,\ldots,v_r$

A phi option in a REDUCEQN call identifies variables that are to be replaced by the constant /OMEGA before the expression is reduced. Every occurrence of each v_i, i=1,2,...,r, in the expression is replaced by the constant /OMEGA prior to the expansion of the expression.

An except complement option has one of the forms:

a. EXCEPTCMP$

b. EXCEPTCMP$ $v_1,v_2,...,v_r$

An except complement option in a REDUCEQN call is used in conjunction with truncation value t_1. The complement variables specified in an except complement option do not count toward the truncation value. If the except complement option does not contain a variable list (form a), every complement variable is excepted from the counting process. If a variable list is present (form b), only the complement variables corresponding to the noncomplement variables on the list are excepted from the counting process.

An except noncomplement option has one of the forms:

a. EXCEPTNONCMP$

b. EXCEPTNONCMP$ $v_1,v_2,...,v_r$

An except noncomplement option in a REDUCEQN call is like the except complement option, but it is used to except noncomplement variables from counting toward truncation value t_1.

A delete complement option has one of the forms:

a. DELETECMP$

b. DELETECMP$ $v_1,v_2,...,v_r$

A delete complement option is used to delete complement variables from the expression. The complement variables specified in a delete complement option are deleted from the terms of the expression after the disjunctive normal form has been produced in step 1c of expansion. Terms that contain complementary variables are identified and deleted before any variables are deleted from the terms. If truncation value t_1 is specified, variables specified in delete complement options do not count toward the truncation value.

If the delete complement option does not contain a variable list (form a), every occurrence of every complement variable is deleted from the terms of the disjunctive normal form of the expression. If a variable list is present (form b), only the complement variables corresponding to the noncomplement variables in the list are deleted from the expression. If every variable in a term is deleted, the term and, therefore, the expression is OMEGA.

A delete noncomplement option has one of the forms:

a. DELETENONCMP$

b. DELETENONCMP$ $v_1,v_2,\ldots,v_r$

A delete noncomplement option is like the delete complement option, but it is used to delete noncomplement variables from the expression.

Printed Output

If the equation file does not contain an equation for variable e_1 contained in parameter p_1, the message

THERE IS NO EQUATION FOR e_1

is printed, and execution of the REDUCEQN call is terminated. If the equation file contains an equation for e_1, the printed output produced is for reduction of the expression for e_1.

There are three parts to the printed output produced by the execution of a REDUCEQN call. They correspond to the three steps of the reduction process and they can be described using Exp. 4-9 from the previous section.

Execution of the expansion step of the reduction process for Exp. 4-9 produces the following printed output:

```
THE MAXIMUM NUMBER OF TERMS THAT CAN BE
GENERATED BY EXPANSION IS                8.

TERMS GENERATED BY EXPANSION
        2 TERMS CONTAIN  2 VARIABLES
        3 TERMS CONTAIN  3 VARIABLES
        2 TERMS CONTAIN  4 VARIABLES
    TOTAL TERMS GENERATED        7.
EXPANSION TOOK     .007 SECONDS
```

The first part of the printed output produced by expansion indicates the maximum number of terms that can be generated. The maximum number of terms that can be generated is computed at the beginning of step 1c of expansion. For Exp. 4-9, the maximum number of terms that can be generated is 8.

The second part of the printed output produced by expansion is an accounting of the terms generated. This accounting shows a breakdown of the number of terms generated with respect to the number of variables in each term, and a final message indicates the total number of terms generated. The messages for the breakdown of terms with respect to the number of variables per term are printed according to an increasing number of variables per term. For Exp. 4-9, expansion generates 2 terms that contain 2 variables, 3 terms that contain 3 variables, and 2 terms that contain 4 variables. A total of 7 terms are generated.

The number of terms generated by expansion is always less than or equal to the maximum number of terms that can be generated. If these two numbers are different, they differ by the number of terms deleted. Terms are deleted because they contain complementary variables or because they exceed truncation value t_1, if t_1 is specified. If all of the terms are deleted, the expression is /OMEGA. When this happens, the breakdown of terms by number of variables per term is not printed. Instead, the message

```
THE EXPRESSION IS /OMEGA
```

is printed and the total number of terms generated by expansion is 1. For Exp. 4-9, a maximum of 8 terms can be generated, and 7 terms are generated. One term is deleted because it contains complementary variables I and /I.

Following the accounting of the terms produced by expansion, if any terms were deleted because they contained more variables than allowed by truncation value t_1, the message

```
THERE WERE TERMS DELETED BECAUSE OF
THE TRUNCATION VALUE OF t1
```

is printed.

During expansion, every variable of a term may be deleted if delete options occur in the REDUCEQN call. If every variable of a term is deleted, the term and, therefore, the expression is OMEGA. When this happens, the breakdown of terms with respect to number of variables per term is not printed. Instead, the message

```
EVERY VARIABLE IN SOME TERM WAS DELETED
THE EXPRESSION IS OMEGA
```

is printed, and the total number of terms generated by expansion is 1.

The final printed output produced by expansion is a message indicating the time required for expansion.

Execution of the simplification step of the reduction process for Exp. 4-9 produces the following printed output:

```
        TERMS RETAINED BY SIMPLIFICATION
              2 TERMS CONTAIN  2 VARIABLES
              3 TERMS CONTAIN  3 VARIABLES
              1 TERMS CONTAIN  4 VARIABLES
      TOTAL TERMS RETAINED          6.
SIMPLIFICATION TOOK          .003 SECONDS
```

The printed output produced by simplification is an accounting of the terms retained by simplification. It is similar to the accounting produced by expansion. It shows a breakdown of the terms retained by simplification and a final message that indicates the total number of terms retained. For Exp. 4-9, all of the 2 and 3 variable terms and all but one of the 4 variable terms are retained. A total of 6 terms is retained by simplification.

The number of terms retained by simplification is always less than or equal to the number of terms generated by expansion. These two numbers differ by the number of subsuming terms that are deleted using the identity P + P*Q = P.

The final printed output produced by simplification is a message indicating the time required for simplification.

Execution of the factorization step of the reduction process for Exp. 4-9 produces the following printed output:

```
FACTORIZATION TOOK     .018 SECONDS
```

The output for factorization is a message indicating the time required for factorization.

Example

Consider SETS user program:

```
PROGRAM$ REDUCEQN-EX.
   REDUCEQN (NO-EQN, NEW-EQN).
   PRTEQN (NEW-EQN).
   OMEGA-EQN = X1 + X2 + /(X3*/OMEGA).
   REDUCEQN (OMEGA-EQN, NEW-OMG-EQN).
   PRTEQN (NEW-OMG-EQN).
   EQN-1 = /(/X1 + X2*OMEGA)*(X3 + /X1 + X4*(X1 + X5*(X2 + /X3))).
   REDUCEQN (EQN-1, EQN-1).
   PRTEQN (EQN-1).
   EQN-2 = (X1 + X4*X5 + X6)*(X2 + /X2*X5)*(X3 + X2*X4).
   PRTEQNDNF (EQN-2).
   REDUCEQN (EQN-2, EQN-2-OMG-X2* OMEGA$ X2).
   PRTEQNDNF (EQN-2-OMG-X2).
   REDUCEQN (EQN-2/2, EQN-2-ORD-2).
   PRTEQN (EQN-2-ORD-2).
   EQN-3 = (X1 + X3 + /X6)*(X4 + /X4*X6)*
           /X2*(X5 + /X5*/X3 + /X5*X3*/X7).
   PRTEQNDNF (EQN-3).
   REDUCEQN (EQN-3/2, EQN-3-ORD-2-EXC* EXCEPTNONCMP$/
                                       EXCEPTCMP$ X4, X7).
   PRTEQNDNF (EQN-3-ORD-2-EXC).
   REDUCEQN (EQN-3, EQN-3-DLTCMP* DELETECMP$).
   PRTEQNDNF (EQN-3-DLTCMP).
```

The equation file is empty when execution of REDUCEQN-EX begins. Then, a REDUCEQN call shows the equation file does not contain an equation for NO-EQN, and the following PRTEQN call shows an equation was not created for NEW-EQN. The printed output produced by these two procedures is as follows:

```
REDUCEQN (NO-EQN, NEW-EQN).

          THERE IS NO EQUATION FOR NO-EQN

     STATEMENT EXECUTION REQUIRED      .001 SECONDS FOR REDUCEQN


PRTEQN (NEW-EQN).

          THERE IS NO EQUATION FOR NEW-EQN

     STATEMENT EXECUTION REQUIRED      .001 SECONDS FOR PRTEQN
```

The next statement in the SETS user program is an equation statement that enters an equation for OMEGA-EQN into the equation file. Then, a REDUCEQN call creates an equation for NEW-OMG-EQN. (The printed output produced by the REDUCEQN call shows the right-side expression of the equation for OMEGA-EQN reduces to OMEGA during expansion.) The following PRTEQN call shows the right-side expression of the new equation is OMEGA. The printed output produced by these three statements is as follows:

```
          OMEGA-EQN = X1 + X2 + /( X3 * /OMEGA )
                               1           1

     STATEMENT EXECUTION REQUIRED      .004 SECONDS FOR LDTMPBLK


REDUCEQN (OMEGA-EQN, NEW-OMG-EQN).

               THE EXPRESSION IS OMEGA
               TOTAL TERMS GENERATED        1.
          EXPANSION TOOK      .001 SECONDS.

     STATEMENT EXECUTION REQUIRED      .002 SECONDS FOR REDUCEQN


PRTEQN (NEW-OMG-EQN).

          NEW-OMG-EQN = OMEGA

     STATEMENT EXECUTION REQUIRED      .002 SECONDS FOR PRTEQN
```

The next statement in the SETS user program is an equation statement that enters an equation for EQN-1 into the equation file. Then, a REDUCEQN call creates a reduced equation for EQN-1 which replaces the equation for EQN-1 in the equation file. This REDUCEQN call does not contain a truncation value or any options. Thus, the right-side expression of the created equation is equivalent to the right-side expression of the original equation for EQN-1. A PRTEQN call prints the new equation for EQN-1. The printed output produced by these three statements is as follows:

```
          EQN-1 = /( /X1 + X2 * OMEGA ) * ( X3 + /X1 + X4 * ( X1 +
                  1                       1   1                2

          X5 * ( X2 + /X3 ) ) )
               3          3 2 1

     STATEMENT EXECUTION REQUIRED      .004 SECONDS FOR LDTMPBLK


REDUCEQN (EQN-1, EQN-1).

               THE MAXIMUM NUMBER OF TERMS THAT CAN BE
               GENERATED BY EXPANSION IS          5.

               TERMS GENERATED BY EXPANSION
                    2 TERMS CONTAIN  3 VARIABLES
                    1 TERMS CONTAIN  5 VARIABLES
               TOTAL TERMS GENERATED        3.
          EXPANSION TOOK      .007 SECONDS.

               TERMS RETAINED BY SIMPLIFICATION
                    2 TERMS CONTAIN  3 VARIABLES
               TOTAL TERMS RETAINED         2.
          SIMPLIFICATION TOOK      .002 SECONDS.
```

```
          FACTORIZATION TOOK       .004 SECONDS.

     STATEMENT EXECUTION REQUIRED       .016 SECONDS FOR REDUCEQN


PRTEQN (EQN-1).

          EQN-1 = X1 * /X2 * ( X4 + X3 )
                             1        1

     STATEMENT EXECUTION REQUIRED       .002 SECONDS FOR PRTEQN
```

The next statement in the SETS user program is an equation statement that enters an equation for EQN-2 into the equation file. Then, a PRTEQNDNF call prints the nine term disjunctive normal form of EQN-2 produced by expansion. Results obtained by subsequent REDUCEQN calls that process EQN-2 are easier to understand if they can be compared to a disjunctive normal form of all of the terms of EQN-2. The printed output produced by the equation statement and the PRTEQNDNF call is as follows:

```
          EQN-2 = ( X1 + X4 * X5 + X6 ) * ( X2 + /X2 * X5 ) * ( X3 +
                  1                    1   1               1   1

          X2 * X4 )
                  1

     STATEMENT EXECUTION REQUIRED       .004 SECONDS FOR LDTMPBLK


PRTEQNDNF (EQN-2).

               SUBSUMING TERMS MAY OCCUR IN THE
               EQUATION FOR EQN-2

               THE MAXIMUM NUMBER OF TERMS THAT CAN BE
               GENERATED BY EXPANSION IS         12.

               TERMS GENERATED BY EXPANSION
                    5 TERMS CONTAIN  3 VARIABLES
                    4 TERMS CONTAIN  4 VARIABLES
               TOTAL TERMS GENERATED        9.
          EXPANSION TOOK       .007 SECONDS.


                                   * * * * VARIABLE OCCURRENCE TABLE * * * *

                    NONCOMPLEMENT        NUMBER OF          COMPLEMENT        NUMBER OF
                      VARIABLE          OCCURRENCES          VARIABLE        OCCURRENCES

                      X1                    3
                      X2                    6               /X2                  3
                      X3                    6
                      X4                    5
                      X5                    5
                      X6                    3

               THERE ARE   7 DIFFERENT VARIABLES IN THE
               EQUATION FOR EQN-2
```

```
 TERM   NUMBER OF
NUMBER  VARIABLES

                    EQN-2 =

   1        3      X1 * X2 * X3 +

   2        3      X2 * X3 * X6 +

   3        3      X1 * X2 * X4 +

   4        3      X2 * X4 * X5 +

   5        3      X2 * X4 * X6 +

   6        4      X2 * X3 * X4 * X5 +

   7        4      X1 * X3 * X5 * /X2 +

   8        4      X3 * X4 * X5 * /X2 +

   9        4      X3 * X5 * X6 * /X2

STATEMENT EXECUTION REQUIRED      .019 SECONDS FOR PRTEQNDNF
```

The next statement in the SETS user program is a REDUCEQN call that creates an equation for EQN-2-OMG-X2. It contains an omega option that causes every occurrence of X2 in the copy of the right-side expression of the equation for EQN-2 to be replaced by OMEGA before the expression is reduced. A PRTEQNDNF call shows the five terms in the reduced equation for EQN-2-OMG-X2. The printed output produced by the REDUCEQN call and part of the printed output produced by the PRTEQNDNF call is as follows:

```
REDUCEQN (EQN-2, EQN-2-OMG-X2* OMEGA$ X2).

          THE MAXIMUM NUMBER OF TERMS THAT CAN BE
          GENERATED BY EXPANSION IS          6.

          TERMS GENERATED BY EXPANSION
               5 TERMS CONTAIN  2 VARIABLES
               1 TERMS CONTAIN  3 VARIABLES
          TOTAL TERMS GENERATED         6.
     EXPANSION TOOK       .005 SECONDS.

           TERMS RETAINED BY SIMPLIFICATION
                5 TERMS CONTAIN  2 VARIABLES
           TOTAL TERMS RETAINED         5.
     SIMPLIFICATION TOOK       .002 SECONDS.

     FACTORIZATION TOOK       .012 SECONDS.

STATEMENT EXECUTION REQUIRED       .020 SECONDS FOR REDUCEQN


PRTEQNDNF (EQN-2-OMG-X2).

          •

          •

          •
```

```
 TERM   NUMBER OF
NUMBER  VARIABLES

                      EQN-2-OMG-X2 =

   1        2         X3 * X6 +

   2        2         X1 * X3 +

   3        2         X4 * X6 +

   4        2         X4 * X5 +

   5        2         X1 * X4

STATEMENT EXECUTION REQUIRED      .014 SECONDS FOR PRTEQNDNF
```

The next statement in the SETS user program is a REDUCEQN call that creates an equation for EQN-2-ORD-2. This call contains truncation value 2. Since every term of EQN-2 contains more than 2 variables, every term is deleted during expansion. Then, a PRTEQN call shows the right-side expression of the equation for EQN-2-ORD-2 is /OMEGA. The printed output produced by these two procedures is as follows:

```
REDUCEQN (EQN-2/ 2, EQN-2-ORD-2).

                THE MAXIMUM NUMBER OF TERMS THAT CAN BE
                GENERATED BY EXPANSION IS         12.

                TERMS GENERATED BY EXPANSION
                THE EXPRESSION IS /OMEGA
                TOTAL TERMS GENERATED        1.

                THERE WERE TERMS DELETED BECAUSE OF
                THE TRUNCATION VALUE OF  2

          EXPANSION TOOK      .004 SECONDS.

STATEMENT EXECUTION REQUIRED      .005 SECONDS FOR REDUCEQN


PRTEQN (EQN-2-ORD-2).

          EQN-2-ORD-2 = /OMEGA

STATEMENT EXECUTION REQUIRED      .002 SECONDS FOR PRTEQN
```

The next statement in the SETS user program is an equation statement that enters an equation for EQN-3 into the equation file. Then, a PRTEQNDNF call prints the thirteen term disjunctive normal form of EQN-3 produced by expansion. Results obtained by subsequent REDUCEQN calls that process EQN-3 are easier to understand if they can be compared to a disjunctive normal form of all of the terms of EQN-3. The printed output produced by the equation statement and part of the printed output produced by the PRTEQNDNF call is as follows:

```
EQN-3 = ( X1 + X3 + /X6 ) * ( X4 + /X4 * X6 ) * /X2 * (
          1              1   1              1        1

X5 + /X5 * /X3 + /X5 * X3 * /X7 )
                              1

STATEMENT EXECUTION REQUIRED      .005 SECONDS FOR LDTMPBLK
```

```
PRTEQNDNF (EQN-3).

              SUBSUMING TERMS MAY OCCUR IN THE
              EQUATION FOR EQN-3

              THE MAXIMUM NUMBER OF TERMS THAT CAN BE
              GENERATED BY EXPANSION IS         18.

              TERMS GENERATED BY EXPANSION
                    3 TERMS CONTAIN  4 VARIABLES
                    5 TERMS CONTAIN  5 VARIABLES
                    4 TERMS CONTAIN  6 VARIABLES
                    1 TERMS CONTAIN  7 VARIABLES
              TOTAL TERMS GENERATED       13.
        EXPANSION TOOK      .009 SECONDS.

          •

          •

          •

         TERM   NUMBER OF
        NUMBER  VARIABLES

                          EQN-3 =

            1        4      X1 * X4 * X5 * /X2 +

            2        4      X3 * X4 * X5 * /X2 +

            3        4      X4 * X5 * /X2 * /X6 +

            4        5      X1 * X4 * /X2 * /X3 * /X5 +

            5        5      X4 * /X2 * /X3 * /X5 * /X6 +

            6        5      X3 * X4 * /X2 * /X5 * /X7 +

            7        5      X1 * X5 * X6 * /X2 * /X4 +

            8        5      X3 * X5 * X6 * /X2 * /X4 +

            9        6      X1 * X3 * X4 * /X2 * /X5 * /X7 +

           10        6      X3 * X4 * /X2 * /X5 * /X6 * /X7 +

           11        6      X1 * X6 * /X2 * /X3 * /X4 * /X5 +

           12        6      X3 * X6 * /X2 * /X4 * /X5 * /X7 +

           13        7      X1 * X3 * X6 * /X2 * /X4 * /X5 * /X7

    STATEMENT EXECUTION REQUIRED      .028 SECONDS FOR PRTEQNDNF
```

The next statement in the SETS user program is a REDUCEQN call that creates an equation for EQN-3-ORD-2-EXC. This call contains truncation value 2, and it also contains an except noncomplement and an except complement option. The except noncomplement option does not contain a variable list, so it means that none of the noncomplement variables in the expression counts toward the truncation value. The except complement option contains the variables X4 and X7 which means that the complement variables /X4 and /X7 in the expression do not count toward the truncation value. Expansion of EQN-3 by this REDUCEQN

call produces only nine terms because, even with the exceptions, four terms in the full expansion of EQN-3 exceed truncation value 2. Of the nine terms produced by expansion, seven are retained by simplification. A call of PRTEQNDNF prints the seven terms in the equation for EQN-3-ORD-2-EXC. The printed output produced by the REDUCEQN call and part of the output produced by the PRTEQNDNF call is as follows:

```
REDUCEQN (EQN-3/ 2, EQN-3-ORD-2-EXC* EXCEPTNONCMP$ / EXCEPTCMP$ X4, X7).

                THE MAXIMUM NUMBER OF TERMS THAT CAN BE
                GENERATED BY EXPANSION IS        18.

                TERMS GENERATED BY EXPANSION
                     3 TERMS CONTAIN  4 VARIABLES
                     3 TERMS CONTAIN  5 VARIABLES
                     2 TERMS CONTAIN  6 VARIABLES
                     1 TERMS CONTAIN  7 VARIABLES
                TOTAL TERMS GENERATED        9.

                THERE WERE TERMS DELETED BECAUSE OF
                THE TRUNCATION VALUE OF  2

          EXPANSION TOOK      .009 SECONDS.

                TERMS RETAINED BY SIMPLIFICATION
                     3 TERMS CONTAIN  4 VARIABLES
                     3 TERMS CONTAIN  5 VARIABLES
                     1 TERMS CONTAIN  6 VARIABLES
                TOTAL TERMS RETAINED        7.
          SIMPLIFICATION TOOK      .004 SECONDS.

          FACTORIZATION TOOK      .019 SECONDS.

     STATEMENT EXECUTION REQUIRED      .035 SECONDS FOR REDUCEQN


PRTEQNDNF (EQN-3-ORD-2-EXC).

               •

               •

               •

           TERM   NUMBER OF
          NUMBER  VARIABLES

                                EQN-3-ORD-2-EXC =

             1        4        X4 * X5 * /X2 * /X6 +

             2        4        X3 * X4 * X5 * /X2 +

             3        4        X1 * X4 * X5 * /X2 +

             4        5        X3 * X5 * X6 * /X2 * /X4 +

             5        5        X1 * X5 * X6 * /X2 * /X4 +

             6        5        X3 * X4 * /X2 * /X5 * /X7 +

             7        6        X3 * X6 * /X2 * /X4 * /X5 * /X7

     STATEMENT EXECUTION REQUIRED      .020 SECONDS FOR PRTEQNDNF
```

The next statement in the SETS user program is a REDUCEQN call that creates an equation for EQN-3-DLTCMP. This call contains a delete complement option without a variable list. This means every complement variable is deleted from the terms of the disjunctive normal form of the expression at the end of expansion. Since no term in the full thirteen term expansion of EQN-3 contains all complement variables, no term is deleted as a result of the delete complement option. The complement variables, however, are deleted from each of the thirteen terms at the end of expansion. Simplification retains only three of these terms. A call of PRTEQNDNF prints the three terms of EQN-3-DLTCMP. The printed output produced by the REDUCEQN call and part of the printed output produced by the PRTEQNDNF call is as follows:

```
REDUCEQN (EQN-3, EQN-3-DLTCMP* DELETECMP$ ).

                THE MAXIMUM NUMBER OF TERMS THAT CAN BE
                GENERATED BY EXPANSION IS          18.

                TERMS GENERATED BY EXPANSION
                      1 TERMS CONTAIN  1 VARIABLES
                      6 TERMS CONTAIN  2 VARIABLES
                      6 TERMS CONTAIN  3 VARIABLES
                TOTAL TERMS GENERATED        13.
           EXPANSION TOOK       .007 SECONDS.
                TERMS RETAINED BY SIMPLIFICATION
                      1 TERMS CONTAIN  1 VARIABLES
                      2 TERMS CONTAIN  2 VARIABLES
                TOTAL TERMS RETAINED         3.
           SIMPLIFICATION TOOK       .002 SECONDS.

           FACTORIZATION TOOK       .007 SECONDS.

     STATEMENT EXECUTION REQUIRED       .018 SECONDS FOR REDUCEQN


PRTEQNDNF (EQN-3-DLTCMP).

                •

                •

                •

            TERM    NUMBER OF
           NUMBER   VARIABLES

                                EQN-3-DLTCMP =

                1         1      X4 +

                2         2      X3 * X6 +

                3         2      X1 * X6

     STATEMENT EXECUTION REQUIRED       .011 SECONDS FOR PRTEQNDNF
```

4.1.6 Compute Term Value

The Compute Term Value procedure is used to compute and print term values for equations in the equation file. For each equation specified in the procedure call, a disjunctive normal form of the right-side expression of the equation is obtained by expansion and values are computed for each term in the expression. Parameters in the procedure call specify what computations to make and which variable values to use in the computations. (Variable values must be nonnegative and every computation produces nonnegative term values.) In addition, for each computation specified in the procedure call, an extreme value is determined from the term values for that computation. The right-side expression of the equation is printed in a disjunctive normal form, and the values for each term are printed with the term. The extreme values are printed following the printing of the equation and the term values. For example, suppose the equation file contains the equation

A=B*(C+D*F)+D*(E+C*(F+G)).

Also, suppose the value block file contains value block VB1 which contains variable values:

B=5
C=8
D=2
E=5
F=1
G=8

Execution of procedure call statement

COMTRMVAL (SUM, VB1* A)

computes a value for each term in a disjunctive normal form of the right-side expression of the equation for A and prints the term values, the equation for A, and the extreme value in the form:

TERM NUMBER	SUM OF TERM	
		A =
1	1.8000E+01	C * D * G +
2	1.3000E+01	B * C +
3	1.1000E+01	C * D * F +
4	8.0000E+00	B * D * F +
5	7.0000E+00	D * E

THE MINIMUM TERM VALUE FOR COMPUTATION 1 IS 7.0000E+00

For this example, a disjunctive normal form of the right-side expression of the equation for A is obtained by expansion, and a value is computed for each term in the expression using variable values from value block VB1. For computation type SUM, term value is the sum of the variable values for the variables in the term. The extreme value, which for SUM is the minimum term value, is determined from the term values. Terms are sorted in order of decreasing value and the term values, the equation for A, and the extreme value are printed.

Procedure Call

A call of the Compute Term Value procedure has one of the forms:

a. COMTRMVAL (q_1,vb_1,q_2,vb_2,...,q_s,vb_s* p_1,p_2,...,p_n).

b. COMTRMVAL (DECREASE$ q_1,vb_1,q_2,vb_2,...,q_s,vb_s* p_1,p_2,...,p_n).

c. COMTRMVAL (INCREASE$ q_1,vb_1,q_2,vb_2,...,q_s,vb_s* p_1,p_2,...,p_n).

Each parameter pair, q_k and vb_k, k=1,2,...,s, specifies a computation. Each q_k is either c_k or c_k/l_k which contains computation type c_k and truncation value l_k, if l_k is present. The corresponding vb_k identifies the value block which contains the nonnegative variable values to be used in the computation specified by q_k. At least one computation must be specified in a COMTRMVAL call, and no more than ten computations can be specified, ($1 \leq s \leq 10$).

Parameters, p_i, i=1,2,...,n, specify the equations to be processed. Each p_i is either e_i or e_i/t_i which contains left-side variable e_i and truncation value t_i, if t_i is present. Truncation value t_i must be an integer. There must be at least one p_i parameter, ($1 \leq n$).

Parameters DECREASE and INCREASE control the order in which terms are sorted for printing. Only one sort parameter can occur in a COMTRMVAL call and, if one occurs, it must be the first parameter in the call.

Processing Achieved

Term values are computed in both the Compute Term Value procedure and the Truncate on Term Value procedure. The computations that can be specified in the two procedures are the same; the way term values are used in the two procedures is quite different. Compute Term Value computes term values and prints them. Truncate on Term Value computes term values and uses them to determine whether or not to delete terms from an expression during expansion.

There are nine different computations. A computation is recognized by its computation type c_k, which specifies a rule for computing a term value as a function of the values for the variables in the term. Each computation type is listed in Table 4.1 together with a description of the corresponding computation.

Variable values used in computations are obtained from value blocks stored in the value block file. A distinction is made between a variable and its value; variable v_i has value $V(v_i)$. When variable values are known to be probabilities, V is replaced by P and $V(v_i)$ becomes $P(v_i)$.

Variable values are restricted to make each computation either monotone decreasing or monotone increasing. A computation is monotone decreasing (increasing) if, for any ordering of the variables in the term, the value computed through j+1 variables of the term is $\leq$ ($\geq$) the value computed thru j variables of the term. The initial value of a term is the identity element for the computation: 0 for SUM, COUNT, DISTINCT-SUM, DISTINCT-COUNT, and MAXIMUM; 1 for PRODUCT, DISTINCT-PRODUCT, and PROBABILITY; and 1×10^{100} for MINIMUM. PROBABILITY and MINIMUM are monotone decreasing computations, and all other computations are monotone increasing.

Table 4.1 Computation Types

	Computation Type (c_k)	Term Value
1.	SUM	Sum of the variable values.
2.	PRODUCT	Product of the variable values.
3.	COUNT	Number of variables.
4.	DISTINCT-SUM	Sum of the different variable values.
5.	DISTINCT-PRODUCT	Product of the different variable values.
6.	DISTINCT-COUNT	Number of variables with different values.
7.	MAXIMUM	Maximum variable value.
8.	MINIMUM	Minimum variable value.
9.	PROBABILITY	Product of the variable probabilities.

Variables cannot be assigned negative values, i.e., value blocks contain only nonnegative values. This condition is tested when value blocks are created. Also, nonnegative variable values are restricted with respect to computation type. These restrictions cannot be tested until a computation type and value block are paired together in a procedure call. The restriction on nonnegative variable values for each computation type is shown as a value range in Table 4.2.

Complement variables cannot be assigned values directly, i.e., value blocks do not contain values for complement variables. The value used for a complement variable is determined by computation type. For computation type PROBABILITY, the value assigned to complement variable $/v_i$ is determined from the value for noncomplement variable v_i using the relationship $P(/v_i) = 1-P(v_i)$. For all other computation types, the value assigned to every complement variable is the identity element for the computation. Thus, complement variables do not change term value for any computation type except PROBABILITY. Complement variable value assignments for each computation type are shown in Table 4.2.

Because of the restrictions on variable values, each computation type produces term values which lie in the same range as the variable values. One consequence is that all term values are nonnegative.

Table 4.2 Variable Value Restrictions

Computation Type (c_k)	Value Range for Noncomplement Variable	Complement Variable Value
1. PROBABILITY	$0 \leq P(v_i) \leq 1$	$1-P(v_i)$
2. MINIMUM	$0 \leq V(v_i)$	1×10^{100}
3. PRODUCT and DISTINCT-PRODUCT	$1 \leq V(v_i)$	1
4. All other computation types	$0 \leq V(v_i)$	0

The nine computations can be shown using a single term

B*/C*D*E

and value blocks VALBLK-1, VALBLK-2, and VALBLK-3 which contain variable values as shown in Table 4.3.

Table 4.3 Value Blocks VALBLK-1, VALBLK-2, and VALBLK-3

Variable	VALBLK-1	VALBLK-2	VALBLK-3
B	2.0×10^{-3}	.6	5
C	1.0×10^{-1}	4.0	5
D	6.0×10^{-2}	7.0	13
E	2.0×10^{-3}	.6	5

Values in VALBLK-1 are in the range $0 \leq V(v_i) \leq 1$, and can be used for every computation type except PRODUCT and DISTINCT-PRODUCT. Values in VALBLK-2 are in the range $0 \leq V(v_i)$, and can be used for every computation type except PRODUCT, DISTINCT-PRODUCT, and PROBABILITY. Values in VALBLK-3 are in the range $1 \leq V(v_i)$, and can be used with every computation type except PROBABILITY. The value for complement variable /C must be determined with regard to computation type before values for B*/C*D*E can be computed. The values for B*/C*D*E for every computation type using the values from all three value blocks, if appropriate, are presented in Table 4.4.

Table 4.4 Values Computed for B*/C*D*E

Computation Type (c_k)	VALBLK-1	VALBLK-2	VALBLK-3
1. SUM	6.40×10^{-2}	8.2	23
2. PRODUCT	-----	-----	325
3. COUNT	4	4	4
4. DISTINCT-SUM	6.20×10^{-2}	7.6	18
5. DISTINCT-PRODUCT	-----	-----	65
6. DISTINCT-COUNT	3	3	3
7. MAXIMUM	6.00×10^{-2}	7.0	13
8. MINIMUM	2.00×10^{-3}	.6	5
9. PROBABILITY	2.16×10^{-7}	-----	-----

Parameters in a COMTRMVAL call that specify the order of sorting terms and the computations to be made must be processed before term values can be computed and printed for any equation. If the first parameter is not a sort parameter or if it is DECREASE, the order for sorting terms is determined to be decreasing. If INCREASE is the first parameter, the order for sorting terms is determined to be increasing.

As each q_k is encountered, the computation type and truncation value for the computation are determined. The computation type is specified by c_k contained in parameter q_k. If truncation value l_k occurs in parameter q_k, the truncation value for the k-th computation is set to l_k. Otherwise, the truncation value is set in accordance with computation type c_k so that no truncation will occur. If c_k is monotone decreasing, the truncation value for the k-th computation is set to 0; and, if c_k is monotone increasing, the truncation value for the k-th computation is set to 1×10^{100}.

As each vb_k is encountered, variable values from value block vb_k are established as the values to be used for the k-th computation.

After the pairs of q_k and vb_k parameters have been processed, term values can be computed for each equation specified in the procedure call. Parameters, p_i, i=1,2,...,n, are processed one at a time from left to right and, as each one is encountered, a disjunctive normal form of the right-side expression of the specified equation is obtained by expansion. Expansion of an expression to obtain a disjunctive normal form for computing and printing term values is the same expansion described briefly in Section 4.1.2, Print Equation in Disjunctive Normal Form and in detail in Section 4.1.5, Reduce Equation. If the equation file does not contain an equation for variable e_i contained in parameter p_i, a message is printed and processing continues with the next parameter.

After expansion, the variables in each value block vb_k specified in the COMTRMVAL call are compared to the variables in the disjunctive normal form of the expression. Each value block must contain a value for every noncomplement variable that occurs in the expression. Also, for every complement variable that occurs in the expression, each value block must contain a value for the corresponding noncomplement variable.

Once the variables in the value blocks have been compared to the variables in the expression, values for every computation specified in a COMTRMVAL call are computed for each term in the expression. A term value is said to exceed the corresponding truncation value if, for monotone decreasing computations, it is less than the truncation value and, for monotone increasing computations, it is greater than the truncation value. A term is deleted if any of its values exceeds the corresponding truncation value.

In addition to the term values that are computed and printed, an extreme value is determined and printed for every computation specified in a COMTRMVAL call. The extreme value for a computation may not be among the printed term values for the computation. This happens when each term having the extreme value for one computation is deleted because it exceeds the truncation value for another computation. The extreme value for a monotone decreasing computation (MINIMUM or PROBABILITY) is the maximum value over all term values for the computation. The extreme value for a monotone increasing computation (SUM, PRODUCT, COUNT, DISTINCT-SUM, DISTINCT-PRODUCT, DISTINCT-COUNT, or MAXIMUM) is the minimum value over all term values for the computation.

For computation type PROBABILITY, the sum of the values for the remaining terms is also computed and printed. This sum is often interpreted as an approximation to the probability of the expression. If the expression is not reduced prior to the COMTRMVAL call, subsuming terms can contribute unnecessarily to the approximation.

After term values have been computed and extreme values determined, the terms that remain are sorted into the appropriate order according to the values for the first computation in the COMTRMVAL call. Then, each term, its values, and the extreme values are printed.

Printed Output

If the equation file does not contain an equation for variable e_i contained in parameter p_i, the message

THERE IS NO EQUATION FOR e_i

is printed and the processing of the parameters continues.

If the equation file contains an equation for e_i, a disjunctive normal form expression for p_i is obtained (p_i is e_i or e_i truncated by t_i), and variables in the expression are compared to variables in each value block specified in the

COMTRMVAL call. Variables in the expression that do not have values in every value block are listed with a message in the form

VARIABLES THAT DO NOT HAVE A VALUE IN EVERY
VALUE BLOCK SPECIFIED IN THIS PROCEDURE CALL

v_1
v_2
.
.
.
v_n

and a SETS user program error occurs (see APPENDIX A, Section 3.3, Numbered Errors, error 67). (Complement variables must have values for the corresponding noncomplement variables.)

There are four parts to the printed output associated with computing and printing values for the terms of the equation. There is the output associated with changing the right-side expression of the equation into a disjunctive normal form; there is a Variable Occurrence Table that contains information about the variables in the printed equation; there is the printed equation and the values for its terms; and there are the extreme values for the computations specified in the call.

The printed output associated with changing the right-side expression into a disjunctive normal form can begin with the warning:

SUBSUMING TERMS MAY OCCUR IN THE
EQUATION FOR e_i

The warning occurs if the equation for e_i was not created by a call of REDUCEQN, TRNTRMVAL, or DLTRM. It means that, even though the right-side expression of the equation may not contain subsuming terms, the absence of subsuming terms in the right-side expression has not been ensured by the execution of one of these procedures in a SETS user program.

The next part of the printed output associated with changing the right-side expression into a disjunctive normal form is the output produced by expansion. A summary description of this output appears in Section 4.1.2, Print Equation in Disjunctive Normal Form, and a detailed description of it appears in Section 4.1.5, Reduce Equation.

Briefly, the printed output produced by expansion indicates the maximum number of terms that can be generated by expansion; a breakdown of the terms generated with respect to the number of variables per term; the total number of terms generated; and the time required for expansion. If the equation was created by a call of REDUCEQN, TRNTRMVAL, or DLTRM and the right-side expression is constant, i.e., OMEGA or /OMEGA, a message indicating the expression is OMEGA or /OMEGA, respectively, replaces the breakdown of terms.

Terms that contain complementary variables and terms that contain more than t_i variables, if t_i is present in the call, are identified and deleted during expansion. If every term is deleted, the breakdown of terms is replaced with a message indicating the expression is /OMEGA. If any terms are deleted because of truncation value t_i, an additional message indicating this fact appears in the printed output.

Term values are computed and compared to truncation values after a disjunctive normal form of the right-side expression of an equation has been obtained by expansion. If every term is deleted because of truncation values, the message

EVERY TERM WAS DELETED BECAUSE OF
TRUNCATION VALUES

is printed and, if there are no other equations specified in the call of COMTRMVAL, execution of the procedure is terminated. Otherwise, the next equation specified in the call is processed.

If some terms are deleted because of truncation values, but not all terms are deleted, the message

THERE WERE TERMS DELETED BECAUSE OF
TRUNCATION VALUES

is printed and processing of the current equation continues.

A Variable Occurrence Table is printed preceding each equation printed. All and only the variables that occur in the terms of the printed equation are listed in the table. Terms are deleted from the expression during expansion if they contain complementary variables, or if they contain more than the number of variables specified by truncation value t_i, if t_i is present in the call. Terms are also deleted after expansion when term values are computed if any of their values exceeds the corresponding truncation value. If terms are deleted, some variables that occur in the equation for e_i may not appear in the table because they do not occur in the terms of the printed equation.

The number of times that each variable occurs in the printed equation is printed with the variable in the table. Since all occurrences of a repeated variable in a term except one have been deleted during expansion, the number of times a variable occurs is the number of terms which contain that variable. A message indicating the number of different variables that occur in the terms of the printed equation is printed at the end of the table.

After the Variable Occurrence Table has been printed, a disjunctive normal form of the remaining terms of the right-side expression and the values for each term are printed. Each term is numbered and up to three values can be printed with the term. If more than three computations are specified in a COMTRMVAL call, and there can be as many as ten, the equation must be reprinted to accommodate the additional term values. The term number and the values for the first computation

are repeated with every printing of the equation. The values for the first computation are repeated because the terms are sorted according to the values for this computation. Thus, term values for the first, second, and third computations are printed with the first printing of the equation; term values for the first, fourth, and fifth computations are printed with the second printing of the equation; etc., until the first and tenth computations are printed with the fifth printing of the equation.

After the term values have been printed for every computation specified in the call, the extreme value for each computation is printed. The extreme value for a monotone decreasing computation is printed in the form

THE MAXIMUM TERM VALUE FOR COMPUTATION k IS w

and the extreme value for a monotone increasing computation is printed in the form

THE MINIMUM TERM VALUE FOR COMPUTATION k IS w

where w is the extreme value for the k-th computation in the COMTRMVAL call. For computation type PROBABILITY, the sum of the term values is also computed and printed below the extreme value in the form

(THE SUM OF THE TERM VALUES IS s)

where s is the sum of the term probabilities.

Example

Consider SETS user program:

```
PROGRAM$ COMTRMVAL-EX.
   RDVALBLK (VB-COSTS, VB-PROBS).
   COMTRMVAL (SUM, VB-COSTS* NO-EQN).
   EQN-4 = X1*(X2 + X3*/X4) + X4*(X5*(/X1*X6 + X3*X2) +
           X6*(X2 + X7)) + X2*(X6 + /X4*X5).
   REDUCEQN (EQN-4, EQN-4).
   COMTRMVAL (PROBABILITY, VB-PROBS* EQN-4).
   COMTRMVAL (INCREASE$ SUM, VB-COSTS* EQN-4).
   COMTRMVAL (PROBABILITY/1E-3, VB-PROBS* EQN-4).
   COMTRMVAL (PROBABILITY/2E-5, VB-PROBS, MINIMUM, VB-PROBS,
              SUM/20, VB-COSTS, MAXIMUM, VB-COSTS* EQN-4).
```

The first statement in the SETS user program is a RDVALBLK call. The Read Value Block procedure is described in Section 4.3.1, but it must be used here to create the value blocks for this example. Two value blocks are created and added to the value block file. One of them is named VB-PROBS and it contains a failure probability for each of the variables X1 through X7. The other value block is named VB-COSTS and it contains a cost associated with each of the variables X1 through X7. The variable values in the two value blocks are shown in Table 4.5.

Table 4.5 Variable Values for VB-PROBS and VB-COSTS

Variable	VB-PROBS	VB-COSTS
X1	7.2×10^{-3}	17
X2	3.0×10^{-2}	2
X3	7.2×10^{-3}	5
X4	1.0×10^{-1}	2
X5	3.0×10^{-2}	7.3
X6	2.7×10^{-2}	7.3
X7	7.2×10^{-3}	17

The equation file is empty when execution of COMTRMVAL-EX begins. The RDVALBLK statement enters value blocks VB-PROBS and VB-COSTS into the value block file. Then, a COMTRMVAL call shows the equation file does not contain an equation for NO-EQN. The printed output produced by this procedure is as follows:

```
COMTRMVAL (SUM, VB-COSTS* NO-EQN).

          THERE IS NO EQUATION FOR NO-EQN

     STATEMENT EXECUTION REQUIRED      .003 SECONDS FOR COMTRMVAL
```

The next statement in the SETS user program is an equation statement that enters an equation for EQN-4 into the equation file. Then a call of REDUCEQN redefines the equation for EQN-4 and deletes subsuming terms from it. There are eight terms in the disjunctive normal form of EQN-4 produced by expansion and seven terms are retained by simplification. The printed output produced by the REDUCEQN call is unimportant. The printed output produced by the equation statement is as follows:

```
EQN-4 = X1 * ( X2 + X3 * /X4 ) + X4 * ( X5 * ( /X1 * X6 +
             1              1       1      2

X3 * X2 ) + X6 * ( X2 + X7 ) ) + X2 * ( X6 + /X4 * X5 )
       2       2         2 1       1              1

     STATEMENT EXECUTION REQUIRED      .006 SECONDS FOR LDTMPBLK
```

The next statement in the SETS user program is a COMTRMVAL call that produces a disjunctive normal form of the right-side expression of the equation for EQN-4, computes the probability of each term in the expression using the values in value block VB-PROBS, and prints the terms, the term probabilities, and the extreme values. The printed output produced by the COMTRMVAL call is as follows:

```
COMTRMVAL (PROBABILITY, VB-PROBS* EQN-4).

              THE MAXIMUM NUMBER OF TERMS THAT CAN BE
              GENERATED BY EXPANSION IS         7.

              TERMS GENERATED BY EXPANSION
                   2 TERMS CONTAIN  2 VARIABLES
                   3 TERMS CONTAIN  3 VARIABLES
                   2 TERMS CONTAIN  4 VARIABLES
              TOTAL TERMS GENERATED        7.
         EXPANSION TOOK      .006 SECONDS.


                                  * * * * VARIABLE OCCURRENCE TABLE * * * *

                    NONCOMPLEMENT        NUMBER OF          COMPLEMENT          NUMBER OF
                      VARIABLE           OCCURRENCES         VARIABLE           OCCURRENCES

                      X1                    2               /X1                   1
                      X2                    4
                      X3                    2
                      X4                    3               /X4                   2
                      X5                    3
                      X6                    3
                      X7                    1

                   THERE ARE   9 DIFFERENT VARIABLES IN THE
                   EQUATION FOR EQN-4


                TERM      PROB.
               NUMBER    OF TERM

                                  EQN-4 =

                   1   8.1000E-04    X2 * X6 +

                   2   8.1000E-04    X2 * X5 * /X4 +

                   3   2.1600E-04    X1 * X2 +

                   4   8.0417E-05    X4 * X5 * X6 * /X1 +

                   5   4.6656E-05    X1 * X3 * /X4 +

                   6   1.9440E-05    X4 * X6 * X7 +

                   7   6.4800E-07    X2 * X3 * X4 * X5

                     THE MAXIMUM TERM VALUE FOR COMPUTATION 1 IS   8.100000000000E-04
                         (THE SUM OF THE TERM VALUES IS   1.983160800000E-03)

          STATEMENT EXECUTION REQUIRED      .020 SECONDS FOR COMTRMVAL
```

The next statement in the SETS user program is a COMTRMVAL call that produces a disjunctive normal form of the right-side expression of the equation for EQN-4, computes the term sum for each term in the expression using the values in value block VB-COSTS, and prints the terms, the term values, and the extreme value. Terms are printed in order of increasing term sum because parameter INCREASE occurs in the call. The printed output for expansion and the Variable Occurrence

Table produced by this COMTRMVAL call is the same printed output shown for the previous COMTRMVAL call. The remainder of the printed output for this COMTRMVAL call is as follows:

```
COMTRMVAL (INCREASES SUM, VB-COSTS* EQN-4).

                .
                .
                .
   TERM      SUM
  NUMBER   OF TERM

                          EQN-4 =

      1   9.3000E+00     X2 * X6 +

      2   9.3000E+00     X2 * X5 * /X4 +

      3   1.6300E+01     X2 * X3 * X4 * X5 +

      4   1.6600E+01     X4 * X5 * X6 * /X1 +

      5   1.9000E+01     X1 * X2 +

      6   2.2000E+01     X1 * X3 * /X4 +

      7   2.6300E+01     X4 * X6 * X7

        THE MINIMUM TERM VALUE FOR COMPUTATION 1 IS   9.300000000000E+00

STATEMENT EXECUTION REQUIRED      .021 SECONDS FOR COMTRMVAL
```

The next statement in the SETS user program is a COMTRMVAL call that contains probability truncation value 1×10^{-3}. Terms in the disjunctive normal form of the right-side expression of the equation for EQN-4 having values that exceed the truncation value are not printed. Since every term in the right-side expression has a probability value less than the truncation value, no terms are printed. The printed output produced by this procedure is as follows:

```
COMTRMVAL (PROBABILITY/ 1E-3, VB-PROBS* EQN-4).

          THE MAXIMUM NUMBER OF TERMS THAT CAN BE
          GENERATED BY EXPANSION IS          7.

          TERMS GENERATED BY EXPANSION
               2 TERMS CONTAIN  2 VARIABLES
               3 TERMS CONTAIN  3 VARIABLES
               2 TERMS CONTAIN  4 VARIABLES
          TOTAL TERMS GENERATED        7.
     EXPANSION TOOK      .006 SECONDS.

          EVERY TERM WAS DELETED BECAUSE OF
          TRUNCATION VALUES

STATEMENT EXECUTION REQUIRED      .011 SECONDS FOR COMTRMVAL
```

The next statement in the SETS user program is a COMTRMVAL call that specifies four computations: PROBABILITY and MINIMUM are computed using value block VB-PROBS; and SUM and MAXIMUM are computed using value block VB-COSTS. The call contains two truncation values. Terms of the right-side expression of the equation for EQN-4 which have a term probability $< 2 \times 10^{-5}$ or a term sum > 20 are deleted. Four of the seven terms in the expression do not exceed either truncation value. They are printed in order of decreasing value for the first computation. The printed output produced by the COMTRMVAL call is as follows:

```
COMTRMVAL (PROBABILITY/ 2E-5, VB-PROBS, MINIMUM, VB-PROBS, SUM/ 20,
          VB-COSTS, MAXIMUM, VB-COSTS* EQN-4).

          THE MAXIMUM NUMBER OF TERMS THAT CAN BE
          GENERATED BY EXPANSION IS          7.

          TERMS GENERATED BY EXPANSION
               2 TERMS CONTAIN  2 VARIABLES
               3 TERMS CONTAIN  3 VARIABLES
               2 TERMS CONTAIN  4 VARIABLES
          TOTAL TERMS GENERATED        7.
     EXPANSION TOOK       .007 SECONDS.

          THERE WERE TERMS DELETED BECAUSE OF
          TRUNCATION VALUES


                              * * * * VARIABLE OCCURRENCE TABLE * * * *

                    NONCOMPLEMENT        NUMBER OF          COMPLEMENT          NUMBER OF
                      VARIABLE          OCCURRENCES          VARIABLE          OCCURRENCES

                      X1                    1               /X1                   1
                      X2                    3
                      X4                    1               /X4                   1
                      X5                    2
                      X6                    2

          THERE ARE   7 DIFFERENT VARIABLES IN THE
          EQUATION FOR EQN-4


      TERM      PROB.       MINIMUM       SUM
     NUMBER   OF TERM     OF TERM     OF TERM

                                                    EQN-4 =

        1   8.1000E-04 2.7000E-02 9.3000E+00    X2 * X6 +

        2   8.1000E-04 3.0000E-02 9.3000E+00    X2 * X5 * /X4 +

        3   2.1600E-04 7.2000E-03 1.9000E+01    X1 * X2 +

        4   8.0417E-05 2.7000E-02 1.6600E+01    X4 * X5 * X6 * /X1
```

```
          TERM      PROB.      MAXIMUM
         NUMBER    OF TERM    OF TERM

                                        EQN-4 =

             1   8.1000E-04 7.3000E+00    X2 * X6 +

             2   8.1000E-04 7.3000E+00    X2 * X5 * /X4 +

             3   2.1600E-04 1.7000E+01    X1 * X2 +

             4   8.0417E-05 7.3000E+00    X4 * X5 * X6 * /X1

               THE MAXIMUM TERM VALUE FOR COMPUTATION 1 IS   8.100000000000E-04
                    (THE SUM OF THE TERM VALUES IS   1.916416800000E-03)

               THE MAXIMUM TERM VALUE FOR COMPUTATION 2 IS   3.000000000000E-02

               THE MINIMUM TERM VALUE FOR COMPUTATION 3 IS   9.300000000000E+00

               THE MINIMUM TERM VALUE FOR COMPUTATION 4 IS   7.300000000000E+00

STATEMENT EXECUTION REQUIRED      .029 SECONDS FOR COMTRMVAL
```

4.1.7 Truncate on Term Value

The Truncate on Term Value procedure is used to create a new equation and enter it into the equation file. This procedure is a special version of the Reduce Equation procedure. During the expansion step of the reduction process, term values are computed and terms having values that exceed truncation values specified in the procedure call are deleted. Parameters in the procedure call specify what computations to make and which variable values to use in the computations. (Variable values and the computed term values are all nonnegative.) In addition, for each computation specified in the procedure call, an extreme value is determined from the term values for that computation. For example, suppose the equation file contains the equation

A=B*(C+D*F)+D*(E+C*(F+G)).

Also, suppose the value block file contains value block VB1 which contains variable values:

B=5
C=8
D=2
E=5
F=1
G=8

Execution of procedure call statement

TRNTRMVAL (SUM/12, VB1* A, H)

creates the new equation

H=D*(F*(C+B)+E)

and enters it into the equation file. The right-side expression of the created equation is produced by reducing a copy of the right-side expression of the equation for A. During the expansion step of the reduction process, a value is computed for each term in the expression using the variable values in value block VB1. For computation type SUM, the term value is the sum of the variable values for the variables in the term. Terms having a value greater than truncation value 12 are deleted. In the example, terms B*C and D*C*G are deleted because their values, 13 and 18, respectively, are greater than 12. Remaining terms D*E, B*D*F, and D*C*F, having values 7, 8, and 11, respectively, are simplified and factored to produce the right-side expression for the new equation. The left-side variable for the new equation is H. The extreme value, which for SUM is the minimum term value, is determined from the term values and printed in the form:

THE MIMIMUM TERM VALUE FOR COMPUTATION 1 IS 7.0000E+00

Procedure Call

A call of the Truncate on Term Value procedure has one of the forms:

a. TRNTRMVAL (q_1,vb_1,q_2,vb_2,...,q_s,vb_s*

<any parameter form of REDUCEQN>).

b. TRNTRMVAL (EXTREME$ q_1,vb_1,q_2,vb_2,...,q_s,vb_s*

<any parameter form of REDUCEQN>).

Each parameter pair, q_k and vb_k, k=1,2,...,s, specifies a computation. Each q_k is either c_k or c_k/l_k which contains computation type c_k and truncation value l_k, if l_k is present. The corresponding vb_k identifies the value block that contains the variable values to be used in the computation specified by q_k. At least one computation must be specified in a TRNTRMVAL call, and no more than ten computations can be specified, ($1 \leq s \leq 10$).

Following the pairs of q_k and vb_k parameters, a parameter form of the Reduce Equation procedure must occur. This portion of the parameter part of TRNTRMVAL has the same form,

$$(p_1, e_2)$$

or

$$(p_1\ e_2{*}\ O_1/O_2/.../O_m),$$

and meaning described in Section 4.1.5, Reduce Equation. Parameter p_1 specifies the equation in the equation file that is used to generate the right-side expression for the new equation. Parameter p_1 is either e_1 or e_1/t_1 which contains left-side variable e_1 and truncation value t_1, if t_1 is present. Truncation value t_1 must be an integer. Parameter e_2 is the left-side variable for the new equation.

Parameter EXTREME causes terms that do not have the extreme value for every computation specified in a TRNTRMVAL call to be deleted during expansion of the expression for e_1. If parameter EXTREME occurs in a TRNTRMVAL call, it must be the first parameter in the call.

Options, O_j, j=1,2,...,m, retain all of the meaning that they have in the Reduce Equation procedure. In addition, the meaning of the except and delete options is extended to cover term value computations.

Processing Achieved

The Truncate on Term Value and Reduce Equation procedures achieve similar processing. For both procedures, a new equation is created and entered into the equation file. Parameter e_2 is the left-side variable for the new equation, and the right-side expression is created from the right-side expression of an equation in the equation file. A copy of the right-side expression of the equation for e_1, the left-side variable contained in parameter p_1, is reduced and becomes the right-side

expression for the new equation. A detailed description of the reduction process appears in Section 4.1.5, Reduce Equation, but the three steps in the process can be summarized as follows:

1. Expansion

   a. Apply DeMorgan's rules to the factored expression.

   b. Apply the identities P+OMEGA = OMEGA, P*OMEGA = P, P*/OMEGA = /OMEGA, and P+/OMEGA = P to the factored expression produced by step 1a.

   c. Expand the factored expression produced by step 1b by the distributive law and apply the identities P*/P = /OMEGA, P*/OMEGA = /OMEGA, P+/OMEGA = P, and P*P = P, to obtain a disjunctive normal form of the expression.

2. Simplification

   Apply the identity P + P*Q = P to the disjunctive normal form expression produced by expansion.

3. Factorization

   Factor the disjunctive normal form of the expression produced by simplification to create a factored form of the expression.

The processing achieved by the Truncate on Term Value and Reduce Equation procedures differs in the last part of expansion (step 1c). In the Truncate on Term Value procedure, term values are computed as the expression is expanded into a disjunctive normal form. For each term in the expression, values are computed for every computation specified in the TRNTRMVAL call. The values for each term determine whether or not the term is deleted.

The computations that can be specified in a TRNTRMVAL call are the same computations that can be specified in a COMTRMVAL call. There are nine different computations: SUM, PRODUCT, COUNT, DISTINCT-SUM, DISTINCT-PRODUCT, DISTINCT-COUNT, MAXIMUM, MINIMUM, and PROBABILITY. Each computation is recognized by a computation type which specifies a rule for computing term values from variable values. Variable values must be nonnegative and they are further restricted so each computation is either monotone decreasing or monotone increasing. A detailed description of the computations is presented in Section 4.1.6, Compute Term Value.

In a TRNTRMVAL call, parameter EXTREME and the parameters that specify the computations must be processed before the right-side expression of the specified equation can be reduced and truncated. As each q_k is encountered, the computation type and truncation value for the k-th computation are determined. The computation type is specified by c_k contained in parameter q_k. If truncation value 1_k occurs in parameter q_k, the truncation value for the k-th computation is set to 1_k. Otherwise, the truncation value is set in accordance with computation type c_k so that no truncation will occur. If c_k is monotone decreasing, the

truncation value for the k-th computation is set to 0; and, if c_k is monotone increasing, the truncation value for the k-th computation is set to $1x10^{100}$. Truncation values remain fixed at their initial values if parameter EXTREME does not occur in the call (form a); but, if EXTREME occurs in the call (form b), truncation values are updated whenever a better, i.e., closer to the extreme, term value is obtained.

As each vb_k is encountered, variable values from value block vb_k are established as the values to be used for the k-th computation.

After the pairs of q_k and vb_k parameters have been processed a copy of the right-side expression of the equation for e_l is obtained from the equation file. If the equation file does not contain an equation for e_l, a new equation is not created; the equation file is not changed; and execution of the TRNTRMVAL call is terminated. Otherwise, the variables in every value block specified in the call are compared to the variables in the expression for e_l before it is reduced. Except for variables that do not need values because they are specified in except or delete options (see this procedure, Options), each value block must contain a value for every noncomplement variable that occurs in the expression. Also, for every complement variable that occurs in the expression, each value block must contain a value for the corresponding noncomplement variable.

As the copy of the expression for e_l is expanded by the distributive law, values for every computation specified in the TRNTRMVAL call are computed for each term in the expression. A term is deleted if any of its values exceed the corresponding truncation value. (A term value exceed the truncation value if, for monotone decreasing computations, it is less than the truncation value, and, for monotone increasing computations, it is greater than the truncation value.) If every term exceeds at least one truncation value, every term will be deleted. Obviously, if the truncation value for a monotone decreasing computation is greater than the largest term value or if the truncation value for a monotone increasing computation is less than the smallest term value, every term in the expression will be deleted. If every term is deleted, the expression generated is /OMEGA.

The extreme value is determined for each computation specified in a TRNTRMVAL call. The extreme value for the k-th computation is determined according to computation type c_k. For a monotone decreasing computation, the extreme value is the maximum value over all term values for the computation; and for a monotone increasing computation, the extreme value is the minimum value over all term values for the computation.

For computation type PROBABILITY, the sum of the values for the remaining terms is also computed. This sum is often used as an approximation to the probability of the expression. If the expression is not reduced prior to the TRNTRMVAL call, subsuming terms can contribute unnecessarily to the approximation. Subsuming terms are deleted from the expression during the reduction process, but term values, extreme values, and the sum of the term values for PROBABILITY are determined in step 1c of expansion before subsuming terms are deleted in step 2, simplification.

If parameter EXTREME occurs in a TRNTRMVAL call (form b), terms are deleted if any of their values exceed the corresponding extreme value. However, extreme

values are not known at the beginning of step 1c of expansion. Rather, they are determined during step 1c as the expression is expanded by the distributive law to produce a disjunctive normal form. Extreme values are determined by changing truncation values for computations specified in the TRNTRMVAL call. A term value that is closer to the extreme value for a computation than the current truncation value is a better truncation value for the computation. Whenever a term value is computed that is better than the corresponding, current truncation value, the truncation value is changed to the better value and all previously retained terms are deleted. Moreover, if a term that produces an improved truncation value for a computation does not also have the current truncation value for every other computation, it too is deleted. Ultimately, the truncation value for each computation is the extreme value for the computation, and the only terms retained are terms that have the extreme value for every computation.

Specifying truncation values, l_k, k=1,2,...,s, in a call of TRNTRMVAL which contains parameter EXTREME can improve the efficiency of finding extreme valued terms. If l_k is a feasible truncation value for the k-th computation, i.e., less than or equal to the maximum term value for a monotone decreasing computation, or greater than or equal to the minimum term value for a monotone increasing computation, the extreme values determined will be correct and the expression obtained will contain every extreme valued term. However, if l_k is infeasible, the extreme value for the k-th computation will not be determined and the expression obtained will be /OMEGA. From the printed output for extreme values, it will appear the extreme value for the k-th computation is l_k, and none of the terms of the expression is an extreme valued term. In fact, the value printed as the extreme value is not, and the expression may or may not have extreme valued terms.

The determination of extreme values can be shown using the expression

$$B*(D*(E+F*G)+C+E*/H)+I*(/E*(D+/C*H)+C) \qquad (4\text{-}16)$$

and variable values:

$$\begin{aligned} &B=13\\ &C=7\\ &D=3\\ &E=4\\ &F=1\\ &G=1\\ &H=2\\ &I=15 \end{aligned} \qquad (4\text{-}17)$$

Suppose a TRNTRMVAL call contains parameter EXTREME and specifies computations SUM and PRODUCT be made for Exp. 4-16 using the values represented in Eqns. 4-17. The determination of extreme values depends on the order in which terms are produced during expansion. The terms of Exp. 4-16 are produced in the order shown in Table 4.6. The term values for the two computations are also shown in Table 4.6.

Table 4.6 Terms and Term Values for Exp. 4-16

	Term	Term Value (SUM)	Term Value (PRODUCT)
1.	B*C	20	91
2.	I*C	22	105
3.	B*D*E	20	156
4.	B*E*/H	17	52
5.	I*/E*D	18	45
6.	B*D*F*G	18	39
7.	I*/E*/C*H	17	30

Assume truncation values for SUM and PRODUCT are not specified in the call, so both of them are set to 1×10^{100}. The first term produced by expansion, B*C, has values, 20 for SUM and 91 for PRODUCT, that are better, i.e., smaller, than the initial truncation values. The truncation values are changed to the better values, and the first term, having all of the current truncation values, is retained. None of the values for the second and third terms is better than the current truncation values. Moreover, the second and third terms are deleted because they have at least one value that exceeds the corresponding truncation value.

The fourth term has values that are better, 17 for SUM and 52 for PRODUCT, than the corresponding current truncation values for both computations. Whenever a better value is determined for either computation, every previously retained term is deleted. Thus, the truncation values for SUM and PRODUCT are updated to 17 and 52, respectively; the first term is deleted; and the fourth term, having all of the current truncation values, is retained. None of the remaining terms of Exp. 4-16 has a smaller term value for SUM, and the extreme value for SUM is 17. However, a better truncation value for PRODUCT is determined on each of the next three terms produced by expansion.

When the fifth term is produced, it has a better value for PRODUCT and every previously retained term is deleted. Moreover, the fifth term is deleted because its value for SUM, 18, exceeds the current truncation value for SUM, 17. None of the terms of Exp. 4-16 produced thus far have the current truncation values for both computations, namely, 17 for SUM and 45 for PRODUCT. If Exp. 4-16 had no more terms, the right-side expression generated by the TRNTRMVAL call for this example would be /OMEGA. However, Exp. 4-16 has two more terms.

The sixth term of Exp. 4-16 is like the fifth term. It has a better value for PRODUCT, 39, but it is deleted because it does not have the extreme value for SUM.

The seventh term is the only term of Exp. 4-16 that has the extreme value for both computations, 17 for SUM and 30 for PRODUCT. Thus, the right-side expression generated by the TRNTRMVAL call is the single, extreme valued term, I*/E*/C*H.

Options

The options that can occur in a TRNTRMVAL call have the same form and retain all of the meaning described in Section 4.1.5, Reduce Equation. In addition, the meaning of the except complement, except noncomplement, delete complement, and delete noncomplement options is extended in the Truncate on Term Value procedure to cover term value computations. Variables specified in these options do not contribute to term value for any computation specified in a TRNTRMVAL call. If every variable in a term is in an except or delete option but not all of them are in a delete option, the term value is the initial value of the term, namely, the identity element for the computation. If every variable in a term is in a delete option, the term and, therefore, the expression is OMEGA.

Printed Output

The printed output produced by the Truncate on Term Value and Reduce Equation procedures is similar. The output produced by the first two parts of expansion (step 1a and step 1b), simplification (step 2), and Factorization (step 3) is the same as described in Section 4.1.5, Reduce Equation, Printed Output.

The printed output produced by the last part of expansion (step 1c), begins with the message that indicates the maximum number of terms that can be generated by expansion. The number of terms that can be generated is computed at the beginning of step 1c of expansion. The number of terms that are generated by expansion differs from the maximum number of terms that can be generated by the number of terms that are deleted for any reason.

The breakdown of terms with respect to the number of variables per term occurs next in the printed output. If parameter EXTREME does not occur in the TRNTRMVAL call (form a), the breakdown of terms is the same as it is for the Reduce Equation procedure (see Section 4.1.5, Reduce Equation, Printed Output).

If parameter EXTREME occurs in a TRNTRMVAL call (form b), extreme values are determined for the computations specified in the call during the expansion of the expression. Whenever a term value is computed that is better than the corresponding current truncation value, the truncation value is changed to the better value and a message is printed. For monotone decreasing computations, the message has the form

THE NEW MAXIMUM TERM VALUE FOR COMPUTATION k IS w

and, for monotone increasing computations, it has the form

THE NEW MINIMUM TERM VALUE FOR COMPUTATION k IS w

where w is the new truncation value for the k-th computation in the call.

Messages showing improvement in truncation values are interspersed among the messages showing the breakdown of terms generated by expansion. Messages from the breakdown of terms which occur before the last message indicating an improved truncation value are irrelevent, because all previously retained terms are deleted every time an improved truncation value is determined. However, messages from the breakdown of terms which occur after the last truncation value message are relevant.

After the breakdown of terms and the message indicating the total number of terms generated by expansion are printed, messages about the deletion of terms are printed. If any terms are deleted because of truncation value t_1, the message

```
THERE WERE TERMS DELETED BECAUSE OF
THE TRUNCATION VALUE OF t1
```

is printed. Moreover, if some, but not all, of the terms that are not deleted for some other reason are deleted because of truncation values for computations specified in the TRNTRMVAL call, the message

```
THERE WERE TERMS DELETED BECAUSE OF
TRUNCATION VALUES
```

is also printed. If every term not deleted for some other reason is deleted because of truncation values for computations, the message

```
EVERY TERM WAS DELETED BECAUSE OF
TRUNCATION VALUES
```

is printed. If parameter EXTREME occurs in the call, the word TRUNCATION in the last two messages is changed to EXTREME.

The extreme values for the computations specified in the call are printed after messages about the deletion of terms. The extreme value for a monotone decreasing computation is printed in the form

```
THE MAXIMUM TERM VALUE FOR COMPUTATION k IS   w
```

and the extreme value for a monotone increasing computation is printed in the form

```
THE MINIMUM TERM VALUE FOR COMPUTATION k IS   w
```

where w is the extreme value for the k-th computation in the TRNTRMVAL call.

Consider the printed output produced by expansion for the TRNTRMVAL call used to generate an expression of the extreme valued terms from Exp. 4-16.

```
THE MAXIMUM NUMBER OF TERMS THAT CAN BE
GENERATED BY EXPANSION IS                7.

TERMS GENERATED BY EXPANSION
THE NEW MINIMUM TERM VALUE FOR COMPUTATION  1 IS  2.0000E+01
THE NEW MINIMUM TERM VALUE FOR COMPUTATION  2 IS  9.1000E+01
        1 TERMS CONTAIN  2 VARIABLES
THE NEW MINIMUM TERM VALUE FOR COMPUTATION  1 IS  1.7000E+01
THE NEW MINIMUM TERM VALUE FOR COMPUTATION  2 IS  5.2000E+01
THE NEW MINIMUM TERM VALUE FOR COMPUTATION  2 IS  4.5000E+01
THE NEW MINIMUM TERM VALUE FOR COMPUTATION  2 IS  3.9000E+01
THE NEW MINIMUM TERM VALUE FOR COMPUTATION  2 IS  3.0000E+01
        1 TERMS CONTAIN  4 VARIABLES
TOTAL TERMS GENERATED          1.
```

```
    THERE WERE TERMS DELETED BECAUSE OF
    EXTREME VALUES

    THE MINIMUM TERM VALUE FOR COMPUTATION  1 IS   1.7000E+01

    THE MINIMUM TERM VALUE FOR COMPUTATION  2 IS   3.0000E+01

EXPANSION TOOK    .012 SECONDS
```

Example

Consider SETS user program:

```
PROGRAM$ TRNTRMVAL-EX.
   RDVALBLK (VB-COSTS, VB-PROBS).
   TRNTRMVAL (SUM/12, VB-COSTS* NO-EQN, NEW-EQN).
   PRTEQN (NEW-EQN).
   EQN-4 = X1*(X2 + X3*/X4) + X4*(X5*(/X1*X6 + X3*X2) +
           X6*(X2 + X7)) + X2*(X6 + /X4*X5).
   REDUCEQN (EQN-4, EQN-4).
   COMTRMVAL (PROBABILITY, VB-PROBS, SUM, VB-COSTS* EQN-4).
   TRNTRMVAL (SUM/16.5, VB-COSTS* EQN-4, EQN-4-SUM-TRN).
   COMTRMVAL (INCREASE$ SUM, VB-COSTS* EQN-4-SUM-TRN).
   TRNTRMVAL (PROBABILITY/2E-5, VB-PROBS, SUM/20, VB-COSTS*
              EQN-4, EQN-4-SUMPRB-TRN).
   COMTRMVAL (PROBABILITY, VB-PROBS, SUM, VB-COSTS* EQN-4-SUMPRB-TRN).
   TRNTRMVAL (EXTREME$ PROBABILITY, VB-PROBS, SUM, VB-COSTS*
              EQN-4, EQN-4-EXT).
   PRTEQN (EQN-4-EXT).
```

The first statement in the SETS user program is a RDVALBLK call, which must be used here to create value blocks for this example. Two value blocks are created and added to the value block file. They are the same value blocks used in Section 4.1.6, Compute Term Value, Example, and the variables and values contained in the blocks are shown in Table 4.5 in that section.

The equation file is empty when execution of TRNTRMVAL-EX begins. The RDVALBLK statement enters value blocks VB-PROBS and VB-COSTS into the value block file. Then, a TRNTRMVAL call shows the equation file does not contain an equation for NO-EQN, and a PRTEQN call shows an equation was not created for NEW-EQN. The printed output produced by these two procedures is as follows:

```
TRNTRMVAL (SUM/ 12, VB-COSTS* NO-EQN, NEW-EQN).

          THERE IS NO EQUATION FOR NO-EQN

     STATEMENT EXECUTION REQUIRED      .003 SECONDS FOR TRNTRMVAL


PRTEQN (NEW-EQN).

          THERE IS NO EQUATION FOR NEW-EQN

     STATEMENT EXECUTION REQUIRED      .002 SECONDS FOR PRTEQN
```

The next statement in the SETS user program is an equation statement that enters an equation for EQN-4 into the equation file. Then, a call of REDUCEQN redefines the equation for EQN-4 and deletes subsuming terms from it. There are eight terms in the disjunctive normal form of EQN-4 produced by expansion and seven terms are retained by simplification. The printed output produced by the REDUCEQN call is unimportant here. The printed output produced by the equation statement is as follows:

```
EQN-4 = X1 * ( X2 + X3 * /X4 ) + X4 * ( X5 * ( /X1 * X6 +
             1              1       1      2

X3 * X2 ) + X6 * ( X2 + X7 ) ) + X2 * ( X6 + /X4 * X5 )
       2        2         2 1        1               1

STATEMENT EXECUTION REQUIRED      .005 SECONDS FOR LDTMPBLK
```

The next statement in the SETS user program is a COMTRMVAL call that prints the term values for PROBABILITY and SUM for the seven terms in the equation for EQN-4. Results obtained by subsequent TRNTRMVAL calls can be compared to the terms, term values, and extreme values produced by this call of COMTRMVAL. Part of the printed output produced by the COMTRMVAL call is as follows:

```
COMTRMVAL (PROBABILITY, VB-PROBS, SUM, VB-COSTS* EQN-4).

                •

                •

                •

 TERM     PROB.       SUM
NUMBER  OF TERM    OF TERM

                                  EQN-4 =

    1   8.1000E-04 9.3000E+00     X2 * X6 +

    2   8.1000E-04 9.3000E+00     X2 * X5 * /X4 +

    3   2.1600E-04 1.9000E+01     X1 * X2 +

    4   8.0417E-05 1.6600E+01     X4 * X5 * X6 * /X1 +

    5   4.6656E-05 2.2000E+01     X1 * X3 * /X4 +

    6   1.9440E-05 2.6300E+01     X4 * X6 * X7 +

    7   6.4800E-07 1.6300E+01     X2 * X3 * X4 * X5

  THE MAXIMUM TERM VALUE FOR COMPUTATION 1 IS   8.100000000000E-04
      (THE SUM OF THE TERM VALUES IS   1.983160800000E-03)

  THE MINIMUM TERM VALUE FOR COMPUTATION 2 IS   9.300000000000E+00

STATEMENT EXECUTION REQUIRED      .023 SECONDS FOR COMTRMVAL
```

The next statement in the SETS user program is a TRNTRMVAL call that creates an equation for EQN-4-SUM-TRN. Values in VB-COSTS are used for computation SUM to compute term values and delete any term having term sum greater than 16.5. Three of the seven terms in EQN-4 are retained in the right-side expression of the created equation for EQN-4-SUM-TRN. A COMTRMVAL call shows the terms in the equation for EQN-4-SUM-TRN and the values for computation SUM. The printed output produced by the TRNTRMVAL call and part of the printed output produced by the COMTRMVAL call is as follows:

```
TRNTRMVAL (SUM/ 16.5, VB-COSTS* EQN-4, EQN-4-SUM-TRN).

                THE MAXIMUM NUMBER OF TERMS THAT CAN BE
                GENERATED BY EXPANSION IS          7.

                TERMS GENERATED BY EXPANSION
                     1 TERMS CONTAIN  2 VARIABLES
                     1 TERMS CONTAIN  3 VARIABLES
                     1 TERMS CONTAIN  4 VARIABLES
                TOTAL TERMS GENERATED        3.

                THERE WERE TERMS DELETED BECAUSE OF
                TRUNCATION VALUES

                .HE MINIMUM TERM VALUE FOR COMPUTATION 1 IS   9.300000000000E+00

          EXPANSION TOOK      .008 SECONDS.

                TERMS RETAINED BY SIMPLIFICATION
                     1 TERMS CONTAIN  2 VARIABLES
                     1 TERMS CONTAIN  3 VARIABLES
                     1 TERMS CONTAIN  4 VARIABLES
                TOTAL TERMS RETAINED        3.
          SIMPLIFICATION TOOK      .003 SECONDS.

          FACTORIZATION TOOK      .006 SECONDS.

     STATEMENT EXECUTION REQUIRED      .020 SECONDS FOR TRNTRMVAL


COMTRMVAL (INCREASE$ SUM, VB-COSTS* EQN-4-SUM-TRN).

               •

               •

               •

           TERM      SUM
          NUMBER   OF TERM

                                EQN-4-SUM-TRN =

              1   9.3000E+00    X2 * X6 +

              2   9.3000E+00    X2 * X5 * /X4 +

              3   1.6300E+01    X2 * X3 * X4 * X5

                THE MINIMUM TERM VALUE FOR COMPUTATION 1 IS   9.300000000000E+00

     STATEMENT EXECUTION REQUIRED      .018 SECONDS FOR COMTRMVAL
```

The next statement in the SETS user program is a TRNTRMVAL call that creates an equation for EQN-4-SUMPRB-TRN. The call specifies computations PROBABILITY and SUM using value blocks VB-PROBS and VB-COSTS, respectively, and contains two truncation values. The right-side expression for the new equation contains every term of the right-side expression of the equation for EQN-4 that has term probability $\geq 2 \times 10^{-5}$ and term sum ≤ 20. Four of the seven terms of EQN-4 are retained. A call of COMTRMVAL shows the four terms of the created equation and the probability and sum for each of them. The printed output produced by the TRNTRMVAL call and part of the printed output produced by the COMTRMVAL call is as follows:

```
TRNTRMVAL (PROBABILITY/ 2E-5, VB-PROBS, SUM/ 20, VB-COSTS* EQN-4,
          EQN-4-SUMPRB-TRN).

               THE MAXIMUM NUMBER OF TERMS THAT CAN BE
               GENERATED BY EXPANSION IS          7.

               TERMS GENERATED BY EXPANSION
                    2 TERMS CONTAIN  2 VARIABLES
                    1 TERMS CONTAIN  3 VARIABLES
                    1 TERMS CONTAIN  4 VARIABLES
               TOTAL TERMS GENERATED         4.

               THERE WERE TERMS DELETED BECAUSE OF
               TRUNCATION VALUES

               THE MAXIMUM TERM VALUE FOR COMPUTATION 1 IS   8.100000000000E-04
                   (THE SUM OF THE TERM VALUES IS   1.916416800000E-03)

               THE MINIMUM TERM VALUE FOR COMPUTATION 2 IS   9.300000000000E+00

          EXPANSION TOOK       .008 SECONDS.

               TERMS RETAINED BY SIMPLIFICATION
                    2 TERMS CONTAIN  2 VARIABLES
                    1 TERMS CONTAIN  3 VARIABLES
                    1 TERMS CONTAIN  4 VARIABLES
               TOTAL TERMS RETAINED         4.
          SIMPLIFICATION TOOK       .002 SECONDS.

          FACTORIZATION TOOK       .006 SECONDS.

     STATEMENT EXECUTION REQUIRED       .023 SECONDS FOR TRNTRMVAL


COMTRMVAL (PROBABILITY, VB-PROBS, SUM, VB-COSTS* EQN-4-SUMPRB-TRN).
```

•

•

•

```
 TERM     PROB.        SUM
NUMBER   OF TERM     OF TERM

                                    EQN-4-SUMPRB-TRN =

   1   8.1000E-04 9.3000E+00     X2 * X6 +

   2   8.1000E-04 9.3000E+00     X2 * X5 * /X4 +

   3   2.1600E-04 1.9000E+01     X1 * X2 +

   4   8.0417E-05 1.6600E+01     X4 * X5 * X6 * /X1

     THE MAXIMUM TERM VALUE FOR COMPUTATION 1 IS   8.100000000000E-04
         (THE SUM OF THE TERM VALUES IS   1.916416800000E-03)

     THE MINIMUM TERM VALUE FOR COMPUTATION 2 IS   9.300000000000E+00

STATEMENT EXECUTION REQUIRED      .020 SECONDS FOR COMTRMVAL
```

The next statement in the SETS user program is a TRNTRMVAL call which contains parameter EXTREME and creates an equation for EQN-4-EXT. Computations PROBABILITY and SUM are specified in the call. Two terms of EQN-4 have the extreme values for both computations. The extreme value determined for PROBABILITY using values from VB-PROBS is 8.1×10^{-4}, and the extreme value determined for SUM using values from VB-COSTS is 9.3. A PRTEQN call shows the two terms in the right-side expression of the equation for EQN-4-EXT. The printed output produced by these two procedures is as follows:

```
TRNTRMVAL (EXTREME$ PROBABILITY, VB-PROBS, SUM, VB-COSTS* EQN-4,
          EQN-4-EXT).

          THE MAXIMUM NUMBER OF TERMS THAT CAN BE
          GENERATED BY EXPANSION IS          7.

          TERMS GENERATED BY EXPANSION
          THE NEW MAXIMUM TERM VALUE FOR COMPUTATION 1 IS   8.100000000000E-04
          THE NEW MINIMUM TERM VALUE FOR COMPUTATION 2 IS   9.300000000000E+00
               1 TERMS CONTAIN  2 VARIABLES
               1 TERMS CONTAIN  3 VARIABLES
          TOTAL TERMS GENERATED        2.

          THERE WERE TERMS DELETED BECAUSE OF
          EXTREME VALUES

          THE MAXIMUM TERM VALUE FOR COMPUTATION 1 IS   8.100000000000E-04
              (THE SUM OF THE TERM VALUES IS   1.620000000000E-03)

          THE MINIMUM TERM VALUE FOR COMPUTATION 2 IS   9.300000000000E+00

        EXPANSION TOOK       .007 SECONDS.

          TERMS RETAINED BY SIMPLIFICATION
               1 TERMS CONTAIN  2 VARIABLES
               1 TERMS CONTAIN  3 VARIABLES
          TOTAL TERMS RETAINED         2.
        SIMPLIFICATION TOOK       .001 SECONDS.

        FACTORIZATION TOOK       .002 SECONDS.

     STATEMENT EXECUTION REQUIRED       .017 SECONDS FOR TRNTRMVAL


PRTEQN (EQN-4-EXT).

          EQN-4-EXT = X2 * ( X5 * /X4 + X6 )
                           1              1

     STATEMENT EXECUTION REQUIRED       .002 SECONDS FOR PRTEQN
```

This page intentionally left blank.

4.1.8 Delete Term

The Delete Term procedure is used to create a new equation and enter it into the equation file. Parameters in the procedure call specify two equations in the equation file that are used to generate the new equation. Copies of the right-side expressions of the two equations are expanded into disjunctive normal forms, and terms in the first expression that subsume terms in the second expression are deleted from the first expression. Remaining terms of the first expression are simplified and factored to produce the right-side expression for the new equation. The left-side variable for the new equation occurs as a parameter in the procedure call. For example, suppose the equation file contains equations

A=C*D+E*(C+D*F)
B=F*(C+E).

Execution of procedure call statement

DLTRM (A, B, G)

creates the equation

G=C*(D+E)

and enters it into the equation file. Copies of the right-side expressions for A and B are expanded into disjunctive normal forms

C*D + E*C + E*D*F

and

F*C + F*E,

respectively. The term E*D*F is deleted from the expression for A because it subsumes term F*E in the expression for B. Remaining terms of the expression for A, C*D and E*C, do not need simplification, but they are factored to produce the right-side expression for the new equation. The left-side variable for the new equation is G.

The uses of the Delete Term procedure may not be as obvious as the uses of the other procedures described in this manual. One use of the procedure is to delete terms from an expression which contain particular products of variables. In the example above, the expression obtained for G is comprised of terms of the expression for A that do not contain products of variables F*C or F*E, i.e., terms of the expression for B.

Another use of the Delete Term procedure is to obtain an approximation to a product of expressions of a particular form. Given expressions for A and B in the example above, the expression obtained for G is an approximation to the product A*/B which satisfies $A*/B \leq G \leq A$. The use of this approximation in accident sequence analysis is described in References 8 and 10, and the mathematical basis for the approximation is described in References 24 and 25.

Procedure Call

A call of the Delete Term procedure has the form:

DLTRM (p_1, p_2, e_3).

Parameters p_1, p_2, and e_3 are required in the procedure call. Parameters, p_i, i=1,2, specify the two equations used to generate the right-side expression for the new equation. Parameter p_i is either e_i or e_i/t_i which contains left-side variable e_i and truncation value t_i, if t_i is present. Truncation value t_i must be an integer. Parameter e_3 is the left-side variable for the new equation. Variables e_1 and e_2 contained in parameters p_1 and p_2, respectively, can be the same variable. Moreover, either one or both of them can be the same as variable e_3.

Processing Achieved

If the equation file does not contain equations for e_1 and e_2, an equation is not created and the equation file is not changed.

If the equation file contains equations for e_1 and e_2, an equation is created and entered into the equation file. Parameter e_3 is the left-side variable for the new equation. The equation for e_3 is entered into the equation file after copies of the expressions for e_1 and e_2 have been used to generate the right-side expression for the new equation. Thus, if e_3 is the same as e_1 or e_2, the equation for e_3 replaces one or both of the equations in the equation file that were used to generate the expression for e_3.

The right-side expressions of the equations for e_1 and e_2, or truncated versions of them, are used to create the right-side expression for the new equation. Copies of the expressions for e_1 and e_2 are expanded and possibly truncated to obtain disjunctive normal form expressions for p_1 and p_2. Then, terms in the expression for p_1 that subsume terms in the expression for p_2 are deleted from the expression for p_1. Remaining terms of the expression for p_1 are simplified, by deleting subsuming terms, and factored to produce the right-side expression for the new equation.

If subsuming terms occur in the expression for p_1, they are not retained in the final expression even if terms are deleted from the expression for p_1 prior to simplification. Such terms are deleted either because they subsume a term in the expression for p_2, or because they subsume a term in the expression for p_1.

The right-side expression obtained by a DLTRM call depends on the disjunctive normal form expressions for p_1 and p_2. Equivalent but different disjunctive normal form expressions for p_1 or p_2 can produce different right-side expressions for the new equation. The potential to produce different expressions does not exist unless at least one of the equivalent expressions for p_1 or p_2 contains complement variables--and many applications do not involve complement variables. However, to illustrate that different results can be obtained, consider expression

A*C + C*D + A*B*D (4-18)

for e_1, and expression

A*B + /B*C (4-19)

for e_2. Deleting terms from Exp. 4-18 that subsume terms in Exp. 4-19 produces the expression

$$A*C + C*D. \tag{4-20}$$

However, suppose Exp. 4-19 had been represented as the sum of its prime implicants

$$A*B + /B*C + A*C. \tag{4-21}$$

Deleting terms from Exp. 4-18 that subsume terms in Exp. 4-21 produces the expression

$$C*D. \tag{4-22}$$

Expressions 4-19 and 4-21 for e_2 are different but they are equivalent. Using each of these expressions to delete terms from Exp. 4-18 produces Exps. 4-20 and 4-22, respectively. These expressions are not only different, they are also not equivalent unless $A*C \leq C*D$. A similar result can be shown using a single expression for p_2 and choosing different, equivalent expressions for p_1. The delete term process and the results that it produces are described in detail in Reference 25.

Deleting all of the terms of a disjunctive normal form expression produces the constant /OMEGA. This can happen when truncating e_i to form p_i or when deleting terms from the expression for p_1 that subsume some term in the expression for p_2. Also, expressions for p_1 or p_2 can be constants, i.e., OMEGA or /OMEGA. The results produced by the Delete Term procedure for all combinations of constant and nonconstant expressions for p_1 and p_2 are shown in Table 4.7. The result obtained by deleting terms from a nonconstant expression for p_1 is represented by u in the table.

Table 4.7 Results of DLTRM (p_1, p_2, e_3)

p_1	p_2	New Equation
/OMEGA	/OMEGA	e_3=/OMEGA
/OMEGA	nonconstant	e_3=/OMEGA
/OMEGA	OMEGA	e_3=/OMEGA
nonconstant	/OMEGA	e_3=u (u $\equiv p_1$)
nonconstant	nonconstant	e_3=u
nonconstant	OMEGA	e_3=/OMEGA
OMEGA	/OMEGA	e_3=OMEGA
OMEGA	nonconstant	e_3=OMEGA
OMEGA	OMEGA	e_3=/OMEGA

Printed Output

Some printed output is produced even if the equation file does not contain equations for e_1 and e_2 and an equation for e_3 is not created. If there is no equation for e_1, the message

THERE IS NO EQUATION FOR e_1

is printed, and then execution of the DLTRM call is terminated. If there is an equation for e_1 but not for e_2, the right-side expression for e_1 is expanded into a disjunctive normal form before it is known that the equation file does not contain an equation for e_2. Thus, the printed output for expansion of e_1 occurs; the message

THERE IS NO EQUATION FOR e_2

is printed; and then execution of DLTRM is terminated.

The printed output produced by the execution of a DLTRM call which creates an equation for e_3 is similar to the output produced by the reduction process described in Section 4.1.5, Reduce Equation. The only difference is an additional message printed between the output for expansion and simplification.

The output begins with the printed output produced by expansion of the right-side expression of the equation for e_1. (Expansion of the right-side expression of the equation for e_2 does not produce any printed output.) Then, one of three messages is printed. The messages

NONE OF THE TERMS WERE DELETED

ALL OF THE TERMS WERE DELETED

THERE WERE w TERMS DELETED

indicate how many of the terms in the expression for p_1 were deleted because they subsumed terms in the expression for p_2. In the final message, w is the number of terms deleted.

The printed output produced by simplification and factorization of the terms of the expression for p_1 that remain is the same as it is in the reduction process.

Example

Consider SETS user program:

```
PROGRAM$ DLTRM-EX.
   EQN-2 = (X1 + X4*X5 + X6)*(X2 + /X2*X5)*(X3 + X2*X4).
   REDUCEQN (EQN-2, EQN-2).
   DLTRM (NO-EQN, EQN-2, 1RST-NEW-EQN).
   DLTRM (EQN-2, NO-EQN, 2ND-NEW-EQN).
   PRTEQN (1RST-NEW-EQN, 2ND-NEW-EQN).
   DLT-1 = X1*X6.
   DLT-2 = X3 + X2*X4.
   DLT-3 = X3*(X1 + X6).
   PRTEQNDNF (EQN-2, DLT-1, DLT-2, DLT-3).
   DLTRM (EQN-2, DLT-1, EQN-2-DLT-1).
   PRTEQNDNF (EQN-2-DLT-1).
   DLTRM (EQN-2, DLT-2, EQN-2-DLT-2).
   PRTEQN (EQN-2-DLT-2).
   DLTRM (EQN-2, DLT-3, EQN-2-DLT-3).
   PRTEQNDNF (EQN-2-DLT-3).
```

The equation file is empty when execution of DLTRM-EX begins. The first statement in the SETS user program is an equation statement that enters an equation for EQN-2 into the equation file. A REDUCEQN call redefines the equation for EQN-2 so that it does not contain subsuming terms. Then, two DLTRM calls show the equation file does not contain an equation for NO-EQN, first when NO-EQN occurs as parameter p_1 and then when it occurs as parameter p_2. A PRTEQN call shows an equation was not created for either 1RST-NEW-EQN or 2ND-NEW-EQN. The printed output produced by the two DLTRM calls and the PRTEQN call is as follows:

```
DLTRM (NO-EQN, EQN-2, 1RST-NEW-EQN).

          THERE IS NO EQUATION FOR NO-EQN

     STATEMENT EXECUTION REQUIRED      .001 SECONDS FOR DLTRM


DLTRM (EQN-2, NO-EQN, 2ND-NEW-EQN).

               THE MAXIMUM NUMBER OF TERMS THAT CAN BE
               GENERATED BY EXPANSION IS          8.

               TERMS GENERATED BY EXPANSION
                    5 TERMS CONTAIN  3 VARIABLES
                    3 TERMS CONTAIN  4 VARIABLES
               TOTAL TERMS GENERATED        8.
          EXPANSION TOOK      .006 SECONDS.

          THERE IS NO EQUATION FOR NO-EQN

     STATEMENT EXECUTION REQUIRED      .007 SECONDS FOR DLTRM


PRTEQN (1RST-NEW-EQN, 2ND-NEW-EQN).

          THERE IS NO EQUATION FOR 1RST-NEW-EQN

          THERE IS NO EQUATION FOR 2ND-NEW-EQN

     STATEMENT EXECUTION REQUIRED      .002 SECONDS FOR PRTEQN
```

The next three statements in the SETS user program are equation statements that enter equations for DLT-1, DLT-2, and DLT-3 into the equation file. A PRTEQNDNF call shows the terms of EQN-2 and the three equations that will be used to delete terms from EQN-2. Part of the printed output produced by the PRTEQNDNF call is as follows:

```
PRTEQNDNF (EQN-2, DLT-1, DLT-2, DLT-3).
                •
                •
                •
          TERM   NUMBER OF
         NUMBER  VARIABLES

                              EQN-2 =

            1         3       X6 * X2 * X3 +

            2         3       X1 * X2 * X3 +

            3         3       X4 * X6 * X2 +

            4         3       X4 * X5 * X2 +

            5         3       X1 * X4 * X2 +

            6         4       X5 * X6 * X3 * /X2 +

            7         4       X4 * X5 * X3 * /X2 +

            8         4       X1 * X5 * X3 * /X2
                •
                •
                •
          TERM   NUMBER OF
         NUMBER  VARIABLES

                              DLT-1 =

            1         2       X1 * X6
                •
                •
                •
          TERM   NUMBER OF
         NUMBER  VARIABLES

                              DLT-2 =

            1         1       X3 +

            2         2       X4 * X2
```

```
                    •
                    •
                    •
          TERM   NUMBER OF
         NUMBER  VARIABLES

                            DLT-3 =

             1         2    X1 * X3 +

             2         2    X6 * X3

     STATEMENT EXECUTION REQUIRED      .046 SECONDS FOR PRTEQNDNF
```

The next statement in the SETS user program is a call of DLTRM that creates an equation for EQN-2-DLT-1. None of the terms in the right-side expression of the equation for EQN-2 subsume the single term, X1*X6, in the right-side expression of the equation for DLT-1. Thus, no term is deleted from the expression for EQN-2 and the right-side expression for the new equation is the same as the right-side expression for EQN-2. A PRTEQNDNF call shows the eight terms in the rightside expression of the equation for EQN-2-DLT-1. The printed output produced by the call of PRTEQNDNF is not shown because the eight terms printed are the same as those for EQN-2 which were shown previously. The printed output produced by the call of DLTRM is as follows:

```
DLTRM (EQN-2, DLT-1, EQN-2-DLT-1).

          THE MAXIMUM NUMBER OF TERMS THAT CAN BE
          GENERATED BY EXPANSION IS          8.

          TERMS GENERATED BY EXPANSION
               5 TERMS CONTAIN  3 VARIABLES
               3 TERMS CONTAIN  4 VARIABLES
          TOTAL TERMS GENERATED        8.
     EXPANSION TOOK      .007 SECONDS.

     NONE OF THE TERMS WERE DELETED

          TERMS RETAINED BY SIMPLIFICATION
               5 TERMS CONTAIN  3 VARIABLES
               3 TERMS CONTAIN  4 VARIABLES
          TOTAL TERMS RETAINED        8.
     SIMPLIFICATION TOOK      .002 SECONDS.

     FACTORIZATION TOOK      .019 SECONDS.

STATEMENT EXECUTION REQUIRED      .038 SECONDS FOR DLTRM
```

The next statement in the SETS user program is a call of DLTRM that creates an equation for EQN-2-DLT-2. Every term in the right-side expression of the equation for EQN-2 subsumes one of the two terms, X3 or X2*X4, in the right-side expression of the equation for DLT-2. A call of PRTEQN shows the right-side

expression of the new equation is /OMEGA. The printed output produced by these two procedures is as follows:

```
DLTRM (EQN-2, DLT-2, EQN-2-DLT-2).

                THE MAXIMUM NUMBER OF TERMS THAT CAN BE
                GENERATED BY EXPANSION IS          8.

                TERMS GENERATED BY EXPANSION
                     5 TERMS CONTAIN  3 VARIABLES
                     3 TERMS CONTAIN  4 VARIABLES
                TOTAL TERMS GENERATED        8.
           EXPANSION TOOK      .005 SECONDS.

           ALL OF THE TERMS WERE DELETED

     STATEMENT EXECUTION REQUIRED      .016 SECONDS FOR DLTRM


PRTEQN (EQN-2-DLT-2).

           EQN-2-DLT-2 = /OMEGA


     STATEMENT EXECUTION REQUIRED      .002 SECONDS FOR PRTEQN
```

The next statement in the SETS user program is a call of DLTRM that creates an equation for EQN-2-DLT-3. Four of the eight terms in the right-side expression of the equation for EQN-2 subsume a term in the right-side expression of the equation for DLT-3. A PRTEQNDNF call shows the four terms in the right-side expression of the new equation. The printed output produced by the DLTRM call and part of the output produced by the PRTEQNDNF call is as follows:

```
DLTRM (EQN-2, DLT-3, EQN-2-DLT-3).

                THE MAXIMUM NUMBER OF TERMS THAT CAN BE
                GENERATED BY EXPANSION IS          8.

                TERMS GENERATED BY EXPANSION
                     5 TERMS CONTAIN  3 VARIABLES
                     3 TERMS CONTAIN  4 VARIABLES
                TOTAL TERMS GENERATED        8.
           EXPANSION TOOK      .006 SECONDS.

           THERE WERE     4 TERMS DELETED

                TERMS RETAINED BY SIMPLIFICATION
                     3 TERMS CONTAIN  3 VARIABLES
                     1 TERMS CONTAIN  4 VARIABLES
                TOTAL TERMS RETAINED         4.
           SIMPLIFICATION TOOK      .002 SECONDS.

           FACTORIZATION TOOK      .006 SECONDS.

     STATEMENT EXECUTION REQUIRED      .024 SECONDS FOR DLTRM


PRTEQNDNF (EQN-2-DLT-3).
```

```
 TERM   NUMBER OF
NUMBER  VARIABLES

                    EQN-2-DLT-3 =

   1        3       X4 * X6 * X2 +

   2        3       X4 * X5 * X2 +

   3        3       X1 * X4 * X2 +

   4        4       X4 * X5 * X3 * /X2

STATEMENT EXECUTION REQUIRED      .015 SECONDS FOR PRTEQNDNF
```

This page intentionally left blank.

4.1.9 Write Equation in Disjunctive Normal Form

The Write Equation in Disjunctive Normal Form procedure is used to add an equation to the interface file. The right-side expressions of equations added to the interface file are represented in a disjunctive normal form. Parameters in the call specify equations in the equation file to be added to the interface file. For example, suppose the equation file contains the equation

A=B+C.

Execution of procedure call statement

WRTEQNDNF (A)

causes the right-side expression of the equation for A to be expanded into a disjunctive normal form. Then, the equation for A is added to the interface file and the message

WRITE THE EQUATION FOR A

is printed.

The interface file is used as an input file for other computer programs. In particular, equations produced by SETS user programs can become input to two expression evaluation programs, the Set Evaluation Program (SEP) (Ref. 22) and IMPORTANCE (Ref. 23).

Procedure Call

A call of the Write Equation in Disjunctive Normal Form procedure has the form:

WRTEQNDNF (e_1,e_2,...,e_n).

Parameters, e_i, i=1,2,...,n, specify equations to be added to the interface file.

Processing Achieved

If the equation file does not contain an equation for e_i, the interface file is not changed.

If the equation file contains an equation for e_i, the equation is added to the interface file. The right-side expression is expanded into a disjunctive normal form before the equation is added to the file.

Printed Output

If the equation file does not contain an equation for e_i, the message

THERE IS NO EQUATION FOR e_i

is printed, and processing continues with the next parameter. If the equation file

contains an equation for e_i, the equation is added to the interface file; and the message

WRITE THE EQUATION FOR e_i

is printed. Processing then continues with the next parameter.

Example

Consider SETS user program:

```
PROGRAM$ WRTEQNDNF-EX.
   EQN-1 = /(/X1 + X2*OMEGA)*(X3 + /X1 + X4*(X1 + X5*(X2 + /X3))).
   EQN-2 = (X1 + X4*X5 + X6)*(X2 + /X2*X5)*(X3 + X2*X4).
   WRTEQNDNF (NO-EQN, EQN-1, EQN-2).
```

The equation file and the interface file are empty when execution of WRTEQNDNF-EX begins. The first two statements in the SETS user program are equation statements that enter equations for EQN-1 and EQN-2 into the equation file. The next statement is a call of WRTEQNDNF that shows the equation file does not contain an equation for NO-EQN, and then adds the equations for EQN-1 and EQN-2 to the interface file. The printed output produced by the two equation statements is not important here. It is sufficient to know that the equation file contains equations for EQN-1 and EQN-2. The printed output produced by the call of WRTEQNDNF is as follows:

```
WRTEQNDNF (NO-EQN, EQN-1, EQN-2).

           THERE IS NO EQUATION FOR NO-EQN

           WRITE THE EQUATION FOR EQN-1

           WRITE THE EQUATION FOR EQN-2

     STATEMENT EXECUTION REQUIRED      .020 SECONDS FOR WRTEQNDNF
```

4.2 Block Procedures

Procedures that process blocks do not use or change the value block file. Interactions between block procedures and the equation file and block file are shown in Figure 4.3.

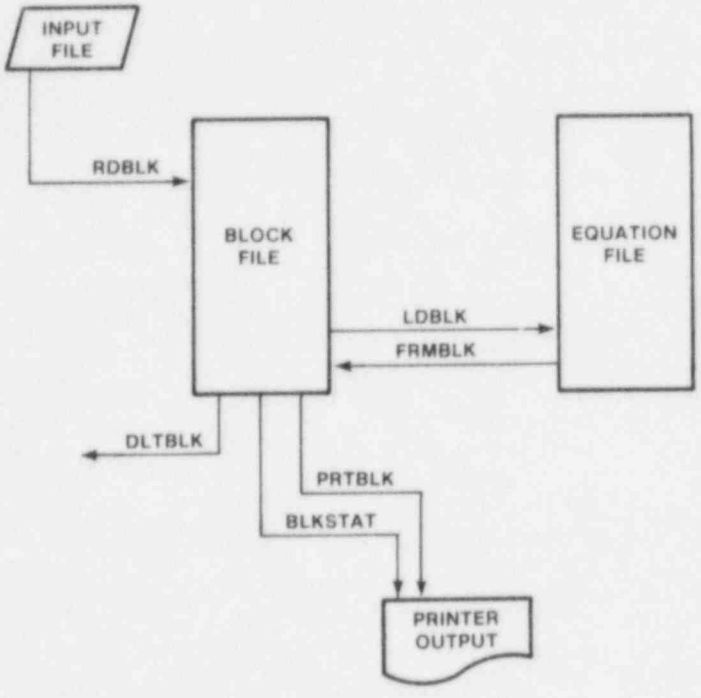


Figure 4.3 Interactions of block procedures with the equation file and block file.

Three of the six procedures used to process blocks change the block file. Procedures RDBLK and FRMBLK enter blocks into the block file. Blocks entered by RDBLK are read from the input file. Blocks entered by FRMBLK are created from equations in the equation file, but the equation file is not changed. A SETS user program error occurs if the name of a new block is the same as the name of a block in the block file (see APPENDIX A, Section 3.3, Numbered Errors, error 60). Procedure DLTBLK deletes blocks from the block file.

Procedure PRTBLK prints the contents of blocks stored in the block file; procedure BLKSTAT lists blocks stored in the block file; and procedure LDBLK loads equations from blocks in the block file into the equation file. Loading equations from a block into the equation file changes the equation file but it does not change the block file.

This page intentionally left blank.

4.2.1 Read Block

The Read Block procedure is used to read equation blocks from the input file and enter them into the block file. Parameters in the call specify the blocks to be read. As the equations of a block are read, they are checked to determine if they are well-defined. If they are well-defined, the block is added to the block file. For example, suppose the input file contains equation block BLK-1 in the form:

```
BLOCK$ BLK-1.
A=B+C.
B=C*(D+E).
```

Execution of procedure call statement

```
RDBLK (BLK-1)
```

reads block BLK-1 from the input file. The name in the header is compared to the parameter in the call to verify that they are the same. The equations for A and B are read and checked. Since both equations are well-defined, block BLK-1 is entered into the block file; and the message

```
EQUATION BLOCK BLK-1
HAS BEEN ADDED TO THE BLOCK FILE
```

is printed. Equation block BLK-1 contains two equations: the equation for A and the equation for B.

Procedure Call

A call of the Read Block procedure has the form:

RDBLK (eb_1,eb_2,...,eb_n).

Parameters eb_i, i=1,2,...,n, specify equation blocks to be read from the input file. There must be at least one block to be read, ($1 \leq i$).

Processing Achieved

An equation block is a collection of one or more Boolean equations. Its input representation begins with a header of the form

BLOCK$ eb.

where eb is the name of the equation block. Boolean equations follow the header. The equations can occur in any order, but each one must be terminated by a period (.). A SETS user program error occurs if there is a header but no equations (see APPENDIX A, Section 3.3, Numbered Errors, error 11).

Parameters eb_i, i=1,2,...,n, are processed one at a time from left to right. As each eb_i is encountered, the next information in the input file is read and processed. The header must occur first, and the name in the header, eb, and parameter eb_i must be the same.

After the header is read and checked, the equations are read one at a time. Each equation is checked to determine if it is well-defined. If there are no errors in the equations, the block is entered into the block file. It has the name from the header and contains all of the equations that follow the header and precede the next end-of-record mark in the input file.

A SETS user program error occurs if the block file already contains a block with the same name as a block that is read (see APPENDIX A, Section 3.3, Numbered Errors, error 60).

Printed Output

As the input representation of equation block eb_i is read, it is also printed and becomes part of the output produced by RDBLK. After the block is read, checked for errors, and added to the block file; the message

EQUATION BLOCK eb_i
HAS BEEN ADDED TO THE BLOCK FILE

is printed.

Example

Consider SETS user program:

```
PROGRAM$ RDBLK-EX.
   DLTBLK.
   BLKSTAT.
   RDBLK (MODEL-1, MODEL-2, MODEL-3).
   BLKSTAT.
```

The first statement in the SETS user program is a call of DLTBLK which initializes the block file and ensures it is empty. The second statement is a call of BLKSTAT which shows the block file is empty. The printed output produced by the BLKSTAT call is as follows:

```
BLKSTAT.

          THE BLOCK FILE IS EMPTY

     STATEMENT EXECUTION REQUIRED       .002 SECONDS FOR BLKSTAT
```

The third statement in the SETS user program is a call of RDBLK that reads equation blocks MODEL-1, MODEL-2, and MODEL-3 and enters them into the block file. Blocks MODEL-1 and MODEL-2 each have three equations and MODEL-3 has one equation. The printed output produced by the RDBLK call is as follows:

```
RDBLK (MODEL-1, MODEL-2, MODEL-3).

          BLOCK$ MODEL-1.
             X1 = Y1*(J1 + K1).
             Y1 = Z1 + J1.
             Z1 = L1 + /(M1 + N1).

     EQUATION BLOCK MODEL-1
     HAS BEEN ADDED TO THE BLOCK FILE
```

```
      BLOCK$ MODEL-2.
         X2 = Y2*(J2 + /K2).
         Y2 = Z2 + L2.
         Z2 = J2*/(K2 + L2).

   EQUATION BLOCK MODEL-2
   HAS BEEN ADDED TO THE BLOCK FILE



      BLOCK$ MODEL-3.
         X3 = J3*/(K3 + L3*(M3 + /K3*N3) + /L3*N3).

   EQUATION BLOCK MODEL-3
   HAS BEEN ADDED TO THE BLOCK FILE

   STATEMENT EXECUTION REQUIRED      .035 SECONDS FOR RDBLK
```

The last statement in RDBLK-EX is another call of BLKSTAT which shows equation blocks MODEL-1, MODEL-2, and MODEL-3 are in the block file. The output produced by this BLKSTAT call is as follows:

```
BLKSTAT.

        THE BLOCK FILE CONTAINS THE FOLLOWING BLOCKS      1.  MODEL-1
                                                          2.  MODEL-2
                                                          3.  MODEL-3

    STATEMENT EXECUTION REQUIRED      .011 SECONDS FOR BLKSTAT
```

This page intentionally left blank.

4.2.2 Print Block

The Print Block procedure is used to print blocks stored in the block file. Parameters in the call specify the blocks to be printed. To print a block means to print all of the equations in the block and, if the block is a fault tree block, to print a Fault Tree Event Table. The table is printed before the equations are printed. It lists each fault tree event and shows its type, its rank, and its inputs and outputs. Block equations are printed in factored form. For example, suppose the block file contains equation block BLK-1 which contains equations:

```
A=B+C
B=C*(D+E)
```

Execution of procedure call statement

```
PRTBLK (BLK-1)
```

causes the equations from block BLK-1 to be printed in the form:

```
* * * * BLOCK EQUATIONS * * * *
  (BLK-1                    )

A = B + C

B = C * ( D + E )
        1       1
```

Procedure Call

A call of the Print Block procedure has the form:

PRTBLK (b_1,b_2,....,b_n).

Parameters b_i, i=1,2,....,n, specify blocks to be printed. There must be at least one block to be printed, ($1 \leq i$).

Processing Achieved

Does not apply.

Printed Output

If the block file does not contain a block for parameter b_i, a SETS user program error occurs (see APPENDIX A, Section 3.3, Numbered Errors, error 37).

If the block file contains a block for parameter b_i, it is either an equation block or a fault tree block. (Fault trees are described in Section 4.4, Fault Tree

Procedures.) If b_i is a fault tree block, a Fault Tree Event Table is printed before the equations are printed. A title of the form

**** FAULT TREE EVENT TABLE ****
(b_i)

is printed first. The table contains the name and type of every event in the fault tree. It also shows how many inputs and outputs each event has (rank), and lists their names. In the table, event numbers begin with 2. (In SETS, the number 1 is reserved for the constant OMEGA; and, even though OMEGA cannot occur in a fault tree, its number cannot be assigned to another event.) Consequently, the fault tree has one less event than the number of the final event in the table.

The order of events in the table is unimportant. Events appear in the order that they are encountered when the fault tree is read (see Section 4.4.1, Read Fault Tree), or in the order determined when several fault trees are merged into one fault tree (see Section 4.4.2, Form New Fault Tree).

A Fault Tree Event Table also shows four indicators used by the Fault Tree Drawing (FTD) program (Ref. 26) and, if there is one, the prefix for each event name. Prefixes and two of the FTD indicators, Generated Set (GS) and Similar Output (SO), do not appear unless the fault tree contains similar trees. The remaining two FTD indicators are slightly more useful. The Plot Separate Tree (PST) indicator identifies intermediate events that are not single-output events. The Plot Transfer (PT) indicator identifies multiple-output intermediate events. For example, an event having a PST indicator and no PT indicator is a top event of the fault tree because it has no outputs.

Consider fault tree EVNT-TBL-FT as shown in Figure 4.4. The Fault Tree Event Table produced by a call of PRTBLK for EVNT-TBL-FT is shown in Figure 4.5.

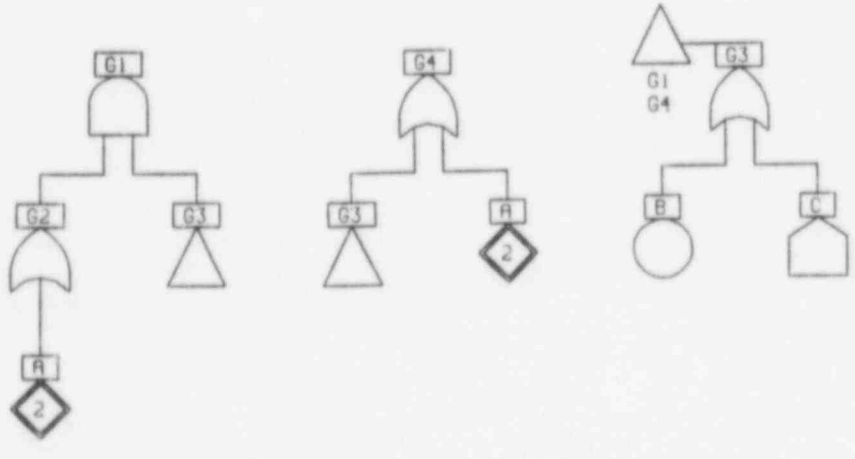


Figure 4.4 Fault tree EVNT-TBL-FT.

* * * * FAULT TREE EVENT TABLE * * * *
(EVNT-TBL-FT)

	EVENT NAME	TYPE	RANK	RELATIONSHIPS	PREFIX	GS	SO	PST	PPT
2	G1	AG	2	IN , G2 IN , G3				X	
3	G2	OG	2	IN , A OUT, G1					
4	G3	OG	4	IN , B IN , C OUT, G1 OUT, G4				X	X
5	A	DE	2	OUT, G2 OUT, G4					
6	B	BE	1	OUT, G3					
7	C	EE	1	OUT, G3					
8	G4	OG	2	IN , G3 IN , A				X	

Figure 4.5 Fault Tree Event Table for fault tree EVNT-TBL-FT.

For equation blocks and fault tree blocks, the equations from block b_i are printed following a title of the form:

* * * * BLOCK EQUATIONS * * * *
(b_i)

Each equation is printed in factored form as described in Section 4.1.1, Print Equation. Operators AND, OR, and NOT are represented by *, +, and /, respectively; and an integer is printed below each parenthesis to aid in matching parentheses in complex expressions.

Example

Consider SETS user program:

```
PROGRAM$ PRTBLK-EX.
   BLKSTAT.
   RDFT (POWER-FAILS).
   PRTBLK (MODEL-1, MODEL-3, POWER-FAILS).
```

The first statement in the SETS user program is a call of BLKSTAT which establishes that the block file contains three blocks when execution of PRTBLK-EX begins. The output produced by the BLKSTAT call is as follows:

```
BLKSTAT.

          THE BLOCK FILE CONTAINS THE FOLLOWING BLOCKS      1.  MODEL-1
                                                            2.  MODEL-2
                                                            3.  MODEL-3

     STATEMENT EXECUTION REQUIRED       .014 SECONDS FOR BLKSTAT
```

The second statement is a call of RDFT that reads fault tree POWER-FAILS and enters it into the block file. The graphic representation of the fault tree is shown in Figure 4.6. (The input representation of the fault tree is printed by the RDFT procedure as the fault tree is read.)

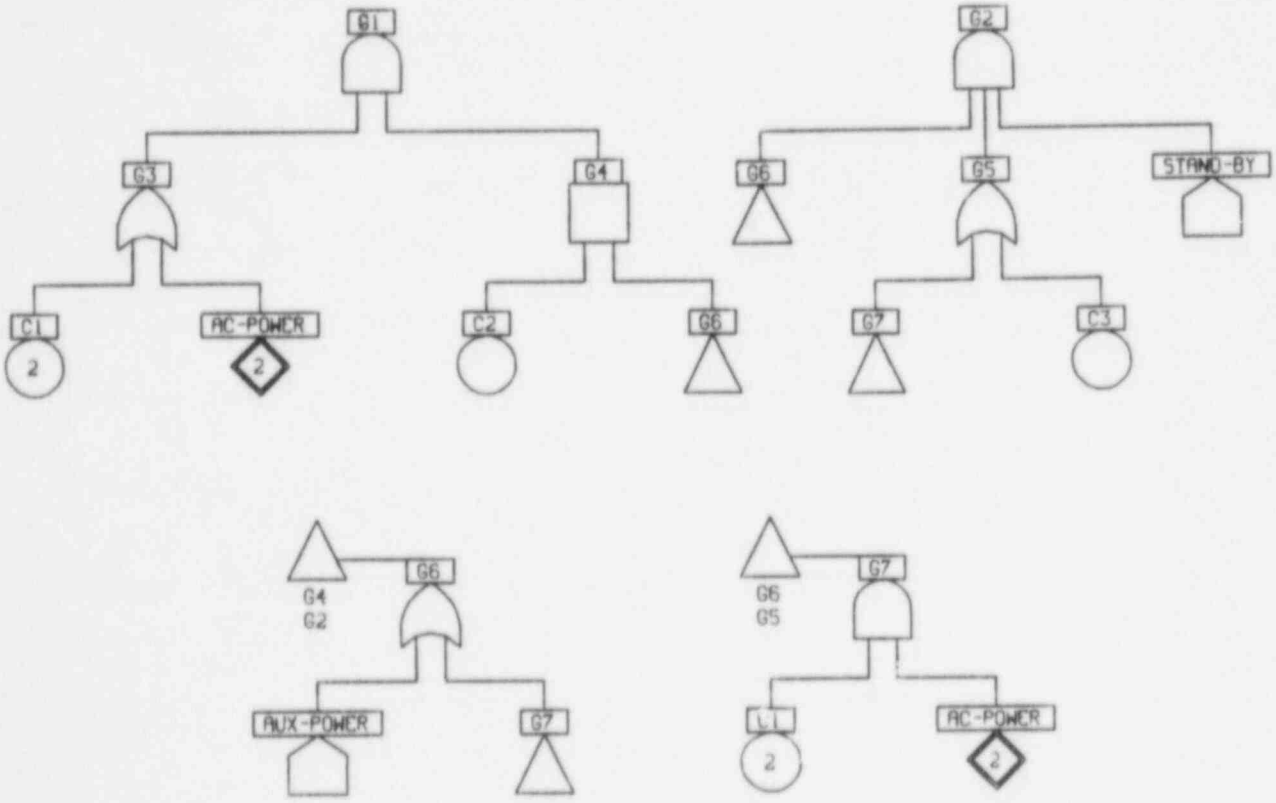


Figure 4.6 Fault tree POWER-FAILS.

The printed output produced by the RDFT call for POWER-FAILS is as follows:

```
RDFT (POWER-FAILS).

          FAULT TREE$ POWER-FAILS.
          AG$ G1.  IN$ G3, G4.
          OG$ G3.  IN$ C1, AC-POWER.  OUT$ G1.
          SG$ G4 = C2 * /G6.  IN$ C2, G6.  OUT$ G1.
          OG$ G6.  IN$ AUX-POWER, G7.  OUT$ G4, G2.
          AG$ G7.  IN$ C1, AC-POWER.  OUT$ G6, G5.
          AG$ G2.  IN$ G6, G5, STAND-BY.
          OG$ G5.  IN$ G7, C3.  OUT$ G2.
          BE$ C1.  OUT$ G3, G7.
          DE$ AC-POWER.  OUT$ G3, G7.
          BE$ C2.  OUT$ G4.
          BE$ C3.  OUT$ G5.
          EE$ AUX-POWER.  OUT$ G6.
          EE$ STAND-BY.  OUT$ G2.

          THERE ARE NO CYCLES IN POWER-FAILS

     FAULT TREE BLOCK POWER-FAILS
     HAS BEEN ADDED TO THE BLOCK FILE

     STATEMENT EXECUTION REQUIRED      .032 SECONDS FOR RDFT
```

The final statement in the SETS user program is a call of PRTBLK which prints equation blocks MODEL-1 and MODEL-3, and fault tree block POWER-FAILS. MODEL-1 has three equations and MODEL-3 has one equation. Fault tree POWER-FAILS has thirteen events as shown in the Fault Tree Event Table. There are seven intermediate events and, therefore, seven equations. The printed output produced by PRTBLK is as follows:

```
PRTBLK (MODEL-1, MODEL-3, POWER-FAILS).


                                        * * * * BLOCK EQUATIONS * * * *
                                             (MODEL-1         )


          X1 = Y1 * ( J1 + K1 )
                    1        1

          Y1 = Z1 + J1


          Z1 = L1 + /( M1 + N1 )
                    1        1


                                        * * * * BLOCK EQUATIONS * * * *
                                             (MODEL-3         )


          X3 = J3 * /( K3 + L3 * ( M3 + /K3 * N3 ) + /L3 * N3 )
                    1          2              2           1
```

*** * * * FAULT TREE EVENT TABLE * * * ***
(POWER-FAILS)

	EVENT NAME	TYPE	RANK	RELATIONSHIPS	PREFIX	GS	SO	PST	PT
2	G1	AG	2	IN , G3 IN , G4				X	
3	G3	OG	3	IN , C1 IN , AC-POWER OUT, G1					
4	G4	SG	3	IN , C2 IN , G6 OUT, G1					
5	C1	BE	2	OUT, G3 OUT, G7					
6	AC-POWER	DE	2	OUT, G3 OUT, G7					
7	C2	BE	1	OUT, G4					
8	G6	OG	4	IN , AUX-POWER IN , G7 OUT, G4 OUT, G2				X	X
9	AUX-POWER	EE	1	OUT, G6					
10	G7	AG	4	IN , C1 IN , AC-POWER OUT, G6 OUT, G5				X	X
11	G2	AG	3	IN , G6 IN , G5 IN , STAND-BY				X	
12	G5	OG	3	IN , G7 IN , C3 OUT, G2					
13	STAND-BY	EE	1	OUT, G2					
14	C3	BE	1	OUT, G5					

**** BLOCK EQUATIONS ****
(POWER-FAILS)

G1 = G3 * G4

G3 = C1 + AC-POWER

G4 = C2 * /G6

G6 = AUX-POWER + G7

G7 = C1 * AC-POWER

G2 = G6 * G5 * STAND-BY

G5 = G7 + C3

STATEMENT EXECUTION REQUIRED .033 SECONDS FOR PRTBLK

This page intentionally left blank.

4.2.3 Block Status

The Block Status procedure is used to list blocks stored in the block file. For example, suppose the block file contains three blocks. Execution of procedure call statement

BLKSTAT

produces a list of the blocks in the form:

THE BLOCK FILE CONTAINS THE FOLLOWING BLOCKS
1. BLK-1
2. EVNT-TBL-FT
3. TRANS-EQNS

Procedure Call

A call of the Block Status procedure has the form:

BLKSTAT.

A call of BLKSTAT never has parameters.

Processing Achieved

Does not apply.

Printed Output

If there are no blocks in the block file, the message

THE BLOCK FILE IS EMPTY

is printed. Otherwise, a list of the blocks in the block file is printed in the form:

THE BLOCK FILE CONTAINS THE FOLLOWING BLOCKS
1. b_1
2. b_2
. .
. .
. .
n. b_n

Example

Consider SETS user program:

```
PROGRAM$ BLKSTAT-EX.
   BLKSTAT.
   RDBLK (GIVEN-COND-EQNS).
   BLKSTAT.
```

The first statement in the SETS user program is a call of BLKSTAT which shows the block file contains four blocks when execution of BLKSTAT-EX begins. The printed output produced by this call of BLKSTAT is as follows:

```
BLKSTAT.

          THE BLOCK FILE CONTAINS THE FOLLOWING BLOCKS      1.  MODEL-1
                                                            2.  MODEL-2
                                                            3.  MODEL-3
                                                            4.  POWER-FAILS

     STATEMENT EXECUTION REQUIRED       .020 SECONDS FOR BLKSTAT
```

The second statement in BLKSTAT-EX reads equation block GIVEN-COND-EQNS and adds it to the block file. Then, another call of BLKSTAT shows there are five blocks in the block file. The output produced by these two procedures is as follows:

```
RDBLK (GIVEN-COND-EQNS).

          BLOCK$ GIVEN-COND-EQNS.
          AUX-POWER = OMEGA.
          STAND-BY = /OMEGA.

     EQUATION BLOCK GIVEN-COND-EQNS
     HAS BEEN ADDED TO THE BLOCK FILE

     STATEMENT EXECUTION REQUIRED       .022 SECONDS FOR RDBLK


BLKSTAT.

          THE BLOCK FILE CONTAINS THE FOLLOWING BLOCKS      1.  MODEL-1
                                                            2.  MODEL-2
                                                            3.  MODEL-3
                                                            4.  POWER-FAILS
                                                            5.  GIVEN-COND-EQNS

     STATEMENT EXECUTION REQUIRED       .020 SECONDS FOR BLKSTAT
```

4.2.4 Delete Block

The Delete Block procedure is used to delete blocks from the block file. For example, suppose the block file contains blocks:

BLK-1
EVNT-TBL-FT
TRANS-EQNS

Execution of procedure call statement

DLTBLK (EVNT-TBL-FT)

deletes fault tree block EVNT-TBL-FT from the block file and leaves blocks BLK-1 and TRANS-EQNS in the block file.

Procedure Call

A call of the Delete Block procedure has one of the forms:

a. DLTBLK.

b. DLTBLK (b_1,b_2,...,b_n).

Parameters b_i, i=1,2,...,n, specify blocks to be deleted from the block file.

Processing Achieved

The specified blocks are deleted from the block file. If the parameter part of a DLTBLK call is empty (form a), every block is deleted from the block file; but if the parameter part is not empty (form b), only the blocks specified in the call are deleted from the block file. If the block file does not contain a block for a parameter specified in a DLTBLK call, the parameter has no effect and processing continues with the next parameter.

A call of DLTBLK without parameters must be used with care; it causes every block in the block file to be deleted.

Printed Output

A DLTBLK call produces only the standard printed output for a procedure call statement.

Example

Consider SETS user program:

```
PROGRAM$ DLTBLK-EX.
   BLKSTAT.
   DLTBLK (MODEL-2, POWER-FAILS).
   BLKSTAT.
   DLTBLK.
   BLKSTAT.
```

The first statement in the SETS user program is a call of BLKSTAT which shows the block file contains five blocks when execution of DLTBLK-EX begins. Part of the printed output produced by this call of BLKSTAT is as follows:

```
          THE BLOCK FILE CONTAINS THE FOLLOWING BLOCKS      1.  MODEL-1
                                                            2.  MODEL-2
                                                            3.  MODEL-3
                                                            4.  POWER-FAILS
                                                            5.  GIVEN-COND-EQNS
```

The second statement in the SETS user program is a DLTBLK call which deletes two blocks, MODEL-2 and POWER-FAILS. The output produced by the DLTBLK call is as follows:

```
DLTBLK (MODEL-2, POWER-FAILS).

     STATEMENT EXECUTION REQUIRED       .028 SECONDS FOR DLTBLK
```

Another call of BLKSTAT shows three blocks remain in the block file. Part of the output for this call of BLKSTAT is as follows:

```
          THE BLOCK FILE CONTAINS THE FOLLOWING BLOCKS      1.  MODEL-1
                                                            2.  MODEL-3
                                                            3.  GIVEN-COND-EQNS
```

The last two statements in the SETS user program are a DLTBLK call without parameters and another call of BLKSTAT. The DLTBLK call deletes every block from the block file, and the BLKSTAT call shows the block file is empty. The printed output produced by these procedures is as follows:

```
DLTBLK.

     STATEMENT EXECUTION REQUIRED       .004 SECONDS FOR DLTBLK


BLKSTAT.

          THE BLOCK FILE IS EMPTY

     STATEMENT EXECUTION REQUIRED       .002 SECONDS FOR BLKSTAT
```

4.2.5 Form Block

The Form Block procedure is used to create an equation block and enter it into the block file. The new block is comprised of some or all of the equations from the equation file. For example, suppose the equation file contains two equations:

A=X*Y
C=Y*/Z

Execution of procedure call statement

FRMBLK (ALL-EQNS)

forms an equation block named ALL-EQNS and adds it to the block file. The new block contains every equation in the equation file, i.e., the equations for A and C.

Procedure Call

A call of the Form Block procedure has one of the forms:

a. FRMBLK (eb).

b. FRMBLK (eb* O).

Parameter eb is the name of a new equation block which contains some or all of the equations from the equation file.

Option O specifies ways to select a subset of the equations in the equation file for inclusion in the block.

Processing Achieved

A new equation block is formed from equations in the equation file and entered into the block file. Unless an option is specified (see this procedure, Options), the block contains every equation in the equation file at the time the block is formed. The equation file is not changed when the block is formed and added to the block file.

If the equation file is empty, a SETS user program error occurs (see APPENDIX A, Section 3.3, Numbered Errors, error 11).

Options

If the FRMBLK call contains an option (form b), the new block contains a subset of the equations in the equation file. Options that can occur in a FRMBLK call are the only option and the except option. A FRMBLK call can contain only one option.

An only option has the form:

ONLY$ $e_1,e_2,\ldots,e_r$

The only option identifies equations in the equation file to be included in the new block. When an only option is used, the new block cannot have an equation that is not specified in the option. If a variable does not have an equation in the equation file, its occurrence in an only option has no effect. A SETS user program error occurs if none of the variables in an only option have equations in the equation file (see APPENDIX A, Section 3.3, Numbered Errors, error 11).

An except option has the form:

EXCEPT$ $e_1,e_2,\ldots,e_r$

The except option identifies equations in the equation file that are not to be included in the new block. The new block contains all of the equations in the equation file except those specified in the option. If a variable does not have an equation in the equation file, its occurrence in an except option has no effect. A SETS user program error occurs if every equation in the equation file is excepted (see APPENDIX A, Section 3.3, Numbered Errors, error 11).

Printed Output

A FRMBLK call creates equation block eb and enters it into the block file. After the block is added to the block file, the message

EQUATION BLOCK eb
HAS BEEN ADDED TO THE BLOCK FILE

is printed.

Example

Consider SETS user program:

```
PROGRAM$ FRMBLK-EX.
   RDBLK (MODEL-1, MODEL-2, MODEL-3).
   LDBLK (MODEL-1, MODEL-2, MODEL-3).
   FRMBLK (ALL-MODELS).
   PRTBLK (ALL-MODELS).
   FRMBLK (Y-EQNS* ONLY$ Y1, Y2, Y3).
   FRMBLK (WITHOUT-Z-EQNS* EXCEPT$ Z1, Z2, Z3).
   PRTBLK (Y-EQNS, WITHOUT-Z-EQNS).
```

The equation file is empty when execution of FRMBLK-EX begins. The first statement in FRMBLK-EX is a RDBLK call that reads equation blocks MODEL-1, MODEL-2, and MODEL-3 and enters them into the block file. The printed output produced by the RDBLK call is as follows:

```
RDBLK (MODEL-1, MODEL-2, MODEL-3).

          BLOCK$ MODEL-1.
             X1 = Y1*(J1 + K1).
             Y1 = Z1 + J1.
             Z1 = L1 + /(M1 + N1).

     EQUATION BLOCK MODEL-1
     HAS BEEN ADDED TO THE BLOCK FILE
```

```
      BLOCK$ MODEL-2.
         X2 = Y2*(J2 + /K2).
         Y2 = Z2 + L2.
         Z2 = J2*/(K2 + L2).

   EQUATION BLOCK MODEL-2
   HAS BEEN ADDED TO THE BLOCK FILE



      BLOCK$ MODEL-3.
         X3 = J3*/(K3 + L3*(M3 + /K3*N3) + /L3*N3).

   EQUATION BLOCK MODEL-3
   HAS BEEN ADDED TO THE BLOCK FILE

   STATEMENT EXECUTION REQUIRED      .038 SECONDS FOR RDBLK
```

The second statement in the SEIS user program is a call of LDBLK which loads the equations from MODEL-1, MODEL-2, and MODEL-3 into the equation file. The next statement in FRMBLK-EX is a call of FRMBLK without an option. This call creates equation block ALL-MODELS and adds it to the block file. A call of PRTBLK shows the block contains every equation in the equation file. The printed output produced by these three procedures is as follows:

```
LDBLK (MODEL-1, MODEL-2, MODEL-3).

   STATEMENT EXECUTION REQUIRED      .007 SECONDS FOR LDBLK


FRMBLK (ALL-MODELS).

   EQUATION BLOCK ALL-MODELS
   HAS BEEN ADDED TO THE BLOCK FILE

   STATEMENT EXECUTION REQUIRED      .018 SECONDS FOR FRMBLK


PRTBLK (ALL-MODELS).

                                    * * * * BLOCK EQUATIONS * * * *
                                          (ALL-MODELS      )


         X1 = Y1 * ( J1 + K1 )
                   1        1

         Y1 = Z1 + J1


         Z1 = L1 + /( M1 + N1 )
                   1        1

         X2 = Y2 * ( J2 + /K2 )
                   1        1

         Y2 = Z2 + L2


         Z2 = J2 * /( K2 + L2 )
                   1        1

         X3 = J3 * /( K3 + L3 * ( M3 + /K3 * N3 ) + /L3 * N3 )
                   1          2              2           1

   STATEMENT EXECUTION REQUIRED      .018 SECONDS FOR PRTBLK
```

The next two statements in the SETS user program are calls of FRMBLK, and each call contains an option. The first call has an only option which specifies block Y-EQNS is to contain only the equations for Y1, Y2, and Y3. However, the equation file does not have an equation for Y3, and the occurrence of this variable in the option has no effect. The second FRMBLK call has an except option which specifies block WITHOUT-Z-EQNS is to contain every equation in the equation file except the equations for Z1, Z2, and Z3. Since the equation file does not have an equation for Z3, the occurrence of this variable in the option has no effect. The last statement in the SETS user program is a call of PRTBLK that prints the equations from blocks Y-EQNS and WITHOUT-Z EQNS. The printed output produced by the two FRMBLK calls and the PRTBLK call is as follows:

```
FRMBLK (Y-EQNS* ONLY$ Y1, Y2, Y3).

      EQUATION BLOCK Y-EQNS
      HAS BEEN ADDED TO THE BLOCK FILE

      STATEMENT EXECUTION REQUIRED      .020 SECONDS FOR FRMBLK


FRMBLK (WITHOUT-Z-EQNS* EXCEPT$ Z1, Z2, Z3).

      EQUATION BLOCK WITHOUT-Z-EQNS
      HAS BEEN ADDED TO THE BLOCK FILE

      STATEMENT EXECUTION REQUIRED      .025 SECONDS FOR FRMBLK


PRTBLK (Y-EQNS, WITHOUT-Z-EQNS).

                                        * * * * BLOCK EQUATIONS * * * *
                                              (Y-EQNS          )


            Y1 = Z1 + J1

            Y2 = Z2 + L2


                                        * * * * BLOCK EQUATIONS * * * *
                                              (WITHOUT-Z-EQNS  )


            X1 = Y1 * ( J1 + K1 )
                      1        1

            Y1 = Z1 + J1

            X2 = Y2 * ( J2 + /K2 )
                      1         1

            Y2 = Z2 + L2

            X3 = J3 * /( K3 + L3 * ( M3 + /K3 * N3 ) + /L3 * N3 )
                       1           2               2           1

      STATEMENT EXECUTION REQUIRED      .024 SECONDS FOR PRTBLK
```

4.2.6 Load Block

The Load Block procedure is used to load equations from blocks in the block file into the equation file. Parameters in the call specify the blocks to be loaded. For example, suppose the equation file contains two equations

A=X*Y
C=Y*/Z

and the block file contains block BLK-1 which contains equations:

A=B+C
B=C*(D+E)

Execution of procedure call statement

LDBLK (BLK-1)

loads the equations in block BLK-1, i.e., the equations for A and B, into the equation file. After BLK-1 is loaded, the equation file contains three equations:

A=B+C
B=C*(D+E)
C=Y*/Z

Procedure Call

A call of the Load Block procedure has the form:

LDBLK (b_1,b_2,...,b_n).

Parameters b_i, i=1,2,...,n, specify blocks whose equations are to be loaded into the equation file. There must be at least one block to be loaded, ($1 \leq i$).

Processing Achieved

Equations from blocks in the block file are loaded into the equation file. Parameters b_i, i=1,2,...,n, are processed one at a time from left to right. As each parameter, b_i, is encountered, the corresponding block is located in the block file. If the block file does not contain block b_i, a SETS user program error occurs (see APPENDIX A, Section 3.3, Numbered Errors, error 37). Otherwise, the equations in block b_i are loaded into the equation file.

To know what equations are in the equation file after block b_i is loaded, the contents of b_i and the contents of the equation file before b_i is loaded must be known. If b_i does not have an equation for X and the equation file does, loading b_i does not change the equation for X in the equation file. If b_i has an equation for X, it will be the equation for X in the equation file after b_i is loaded. In the latter case, if the equation file has an equation for X before b_i is loaded, it is replaced by the equation for X from b_i.

Printed Output

A LDBLK call produces only the standard printed output for a procedure call statement.

Example

Consider SETS user program:

```
PROGRAM$ LDBLK-EX.
   BLKSTAT.
   PRTBLK (MODEL-1, MODEL-3).
   Y1=/Z1 + Z1*/J1.
   Y3=/Z3 + Z3*/J3.
   Z1=OMEGA.
   Z3=OMEGA.
   PRTEQN (Y1, Y3, Z1, Z3).
   LDBLK (MODEL-1, MODEL-3).
   PRTEQN (X1, Y1, Z1, X3, Y3, Z3).
```

The equation file is empty when execution of LDBLK-EX begins. The first statement in the SETS user program is a call of BLKSTAT which establishes the block file contains six blocks when execution of LDBLK-EX begins. Part of the printed output produced by the BLKSTAT call is as follows:

```
THE BLOCK FILE CONTAINS THE FOLLOWING BLOCKS      1.  MODEL-1
                                                  2.  MODEL-2
                                                  3.  MODEL-3
                                                  4.  ALL-MODELS
                                                  5.  Y-EQNS
                                                  6.  WITHOUT-Z-EQNS
```

The second statement in LDBLK-EX prints the equations from two of the blocks in the block file, MODEL-1 and MODEL-3. The printed output produced by this PRTBLK call is as follows:

```
PRTBLK (MODEL-1, MODEL-3).


                                    * * * * BLOCK EQUATIONS * * * *
                                          (MODEL-1         )


      X1 = Y1 * ( J1 + K1 )
                1        1

      Y1 = Z1 + J1

      Z1 = L1 + /( M1 + N1 )
                1        1


                                    * * * * BLOCK EQUATIONS * * * *
                                          (MODEL-3         )


      X3 = J3 * /( K3 + L3 * ( M3 + /K3 * N3 ) + /L3 * N3 )
                1            2              2           1

 STATEMENT EXECUTION REQUIRED      .012 SECONDS FOR PRTBLK
```

Four equation statements are used to establish equations for Y1, Y3, Z1, and Z3. The sixth statement in the SETS user program is a call of PRTEQN which shows the equation file contains these equations. The printed output produced by this call of PRTEQN is as follows:

```
PRTEQN (Y1, Y3, Z1, Z3).

          Y1 = /Z1 + Z1 * /J1

          Y3 = /Z3 + Z3 * /J3

          Z1 = OMEGA

          Z3 = OMEGA

     STATEMENT EXECUTION REQUIRED      .006 SECONDS FOR PRTEQN
```

The next statement in the SETS user program is a call of LDBLK which loads the equations from equation blocks MODEL-1 and MODEL-3 into the equation file. The equation file contains equations for Y1, Y3, Z1, and Z3 when MODEL-1, which contains equations for X1, Y1, and Z1, is loaded. The equation for X1 from MODEL-1 is added to the equation file; and the equations for Y1 and Z1 from MODEL-1 replace the existing equations for these variables in the equation file. After MODEL-1 is loaded, the equation file has equations for X1, Y1, Z1, Y3, and Z3. The equations for Y3 and Z3 are not changed by loading block MODEL-1. Equation block MODEL-3 contains only one equation. When MODEL-3 is loaded, the equation for X3 is added to the equation file.

The final statement in LDBLK-EX is a PRTEQN call which shows the equation file has equations for X1, Y1, Z1, X3, Y3, and Z3. Loading blocks MODEL-1 and MODEL-3 adds equations for X1 and X3; replaces equations for Y1 and Z1; and does not change equations for Y3 and Z3. The printed output produced by the LDBLK and PRTEQN calls is as follows:

```
LDBLK (MODEL-1, MODEL-3).

     STATEMENT EXECUTION REQUIRED      .019 SECONDS FOR LDBLK


PRTEQN (X1, Y1, Z1, X3, Y3, Z3).

          X1 = Y1 * ( J1 + K1 )
                    1       1

          Y1 = Z1 + J1

          Z1 = L1 + /( M1 + N1 )
                    1       1

          X3 = J3 * /( K3 + L3 * ( M3 + /K3 * N3 ) + /L3 * N3 )
                    1            2              2           1

          Y3 = /Z3 + Z3 * /J3

          Z3 = OMEGA

     STATEMENT EXECUTION REQUIRED      .008 SECONDS FOR PRTEQN
```

4.3 Value Block Procedures

Procedures that process value blocks do not use or change the equation file or the block file. Interactions between value block procedures and the value block file are shown in Figure 4.7.

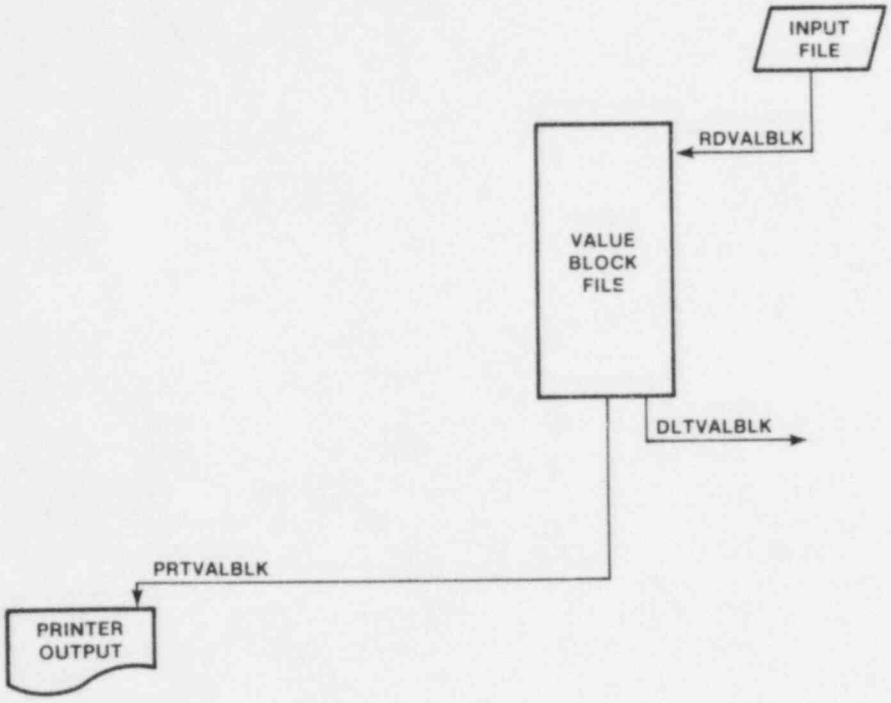


Figure 4.7 Interactions of value block procedures with the value block file.

Two of the three procedures used to process value blocks change the value block file. Procedure RDVALBLK reads value blocks from the input file and enters them into the value block file. If a value block being read and a value block in the file have the same name, the value block in the file is changed to include the new variable values. Procedure DLTVALBLK deletes value blocks from the value block file.

The PRTVALBLK procedure prints value blocks, but it does not change the value block file.

4.3.1 Read Value Block

The Read Value Block procedure is used to read value blocks from the input file and enter them into the value block file. Parameters in the call specify the value blocks to be read. Variable values are checked as they are read to determine if they are both well-defined and nonnegative. If there are no errors, the new value block is added to the value block file. In the event a value block with the same name already exists in the file, it is changed to include the new variable values. For example, suppose the value block file does not have a value block named VLBLK-1, but the input file contains VLBLK-1 in the form:

```
VALUE BLOCK$ VLBLK-1.
1.0E-2 $ A, B, D $
2.6E-4 $ C $
```

Execution of procedure call statement

```
RDVALBLK (VLBLK-1)
```

reads VLBLK-1 from the input file. The header is read first and the name in the header is compared to the parameter in the call to verify they are the same. Since the value block file does not contain a value block named VLBLK-1, the message

```
CREATE VALUE BLOCK VLBLK-1
```

is printed before variable values in VLBLK-1 are read. Then, values are read and, since they are well-defined and nonnegative, value block VLBLK-1 is entered into the value block file. Finally, the message

```
VALUE BLOCK VLBLK-1
HAS BEEN ADDED TO THE VALUE BLOCK FILE
```

is printed. Value block VLBLK-1 contains four variable values: A, B, and D have the value 1×10^{-2}, and C has the value 2.6×10^{-4}.

Procedure Call

A call of the Read Value Block procedure has the form:

RDVALBLK (vb_1,vb_2,...,vb_n).

Parameters vb_i, i=1,2,...,n, specify value blocks to be read from the input file. There must be at least one value block to be read, ($1 \leq i$).

Processing Achieved

A value block contains one or more variables each having a single nonnegative value. The input representation of a value block begins with a header of the form

VALUE BLOCK$ vb.

where vb is the name of the value block. Value assignments follow the header. Each value assignment has the form:

⟨number⟩ $ ⟨variable list⟩ $

The number which begins the value assignment is assigned to every variable in the variable list. If a variable occurs in more than one value assignment, it is assigned the value from the last value assignment in which it occurs. For example,

```
VALUE BLOCK$ INIT-COND.
.007 $       W, Y, Z $
1.0 E-2 $    X $
4 $          V, W $
```

assigns the value .007 to Y and Z; .01 to X; and 4 to V and W.

A SETS user program error occurs if a value assignment has an empty variable list (see APPENDIX A, Section 3.3, Numbered Errors, error 34).

Complement variables cannot be assigned values. Instead, their values are determined by the computation type or, for PROBABILITY, from the values of the corresponding noncomplement variables. Also, variable values are restricted according to the computation involved. Value block INIT-COND, for example, cannot be used for PROBABILITY or PRODUCT computations because all of its values are not ≤ 1 or ≥ 1, respectively. Complement variable values and value restrictions are described in Section 4.1.6, Compute Term Value, Processing Achieved.

Parameters vb_i, i=1,2,...,n, in a RDVALBLK call are processed one at a time from left to right. As each vb_i is encountered, the next information in the input file is read and processed. The value block header must occur first, and parameter vb_i and the name in the header must be the same. After the header is read, it is determined if a new value block will be created or an existing value block will be changed. If the value block file does not have a value block with name vb_i, a new value block will be created. However, if value block vb_i already exists in the value block file, it will be changed to include variable values from value block vb_i in the input file.

If value block vb_i is being changed, every variable value from value block vb_i in the value block file is obtained and treated as if they have already been read as part of value block vb_i in the input file. Once the existing values have been obtained, the processing to change a value block is the same as the processing to create a new value block. Value assignments in vb_i in the input file are read one at a time and checked to determine if they are well-defined and nonnegative. Every variable in a value assignment is assigned the number in the value assignment as its value. If a variable already has a value from a previous value assignment, the new value replaces the old value. Thus, the value for a variable will be the number from the last value assignment in which the variable occurs. After every value assignment has been processed, value block vb_i is entered into the value block file. New vb_i replaces old vb_i in the value block file if value block vb_i was changed.

Printed Output

As the input representation of value block vb_i is read, it is also printed and becomes part of the printed output produced by RDVALBLK. However, between the header and the first value assignment, a message is printed. If a new value block will be created, the message

CREATE VALUE BLOCK vb_i

is printed, but if an existing value block will be changed, the message

CHANGE VALUE BLOCK vb_i

is printed. Following this message, value assignments are read and printed one at a time.

After the value assignments of vb_i in the input file have been read and checked to determine if they are well-defined and nonnegative, the message

VALUE BLOCK vb_i
HAS BEEN ADDED TO THE VALUE BLOCK FILE

is printed.

Example

Consider SETS user program:

```
PROGRAM$ RDVALBLK-EX.
   DLTVALBLK.
   RDVALBLK (VB-PROBS, VB-COSTS).
   PRTVALBLK (VB-PROBS, VB-COSTS).
   RDVALBLK (VB-PROBS).
   PRTVALBLK (VB-PROBS).
   RDVALBLK (SPECIAL-VALUES).
   PRTVALBLK (SPECIAL-VALUES).
```

The first statement in the SETS user program is a DLTVALBLK call without parameters that initializes the value block file and ensures it is empty. The second statement is a call of RDVALBLK that reads value blocks VB-PROBS and VB-COSTS and enters them into the value block file. These value blocks are used in the examples for COMTRMVAL (see Section 4.1.6, Compute Term Value, Example, Table 4.5) and TRNTRMVAL (see Section 4.1.7, Truncate on Term Value, Example). The printed output produced by the RDVALBLK call is as follows:

```
RDVALBLK (VB-PROBS, VB-COSTS).

          VALUE BLOCK$ VB-PROBS.
          CREATE VALUE BLOCK VB-PROBS
             7.2 E-3 $  X1, X6, X7 $
             3   E-2 $  X2, X5 $
             1.0 E-1 $  X4 $

     VALUE BLOCK VB-PROBS
     HAS BEEN ADDED TO THE VALUE BLOCK FILE
```

```
VALUE BLOCK$ VB-COSTS.
CREATE VALUE BLOCK VB-COSTS
17   $  X1 $
 2   $  X2 $
 5   $  X3 $
 2   $  X4 $
 7.3 $  X5 $
 7.3 $  X6 $
17   $  X7 $

VALUE BLOCK VB-COSTS
HAS BEEN ADDED TO THE VALUE BLOCK FILE

STATEMENT EXECUTION REQUIRED      .021 SECONDS FOR RDVALBLK
```

The third statement in RDVALBLK-EX is a call of PRTVALBLK which shows value blocks VB-PROBS and VB-COSTS are in the value block file. The printed output for this call of PRTVALBLK is as follows:

```
PRTVALBLK (VB-PROBS, VB-COSTS).

                    * * * * VALUE BLOCK TABLE * * * *
                            (VB-PROBS        )

     VARIABLE          VALUE                      VARIABLE          VALUE

     X1          7.200000000000E-03               X6          7.200000000000E-03
     X7          7.200000000000E-03               X2          3.000000000000E-02
     X5          3.000000000000E-02               X4          1.000000000000E-01


                    * * * * VALUE BLOCK TABLE * * * *
                            (VB-COSTS        )

     VARIABLE          VALUE                      VARIABLE          VALUE

     X1          1.700000000000E+01               X2          2.000000000000E+00
     X3          5.000000000000E+00               X4          2.000000000000E+00
     X5          7.300000000000E+00               X6          7.300000000000E+00
     X7          1.700000000000E+01

STATEMENT EXECUTION REQUIRED      .009 SECONDS FOR PRTVALBLK
```

Value block VB-PROBS is incorrect. Variable X3 does not have a value and variable X6 has the wrong value. The next statement in the SETS user program is another call of RDVALBLK which changes value block VB-PROBS in the value block file. The value 7.2×10^{-3} is added for variable X3, and the value for X6 is changed from 7.2×10^{-3} to 2.7×10^{-2}. A call of PRTVALBLK shows value block

VB-PROBS has been corrected. The printed output produced by the RDVALBLK and PRTVALBLK calls is as follows:

```
RDVALBLK (VB-PROBS).

          VALUE BLOCK$ VB-PROBS.
          CHANGE VALUE BLOCK VB-PROBS
             7.2 E-3 $  X3 $
             2.7 E-2 $  X6 $

     VALUE BLOCK VB-PROBS
     HAS BEEN ADDED TO THE VALUE BLOCK FILE

     STATEMENT EXECUTION REQUIRED      .020 SECONDS FOR RDVALBLK


PRTVALBLK (VB-PROBS).

                                  * * * * VALUE BLOCK TABLE * * * *
                                          (VB-PROBS        )

          VARIABLE            VALUE                      VARIABLE            VALUE

        X1                7.200000000000E-03          X6                2.700000000000E-02
        X7                7.200000000000E-03          X2                3.000000000000E-02
        X5                3.000000000000E-02          X4                1.000000000000E-01
        X3                7.200000000000E-03

     STATEMENT EXECUTION REQUIRED      .006 SECONDS FOR PRTVALBLK
```

The last two statements in RDVALBLK-EX are calls of RDVALBLK and PRTVALBLK which read and print value block SPECIAL-VALUES. This value block contains numbers for which $d_i+d_f = 14$, and numbers with a very small and a very large exponent. The output for these two procedures is as follows:

```
RDVALBLK (SPECIAL-VALUES).

          VALUE BLOCK$ SPECIAL-VALUES.
          CREATE VALUE BLOCK SPECIAL-VALUES
           1111111111111 $  V1 $
          .22222222222222 $  V2 $
          3333333.4444444 $  V3 $
          .55555555555 E-279 $  V4 $
          666666666666 E 308 $  V5 $

     VALUE BLOCK SPECIAL-VALUES
     HAS BEEN ADDED TO THE VALUE BLOCK FILE

     STATEMENT EXECUTION REQUIRED      .015 SECONDS FOR RDVALBLK
```

```
PRTVALBLK (SPECIAL-VALUES).

                    * * * * VALUE BLOCK TABLE * * * *
                            (SPECIAL-VALUES  )

      VARIABLE              VALUE                    VARIABLE          VALUE

      V1                    1.111111111111E+13       V2                2.222222222222E-01
      V3                    3.333333444444E+06       V4                5.555555555500-280
      V5                    6.666666666660+319

  STATEMENT EXECUTION REQUIRED      .006 SECONDS FOR PRTVALBLK
```

4.3.2 Print Value Block

The Print Value Block procedure is used to print value blocks stored in the value block file. Parameters in the call specify value blocks to be printed. For example, suppose the value block file contains value block VLBLK-1 which contains variable values:

$$A=1.0\times10^{-2}$$
$$B=1.0\times10^{-2}$$
$$C=2.6\times10^{-4}$$
$$D=1.0\times10^{-2}$$

Execution of procedure call statement

PRTVALBLK (VLBLK-1)

causes a table of the variable values in VLBLK-1 to be printed in the form:

* * * * VALUE BLOCK TABLE * * * *
(VLBLK-1)

VARIABLE	VALUE	VARIABLE	VALUE
A	1.000000000000E-02	B	1.000000000000E-02
D	1.000000000000E-02	C	2.600000000000E-04

Procedure Call

A call of the Print Value Block procedure has the form:

PRTVALBLK (vb_1,vb_2,...,vb_n).

Parameters vb_i, i=1,2,...,n, specify value blocks to be printed. There must be at least one value block to be printed, ($1 \leq i$).

Processing Achieved

Does not apply.

Printed Output

If the value block file does not contain a value block for parameter vb_i, a SETS user program error occurs (see APPENDIX A, Section 3.3, Numbered Errors, error 37).

If the value block file contains a value block for parameter vb_i, the variables from the value block are printed with their values. A title of the form:

* * * * VALUE BLOCK TABLE * * * *
(vb_i)

is printed first. Then the variables and their values are printed in two columns.

Example

Consider SETS user program:

```
PROGRAM$ PRTVALBLK-EX.
   DLTVALBLK.
   RDVALBLK (FAIL-PROB, REPAIR-COST).
   PRTVALBLK (FAIL-PROB, REPAIR-COST).
```

The first statement in the SETS user program is a DLTVALBLK call that initializes the value block file. Then, a call of RDVALBLK reads value blocks FAIL-PROB and REPAIR-COST and enters them into the value block file. These value blocks are used in the example for GENFTEQN (see Section 4.4.3, Generate Fault Tree Equation, Example). The printed output produced by the RDVALBLK call is as follows:

```
RDVALBLK (FAIL-PROB, REPAIR-COST).

          VALUE BLOCK$ FAIL-PROB.
          CREATE VALUE BLOCK FAIL-PROB
             .1 E-3$  E7, E11, E24$
             .1 E-2$  E8, E16$
             .5 E-2$  E20$
             .75 E-2$  E10, E17, E34$
             .1 E-1$  E19, E33$
             .25 E-1$  E15, E22, E28$
             .5 E-1$  E9, E12, E14, E21, E23, E26, E30, E32$
             .7 E-1$  E3, E5, E13, E29, E31$
             .75 E-1$  E2, E18$
             .8 E-1$  E1, E6, E27$
             .9 E-1$  E4, E25$

      VALUE BLOCK FAIL-PROB
      HAS BEEN ADDED TO THE VALUE BLOCK FILE



          VALUE BLOCK$ REPAIR-COST.
          CREATE VALUE BLOCK REPAIR-COST
             25$   E3, E9, E24, E32$
             50$   E1, E7, E11, E15, E21, E29$
             80$   E5, E8, E13, E19, E26, E31, E34$
            100$   E6, E14, E25, E28, E30, E33$
            140$   E2, E10, E23$
            210$   E4, E12, E17, E27$
            300$   E16, E18, E20, E22$

      VALUE BLOCK REPAIR-COST
      HAS BEEN ADDED TO THE VALUE BLOCK FILE

      STATEMENT EXECUTION REQUIRED      .029 SECONDS FOR RDVALBLK
```

The last statement in the SETS user program is a call of PRTVALBLK which prints the variables and their values from value blocks FAIL-PROB and REPAIR-COST. The printed output for this procedure is as follows:

```
PRTVALBLK (FAIL-PROB, REPAIR-COST).

                    * * * * VALUE BLOCK TABLE * * * *
                            (FAIL-PROB       )

     VARIABLE          VALUE                      VARIABLE          VALUE

     E7          1.000000000000E-04          E11         1.000000000000E-04
     E24         1.000000000000E-04          E8          1.000000000000E-03
     E16         1.000000000000E-03          E20         5.000000000000E-03
     E10         7.500000000000E-03          E17         7.500000000000E-03
     E34         7.500000000000E-03          E19         1.000000000000E-02
     E33         1.000000000000E-02          E15         2.500000000000E-02
     E22         2.500000000000E-02          E28         2.500000000000E-02
     E9          5.000000000000E-02          E12         5.000000000000E-02
     E14         5.000000000000E-02          E21         5.000000000000E-02
     E23         5.000000000000E-02          E26         5.000000000000E-02
     E30         5.000000000000E-02          E32         5.000000000000E-02
     E3          7.000000000000E-02          E5          7.000000000000E-02
     E13         7.000000000000E-02          E29         7.000000000000E-02
     E31         7.000000000000E-02          E2          7.500000000000E-02
     E18         7.500000000000E-02          E1          8.000000000000E-02
     E6          8.000000000000E-02          E27         8.000000000000E-02
     E4          9.000000000000E-02          E25         9.000000000000E-02

                    * * * * VALUE BLOCK TABLE * * * *
                            (REPAIR-COST     )

     VARIABLE          VALUE                      VARIABLE          VALUE

     E3          2.500000000000E+01          E9          2.500000000000E+01
     E24         2.500000000000E+01          E32         2.500000000000E+01
     E1          5.000000000000E+01          E7          5.000000000000E+01
     E11         5.000000000000E+01          E15         5.000000000000E+01
     E21         5.000000000000E+01          E29         5.000000000000E+01
     E5          8.000000000000E+01          E8          8.000000000000E+01
     E13         8.000000000000E+01          E19         8.000000000000E+01
     E26         8.000000000000E+01          E31         8.000000000000E+01
     E34         8.000000000000E+01          E6          1.000000000000E+02
     E14         1.000000000000E+02          E25         1.000000000000E+02
     E28         1.000000000000E+02          E30         1.000000000000E+02
     E33         1.000000000000E+02          E2          1.400000000000E+02
     E10         1.400000000000E+02          E23         1.400000000000E+02
     E4          2.100000000000E+02          E12         2.100000000000E+02
     E17         2.100000000000E+02          E27         2.100000000000E+02
     E16         3.000000000000E+02          E18         3.000000000000E+02
     E20         3.000000000000E+02          E22         3.000000000000E+02

STATEMENT EXECUTION REQUIRED       .019 SECONDS FOR PRTVALBLK
```

This page intentionally left blank.

4.3.3 Delete Value Block

The Delete Value Block procedure is used to delete value blocks from the value block file. For example, suppose the value block file contains value blocks:

VLBLK-1
INIT-COND

Execution of procedure call statement

DLTVALBLK (INIT-COND)

deletes INIT-COND from the value block file and leaves value block VLBLK-1 in the file.

Procedure Call

A call of the Delete Value Block procedure has one of the forms:

a. DLTVALBLK.

b. DLTVALBLK (vb_1,vb_2,...,vb_n).

Parameters vb_i, i=1,2,...,n, specify value blocks to be deleted from the value block file.

Processing Achieved

The specified value blocks are deleted from the value block file. If the parameter part of a DLTVALBLK call is empty (form a), every value block is deleted from the value block file; but if the parameter part is not empty (form b), only the value blocks specified in the call are deleted from the value block file. If the value block file does not contain a value block for a parameter specified in a DLTVALBLK call, the parameter has no effect and processing continues with the next parameter.

A call of DLTVALBLK without parameters must be used with care; it causes every value block in the value block file to be deleted.

Printed Output

A DLTVALBLK call produces only the standard printed output for a procedure call statement.

Example

Consider SETS user program:

```
PROGRAM$ DLTVALBLK-EX.
   PRTVALBLK (FAIL-PROB).
   DLTVALBLK.
   PRTVALBLK (FAIL-PROB).
```

The value block file contains value blocks FAIL-PROB and REPAIR-COST when execution of DLTVALBLK-EX begins. The first statement in DLTVALBLK-EX is a call of PRTVALBLK which shows value block FAIL-PROB is in the value block file. (PRTVALBLK is used to show the existence of value block FAIL-PROB because there is no status procedure for the value block file.) The output for this call of PRTVALBLK is as follows:

```
PRTVALBLK (FAIL-PROB).

                    * * * * VALUE BLOCK TABLE * * * *
                            (FAIL-PROB       )

        VARIABLE            VALUE                   VARIABLE            VALUE

          E7          1.0000000000000E-04           E11         1.0000000000000E-04
          E24         1.0000000000000E-04           E8          1.0000000000000E-03
          E16         1.0000000000000E-03           E20         5.0000000000000E-03
          E10         7.5000000000000E-03           E17         7.5000000000000E-03
          E34         7.5000000000000E-03           E19         1.0000000000000E-02
          E33         1.0000000000000E-02           E15         2.5000000000000E-02
          E22         2.5000000000000E-02           E28         2.5000000000000E-02
          E9          5.0000000000000E-02           E12         5.0000000000000E-02
          E14         5.0000000000000E-02           E21         5.0000000000000E-02
          E23         5.0000000000000E-02           E26         5.0000000000000E-02
          E30         5.0000000000000E-02           E32         5.0000000000000E-02
          E3          7.0000000000000E-02           E5          7.0000000000000E-02
          E13         7.0000000000000E-02           E29         7.0000000000000E-02
          E31         7.0000000000000E-02           E2          7.5000000000000E-02
          E18         7.5000000000000E-02           E1          8.0000000000000E-02
          E6          8.0000000000000E-02           E27         8.0000000000000E-02
          E4          9.0000000000000E-02           E25         9.0000000000000E-02

     STATEMENT EXECUTION REQUIRED       .009 SECONDS FOR PRTVALBLK
```

The next statement in the SETS user program is a call of DLTVALBLK without parameters which deletes every value block from the value block file. Then, another call of PRTVALBLK produces SETS user program error 37. Although this does not show the value block file is empty, it does show value block FAIL-PROB in not in the value block file. The printed output produced by these two procedure calls is as follows:

```
DLTVALBLK.

     STATEMENT EXECUTION REQUIRED       .006 SECONDS FOR DLTVALBLK


     PRTVALBLK (FAIL-PROB).

******ERROR NUMBER    37,   FAIL-PROB

     STATEMENT EXECUTION REQUIRED       .002 SECONDS FOR PRTVALBLK
```

4.4 Fault Tree Procedures

The fault tree procedures do not use or change the equation file or the value block file. Interactions between fault tree procedures and the block file are shown in Figure 4.8.

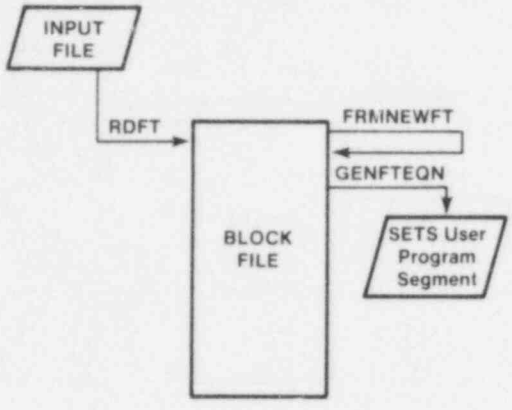


Figure 4.8 Interactions of fault tree procedures with the block file.

Two of the three fault tree procedures, RDFT and FRMNEWFT, change the block file by entering fault tree blocks into the file. Fault trees entered by RDFT are read from the input file. Fault trees entered by FRMNEWFT are created from fault trees in the block file. Fault tree blocks are deleted from the block file by DLTBLK (see Section 4.2.4, Delete Block).

Procedure GENFTEQN produces a SETS user program segment which contains calls of several of the other procedures. Only the block file is used to produce the segment. However, execution of the segment uses and changes the equation file and the block file. The value block file is never changed by execution of the segment, but it may be used by calls in the segment.

Fault trees for use with SETS are defined in Reference 27. These fault trees can have complement events, single-input intermediate events, and more than one top event. The latter two characteristics, single-input intermediate events and multiple top events, are important for the processing achieved by FRMNEWFT and GENFTEQN.

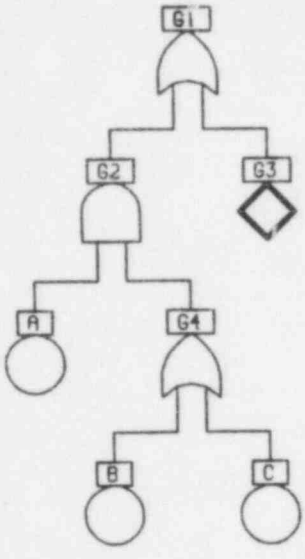
Fault trees used with SETS can also have similar trees. A similar tree is a way to model identical systems only once. Copies of the similar tree, with event names prefixed differently for each system, represent the identical systems in the fault tree. However, fault trees with similar trees cannot be processed by FRMNEWFT or GENFTEQN.

Fault trees in this manual are plotted, i.e., drawn, using the Fault Tree Drawing (FTD) program (Ref. 26). Fault trees are separated at multiple-output intermediate events for plotting. Multiple-output intermediate events are plotted as tops of separate trees, and transfer symbols (triangles) show where the events are used as inputs to other intermediate events. Transfer symbols are not used for multiple-output primary events. Instead, each multiple-output primary event is drawn everywhere it is used as an input to an intermediate event. An integer representing the number of uses of the primary event is printed in its symbol.

Fault trees can be plotted with descriptions that appear in boxes above primary and intermediate event symbols. In this manual, descriptions are not important, and example fault trees are plotted without descriptions to save space.

4.4.1 Read Fault Tree

The Read Fault Tree procedure is used to read fault trees from the input file and enter them into the block file. Parameters in the call specify the fault trees to be read. As each fault tree is read, it is checked for errors and, if there are no errors, it is tested for cycles. If there are no errors and no cycles, a fault tree block is created and added to the block file. For example, suppose FT-1 and its computer input representation are as shown in Figure 4.9.




```
FAULT TREE$ FT-1.
COMMENT$ INTERMEDIATE EVENTS$
   OG$ G1. IN$ G2, G3.
   AG$ G2. IN$ A, G4. OUT$ G1.
   OG$ G4. IN$ B, C. OUT$ G2.
COMMENT$ PRIMARY EVENTS$
   BE$ A.  OUT$ G2.
   BE$ B.  OUT$ G4.
   BE$ C.  OUT$ G4.
   DE$ G3. OUT$ G1.
```

Figure 4.9 Fault tree FT-1.

Execution of procedure call statement

```
RDFT (FT-1)
```

reads the input representation of FT-1. Since there are no errors in the definition of FT-1 and it does not contain cycles, fault tree block FT-1 is created and entered into the block file. The printed output for this call of RDFT includes printing the input representation of FT-1 as it is read, the message

```
THERE ARE NO CYCLES IN FT-1
```

which occurs after the fault tree has been tested for cycles, and the message

```
FAULT TREE BLOCK FT-1
HAS BEEN ADDED TO THE BLOCK FILE
```

which occurs after the fault tree block has been added to the block file.

Procedure Call

A call of the Read Fault Tree procedure has the form:

RDFT (ft_1,ft_2,...,ft_n).

Parameters, ft_i, i=1,2,...,n, specify the fault trees to be read.

Processing Achieved

A graphic and a computer input representation of a fault tree for use with SETS are defined in Reference 27. A summary description of these representations is included here.

A fault tree is comprised of primary and intermediate events and their inputs and outputs. Primary events are elementary; they are not defined as functions of other events in the fault tree. Intermediate events are logical combinations of primary events and other intermediate events in the fault tree. Gates specify the logical combinations of input events that define intermediate (output) events.

There are five different types of primary events that can be specified; but the basic event, developed event, and external event are the ones used most frequently. The graphic and input representations for these three primary events are as follows:

Basic Event	Developed Event	External Event
		
BE$	DE$	EE$

There are six different gate types that can be specified and three of them receive frequent use. The graphic and input representations for the AND gate, OR gate, and SPECIAL gate are as follows:

AND Gate	OR Gate	SPECIAL Gate
		
AG$	OG$	SG$

Each entity in the input representation of a fault tree is terminated by a delimiter such as a dollar sign ($), period (.), or comma (,). Entities terminated by dollar signs, such as BE, OG, IN and OUT, have a predetermined meaning. Fault tree and event names are terminated by commas or periods. Names are chosen by the user.

Names contain up to sixteen characters selected from the digits (0-9), the capital letters of the alphabet (A-Z), and the minus sign (-) used as a hyphen. Spaces do not have any meaning in the input representation of a fault tree; they can be used freely to enhance the appearance of the input.

The input representation of a fault tree begins with a fault tree header of the form

FAULT TREE$ ft.

where ft is the fault tree name.

Event definitions follow the fault tree header. They can occur in any order but there must be one, and only one, for each event in the fault tree.

A basic event definition has the form

BE$ pe. OUT$ ie_1,ie_2,...,ie_s.

Definitions for other types of primary events are similar. For a developed event or an external event, BE is replaced by DE or EE, respectively. Outputs, ie_k, k=1,2,...,s, must be all of the intermediate events in the fault tree that have event pe as an input. A primary event must have at least one output, ($1 \leq k$).

A gate defines an intermediate event as a function of its input events. An intermediate event definition for an event defined by an AND gate has the form

AG$ ie. IN$ f_1,f_2,...,f_m. OUT$ ie_1,ie_2,...,ie_s.

The definition for an intermediate event defined by an OR gate is similar but AG is replaced by OG.

The definition for an event defined by a SPECIAL gate has the form

SG$ ie=<logic expression>. IN$ f_1,f_2,...,f_m. OUT$ ie_1,ie_2,...,ie_s.

The logic expression defines intermediate event ie as a function of inputs f_j, j=1,2,...,m. The logic expression can be any legitimate Boolean expression, but it must contain each f_j and every variable in the expression must be an f_j. (A SPECIAL gate serves as a NOT gate when ie = /f_1.)

Intermediate event inputs, f_j, j=1,2,...,m, can be either primary or intermediate events. Outputs, ie_k, k=1,2,...,s, must be all of the intermediate events in the fault tree that have event ie as an input. An intermediate event must have at least one input, ($1 \leq j$). If an intermediate event does not have any outputs, it defines a top event of the fault tree and output declaration

OUT$ ie_1,ie_2,...,ie_s

does not occur in the event definition.

Parameters, ft_i, i=1,2,...,n, in a RDFT call are processed one at a time from left to right. As each ft_i is encountered, the next information in the input file is read and

processed. The fault tree header must occur first and parameter ft_i and the name in the header must be the same.

Each fault tree is thoroughly checked to verify that it is well defined. As each event definition is read and processed, it is checked for inconsistencies that can be determined from the definition alone. For example, if a primary event has no outputs or an intermediate event has no inputs, an error is detected. Errors detected during this phase of testing cause numbered SETS user program errors to occur (see APPENDIX A, Section 3.3, Numbered Errors).

After every event has been read and checked for errors, each event is checked to ensure that it is consistent with the events related to it by inputs and outputs. During this second phase of testing, special error messages (see APPENDIX A, Section 3.2, Special Fault Tree Errors) occur when inconsistencies between two events are detected.

A fault tree is checked for inconsistencies between an event and events related to it by inputs and outputs whether or not errors were detected in individual event definitions. If errors occur in individual events, the check for consistency between events cannot be complete, but it does provide as much information as possible about errors in the fault tree during a single RDF T call.

Once a fault tree can be read without error, it is tested to determine whether or not it contains cycles. (A cycle in a fault tree produces an unending sequence of substitutions as described in Section 4.1.4, Substitute in Equation, Processing Achieved.) If there are no cycles, a fault tree block is created and added to the block file.

Printed Output

As the input representation of fault tree ft_i is read, it is also printed and becomes part of the output produced by RDF T. After the fault tree is read and checked for errors, it is examined to determine whether or not it contains intermediate events which, by comparing only their gate types and inputs, can be determined to be equivalent. Intermediate events are equivalent if they are defined by the same type of gate, the gate type is not SPECIAL, and their inputs agree in number and name. Equivalent events are listed as a group and, if there is more than one group, each group is listed. For example, if fault tree ft_i has equivalent events, E_j, $j=1,2,\ldots,m$, they are listed after the input representation of the fault tree has been printed. The list has the form:

THE FOLLOWING EVENTS ARE EQUIVALENT

1. E_1
2. E_2
.
.
.
m. E_m

If a fault tree does not have equivalent events, no message is printed.

After the fault tree is examined for equivalent events, it is tested to determine if it contains cycles. If the fault tree contains cycles, a SETS user program error will occur (see APPENDIX A, Section 3.3, Numbered errors, error 48). If there are no cycles, the message

THERE ARE NO CYCLES IN ft_i

is printed. Finally, a fault tree block having the name ft_i is created and added to the block file, and the message

FAULT TREE BLOCK ft_i
HAS BEEN ADDED TO THE BLOCK FILE

is printed.

Example

Consider SETS user program:

```
PROGRAM$ RDFT-EX.
   DLTBLK.
   COMMENT$  G1 AND G2 ARE EQUIVALENT,
             G5, G8, AND G9 ARE EQUIVALENT.$
   RDFT (EQUIV-EVNTS).
   RDFT (FT-3, FT-4).
   PRTBLK( FT-3).
```

The first statement in the SETS user program is a DLTBLK call without parameters which initializes the block file. The second statement is a call of RDFT which reads fault tree EQUIV-EVNTS, shown in Figure 4.10.

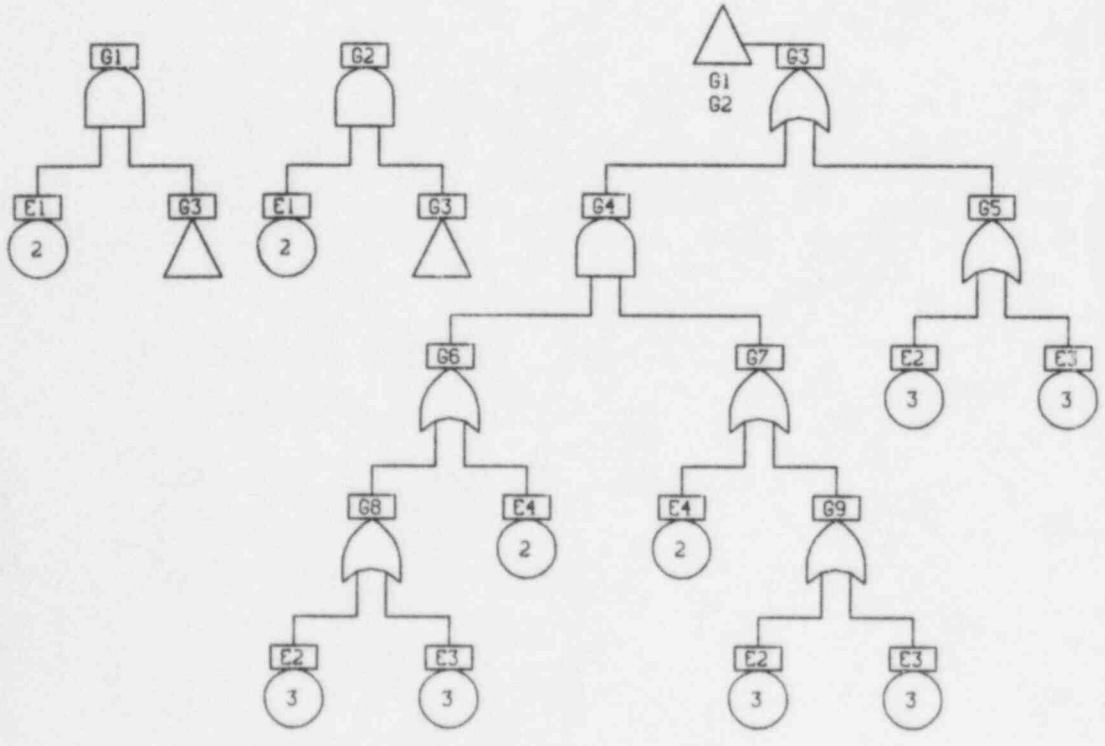


Figure 4.10 Fault tree EQUIV-EVNTS.

Fault tree EQUIV-EVNTS has two groups of events which, considering only gate types and inputs of the events, are determined to be equivalent: G1=G2 and G5=G8=G9. (Events G6 and G7 also happen to be equivalent but this cannot be ascertained from the gate types and inputs of G6 and G7 alone.) The printed output produced by this call of RDFT is as follows:

```
RDFT (EQUIV-EVNTS).

          FAULT TREE$ EQUIV-EVNTS.
          AG$ G1.  IN$ E1, G3.
          AG$ G2.  IN$ G3, E1.
          OG$ G3.  IN$ G4, G5.  OUT$ G1, G2.
          AG$ G4.  IN$ G6, G7.  OUT$ G3.
          OG$ G5.  IN$ E2, E3.  OUT$ G3.
          OG$ G6.  IN$ G8, E4.  OUT$ G4.
          OG$ G7.  IN$ E4, G9.  OUT$ G4.
          OG$ G8.  IN$ E2, E3.  OUT$ G6.
          OG$ G9.  IN$ E2, E3.  OUT$ G7.
          BE$ E1.  OUT$ G1, G2.
          BE$ E2.  OUT$ G5, G8, G9.
          BE$ E3.  OUT$ G5, G8, G9.
          BE$ E4.  OUT$ G6, G7.

          THE FOLLOWING EVENTS ARE EQUIVALENT

               1.  G1
               2.  G2

          THE FOLLOWING EVENTS ARE EQUIVALENT

               1.  G5
               2.  G8
               3.  G9

          THERE ARE NO CYCLES IN EQUIV-EVNTS

     FAULT TREE BLOCK EQUIV-EVNTS
     HAS BEEN ADDED TO THE BLOCK FILE

     STATEMENT EXECUTION REQUIRED      .031 SECONDS FOR RDFT
```

The third statement in SETS user program RDFT-EX is used to read two fault trees. Fault trees FT-3 and FT-4 are used here to illustrate the Read Fault Tree procedure and they are also used in the description of the Form New Fault Tree procedure (see Section 4.4.2, Form New Fault Tree). Fault trees FT-3 and FT-4 are shown in Figure 4.11.

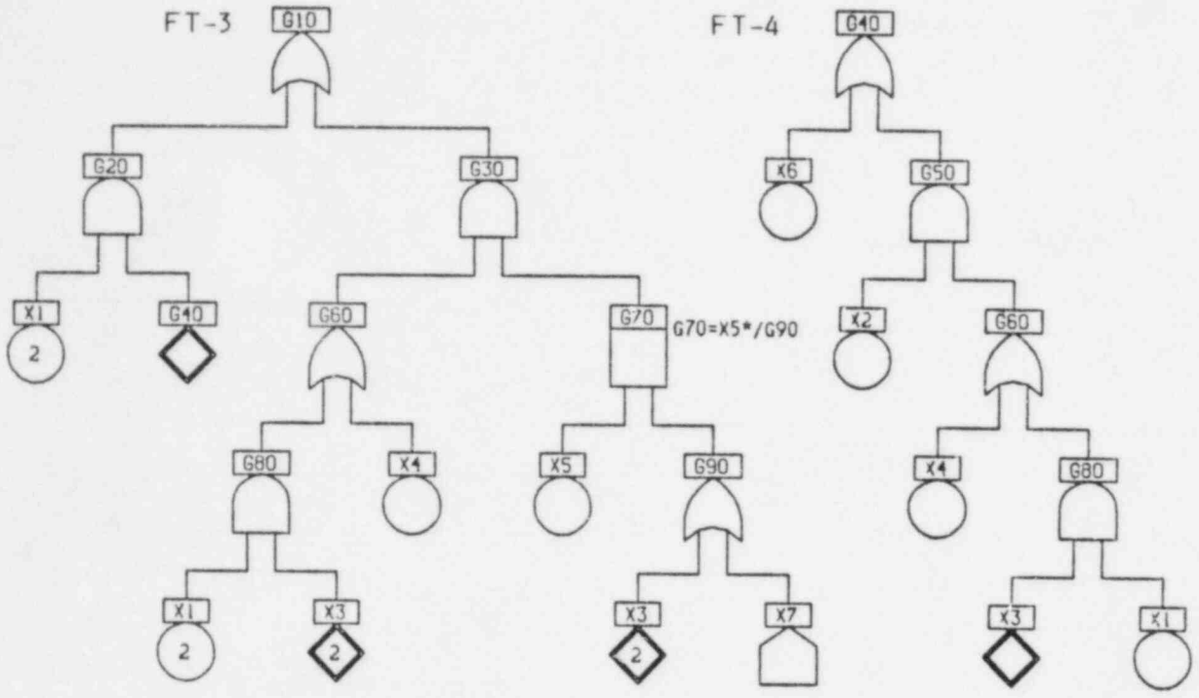


Figure 4.11 Fault trees FT-3 and FT-4.

The printed output produced by the RDFT call which reads FT-3 and FT-4 is as follows:

```
RDFT (FT-3, FT-4).

          FAULT TREE$ FT-3.
             OG$ G10.  IN$ G20, G30.
             AG$ G20.  IN$ X1, G40.  OUT$ G10.
             AG$ G30.  IN$ G60, G70.  OUT$ G10.
             OG$ G60.  IN$ G80, X4.  OUT$ G30.
             SG$ G70=X5*/G90.  IN$ G90, X5.  OUT$ G30.
             AG$ G80.  IN$ X1, X3.  OUT$ G60.
             OG$ G90.  IN$ X7, X3.  OUT$ G70.
             BE$  X1.  OUT$ G20, G80.
             DE$  X3.  OUT$ G80, G90.
             BE$  X4.  OUT$ G60.
             BE$  X5.  OUT$ G70.
             EE$  X7.  OUT$ G90.
             DE$ G40.  OUT$ G20.

          THERE ARE NO CYCLES IN FT-3

     FAULT TREE BLOCK FT-3
     HAS BEEN ADDED TO THE BLOCK FILE
```

```
        FAULT TREE$ FT-4.
        OG$ G40.  IN$ X6, G50.
        BE$  X6.  OUT$ G40.
        AG$ G50.  IN$ X2, G60.  OUT$ G40.
        BE$  X2.  OUT$ G50.
        OG$ G60.  IN$ X4, G80.  OUT$ G50.
        BE$  X4.  OUT$ G60.
        AG$ G80.  IN$ X3, X1.  OUT$ G60.
        DE$  X3.  OUT$ G80.
        BE$  X1.  OUT$ G80.

        THERE ARE NO CYCLES IN FT-4

    FAULT TREE BLOCK FT-4
    HAS BEEN ADDED TO THE BLOCK FILE

    STATEMENT EXECUTION REQUIRED      .060 SECONDS FOR RDFT
```

The final statement in the SETS user program is a call of PRTBLK which displays fault tree FT-3. A Fault Tree Event Table is printed first, and then the equations for the intermediate events are printed. The printed output produced by the PRTBLK call for FT-3 is as follows:

```
PRTBLK (FT-3).
```

* * * * FAULT TREE EVENT TABLE * * * *
(FT-3)

	EVENT NAME	TYPE	RANK	RELATIONSHIPS	PREFIX	GS	SO	PST	PT
2	G10	OG	2	IN , G20 IN , G30				X	
3	G20	AG	3	IN , X1 IN , G40 OUT, G10					
4	G30	AG	3	IN , G60 IN , G70 OUT, G10					
5	X1	BE	2	OUT, G20 OUT, G80					
6	G40	DE	1	OUT, G20					
7	G60	OG	3	IN , G80 IN , X4 OUT, G30					
8	G70	SG	3	IN , X5 IN , G90 OUT, G30					
9	G80	AG	3	IN , X1 IN , X3 OUT, G60					
10	X4	BE	1	OUT, G60					
11	X5	BE	1	OUT, G70					

12	G90	OG	3	IN , X3 IN , X7 OUT, G70
13	X3	DE	2	OUT, G80 OUT, G90
14	X7	EE	1	OUT, G90

* * * * BLOCK EQUATIONS * * * *
(FT-3)

G10 = G20 + G30

G20 = X1 * G40

G30 = G60 * G70

G60 = G80 + X4

G70 = X5 * /G90

G80 = X1 * X3

G90 = X3 + X7

STATEMENT EXECUTION REQUIRED .028 SECONDS FOR PRTBLK

This page intentionally left blank.

4.4.2 Form New Fault Tree

The Form New Fault Tree procedure is used to create one or two new fault trees from existing fault trees. Parameters in the procedure call specify fault trees in the block file that are merged into a single fault tree. The merged fault tree is then modified in accordance with options specified in the call. A parameter determines whether or not the merged, modified fault tree will be restructured, and whether one or two fault trees will be created. Each final fault tree is tested for cycles and, if there are no cycles, a fault tree block is created and entered into the block file. For example, suppose the block file contains fault trees FT-1 and FT-2 as shown in Figure 4.12.

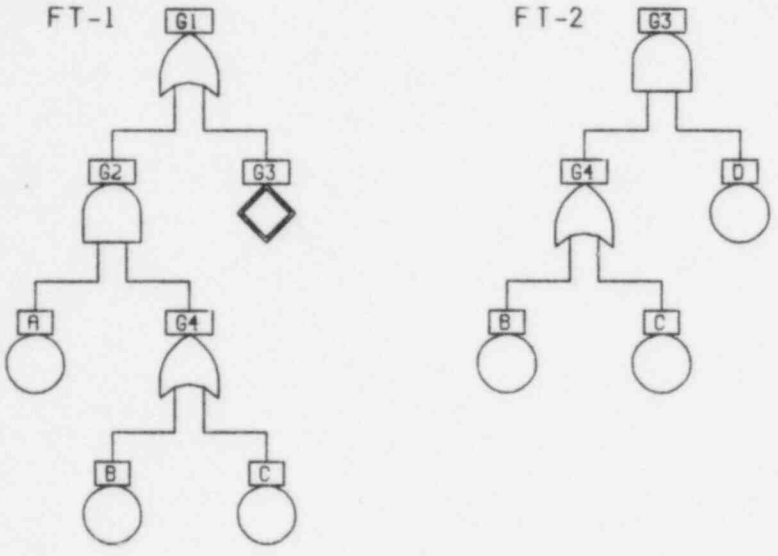


Figure 4.12 Fault trees FT-1 and FT-2.

Execution of procedure call statement

FRMNEWFT (FORM1$ FT-1, FT-2/ NEW-FT)

merges fault trees FT-1 and FT-2 into a single fault tree. Developed event G3 in FT-1 is replaced by fault tree FT-2 because the top event of FT-2 is G3 and, by implication, FT-2 is the development of event G3. Also, event G4, which has the same definition in FT-1 and FT-2, occurs in the merged fault tree with two outputs, G2 and G3. The merged fault tree is not modified because there are no options specified in the call. In accordance with parameter FORM1, the merged fault tree, without restructuring, is the single fault tree to be created. The merged fault tree is tested for cycles and, since it does not contain cycles, fault tree block NEW-FT is created and added to the block file. Fault tree NEW-FT is shown in Figure 4.13.

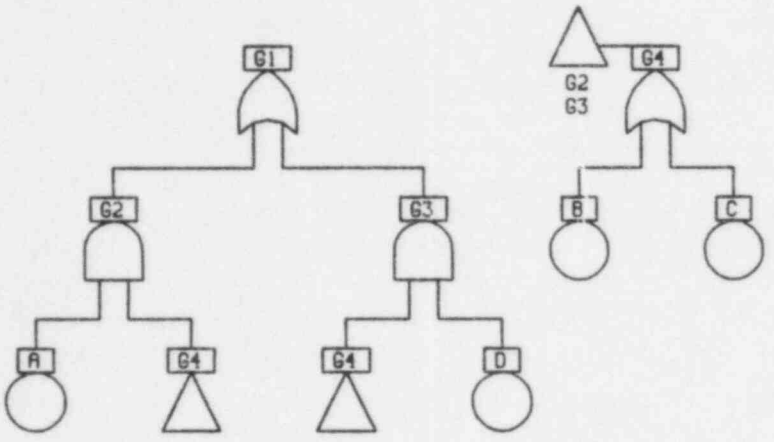


Figure 4.13 Fault tree NEW-FT.

The printed output produced by this call of FRMNEWFT includes the following information:

FAULT TREES MERGED

1. FT-1
2. FT-2

THERE ARE NO DEVELOPED EVENTS IN NEW-FT

THERE ARE NO CYCLES IN NEW-FT

TOP EVENTS OF NEW-FT

1. G1

FAULT TREE BLOCK NEW-FT
HAS BEEN ADDED TO THE BLOCK FILE

Procedure Call

A call of the Form New Fault Tree procedure has one of the forms:

a. FRMNEWFT (FORM1$ ft_1,ft_2,...,ft_n/ ft).

b. FRMNEWFT (FORM2$ ft_1,ft_2,...,ft_n/ ft).

c. FRMNEWFT (FORM3$ ft_1,ft_2,...,ft_n/ ft_S, ft_I).

d. FRMNEWFT (FORM3$ (<prefix>) ft_1,ft_2,...,ft_n/ ft_S, ft_I).

e. FRMNEWFT (<any a through d parameter form>* $O_1/O_2/.../O_m$).

Parameters, ft_i, i=1,2,...,n, specify the fault trees to be merged. There must be at least one fault tree to be merged, ($1 \leq n$).

Parameters FORM1 and FORM2 indicate one new fault tree will be created. For FORM1, the new fault tree is the merged fault tree modified by options specified in the call. For FORM2, the merged, modified fault tree is restructured to form the new fault tree. Single-input events defined by AND, OR, or EXACTLY ONE gates are removed, and single-output events defined by AND or OR gates are coalesced with their output events if they too are defined by AND or OR gates, respectively. The inputs of a coalesced event are the union of the inputs of the events coalesced.

Parameter FORM3 indicates two new fault trees will be created. The merged, modified fault tree is restructured and then separated into its stem and a collection of independent subtrees. The restructuring for FORM3 includes the restructuring described for FORM2. In addition, two or more multiple-output events having the same outputs are combined into one event if the multiple-output events and all of their outputs are defined by AND gates or they are all defined by OR gates. As before, the inputs of a combined event are the union of the inputs of the events combined. Before the restructured fault tree is separated, independent subtrees that produce an equivalent fault tree with a smaller stem are also created. Then, the fault tree is separated into its stem and a collection of independent subtrees to form the two fault trees. If a prefix occurs with parameter FORM3 (form d), it replaces the standard prefix used to generate new event names required when creating independent subtrees.

Parameter ft (forms a and b) is the name of the new fault tree if a single fault tree is created. Parameters ft_S and ft_I (forms c and d) are the names of the two new fault trees if a stem and collection of independent subtrees are created.

Parameters, O_j, j=1,2,...,m, represent options for modifying the merged fault tree.

Processing Achieved

Processing achieved by the Form New Fault Tree procedure involves merging fault trees, restructuring a fault tree, and creating independent subtrees. A general description of these techniques is presented first, and then their use in a call of FRMNEWFT is described.

MERGING FAULT TREES

Two fault trees are merged into a single fault tree by merging the events in the two fault trees. Two cases arise:

1. An event in one fault tree has a name different from every event name in the other fault tree.

2. An event in one fault tree has the same name as an event in the other fault tree.

For case 1, the merged fault tree contains an event with the different name, and it will have the same definition in the merged fault tree as it has in the original fault tree in which it occurs.

For case 2, two events having the same name are combined into a single event in the merged fault tree. Combining two events can produce an event with duplicate outputs except that, whenever duplicates occur, only one occurrence of the event is retained, i.e., only distinct outputs are retained. Two events with the same name are combined if they satisfy one of the following conditions:

a. Both events are primary events of the same type. The merged fault tree contains a primary event having the same name and type as the original events, and its outputs are the distinct outputs from both original events.

b. Both events are intermediate events having the same gate type and inputs, and the gate type is not SPECIAL. The merged fault tree contains an intermediate event having the same name, gate type, and inputs as the original events. Its outputs are all distinct outputs from both original events.

c. One event is a developed event and the other event is an intermediate event. The merged fault tree contains an intermediate event having the same name as the original events and the same gate type and inputs as the original intermediate event. Its outputs are all distinct outputs from both original events.

Two intermediate events defined by SPECIAL gates cannot be merged (see case 2b). However, they can be merged if one of them is changed to a developed event and the other one remains unchanged (see case 2c).

If two events have the same name, but do not satisfy 2a, 2b, or 2c, the events and, therefore, the fault trees cannot be merged. A SETS user program error will occur if two events have the same name but cannot be merged (see APPENDIX A, Section 3.3, Numbered Errors, error 55).

Fault trees FT-3 and FT-4, shown first in Figure 4.11 and here in Figure 4.14, can be merged into a single fault tree.

Events X1, X3 and X4 are examples of Case 2a. Each of these events is a primary event in the merged fault tree. Event X3 is a developed event with outputs G80 and G90. Events X1 and X4 are basic events; X1 has outputs G20 and G80, and X4 has output G60.

Events G60 and G80 are examples of Case 2b. The merged fault tree contains G60 defined by an OR gate with inputs X4 and G80 and outputs G30 and G50. The merged fault tree also contains G80 defined by an AND gate with inputs X1 and X3 and output G60.

Event G40 is an example of Case 2c. The merged fault tree contains G40 defined by an OR gate with inputs X6 and G50 and output G20. The merged fault tree does not contain developed event G40.

The remaining events in FT-3 and FT-4 are examples of Case 1. The merged fault tree contains each of them defined the same as they are defined in either FT-3 or FT-4. The merged fault tree is shown in Figure 4.15.

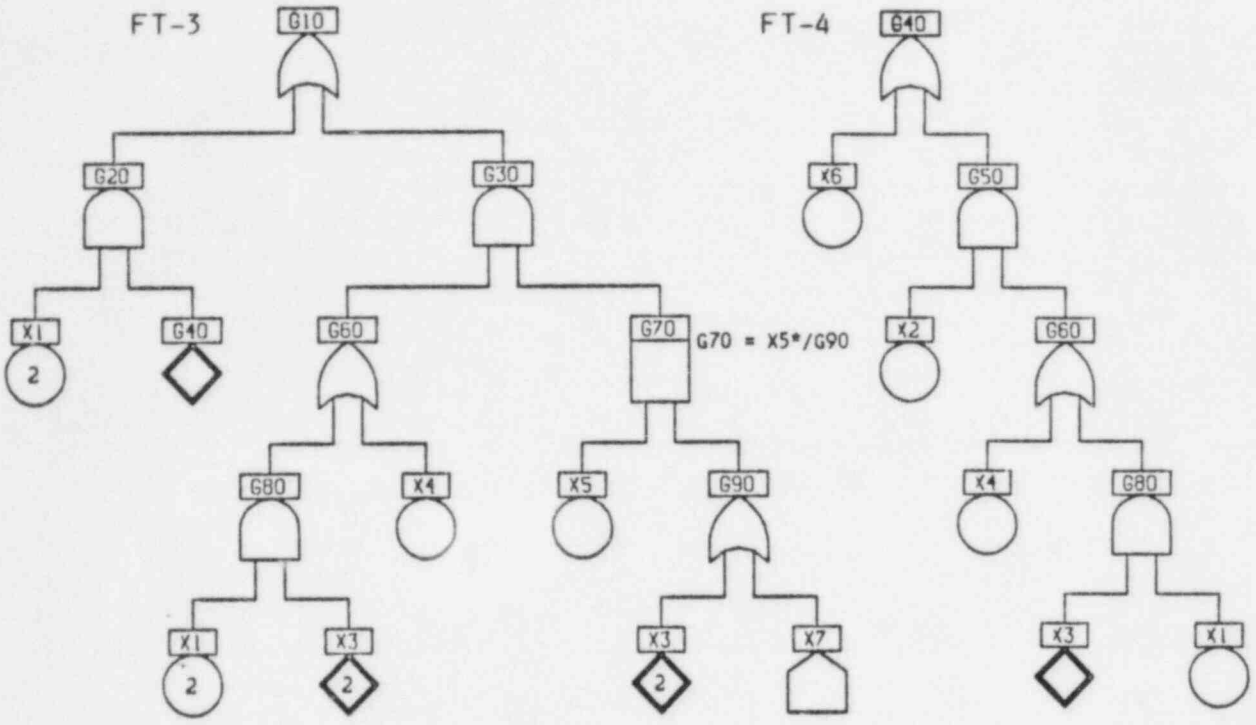


Figure 4.14 Fault trees FT-3 and FT-4.

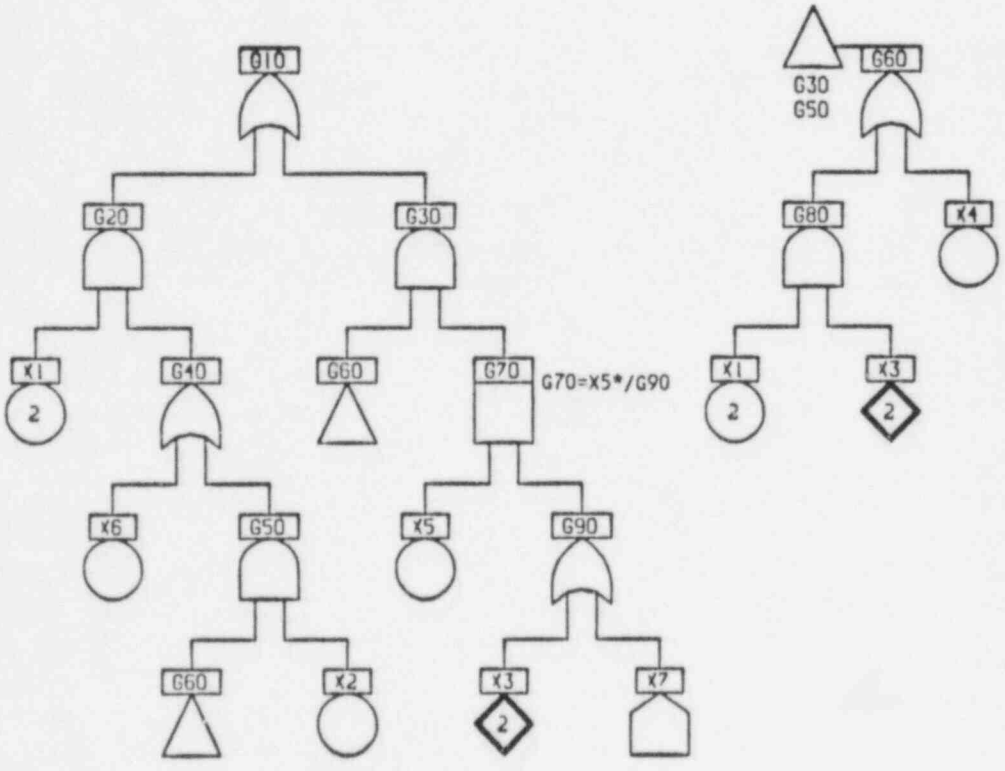


Figure 4.15 Fault tree FT-3-AND-4.

RESTRUCTURING A FAULT TREE

Restructuring a fault tree causes unnecessary events to be removed from the fault tree. However, restructuring is most important because it can lead to the creation of independent subtrees which cannot be formed unless the fault tree is restructured. Restructuring produces a fault tree that is different from the original fault tree, but one which has the same top events as the original fault tree. Furthermore, the minimal cut set equations for the top events of the restructured fault tree are equivalent to the minimal cut set equations for the corresponding top events of the original fault tree.

There are three ways a fault tree is restructured:

1. Single-input intermediate events which have at least one output, i.e., they are not top events, and are defined by AND, OR, or EXACTLY ONE gates are removed from the fault tree.

2. Single-output intermediate events defined by AND or OR gates are coalesced with their output events if the output events are also defined by AND or OR gates, respectively.

3. Two or more multiple-output intermediate events having the same outputs are combined into one event if the multiple-output events and their outputs are all defined by AND gates or they are all defined by OR gates.

Each of these three ways of restructuring a fault tree causes events to be removed from the fault tree. An event is removed by deleting the event and changing the inputs and outputs of the events related to it. These changes can cause repeated inputs or outputs in the remaining events except that, whenever this happens, only one occurrence of the repeated event is retained.

A generalized intermediate event, Y, having outputs Xi, i=1,2,...,N, and inputs Zk, k=1,2,...,S, is used to illustrate the three ways a fault tree is restructured. In the general representation, all of the inputs and outputs of Y are shown. For each Xi and Zk, only the input and output relationships with Y are shown; these events can have additional inputs and outputs consistent with their definitions. Also, every event is depicted as an intermediate event defined by an OR gate. Events Y and Xi, i=1,2,...,N, are always intermediate events, but they can be defined by some gate types other than the OR gate. The gate types allowed for these events are specified in the descriptions of event configurations that can be restructured. Events Zk, k=1,2,...,S, are not restricted; they can be any primary event or an intermediate event defined by any gate type. Generalized event Y is shown in Figure 4.16. (A specific example showing the three ways of restructuring a fault tree is presented at the end of the section entitled "CREATING INDEPENDENT SUBTREES" so the creation of independent subtrees after restructuring can also be shown.)

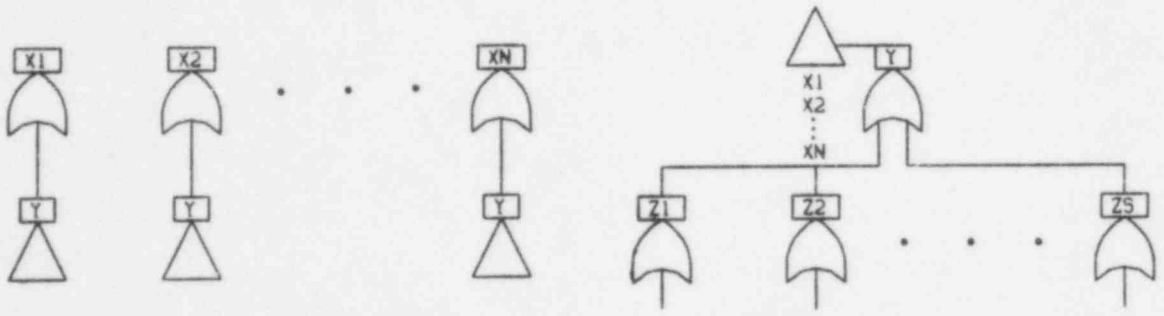


Figure 4.16 Generalized intermediate event Y.

Single-Input Events that can be Removed

If event Y has a single input (S=1), at least one output ($1 \leq N$), and is defined by an AND, OR, or EXACTLY ONE gate, Y can be removed from the fault tree. Each Xi can be defined by any type of gate. Single-input event Y, which can be removed from the fault tree, is shown in Figure 4.17.

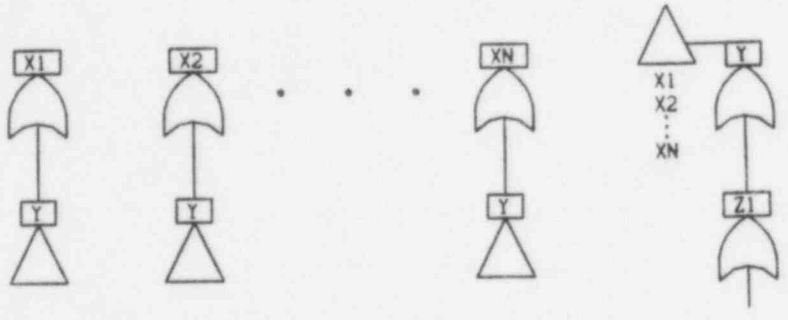


Figure 4.17 Single-input event Y which can be removed.

Changes to Remove Single-Input Event Y

1. Delete Y from the fault tree.
2. For each Xi, i=1,2,...,N, replace input Y by Z1.
3. For Z1, replace output Y by Xi, i=1,2,...,N.

The fault tree representation after single-input event Y has been removed is shown in Figure 4.18.

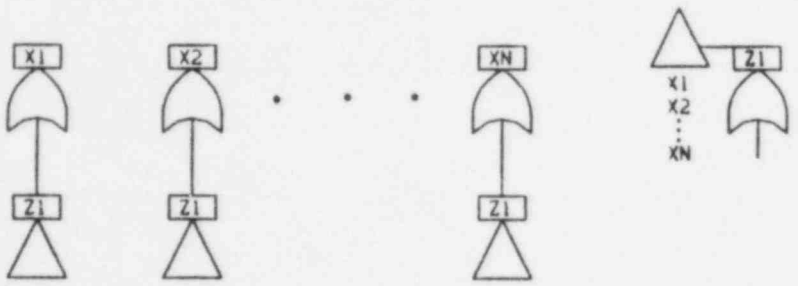


Figure 4.18 After removal of single-input event Y.

Events that can be Coalesced

If event Y has a single output (N=1), and Y and its output event, X1, are both defined by AND gates or they are both defined by OR gates, Y can be coalesced with X1. (Event Y could be coalesced with each of its outputs if it has more than one; but coalescing is not done when $1 < N$ because it can produce undesirable results.) Single-output event Y, which can be coalesced with X1, is shown in Figure 4.19.

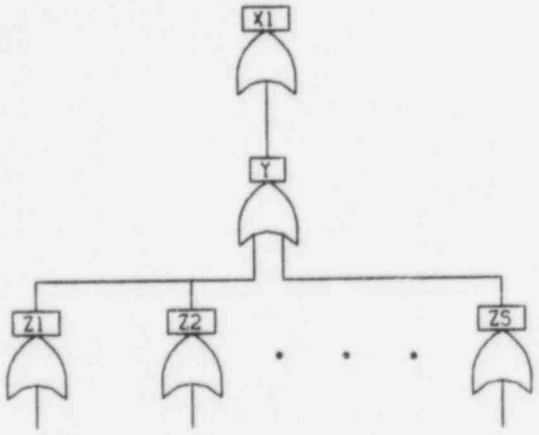


Figure 4.19 Event Y which can be coalesced with X1.

Changes to Coalesce Y with X1

1. Delete Y from the fault tree.
2. For X1, replace input Y by Zk, k=1,2,...,S.
3. For each Zk, k=1,2,...,S, replace output Y by X1.

The fault tree representation after Y has been coalesced with X1 is shown in Figure 4.20.

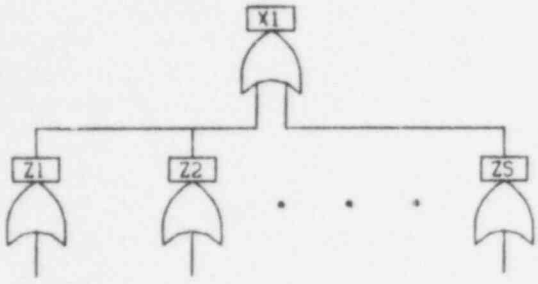


Figure 4.20 After coalescing Y with X1.

Events that can be Combined

Consider events Yj, j=1,2,...,M, which have the same outputs, Xi, i=1,2,...,N, and inputs Z1-k1, k1=1,2,...,S1; Z2-k2, k2=1,2,...,S2; . . .; and ZM-kM, kM=1,2,...,SM, respectively. If each Yj has two or more outputs ($2 \leq N$) and the Yj and Xi events are all defined by AND gates or they are all defined by OR gates, all but one of the Yj events can be combined with the remaining event. (If N=1, each Yj can be coalesced with its output event.) Multiple-output events Yj, j=2,3,...,M, which can be combined with Y1, are shown in Figure 4.21.

Changes to Combine Events Yj, j=2,3,...,M, with Y1

For j=2,3,...,M, do the following steps:

1. Delete Yj from the fault tree.
2. For each Xi, i=1,2,...,N, remove input Yj.
3. For each Zj-kj, kj=1,2,...,Sj, replace output Yj by Y1.
4. Add inputs Zj-kj, kj=1,2,...,Sj, to Y1.

The fault tree representation after events Yj, j=2,3,...,M, have been combined with Y1 is shown in Figure 4.22.

With one exception, each way of restructuring a fault tree can create a single-input event that can be removed, a single-output event that can be coalesced with its output, or a multiple-output event that can be combined with other multiple-output events. The exception is that coalescing an event with its output cannot create a single-input event that can be removed.

Restructuring a fault tree begins with the removal of single-input events -- including those created as others are removed. Then, single-output events are coalesced with their output events -- including those created as others are coalesced. At this stage in the process, the fault tree does not contain any single-input events that can be removed or single-output events that can be coalesced. If only these two ways of restructuring a fault tree are used, the processing is complete.

The third way of restructuring a fault tree occurs after events have been coalesced. Multiple-output events having the same outputs are combined into one event -- including those created as others are combined. However, combining multiple-output events can create both single-input events that can be removed and single-output events that can be coalesced. If all three ways of restructuring are used, they must be repeated until an iteration occurs during which the fault tree does not contain any multiple-output events that can be combined.

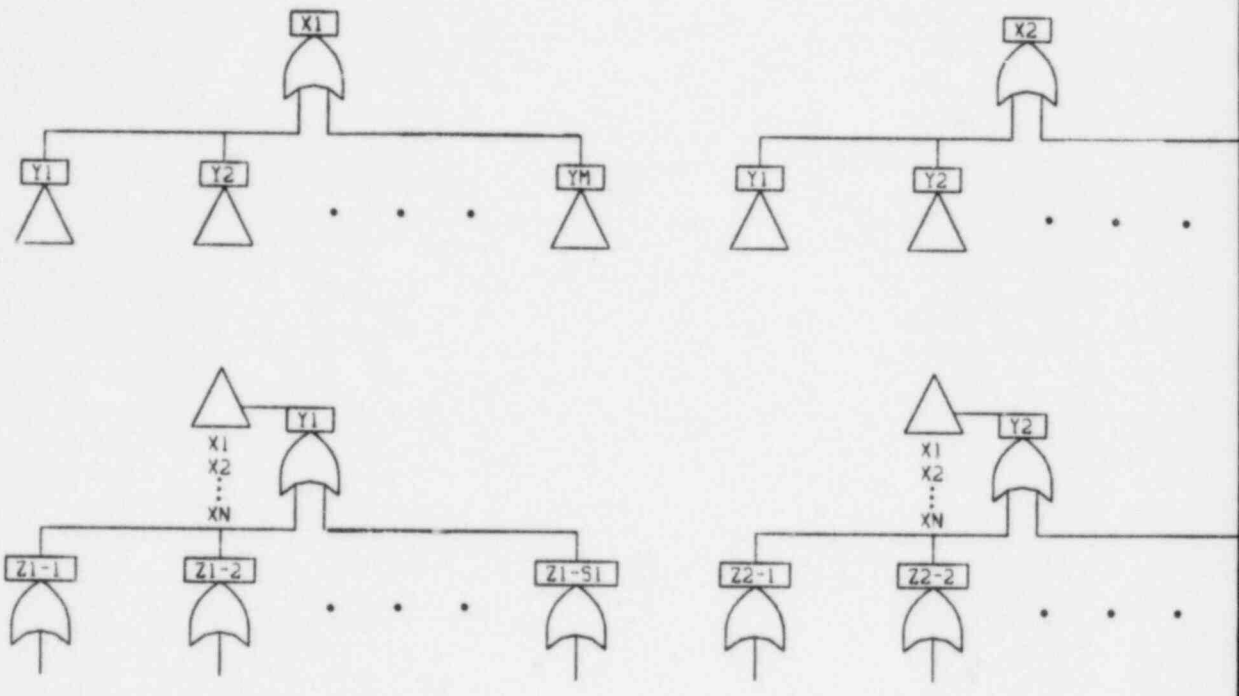


Figure 4.21 Events Yj,

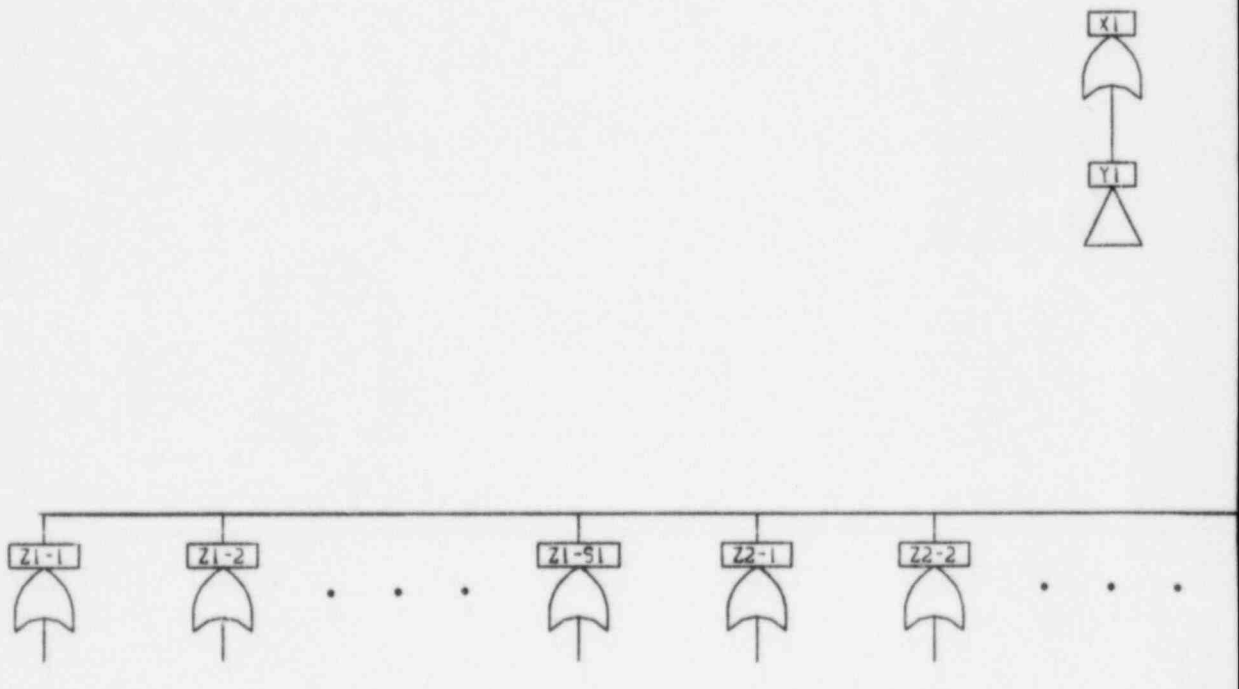
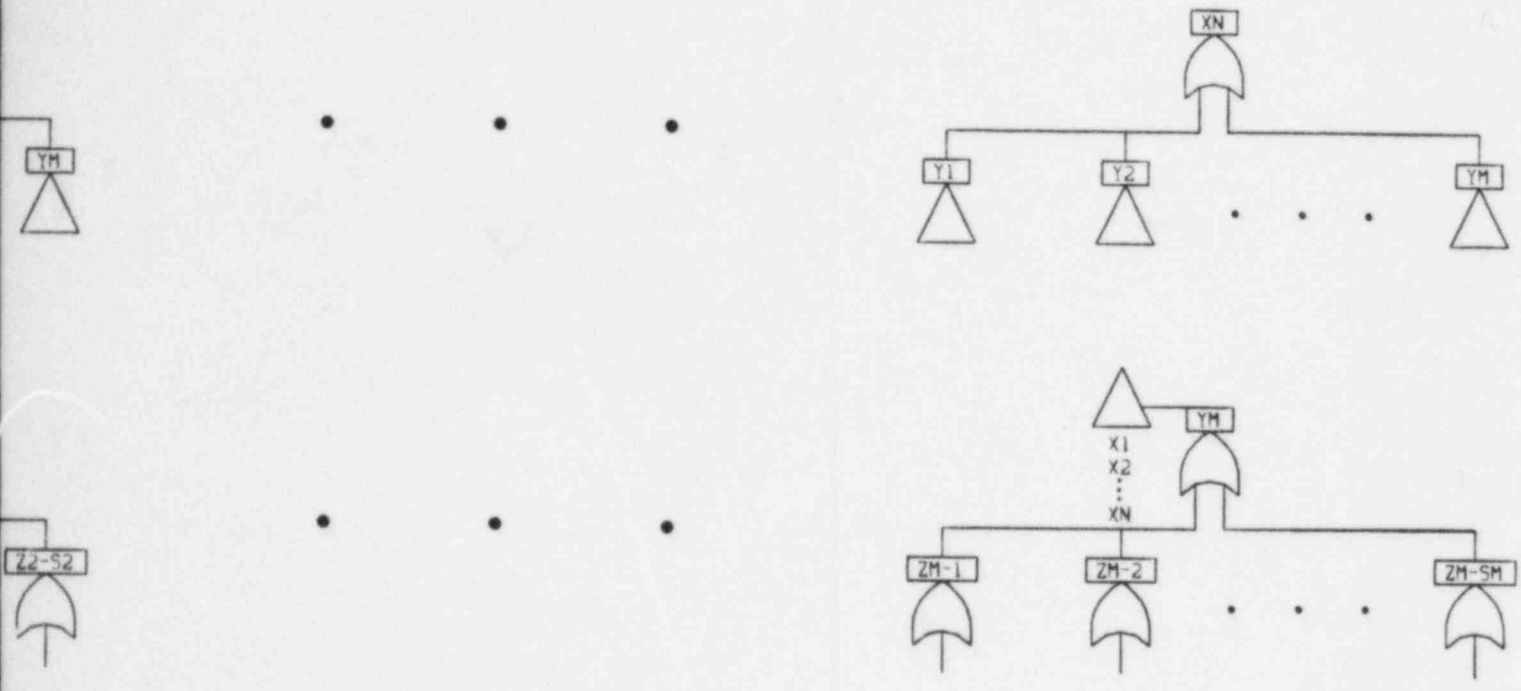
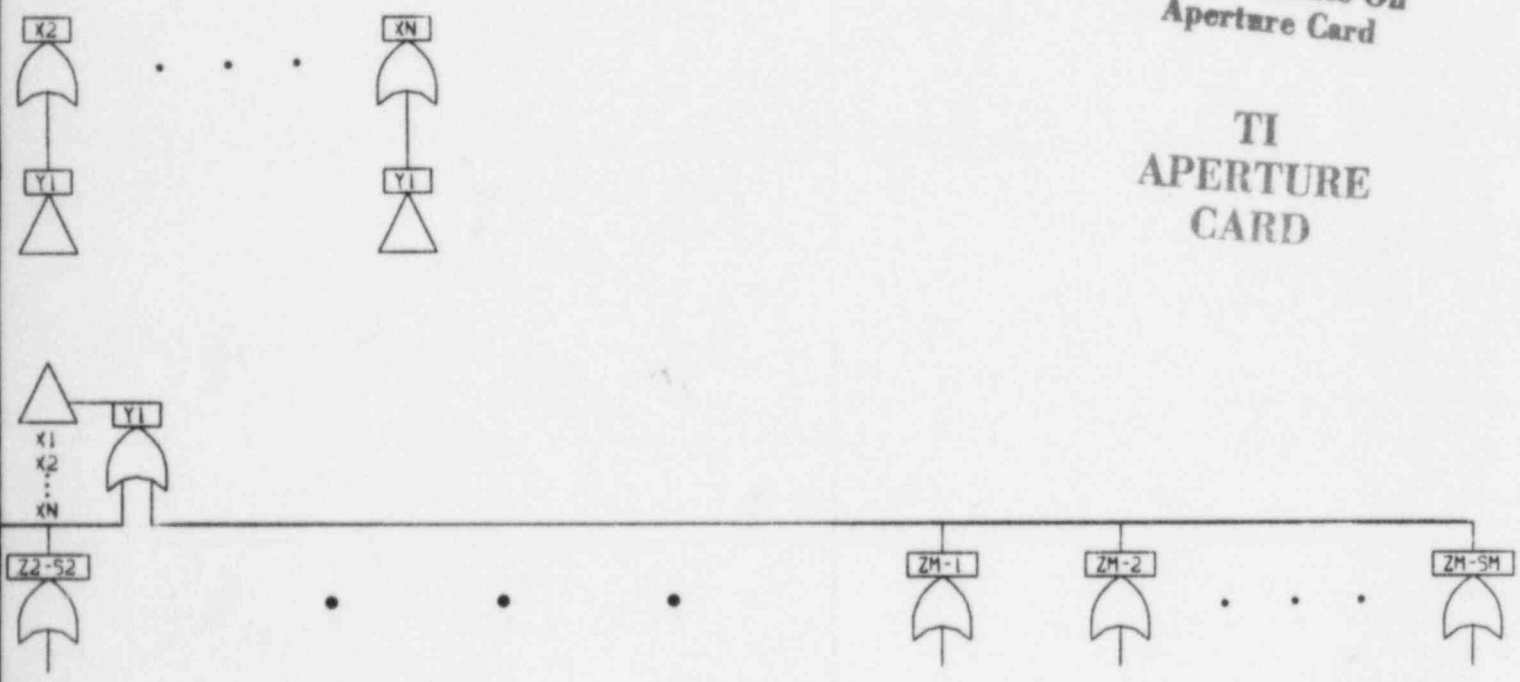


Figure 4.22 After c



j=2,3,...,M, which can be combined with Y1.



ombining events Yj, j=2,3,...,M, with Y1.

CREATING INDEPENDENT SUBTREES

A fault tree is comprised of primary and intermediate events and the inputs and outputs that relate them to each other. From a modeling perspective, a fault tree is comprised of the events, inputs, and outputs used to develop its top events. A subtree is a subset of a fault tree, i.e., a fault tree all of whose events, inputs and outputs belong to the given fault tree. Any set of intermediate events, ie_i, i=1,2,...,n, of a fault tree determines a subtree, $S(ie_1,ie_2,...,ie_n)$. The subtree is comprised of the events, inputs, and outputs used to develop the specified intermediate events. Top events of the subtree are a subset of intermediate events ie_i, i=1,2,...,n. For example, S(G2,G3), circumscribed by a dashed line in Figure 4.23, is a subtree of fault tree IST-FT and has top events G2 and G3. Subtree S(G4,G7,G9), also circumscribed by a dashed line, has top events G4 and G7. (Events G4 and G7 determine the same subtree as events G4, G7, and G9.)

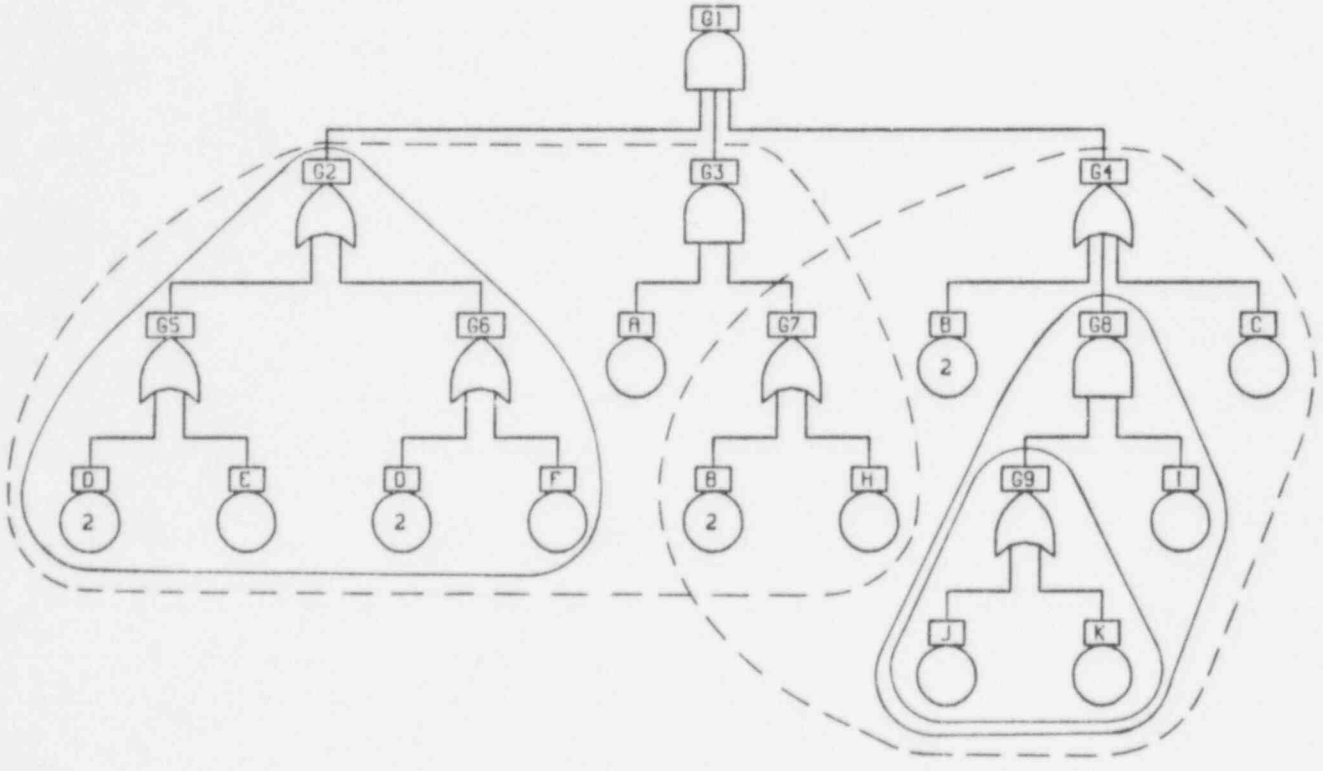


Figure 4.23 Fault tree IST-FT.

A subtree is proper if it is not the entire fault tree. A subtree is independent, meaning it is independent of the rest of the fault tree, if none of the primary events in the subtree are used to develop intermediate events in the rest of the fault tree. In Figure 4.23, S(G2,G3) is a proper subtree of IST-FT; but it is not an independent subtree because one of its primary events, B, is used to develop events G1 and G4 in the remainder of the fault tree. Subtree S(G4,G7,G9) is both independent and proper.

A single-top subtree is a subtree that has one top event. A largest single-top subtree is a single-top subtree that is not contained in another single-top subtree. Largest, single-top, independent, proper subtrees, hereafter referred to as LSIP subtrees, are of special interest.

Separating a fault tree into its stem and collection of LSIP subtrees is part of an efficient technique for finding minimal cut sets for top events of the fault tree (Ref. 2). The stem of a fault tree is the fault tree that remains after every LSIP subtree has been removed and replaced by a developed event having the same name as the top event of the removed subtree. The collection of LSIP subtrees of a fault tree is the fault tree determined by the top events of the LSIP subtrees. The collection of LSIP subtrees fault tree is a collection of disjoint subtrees, i.e., pairwise, they have no event in common. For example, S(G2) and S(G8) are the LSIP subtrees of IST-FT. (Subtree S(G9) is single-top, independent, and proper, but it is not largest since it is contained in S(G8). Thus, S(G2,G8) is the collection of LSIP subtrees of IST-FT. The stem of IST-FT is shown in Figure 4.24.

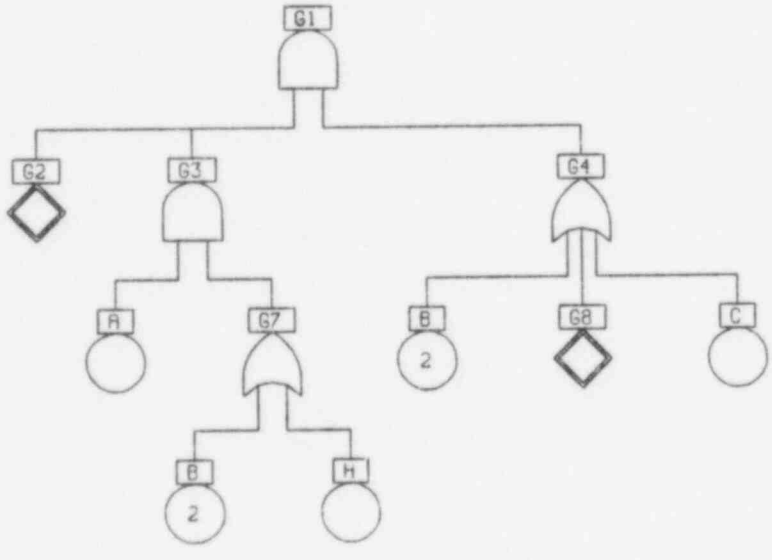


Figure 4.24 Stem of fault tree IST-FT.

The efficiency of the technique for finding minimal cut set equations by separating a fault tree into its stem and collection of LSIP subtrees can be improved by decreasing the size of the stem. This corresponds to creating a new, but equivalent, fault tree with a smaller stem and more or bigger LSIP subtrees. A new event, which changes the fault tree and is the top event of a new LSIP subtree can be created from event X if two conditions are satisfied:

1. Event X is defined by an AND or OR gate.

2. At least two, but not all, of the inputs of X are either single-output primary events or single-output top events of LSIP subtrees.

Event X is changed and new event IST-X is created as follows:

1. Remove inputs of X that are single-output primary events or single-output top events of LSIP subtrees.

2. Add event IST-X as an input of X.

3. Define event IST-X. Event IST-X has the same gate type as X. Its inputs are the inputs removed from X and its output is X.

Changing event X and creating event IST-X according to the above rules does not alter the minimal cut set equations for top events of the fault tree. The equation for original event X is equivalent to the equation produced by substitutions beginning with changed event X and carried through created event IST-X.

For example, event G4 in Figure 4.23 is defined by an OR gate and has inputs B, G8, and C. Event G8 is the top event of LSIP subtree S(G8), and G8 and primary event C each have one output. Thus, a new event, IST-G4, can be created. It is defined by an OR gate, and it has inputs G8 and C and output G4. Creating IST-G4 produces a fault tree slightly different from IST-FT but equivalent to it. LSIP subtree S(IST-G4) in the new fault tree is larger than LSIP subtree S(G8) in IST-FT. Equation

$$G4 = B + G8 + C$$

for event G4 before it is changed is equivalent to equation

$$G4 = B + IST\ \ G4 = B + (G8 + C)$$

for changed event G4 with substitution through created event IST-G4. Subtree S(G4) with created OR gate IST-G4 is shown in Figure 4.25.

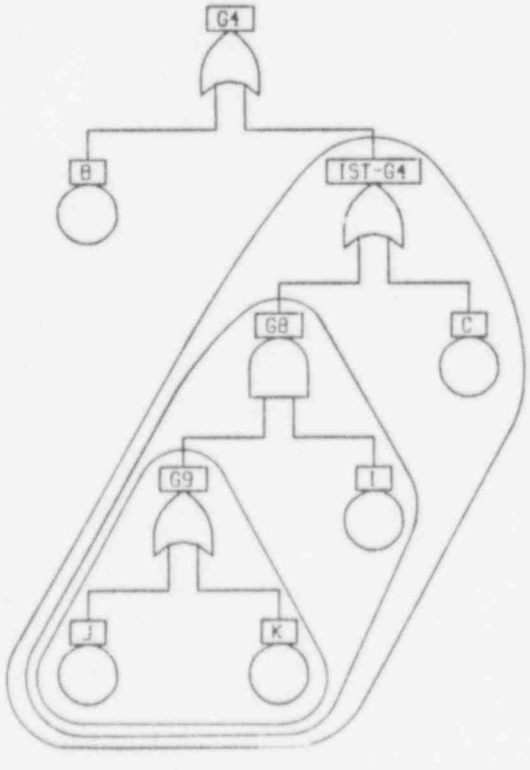


Figure 4.25 Subtree S(G4) with created event IST-G4.

Creating a new LSIP subtree produces a new fault tree which is slightly different from the original fault tree. If the inputs of the top event of the created subtree are all primary events, the new fault tree has one more LSIP subtree than the original fault tree. If there are any LSIP subtrees of the original fault tree among the inputs of the top event of the created subtree, they are contained in the larger created subtree. In the latter case, the new fault tree has no more LSIP subtrees than the original fault tree, and it can have fewer. However, in both cases, the creation of an additional or larger LSIP subtree makes the stem of the new fault tree smaller than the stem of the original fault tree.

Restructuring a fault tree can result in the creation of LSIP subtrees that cannot be formed without restructuring the fault tree. Consider fault tree CREATE-IND-ST in Figure 4.26. Event G4 is a single-input event that can be removed; event G5 can be coalesced with its output event G3; and one of the multiple-output events G6 or G7 can be combined with the other one. Fault tree CREATE-IND-ST contains only one LSIP subtree, S(G4), and no additional or larger LSIP subtrees can be created in its stem. However, removing G4, coalescing G5 with G3, and combining G7 with G6 leads to the creation of LSIP subtrees S(IST-1), S(IST-2), and S(IST-3), respectively. The fault tree with the created LSIP subtrees is shown in Figure 4.27.

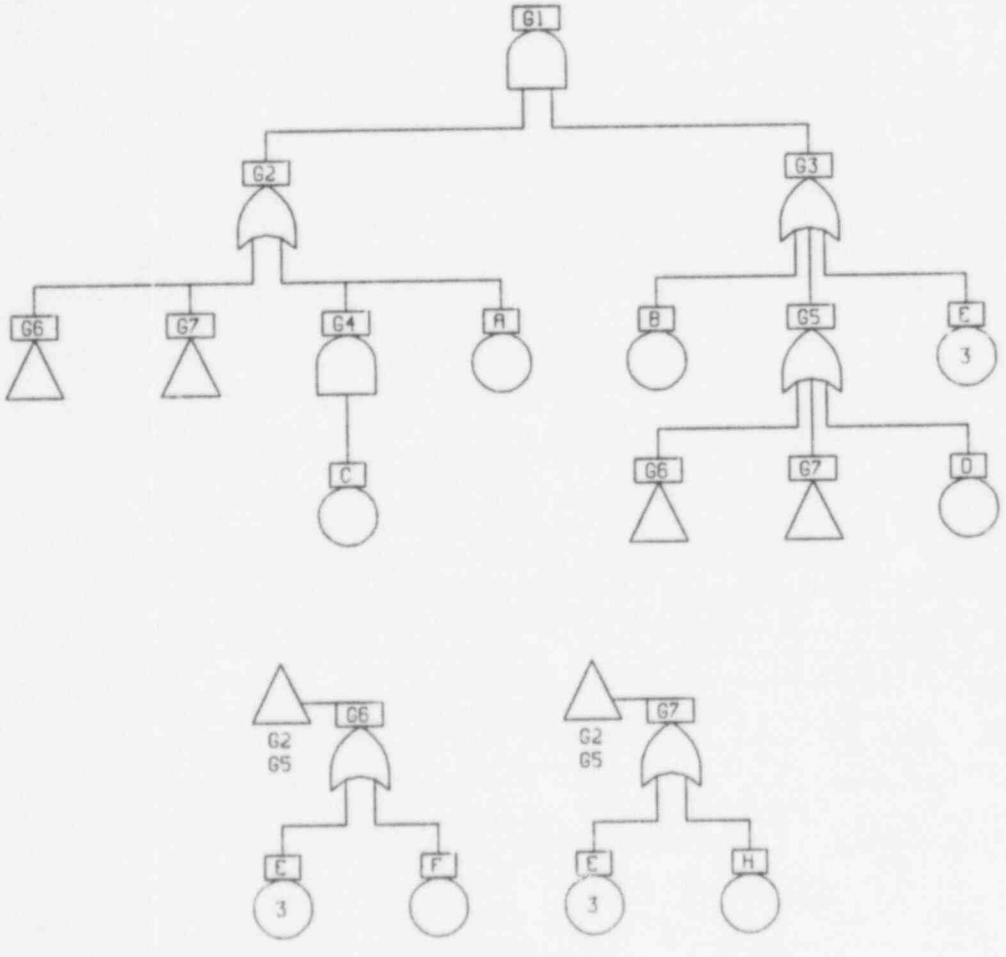


Figure 4.26 Fault tree CREATE-IND-ST.

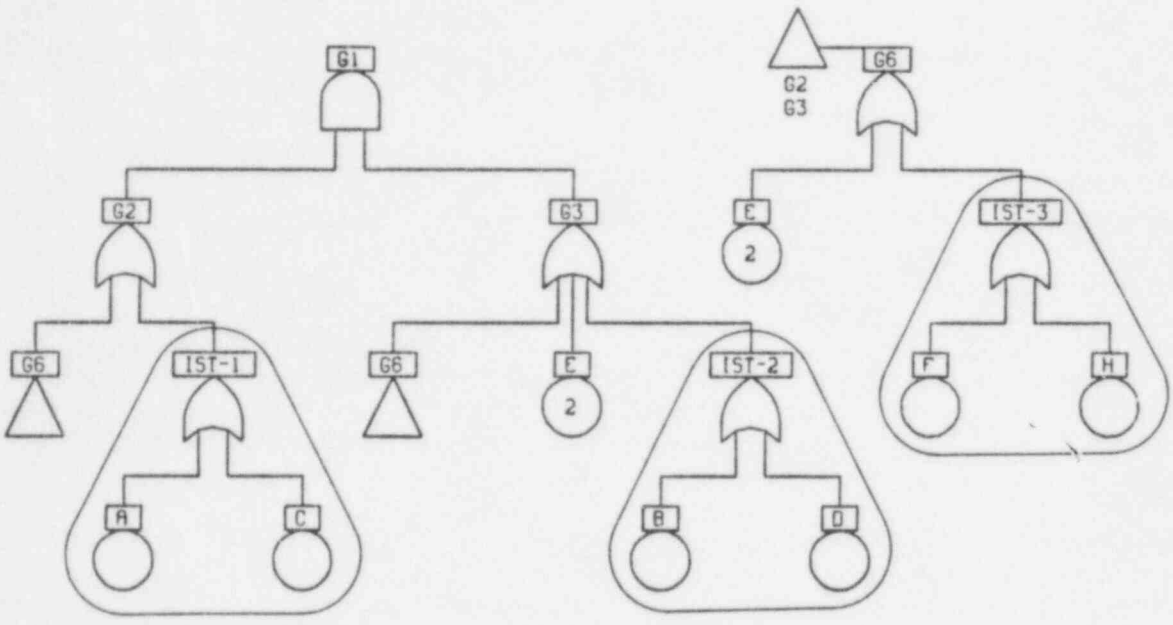


Figure 4.27 Fault tree produced from CREATE-IND-ST by removing G4, coalescing G5 with G3, combining G7 with G6, and creating three LSIP subtrees.

The processing achieved by a call of FRMNEWFT using the techniques of merging fault trees, restructuring a fault tree, and creating independent subtrees can now be described.

The Form New Fault Tree procedure creates a new fault tree by merging existing fault trees. If the block file does not contain a fault tree for every ft_i specified in the call, the merging process will not be completed and a SETS user program error will occur (see APPENDIX A, Section 3.3, Numbered Errors, error 37). However, a merged fault tree will be produced if fault trees exist for every ft_i and no errors are detected when the fault trees are merged.

Fault trees specified by parameters, ft_i, i=1,2,...,n, in a FRMNEWFT call are merged into a single fault tree. If only one fault tree is specified in the call (n=1), the "merged" fault tree is ft_1. Two or more fault trees are merged into a single fault tree by merging pairs of fault trees as described in the section entitled "MERGING FAULT TREES". First, ft_1 and ft_2 are merged into a single fault tree, then this fault tree and ft_3 are merged into a single fault tree, etc., until the fault tree produced by merging ft_1 through ft_{n-1} is finally merged with ft_n. The fault tree obtained by merging n fault trees, $1 \leq n$, is the same regardless of the order in which the fault trees are merged.

After the merged fault tree has been obtained it is modified by any options in the procedure call (see this procedure, Options).

Parameters FORM1, FORM2, and FORM3 determine whether or not the merged, modified fault tree will be restructured and how many fault trees will be created.

One new fault tree is created for parameters FORM1 and FORM2. For FORM1, it is the merged, modified fault tree without any restructuring. For FORM2, it is the merged, modified fault tree restructured in two of the three ways described in the section entitled "RESTRUCTURING A FAULT TREE." Single-input events defined by AND, OR, or EXACTLY ONE gates are removed, and single-output events defined by AND or OR gates are coalesced with their output events if the outputs are also defined by AND or OR gates, respectively.

Two new fault trees are created for parameter FORM3. The merged, modified fault tree is restructured in all of the ways described in the section entitled "RESTRUCTURING A FAULT TREE." Restructuring for FORM3 includes the restructuring described for FORM2. In addition, multiple-output intermediate events having the same outputs are combined into one event if the multiple-output events and their outputs are all defined by AND gates or they are all defined by OR gates.

After the fault tree is restructured, an attempt is made to create an equivalent fault tree which has more or bigger LSIP subtrees and thereby decrease the size of the stem. First, the stem and LSIP subtrees of the restructured fault tree are identified. Then, looking only in the stem, an attempt is made to find an event that can be changed to create a new LSIP subtree using the method described in the section entitled "CREATING INDEPENDENT SUBTREES." After each new subtree is created and looking only in the steadily diminishing stem, the search to find another event that can be changed to create a new LSIP subtree continues. The process terminates when a fault tree has been produced that has a stem in which no additional or larger LSIP subtrees can be created.

A new name is created for the top event of each created subtree. It begins with a standard prefix, IST-, and is followed by an integer which begins with 1 for each call of FRMNEWFT. However, the standard prefix can be replaced by a prefix specified in the procedure call (form d). Using the standard prefix, created names are IST-1, IST-2, etc., except when a created name is the same as a name already in the fault tree. When this happens, the created name having the next higher integer is formed until a distinct name is produced.

After attempts to create additional or larger LSIP subtrees in the restructured fault tree are complete, the resulting fault tree is separated into its stem and collection of LSIP subtrees. The stem fault tree is produced first. If the stem is empty, it means the fault tree is two or more LSIP subtrees which should be analyzed separately. Otherwise, the stem fault tree is formed. After the stem fault tree has been added to the block file, the collection of LSIP subtrees is formed by retaining these subtrees and deleting the rest of the fault tree. If the collection of LSIP subtrees is empty, it means the fault tree cannot be analyzed by separating it in this particular way.

A SETS user program error will occur if either the stem fault tree or the collection of LSIP subtrees fault tree is empty (see APPENDIX A, Section 3.3, Numbered Errors, error 11). If the stem is empty, the block file will not be changed. If the stem is not empty and the collection of LSIP subtrees is empty, the stem fault tree will be added to the block file before the error occurs.

Hence, for parameter FORM3, fault trees ft_i, i=1,2,...,n, are merged into a single fault tree and modified by options specified in the call. The merged, modified fault tree is restructured and more or bigger LSIP subtrees which decrease the size of the stem are created. Then, the resulting fault tree is separated into its stem and collection of LSIP subtrees.

Each final fault tree, one for FORM1 and FORM2 and two for FORM3, is checked to determine if it contains cycles (unending sequences of substitutions). If a fault tree contains a cycle, a SETS user program error will occur (see APPENDIX A, Section 3.3, Numbered Errors, error 48); otherwise, fault tree block ft for FORM1 and FORM2, or ft_S and ft_I for FORM3 will be created and added to the block file.

Options

Options in a call of FRMNEWFT (form e) specify fault tree changes that are made after the merged fault tree has been produced and before parameter FORM1, FORM2, or FORM3 is processed. Options provide ways to identify and retain a particular subtree of a fault tree, alter a fault tree by setting specified primary events to constants, change event names, and change primary and intermediate event types.

Options that can occur in a FRMNEWFT call are the top option, the trim option, the omega option, the phi option, the name option and the type option. Any combination of options can occur and they can occur in any order. Moreover, there can be more than one occurrence of the same kind of option. However, options are executed one at a time in the left-to-right order of their occurrence in the call. A new fault tree exists after each option is executed and it is this fault tree to which the next option applies.

At least one event must be specified in each FRMNEWFT option and, for the name option, at least one pair of events must be specified. An event not in the fault tree can be specified in an option, but the event does not cause any changes in the fault tree. Also, repeating an event in an option has no effect except in the name option whose elements cause name changes to occur as each element is encountered when processing the option from left to right. The constant OMEGA cannot occur in an option in FRMNEWFT.

Every FRMNEWFT option except the name option requires identification and separation of a subtree from the remainder of the fault tree so that the subtree can be retained and the remainder of the fault tree can be deleted. A subtree is separated from the remainder of the fault tree by removing outputs from events in the subtree to events not in the subtree. Consider merged fault tree FT-3-AND-4, which appears first in Figure 4.15, and is reproduced in Figure 4.28 because it will be used to illustrate the FRMNEWFT options. Subtree S(G50) has four events which have outputs outside of the subtree: X1, X3, G50, and G60. Removing the outputs from X1 to G20, X3 to G90, G50 to G40, and G60 to G30 separates S(G50) from the remainder of the fault tree.

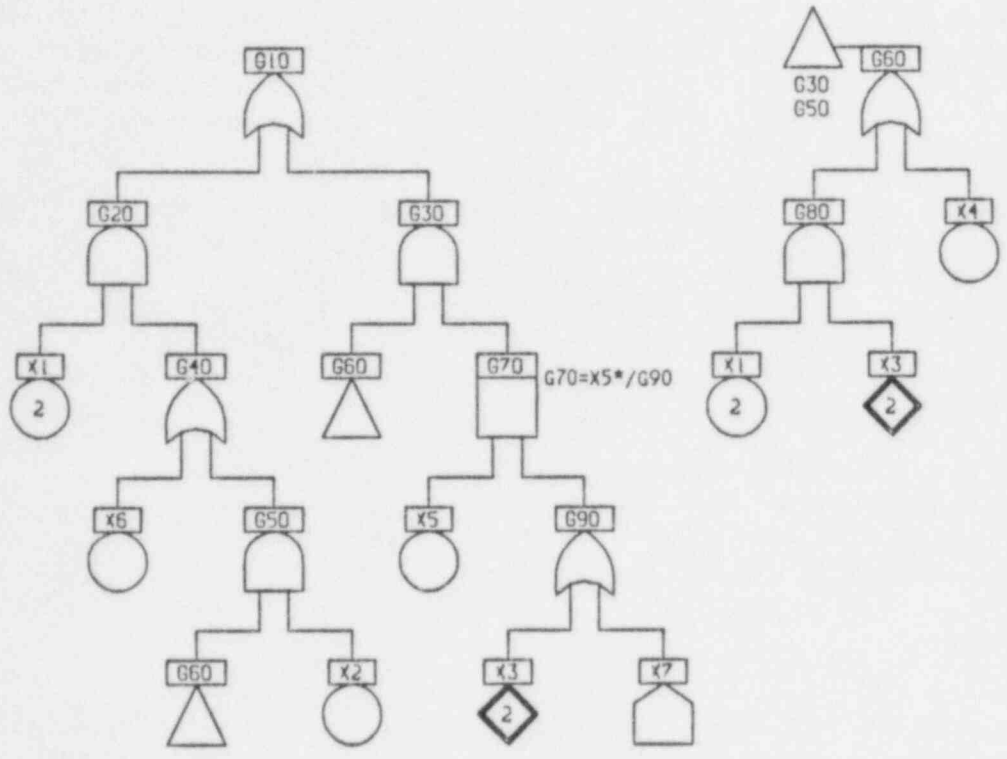


Figure 4.28 Fault tree FT-3-AND-4.

Retaining a subtree and deleting the remainder of the fault tree produces an empty fault tree whenever the retained subtree is empty. This can happen for every option except the name option. If an option produces an empty fault tree, a SETS user program error is detected (see APPENDIX A, Section 3.3, Numbered Errors, error 11).

A top option has the form:

TOP$ $ie_1,ie_2,...,ie_r$

A top option identifies a subtree that will be retained. A top option cannot contain a primary event of the fault tree.

Subtree $S(ie_1,ie_2,...,ie_r)$, determined by intermediate events ie_i, i=1,2,...,r, is identified and separated from the remainder of the fault tree. Then, the remainder of the fault tree is deleted leaving $S(ie_1,ie_2,...,ie_r)$ as the new fault tree. Top events of the new fault tree are a subset of events ie_i, i=1,2,...,r.

Consider fault tree FT-3-AND-4 in Figure 4.28. Top option

TOP$ G30

determines subtree S(G30) having top event G30. Events X1, G30, and G60 all have one output to an event outside of S(G30). Removal of these outputs separates S(G30) from the rest of the fault tree which is deleted. The fault tree produced by this option, S(G30), is shown in Figure 4.29.

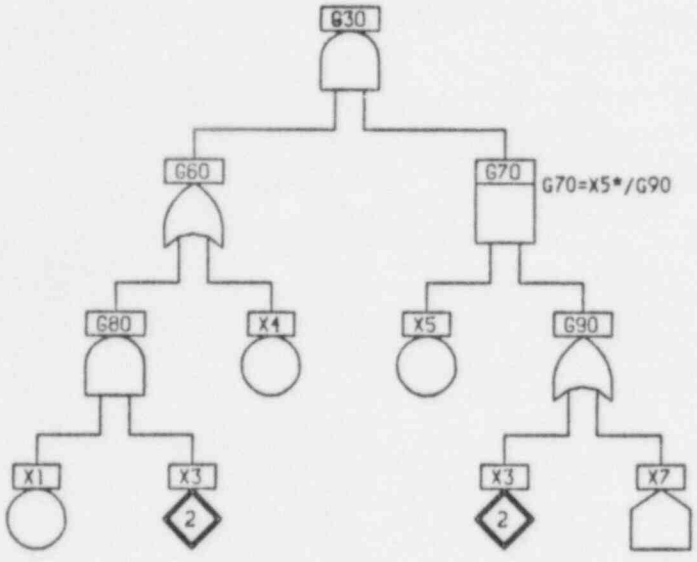


Figure 4.29 Fault tree defined by TOP$ G30 from fault tree FT-3-AND-4.

A trim option has the form:

TRIM$ $f_1, f_2, \ldots, f_r$

A trim option identifies a subtree to be retained. An input of an intermediate event defined by a SPECIAL gate cannot be specified in a trim option. (If inputs of an intermediate event defined by a SPECIAL gate were trimmed, the logic expression for the event would have to be changed so it does not depend on the trimmed inputs. Currently, this is not done.)

Imagine fault tree, ft', created from original fault tree, ft, by removing every output of each event specified in the trim option. Removing every output of a primary event produces an event that is deleted because a primary event must have at least one output. Removing every output of an intermediate event that has at least one output, produces a top event of ft' that is not a top event of ft. Also, if an intermediate event has at least one output and all of its inputs are removed because they are also outputs of events specified in a trim option, the event is changed to a developed event having the same name as the intermediate event. An intermediate event is deleted if it has no outputs and all of its inputs are removed. The subtree retained by a trim option is the subtree of ft' determined by top events of ft' that are not specified in the trim option.

Consider fault tree FT-3-AND-4 in Figure 4.28. Trim option

TRIM$ X2, G60

defines fault tree FT-3-AND-4' in which outputs of X2 and G60 have been removed. Event X2 is deleted because it is a primary event and all of its outputs have been removed. Also, intermediate event G50 in FT-3-AND-4 is a developed event in FT-3-AND-4' because all of its inputs have been removed. Events G10 and G60 are top events of FT-3-AND-4', but only G10 is not specified in the trim

option. Thus, as shown circumscribed by a dashed line in Figure 4.30, subtree S(G10) of fault tree FT-3-AND-4' is retained as the new fault tree.

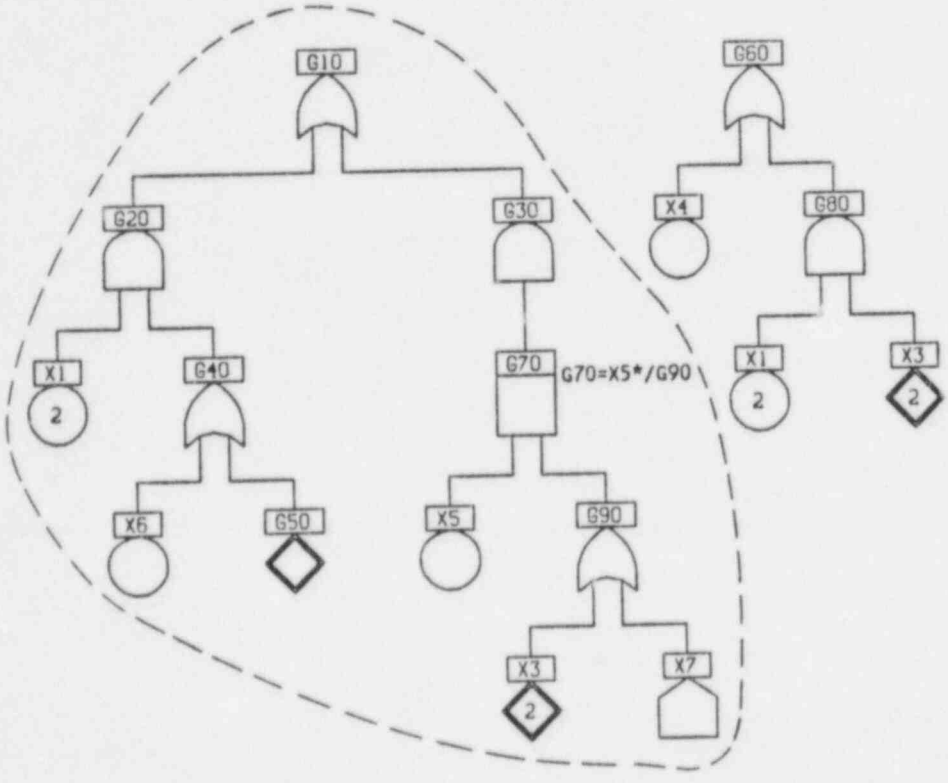


Figure 4.30 Fault tree FT-3-AND-4' and subtree S(G10) defined by TRIM$ X2, G60.

An omega option has the form:

OMEGA$ pe_1,pe_2,...,pe_r

An omega option specifies primary events that are set to the constant OMEGA. An omega option cannot contain an intermediate event.

After primary events, pe_i, i=1,2,...,r, are set to OMEGA, intermediate events that can be determined to be constant (OMEGA or /OMEGA) as a result of the primary event settings are identified. Determining that an intermediate event is constant depends on whether or not any of its inputs are constant and the type of gate which defines the event.

An intermediate event defined by an OR gate is set to OMEGA if any input is OMEGA, and it is set to /OMEGA if all of its inputs are /OMEGA.

An intermediate event defined by an AND, PRIORITY AND, or INHIBIT gate is set to /OMEGA if any input is /OMEGA, and it is set to OMEGA if all of its inputs are OMEGA.

Two cases arise for an intermediate event defined by an EXACTLY ONE gate:

1. All inputs are constant.

   If exactly one input is OMEGA, the intermediate event is set to OMEGA; but, if there is not exactly one input that is OMEGA, the intermediate event is set to /OMEGA.

2. Some, but not all, inputs are constant.

   If exactly one of the constant inputs is OMEGA, the EXACTLY ONE gate is changed to a SPECIAL gate whose logic expression is a product of complements of inputs that are not constant. If more than one constant input is OMEGA, the intermediate event is set to /OMEGA.

If any inputs of an intermediate event defined by a SPECIAL gate are constant, the logic expression which defines the event is expanded, simplified, and factored using the reduction process described in Section 4.1.5, Reduce Equation. If the expression reduces to OMEGA or /OMEGA, the event is set to this constant; otherwise, the reduced expression replaces the logic expression which defines the event.

After every intermediate event in the fault tree that can be determined to be constant has been set to OMEGA or /OMEGA, constant events are trimmed from the fault tree. This means all primary and intermediate events set to OMEGA or /OMEGA are treated as if they had been specified in a trim option. (A constant input of an intermediate event defined by a SPECIAL gate can be trimmed in this situation because the logic expression for the event has been correctly reduced.)

Consider fault tree FT-3-AND-4 in Figure 4.28. Omega option

OMEGA$ X1, X3

causes primary events X1 and X3 to be set to OMEGA. Intermediate events G60, G80, and G90 are determined to be OMEGA; and events G30 and G70 are determined to be /OMEGA. Fault tree FT-3-AND-4' is created from FT-3-AND-4 by removing all outputs of events X1, X3, G60, G80, G90, G30 and G70 as if the events had been specified in a trim option.

Events X1, X3, G30, and G80 are not in FT-3-AND-4'; they are deleted because all of their inputs and outputs are removed during the trim process. The new fault tree has top events G10, G60, G70, and G90, but only G10 is not specified in the trim option. Subtree S(G10) of fault tree FT-3-AND-4', circumscribed by a dashed line in Figure 4.31, is the fault tree created by this option.

A phi option has the form:

PHI$ $pe_1, pe_2, \ldots, pe_r$

A phi option specifies primary events that are set to the constant /OMEGA. A phi option cannot contain an intermediate event.

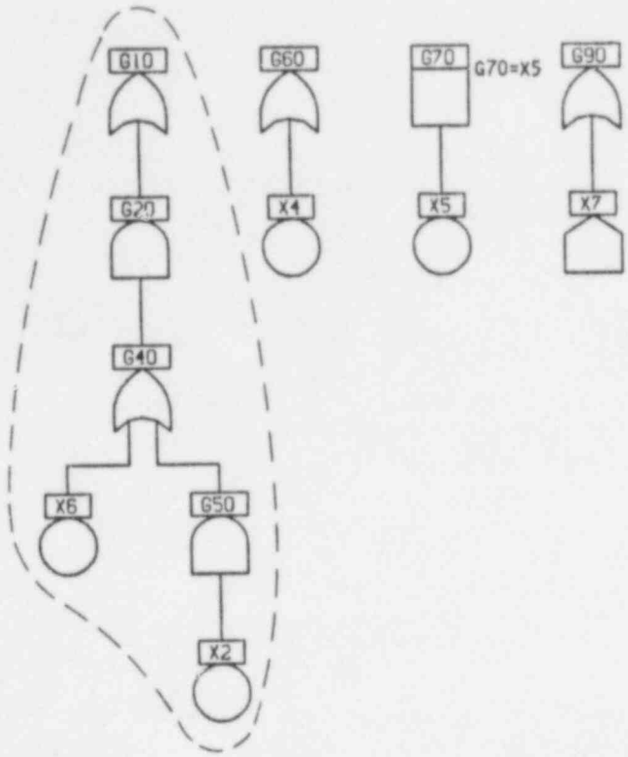


Figure 4.31 Fault tree FT-3-AND-4' and subtree S(G10) defined by OMEGA$ X1, X3.

A phi option is processed the same way an omega option is processed except that primary events, pe_i, i=1,2,...,r, are set to /OMEGA instead of OMEGA. Consider fault tree FT-3-AND-4 in Figure 4.28. Phi option

PHI$ X3, X7

causes primary events X3 and X7 to be set to /OMEGA. Intermediate events G80 and G90 are determined to be /OMEGA. Fault tree FT-3-AND-4' is created from FT-3-AND-4 by removing all outputs of events X3, X7, G80, and G90 as if they had been specified in a trim option. Events X3, X7, and G90 are not in the new fault tree; they are deleted because all of their inputs and outputs are removed during the trim process. The new fault tree has top events G10 and G80, but only G10 is not specified in the trim option. Subtree S(G10) of fault tree FT-3-AND-4', circumscribed by a dashed line in Figure 4.32, is the fault tree created by this option.

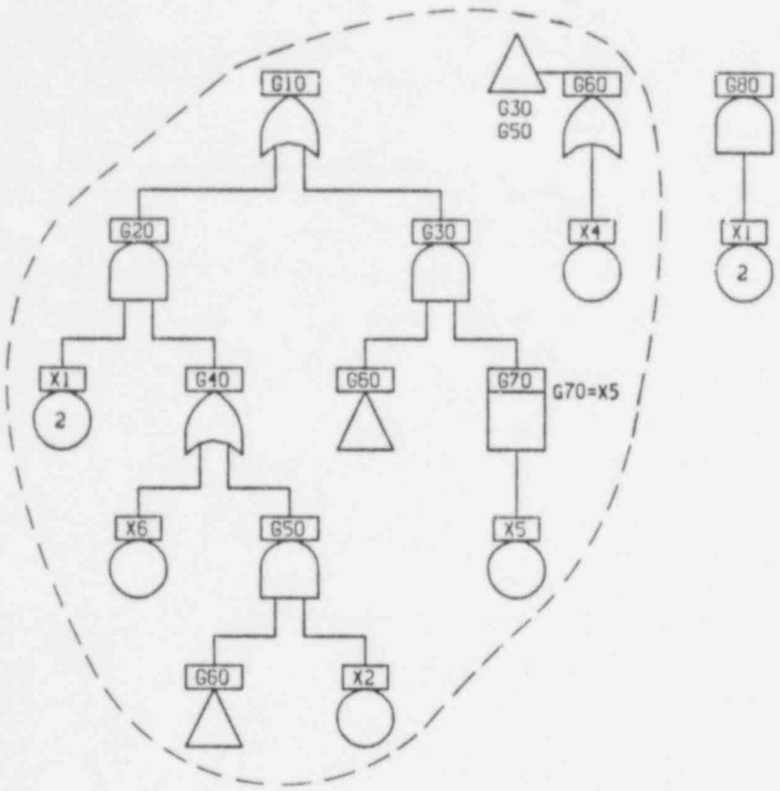


Figure 4.32 Fault tree FT-3-AND-4' and subtree S(G10) defined by PHI$ X3, X7.

A name option has the form:

NAME$ f_1=f_1',f_2=f_2',....,f_r=f_r'

A name option changes event names in a fault tree. Event name f_1 is changed to f_1', event name f_2 is changed to f_2', etc., until event name f_r is changed to f_r'. Each name change takes place as the old and new name pair for the change is encountered when processing the option from left to right. An error is detected if the fault tree already has an event with the same name as a new name specified in the option (see APPENDIX A, Section 3.3, Numbered Errors, error 15).

Consider fault tree FT-3-AND-4 in Figure 4.28. Name option

NAME$ X3=AUX-POWER, X7=POWER

causes the names of primary events X3 and X7 to be changed to AUX-POWER and POWER, respectively. The fault tree created by this option is shown in Figure 4.33.

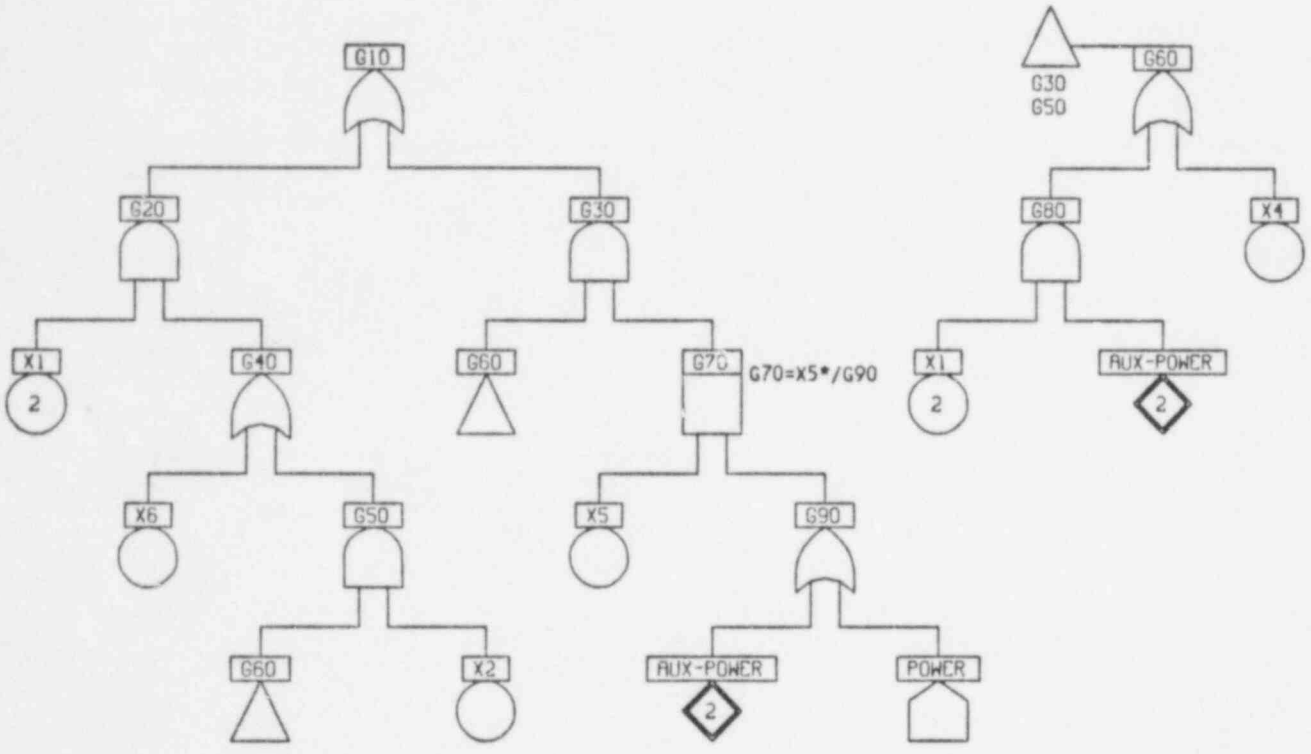


Figure 4.33 Fault tree defined by NAME$ X3=AUX-POWER, X7=POWER from fault tree FT-3-AND-4.

A type option has the form:

⟨type⟩$ $f_1,f_2,\ldots,f_r$

A type option identifies events whose types will be changed to the type specified in the option. Primary events can be changed to another type of primary event. Intermediate events can be changed to a type of primary event or to another type of intermediate event. However, a primary event cannot be changed to an intermediate event, and no event can be changed to an intermediate event defined by a SPECIAL gate.

A type option can be used to change any event to a primary event. The type of events f_i, i=1,2,...,r, will be set to the primary event type specified by ⟨type⟩ in the option. Choosing primary event type BE, EE, DE, UE, or CE for ⟨type⟩ changes the events specified in the option to basic, external, developed, undeveloped, or conditioning events, respectively.

A primary event is changed to another type of primary event simply by changing the type of the event to the type specified in the option. Changing an intermediate event to a primary event is achieved by removing all inputs of the intermediate event and replacing its gate type by the primary event type specified in the option. Removing all inputs of every intermediate event specified in the option can produce events without any outputs. (If every output of event X is an input of an intermediate event specified in the option, removing inputs of the intermediate

events leaves X without any outputs.) Primary events without outputs are deleted because primary events must have at least one output. Intermediate events without outputs are identified and treated as if they had been specified in a trim option.

A type option can be used to change an intermediate event to another type of intermediate event. The gate type of intermediate events f_i, i=1,2,...,r, will be set to the gate type specified by ⟨type⟩ in the option. Choosing gate type AG, OG, EOG, PAG, or IG (but not SG) for ⟨type⟩ changes the logic which defines the intermediate event to AND, OR, EXACTLY ONE, PRIORITY AND or INHIBIT, respectively. Changing the logic of an intermediate event is achieved simply by setting the gate type of the event to the gate type specified in the option.

Consider fault tree FT-3-AND-4 in Figure 4.28. Type option

DE$ X4, G30

changes basic event X4 and intermediate event G30 into developed events. No further processing is required for primary event X4. Inputs of intermediate event G30 are removed leaving G60 with one output and G70 without any outputs. Thus, G70 is the only event treated as if it had been specified in a trim option. Outputs of G70 have already been removed so the fault tree has top events G10 and G70, but only G10 is not specified in the trim option. Subtree S(G10) of fault tree FT-3-AND-4', circumscribed by a dashed line in Figure 4.34, is the fault tree created by this option.

Printed Output

The printed output produced by the Form New Fault Tree procedure begins with the output produced by merging fault trees ft_i, i=1,2,...,n. This output consists of a list of the fault trees merged and has the form:

FAULT TREES MERGED

1. ft_1
2. ft_2
. .
. .
. .
n. ft_n

Options specified in a FRMNEWFT call are processed after the merged fault tree has been formed. They are processed one at a time in the left-to-right order that they appear in the call. The fault tree is modified as each option is processed and information about the modification is printed. The printed output for every option begins with a representation of the option and ends with a message indicating the number of events removed from the fault tree because of the option. This is the only output produced for the top, trim, name, and type options. For omega and phi options, a list of constant events is also printed. If an event specified in an option is not an event in the fault tree, a message indicating the fault tree does not contain the event is printed. Suppose a top option specifies events A, B, and C,

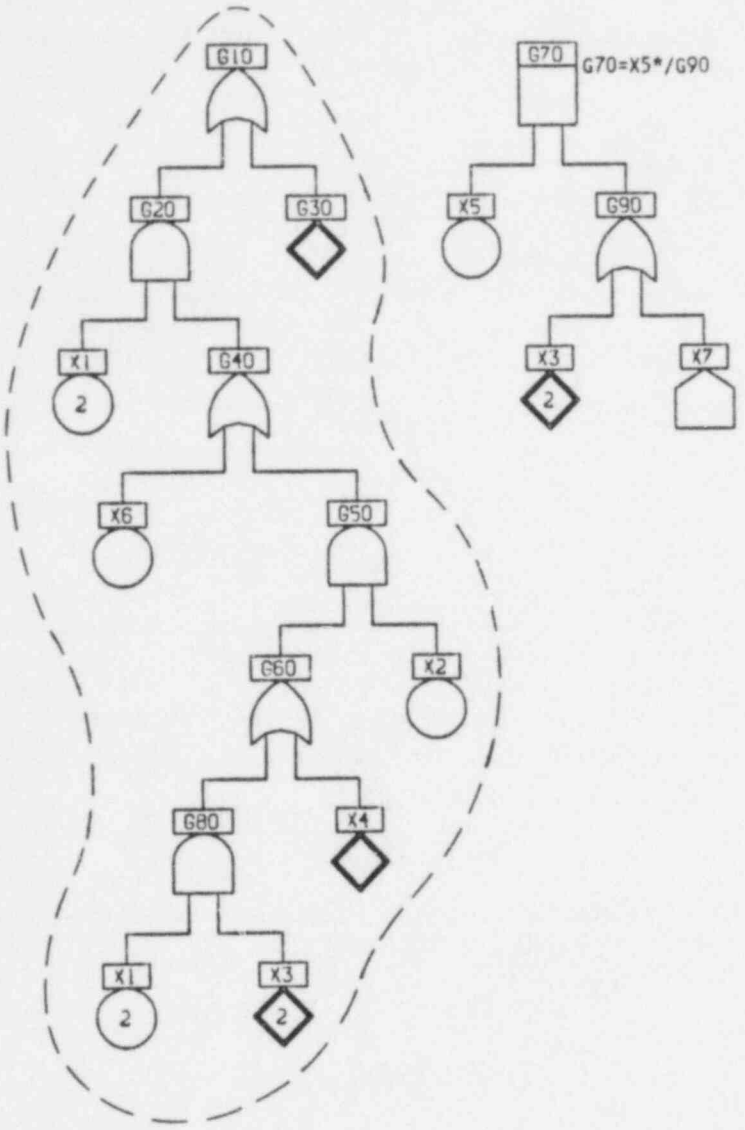


Figure 4.34 Fault tree FT-3-AND-4' and subtree S(G10) defined by DE$ X4, G30.

and that A and C are in the fault tree, but B is not. If this option does not cause any events to be removed from the fault tree, the output has the form:

```
TOP$ A, B, C

     THE FAULT TREE DOES NOT CONTAIN B

     THERE WERE NO EVENTS REMOVED
     BECAUSE OF THIS TOP OPTION
```

If the option causes w > 0 events to be removed from the fault tree, the output has the form:

```
TOP$ A, B, C

     THE FAULT TREE DOES NOT CONTAIN B

     THERE WERE     w EVENTS REMOVED
     BECAUSE OF THIS TOP OPTION
```

(The name option does not cause events to be removed from the fault tree, so the form of the output for w > 0 never occurs for this option.)

The list of constant events printed for omega and phi options contains primary events set to OMEGA or /OMEGA because they were specified in the option and intermediate events determined to be OMEGA or /OMEGA from the primary event settings. Suppose primary event A is specified in a phi option and intermediate events B and C are determined to be OMEGA and /OMEGA, respectively. Suppose also, that treating these constant events as if they had been specified in a trim option, which is part of the processing for the omega and phi options, causes w > 0 events to be removed from the fault tree. The printed output for this option has the form:

```
PHI$ A

          A=/OMEGA
          B=OMEGA
          C=/OMEGA

     THERE WERE     w EVENTS REMOVED
     BECAUSE OF THIS PHI OPTION
```

It is possible that an omega or phi option will produce the message that there were no events removed from the fault tree. If none of the events specified in the option are in the fault tree, none of the fault tree events are constant and no events are trimmed from the fault tree. Suppose event A in the previous example is not in the fault tree. The printed output has the form:

```
PHI$ A

     THE FAULT TREE DOES NOT CONTAIN A

     THERE WERE NO EVENTS REMOVED
     BECAUSE OF THIS PHI OPTION
```

The output produced by merging fault trees and applying options occurs for every call of FRMNEWFT. Differences in the output arise from the different processing indicated by parameters FORM1, FORM2, and FORM3.

Parameter FORM1 creates a single fault tree, ft, which is the merged, modified fault tree. It does not require further processing. The rest of the printed output

for FORM1 consists of information about ft. Developed event information is printed first. If ft does not have any developed events, the message

THERE ARE NO DEVELOPED EVENTS IN ft

is printed; otherwise, a list of the developed events in the fault tree, D_j, j=1,2,...,m, is printed in the form:

DEVELOPED EVENTS IN ft

1. D_1
2. D_2
. .
. .
. .
m. D_m

Next, information about equivalent intermediate events in ft is printed. No message is printed if there are no equivalent events. Otherwise, equivalent intermediate events are listed as a group. If there is more than one group, each group is listed. Equivalent events do not cause any changes in the fault tree; they are simply identified and listed.

After equivalent events have been listed, a message indicating the fault tree does not contain cycles is printed. (SETS user program error 48 occurs if the fault tree contains cycles.) Then, top events of the fault tree are listed, and a message showing fault tree block ft has been added to the block file is printed. Suppose ft has one group of equivalent events E_j, j=1,2,...,m, and top events T_k, k=1,2,...,s. If ft does not contain cycles, the following output is printed:

THE FOLLOWING EVENTS ARE EQUIVALENT

1. E_1
2. E_2
. .
. .
. .
m. E_m

THERE ARE NO CYCLES IN ft

TOP EVENTS OF ft

1. T_1
2. T_2
. .
. .
. .
s. T_s

FAULT TREE BLOCK ft
HAS BEEN ADDED TO THE BLOCK FILE

Developed event information printed for parameter FORM1 is also printed for parameter FORM2. Printed output about equivalent events, cycles, top events, and adding the fault tree to the block file is printed for every fault tree created by FRMNEWFT.

The printed output for parameter FORM2 differs from the output for FORM1 only by the output produced when the merged, modified fault tree is restructured. Restructuring occurs after options in the procedure call have been processed and before developed events are identified and printed. For FORM2, restructuring consists of removing certain single-input events and coalescing certain single-output events with their output events. Each of these ways of restructuring the fault tree produces printed output. Suppose single-input events U_j, j=1,2,...,m, are removed from the fault tree and events V_k, k=1,2,...,s, are coalesced with their respective output events. The printed output has the form:

SINGLE-INPUT EVENTS REMOVED

1. U_1
2. U_2
. .
. .
. .
m. U_m

EVENTS REMOVED BY COALESCING

1. V_1
2. V_2
. .
. .
. .
s. V_s

If no single-input events are removed from the fault tree, the message

NO SINGLE-INPUT EVENTS WERE REMOVED

is printed instead of a list of single-input events removed; and if no events are coalesced, the message

NO EVENTS WERE COALESCED

is printed instead of a list of events removed by coalescing.

Parameter FORM3 produces two fault trees. The merged, modified fault tree is restructured and an attempt is made to create independent subtrees which decrease the size of the stem, before the resulting fault tree is separated into its stem and collection of LSIP subtrees.

Restructuring for FORM3 includes all three ways of restructuring a fault tree: removing single-input events, coalescing events with their outputs, and combining multiple-output events which have the same outputs. For FORM3, restructuring is

an iterative process because combining multiple-output events can produce single-input events that can be removed or events that can be coalesced with their output events.

A message identifies the beginning of each iteration of the process. It is followed by information about single-input events removed, events coalesced, and events combined during that iteration. Single-input events removed and events removed by coalescing are simply listed for each iteration. Events removed by combining are listed in groups. A separate message is printed for each group of multiple-output events that are combined; it identifies the event that remains in the fault tree and lists the events that are combined with it and then removed from the fault tree. Suppose single-input events U_j, j=1,2,...,m, are removed from the fault tree and events V_k, k=1,2,...,s, are coalesced with their output events during the p-th iteration. Suppose also, that multiple-output events $W_{X,g}$, g=1,2,...,r, which have the same outputs as event X, and multiple-output events $W_{Y,h}$, h=1,2,...,q, which have the same outputs as event Y, are combined with events X and Y, respectively. The printed output for the p-th iteration has the form:

ITERATION p

SINGLE-INPUT EVENTS REMOVED

1. U_1
2. U_2
. .
. .
. .
m. U_m

EVENTS REMOVED BY COALESCING

1. V_1
2. V_2
. .
. .
. .
s. V_s

EVENTS REMOVED AFTER BEING COMBINED WITH EVENT X

1. $W_{X,1}$
2. $W_{X,2}$
. .
. .
. .
r. $W_{X,r}$

EVENTS REMOVED AFTER BEING
COMBINED WITH EVENT Y

1. $W_{Y,1}$
2. $W_{Y,2}$
. .
. .
. .
q. $W_{Y,q}$

If, on a particular iteration, no single-input events are removed, the message

NO SINGLE-INPUT EVENTS WERE REMOVED

is printed instead of a list of events removed. Similarly, if no events are coalesced on a particular iteration, the message

NO EVENTS WERE COALESCED

is printed instead of a list of events removed by coalescing. Finally, if no events are combined on a particular iteration, the message

NO EVENTS WERE COMBINED

is printed instead of groups of events removed by combining. Following the final iteration, which is an iteration during which no multiple-output events are combined, the message

ITERATIONS COMPLETED

is printed.

After the fault tree has been restructured, an attempt is made to create LSIP subtrees that decrease the size of the stem. If no new subtrees can be created, the message

NO LSIP SUBTREES WERE CREATED

is printed. Otherwise, the top event and its inputs for each created subtree are listed in the form:

TOP EVENTS AND THEIR INPUTS
FOR CREATED LSIP SUBTREES

IST-1
input 1 for IST-1
input 2 for IST-1
.
.
.
input k_{IST-1} for IST-1

IST-2
  input 1 for IST-2
  input 2 for IST-2
      .
      .
      .
  input k_{IST-2} for IST-2

    •

    •

    •

IST-w
  input 1 for IST-w
  input 2 for IST-w
      .
      .
      .
  input k_{IST-w} for IST-w

After the fault tree is restructured and subtrees have been created, stem fault tree, ft_S, is formed by deleting every LSIP subtree. Each subtree removed is replaced by a developed event having the same name as the top event of the removed subtree. However, all other events in the LSIP subtrees are deleted. The message

THERE WERE w EVENTS REMOVED
TO FORM THE STEM FAULT TREE ft_S

indicates how many events were removed to form the stem fault tree. Following this message, the messages for ft_S about equivalent events, cycles, top events, and adding the fault tree to the block file are printed.

If ft_S is empty, the fault tree is two or more LSIP subtrees and a SETS user program error occurs when an attempt is made to form ft_S (see APPENDIX A, Section 3.3, Numbered Errors, error 11). For this case, neither ft_S nor ft_I will be added to the block file.

After the stem fault tree is formed, the collection of LSIP subtrees fault tree, ft_I, is formed. All of the events except those in the LSIP subtrees are removed to form ft_I. The message

THERE WERE w EVENTS REMOVED
TO FORM THE COLLECTION OF LSIP
SUBTREES FAULT TREE ft_I

indicates how many events were removed to form ft_I. Following this message, the messages for ft_I about equivalent events, cycles, top events, and adding the fault tree to the block file are printed.

If the fault tree does not have any LSIP subtrees, ft_I is empty and a SETS user program error occurs when an attempt is made to form ft_I (see APPENDIX A, Section 3.3, Numbered Errors, error 11). For this case, ft_S has already been added to the block file when the error occurs, but ft_I will not be added to the block file.

Example

Consider SETS user program:

```
PROGRAM$ FRMNEWFT-EX.
   DLTBLK.
   RDFT (SYS-1, SYS-2, ELEC).
   FRMNEWFT (FORM1$ SYS-1, SYS-2/ SYS-FORM1).
   FRMNEWFT (FORM1$ SYS-1/ TMP-SYS-1* DE$ G3).
   RDFT (NEW-G3).
   FRMNEWFT (FORM1$ TMP-SYS-1, NEW-G3/ NEW-SYS-1).
   FRMNEWFT (FORM3$(NEW-ST-) NEW-SYS-1, SYS-2, ELEC/ SYS-STEM, SYS-LSIP).
```

The first statement in the SETS user program deletes every block from the block file and initializes the file. The second statement reads fault trees SYS-1, SYS-2, and ELEC which are shown in Figure 4.35.

(a) SYS-1

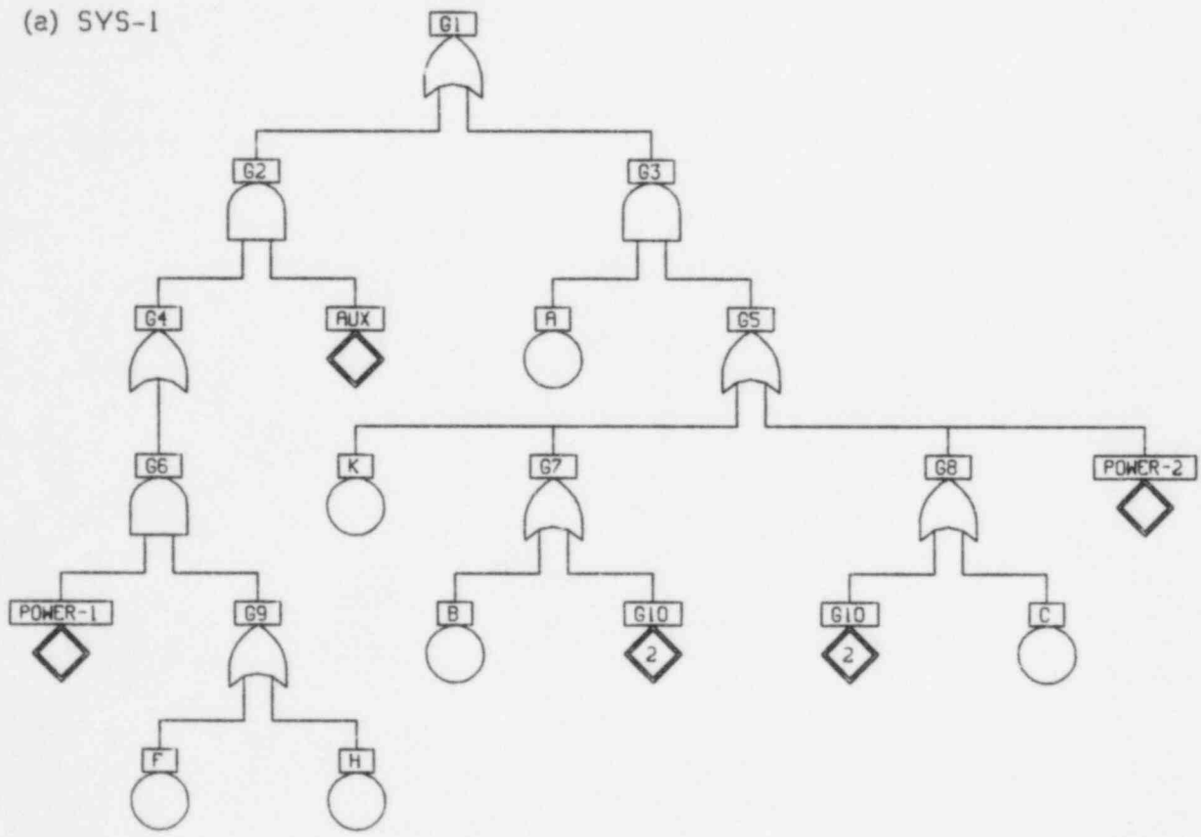
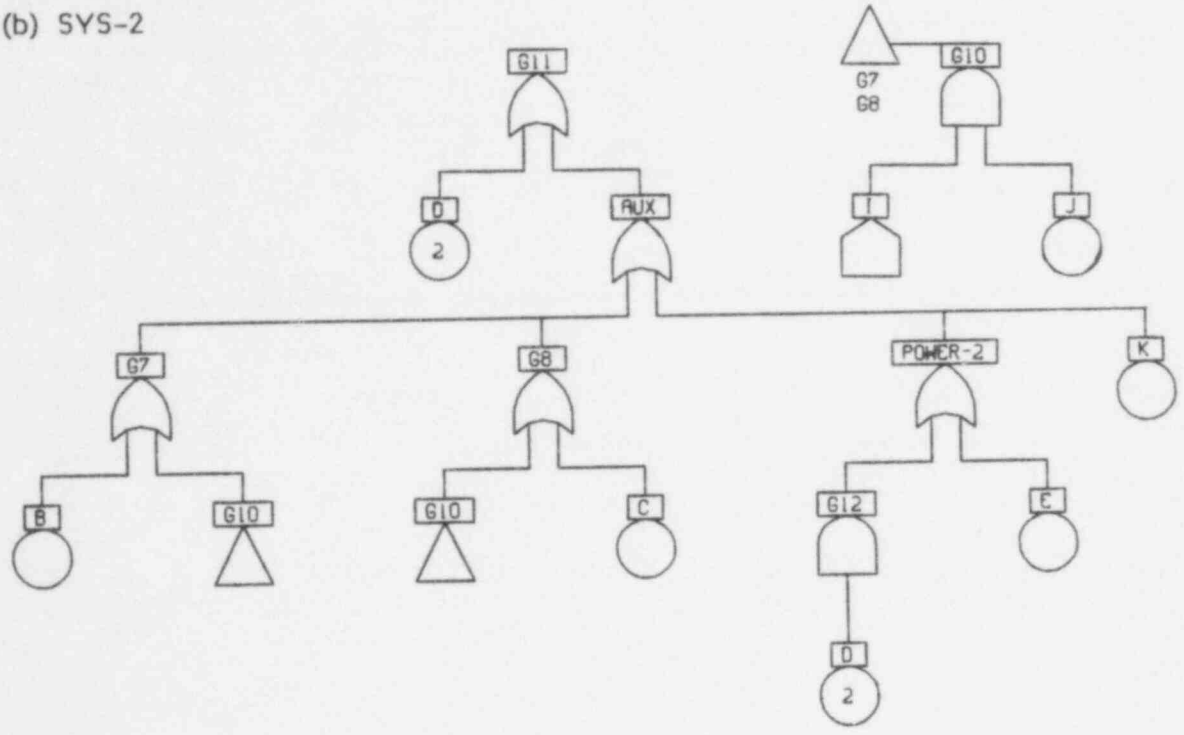


Figure 4.35 Fault trees (a) SYS-1, (b) SYS-2, and (c) ELEC.

(b) SYS-2



(c) ELEC

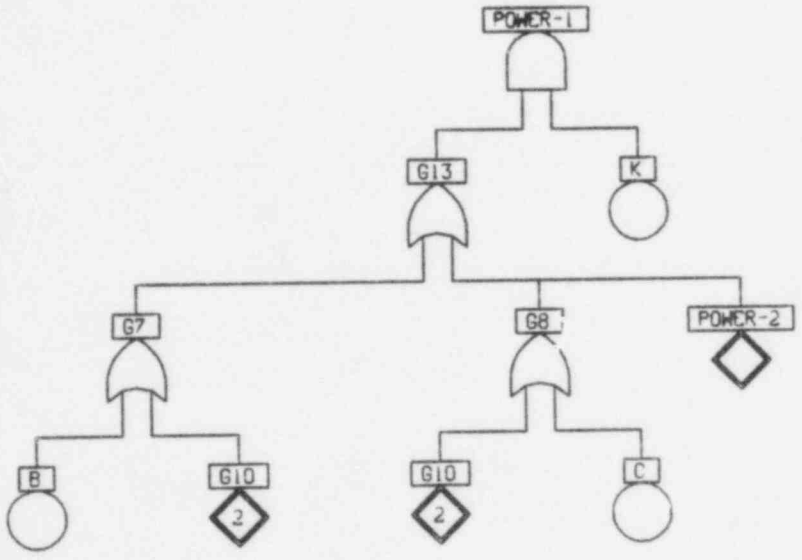


Figure 4.35 Fault trees (a) SYS-1, (b) SYS-2, and (c) ELEC (continued).

The first FRMNEWFT statement merges fault trees SYS-1 and SYS-2 and, in accordance with parameter FORM1, produces the single fault tree, SYS-FORM1. The output from this call of FRMNEWFT shows the merged fault tree contains developed event POWER-1. This suggests event POWER-1 was not developed, or its development was omitted from the list of fault trees to be merged. In this example, fault tree ELEC, having top event POWER-1, was omitted from the call. The output also shows events AUX and G5 are equivalent. The printed output is as follows:

```
FRMNEWFT (FORM1$ SYS-1, SYS-2/ SYS-FORM1).

          FAULT TREES MERGED

              1.  SYS-1
              2.  SYS-2

          DEVELOPED EVENTS IN SYS-FORM1

              1.  POWER-1

          THE FOLLOWING EVENTS ARE EQUIVALENT

              1.  AUX
              2.  G5

          THERE ARE NO CYCLES IN SYS-FORM1

          TOP EVENTS OF SYS-FORM1

              1.  G1
              2.  G11

     FAULT TREE BLOCK SYS-FORM1
     HAS BEEN ADDED TO THE BLOCK FILE

     STATEMENT EXECUTION REQUIRED      .060 SECONDS FOR FRMNEWFT
```

Suppose events AUX and G5 were known to be equivalent when SETS user program FRMNEWFT-EX was written, and event G5 should be changed to AUX. The name option cannot be used to change G5 to AUX because SYS-1 already has event AUX. The fault tree input can be changed and read again, but FRMNEWFT calls can also be used to change G5 to AUX. The FRMNEWFT call with option DE$ G3 creates fault tree TMP-SYS-1 having developed event G3. Primary event A and subtree S(G5) have been trimmed; six primary events (A, B, C, K, G10, and POWER-2) and three intermediate events (G5, G7, and G8) have been removed from the fault tree. The printed output produced by this FRMNEWFT call is as follows:

```
FRMNEWFT (FORM1$ SYS-1/ TMP-SYS-1* DE$ G3).

          FAULT TREES MERGED

              1.  SYS-1

     DE$ G3).

          THERE WERE    9 EVENTS REMOVED
          BECAUSE OF THIS TYPE OPTION

          DEVELOPED EVENTS IN TMP-SYS-1

              1.  G3
              2.  AUX
              3.  POWER-1

          THERE ARE NO CYCLES IN TMP-SYS-1

          TOP EVENTS OF TMP-SYS-1

              1.  G1

     FAULT TREE BLOCK TMP-SYS-1
     HAS BEEN ADDED TO THE BLOCK FILE

     STATEMENT EXECUTION REQUIRED      .046 SECONDS FOR FRMNEWFT
```

The next statement of the SETS user program adds fault tree NEW-G3 to the block file. This fault tree redefines the G5 input of event G3 as shown in Figure 4.36.

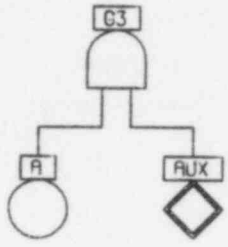


Figure 4.36 Fault tree NEW-G3.

Merging TMP-SYS-1 and NEW-G3 in the next call of FRMNEWFT, creates a new system-1 fault tree. Subtree S(G5) has been replaced by developed event AUX which now has two outputs: G2 and G3. Fault tree NEW-SYS-1, Created by this call of FRMNEWFT, is shown in Figure 4.37.

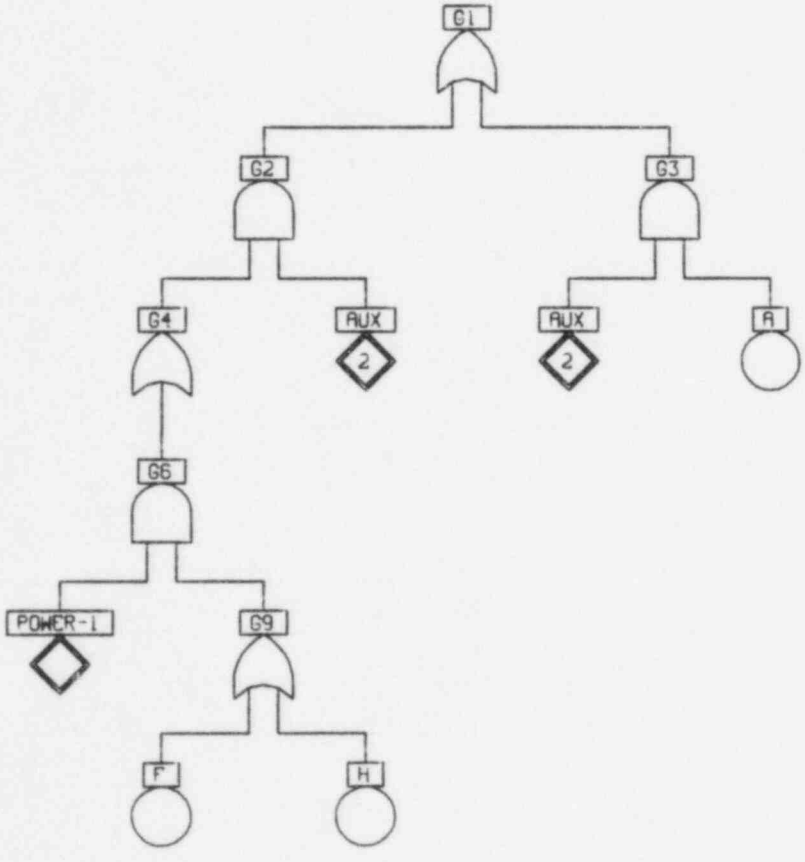


Figure 4.37 Fault tree NEW-SYS-1.

The last statement is a FRMNEWFT call with parameter FORM3. Fault trees NEW-SYS-1, SYS-2, and ELEC are merged into a single fault tree. The merged fault tree is restructured to remove single-input events G4 and G12, coalesce event G6 with G2 and then POWER-1 with G2, combine events G8 and POWER-2 with G7, and remove single-input event G13 (created when G8 and POWER-2 were combined with G7). A new LSIP subtree S(NEW-ST-1) is created using the prefix specified in the call to form the name of its top event. It contains and is bigger than single-top, independent, proper subtree S(G10). Two fault trees, SYS-STEM and SYS-LSIP, are created by this FRMNEWFT call and they are shown in Figure 4.38.

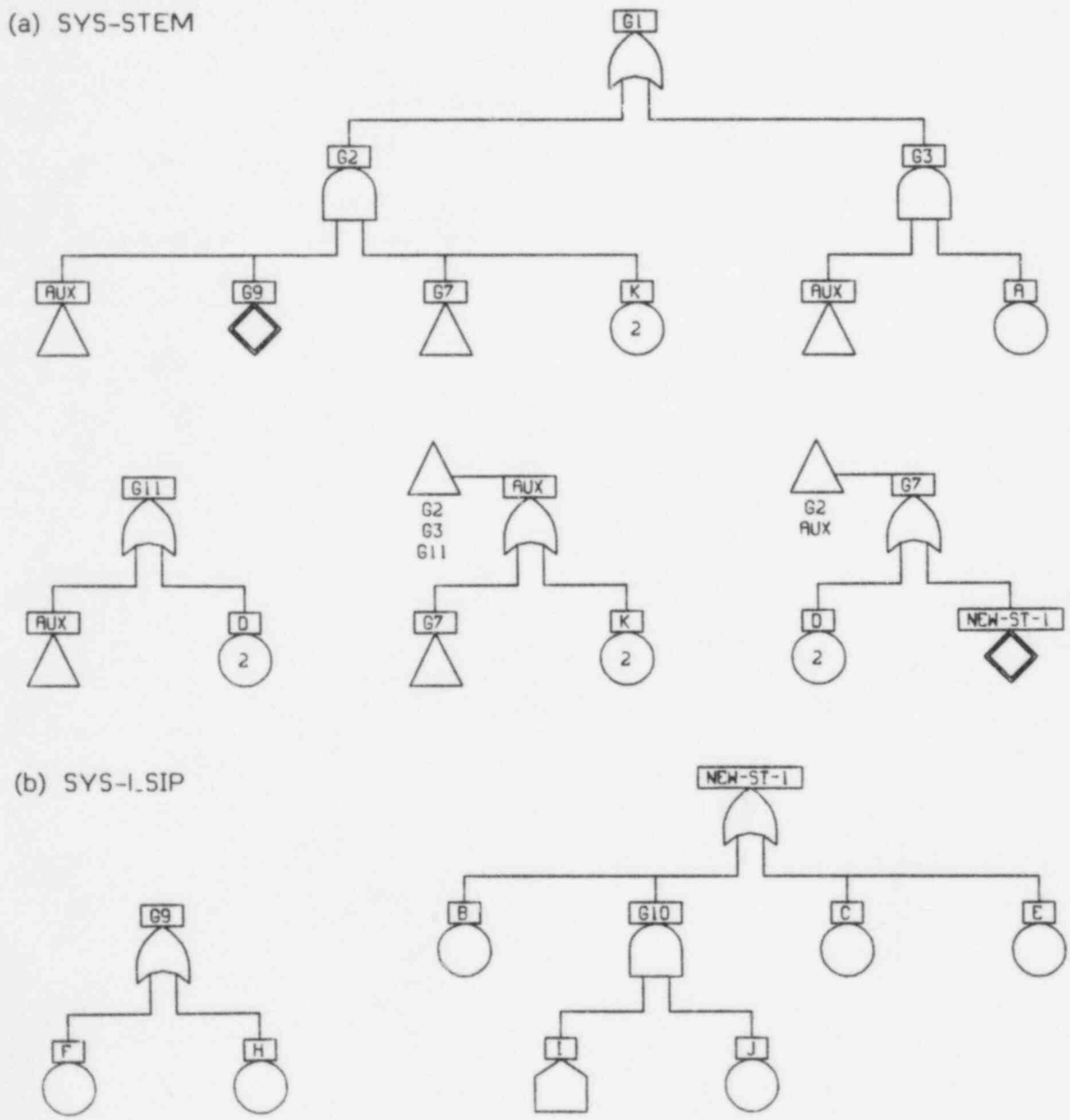


Figure 4.38 Fault trees (a) SYS-STEM and (b) SYS-LSIP.

The printed output produced by the FRMNEWFT call that creates fault trees SYS-STEM and SYS-LSIP is as follows:

```
FRMNE' ( (FORM3$(NEW-ST-) NEW-SYS-1, SYS-2, ELEC/ SYS-STEM, SYS-LSIP).

          FAULT TREES MERGED

               1.  NEW-SYS-1
               2.  SYS-2
               3.  ELEC

     ITERATION  1

          SINGLE-INPUT EVENTS REMOVED

               1.  G4
               2.  G12

          EVENTS REMOVED BY COALESCING

               1.  G6
               2.  POWER-1

          EVENTS REMOVED AFTER BEING
          COMBINED WITH EVENT G7

               1.  G8
               2.  POWER-2

     ITERATION  2

          SINGLE-INPUT EVENTS REMOVED

               1.  G13

          NO EVENTS WERE COALESCED

          NO EVENTS WERE COMBINED

     ITERATIONS COMPLETED

          TOP EVENTS AND THE . INPUTS
          FOR CREATED LSIP .UBTREES

               NEW-ST-1
                 B
                 G10
                 C
                 E

          THERE WERE    8 EVENTS REMOVED
          TO FORM THE STEM FAULT TREE SYS-STEM

          THERE ARE NO CYCLES IN SYS-STEM

          TOP EVENTS OF SYS-STEM

               1.  G1
               2.  G11

     FAULT TREE BLOCK SYS-STEM
     HAS BEEN ADDED TO THE BLOCK FILE

          THERE WERE    9 EVENTS REMOVED
          TO FORM THE COLLECTION OF LSIP
          SUBTREES FAULT TREE SYS-LSIP

          THERE ARE NO CYCLES IN SYS-LSIP
```

```
     TOP EVENTS OF SYS-LSIP

          1.  G9
          2.  NEW-ST-1

FAULT TREE BLOCK SYS-LSIP
HAS BEEN ADDED TO THE BLOCK FILE

STATEMENT EXECUTION REQUIRED      .178 SECONDS FOR FRMNEWFT
```

This page intentionally left blank.

4.4.3 Generate Fault Tree Equation

The Generate Fault Tree Equation procedure is used to obtain minimal cut set equations for every top event in a fault tree. A SETS user program segment is generated which, when executed, produces minimal cut set equations for the top events, creates a block which contains the equations, and enters the block into the block file. Parameters in the call specify the fault tree to be processed and the name of the block to be created. Another parameter determines whether a top-down or a bottom-up method is used to obtain the equations. If parameters for computing term values occur in the call, truncated minimal cut set equations are obtained. For example, suppose fault tree NEW-FT, which is first shown in Figure 4.13 and is reproduced here for convenience, is in the block file.

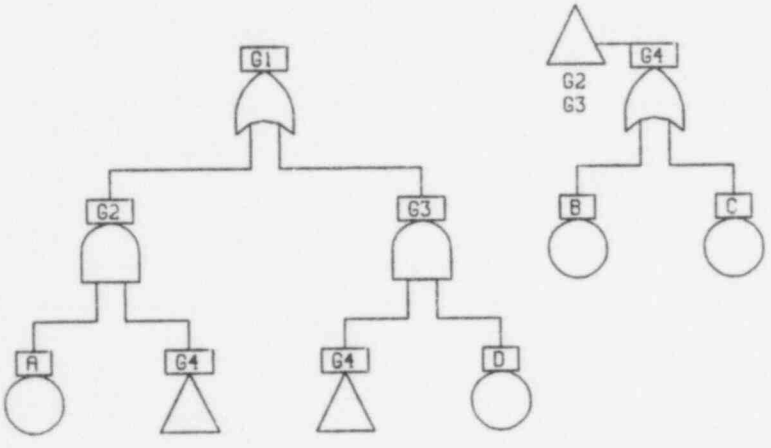


Figure 4.39 Fault tree NEW-FT.

Execution of procedure call statement

GENFTEQN (METHOD1$ NEW-FT/ G1-MCS-EQN)

produces SETS user program segment:

```
LDBLK (NEW-FT).
SUBINEQN (G1, G1* STOP$ G4).
REDUCEQN (G1, G1).
SUBINEQN (G1, G1).
REDUCEQN (G1, G1).
FRMBLK (G1-MCS-EQN* ONLY$ G1).
```

This program segment is executed as if it had occurred in the SETS user program in place of the GENFTEQN call. The equations for all of the intermediate events of NEW-FT are loaded into the equation file. An equation for G1 that is a function of primary events and multiple output event G4 is formed and reduced. This equation,

$$G1 = G4*(A + D)$$

replaces the original equation for G1 in the equation file. Substituting into the new equation for G1 without any stops produces an equation for G1 that is a function of

only primary events. After reduction, although no simplifications occur in this case, the equation

$$G1 = (B + C)*(A + D)$$

contains the minimal cut sets for G1. A block, G1-MCS-EQN, containing only the minimal cut set equation for G1, is formed and added to the block file. This top-down approach, determined by parameter METHOD1, produces a minimal cut set equation for G1 in two stages.

Procedure Call

A call of the Generate Fault Tree Equation procedure has one of the forms:

a. GENFTEQN (METHOD1$ ft/ eb).

b. GENFTEQN (METHOD2$ ft/ eb).

c. GENFTEQN (PUNCH$ ‹parameter form a or b›).

d. GENFTEQN (‹any a through c parameter form›*

$q_1,vb_1,q_2,vb_2,\ldots,q_s,vb_s$).

e. GENFTEQN (‹any a through c parameter form›*

EXTREME$ $q_1,vb_1,q_2,vb_2,\ldots,q_s,vb_s$).

Parameter ft indicates the fault tree to be processed. A SETS user program segment is produced which, when executed, generates a minimal cut set equation for every top event of ft.

Parameter eb is the equation block that is formed by the user program segment; it contains minimal cut set equations for top events of ft.

Parameters METHOD1 and METHOD2 determine the method that will be used to generate the minimal cut set equations. METHOD1 specifies a top-down approach and METHOD2 specifies a bottom-up approach.

Parameter PUNCH causes the SETS user program segment to be punched instead of executed. After the segment is punched, execution of the SETS user program is terminated. If parameter PUNCH occurs in a GENFTEQN call, it must be the first parameter in the call.

Each parameter pair, q_k and vb_k, k=1,2,...,s, specifies a computation. Each q_k is either c_k or c_k/l_k which contains computation type c_k and truncation value l_k, if l_k is present. The corresponding vb_k specifies the value block that contains the variable values to be used in the computation specified by q_k. If computations are specified in a GENFTEQN call, TRNTRMVAL calls replace REDUCEQN calls in the SETS user program segment produced. Execution of a user program segment having TRNTRMVAL calls produces a truncated minimal cut set equation for every top event of ft. If parameter EXTREME occurs, only minimal cut sets having the extreme value for every computation are retained.

Processing Achieved

The Generate Fault Tree Equation procedure produces a SETS user program segment which, when executed, generates minimal cut set equations for top events of a fault tree. The right-side expression of a minimal cut set equation for any intermediate event can be generated by forming an expression for the intermediate event that is a function of only primary events and then reducing it to obtain an expression which contains all and only minimal cut sets. In GENFTEQN, two methods are used to generate minimal cut set expressions: top-down and bottom-up.

TOP-DOWN METHOD

An intermediate event expression that is a function of only primary events can be built up by a series of substitutions. The expression begins as the intermediate event, and the first substitution replaces the intermediate event by its equivalent right-side expression. Then, intermediate events introduced by the first or subsequent substitutions are replaced by their right-side expressions until only primary events remain in the expression. The resulting expression is then reduced to obtain minimal cut sets. This is a top-down method because the substitutions begin with the substitution for the intermediate event whose expression is being built up and proceed with substitutions for intermediate events at successively lower levels in the subtree for the event. However, there are more efficient ways to apply the top-down method than to complete all of the substitutions and then try to reduce the expression in one step.

In GENFTEQN, the top-down method is applied in stages. A stage in the top-down development of an expression is created by stopping the substitution process at selected intermediate events. After reduction of the expression at this stage, substitution is reinitiated -- perhaps with stop points that create another stage in the development of the expression. Ultimately, the substitution process is reinitiated without stop points which creates the final stage.

The efficiency of applying the top-down method in stages derives from reducing the expression at each stage in its development. Subsuming terms are removed by applying P + P*Q = P; terms having complementary variables are removed by applying P*/P = /OMEGA, P*/OMEGA = /OMEGA, and P+/OMEGA = P; all except one occurrence of repeated variables in a term are removed by applying P*P = P; and terms are factored to reduce the number of occurrences of common variables. Removing terms and variables from an expression at an interim stage of its development precludes further substitutions for these terms and variables, thus avoiding the subsequent work required to identify and remove terms and variables introduced by these substitutions. Suppose

$$(X + Y)*(X + Z) \qquad (4\text{-}23)$$

occurs at some stage in the development of an expression. Continued development of Exp. 4-23 without reduction involves four two-variable terms: X*X, X*Z, Y*X, and Y*Z. However, reduction of Exp. 4-23 at this stage by P*P = P and P + P*Q = P produces

$$X + Y*Z \qquad (4\text{-}24)$$

which involves only one one-variable term and one two-variable term. The amount of work saved by substituting into and reducing Exp. 4-24 instead of Exp. 4-23 can be enormous.

Expressions from coherent fault trees do not have terms that contain complementary variables. However, removing such terms from expressions developed from noncoherent fault trees by applying P*/P = /OMEGA, P*/OMEGA = /OMEGA, and P+/OMEGA = P can also produce huge savings.

Factoring at an interim stage also contributes to the efficiency of developing an expression in stages. Factoring out a common variable decreases the number of occurrences of the variable in the expression. Subsequent expansion by the distributive law requires less work and saves computer time. Suppose

$$(X*Y) + (X*Z) \qquad (4\text{-}25)$$

occurs at some stage in the development of an expression. Continued development of Exp. 4-25 without factoring means that X must be expanded twice. However, factoring Exp. 4-25 produces

$$X*(Y + Z) \qquad (4\text{-}26)$$

in which X must be expanded only once.

Stop points for stages can be arbitrarily chosen. However, except for deleting terms because they contain complementary variables, each way of removing variables and terms from an expression (i.e., P + P*Q = P, P*P = P, and factoring) involves repeated occurrences of the same variable; and choosing multiple-output intermediate events as stop points tends to exploit this characteristic.

In GENFTEQN, stop points for the top-down development of a minimal cut set equation for top event t are chosen in the following way. Multiple-output intermediate events in S(t) are identified. Then, starting with event t, every branch of S(t) is followed until it terminates with a primary event or until a multiple-output event of S(t) is encountered. Distinct multiple-output events of S(t) encountered in this way are the stop points for the first stage. Second stage stop points are determined by repeating this process for every stop point of the first stage. For each stop point of the first stage, start with the stop point and follow every branch of its subtree until it terminates with a primary event or until it encounters a multiple-output event of S(t). The distinct multiple-output events of S(t) encountered following branches of the subtrees for stage 1 stop points are the stop points for the second stage. Stop points for successive stages are found in a similar way, i.e., by following branches of the subtrees for all stop points of the previous stage. Ultimately, a stage occurs for which none of the subtrees for the stop points of that stage contain multiple-output events of S(t). This produces the final stage which has no stop points.

Fault tree EQN-METHODS, shown in Figure 4.40, has top events G1 and G6. Consider finding stop points for the top-down development of a minimal cut set equation for G1. Events G3, G5, and G10 are multiple-output intermediate events in S(G1). (Event G8, a multiple-output event in EQN-METHODS, is a single-output event in S(G1).) Starting with event G1 and following branches of S(G1), two distinct multiple-output events of S(G1) are encountered: G3 and G5. These events

are stop points for stage 1. Stop points for stage 2 are determined by applying the same procedure, first to G3 and then to G5. Two multiple-output events of S(G1), G5 and G10, are encountered while following branches of S(G3); and event G10 is encountered again while following branches of S(G5). Thus, distinct stop points for stage 2 are G5 and G10. Following branches of S(G10) does not produce any stop points for the third stage, but following branches of S(G5) produces G10 which is the only stop point of stage 3. As before, following S(G10) does not encounter any multiple-output events of S(G1); and the final stage, stage 4, has no stop points. Thus, stop points for the top-down development of a minimal cut set equation for G1 in four stages are as follows:

Stage 1 - G3, G5
Stage 2 - G5, G10
Stage 3 - G10
Stage 4 - none

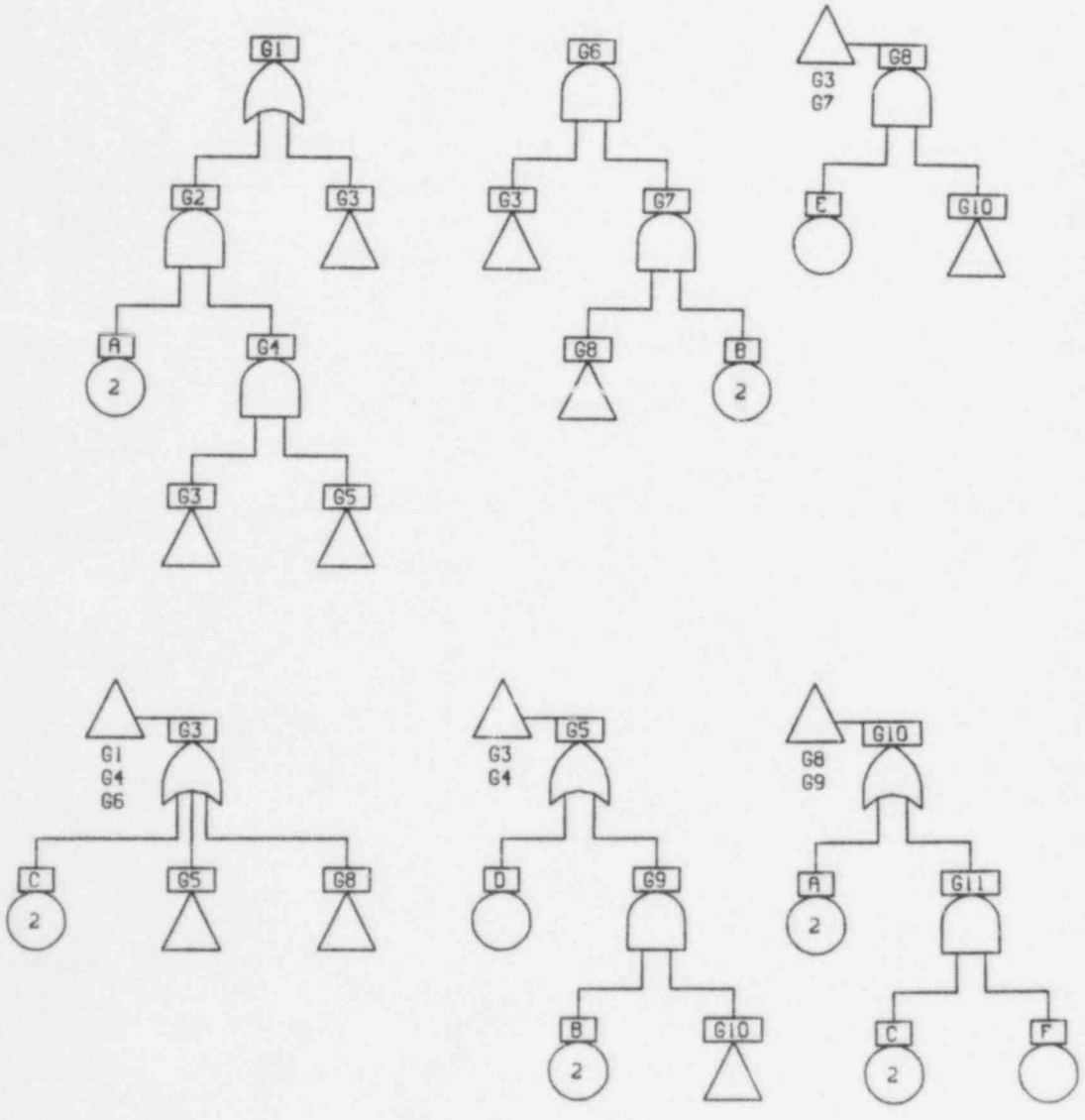


Figure 4.40. Fault tree EQN-METHODS.

One way to determine stop points for the top-down method has been described. This is the way stop points are chosen in GENFTEQN, and it is usually much more efficient than applying the method in one step. However, difficult problems that cannot be solved using stages chosen this way, sometimes yield to a different choice of stages.

The SETS user program segment produced by GENFTEQN for the top-down method begins with a call of LDBLK which loads the intermediate event equations of ft into the equation file. Then, for each top event of ft, the program segment contains a series of statements which generates a minimal cut set equation for the top event in stages. Each stage requires a SUBINEQN call and a REDUCEQN call. Stop points for each stage appear in a stop option in the SUBINEQN call for that stage. Since the final stage has no stop points, the corresponding SUBINEQN call has no stop option. The final statement in the segment is a FRMBLK call that creates equation block eb and enters it into the block file. An only option in the FRMBLK call lists top events of ft. Thus, block eb contains minimal cut set equations for all and only top events of ft.

Application of the top-down method to fault tree EQN-METHODS in Figure 4.40, assuming eb is MCS-TOP-DWN, is achieved by the GENFTEQN call

```
GENFTEQN (METHOD1$ EQN-METHODS/ MCS-TOP-DWN)
```

which produces the following SETS user program segment:

```
LDBLK (EQN-METHODS).

SUBINEQN (G1, G1* STOP$ G3, G5).
REDUCEQN (G1, G1).
SUBINEQN (G1, G1* STOP$ G5, G10).
REDUCEQN (G1, G1).
SUBINEQN (G1, G1* STOP$ G10).
REDUCEQN (G1, G1).
SUBINEQN (G1, G1).
REDUCEQN (G1, G1).

SUBINEQN (G6, G6* STOP$ G8, G10).
REDUCEQN (G6, G6).
SUBINEQN (G6, G6* STOP$ G10).
REDUCEQN (G6, G6).
SUBINEQN (G6, G6).
REDUCEQN (G6, G6).

FRMBLK (MCS-TOP-DWN* ONLY$ G1, G6).
```

Upon execution, this SETS user program segment generates the minimal cut set equation for top event G1 in four stages and the minimal cut set equation for G6 in three stages and saves both equations in equation block MCS-TOP-DWN. After the segment is executed, minimal cut set equations for G1 and G6 are also in the equation file.

The top-down method can be used to generate truncated minimal cut set equations. Computation parameters

$$q_1, vb_1, q_2, vb_2, \ldots, q_s, vb_s$$

in a GENFTEQN call, either by themselves (form d) or preceded by EXTREME (form e), cause TRNTRMVAL calls to replace REDUCEQN calls in the SETS user program segment produced by GENFTEQN. Computation parameters specified in the GENFTEQN call, and EXTREME too if it occurs, are duplicated in each TRNTRMVAL call. This change alone, however, is not enough to produce a user program segment that generates truncated expressions.

Values cannot be computed for the terms of an expression unless every variable in the expression has a value or variables that do not have values are specified in an option that excludes them from the computations. An excluded variable acts like an identity element; it does not change the values of terms in which it occurs. However, truncation at interim stages in the development of an expression is still meaningful even though terms are deleted if a partial term value exceeds its corresponding truncation value. Since restricted variable values ensure every term computation is either monotone increasing or monotone decreasing, variables excluded from the computations at some stage cannot have a value, either assigned or computed, that allows a partial term value to exceed the term truncation value while the term value does not.

Stop points are intermediate events that determine a stage in the development of an expression. Ordinarily, intermediate events are not assigned values and stop points must, therefore, be excluded from computations specified in the user program segment. Consequently, the TRNTRMVAL call corresponding to each SUBINEQN call with a stop option must have an except noncomplement option which contains the same variables as the stop option and excludes them from the computations.

Suppose truncated minimal cut set equations for top events G1 and G6 in fault tree EQN-METHODS are to be obtained using the top-down method. Suppose also that term probabilities are to be computed using value block VAR-PROBS; terms having a probability less than 2.5×10^{-5} are to be discarded; and the truncated equations are to be saved in block TRN-MCS-TOP-DWN. The GENFTEQN call would look like

```
GENFTEQN (METHOD1$ EQN-METHODS/ TRN-MCS-TOP-DWN*
          PROBABILITY/ 2.5E-5, VAR-PROBS).
```

and it would produce SETS user program segment:

```
LDBLK (EQN-METHODS).

SUBINEQN (G1, G1* STOP$ G3, G5).
TRNTRMVAL (PROBABILITY/ 2.5E-5, VAR-PROBS* G1, G1*
           EXCEPTNONCMP$ G3, G5).
SUBINEQN (G1, G1* STOP$ G5, G10).
TRNTRMVAL (PROBABILITY/ 2.5E-5, VAR-PROBS* G1, G1*
           EXCEPTNONCMP$ G5, G10).
```

```
SUBINEQN (G1, G1* STOP$ G10).
TRNTRMVAL (PROBABILITY/ 2.5E-5, VAR-PROBS* G1, G1*
        EXCEPTNONCMP$ G10).
SUBINEQN (G1, G1).
TRNTRMVAL (PROBABILITY/ 2.5E-5, VAR-PROBS* G1, G1).

        SUBINEQN (G6, G6* STOP$ G8, G10).
TRNTRMVAL (PROBABILITY/ 2.5E-5, VAR-PROBS* G6, G6*
        EXCEPTNONCMP$ G8, G10).
SUBINEQN (G6, G6* STOP$ G10).
TRNTRMVAL (PROBABILITY/ 2.5E-5, VAR-PROBS* G6, G6*
        EXCEPTNONCMP$ G10).
SUBINEQN (G6, G6).
TRNTRMVAL (PROBABILITY/ 2.5E-5, VAR-PROBS* G6, G6).

FRMBLK (TRN-MCS-TOP-DWN* ONLY$ G1, G6).
```

BOTTOM-UP METHOD

An intermediate event expression that is a function of only primary events can also be built up by beginning at the bottom of the subtree for the intermediate event and generating an expression that is a function of only primary events for every intermediate event in the subtree. Ultimately, an expression having only primary events is produced for the top event of the subtree. This expression is then reduced to obtain minimal cut sets. This is a bottom-up method because it begins with intermediate events at the bottom of the subtree and progresses through successively higher levels of the subtree to produce expressions that are functions of only primary events. Like the top-down method, there are more efficient ways to apply the bottom-up method than to complete the formation of the expression for the top event of the subtree and then try to reduce it in one step.

In the bottom-up method in GENFTEQN, expressions that are functions of only primary events are not generated for every intermediate event in the subtree. Instead, expressions are generated only for multiple-output intermediate events, certain intermediate events defined by AND gates, and top events. Also, each expression is reduced as it is generated.

Expressions for multiple-output intermediate events are generated first and in groups. Each group is determined by identifying all multiple-output intermediate events for which equations have not yet been generated and, from among them, choosing those that are not a function of any of the others. After the expressions in a group have been formed and reduced, all of the equations in the equation file are saved in an equation block that is added to the block file. The process of choosing the next group continues until expressions for multiple-output intermediate events in the final group have been formed, reduced, and saved.

After the expressions for the multiple-output intermediate events have been generated, expressions for certain intermediate events defined by AND gates are generated. First, the subtree determined by all of the multiple-output intermediate events is identified, and then, single-output intermediate events defined by AND gates which are not in the subtree are identified. Expressions for these events are also generated in groups. Each group is determined by identifying

the single-output intermediate events defined by AND gates for which equations have not yet been generated and, from among them, choosing those that are not a function of any of the others. After expressions have been formed, reduced, and saved for the events in the final group of intermediate events defined by AND gates, expressions for top events of the fault tree are generated.

The efficiency of applying the bottom-up method in stages derives from reducing each generated expression before it is substituted into the expression for an intermediate event at a higher level in the subtree. Reduction removes terms and variables from generated expressions and precludes further substitutions for them. This avoids the subsequent work that would be required to identify and remove terms and variables introduced by these substitutions.

Application of the bottom-up method to fault tree EQN-METHODS in Figure 4.40, assuming eb is MCS-BOT-UP, is achieved by the GENFTEQN call

```
GENFTEQN (METHOD2$ EQN-METHODS/ MCS-BOT-UP)
```

which produces the following SETS user program segment:

```
LDBLK (EQN-METHODS).

SUBINEQN (G10, G10).
REDUCEQN (G10, G10).
DLTBLK (MCS-BOT-UP).
FRMBLK (MCS-BOT-UP).

SUBINEQN (G5, G5).
REDUCEQN (G5, G5).
SUBINEQN (G8, G8).
REDUCEQN (G8, G8).
DLTBLK (MCS-BOT-UP).
FRMBLK (MCS-BOT-UP).

SUBINEQN (G3, G3).
REDUCEQN (G3, G3).
DLTBLK (MCS-BOT-UP).
FRMBLK (MCS-BOT-UP).

SUBINEQN (G4, G4).
REDUCEQN (G4, G4).
SUBINEQN (G7, G7).
REDUCEQN (G7, G7).
DLTBLK (MCS-BOT-UP).
FRMBLK (MCS-BOT-UP).

SUBINEQN (G2, G2).
REDUCEQN (G2, G2).
DLTBLK (MCS-BOT-UP).
FRMBLK (MCS-BOT-UP).
```

```
SUBINEQN (G1, G1).
REDUCEQN (G1, G1).
SUBINEQN (G6, G6).
REDUCEQN (G6, G6).
DLTBLK (MCS-BOT-UP).
FRMBLK (MCS-BOT-UP* ONLY$ G1, G6).
```

Upon execution, this SETS user program segment generates equations for multiple-output events in three groups: G10; G5 and G8; and G3. Equations for intermediate events defined by AND gates are generated in two groups: G4 and G7; and G2. Finally, minimal cut set equations for top events G1 and G6 are generated and saved in block MCS-BOT-UP. After the segment is executed, minimal cut set equations for G1 and G6 are also in the equation file.

Like the top-down method, the bottom-up method can be used to generate truncated minimal cut set equations. A TRNTRMVAL call replaces each REDUCEQN call in the user program segment. The computation parameters in the TRNTRMVAL calls are duplicates of the computation parameters in the GENFTEQN call. Unlike the top-down method, except options do not have to be used in the bottom-up method because each generated equation is a function of only primary events and primary events have values.

Suppose truncated minimal cut set equations for top events G1 and G6 in fault tree EQN-METHODS are to be obtained using the bottom-up method. Value block VAR-PROBS is to be used to compute term probability, and terms having a probability less than 2.5×10^{-5} are to be discarded. The truncated equations are to be saved in block TRN-MCS-BOT-UP. The GENFTEQN call would look like

```
GENFTEQN (METHOD2$ EQN-METHODS/ TRN-MCS-BOT-UP*
          PROBABILITY/ 2.5E-5, VAR-PROBS).
```

and it would produce SETS user program segment:

```
LDBLK (EQN-METHODS).

SUBINEQN (G10, G10).
TRNTRMVAL (PROBABILITY/ 2.5E-5, VAR-PROBS* G10, G10).
DLTBLK (TRN-MCS-BOT-UP).
FRMBLK (TRN-MCS-BOT-UP).

SUBINEQN (G5, G5).
TRNTRMVAL (PROBABILITY/ 2.5E-5, VAR-PROBS* G5, G5).
SUBINEQN (G8, G8).
TRNTRMVAL (PROBABILITY/ 2.5E-5, VAR-PROBS* G8, G8).
DLTBLK (TRN-MCS-BOT-UP).
FRMBLK (TRN-MCS-BOT-UP).

SUBINEQN (G3, G3).
TRNTRMVAL (PROBABILITY/ 2.5E-5, VAR-PROBS* G3, G3).
DLTBLK (TRN-MCS-BOT-UP).
FRMBLK (TRN-MCS-BOT-UP).
```

```
SUBINEQN (G4, G4).
TRNTRMVAL (PROBABILITY/ 2.5E-5, VAR-PROBS* G4, G4).
SUBINEQN (G7, G7).
TRNTRMVAL (PROBABILITY/ 2.5E-5, VAR-PROBS* G7, G7).
DLTBLK (TRN-MCS-BOT-UP).
FRMBLK (TRN-MCS-BOT-UP).

SUBINEQN (G2, G2).
TRNTRMVAL (PROBABILITY/ 2.5E-5, VAR-PROBS* G2, G2).
DLTBLK (TRN-MCS-BOT-UP).
FRMBLK (TRN-MCS-BOT-UP).

SUBINEQN (G1, G1).
TRNTRMVAL (PROBABILITY/ 2.5E-5, VAR-PROBS* G1, G1).
SUBINEQN (G6, G6).
TRNTRMVAL (PROBABILITY/ 2.5E-5, VAR-PROBS* G6, G6).
DLTBLK (TRN-MCS-BOT-UP).
FRMBLK (TRN-MCS-BOT-UP* ONLY$ G1, G6).
```

Regardless of the method specified in a GENFTEQN call and whether or not truncation parameters occur, the SETS user program segment produced by GENFTEQN is executed after it is produced unless parameter PUNCH occurs in the call. It is as if the segment had occurred in the SETS user program in place of the GENFTEQN call. If parameter PUNCH occurs in the call, the user program segment is assigned to the system's PUNCH file and execution of the SETS user program is terminated.

Printed Output

The printed output for GENFTEQN consists of a list of top events of fault tree ft and a listing of the SETS user program segment produced by GENFTEQN. The list of top events, T_k, k=1,2,...,s, has the form:

TOP EVENTS OF ft

1. T_1
2. T_2
. .
. .
. .
s. T_s

The listing of the SETS user program segment is preceded by the message:

SETS USER PROGRAM SEGMENT PRODUCED BY GENFTEQN

Example

The Figure 6 fault tree from Reference 2 (pp. 33-34) is used to show various ways GENFTEQN can be used to find minimal cut set equations. The fault tree is reproduced in Figure 4.41. The fault tree has one top event, G1, which has 5,769,050 cut sets that reduce to 1,053 minimal cut sets each having from two to twelve variables.

This page intentionally left blank.

DOCUMENT/ PAGE PULLED

ANO. 8508090642

NO. OF PAGES 1

REASON:

☐ PAGE ILLEGIBLE:

☐ HARD COPY FILED AT: PDR CF

OTHER _____

☐ BETTER COPY REQUESTED ON ____/____/____

☐ PAGE TOO LARGE TO FILM:

☒ HARD COPY FILED AT: PDR (CF)

OTHER _____

☒ FILMED ON APERTURE CARD NO 8508090642-02

This example is not intended to find the most efficient way to obtain minimal cut sets for G1, nor is it intended to provide a basis for comparing the efficiency of the ways that are shown. Ordinarily, a fault tree that is difficult to solve, is separated into its stem and collection of LSIP subtrees using FRMNEWFT. Then, the stem and LSIP subtrees are solved separately using the most efficient approach possible on each of these fault trees. The example simply illustrates the top-down and bottom-up methods applied to fault tree FIG-4-41-FT without separating it into its stem and LSIP subtrees. Both methods are shown with and without truncation.

Consider SETS user program:

```
PROGRAM$ GENFTEQN-EX.
   RDFT (FIG-4-41-FT).

COMMENT$  APPLY TOP-DOWN METHOD TO FIG-4-41-FT.$
   GENFTEQN (METHOD1$ FIG-4-41-FT/ MCS-METH-1).

COMMENT$  APPLY BOTTOM-UP METHOD TO FIG-4-41-FT.$
   DLTEQN.
   GENFTEQN (METHOD2$ FIG-4-41-FT/ MCS-METH-2).

COMMENT$  TRUNCATE MCS EQUATION FOR G1 BY DELETING TERMS HAVING
          FAILURE PROBABILITY LESS THAN 1.0E-6 OR REPAIR COST
          GREATER THAN 450 THOUSAND DOLLARS.$
   DLTEQN.
   RDVALBLK (FAIL-PROB, REPAIR-COST).
   LDBLK (MCS-METH-1).
   TRNTRMVAL (PROBABILITY/1E-6, FAIL-PROB, SUM/450, REPAIR-COST*
              G1, TRN-G1-MCS).
   COMTRMVAL (PROBABILITY, FAIL-PROB, SUM, REPAIR-COST* TRN-G1-MCS).

COMMENT$  TRUNCATE WHILE APPLYING TOP-DOWN METHOD TO FIG-4-41-FT.$
   DLTEQN.
   GENFTEQN (METHOD1$ FIG-4-41-FT/ TRN-MCS-BLK-1*
             PROBABILITY/1E-6, FAIL-PROB, SUM/450, REPAIR-COST).
   COMTRMVAL (PROBABILITY, FAIL-PROB, SUM, REPAIR-COST* G1).

COMMENT$  TRUNCATE WHILE APPLYING BOTTOM-UP METHOD TO FIG-4-41-FT.$
   DLTEQN.
   GENFTEQN (METHOD2$ FIG-4-41-FT/ TRN-MCS-BLK-2*
             PROBABILITY/1E-6, FAIL-PROB, SUM/450, REPAIR-COST).
   COMTRMVAL (PROBABILITY, FAIL-PROB, SUM, REPAIR-COST* G1).
```

The equation file is empty when execution of GENFTEQN-EX begins. The first statement in the SETS user program is a RDFT call that reads fault tree FIG-4-41-FT and enters it into the block file. The second statement is a GENFTEQN call that uses the top-down method to generate a minimal cut set equation for G1 and save it in block MCS-METH-1. The printed output produced by the GENFTEQN call is as follows:

```
GENFTEQN (METHOD1$ FIG-4-41-FT/ MCS-METH-1).

          TOP EVENTS OF FIG-4-41-FT

               1.  G1

          SETS USER PROGRAM SEGMENT PRODUCED BY GENFTEQN

               LDBLK (FIG-4-41-FT).
               SUBINEQN (G1, G1* STOP$ G8, G10, G14).
               REDUCEQN (G1, G1).
               SUBINEQN (G1, G1* STOP$ G32, G14).
               REDUCEQN (G1, G1).
```

```
SUBINEQN (G1, G1* STOP$ G32).
REDUCEQN (G1, G1).
SUBINEQN (G1, G1).
REDUCEQN (G1, G1).
FRMBLK (MCS-METH-1* ONLY$ G1).

STATEMENT EXECUTION REQUIRED      .078 SECONDS FOR GENFTEQN
```

Execution of the user program segment produced by GENFTEQN generates a minimal cut set equation for G1. Output for the simplification part of the final REDUCEQN call in the segment shows an accounting of the 1,053 minimal cut sets:

```
TERMS RETAINED BY SIMPLIFICATION
      1 TERMS CONTAIN  2 VARIABLES
      2 TERMS CONTAIN  3 VARIABLES
     11 TERMS CONTAIN  4 VARIABLES
     11 TERMS CONTAIN  5 VARIABLES
     38 TERMS CONTAIN  6 VARIABLES
    146 TERMS CONTAIN  7 VARIABLES
    290 TERMS CONTAIN  8 VARIABLES
    312 TERMS CONTAIN  9 VARIABLES
    183 TERMS CONTAIN 10 VARIABLES
     53 TERMS CONTAIN 11 VARIABLES
      6 TERMS CONTAIN 12 VARIABLES
  TOTAL TERMS RETAINED     1053.
SIMPLIFICATION TOOK      .212 SECONDS.
```

The next statement in the SETS user program is a call of DLTEQN that deletes every equation from the equation file. Then, another call of GENFTEQN generates a minimal cut set equation for G1 using the bottom-up method. The output produced by this call of GENFTEQN is as follows:

```
GENFTEQN (METHOD2$ FIG-4-41-FT/ MCS-METH-2).

          TOP EVENTS OF FIG-4-41-FT

               1.  G1

          SETS USER PROGRAM SEGMENT PRODUCED BY GENFTEQN

               LOBLK (FIG-4-41-FT).
               SUBINEQN (G32, G32).
               REDUCEQN (G32, G32).
               DLTBLK (MCS-METH-2).
               FRMBLK (MCS-METH-2).
               SUBINEQN (G14, G14).
               REDUCEQN (G14, G14).
               DLTBLK (MCS-METH-2).
               FRMBLK (MCS-METH-2).
               SUBINEQN (G8, G8).
               REDUCEQN (G8, G8).
               SUBINEQN (G10, G10).
               REDUCEQN (G10, G10).
               DLTBLK (MCS-METH-2).
               FRMBLK (MCS-METH-2).
               SUBINEQN (G4, G4).
               REDUCEQN (G4, G4).
               SUBINEQN (G6, G6).
               REDUCEQN (G6, G6).
               SUBINEQN (G17, G17).
               REDUCEQN (G17, G17).
               DLTBLK (MCS-METH-2).
               FRMBLK (MCS-METH-2).
               SUBINEQN (G13, G13).
               REDUCEQN (G13, G13).
               DLTBLK (MCS-METH-2).
               FRMBLK (MCS-METH-2).
```

```
SUBINEQN (G5, G5).
REDUCEQN (G5, G5).
DLTBLK (MCS-METH-2).
FRMBLK (MCS-METH-2).
SUBINEQN (G1, G1).
REDUCEQN (G1, G1).
DLTBLK (MCS-METH-2).
FRMBLK (MCS-METH-2* ONLY$ G1).

STATEMENT EXECUTION REQUIRED      .089 SECONDS FOR GENFTEQN
```

Execution of this program segment also generates a minimal cut set equation for G1. Simplification output from the final REDUCEQN call in the segment confirms the 1,053 minimal cut sets:

```
TERMS RETAINED BY SIMPLIFICATION
      1 TERMS CONTAIN  2 VARIABLES
      2 TERMS CONTAIN  3 VARIABLES
     11 TERMS CONTAIN  4 VARIABLES
     11 TERMS CONTAIN  5 VARIABLES
     38 TERMS CONTAIN  6 VARIABLES
    146 TERMS CONTAIN  7 VARIABLES
    290 TERMS CONTAIN  8 VARIABLES
    312 TERMS CONTAIN  9 VARIABLES
    183 TERMS CONTAIN 10 VARIABLES
     53 TERMS CONTAIN 11 VARIABLES
      6 TERMS CONTAIN 12 VARIABLES
  TOTAL TERMS RETAINED     1053.
SIMPLIFICATION TOOK      .379 SECONDS.
```

The next statement is a call of DLTEQN which deletes every equation from the equation file. Then, a call of RDVALBLK reads failure probabilities and repair costs for primary events in fault tree FIG-4-41-FT, and enters value blocks FAIL-PROB and REPAIR-COST into the value block file. The printed output produced by the RDVALBLK call is as follows:

```
RDVALBLK (FAIL-PROB, REPAIR-COST).

        VALUE BLOCK$ FAIL-PROB.
        CREATE VALUE BLOCK FAIL-PROB
           .1 E-3$  E7, E11, E24$
           .1 E-2$  E8, E16$
           .5 E-2$  E20$
           .75 E-2$  E10, E17, E34$
           .1 E-1$  E19, E33$
           .25 E-1$  E15, E22, E28$
           .5 E-1$  E9, E12, E14, E21, E23, E26, E30, E32$
           .7 E-1$  E3, E5, E13, E29, E31$
           .75 E-1$  E2, E18$
           .8 E-1$  E1, E6, E27$
           .9 E-1$  E4, E25$

  VALUE BLOCK FAIL-PROB
  HAS BEEN ADDED TO THE VALUE BLOCK FILE
```

```
          VALUE BLOCK$ REPAIR-COST.
          CREATE VALUE BLOCK REPAIR-COST
             25$   E3, E9, E24, E32$
             50$   E1, E7, E11, E15, E21, E29$
             80$   E5, E8, E13, E19, E26, E31, E34$
            100$   E6, E14, E25, E28, E30, E33$
            140$   E2, E10, E23$
            210$   E4, E12, E17, E27$
            300$   E16, E18, E20, E22$

     VALUE BLOCK REPAIR-COST
     HAS BEEN ADDED TO THE VALUE BLOCK FILE

     STATEMENT EXECUTION REQUIRED      .029 SECONDS FOR RDVALBLK
```

The next statement in GENFTEQN-EX is a call of LDBLK that loads a minimal cut set equation for G1 into the equation file. Then, a TRNTRMVAL call identifies six of the 1,053 minimal cut sets of G1 whose failure probability is $\geq 1 \times 10^{-6}$ and repair cost is $\leq$ 450,000 dollars. A call of COMTRMVAL prints the six terms so that results obtained later can be compared to them. The printed output produced by TRNTRMVAL and part of the output produced by COMTRMVAL is as follows:

```
TRNTRMVAL (PROBABILITY/ 1E-6, FAIL-PROB, SUM/ 450, REPAIR-COST* G1,
          TRN-G1-MCS).

               THE MAXIMUM NUMBER OF TERMS THAT CAN BE
               GENERATED BY EXPANSION IS       1053.

               TERMS GENERATED BY EXPANSION
                     1 TERMS CONTAIN  2 VARIABLES
                     1 TERMS CONTAIN  3 VARIABLES
                     4 TERMS CONTAIN  4 VARIABLES
               TOTAL TERMS GENERATED        6.

               THERE WERE TERMS DELETED BECAUSE OF
               TRUNCATION VALUES

               THE MAXIMUM TERM VALUE FOR COMPUTATION 1 IS   4.000000000000E-03
                    (THE SUM OF THE TERM VALUES IS   4.445200000000E-03)

               THE MINIMUM TERM VALUE FOR COMPUTATION 2 IS   7.500000000000E+01

          EXPANSION TOOK      .063 SECONDS.

               TERMS RETAINED BY SIMPLIFICATION
                     1 TERMS CONTAIN  2 VARIABLES
                     1 TERMS CONTAIN  3 VARIABLES
                     4 TERMS CONTAIN  4 VARIABLES
               TOTAL TERMS RETAINED        6.
          SIMPLIFICATION TOOK      .003 SECONDS.

          FACTORIZATION TOOK      .024 SECONDS.

     STATEMENT EXECUTION REQUIRED      .095 SECONDS FOR TRNTRMVAL
```

•

•

•

```
 TERM      PROB.       SUM
NUMBER   OF TERM     OF TERM

                                  TRN-G1-MCS =

   1   4.0000E-03 7.5000E+01    E1 * E9 +

   2   4.2000E-04 2.7000E+02    E1 * E2 * E13 +

   3   1.0000E-05 3.3000E+02    E1 * E14 * E30 * E26 +

   4   7.5000E-06 3.4000E+02    E1 * E2 * E14 * E15 +

   5   5.0000E-06 3.3000E+02    E1 * E30 * E26 * E28 +

   6   2.7000E-06 4.4000E+02    E1 * E4 * E30 * E34

      THE MAXIMUM TERM VALUE FOR COMPUTATION 1 IS   4.000000000000E-03
          (THE SUM OF THE TERM VALUES IS   4.445200000000E-03)

      THE MINIMUM TERM VALUE FOR COMPUTATION 2 IS   7.500000000000E+01

STATEMENT EXECUTION REQUIRED      .024 SECONDS FOR COMTRMVAL
```

The next statement in the SETS user program is a DLTEQN call that deletes every equation from the equation file. Then, a call of GENFTEQN, which uses the top-down method and truncates the equation for G1 as it is generated, produces the six minimal cut sets that do not exceed the truncation values. A call of COMTRMVAL confirms the six terms are the same as those obtained earlier by generating all minimal cut sets and then truncating. The printed output produced by GENFTEQN and part of the COMTRMVAL output is as follows:

```
GENFTEQN (METHOD1$ FIG-4-41-FT/ TRN-MCS-BLK-1* PROBABILITY/ 1E-6,
         FAIL-PROB, SUM/ 450, REPAIR-COST).

          TOP EVENTS OF FIG-4-41-FT

               1.  G1

          SETS USER PROGRAM SEGMENT PRODUCED BY GENFTEQN

               LDBLK (FIG-4-41-FT).
               SUBINEQN (G1, G1* STOP$ G8, G10, G14).
               TRNTRMVAL (PROBABILITY/ 1E-6, FAIL-PROB, SUM/ 450,
                          REPAIR-COST* G1, G1* EXCEPTNONCMP$ G8, G10,
                          G14).
               SUBINEQN (G1, G1* STOP$ G32, G14).
               TRNTRMVAL (PROBABILITY/ 1E-6, FAIL-PROB, SUM/ 450,
                          REPAIR-COST* G1, G1* EXCEPTNONCMP$ G32,
                          G14).
               SUBINEQN (G1, G1* STOP$ G32).
               TRNTRMVAL (PROBABILITY/ 1E-6, FAIL-PROB, SUM/ 450,
                          REPAIR-COST* G1, G1* EXCEPTNONCMP$ G32).
               SUBINEQN (G1, G1).
               TRNTRMVAL (PROBABILITY/ 1E-6, FAIL-PROB, SUM/ 450,
                          REPAIR-COST* G1, G1).
               FRMBLK (TRN-MCS-BLK-1* ONLY$ G1).

STATEMENT EXECUTION REQUIRED      .084 SECONDS FOR GENFTEQN
```

•

•

•

```
 TERM      PROB.        SUM
NUMBER   OF TERM     OF TERM

                                  G1 =

   1   4.0000E-03 7.5000E+01    E1 * E9 +

   2   4.2000E-04 2.7000E+02    E1 * E2 * E13 +

   3   1.0000E-05 3.3000E+02    E1 * E14 * E30 * E26 +

   4   7.5000E-06 3.4000E+02    E1 * E2 * E14 * E15 +

   5   5.0000E-06 3.3000E+02    E1 * E30 * E26 * E28 +

   6   2.7000E-06 4.4000E+02    E1 * E4 * E30 * E34

     THE MAXIMUM TERM VALUE FOR COMPUTATION 1 IS   4.000000000000E-03
         (THE SUM OF THE TERM VALUES IS   4.445200000000E-03)

     THE MINIMUM TERM VALUE FOR COMPUTATION 2 IS   7.500000000000E+01

STATEMENT EXECUTION REQUIRED      .024 SECONDS FOR COMTRMVAL
```

The next statement is another call of DLTEQN that deletes every equation from the equation file. The final two statements are calls of GENFTEQN and COMTRMVAL. This time, the GENFTEQN call uses the bottom-up method and truncates as the equation for G1 is generated to produce the six minimal cut sets that do not exceed the truncation values. Again, the COMTRMVAL call confirms the six terms obtained are correct. The output produced by GENFTEQN and part of the output for COMTRMVAL is as follows:

```
GENFTEQN (METHOD2$ FIG-4-41-FT/ TRN-MCS-BLK-2* PROBABILITY/ 1E-6,
         FAIL-PROB, SUM/ 450, REPAIR-COST).

     TOP EVENTS OF FIG-4-41-FT

         1.  G1

     SETS USER PROGRAM SEGMENT PRODUCED BY GENFTEQN

          LDBLK (FIG-4-41-FT).
          SUBINEQN (G32, G32).
          TRNTRMVAL (PROBABILITY/ 1E-6, FAIL-PROB, SUM/ 450,
                    REPAIR-COST* G32, G32).
          DLTBLK (TRN-MCS-BLK-2).
          FRMBLK (TRN-MCS-BLK-2).
          SUBINEQN (G14, G14).
          TRNTRMVAL (PROBABILITY/ 1E-6, FAIL-PROB, SUM/ 450,
                    REPAIR-COST* G14, G14).
          DLTBLK (TRN-MCS-BLK-2).
          FRMBLK (TRN-MCS-BLK-2).
          SUBINEQN (G8, G8).
          TRNTRMVAL (PROBABILITY/ 1E-6, FAIL-PROB, SUM/ 450,
                    REPAIR-COST* G8, G8).
          SUBINEQN (G10, G10).
          TRNTRMVAL (PROBABILITY/ 1E-6, FAIL-PROB, SUM/ 450,
                    REPAIR-COST* G10, G10).
          DLTBLK (TRN-MCS-BLK-2).
          FRMBLK (TRN-MCS-BLK-2).
          SUBINEQN (G4, G4).
          TRNTRMVAL (PROBABILITY/ 1E-6, FAIL-PROB, SUM/ 450,
                    REPAIR-COST* G4, G4).
          SUBINEQN (G6, G6).
          TRNTRMVAL (PROBABILITY/ 1E-6, FAIL-PROB, SUM/ 450,
                    REPAIR-COST* G6, G6).
          SUBINEQN (G17, G17).
          TRNTRMVAL (PROBABILITY/ 1E-6, FAIL-PROB, SUM/ 450,
                    REPAIR-COST* G17, G17).
          DLTBLK (TRN-MCS-BLK-2).
          FRMBLK (TRN-MCS-BLK-2).
```

```
SUBINEQN (G13, G13).
TRNTRMVAL (PROBABILITY/ 1E-6, FAIL-PROB, SUM/ 450,
          REPAIR-COST* G13, G13).
DLTBLK (TRN-MCS-BLK-2).
FRMBLK (TRN-MCS-BLK-2).
SUBINEQN (G5, G5).
TRNTRMVAL (PROBABILITY/ 1E-6, FAIL-PROB, SUM/ 450,
          REPAIR-COST* G5, G5).
DLTBLK (TRN-MCS-BLK-2).
FRMBLK (TRN-MCS-BLK-2).
SUBINEQN (G1, G1).
TRNTRMVAL (PROBABILITY/ 1E-6, FAIL-PROB, SUM/ 450,
          REPAIR-COST* G1, G1).
DLTBLK (TRN-MCS-BLK-2).
FRMBLK (TRN-MCS-BLK-2* ONLY$ G1).

STATEMENT EXECUTION REQUIRED      .097 SECONDS FOR GENFTEQN
```

•

•

•

TERM NUMBER	PROB. OF TERM	SUM OF TERM	G1 =
1	4.0000E-03	7.5000E+01	E1 * E9 +
2	4.2000E-04	2.7000E+02	E1 * E2 * E13 +
3	1.0000E-05	3.3000E+02	E1 * E14 * E30 * E26 +
4	7.5000E-06	3.4000E+02	E1 * E2 * E14 * E15 +
5	5.0000E-06	3.3000E+02	E1 * E30 * E26 * E28 +
6	2.7000E-06	4.4000E+02	E1 * E4 * E30 * E34

```
  THE MAXIMUM TERM VALUE FOR COMPUTATION 1 IS   4.000000000000E-03
      (THE SUM OF THE TERM VALUES IS   4.445200000000E-03)

  THE MINIMUM TERM VALUE FOR COMPUTATION 2 IS   7.500000000000E+01

STATEMENT EXECUTION REQUIRED      .024 SECONDS FOR COMTRMVAL
```

This page intentionally left blank.

REFERENCES

1. R. B. Worrell, Set Equation Transformation System (SETS), Sandia National Laboratories, Albuquerque, NM, SLA-73-0028A, July 1973.

2. R. B. Worrell and D. W. Stack, A SETS User's Manual for the Fault Tree Analyst, Sandia National Laboratories, Albuquerque, NM, NUREG/CR-0465, SAND77-2051, November 1978.

3. U.S. Nuclear Regulatory Commission, Reactor Safety Study - An Assessment of Accident Risks in U.S. Commercial Nuclear Power Plants, Executive Summary, WASH-1400, NUREG-75/014, October 1975.

4. D. R. Gallup, G. B. Varnado, and S. L. Daniel, Generic Modeling Approach for Light Water Reactor Systems Analysis, Sandia National Laboratories, Albuquerque, NM, NUREG/CR-3121, SAND83-0013, May 1983.

5. D. W. Stack and M. S. Hill, A SETS User's Manual for Vital Area Analysis, Sandia National Laboratories, Albuquerque, NM, NUREG/CR-3134, SAND83-0074, April 1984.

6. R. B. Worrell and D. W. Stack, Common-Cause Analysis Using SETS, Sandia National Laboratories, Albuquerque, NM, SAND77-1832, December 1977.

7. R. B. Worrell and D. W. Stack, "A Boolean Approach to Common Cause Analysis," 1980 Proceedings Annual Reliability and Maintainability Symposium, San Francisco, CA, January 1980.

8. D. D. Carlson, et. al., Interim Reliability Evaluation Program Procedures Guide, Sandia National Laboratories, Albuquerque, NM, NUREG/CR-2728, SAND82-1100, January 1983.

9. G. B. Varnado, W. H. Horton, and P. R. Lobner, Modular Fault Tree Analysis Procedures Guide, Volume 1 - Main Report, Sandia National Laboratories, Albuquerque, NM, NUREG/CR-3268/1 of 4, SAND83-0963/ 1 of 4, August 1983.

10. D. W. Stack, A SETS User's Manual for Accident Sequence Analysis, Sandia National Laboratories, Albuquerque, NM, NUREG/CR-3547, SAND83-2238, January 1984.

11. B. L. Hulme and R. B. Worrell, "A Prime Implicant Algorithm with Factoring," pp. 1129-1131, IEEE Transactions on Computers, Vol. C-24, No. 11, November 1975.

12. R. B. Worrell, D. W. Stack, and B. L. Hulme, "Prime Implicants of Noncoherent Fault Trees," pp. 98-100, IEEE Transactions on Reliability, Vol. R-30, No. 2, June 1981.

13. L. H. Goldstein and R. B. Worrell, Combinational Circuit Design Verification Using SETS, Sandia National Laboratories, Albuquerque, NM, SAND78-1933, October 1978.

14. B. L. Hulme, A. W. Shiver, and P. J. Slater, Computing Minimum Cost Fire Protection, Sandia National Laboratories, Albuquerque, NM, SAND82-0809, June 1982.

15. B. L. Hulme, A. W. Shiver, and P. J. Slater, "A Boolean Algebraic Analysis of Fire Protection," Albegraic and Combinatorial Methods in Operations Research, R. E. Burkard, R. A. Cuninghame-Green, and U. Zimmermann, editors, Annals of Discrete Mathematics (19), North-Holland, Amsterdam, 1984.

16. B. L. Hulme and R. B. Worrell, Combinatorial Optimization with Boolean Constraints, Sandia National Laboratories, Albuquerque, NM, SAND83-0085, February 1983.

17. B. L. Hulme, L. S. Baca, A. W. Shiver, and R. B. Worrell, Boolean Computation of Optimum Hitting Sets, Sandia National Laboratories, Albuquerque, NM, SAND84-0448, March 1984.

18. B. L. Hulme and L. S. Baca, Set Covering, Partition and Packing, Sandia National Laboratories, Albuquerque, NM, SAND84-0447, March 1984.

19. B. L. Hulme, "Boolean Methods of Optimization over Independence Systems," pp. 255-262, SIAM J. Algebraic and Discrete Methods, Vol. 5, No. 2, June 1984.

20. B. L. Hulme and L. S. Baca, A Boolean Approach to Zero-One Linear Programming, Sandia National Laboratories, Albuquerque, NM, SAND84-1409, July 1984.

21. D. D. Boozer and R. B. Worrell, A Method for Determining the Susceptibility of a Facility to Sensor System Nullification by Insiders, Sandia National Laboratories, Albuquerque, NM, SAND77-1916C, February 1978.

22. M. D. Olman, Quantitative Fault Tree Analysis Using the Set Evaluation Program (SEP), Sandia National Laboratories, Albuquerque, NM, NUREG/CR-1935, SAND80-2712, September 1982.

23. H. E. Lambert and B. J. Davis, The Use of the Computer Code IMPORTANCE with SETS Input, Sandia National Laboratories, Albuquerque, NM, NUREG/CR-1965, SAND81-7068, March 1981.

24. B. L. Hulme, Monotone Boolean Approximation, Sandia National Laboratories, Albuquerque, NM, SAND82-2357, December 1982.

25. R. B. Worrell and B. L. Hulme, "Algebraic Approximation of Event Tree Sequences," pp. 61-62, 1983 Proceedings Annual Reliability and Maintainability Symposium, Orlando, FL, January 1983.

26. D. A. Oliver, Fault Tree Drawing Program Users Instructions, Sandia National Laboratories, Albuquerque, NM, SLA-73-0409, April 1973.

27. M. D. Olman and R. B. Worrell, A Fault Tree Representation Designed for Computer Analysis, Sandia National Laboratories, Albuquerque, NM, SC-RR-71 0615A, July 1972.

APPENDIX A

DIAGNOSTICS

1. INTRODUCTION

Errors may be detected during the reading of a SETS user program or during its interpretation and execution. There are two kinds of errors: SETS errors and SETS user program errors. The SETS errors may indicate a computer malfunction or an error in the logic or implementation of the SETS program. The SETS user program errors indicate an error in a SETS user program or its data.

2. SETS ERRORS

The SETS errors are fatal. They indicate such a serious condition that it is either impossible or undesirable to attempt to recover and continue the processing. The SETS errors can occur during any phase of the processing of a SETS user program. When a SETS error is detected, the appropriate error message is printed and the processing of the SETS user program is then terminated. The SETS errors are comprised of illegal branch errors, file processing errors, and an output format error.

2.1 Illegal Branch Errors

There are three illegal branch errors that can occur. The messages printed for these SETS errors are as follows:

```
AN ILLEGAL TRANSFER HAS OCCURRED FROM A
COMPUTED GOTO STATEMENT

AN ITERATION PROCESS HAS BEEN COMPLETED WHICH
SHOULD HAVE BEEN EXITED PRIOR TO COMPLETION

THERE HAS BEEN A COMPUTER MALFUNCTION
OR AN ERROR EXISTS IN THE SETS PROGRAM
```

An illegal branch error can be caused by a computer malfunction, or it can occur because of an error in the SETS program. If an illegal branch error is caused by a computer malfunction, it can be eliminated by running the job again when the computer is functioning properly. However, if an illegal branch error occurs because of an error in the SETS program, then the SETS program must be changed to correct the error.

2.2 File Processing Errors

There are three file processing errors that can occur. The messages printed for these SETS errors are as follows:

```
AN END OF FILE ERROR HAS OCCURRED

A PARITY ERROR HAS OCCURRED

A READY ERROR HAS OCCURRED
```

A file processing error can be caused by the use of the wrong file or a bad file, or it can occur because of an error in the SETS program. If a file processing error occurs, the user should first verify that the correct files are being used. If the correct files are being used and the error persists, it may simply indicate a bad file that needs to be regenerated. However, if a file processing error occurs because of an error in the SETS program, then the SETS program must be changed to correct the error.

2.3 Output Format Error

There is one output format error that can occur. The printed message for this SETS error is as follows:

```
THE MAXIMUM NUMBER OF LINES PER PAGE IS TOO
SMALL TO ALLOW PROPER PAGING OF THE OUTPUT
```

This error can only occur if the SETS program constant that controls the maximum number of printed lines per page, has been reduced to a value that is too small to allow the headings that can occur in the printed output to be printed. The error can be eliminated by using a version of SETS that has a larger value for the constant that controls the maximum number of printed lines per page.

3. SETS USER PROGRAM ERRORS

The processing of a SETS user program occurs in two steps. First, the statements of the SETS user program are read and each statement is checked to determine whether or not it is syntactically correct. Then, if there are no syntax errors detected as the program is read, the statements of the SETS user program are interpreted and executed one at a time in the order that they occur in the program. Usually, the detection of a SETS user program error in either of these two steps does not cause immediate termination of the processing of the SETS user program. However, if a SETS user program error is detected in the program header while reading a SETS user program, the appropriate numbered error message is printed and processing is then terminated. Once the program header is read without error, the occurrence of a SETS user program error does not cause processing to be terminated, but the nature of the subsequent processing is altered.

3.1 Altered Mode of Processing

The processing that occurs after the detection of a SETS user program error is significantly different than normal processing. Further processing is restricted to trying to determine whether or not any remaining input is syntactically correct. If a SETS user program error is detected while the SETS user program is being read, an attempt is made to read the remaining statements of the program to ascertain whether or not they are syntactically correct. Only after a SETS user program is read without error are the statements of the program interpreted and executed.

The interpretation and execution of a SETS user program proceed normally unless an error is detected. If the detected error is a SETS error, processing is terminated. If the detected error is a SETS user program error, an attempt is made to execute remaining calls of BLKSTAT and to partially execute remaining calls of the input procedures RDBLK, RDFT, and RDVALBLK. The execution of

any remaining BLKSTAT call prints the names of the blocks that will be in the block file when the job terminates. The partial execution of any remaining input procedures causes the corresponding equation blocks, fault trees, or value blocks to be read and checked for syntax errors. However, once an error has occurred during the interpretation and execution of a SETS user program, the block file and the value block file are not changed in any way. Equation blocks, fault trees, and value blocks that are read after the occurrence of a SETS user program error are not added to their respective files -even if they are syntactically correct.

3.2 Special Fault Tree Errors

There are some special fault tree errors that are detected during the execution of the RDFT procedure. The messages for these SETS user program errors are as follows:

ERRORS OCCURRED IN THE DEFINITION OF f_i

THERE WAS NO DEFINITION FOR f_i

THE DEFINITION FOR f_i DOES NOT INCLUDE ITS RELATIONSHIP TO f_j

THE RELATIONSHIP BETWEEN f_i AND f_j IS INCONSISTENT

where f_i and f_j are names of fault tree events.

These special fault tree errors are the result of tests performed after a fault tree has been read by RDFT. They sometimes provide information which is already known. Suppose the event definition for some fault tree event, X, is being processed and a name with too many characters is encountered in the definition. Numbered error 33 (see Section 3.3, Numbered Errors) is detected when the name with too many characters is processed, and a numbered error message is printed. Later in the processing, after all of the event definitions of the fault tree have been read, the special message

ERRORS OCCURRED IN THE DEFINITION OF X

is also printed, even though both messages are the result of the same error. Nevertheless, the special fault tree error messages are helpful in locating errors in the structure of the fault tree.

3.3 Numbered Errors

Except for the special fault tree error messages, the detection of SETS user program errors causes numbered error messages to be printed. A numbered error message has the form:

******ERROR NUMBER: n, s

where n is the error number and s is either empty, or it is a string of characters from the SETS user program or its input. The numbered errors are described below. The information for each numbered error includes a description of the error, a description of s, and remedies for correcting the error that can be carried out by the user.

Error Number	Error Information
1	DESCRIPTION: A special character is incorrect in the context in which it occurs. Some special character is required but the one that has occurred is wrong. s: The characters that occur between the previous special character and the incorrect one will be printed. REMEDY: Correct the input.
2	DESCRIPTION: The initial characters of a header are incorrect. A SETS user program header, an equation block header, a fault tree header, or a value block header does not begin with the characters PROGRAM, BLOCK, FAULTTREE, or VALUEBLOCK, respectively. s: The characters that begin the header will be printed. REMEDY: Correct the input.
3	DESCRIPTION: The SETS user program exceeds the size of the vector used to store the program. The SETS user program being read is too large, or the unexecuted portion of the SETS user program together with the program segment generated by a call of GENFTEQN is too large. s: The name of the SETS user program will be printed. REMEDY: Use a version of SETS that has a larger program vector (PGMVC), or break up the SETS user program into several smaller SETS user programs that will achieve the same processing.
4	DESCRIPTION: A procedure identifier is incorrect. s: The incorrect procedure identifier will be printed. REMEDY: Correct the input.
5	DESCRIPTION. The parameter part of a procedure call is incorrect. s: The procedure identifier of the procedure call will be printed. REMEDY: Correct the input.

Error Number	Error Information
6	DESCRIPTION: The name in an equation block header, a fault tree header, or a value block header is not the same as the next parameter in a RDBLK, RDFT, or RDVALBLK call, respectively. s: The name from the header will be printed. REMEDY: Correct the input.
7	DESCRIPTION: The number of active variables exceeds the size of both the table that is used to hold the variable names and the vector that is used to hold variable information. s: Empty. REMEDY: Use a version of SETS that has a larger set table (SETB) and set vector (SETVC), or modify the SETS user program so that unneeded variables are eliminated from the active variables just prior to the execution of the procedure call during which the error was detected. Unneeded variables can be eliminated from the active variables at any point during the execution of a SETS user program by inserting statements into the SETS user program that will achieve the following processing: a. Form an equation block to save any meaningful equations that are in the equation file. b. Use a call of DLTEQN with an empty parameter part to delete all equations (and consequently all active variables). c. Load only the equations (and therefore only the variables) that are required for the processing to be done. Also, whenever possible, minimize the number of active variables before calls of RDBLK, RDFT, RDVALBLK, PRTBLK, and PRTVALBLK since execution of these procedures temporarily increases the number of active variables.
8	DESCRIPTION: One of the records of an equation block, a fault tree block, or a value block exceeds the size of the vector that is used as a transfer area when deleting specific equation or fault tree blocks from the block file, or when deleting specific value blocks from the value block file. s: Empty. REMEDY: Use a version of SETS that has a larger expression vector (EXPVC).

Error Number	Error Information
9	DESCRIPTION: OMEGA occurs in an option; a variable occurs more than once in the same option (different occurrences of the same kind of option are considered to be the same option); or a variable occurs in incompatible options. In SUBINEQN, a variable can occur only once in all of the options. In REDUCEQN and TRNTRMVAL, if a variable occurs in an omega or phi option, it cannot occur in any other option; if a variable does not occur in either an omega or phi option, it can occur in every other option. s: The variable that caused the error will be printed. REMEDY: Correct the input.
10	DESCRIPTION: The left-side variable of an equation is OMEGA. s: Empty. REMEDY: Correct the input.
11	DESCRIPTION: An empty block has been formed. s: The block name will be printed. REMEDY: Correct the input if the empty block is being read by RDBLK or RDFT; or change the logic in a call of FRMNEWFT that produces an empty fault tree block.
12	DESCRIPTION: A fault tree contains an incorrect key word. s: The incorrect key word will be printed. REMEDY: Correct the input.
13	DESCRIPTION: An event definition does not have any relationship declarations. s: The name of the event with no relationships will be printed. REMEDY: Correct the input.
14	DESCRIPTION: A fault tree contains OMEGA as an event name. s: Empty. REMEDY: Correct the input.

Error Number	Error Information
15	DESCRIPTION: A fault tree event has more than one definition. s: The name of the event with multiple definitions will be printed. REMEDY: Correct the input.
16	DESCRIPTION: A fault tree begins with a relationship declaration instead of an event declaration. s: The name of the fault tree will be printed. REMEDY: Correct the input.
17	DESCRIPTION: The number of prefixes in a fault tree exceeds the size of the table used to hold them. s: The prefix that caused the error will be printed. REMEDY: Correct the input.
18	DESCRIPTION: The number of relationships in a fault tree exceeds the size of the vector used to hold them. s: The fault tree name will be printed. REMEDY: Use a version of SETS that has a larger relationship vector (RELVC).
19	DESCRIPTION: A fault tree event has too many relationships. s: The name of the event with too many relationships will be printed. REMEDY: Change the fault tree being read by RDFT or generated by FRMNEWFT into an equivalent fault tree that does not have any events with too many relationships.
20	DESCRIPTION: An event in a relationship declaration is the same as the event being defined, or it occurs in more than one relationship declaration in the same event definition. (The same event can occur in a similar input or a similar output declaration if the prefixes are not identical.) s: The name of the event being defined will be printed. REMEDY: Correct the input.

Error Number	Error Information
21	DESCRIPTION: An intermediate event definition that has a similar output declaration, also has an output declaration or a similar input declaration. s: The name of the event being defined will be printed. REMEDY: Correct the input.
22	DESCRIPTION: A primary event definition has relationship declarations other than output declarations. s: The name of the event being defined will be printed. REMEDY: Correct the input.
23	DESCRIPTION: A special intermediate event definition contains a similar input declaration. s: The name of the event being defined will be printed. REMEDY: Correct the input.
24	DESCRIPTION: An intermediate event definition does not contain any input declarations. s: The name of the event being defined will be printed. REMEDY: Correct the input.
25	DESCRIPTION: The right-side expression in a specia intermediate event definition does not contain all of the events that occur in the input declarations. s: The name of the special intermediate event will be printed. REMEDY: Correct the input.
26	DESCRIPTION: The right-side expression in a special intermediate event definition contains at least one event that does not occur in an input declaration. s: The name of the special intermediate event will be printed. REMEDY: Correct the input.

Error Number	Error Information
27	DESCRIPTION: An expression exceeds the size of the vector used to hold it for various kinds of processing. s: Empty. REMEDY: Use a version of SETS that has a larger expression vector (EXPVC), or try to break up the expression into several smaller expressions that can be processed separately to achieve an equivalent result.
28	DESCRIPTION: A conditioning event is related to an event that is not defined by a PRIORITY AND gate or an INHIBIT gate. s: The name of the conditioning event will be printed. REMEDY: Correct the input.
29	DESCRIPTION: An event defined by a PRIORITY AND gate or an INHIBIT gate does not have exactly one conditioning event related to it. s: The name of the event defined by the PRIORITY AND gate or the INHIBIT gate will be printed. REMEDY: Correct the input.
30	DESCRIPTION: An event defined by a PRIORITY AND gate does not have at least two input events related to it. s: The name of the event defined by the PRIORITY AND gate will be printed. REMEDY: Correct the input.
31	DESCRIPTION: An event defined by an INHIBIT gate does not have exactly two input events related to it. s: The name of the event defined by the INHIBIT gate will be printed. REMEDY: Correct the input.

Error Number	Error Information
32	DESCRIPTION: A fault tree contains at least two similar trees that overlap, i.e., a generated event name has more than one prefix. s: The generated name of the event that caused the error will be printed. REMEDY: Correct the input.
33	DESCRIPTION: The number of characters in a name or key word exceeds the size of the vector used to catenate the characters. s: The first sixteen characters of the name or key word will be printed. REMEDY: Correct the input.
34	DESCRIPTION: Two special characters are adjacent in a context where such an occurrence is incorrect. s: Empty. REMEDY: Correct the input.
35	DESCRIPTION: Two special characters are not adjacent in a context where such an occurrence is required. s: The characters that occur between the two special characters will be printed. REMEDY: Correct the input.
36	DESCRIPTION: A generated event name is OMEGA, or it is identical to a nongenerated event name, or it is identical to another generated name but the prefixes are different. s: The generated event name will be printed. REMEDY: Correct the input.
37	DESCRIPTION: The block file does not contain the specified equation block or fault tree, or the value block file does not contain the specified value block. s: The name of the specified equation block, fault tree, or value block will be printed. REMEDY: Correct the input.

Error Number	Error Information
38	DESCRIPTION: An equation cannot be printed without exceeding the maximum length allowed for each line of print. s: Empty. REMEDY: Use a version of SETS that allows a larger maximum line length.
39	DESCRIPTION: The right-side expression of an equation is incorrect. A variable follows a right parenthesis. s: The left-side variable of the equation will be printed. REMEDY: Correct the input.
40	DESCRIPTION: The right-side expression of an equation is incorrect. There is at least one unpaired left parenthesis. s: The left-side variable of the equation will be printed. REMEDY: Correct the input.
41	DESCRIPTION: The right-side expression of an equation is incorrect. The period that terminates the right-side expression follows the equivalence operator, a left parenthesis, or an operator. s: The left-side variable of the equation will be printed. REMEDY: Correct the input.
42	DESCRIPTION: The right-side expression of an equation is incorrect. An AND or OR operator follows a left parenthesis or another operator. s: The left-side variable of the equation will be printed. REMEDY: Correct the input.
43	DESCRIPTION: The right-side expression of an equation is incorrect. A NOT operator follows a NOT operator, a right parenthesis, or a variable. s: The left-side variable of the equation will be printed. REMEDY: Correct the input.

Error Number	Error Information
44	DESCRIPTION: The right-side expression of an equation is incorrect. A left parenthesis follows a right parenthesis or a variable. s: The left-side variable of the equation will be printed. REMEDY: Correct the input.
45	DESCRIPTION: The right-side expression of an equation is incorrect. A right parenthesis follows a left parenthesis or an operator. s: The left-side variable of the equation will be printed. REMEDY: Correct the input.
46	DESCRIPTION: The right-side expression of an equation is incorrect. There is at least one unpaired right parenthesis. s: The left-side variable of the equation will be printed. REMEDY: Correct the input.
47	DESCRIPTION: The level of an AND or OR operator exceeds the maximum level that can be represented in the form of an expression that is required for expansion. s: Empty. REMEDY: Break up the expression into several smaller expressions that can be processed separately to achieve an equivalent result.
48	DESCRIPTION: The left-side variable of an equation or its complementary variable occurs in the right-side expression of the equation, or a sequence of substitutions is unending. s: For the first case, the left-side variable will be printed. For the second case, the left-side variable from the first repeated equation in the unending sequence of substitutions will be printed. REMEDY: Change one or more of the equations so that the circular definition of the equation is eliminated.
49	DESCRIPTION: The form of a number is incorrect. s: Empty. REMEDY: Correct the input.

Error Number	Error Information
50	DESCRIPTION: The number of variables in a term of an expression exceeds the size of the vector that is used to record the positions of those variables in the expression during the expansion of the expression into a disjunctive normal form. s: Empty. REMEDY: Break up the expression into several smaller expressions that can be processed separately to achieve an equivalent result.
51	DESCRIPTION: The expression number for the right-side expression of an equation to be entered into the equation file exceeds the maximum expression number allowed. s: The name of the SETS user program will be printed. REMEDY: Initialize the expression number by inserting statements into the SETS user program that will achieve the following processing: a. Form an equation block to save any meaningful equations in the equation file. b. Use a call of DLTEQN with an empty parameter part to delete all equations from the equation file. This deletes all right-side expressions and causes the expression number to be initialized.
52	DESCRIPTION: The number of pattern words used to indicate the active terms in a disjunctive normal form of an expression exceeds the size of the vector used to hold pattern words. s: Empty. REMEDY: Break up the expression into several smaller expressions that can be processed separately to achieve an equivalent result.
53	DESCRIPTION: The number of variables in an expression exceeds the size of the vector used to count the number of occurrences of each variable. s: Empty. REMEDY: Break up the expression into several smaller expressions that can be processed separately to achieve an equivalent result.

Error Number	Error Information
54	DESCRIPTION: A fault tree specified in a FRMNEWFT or GENFTEQN call contains similar trees. s: The name of the fault tree that contains similar trees will be printed. REMEDY: Correct the input.
55	DESCRIPTION: Two fault tree events that have the same name cannot be merged. The reasons that two fault tree events with the same name cannot be merged are as follows: a. Both events are primary events but their types are different. b. Both events are intermediate events but their types are different. c. Both events are intermediate events of the same type but they are defined by SPECIAL gates, or the inputs of the two events do not agree in both number and name. d. One event is a primary event and one event is an intermediate event, but the primary event is not a developed event. s: The name of the two events will be printed. REMEDY: Change the fault trees so that they do not contain events that cannot be merged.
56	DESCRIPTION: An intermediate event is specified in an omega or a phi option in a call of FRMNEWFT. Only primary events can be set to constant OMEGA or /OMEGA in a call of FRMNEWFT. s: The name of the intermediate event will be printed. REMEDY: Correct the input.
57	DESCRIPTION: A primary event is specified in an intermediate event type option in a FRMNEWFT call. A primary event cannot be changed to an intermediate event in a call of FRMNEWFT. s: The name of the primary event will be printed. REMEDY: Correct the input.

Error Number	Error Information
58	DESCRIPTION: A special intermediate event type option occurs in a call of FRMNEWFT. A fault tree event cannot be changed to a special intermediate event in a call of FRMNEWFT. s: Empty. REMEDY: Correct the input.
59	DESCRIPTION: A top option contains a primary event. s: The primary event name will be printed. REMEDY: Correct the input.
60	DESCRIPTION: A block to be added to the block file has the same name as a block already in the block file. s: The name of the blocks will be printed. REMEDY: Correct the input.
61	DESCRIPTION: The information that describes the fault tree changes required to coalesce a group of adjacent events or to remove a single input event in a call of FRMNEWFT, exceeds the vector that is used to store the information. s: Empty. REMEDY: Change the fault tree so the group of events to be coalesced is smaller, or the single input event has fewer output events.
62	DESCRIPTION: A SETS user program, an equation block, a fault tree, or a value block contains an illegal character. s: Empty. REMEDY: Correct the input.
63	DESCRIPTION: A value block is empty. s: The name of the value block will be printed. REMEDY: Correct the input.

Error Number	Error Information
64	DESCRIPTION: A computation type is incorrect. s: The procedure identifier of the procedure that contains the incorrect computation type will be printed. REMEDY: Correct the input.
65	DESCRIPTION: A value block contains at least one value that is incompatible with the variable range for the computation type it is paired with in a COMTRMVAL or TRNTRMVAL call. s: The name of the value block will be printed. REMEDY: Correct the input.
66	DESCRIPTION: The number of computations specified in a COMTRMVAL or TRNTRMVAL call exceeds the maximum number allowed. s: Empty. REMEDY: Correct the input.
67	DESCRIPTION: The right-side expression of an equation specified in a call of COMTRMVAL or TRNTRMVAL contains at least one variable that does not have a value in every value block specified in the call. s: Empty. REMEDY: Correct the input.
68	DESCRIPTION: The number of term values computed for an equation specified in a COMTRMVAL call, exceeds the size of the vector used to store the term values. s: The left-side variable of the equation will be printed. REMEDY: Use a version of SETS that has a larger relationship vector (RELVC).
69	DESCRIPTION: The number of partial term values that could exist during the expansion of the right-side expression of the equation specified in a TRNTRMVAL call, would exceed the size of the vector used to hold the partial term values. s: Empty. REMEDY: Use a version of SETS that has a larger relationship vector (RELVC).

Error Number	Error Information
70	DESCRIPTION: A parameter that is supposed to be an integer is not. s: Empty. REMEDY: Correct the input.
71	DESCRIPTION: The number of next largest subtrees that are to be tested to determine whether or not they are independent subtrees, exceeds the size of the vector that is used to save the top events of these subtrees. s: Empty. REMEDY: Break up the fault tree into several smaller fault trees that can be processed separately to achieve an equivalent result.
72	DESCRIPTION: The number of variable values that must be recorded during the computation of distinct computation types exceeds the size of the vector that is used to hold them. s: Empty. REMEDY: Decrease the number of distinct computation types that are specified in a COMTRMVAL or TRNTRMVAL call, or break up the expression into several smaller expressions whose terms can be processed separately.
73	DESCRIPTION: A value block contains a value for the constant OMEGA. s: The name of the value block will be printed. REMEDY: Correct the input.
74	DESCRIPTION: An option of FRMNEWFT contains OMEGA. s: The procedure identifier will be printed. REMEDY: Correct the input.

This page intentionally left blank.

APPENDIX B

PARAMETER FORMS FOR PROCEDURE CALLS

1. INTRODUCTION

Parameter forms for each procedure are described using the following notation:

v_i - variable name

e_i - equation name, i.e., the left-side variable of an equation

t_i - truncation value (for number of variables)

p_i - e_i or e_i/t_i

ft or ft_i - fault tree block name

eb or eb_i - equation block name

b_i - block name (fault tree or equation)

vb_i - value block name

O or O_i - procedure option

c_i - computation type

l_i - truncation value (for computations)

q_i - c_i or c_i/l_i

f_i or f_i' - fault tree event name (intermediate or primary)

ie or ie_i - fault tree intermediate event name

pe or pe_i - fault tree primary event name

Also, a character string enclosed by angle brackets ⟨ ⟩ represents any example of the entity described by the character string. For example, ⟨variable list⟩ represents any list of variable names separated by commas.

2. EQUATION PROCEDURES

Print Equation

PRTEQN (e_1,e_2,...,e_n).

Print Equation in Disjunctive Normal Form

PRTEQNDNF (p_1,p_2,...,p_n).

Delete Equation

a. DLTEQN.

b. DLTEQN (e_1,e_2,...,e_n).

Substitute in Equation

a. SUBINEQN (e_1, e_2).

b. SUBINEQN (e_1, e_2* O_1/O_2/.../O_m).

where O_j, j=1,2,...,m has one of the forms:

STOP$ v_1,v_2,...,v_r

OMEGA$ v_1,v_2,...,v_r

PHI$ v_1,v_2,...,v_r

Reduce Equation

a. REDUCEQN (p_1, e_2).

b. REDUCEQN (p_1, e_2* O_1/O_2/.../O_m).

where O_j, j=1,2,...,m has one of the forms:

OMEGA$ v_1,v_2,...,v_r

PHI$ v_1,v_2,...,v_r

EXCEPTCMP$

EXCEPTCMP$ v_1,v_2,...,v_r

EXCEPTNONCMP$

EXCEPTNONCMP$ v_1,v_2,...,v_r

DELETECMP$

DELETECMP$ v_1,v_2,...,v_r

DELETENONCMP$

DELETENONCMP$ v_1,v_2,...,v_r

Compute Term Value

a. COMTRMVAL (q_1,vb_1,q_2,vb_2,...,q_s,vb_s* p_1,p_2,...,p_n).

b. COMTRMVAL (DECREASE$ q_1,vb_1,q_2,vb_2,...,q_s,vb_s* p_1,p_2,...,p_n).

c. COMTRMVAL (INCREASE$ q_1,vb_1,q_2,vb_2,...,q_s,vb_s* p_1,p_2,...,p_n).

Truncate on Term Value

a. TRNTRMVAL (q_1,vb_1,q_2,vb_2,...,q_s,vb_s*

‹any parameter form of REDUCEQN›).

b. TRNTRMVAL (EXTREME$ q_1,vb_1,q_2,vb_2,...,q_s,vb_s*

‹any parameter form of REDUCEQN›).

Delete Term

DLTRM (p_1,p_2,e_3).

Write Equation in Disjunctive Normal Form

WRTEQNDNF (e_1,e_2,...,e_n).

3. BLOCK PROCEDURES

Read Block

RDBLK (eb_1,eb_2,...,eb_n).

Print Block

PRTBLK (b_1,b_2,...,b_n).

Block Status

BLKSTAT.

Delete Block

a. DLTBLK.

b. DLTBLK (b_1,b_2,...,b_n).

Form Block

a. FRMBLK (eb).

b. FRMBLK (eb* O).

where O has one of the forms:

ONLY$ $e_1,e_2,...,e_r$

EXCEPT$ $e_1,e_2,...,e_r$

Load Block

LDBLK ($b_1,b_2,...,b_n$).

4. VALUE BLOCK PROCEDURES

Read Value Block

RDVALBLK ($vb_1,vb_2,...,vb_n$).

Print Value Block

PRTVALBLK ($vb_1,vb_2,...,vb_n$).

Delete Value Block

a. DLTVALBLK.

b. DLTVALBLK ($vb_1,vb_2,...,vb_n$).

5. FAULT TREE PROCEDURES

Read Fault Tree

RDFT ($ft_1,ft_2,...,ft_n$).

Form New Fault Tree

a. FRMNEWFT (FORM1$ ft_1,ft_2,...,ft_n/ ft).

b. FRMNEWFT (FORM2$ ft_1,ft_2,...,ft_n/ ft).

c. FRMNEWFT (FORM3$ ft_1,ft_2,...,ft_n/ ft_S, ft_I).

d. FRMNEWFT (FORM3$ (<prefix>) ft_1,ft_2,...,ft_n/ ft_S, ft_I).

e. FRMNEWFT (<any a through d parameter form>* $O_1/O_2/.../O_m$).

where O_j, j=1,2,...,m has one of the forms:

TOP$ ie_1,ie_2,...,ie_r

TRIM$ f_1,f_2,...,f_r

OMEGA$ pe_1,pe_2,...,pe_r

PHI$ pe_1,pe_2,...,pe_r

NAME$ f_1='f_1',f_2='f_2',...,f_r='f_r'

<type>$ f_1,f_2,...,f_r

Generate Fault Tree Equation

a. GENFTEQN (METHOD1$ ft/ eb).

b. GENFTEQN (METHOD2$ ft/ eb).

c. GENFTEQN (PUNCH$ <parameter form a or b>).

d. GENFTEQN (<any a through c parameter form>*

q_1,vb_1,q_2,vb_2,...,q_s,vb_s).

e. GENFTEQN (<any a through c parameter form>*

EXTREME$ q_1,vb_1,q_2,vb_2,...,q_s,vb_s).

This page intentionally left blank.

APPENDIX C

FACTORIZATION

1. INTRODUCTION

A disjunctive normal form expression can be factored in many different ways. Factorization in the SETS program is based on selecting the most often occurring variable in a disjunctive normal form expression as the next variable to be factored out of the expression. Factorization produces a factored expression which, when expanded by the distributive law, yields the disjunctive normal form that was used to produce the factored expression.

2. THE FACTORING ALGORITHM

A disjunctive normal form expression is factorable if it contains at least two terms that have a common product of variables. The common product of variables is chosen so that it contains the most often occurring variable in the expression, and the expression is written using the formula

$$P*(Q) + R \qquad (C-1)$$

where P is the common product of variables that contains the most often occurring variable, Q is the quotient, and R is the remainder. Each Q and each nonempty R generated by an application of formula C-1 is a disjunctive normal form expression that may also be factorable. Repeated applications of formula C-1, first to the original disjunctive normal form of an expression and then to each generated Q and R that is factorable, produces a factored form of the expression.

The factored form of an expression obtained by repeated applications of formula C-1 is not the final factored form of an expression produced by factorization. After a factored form of an expression is obtained, nonfactorable Q's and R's in the expression are replaced by single variables and the expression is expanded into a disjunctive normal form. (Nonfactorable Q's and R's that are the same are replaced by the same variable.) If this expression is factorable, a factored form of it is obtained by repeated applications of formula C-1. This process of factoring an expression by repeated applications of formula C-1, replacing nonfactorable Q's and R's in the expression by single variables, expanding the expression into a disjunctive normal form, and then factoring the expression again is repeated until an expression that is not factorable is obtained. Then, beginning with the last expression obtained, i.e., the one that is not factorable, the variables that were introduced for the nonfactorable Q's and R's are replaced by their respective Q or R to produce the final factored expression.

3. EXAMPLE

Consider the expression

$$A*C*D*E + A*C*E*F + G*D + A*B + G*F. \qquad (C-2)$$

Expression C-2 is factorable; its most often occurring variable is A; and it can be written in the form

$$A*(B + C*D*E + C*E*F) + G*D + G*F. \quad (C-3)$$

Both the quotient

$$B + C*D*E + C*E*F$$

and the remainder

$$G*D + G*F$$

generated by rewriting the original expression are factorable. Thus, Exp. C-3 can be rewritten using formula C-1 to obtain

$$A*(C*E*(D + F) + B) + G*(D + F). \quad (C-4)$$

No further factoring of Exp. C-4 can be achieved using formula C-1 because D + F, which occurs twice as a quotient, and B, which occurs once as a remainder, are not factorable. Expression C-4, a factored form of Exp. C-2, has been obtained. Now, the quotients D + F in Exp. C-4 are replaced by a variable X1 and the remainder B in Exp. C-4 is replaced by a variable X2 to obtain

$$A*(C*E*(X1) + X2) + G*(X1). \quad (C-5)$$

Expansion of Exp. C-5 produces the disjunctive normal form expression

$$A*C*E*X1 + A*X2 + G*X1. \quad (C-6)$$

Expression C-6 is factorable. Two variables, A and X1, are most often occurring variables in Exp. C-6. The variable X1 is chosen as the next factor because, in general, a variable that represents a Q or an R is preferred as a factor over the original variables in the expression. Expression C-6 is rewritten using formula C-1 to obtain

$$X1*(A*C*E + G) + A*X2. \quad (C-7)$$

No further factoring of Exp. C-7 can be achieved using formula C-1 because the quotient A*C*E + G and the remainder A*X2 are not factorable. Expression C-7, a factored form of Exp. C-6, has been obtained. Now, the quotient A*C*E + G in Exp. C-7 is replaced by X3 and the remainder A*X2 in Exp. C-7 is replaced by X4 to obtain

$$X1*(X3) + X4. \quad (C-8)$$

Expansion of Exp. C-8 produces the disjunctive normal form expression

$$X1*X3 + X4. \quad (C-9)$$

Expression C-9 is not factorable and signals the end of the iterations of factoring. Beginning with Exp. C-9, the final factored form of the expression produced by factorization is built up by replacing every occurrence of each X1, X2, X3, and X4 by its respective Q or R surrounded by parentheses. The final factored form of Exp. C-2 is

$$(D + F)*(A*C*E + G) + (A*(B)).$$

DISTRIBUTION

U.S. Government Printing Office
Receiving Branch (Attn: NRC Stock)
8610 Cherry Lane
Laurel, MD 20707
350 copies for RG, 1S

US Nuclear Regulatory Commission (5)
Office of Nuclear Regulatory Research
Washington, DC 20555
Attn: J. A. Murphy/DRAO
P. Baranowsky/DRAO
B. Buchbinder/DRAO
G. Burdick/DRAO
D. M. Rasmuson/DRAO

US Nuclear Regulatory Commission (3)
Office of Nuclear Reactor Regulation
Washington, DC 20555
Attn: R. J. Mattson/DSI
A. A. El-Bassioni/RRAB
A. Thadani/RRAB

AFWL (3)
NTSS
Kirtland AFB, NM 87117
Attn: Bob Knudson

Ansaldo Div. Impianti
via G. D'Annunzio, 113
16100 Genova
Italy
Attn: Armando Desirello

Applied Risk Technology Corporation
P.O. Box 175
Columbia, MD 21045
Attn: Paul J. Amico

Arizona Public Service (3)
11226 N23 Ave.
Phoenix, AZ 85029
Attn: Chuck Stevens

Atomic Energy Council (5)
Probabilistic Risk Assessment Section
Nuclear Regulatory Division
67, Lane 144
Keelung Rd. Sec. 4
Taipei, Taiwan
Republic of China
Attn: You-Hsin Yang

Atomic Energy of Canada, Ltd. (2)
Mississauga, Ontario
Canada, L5K1B2
Attn: R. Belluz
P. Gumley

Atomics International Division
Energy Systems Group
8900 De Soto Avenue
Canoga Park, CA 91304
Attn: Hing Ho

Babcock & Wilcox
P.O. Box 1260
Lynchburg, VA 24505
Attn: D. Westcott

Battelle Columbus Laboratories
505 King Avenue
Columbus, OH 43202
Attn: Dave Heffe

Boeing Computer Services (2)
565 Andover Park West
Tukwila, WA 98488
Attn: P. Konichek
S. Pruitt

Brookhaven National Laboratory (5)
Department of Nuclear Energy
Bldg. 130
Upton, NY 11973
Attn: R. Youngblood
J. L. Boccio
K. Shiu
R. Hall
S. Karimian

Burns & Roe, Inc.
700 Kidnerkamack Road
Oradell, NJ 07675
Attn: Ed Rodrick

Categratico Tecnologia Nuclear (3)
F.T.S. Ingenieros Industriales
Jose Gutierrez Abascal, 2
28010 Madrid
Spain
Attn: A. Alonso

Chemical and Nuclear Engineering
Department
University of Maryland
College Park, MD 20742
Attn: Mohammed Modarres

Citicorp
Information Resources
1600 Summer Street
Stamford, CT 06905
Attn: G. R. Ellison

Clinch River Breeder Reactor
Project Office (2)
P.O. Box U
Oak Ridge, TN 37830
Attn: R. Emmett
S. Frye

Combustion Engineering, Inc.
Department 9485-2405
1000 Prospect Hill Road
Windsor, CT 06095
Attn: Rupert Weston

Comision Nacional de Seguridad (4)
Nuclear y Salvaguardias
Av. Insurgentes Sur No. 1806
Col. Florida
Mexico D. F., 01030
Mexico
Attn: C. Arredondo
D. Meade

Commonwealth Edison Co. (2)
P.O. Box 767
Chicago, IL 60690
Attn: G. Crane
Bernie Weber

Control Data Corporation (2)
2300 Berkshire Lane
Minneapolis, MN 55441
Attn: A. Evinay
J. Hertzeg

Control Data Corporation
1330 Orleans Drive
Sunnyvale, CA 94086
Attn: Barbara Garvin

Control Data Corporation
1801 West County Road B
Roseville, MN 55113
Attn: Charles Swanson

Control Data Corporation
59 Hallcrown Place
Willowdale, Ontario, M2J1P7
Canada
Attn: Lionel Wolpert

CRAY Research (3)
1440 Northland Drive
Mendota Heights, MN 55120
Attn: S. Graffunder

CRAY Research
5776 Stoneridge Mall Road
Pleasonton, CA 94566
Attn: Bence Gerber

CYGNA Incorporated
101 California Street
10th Floor
San Francisco, CA 94111
Attn: R. T. Cantrell

Delian Corporation (2)
545 Shoup Ave.
Suite 310
Idaho Falls, ID 83402
Attn: W. Sullivan
S. Mays

Delian Corporation
1340 Saratoga-Sunnyvale Road
Suite 206
San Jose, CA 95129
Attn: W. Brinsfield

Delian Corporation
296 Duff Road
Monroeville, PA 15146
Attn: B. Brogan

Department of Nuclear Engineering
Suwon Campus
Kyung-Hee University
Seoul
Republic of Korea
Attn: Won-Guk Hwang

Department of Nuclear Energy, BF-10
University of Washington
Seattle, WA 98195
Attn: Norman McCormick

Duke Power Company (5)
Nuclear Production Department
422 South Church Street
Charlotte, NC 28242
Attn: T. Daniels
L. D. McClain
L. Reed
R. Summitt
J. Taylor

EBASCO Services, Inc. (3)
160 Chubb Avenue
Lyndhurst, NJ 07071
Attn: T. J. Raney
W. Rish
J. Circle

EDS Nuclear, Inc.
350 Lennon Lane
Walnut Creek, CA 94598
Attn: Tim Lee-Thorp

EDS Nuclear
333 Technology Park
Norcross, GA 30092
Attn: Kevin Kimball

EG&G Idaho, Inc.
1580 Sawtelle Road
Idaho Falls, ID 83401
Attn: E. A. Krantz

Electric Power Research Institute (3)
Nuclear Power Division
3412 Hillview Avenue
Palo Alto, CA 94304
Attn: David H. Worledge

ENEL-CTN (5)
Viale Regina Margherita, 137
00158 Rome
Italy
Attn: V. Cavicchia
G. Del Nero
G. Granato
S. Sicurezza
R. Tononi

Energy Inc. (5)
5345 Wyoming Blvd., NE
Suite 101
Albuquerque, NM 87109
Attn: S. Asselin
J. LaChance
Kiyoto Aizawa
T. Zimmerman

Energy Inc. (2)
1851 South Central Place
Suite 201
Kent, WA 28031
Attn: Samuel M. Lainoff
J. Young

Evaluation Associates, Inc.
One Belmont Avenue
Bala Cynwyd, PA 19004
Attn: Tom Weir

FBR Development Project
PNC
9-13, 1-Chome, Akasaka
Minato-Ku, Tokyo, T107
Japan
Attn: Yoshio Kani

FBR Safety Research
PNC
9-13, 1-Chome, Akasaka
Minato-Ku, Tokyo, T107
Japan
Attn: Akira Watanabe

Fluor Technology, Inc.
3333 Michelson Drive
Irvine, CA 92730
Attn: Robert Mulvihill F4U

FPL
P.O. Box 14000
Juno Beach, FL 33408
Attn: John O'Neill

GA Technologies Inc. (2)
P.O. Box 85608
San Diego, CA 92138
Attn: R. Bolig
D. Ligon

General Dynamics (3)
Fort Worth Division
P.O. Box 748
Fort Worth, TX 76101
Attn: J. Nash
M. D. Krueger
C. Strukley

General Electric Company (3)
Mail Code: 166
175 Curtner Avenue
San Jose, CA 95125
Attn: A. Bartu

Howard E. Lambert
3728 Brunell Drive
Oakland, CA 94602

Hughes Aircraft Company
Ground Systems Group
MS-C315
P.O. Box 3310
Fullerton, CA 92634
Attn: G. Berg

IBM (2)
Bodle Hill Road
Owego, NY 13827
Attn: Ray Conrad
N. Kachman

Imatran Voima OY (3)
Loviisa NPS
07900 Loviisa
Finland
Attn: Jussi K. Vaurio

Interdevelopment, Inc. (2)
2361 Jefferson Davis Highway
Suite 1014
Arlington, VA 22202
Attn: F. Faris
J. Farmer

International Energy Associates, Ltd.
600 New Hampshire Avenue NW
Washington, DC 20037
Attn: J. Mark Elliott

International Energy Associates, Ltd.
1717 Louisiana, NE
Albuquerque, NM 87110
Attn: G. Bruce Varnado

JBF Associates, Inc. (2)
Technology Drive
1000 Technology Park Center
Knoxville, TN 37932
Attn: J. B. Fussell
J. J. Rooney

Korea Advanced Energy Research Institute
Reactor Systems Research Division
Cheong Ryang
P.O. Box 7
Seoul
Republic of Korea 131
Attn: Kun Joong Yoo

Lawrence Livermore National Laboratory (4)
P.O. Box 808
Livermore, CA 94550
Attn: J. Wells
L. George
D. Lappa
A. Garcia

Judy Lim
2615 Buenos Aires Court
Walnut Creek, CA 94596

Los Alamos National Laboratory (6)
Los Alamos, NM 87545
Attn: R. Bartholomew
C. A. Coulter
Marlene Miller
W. J. Whitty
D. W. Stack
R. Harmon

Los Alamos Technical Associates
1650 Trinity
Los Alamos, NM 87554
Attn: Paul Pan

Nancy Mann
UCLA School of Medicine
Dept. of Biomathematics
Los Angeles, CA 90024

McAuto (2)
P.O. Box 516
St. Louis, MO 63166
Attn: L. Siebern
D. Niesse

McDonnell Douglas (2)
5301 Bolsa MC11-3
Huntington Beach, CA 92647
Attn: Russ Brown

MHB Associates
1723 Hamilton Avenue
Suite K
San Jose, CA 95125
Attn: Curran Rollier

MIT
NW13-240
Cambridge, MA 02139
Attn: Michael Fellows

Mission Research Corporation
1720 Randolph Rd., SE
Albuquerque, NM 87106
Attn: Dave Mavis

National Energy Software Center (40)
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439
Attn: Margaret Butler

New York Power Authority (3)
123 Main Street
White Plains, NY 10601
Attn: R. Deem
J. Favara
I. D. Mew

Niagara Mohawk
300 Erie Boulevard West El
Syracuse, NY 13202
Attn: Nancy Stoeckert

Northeast Utilities Service Co.
P.O. Box 270
Hartford, CT 06141
Attn: Nirmal K. Jain

Nuclear Power Plant BEZNAU (3)
CH-5312 Doettingen
Switzerland
Attn: S. Sahgel Suresh

NUS Corporation (2)
Consulting Division
910 Clopper Road
Gaithersburg, MD 20878
Attn: R. Karimi
S. Kalsenelenbogen

NUTECH Engineers (3)
6835 Via Del Oro
San Jose, CA 95119
Attn: S. Loghmanpour
E. Kujawski
L. Voice

Ontario Hydro (3)
700 University Avenue
Toronto, Ontario, M5G1X6
Canada
Attn: B. R. Collingwood
Peter Chow
R. Starr

Pacific Gas & Electric
Engineering Computer Applications
Department
77 Beale Street
Room 2461
San Francisco, CA 94106
Attn: Bernard Davis

Pennsylvania Power and Light
Two North 9th Street
Allentown, PA 18101
Attn: Casimir Kukielka

Pickard, Lowe & Garrick, Inc. (3)
17840 Skypark Blvd.
Irvine, CA 92714
Attn: M. Kazarians
J. C. Lin
D. Wheeler

Public Service Electric & Gas Co.
Nuclear Department
Administration Building (TB-2)
P.O. Box 236
Hancocks Bridge, NJ 08038
Attn: Frank Thomson

Ralph M. Parsons Co. (2)
100 West Walnut Street
Pasadena, CA 91124
Attn: Song Huang
B. Kolodji

Rheinisch-Westfalischer TuV e.V.
Leiter der Zentralabteilung IV.3.8
Postfach 10 32 61
D-4300, Essen 1
West Germany
Attn: K. D. Paul

Risk Management Associates
2309 Dietz Farm Road, NW
Albuquerque, NM 87107
Attn: Peter Bieniarz

Rockwell International Corporation (2)
Safety Analysis Engineering
P.O. Box 464
Golden, CO 80401
Attn: Margaret F. Hickey, Manager

Safety and Reliability Optimization Services, Inc.
624 Glen Willow Drive
Knoxville, TN 37922
Attn: Gary J. Boyd

Sandia National Laboratories (20)
Organization 6400
P. O. Box 5800
Albuquerque, NM 87185
Attn: R. Cochrell

Science and Engineering Associates (6)
2500 Louisiana Blvd. NE, Suite 610
Albuquerque, NM 87110
Attn: J. L. Darby
S. V. Walker
G. Williams
B. Biringer
E. Clayburn
J. Simons

Science Applications International Corporation
505 Marquette NW
Suite 1200
Albuquerque, NM 87102
Attn: Alan Kolaczkowski

Science Applications International Corporation (3)
5150 El Camino Real
Suite C-31
Los Altos, CA 94022
Attn: R. Fullwood
B. Putney
J. Riley

Science Applications International Corporation (2)
P.O. Box 2351
La Jolla, CA 92038
Attn: P. Lobner
W. Horton

Science Applications International
Corporation (2)
274 Madison Avenue
Suite 1501
New York, NY 10016
Attn: J. R. Fragola
E. Collins

Science Applications, International
Corporation
1710 Goodridge Drive
McLean, VA 22102
Attn: Phuoc Le

Structural Engineering Laboratory
Kobe Steel, Ltd.
6-32 DOI
Amagasaki, 660
Japan
Attn: Masayuki Fujino

Structural Mechanics Associates
5160 Birch Street, 2nd Floor
Newport Beach, CA 92660
Attn: M. K. Ravindra

Systems Reliability Service (2)
National Centre of Systems
Reliability
UKAEA
Wigshaw Lane
Culcheth Warrington
WA3 4NE
United Kingdom
Attn: I. A. Watson

Technology for Energy Corporation
One Energy Center
Pellissippi Parkway
Knoxville, TN 37922
Attn: M. Akhtar

UCCEL Corporation (3)
1930 Hi Line Drive
Dallas, TX 75207
Attn: R. Swanson

UNC
310 S. Buchanan Ct.
Kennewick, WA 99336
Attn: Michael D. Zentner

United Information Services
7700 Leesburg Pike
Suite 200
Falls Church, VA 22043
Attn: Randy Foltz

Universite De Bordeaux I
Institut Universitaire de
Technologie "A"
Departement Hygiene et Securite
33405 Talence Cedex
France
Attn: Dr. Y. Dutuit

University of Bradford
Bradford
West Yorkshire
BD7 1DP
United Kingdom
Attn: A. Z. Keller

University of Santander
Santander
Spain
Attn: Prof. Enrique Castillo

Westinghouse (OPS)
P.O. Box 8000
Jacksonville, FL 32211
Attn: Bill Nichols

Westinghouse Electric Corporation
P.O. Box 355
Pittsburgh, PA 15230
Attn: D. Paddleford
W. E. Shopsky

Yankee Atomic (3)
1671 Worcester Rd.
Framingham, MA 01701
Attn: J. Chapman
S. M. Follen
S. Dinsmore

0310 P. A. Stokes
0334 H. A. Bennett
1640 G. J. Simmons
1641 D. B. Holdridge
1642 B. L. Hulme
2100 B. L. Gregory
2600 R. J. Detry

DISTRIBUTION (Cont.)

2646 M. R. Scott
3141 C. M. Ostrander (5)
3151 W. L. Garner
4041 R. P. Stromberg
5268 C. E. Olson
6321 M. G. Vannoni
6400 A. W. Snyder
6410 J. W. Hickman
6411 D. M. Kunsman
6412 A. L. Camp
6412 M. P. Bohn
6412 F. T. Harper
6412 G. J. Kolb
6412 S. H. McAhren
6412 A. C. Payne, Jr.
6412 T. A. Wheeler
6412 D. W. Whitehead
6412 R. B. Worrell (6)
6414 D. M. Ericson, Jr.
6414 W. R. Cramond
6414 S. L. Daniel
6414 S. W. Hatch
6415 F. E. Haskin
6415 R. L. Iman
6417 D. D. Carlson
6430 N. R. Ortiz
6432 L. D. Chapman
7200 J. M. Wiesen
7220 R. R. Prairie
7222 G. T. Merren
7230 W. L. Stevens
7233 R. E. Smith
8024 M. A. Pound
8230 W. D. Wilson

NRC FORM 335 (2-84) NRCM 1102, 3201, 3202

U.S. NUCLEAR REGULATORY COMMISSION

BIBLIOGRAPHIC DATA SHEET

SEE INSTRUCTIONS ON THE REVERSE

1. REPORT NUMBER (Assigned by TIDC add Vol. No., if any)

NUREG/CR-4213
SAND83-2675

2. TITLE AND SUBTITLE

SETS Reference Manual

3. LEAVE BLANK

4. DATE REPORT COMPLETED

MONTH: April | YEAR: 1985

5. AUTHOR(S)

Richard B. Worrell

6. DATE REPORT ISSUED

MONTH: May | YEAR: 1985

7. PERFORMING ORGANIZATION NAME AND MAILING ADDRESS (Include Zip Code)

Reactor Systems Safety Analysis
Division 6412
Sandia National Laboratories
Albuquerque, NM 87185

8. PROJECT TASK WORK UNIT NUMBER

9. FIN OR GRANT NUMBER

A1360

10. SPONSORING ORGANIZATION NAME AND MAILING ADDRESS (Include Zip Code)

Reactor Risk Branch
Office of Nuclear Regulatory Research
US Nuclear Regulatory Commission
Washington, DC 20555

11a. TYPE OF REPORT

Final Report

b. PERIOD COVERED (Inclusive dates)

12. SUPPLEMENTARY NOTES

13. ABSTRACT (200 words or less)

The Set Equation Transformation System (SETS) is used to achieve the symbolic manipulation of Boolean equations. Symbolic manipulation involves changing equations from their original forms into more useful forms -- particularly by applying Boolean identities. The SETS program is an interpreter which reads, interprets, and executes SETS user programs. The user writes a SETS user program specifying the processing to be achieved and submits it, along with the required data, for execution by SETS. Because of the general nature of SETS, i.e., the capability to manipulate Boolean equations regardless of their origin, the program has been used for many different kinds of analysis.

14. DOCUMENT ANALYSIS - a. KEYWORDS/DESCRIPTORS

Boolean Algebra, Symbolic Manipulation, Fault Tree Analysis, Common Cause Analysis, Probabilistic Risk Analysis

b. IDENTIFIERS/OPEN-ENDED TERMS

15. AVAILABILITY STATEMENT

Unlimited

16. SECURITY CLASSIFICATION

(This page) Unclassified

(This report) Unclassified

17. NUMBER OF PAGES

18. PRICE

U.S. GOVERNMENT PRINTING OFFICE: 1985-0-576-037/20,035