

Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Version 8

Volume 2: Technical Reference

**AVAILABILITY OF REFERENCE MATERIALS
IN NRC PUBLICATIONS**

NRC Reference Material

As of November 1999, you may electronically access NUREG-series publications and other NRC records at NRC's Public Electronic Reading Room at <http://www.nrc.gov/reading-rm.html>. Publicly released records include, to name a few, NUREG-series publications; *Federal Register* notices; applicant, licensee, and vendor documents and correspondence; NRC correspondence and internal memoranda; bulletins and information notices; inspection and investigative reports; licensee event reports; and Commission papers and their attachments.

NRC publications in the NUREG series, NRC regulations, and *Title 10, Energy*, in the Code of *Federal Regulations* may also be purchased from one of these two sources.

1. The Superintendent of Documents
U.S. Government Printing Office
Mail Stop SSOP
Washington, DC 20402-0001
Internet: bookstore.gpo.gov
Telephone: 202-512-1800
Fax: 202-512-2250
2. The National Technical Information Service
Springfield, VA 22161-0002
www.ntis.gov
1-800-553-6847 or, locally, 703-605-6000

A single copy of each NRC draft report for comment is available free, to the extent of supply, upon written request as follows:

Address: U.S. Nuclear Regulatory Commission
Office of Administration
Publications Branch
Washington, DC 20555-0001

E-mail: DISTRIBUTION.RESOURCE@NRC.GOV
Facsimile: 301-415-2289

Some publications in the NUREG series that are posted at NRC's Web site address <http://www.nrc.gov/reading-rm/doc-collections/nuregs> are updated periodically and may differ from the last printed version. Although references to material found on a Web site bear the date the material was accessed, the material available on the date cited may subsequently be removed from the site.

Non-NRC Reference Material

Documents available from public and special technical libraries include all open literature items, such as books, journal articles, and transactions, *Federal Register* notices, Federal and State legislation, and congressional reports. Such documents as theses, dissertations, foreign reports and translations, and non-NRC conference proceedings may be purchased from their sponsoring organization.

Copies of industry codes and standards used in a substantive manner in the NRC regulatory process are maintained at—

The NRC Technical Library
Two White Flint North
11545 Rockville Pike
Rockville, MD 20852-2738

These standards are available in the library for reference use by the public. Codes and standards are usually copyrighted and may be purchased from the originating organization or, if they are American National Standards, from—

American National Standards Institute
11 West 42nd Street
New York, NY 10036-8002
www.ansi.org
212-642-4900

Legally binding regulatory requirements are stated only in laws; NRC regulations; licenses, including technical specifications; or orders, not in NUREG-series publications. The views expressed in contractor-prepared publications in this series are not necessarily those of the NRC.

The NUREG series comprises (1) technical and administrative reports and books prepared by the staff (NUREG-XXXX) or agency contractors (NUREG/CR-XXXX), (2) proceedings of conferences (NUREG/CP-XXXX), (3) reports resulting from international agreements (NUREG/IA-XXXX), (4) brochures (NUREG/BR-XXXX), and (5) compilations of legal decisions and orders of the Commission and Atomic and Safety Licensing Boards and of Directors' decisions under Section 2.206 of NRC's regulations (NUREG-0750).

DISCLAIMER: This report was prepared as an account of work sponsored by an agency of the U.S. Government. Neither the U.S. Government nor any agency thereof, nor any employee, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for any third party's use, or the results of such use, of any information, apparatus, product, or process disclosed in this publication, or represents that its use by such third party would not infringe privately owned rights.

Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Version 8

Volume 2: Technical Reference

Manuscript Completed: March 2011
Date Published: June 2011

Prepared by
C.L. Smith, S.T. Wood

Idaho National Laboratory
Battelle Energy Alliance
Idaho Falls, ID 83415

D. O'Neal, NRC Project Manager

NRC Job Code N6423

Office of Nuclear Regulatory Research

PREVIOUS REPORTS

- S. T. Wood, C. L. Smith, K. J. Kvarfordt, S. T. Beck, *Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Vol. 1 Summary Manual*, NUREG/CR-6952, August 2008.
- C. L. Smith, S. T. Wood, W. J. Galyean, J. A. Schroeder, S. T. Beck, M. B. Sattison, *Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Vol. 2 Technical Reference*, NUREG/CR-6952, August 2008.
- K. J. Kvarfordt, S. T. Wood, C. L. Smith, *Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Vol. 3 Code Reference Manual*, NUREG/CR-6952, August 2008.
- S. T. Beck, S. T. Wood, C. L. Smith, *Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Vol. 4 Tutorial*, NUREG/CR-6952, August 2008.
- C. L. Smith, J. Schroeder, S. T. Beck, *Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Vol. 5 GEM Manual*, NUREG/CR-6952, August 2008.
- C. L. Smith, R. Nims, K. J. Kvarfordt, C. Wharton, *Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Vol. 6 Quality Assurance Manual*, NUREG/CR-6952, August 2008.
- K. J. Kvarfordt, S. T. Wood, C. L. Smith, *Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Vol. 7 Data Loading Manual*, NUREG/CR-6952, August 2008.
- Smith, C. L., et al., *Testing, Verifying, and Validating SAPHIRE Versions 6.0 and 7.0*, NUREG/CR-6688, October 2000.
- K. D. Russell, et al. *Systems Analysis Programs for Hands-on Reliability Evaluations (SAPHIRE) Version 6.0 - System Overview Manual*, NUREG/CR-6532, May 1999.
- K. D. Russell et al., *Integrated Reliability and Risk Analysis System (IRRAS) Version 5.0, Volume 2 - Reference Manual*, NUREG/CR-6116, EGG-2716, July 1994.
- K. D. Russell et al., *Verification and Validation (V&V), Volume 9 – Reference Manual*, NUREG/CR-6116, EGG-2716, July 1994.
- K. D. Russell et al., *Integrated Reliability and Risk Analysis System (IRRAS) Version 4.0, Volume 1 - Reference Manual*, NUREG/CR-5813, EGG-2664, January 1992.
- K. D. Russell et al., *Integrated Reliability and Risk Analysis System (IRRAS) Version 2.5 Reference Manual*, NUREG/CR-5300, EGG-2613, March 1991.
- K. D. Russell, M. B. Sattison, D. M. Rasmuson, *Integrated Reliability and Risk Analysis System (IRRAS) - Version 2.0 User's Guide*, NUREG/CR-5111, EGG-2535, manuscript completed March 1989, published June 1990.
- K. D. Russell, D. M. Snider, M. B. Sattison, H. D. Stewart, S.D. Matthews, K. L. Wagner, *Integrated Reliability and Risk Analysis System (IRRAS) User's Guide - Version 1.0 (DRAFT)*, NUREG/CR-4844, EGG-2495, June 1987.

ABSTRACT

The Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) refers to a set of computer programs that were developed to create and analyze probabilistic risk assessment (PRAs). Herein information is provided on the principles used in the construction and operation of Version 8.0 of the SAPHIRE system. This report summarizes the fundamental mathematical concepts of sets and logic, fault trees, and probability. This volume then describes the algorithms used to construct a fault tree and to obtain the minimal cut sets. It gives the formulas used to obtain the probability of the top event from the minimal cut sets, and the formulas for probabilities that apply for various assumptions concerning reparability and mission time. It defines the measures of basic event importance that SAPHIRE can calculate. This volume gives an overview of uncertainty analysis using simple Monte Carlo sampling or Latin Hypercube sampling, and states the algorithms used by this program to generate random basic event probabilities from various distributions. Also covered are enhance capabilities such as seismic analysis, Workspace algorithms, cut set "recovery," end state manipulation, and use of "compound events."



FOREWORD

The U.S. Nuclear Regulatory Commission (NRC) has developed the Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) software that is used to perform probabilistic risk assessments (PRAs) on a personal computer. SAPHIRE enables users to supply basic event data, create and solve fault and event trees, perform uncertainty analyses, and generate reports. In that way, analysts can perform PRAs for any complex system, facility, or process.

For nuclear power plant PRAs, SAPHIRE can be used to model a plant's response to initiating events, quantify core damage frequencies, and identify important contributors to core damage (Level 1 PRA). The program also can be used to evaluate containment failure and release models for severe accident conditions given that core damage has occurred (Level 2 PRA). In so doing, the analyst could build the PRA model assuming that the reactor is initially at full power, low power, or shutdown. In addition, SAPHIRE can be used to analyze both internal and external events and, in a limited manner, to quantify the frequency of release consequences (Level 3 PRA). Because this software is a very detailed technical tool, users should be familiar with PRA concepts and methods used to perform such analyses.

SAPHIRE has evolved with advances in computer technology and users' needs. Starting with Version 5, SAPHIRE operated in the Microsoft Windows™ environment. Versions 6 and 7 included features and capabilities for developing and using larger, more complex models. SAPHIRE Version 8 includes significant new features and capabilities to meet user needs for NRC risk-informed programs. In general, these include:

Improved user interfaces supporting NRC's Significance Determination Process, event and condition assessments, and more detailed types of PRA analyses.

Development and use of NRC's Standardized Plant Analysis Risk models.

New and improved solving algorithms.

Support features for user-friendliness.

This NUREG-series report comprises seven volumes as outlined below and incorporates new features and capabilities of Version 8.

Volume 1, "Overview and Summary"

Volume 1 provides an overview of the functions and features available in SAPHIRE Version 8 and presents general instructions for using the software.

Volume 2, "Technical Reference"

Volume 2 summarizes the fundamental mathematical concepts of sets and logic, fault trees, and probability. It then describes the algorithms used to construct a fault tree and to obtain the minimal cut sets. This report presents the formulas used to obtain the probability of the top event from the minimal cut sets and the formulas for probabilities that apply for various assumptions concerning reparability and mission time. In addition, it defines the measures of basic event importance that SAPHIRE can calculate. This volume also gives an overview of uncertainty analysis using simple Monte Carlo sampling or Latin Hypercube sampling and states

the algorithms used by this program to generate random basic event probabilities from various distributions. Finally, this report discusses enhanced and new capabilities such as post-processing rules, integrated model solving using model types, and workspace analysis routines.

Volume 3, "Users' Guide"

Volume 3 provides a brief discussion of the purpose and history of the software as well as general information such as installation instructions, starting and stopping the program, and some pointers on how to get around inside the program. Next, it discusses database concepts and structure. The following nine sections (one for each of the menu options on the SAPHIRE main menu) furnish the purpose and general capabilities for each option. Finally, Volume 3 provides the capabilities and limitations of the software.

Volume 4, "Tutorial"

Volume 4 provides a series of lessons that guide the user through basic steps common to most analyses performed with SAPHIRE.

Volume 5, "Workspaces"

Volume 5 describes the functionality and process behind SAPHIRE Version 8 workspaces. Workspaces provide an area in which a PRA model can be analyzed to obtain risk insights for a given initiating event or condition. Workspaces replace the "Graphical Evaluation Module" in earlier SAPHIRE versions.

Volume 6, "Quality Assurance"

Volume 6 is designed to describe how the SAPHIRE software quality assurance (QA) is performed for Version 8, what constitutes its parts, and the limitations of those processes. In addition, this report describes the Independent Verification and Validation that was conducted for Version 8 as part of an overall QA process.

Volume 7, "Data Loading"

Volume 7 is designed to guide the user through the basic procedures necessary to enter PRA data into the SAPHIRE program using SAPHIRE's MAR-D ASCII-text (or "flat file") data formats. In addition, this manual covers loading data through the new Accident Sequence Matrix and discusses the Project Integrate interfaces with SAPHIRE.

Christiana H. Lui, Director
Division of Risk Analysis
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission

CONTENTS

<u>Section</u>	<u>Page</u>
PREVIOUS REPORTS	ii
ABSTRACT	iii
FOREWORD	v
LIST OF FIGURES	xi
LIST OF TABLES	xii
EXECUTIVE SUMMARY	xiii
ACKNOWLEDGEMENTS	xv
ACRONYMS	xvii
1. INTRODUCTION	1
1.1 Background	1
2. SET THEORETIC AND LOGICAL CONCEPTS	5
2.1 Set Theoretic Concepts	5
2.2 Operations on Sets	6
2.3 Summary of Useful Identities	10
2.4 Concepts of Statement Logic	10
2.5 Set Theory and Statement Logic	12
3. FAULT TREE CONCEPTS	15
3.1 SAPHIRE Fault Tree Approach	15
3.2 SAPHIRE Fault Tree Symbols	17
3.2.1 Basic Event	18
3.2.2 Logic Gates	18
4. PROBABILITY CONCEPTS	23
4.1 Definition and Rules of Probability	23
5. DETERMINATION OF MINIMAL CUT SETS	27
5.1 Recursive Algorithms	28
5.2 Loading and Restructuring	28
5.3 N/M Gate Expansion	29
5.4 TOP Gate Determination	30
5.5 Logic Loop Error Detection	30
5.6 Complemented Gate Conversion	32

5.7	House Event Pruning.....	34
5.8	Coalescing Like Gates.....	36
5.9	Modules versus Independent Subtrees	38
5.10	Module Determination and Creation	39
5.11	Independent Gate, Subtree, and Event Determination.....	40
5.12	Determining Gate Levels	40
5.13	Fault Tree Reduction and Truncation	41
5.14	Intermediate Result Caching and Initialization	41
5.15	Fault Tree Gate Expansion.....	43
5.16	Cut Set Absorption	44
5.17	Boolean Absorption	45
5.18	Data Storage Considerations.....	45
5.19	Sequence Cut Set Generation.....	46
5.20	Parallel Processing During Cut Set Generation	48
6.	QUANTIFICATION TOOLS FOR PROBABILITIES AND FREQUENCIES	49
6.1	Quantifying Minimal Cut Sets	49
6.2	Quantifying Fault Trees	49
6.3	Quantifying Sequences	53
7.	EVENT PROBABILITY CALCULATION TYPES.....	55
7.1	Calculation Type 1.....	55
7.2	Calculation Type 3.....	56
7.3	Calculation Type 5.....	56
7.4	Calculation Type 7.....	56
7.5	Calculation Types T, F, and I.....	57
7.6	Calculation Type C	57
7.7	Calculation Type S	58
7.8	Calculation Type E	58
7.9	Calculation Type G.....	58
7.10	Calculation Type H.....	59
7.11	Calculation Type V	59
7.12	Calculation Type N.....	59
7.13	Calculation Type X	59

7.14	Calculation Types R, U, D, and Y	59
8.	IMPORTANCE MEASURES.....	61
8.1	Types of Importance Measures	61
8.2	Calculation Details.....	62
9.	UNCERTAINTY AND MONTE CARLO	69
9.1	Uncertainty and Monte Carlo	69
9.2	Basic Uncertainty Output.....	69
9.3	Uncertainty Analysis Input Data.....	71
9.4	Distributions	72
9.5	Histograms	82
9.5	Correlation Classes	83
9.6	Sampling Techniques.....	85
9.7	Inverse C.D.F Method	93
10.	SEISMIC EVENTS	95
10.1	Fragility and Component Failure Probabilities	95
10.2	Frequencies of Seismic Events	98
10.3	SAPHIRE Seismic Implementation.....	98
	10.3.1 The Hazard Curve	99
	10.3.2 Modeling Seismic Event and Fault Trees.....	99
	10.3.3 Basic Event Data - Fragilities and Uncertainty Data.....	100
	10.3.4 Generating and Quantifying Cut Sets.....	101
11.	COMPOUND EVENTS.....	103
11.1	General Structure of the Common Cause Failure Plug-in.....	104
11.2	Using the Common Cause Failure Plug-in.....	111
11.3	General Structure of the LOOP Recovery Plug-in.....	119
12.	PHASE MISSION CALCULATIONS	123
	12.1.1 General Phase Information	123
	12.1.2 Event Tree Phase or Applicability Modifications.....	124
	12.1.3 Analysis Capabilities Phase Modifications	124
13.	INTEGRATED MODEL SOLVING USING MODEL TYPES.....	133
14.	POST-PROCESSING RULES.....	135
15.	DEFINING END STATES USING PARTITIONING RULES.....	143
16.	WORKSPACE ANALYSIS ROUTINES	149
16.1	Multi-Pass Solution.....	149

16.2	Events and Condition Assessment Uncertainty Analysis	150
17.	USING MACROS IN SAPHIRE	153
18.	REFERENCES.....	169
	Appendix A - Fault Tree Quantification Example.....	A-1
	Appendix B - Common Cause Failure Calculations.....	B-1
	Appendix C - Maximum Entropy Distribution	C-1

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
Figure 1. Generalized Venn diagram representing two sets, A and B	5
Figure 2. The union of two sets illustrated via a Venn diagram	7
Figure 3. The intersection of two sets illustrated via a Venn diagram	8
Figure 4. The complement of a set illustrated via a Venn diagram	8
Figure 5. Mutually exclusive sets illustrated via a Venn diagram	9
Figure 6. Example of fault tree graphic (without descriptions) from the SAPHIRE editor	15
Figure 7. Example of a fault tree logic loop	17
Figure 8. Example use of a transfer gate	19
Figure 9. Equivalence for the NAND gate	20
Figure 10. Equivalence for the NOR gate.....	21
Figure 11. Event tree with fault tree linked to sequences where house events are used	36
Figure 12 Examples of independent a subtree and module fault tree.....	39
Figure 13 Example SAPHIRE event tree.....	46
Figure 14. Constrained noninformative alpha parameter as a function of the mean value.....	75
Figure 15. Example of range histogram	82
Figure 16. Illustration of the uncertainty on accident sequence probability results.....	86
Figure 17. Uncertainty distribution for Component A.....	87
Figure 18. Uncertainty distribution for Component B.....	87
Figure 19. Latin hypercube sample for Component A	89
Figure 20. Latin hypercube sample for Component B	89
Figure 21. Cells sampled in the LHS example.....	91
Figure 22. Cumulative distribution plots for example using Monte Carlo and LHS.....	92
Figure 23. Event and fault trees used for Case 1 calculation.....	126

Figure 24. Event and Fault Trees used for Case 2 Calculation	129
Figure 25. Graph of ECA uncertainty analysis.....	152

LIST OF TABLES

<u>Table</u>	<u>Page</u>
Table 1. SAPHIRE Calculation Types.....	55
Table 2. Standard deviation calculations for the supported uncertainty distributions.....	66
Table 3. Uncertainty distributions supported in SAPHIRE.....	73
Table 4. Conversion equations for various lognormal distribution parameters.....	79
Table 5. Random samples for the S system example.....	88
Table 6. Randomly selected intervals for the LHS example cells.....	90
Table 7. Randomly selected values for A and B using the LHS example.....	90
Table 8. Comparison of Monte Carlo and LHS for sample problem.....	92
Table 9. Applicable objects with post-processing rules.....	135
Table 10. Examples of search criteria used in the SAPHIRE post-processing rules.....	136
Table 11. Keywords utilized in the post-processing rule process in SAPHIRE.....	139
Table 12. Keywords utilized in the SAPHIRE end state partition rules.....	146
Table 13. Keywords for SAPHIRE Macros.....	154
Table 14. Example macro using the Demo project.....	166

EXECUTIVE SUMMARY

The Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) refers to a set of microcomputer programs that were developed to create and analyze probabilistic risk assessments (PRAs). Within this report, information is provided on the principles used in the construction and operation of Version 8.0 of the SAPHIRE system. Set theoretic operations and relations, their relation to Boolean logic, and the rules of probability are explained. Fault trees are reviewed, including all of the gate types allowed by SAPHIRE.

The procedure by which SAPHIRE builds a fault tree from the user inputs, simplifies and truncates it according to the user's specifications, and determines the minimal cut sets is outlined. Once cut sets are generated, "recovery" rules may be used to manipulate these cut sets. SAPHIRE 8 is written in a recursive language, and performs many operations by recursive procedures. To perform operations on logic models, SAPHIRE takes the user's input (a logic expression either by fault or event trees) and builds a simplified internal representation of the tree. This involves several steps:

- Linking portions that were connected by transfer gates,
- Expanding N/M gates as combinations of OR and AND gates,
- Determining the unique TOP gate,
- Checking for logical loops,
- Pruning portions of the tree having house events, and
- Coalescing like gates.

To obtain the minimal cut sets in an efficient way, SAPHIRE searches for special parts of the tree known as independent subtrees and modules, both of which are treated as single tokens until very late in the process. It then determines the optimal order for processing the tree, based on the levels of the gates, and begins making a list of cut sets. Based on the basic event probabilities or sizes (and the user's truncation specifications), SAPHIRE is able to eliminate some cut sets early in the process. It also eliminates non-minimal cut sets, those that can be absorbed by other simpler cut sets, and finally obtains a list of minimal cut sets that the user has specified should not be truncated. The last step is to combine the fault trees for failures of different systems, to obtain the fault tree for an accident sequence involving the failure of certain systems and the success of others, and then quantify the resulting cut sets.

Within the minimal cut sets are basic events – these events provide SAPHIRE with the probabilistic information needed to quantify the cut sets. Discussions related to the basic events includes the probability models provided by the SAPHIRE calculation types, how recovery events are applied, and how complex models may be used to determine event probabilities via the "compound" approach.

After cut sets are generated, uncertainty analyses may be performed by Monte Carlo sampling, with the basic event probabilities drawn from user-specified distributions. Two types of sampling are possible in SAPHIRE, simple Monte Carlo sampling and Latin Hypercube sampling. The distributions that are supported by SAPHIRE are presented, and the algorithms used for generating random numbers from these distributions are documented. Correlation

classes, allowing the user to state that certain basic event probabilities are equal, although both basic events are uncertain, are also explained. A simple example illustrates the two types of sampling.

Also included is an example showing the process SAPHIRE uses to find the minimal cut sets of a fairly simple fault tree, and how SAPHIRE finds the probability of the TOP event and the importance of the individual basic events. Specific calculation routines for Workspace analysis are also provided.

This report provides the SAPHIRE user with a basic understanding of the mathematical and probabilistic concepts needed to understand the basic principles used in the software. In addition, it gives an overview of the algorithms used in the program. This report is not intended to provide all of the details some readers may desire. Therefore, references are provided that contain more detail for the interested reader. However, information is provided on special analysis topics such as seismic and end state evaluations.

ACKNOWLEDGEMENTS

We would like to specifically acknowledge Mr. Dan O'Neal of the U.S. Nuclear Regulatory Commission for his contribution to the development this report.

ACRONYMS

BPC	bound on the probability contribution
CDF	cumulative distribution function
DOD	Department of Defense
DOE	Department of Energy
EF	<i>error factor</i>
FEP	Fault Tree, Event Tree, and Piping and Instrumentation Diagram
GEM	Graphical Evaluation Module
HEP	human error probability
HRA	human reliability analysis
INEEL	Idaho National Engineering and Environmental Laboratory
INL	Idaho National Laboratory
IRRAS	Integrated Reliability and Risk Analysis System
LHS	Latin Hypercube Sampling
MAR-D	Models and Results Database
NRC	Nuclear Regulatory Commission
PC	<i>personal computer</i>
PRA	probabilistic risk analysis
PSF	performance shaping factor
RAW	risk achievement worth
SAPHIRE	Systems Analysis Programs for Hands-on Integrated Reliability Evaluations
SARA	System Analysis and Risk Assessment
SPAR	Standardized Plant Analysis Risk

Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Version 8

Volume 2 Technical Reference

1. INTRODUCTION

1.1 Background

The U.S. Nuclear Regulatory Commission (NRC) has developed a powerful personal computer (PC) software application for performing probabilistic risk assessments (PRAs), called Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Version 8.

Using SAPHIRE 8 on a PC, an analyst can perform a PRA for any complex system, facility, or process. Regarding nuclear power plants, SAPHIRE can be used to model a plant's response to initiating events, quantify associated core damage frequencies, and identify important contributors to core damage (Level 1 PRA). It can also be used to evaluate containment failure and release models for severe accident conditions, given that core damage has occurred (Level 2 PRA). It can be used for a PRA assuming that the reactor is at full power, at low power, or at shutdown conditions. Furthermore, it can be used to analyze both internal and external initiating events, and it has special features for transforming models built for internal event analysis to models for external event analysis. It can also be used in a limited manner to quantify risk for release consequences to both the public and the environment (Level 3 PRA). For all of these models, SAPHIRE can evaluate the uncertainty inherent in the probabilistic models.

SAPHIRE development and maintenance has been undertaken by the Idaho National Laboratory (INL). The INL began development of a PRA software application on a PC in the mid 1980s when the enormous potential of PC applications started being recognized. The initial version, *Integrated Risk and Reliability Analysis System* (IRRAS), was released by the Idaho National Engineering Laboratory (now Idaho National Laboratory) in February 1987. IRRAS was an immediate success, because it clearly demonstrated the feasibility of performing reliability and risk assessments on a PC and because of its tremendous need (Russell 1987). Development of IRRAS continued over the following years. However, limitations to the state of the-art during those initial stages led to the development of several independent modules to complement IRRAS capabilities (Russell 1990; 1991; 1992; 1994). These modules were known as Models and Results Database (MAR-D), System Analysis and Risk Assessment (SARA), and Fault Tree, Event Tree, and Piping and Instrumentation Diagram (FEP).

IRRAS was developed primarily for performing a Level 1 PRA. It contained functions for creating event trees and fault trees, defining accident sequences and basic event failure data, solving system fault trees and accident sequence event trees, quantifying cut sets, performing sensitivity and uncertainty analyses, documenting the results, and generating reports.

MAR-D provided the means for loading and unloading PRA data from the IRRAS relational database. MAR-D used a simple ASCII data format. This format allowed interchange of data

between PRAs performed with different types of software; data of PRAs performed by different codes could be converted into the data format appropriate for IRRAS, and vice-versa.

SARA provided the capability to access PRA data and results (descriptive facility information, failure data, event trees, fault trees, plant system model diagrams, and dominant accident sequences) stored in MAR-D. With SARA, a user could review and compare results of existing PRAs. It also provided the capability for performing limited sensitivity analyses. SARA was intended to provide easier access to PRA results to users that did not have the level of sophistication required to use IRRAS.

FEP provided common access to the suite of graphical editors. The fault tree and event tree editors were accessible through FEP as well as through IRRAS, whereas the piping and instrumentation diagram (P&ID) editor was only accessible through FEP. With these editors an analyst could construct from scratch as well as modify fault tree, event tree, and plant drawing graphical figures needed in a PRA.

Previous versions of SAPHIRE consisted of the suite of these modules. Taking advantage of the Windows 95 (or Windows NT) environment, all of these modules were integrated into SAPHIRE Version 6; more features were added; and the user interface was simplified.

Work began on a new version of SAPHIRE, Version 8, in 2004. Version 8 was designed to meet current NRC program needs such as those related to SPAR model development, the Significance Determination Process (SDP) program, the Risk Assessment Standardization Project (RASP), as well as the Accident Sequence Precursor (ASP) Program. The development of the SAPHIRE 8 version includes new features and capabilities. These features and capabilities are related to working with larger, more complex models and improving the user-friendliness of SAPHIRE's interfaces while retaining key functionality of Version 7.

Version 8 is being developed to support the SPAR models and to run them as an integrated model (e.g., Level 1 with external events). The graphical user interface has also improved from SAPHIRE 7. A tailored interface for the SDP and the ASP programs has been developed. The interfaces for the SDP, ASP, and general analysis introduce the concept of a "workspace" in which the analyst may run and save different analyses. The use of workspaces enables the user to separate the model construction from the model analysis.

The report contains the following topics:

- An introduction to sets and set operations and to the corresponding logical operations
- A review of fault tree construction principles and the philosophy used in SAPHIRE
- An overview of probability theory
- An overview of cut set algorithms in SAPHIRE, including post-processing via recovery rules
- A review the quantification techniques used in SAPHIRE

- A summary of the calculation types used for the basic events
- An overview of importance measures, including uncertainty importance
- A discussion of the uncertainty analysis and an introduction to Monte Carlo sampling and Latin Hypercube sampling
- An overview of seismic events and general external events applications
- An overview of phased-mission types of evaluations
- An review of the use of compound events, including the common-cause module in SAPHIRE
- A discussion of the methods used to treat analysis on event tree end-states.
- A list of applicable references
- An example of the details of an SAPHIRE analysis to a simple fault tree

2. SET THEORETIC AND LOGICAL CONCEPTS

This section presents basic definitions of sets and a summary of useful identities. The reader can obtain more information from Hahn and Shapiro (1967), Mood et al. (1974), Henley and Kumamoto (1985), or Andrews and Moss (2002). Within this section, we present several topics, including: set theory and set operation concepts; listing of useful Boolean identities; concepts of statement logic; and the relations between set theory and statement logic.

2.1 Set Theoretic Concepts

A useful tool to illustrate set relations pictorially is the Venn diagram. Figure 1 shows the Venn diagram (Venn, 1880) for two sets, A and B , where B is a proper subset of A .

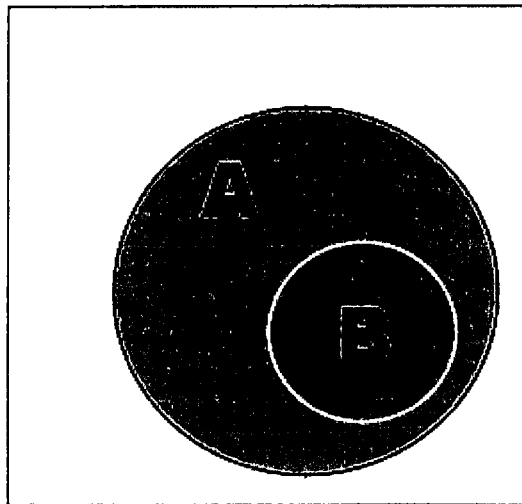


Figure 1. Generalized Venn diagram representing two sets, A and B

For SAPHIRE, we are interested in what could occur at a facility such as a nuclear power plant, chemical processing facility, or transportation system. Therefore, when set theory is used for SAPHIRE applications, we usually let the population Ω consist of all possible conditions of the plant or system. Any one element of this set consists of a detailed specification of the condition of every part of the plant or system. Consequently, Ω has a huge number of elements.

Events are subsets of this population. For example, an event such as "AFW pump B fails to start" is a subset, consisting of all conditions of the pump and its supporting equipment that result in failure to start, together with all possible conditions of the rest of the plant. The event "core melt" is also a subset of the population, containing all the detailed plant conditions that result in core melt.

In this document, we will note definitions by enclosing the terms and associated definition in a box, such as those below:

Set:	A set is a user-defined collection of objects.
Elements:	Objects belonging to a set.
Population:	A set of all possible elements.
Null Set:	A set that has no elements. It is also called the empty set and is denoted by Φ .
Subset:	A set A is called a subset of set B if <i>all</i> the elements of A are also elements of B . If set B contains addition elements, then set A is called a proper subset.
Set Equality:	A set A is equal to set B if they contain the same elements.

2.2 Operations on Sets

Three basic operations exist for sets. They are:

- Union
- Intersection
- Complementation

A fourth operation, called set difference, is sometimes considered; it is expressed as a combination of the other set operations. See Smith, Knudsen, and Calley (1999) for additional information on set differences.

The *union* of two sets is a set consisting of all the distinct elements in A or all of the elements in B or both. The union operation is also called an OR operation, and is sometimes denoted by $C=A+B$.

The union is denoted by $C = A \cup B$

Inexperienced analysts are wise always to use the symbol \cup to combine sets and the symbol $+$ to combine numbers, but adept symbol jugglers learn to use $+$ safely in both contexts. Computer programs that use only the 128 ASCII characters or the characters on a line printer

are forced to use + instead of U. The reader of this document should realize that when dealing with sets, use of the + symbol for sets implies the logical union of the sets.

The union of two sets is shown in Figure 2.

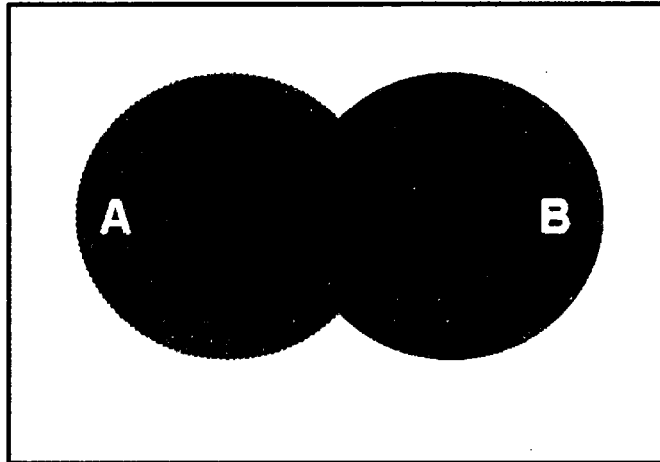


Figure 2. The union of two sets illustrated via a Venn diagram

The union of any number of sets A_1, A_2, \dots is the set of all elements that are in any of the A_i 's. It can be written with notation analogous to summation notation:

$$\bigcup_{i=1}^n A_i \quad \text{for } n \text{ sets and}$$

$$\bigcup_{i=1}^{\infty} A_i \quad \text{for infinitely many sets.}$$

The *intersection* of two sets is the set consisting of all the elements common to both A and B . That is, the elements belong to A and to B . It is also called the AND operation.

**The intersection is denoted by $C = A \cap B$
or sometimes $C = A * B$
or simply, $C = AB$.**

The intersection of two sets is shown as the AB region in Figure 3.

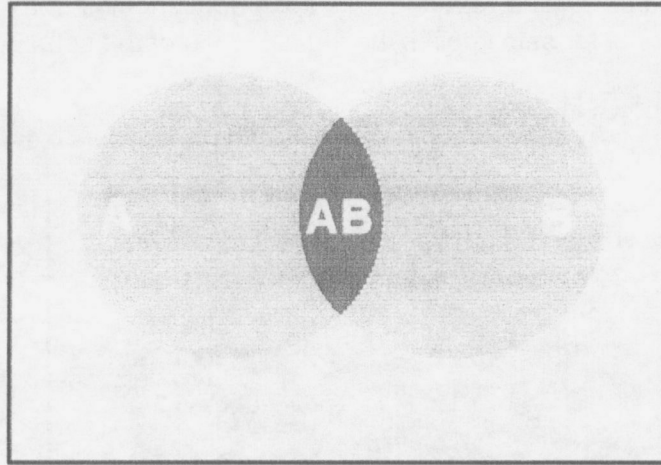


Figure 3. The intersection of two sets illustrated via a Venn diagram

The intersection of A_1, A_2, \dots is the set of all elements that are in all the A_i 's. The intersection of n sets can be written as:

$$\bigcap_{i=1}^n A_i$$

or, using product notation, as $A_1 A_2 \dots A_n$.

The *complement* of a set A is the set consisting of all elements in the population that are not contained in A . It is sometimes called the NOT operation. It is denoted by a bar over A , A' , or A^c . The complement of a set is shown in Figure 4.

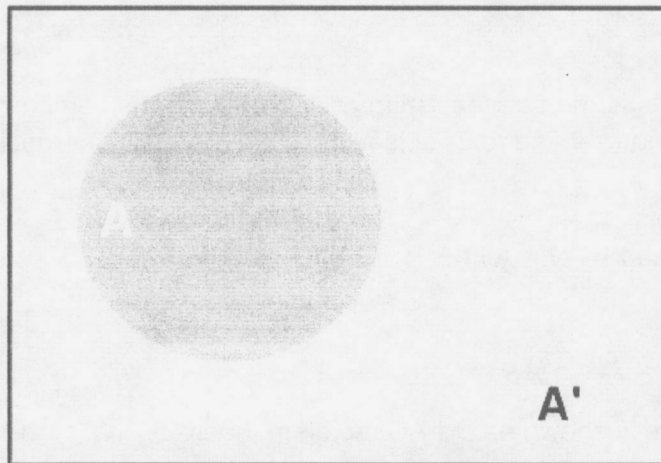


Figure 4. The complement of a set illustrated via a Venn diagram

The set of all elements in A and not in the set B is called the set *difference*.

The set difference is denoted by $A \cap B'$

It can also be written as $A-B$. The lighter portion of set A (shown in Figure 3) represents the set difference $A-B$.

Two sets are said to be *mutually exclusive* or *disjoint* if and only if they contain no elements in common.

The intersection of mutually exclusive sets is the null set, $A \cap B = \Phi$.

Mutually exclusive sets are shown in Figure 5.

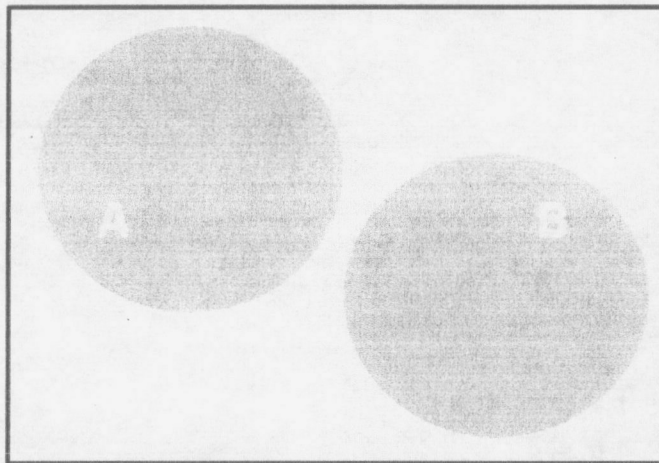


Figure 5. Mutually exclusive sets illustrated via a Venn diagram

The sets A_1, A_2, \dots are mutually exclusive if each pair is mutually exclusive, that is, no element of W is in more than one A_i . The term "mutually exclusive" can therefore refer even to an infinite collection of sets.

A collection of sets A_1, A_2, \dots is *exhaustive* if the union of the sets is the population Ω , that is, every element of Ω is in at least one A_i . In most applications, exhaustive sets are also chosen to be mutually exclusive. When the sets A_1, A_2, \dots are both mutually exclusive and exhaustive, they form a *partition* of Ω : every element of Ω is in one and only one of the A_i 's.

2.3 Summary of Useful Identities

The following are useful identities in working with sets:

Commutative Laws

$$A \cup B = B \cup A$$

$$A \cap B = B \cap A$$

Associative Laws

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

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

Distributive Laws

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

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

Idempotent Laws

$$A \cap A = A$$

$$A \cup A = A$$

Laws of Absorption

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

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

Complementation

$$A \cap A' = A \cap A^c = A \cap \mu = \Phi$$

$$A \cup A' = A \cup A^c = A \cup \mu = \Omega$$

$$(A')' = (A^c)^c = A$$

Operations Involving Null Set and Population

$$\Phi \cap A = \Phi$$

$$\Phi \cup A = A$$

$$\Omega \cap A = A$$

$$\Omega \cup A = \Omega$$

$$\Phi' = \Phi^c = \Omega$$

$$\Omega' = \Omega^c = \Phi$$

DeMorgan's Laws

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

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

Other Identities

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

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

2.4 Concepts of Statement Logic

A *statement* is defined here as a sentence that can be declared either true or false. Examples are "generator DG1 fails to start" and "rocket nozzle structural integrity is maintained." English

statements that are not clearly true or false, such as "This is a nice looking control panel," are not considered. Mathematically, a statement is an object that can take one of two values, either TRUE or FALSE. In this discussion, we will use the letters p , q , r , etc. to denote statements. New, more complex statements can be built by combining simpler statements using AND, OR, and NOT, defined as follows:

$(p \text{ AND } q)$ is TRUE if
both p and q are TRUE.

It is FALSE if
 p is FALSE, q is FALSE, or both are FALSE.

$(p \text{ OR } q)$ is TRUE if
 p is TRUE, q is TRUE, or both are TRUE.

It is FALSE if
both p and q are FALSE.

$(\text{NOT } p)$ is TRUE
if p is FALSE.

It is FALSE if
 p is TRUE.

For ease of input from a computer, SAPHIRE uses the notation "/" for NOT. That is $/X$ is the notation for NOT X in SAPHIRE input.

Working from the previous basic definitions, one can prove many simple facts about statements, similar to those listed for sets. For example, one distributive law says

$$p \text{ AND } (q \text{ OR } r) = (p \text{ AND } q) \text{ OR } (p \text{ AND } r)$$

and one of DeMorgan's laws says

$$\text{NOT } (p \text{ AND } q) = (\text{NOT } p) \text{ OR } (\text{NOT } q).$$

These equations mean that the statement on the left-hand side is TRUE if and only if the statement on the right-hand side is TRUE. Mathematics that uses the formal manipulation of these logical relations is sometimes called *Boolean*, after the mathematician George Boole.

2.5 Set Theory and Statement Logic

Parallel structures for sets and for statements exist: the terms AND, OR, and NOT were used for both, and similar rules such as the distributive laws and DeMorgan's laws applied to both. The relation is made explicit here.

Let Ω be the population, and consider statements about the elements of Ω . Any statement has a corresponding *truth set*, defined as the set of all elements for which the statement is true. An element is in the truth set if and only if the statement is true for that element. For example, the statement "core melt occurs" corresponds to the set of all possible plant conditions that result in core melt. Suppose that

A is the set of elements for which p is TRUE

B is the set of elements for which q is TRUE.

Then the rules for combining sets and for combining statements are related as follows:

$A \cup B$ is the set of elements for which (p OR q) is TRUE

$A \cap B$ is the set of elements for which (p AND q) is TRUE

A' is the set of elements for which (NOT p) is TRUE.

Because the correspondence between set and statement definitions is so direct, we sometimes interchange the languages: **A OR B** instead of **$A \cup B$** .

For SAPHIRE applications, the statements of interest describe events. For example, the event "AFW pump B fails to start" may be thought of as a statement p that can be combined with other statements as described in the *Concepts of Statement Logic* section. The event *occurs* if the statement defining the event is TRUE. This defines an event as a statement.

Alternatively, the event can be thought of as naming the set A of all plant conditions that result in failure of the pump to start. Similarly, the statement "MOV134 fails to open" can be thought of as corresponding to a set B of plant conditions.

The statement that both these events occur, MOV134 fails to open AND AFW pump B fails to start, corresponds to the intersection $B \cap A$.

The relation between statements and sets is so direct that most people switch back and forth between the two without even realizing it. This is why the terms AND, OR, and NOT were introduced in *Operations on Sets* as alternative terms for intersection, union, and complementation. The terminology in this document allows for this back-and-forth thinking, not carefully distinguishing between statement logic and set theory.

One reason we did not list all the facts about statements is that they are simply re-expressions of the facts in the *Summary of Useful Identities* section. Any fact about sets in the *Summary of*

Useful Identities section can be translated to a fact about statement logic by replacing set symbols with statements:

<i>Set Symbols</i>	<i>Statement Logic</i>
$A, B,$ and C	$p, q,$ and r
$\cup, \cap,$ and $'$	OR, AND, and NOT
The population Ω	always true
The null set Φ	always false

3. FAULT TREE CONCEPTS

This section provides the reader with an overview of the concepts used by SAPHIRE in the creation of fault tree models. More information can be found in Vesely et al. (1981). In this section, we will address two topics; first a general overview of the SAPHIRE fault tree approach is described followed by the symbols used in the graphical editor.

3.1 SAPHIRE Fault Tree Approach

SAPHIRE allows the user to input fault tree models in either of two ways: graphically (as shown in Figure 6) or alphanumerically (shown below) via a file import option.

TOPGATE	OR	EVENT - A	GATE - 2	GATE - 1
GATE - 1	OR	GATE - 3	EVENT - B	
GATE - 2	AND	EVENT - D	EVENT - C	
GATE - 3	AND	EVENT - E	EVENT - G	EVENT - F

Both methods produce equivalent results and use the same basic approach to modeling.

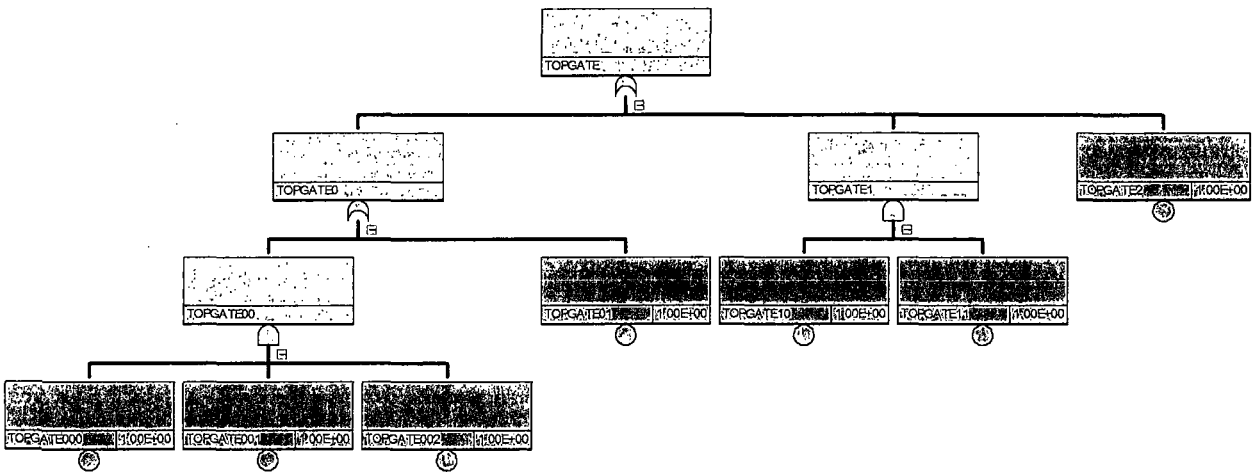


Figure 6. Example of fault tree graphic (without descriptions) from the SAPHIRE editor

A *fault tree* model consists of a *top event* (usually defined by a heading in an event tree) and a connecting logic structure that models the combinations of events that must take place to result in the undesired top event. In SAPHIRE, a **fault tree generally represents a failure model**. Thus, all the elements in the fault tree represent failures, whether they are equipment failures, human errors, or adverse conditions that can contribute to failure of the modeled event. Successful events (those things that should happen) that can contribute to failure of the

top event can be included in the fault tree also, but special care must be exercised. Specifically, success events should be names such that they begin with the "/" character (e.g., success of EVENT-B would be entered as /EVENT-B).

The logic structure must contain only one top event. SAPHIRE will provide an error message if more than one top event is discovered. A simple way to guarantee only one top event per fault tree is to develop the fault tree model from the top down and complete each level of the fault tree model before proceeding to the next level.

The fault tree logic structure can consist of any combination of the logic symbols shown below that do not result in a logical loop. All the symbols below are buttons and are located in the graphical tool bar of the Fault Tree graphical editor.



BE This symbol represents a **basic event** such as a hardware failure, human error or an adverse condition.



AND Represents an AND gate. The logic operation for this gate requires all inputs into the AND gate must occur for failure to occur.



OR Represents an OR gate. The logic operation for this gate requires only one of the total number of inputs into the OR gate to occur for failure to occur.



N/M Represents an "N-OUT-OF-M" gate (or N/M gate). The logic operation for this gate requires that N of the M inputs into the gate must occur for failure to occur. For example, for a 2/4 gate, any combination of 2 of the 4 input events must occur for a failure to occur.



TRAN This symbol represents a TRANSFER gate. The transfer gate indicates that logic is continued from some other location. The location could be a new separate fault tree (top event) or a gate from some other location in the same fault tree.

A *logical loop* is a chain of events that comes back on itself. For example, a service water system can fail due to a loss of electrical power. Part of the electric power model contains failure of the emergency diesel generators. The emergency diesel generators can fail due to a loss of cooling water supplied by the service water system. The combination of events resulting in the loss of service water due to a loss of electrical power caused by failure of the diesel generators that was due to the loss of service water is a logical loop (see Figure 7).

3.2.1 Basic Event

A basic event represents a simple failure or fault. It may be a hardware failure, a human error, or an adverse condition. Hardware failures are usually expressed in terms of a specific component and a failure mode, such as "Service Water Pump 1A fails to start on demand." Human errors can be failure to carry out a desired task (failure to open a valve), failure to perform a specific recovery action (failure to start a backup system), or execution of a wrong action that has adverse effects on the fault tree top event (isolated the source of water for a cooling system). An adverse condition is not necessarily a failure but in combination with other events can lead to failure. For example, the temperature being below 32°F is an adverse condition necessary for the failure of flow reduction due to a frozen pipe.

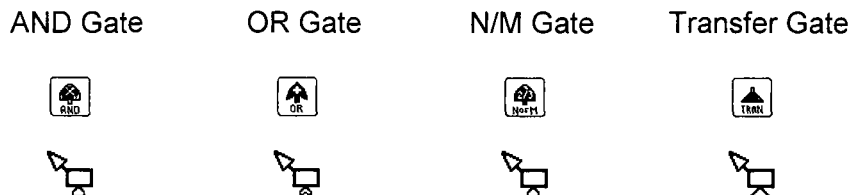
Even though a basic event does not necessarily describe a failure, the vast majority of basic events are failures. This leads to lax but understandable language such as "the event is in the failed state" instead of the more correct "the event occurs."

Basic events are always assumed to be independent of each other, in the statistical sense defined in the section on Independent Events. This means that the occurrence of one basic event does not influence the probability of occurrence of any other basic event. For example, suppose that there are two diesel generators, and the failure of either to start on demand is a basic event. Independence of the basic events says that if one diesel generator fails to start on demand, this does not alter the probability that the second diesel generator will fail to start.

A common cause event, such as "two diesel generators fail to start because of unusually cold weather," must be modeled as its own basic event, and be assigned its own failure probability or failure rate. This event is then regarded as statistically independent of all other basic events.

3.2.2 Logic Gates

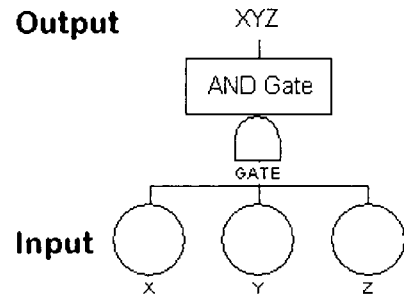
Logic gates are used to indicate how the basic events must combine to result in failure of the top event. Every logic gate has one or more inputs at the bottom and an output at the top. Inputs may be basic events or other logic gates. The output must serve as the input to another logic gate or result in the top event. Each logic gate derives its name from the manner in which the inputs must combine to pass through it to the next level. The input to a logic gate is a set of events. The output is a single event, formed by using the set operations AND and OR on the input events. The logic gates (and the icon and mouse cursor symbol) in SAPHIRE 8 are:



In addition to the regular gates, the AND and OR gates may be complemented, resulting in a NOT AND (NAND) and a NOT OR (NOR) gate, respectively.

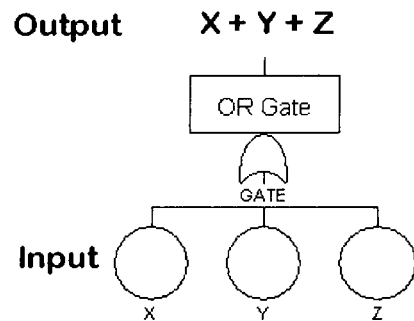
3.2.2.1 AND Gate

This gate states that the output event is the simultaneous occurrence of all the input events. In set language, the output set is the intersection of the input sets. In terms of statement logic, the output is a compound statement ($X \text{ AND } Y \text{ AND } Z$).



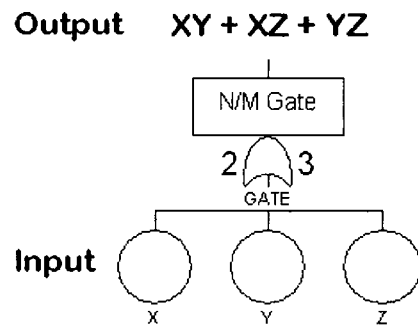
3.2.2.2 OR Gate

This gate combines the inputs by the OR operation. The output set is the union of the three input sets. Alternatively, the output statement is $X \text{ OR } Y \text{ OR } Z$.



3.2.2.3 N/M Gate

This gate states that N of the M input events occur. It is sometimes called an N-out-of-M gate or a *combination gate*. For a 2/3 gate, illustrated here, 2 of the 3 input events must occur. The output statement is $(X \text{ AND } Y) \text{ OR } (X \text{ AND } Z) \text{ OR } (Y \text{ AND } Z)$.



3.2.2.4 Transfer Gate

This gate does not require any special logic to result in an output, rather it is used to link logic structures together without introducing any new logic of its own. This is used primarily as a convenience for the modeler. All but the simplest of fault trees take up more than one page. The TRANSFER GATE indicates where the logic on a given page is continued on another page. A TRANSFER GATE may also be used to indicate where the logic is continued on the same page. The symbol for the transfer gate is shown in Figure 8.

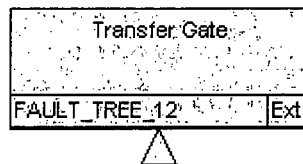


Figure 8. Example use of a transfer gate

- The TRANSFER GATE name must be the same as the name of the gate where the logic continues.
- When transferring on the same page, the gate being transferred to can be anywhere on the page, except where it would create a logic loop.
- When transferring to another page, the gate being transferred to must be the top gate on the page.
- When transferring to another page, the transfer gate name, the file name for the page being transferred to and the name of the gate being transferred to must all be the same. For example, if the TRANSFER GATE is called TRANS1, then the page being transferred to must be called TRANS1 and the top gate on that page must be called TRANS1.

3.2.2.5 NOT AND Gate

This gate is also called a NAND GATE. It can be thought of as the negation of an AND GATE. The output occurs if any one of the inputs does not occur. This is best explained through an example.

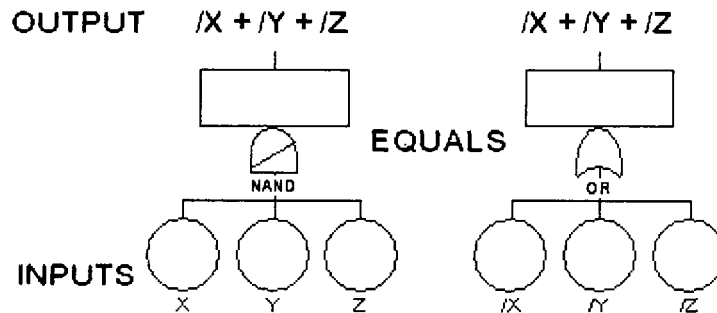


Figure 9. Equivalence for the NAND gate

The left side of Figure 9 shows a NOT AND GATE with inputs X, Y, and Z. If any one of the inputs does not occur, then an output occurs. Any of three possibilities satisfy this condition: 1) X does not occur, 2) Y does not occur, or 3) Z does not occur. Since any event (X) and its complement ($/X$) are mutually exclusive, we can say that

$$X \text{ does not occur} = /X \text{ occurs.}$$

Therefore, the output of the NOT AND GATE in Figure 10 is $/X$ (read not X), or $/Y$, or $/Z$.

Another way of looking at the problem is the way SAPHIRE actually processes a NOT AND GATE. The gate is transformed into an OR GATE with all of the inputs transformed into their complements. Any single complement event occurring results in an output.

3.2.2.6 NOT OR Gate

This gate is also called a NOR GATE. It is the negation of an OR GATE. The output occurs if none of the inputs occur. This is shown in Figure 10. There is only one combination of events where none of the inputs occur; X does not occur and Y does not occur and Z does not occur. In terms of complemented events this is \bar{X} and \bar{Y} and \bar{Z} .

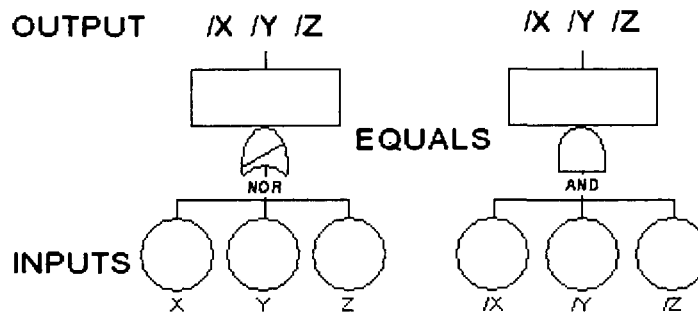


Figure 10. Equivalence for the NOR gate

SAPHIRE processes a NOT OR GATE by transforming it into an AND GATE with all of the inputs transformed into their complements. All of the not events must occur for the output event to occur. This is the same as none of the original events occurring. The other symbols in a fault tree are used to add clarity to the diagram and to connect the various gates and events together properly.

4. PROBABILITY CONCEPTS

This section provides the reader with an overview of the concepts of probability associated with the uncertainty analysis used in PRA. This discussion will not be inclusive, but it will present the basic concepts and principles. For a more detailed discussion of these topics, the reader can obtain more information from Press (1989), Lindley (1985), Singpurwalla (1988), Jaynes (2003), USNRC (2003), and NASA (2009).

4.1 Definition and Rules of Probability

Probability is the only satisfactory way to quantify our uncertainty about an uncertain event E. Probability is always *conditional*; it is conditioned on all of the background information we have at the time we are quantifying our uncertainty. This background information is denoted by H and the probability of E conditional on H is denoted by $P(E|H)$. To make the notation less cumbersome, we write this simply as $P(E)$; nevertheless, the conditioning H should be understood.

The range of a probability is between 0 and 1. $P(E) = 0$ means E will never occur, and $P(E) = 1$ means E will always occur. From now on, assume that a probability is defined for all events in the population (where Ω consist of all possible conditions).

The rules of probability tell us how to relate our uncertainty about events. Specifically, they tell us how various probabilities combine or cohere. These rules are motivated by preferences between events and a scoring rule argument. The scoring rule approach can be used to show that the following three rules of probability hold for discrete cases.

For any event,

$$0 \leq P(E) \leq 1, \text{ and } P(\Omega) = 1 \quad (4-1)$$

For any mutually exclusive events E_1, E_2, \dots

$$P\left(\bigcup_{i=1}^{\infty} E_i\right) = \sum_{i=1}^{\infty} P(E_i) \quad (4-2)$$

The *conditional probability* of an event F given an event E is

$$P(F|E) = P(F \cap E) / P(E) \quad (4-3)$$

which is equivalent to the multiplication rule

$$P(F \cap E) = P(F|E) P(E)$$

These are the basic rules of probability, from which all others can be derived. One logical development of probability, due to Kolmogorov (Press 1989), takes Equations (4-1) and (4-2) as axioms, and Equation (4-3) as a definition. A more recent approach by Renyi (Press 1989) uses conditional probability as the fundamental concept, rewrites every unconditional probability above as a conditional one, and uses the rewritten Equations (4-1), (4-2) and (4-3) as axioms. These mathematical fine points are not important to this report. It is enough to note that every treatment of probability uses the rules given above, and the rules that follow as consequences in the sections below.

Equation (4-2) says that the probability of the union of disjoint events is the sum of the probabilities. This fact motivated the use of + as an alternate notation for U in Section 2.2.

Law of Total Probability

For any events E and F,

$$P(E) = P(E \cap F) + P(E \cap F^c) = P(E|F) P(F) + P(E|F^c) P(F^c)$$

This law can be extended to a set of n mutually exclusive and exhaustive events F_1, F_2, \dots, F_n as follows:

$$P(E) = \sum_{k=1}^n P(E | F_k) P(F_k) \quad (4-4)$$

Basic Probability Relations

$$P(\Omega) = 1$$

$$P(\emptyset) = 0$$

$$P(\bar{A}) = P(A^c) = 1 - P(A)$$

$$P(A \cup \bar{A}) = P(A \cup A^c) = P(\Omega) = 1$$

$$P(A \cap \bar{A}) = P(A \cap A^c) = P(\emptyset) = 0$$

If E and F are two events and E is a subset of F, then $P(E) < P(F)$.

Bayes' Theorem

Consider any two events E and F. By the multiplication law

$$P(E \cap F) = P(E|F) P(F) = P(F|E) P(E).$$

so

$$P(E|F) = \frac{P(F|E)P(E)}{P(F)} \quad (4-5)$$

We use Equation (4-5) to change our uncertainty about E given background information H to our uncertainty about E given F and H. We can think of F as new data.

For example, suppose that turbine-driven pumps fail to start with some frequency p . We quantify our background knowledge about turbine-driven pumps through a probability distribution on p . (For ease of explanation, suppose that this distribution is discrete, a list of possible values p_i , each with a probability reflecting our degree of belief.)

To continue this example, let E be the event " $p = 0.01$ ". Let F be the event "3 failures in 100 attempts to start." We know $P(E)$ from the probability distribution that quantifies our background knowledge. How should this probability be changed to account for the new information? That is, what is $P(E|F)$?

This question is answered using Bayes' Theorem. The theory of binomial random variables shows that

$$P(F|p) = \binom{100}{3} p^3 (1-p)^{100-3}$$

is the probability of the event F given some value of p . Therefore $P(F|E)$ is $P(F|p)$ with the value 0.01 substituted for p . The value of $P(F)$ is obtained from the law of total probability, Equation (4-4):

$$P(F) = \sum P(F|P_i)P(p = p_i)$$

summed over all the possible values p_i . Then finally, $P(E|F)$ is obtained by substituting the values for

$P(E)$, $P(F|E)$, and $P(F)$ into Equation (4-5).

In summary, we used Equation (4-5) to change a belief about E given the background information to a belief about E given both the background information and F. The belief was updated based on new data. Additional information on Bayesian methods may be found in Jaynes (2003), USNRC (2003), and NASA (2009).

Independent Events

We say an event E is *independent* of another event F if the probability of E, $P(E)$, is unaltered by any information concerning event F. We write

$$P(E|F) = P(E|F') = P(E) .$$

This is also called *statistical independence*. From this definition we obtain the following relationship for independent events

$$P(E \cap F) = P(E|F)P(F) = P(E)P(F) .$$

Beginners often confuse mutually exclusive events with independent events. The two concepts are different: Mutually exclusive events satisfy

$$P(A \cap B) = P(\Phi) = 0$$

whereas independent events satisfy

$$P(A \cap B) = P(A) P(B) .$$

Therefore, if $P(A) > 0$ and $P(B) > 0$, A and B cannot be both mutually exclusive and independent. The mutual exclusiveness introduces a dependence, if A occurs, B cannot occur.

Additional Probability Relations

The probability of the union of n events is

$$P(A_1 \cup A_2 \cup \dots \cup A_n) = \sum P(A_i) - \sum_{i < j} P(A_i A_j) + \dots + (-1)^{n+1} P(A_1 A_2 \dots A_n) \quad (4-6)$$

The probability of the intersection of n events is

$$P(A_1 A_2 \dots A_n) = P(A_n | A_1 \dots A_{n-1}) \dots P(A_2 | A_1) P(A_1) \quad (4-7)$$

The probability of the intersection of n events when the events are statistically independent is

$$P(A_1 A_2 \dots A_n) = P(A_1) P(A_2) \dots P(A_n) \quad (4-8)$$

For any n events (dependent or independent), we have

$$P(A_1 A_2 \dots A_n) \leq \min[P(A_1), P(A_2), \dots, P(A_n)] \quad (4-9)$$

For independent events, the probability of the intersection equals the product of the probabilities. This fact motivated the product notation that was introduced as an alternate to the intersection in the discussion on the intersection of sets. Because of its compactness, the product notation has been used for intersections in Equations (4-6) through (4-9).

5. DETERMINATION OF MINIMAL CUT SETS

When considering the development of a fault tree minimal cut set algorithm, it is good to review the general processes involved. First, we have the definition of the fault tree logic. Typically, the logic is defined using an alphanumeric file containing names of gates and basic events. Along with the logic file is another alphanumeric file containing basic event names and a failure probability associated with each event. These failure probabilities are used during the fault tree solution process to simplify the tree by truncation. Additional processing information may be used, but this is typically the minimum information required.

The above information is loaded into the computer's memory and converted into a format that is easier to process. For example, basic event names are converted to 4-byte long numbers for smaller size and ease of manipulation. Certain optimization functions are also performed on the logic before it is processed. Next, the logic for each gate starting with the TOP is recursively replaced with its inputs until the resulting logic is in terms of basic events only. This results in a list of event intersections. Each event intersection is a *cut set* of the fault tree and identifies a set of events that will cause the function modeled by the fault tree to occur. The list of cut sets identifies all the logical combinations of events that will cause the top event to occur.

The cut sets described above may need further reduction due to rules defined for Boolean reduction. These reductions are applied to obtain a simpler collection of cut sets. For example, the cut sets generated should be *minimal*, that is, the list should not be simplifiable.

For example, if $A \cap B \cap C$ causes the top event to occur, then $A \cap B \cap C$ is a cut set. If $A \cap B$ is also a cut set, then $A \cap B \cap C$ is not minimal, and it is discarded from the list. If neither A alone nor B alone causes the top event to occur, $A \cap B$ is a minimal cut set, and it is retained in the list. This is an application of the absorption identity:

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

The event probabilities are then used to calculate a probability for each cut set using Equation (4-7). This value is the probability that the given set of events will occur. Any cut set whose probability falls below a user-defined (truncation) value is then eliminated. The remaining cut sets are the minimal cut sets for the fault tree and are the desired end product of the fault tree solution. In SAPHIRE, the minimal cut sets are always in terms of basic events unless the analyst specifically indicates that certain gates are to be treated as basic events.

Once the minimal cut sets have been determined, the quantification routines must be employed to determine a point estimate for the probabilities of the cut sets. The routines that find importance measures would then be used to calculate the importance of each basic event in the cut sets, and the uncertainty routines would be used to perform uncertainty analysis on the cut sets.

The steps described above need not be applied in the order indicated, but each step is usually present in any fault tree software. We will now present a more detailed overview of each of these steps as they relate to SAPHIRE.

In order to solve a fault tree, there are a number of operations that must be performed on the tree before it can be solved. Some of these operations relate to converting the tree into a format that is ready to solve, while others involve optimizing the tree to make the processing of the tree more efficient.

5.1 Recursive Algorithms

Many of the processes associated with fault tree reduction and quantification can be implemented easily using recursive procedures. A simple definition of a *recursive* procedure is "a procedure that calls itself."

An example of where a recursive procedure might be used is in checking a gate for "valid" inputs. A recursive implementation of this procedure has as an argument, the gate to be checked. This procedure checks each input to the gate passed as an argument. If an input is a basic event, then it checks to see if it is valid. If the input is a gate, however, it calls itself to see if the inputs to this gate are valid. When all the inputs to a gate have been processed, the procedure exits and continues processing the gate it was checking before the recursive call. The algorithm stops when all inputs to all gates have been checked.

Many computer languages do not support recursive procedures, but in those languages recursion can be simulated by using arrays to keep track of the arguments passed to the procedure. SAPHIRE takes advantage of recursive procedures in many areas due to its reliance on the Modula-2 and Delphi programming languages.

5.2 Loading and Restructuring

SAPHIRE was designed to allow the user to structure very large fault trees into smaller pieces or pages. The concept of *pages* comes from the graphical fault tree editor. One page represented the portion of a fault tree that could be easily displayed on a graphical screen or printed on a standard sheet of paper. This idea expanded to allow the pages of the fault tree to be connected together with transfer gates. SAPHIRE stores fault trees by pages, in a relational database. The name of each system is the key to locate the system (fault tree) in the database. Transfer gates are stored as subsystems. Again, the name of the transfer gate is the name of the subsystem. During the load process, these names are used to connect the fault tree logic.

Fault Tree Page:	A portion of a fault tree designed to fit on one printed page.
------------------	--

Because SAPHIRE stores the logic of these fault trees as physically separate pages, connected by transfer gates, the first task is to load these pages into memory and combine them into one connected fault tree. This is done by reading in the logic for the first page of the tree, then recursively scanning the loaded logic for a transfer gate that has not been processed. SAPHIRE allows the user to specify whether a transfer gate is to be expanded or not. The gates that are flagged (identified as not to be expanded) are converted to basic events at this time.

During the load process, SAPHIRE connects gates to the tree by name. The gates are maintained in a sorted list that is searched using a binary search, when required. When a new gate is encountered, it is inserted into the gate list in sorted order. As the tree is loaded, transfer gates are replaced by gates with developed logic beneath them. During this process, if SAPHIRE encounters a gate that is not a transfer and has the same name as another gate, it checks to see if it is an identical gate (i.e., it is the same type and has the same inputs). If the gates are not identical, SAPHIRE displays an error message and terminates the process after the tree is loaded.

When all transfer gates have been processed, any transfer gates remaining are considered to be unresolved transfer gates. The user is notified of these and they are converted to basic events with the same name as the transfer gate. This allows SAPHIRE to continue processing the fault tree. These unresolved transfers will appear as basic events in the cut sets.

If the tree is successfully loaded, SAPHIRE checks to see if the user has specified a gate name to be used as the top gate. If so, then the tree is pruned to eliminate any logic that is not connected beneath this gate. This process simplifies the tree and frees any memory used by the excess logic. At this point, the tree is ready for further processing.

5.3 N/M Gate Expansion

The next step is to convert N/M gates to their representative logic in terms of AND and OR gates. This type of gate is used in SAPHIRE to simplify the definition of the logic for situations where the user needs to define a structure representing the combination of M things taken N at a time. The user may specify any combination where N and M range from 98 to 99 and $N < M$. SAPHIRE automatically converts these gate structures by first generating a number of intermediate AND gates containing as inputs the combinations of inputs represented, then these gates are input to the original N/M gate. Once this is complete, the N/M gate type is changed to an OR gate. The number of AND gates under the OR gates is determined by the total number of combinations of N failures out of a population of M events. The equation for this number of combinations is

$$\binom{M}{N} = \frac{M!}{N!(M-N)!}$$

An example of this process can be illustrated with the following "2/3" gate.

GATE1	2/3	INPUT1	INPUT2	INPUT3
-------	-----	--------	--------	--------

is converted to the following structure:

GATE1	OR	N/M-1	N/M-2	N/M-3
N/M-1	AND	INPUT1	INPUT2	
N/M-2	AND	INPUT1	INPUT3	
N/M-3	AND	INPUT2	INPUT3	

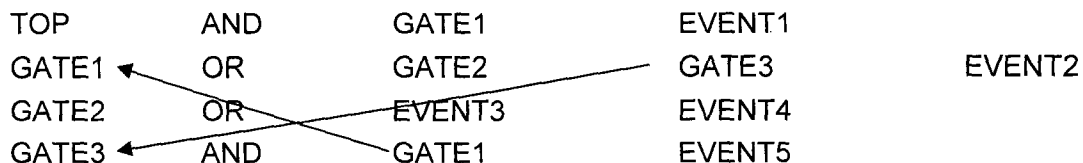
Thus, for 2 out of 3 gates, there are 3 unique combinations of 2 failures. This generates 3 AND gates under the OR gate. If the number of inputs to the gate does not equal M , then an error message is generated. In this case, SAPHIRE will not try to solve the fault tree due to the inconsistency. Note that N/M gates may yield a large number of combinations – consequently, SAPHIRE attempts to solve this portion of the logic late in the analysis in order to minimize having to keep track of these combinations.

5.4 TOP Gate Determination

If the user has not specified the gate to be used as the top gate of the fault tree, the next step in solving the fault tree is to determine which gate is the "TOP" gate. This process is carried out by counting the references to each gate. A gate is referenced if it appears as input to any other gate. The top gate is the only gate that will not be referenced by any other gate. If SAPHIRE detects more than one gate that qualifies as the TOP gate, then the user is notified and given the opportunity to select the gate to be used as the TOP gate. If no gate is selected, SAPHIRE will not try to solve the fault tree. If, however, the user selects one of the gates, SAPHIRE will prune all other logic not connected to this gate and continue with the solution.

5.5 Logic Loop Error Detection

Now that the TOP gate of the fault tree has been determined, SAPHIRE can proceed to check for loops in the fault tree. A loop is a situation where a gate either directly or indirectly references itself. A simple example of a loop is represented by the following fault tree logic:



In this example, GATE1 indirectly references itself since GATE1 references GATE3, and GATE3 references GATE1, as indicated by the arrows.

To determine if there is a loop in the fault tree logic, SAPHIRE defines a Boolean array containing one element for each gate in the fault tree. This list is then initialized to FALSE. During processing of a gate, the Boolean variable for that gate is TRUE when processing that gate or any of its inputs, otherwise it is FALSE. Starting with the TOP gate, SAPHIRE traverses the fault tree by following the gates defined in the inputs to each gate. As a gate is encountered, its Boolean variable is tested. If the value of this variable is TRUE, then a previous reference to this gate must have occurred indicating a loop exists in the fault tree at this point. If Boolean variable is FALSE, then it is set to TRUE to indicate that this gate is currently being processed and the inputs for this gate are traversed. When all the inputs to a gate have been checked, the Boolean variable for the gate is set to FALSE before exiting. Using the previous loop example, the processing proceeds as follows:

- (1) Initialize a Boolean array.

TOP	GATE1	GATE2	GATE3
FALSE	FALSE	FALSE	FALSE

- (2) Start processing the TOP gate.
Set flag for TOP gate.

TOP	GATE1	GATE2	GATE3
TRUE	FALSE	FALSE	FALSE

- (3) Process the first input to the TOP gate.
First input is GATE1.
Set flag for GATE1 and continue.

TOP	GATE1	GATE2	GATE3
TRUE	TRUE	FALSE	FALSE

- (4) Process the first input to GATE1.
First input is GATE2.
Set flag for GATE2 and continue.

TOP	GATE1	GATE2	GATE3
TRUE	TRUE	TRUE	FALSE

- (5) No gates input to GATE2.

Reset flag for GATE2 and exit.

TOP	GATE1	GATE2	GATE3
TRUE	TRUE	FALSE	FALSE

- (6) Continue processing inputs to GATE1.
Next input is GATE3.
Set flag for GATE3 and continue.

TOP	GATE1	GATE2	GATE3
TRUE	TRUE	FALSE	TRUE

- (7) Process inputs to GATE3.
First input is GATE1.
Set flag for GATE1.
Flag is already set (TRUE) therefore a loop is detected!

TOP	GATE1	GATE2	GATE3
TRUE	TRUE	FALSE	TRUE

Two points of optimization can be considered in this approach. First, each gate only needs to be processed once. If it is referenced several times in the fault tree, repeated processing can be time consuming. SAPHIRE maintains a list of those gates that have been processed and only traverses those that have not been previously processed. Second, this algorithm is quite repetitive and can be implemented quite nicely as a recursive procedure.

If SAPHIRE detects a loop in the fault tree, a fatal error is generated along with a traceback. This traceback defines exactly the gate reference list that caused the loop. Currently, SAPHIRE will not process a fault tree that has loops. The user must modify the logic to remove the loop before SAPHIRE will solve the fault tree.

5.6 Complemented Gate Conversion

Once SAPHIRE has ensured that the fault tree logic does not contain any loops, the complemented gates in the fault tree are processed. Two types of complemented gates are allowed in SAPHIRE. The user may indicate a complemented gate by using either the NAND or the NOR gate or by putting a forward slash (/) in front of a gate name. If the complemented gate types are used, then all references to the gate name will use the complemented logic. If the user wants to complement only a specific reference to a gate, then the slash character may be used in front of the gate name where it is referenced.

SAPHIRE processes complemented gates by first complementing the gate type and then complementing the inputs to the gate. The following example demonstrates this process:

TOP	AND	GATE1	GATE2
GATE1	AND	GATE3	EVENT1
GATE2	AND	GATE3	EVENT2
GATE3	OR	EVENT3	EVENT4

becomes (internal to SAPHIRE)

TOP	AND	GATE1	GATE2
GATE1	OR	/GATE3	/EVENT1
GATE2	AND	GATE3	EVENT2
GATE3	AND	/EVENT4	/EVENT5

where the "/" character represents the complement of the input.

Notice that GATE3 is referenced as both a complemented gate and a noncomplemented gate. To handle this, SAPHIRE generates a new gate called NOT3 that contains the complemented version of GATE3. Now, the new (internal) fault tree is as follows:

TOP	AND	GATE1	GATE2
GATE1	OR	NOT3	/EVENT1
GATE2	AND	GATE3	EVENT2
GATE3	AND	/EVENT4	/EVENT5
NOT3	OR	EVENT4	EVENT5

If every gate in the tree is referenced in the fault tree as both complemented and noncomplemented, then this approach to processing the complemented gates can result in a fault tree with twice the number of gates as in the original tree. This, however, is not usually the case and the number of additional gates is substantially smaller. When SAPHIRE first encounters a reference to a complemented gate in the fault tree, it assumes that this will be the only reference to the gate; therefore, it complements the original gate. If later on it encounters a reference to the noncomplemented version of the gate, it then generates a new gate that is identical to the original uncomplemented gate.

5.7 House Event Pruning

SAPHIRE allows the user to modify the logic structure of a fault tree by using "house" events. House events are events that can be set to logical TRUE (T) or FALSE (F). This forces the event to occur with house event TRUE, or forces it not to occur with house event FALSE. SAPHIRE also allows the user to specify that an event is to be ignored with house event IGNORE (I) which says to remove the event from the fault tree logic. An event set to house event IGNORE will be treated as if it did not exist in the fault tree.

Normally, house events are treated as special events that must be designated as house events. In SAPHIRE, however, the user may treat any event as a house event. Since SAPHIRE creates an event for each transfer gate in the tree, house events may also be used to prune subsystems from a fault tree. At various times, SAPHIRE will use house events to simplify or optimize the processing of the fault tree. There are two of these situations. First, if the user is truncating on probability and the probability of an event is below the truncation value, then we know that this event has negligible probability of occurring. To prune the fault tree, we set these events to house event FALSE. This same technique could be used for other truncation criteria that can be determined before the fault tree is solved to further simplify the tree. A group of house event settings (for multiple basic events) may be defined via the "Flag Set" option where the flag set is then assigned to one (or more) fault trees.

Second, SAPHIRE uses house events when solving sequence cut sets. In SAPHIRE, accident sequences are defined using an event tree to indicate the failure or success of top events. Each top event in the event tree is associated with a system fault tree. To solve the accident sequence, SAPHIRE constructs a fault tree for those systems that are defined to be failed in the sequence logic by creating a dummy AND gate with these systems as inputs. SAPHIRE then solves this fault tree using the specified truncation values. This process results in a list of cut sets for the failed systems in the accident sequence. SAPHIRE then uses a "delete term" technique (as a default option, this option may be changed by the user) to further reduce this list of failed system cut sets. This technique uses the cut sets determined from solving the successful system fault trees in the accident sequence logic to eliminate cut sets from the list of failed system cut sets. To do this, SAPHIRE first scans the list of failed-system cut sets and assigns a value of FALSE to any event in SAPHIRE that does not appear in this list. Once this is done, the fault tree representing the successful systems in the accident sequence logic is constructed, pruned by the house events, and solved. The events that are set to FALSE in the previous step result in a significantly reduced success system fault tree. We can do this since we know that for any successful-system cut set to eliminate a failed-system cut set, it must contain only events in the list of failed-system cut sets. Setting these events to house event FALSE will ensure that the cut sets with these events in them will be eliminated at the fault tree restructuring step. This process greatly speeds up the solution of the successful system fault tree. For example, let the following cut sets represent the failed systems cut sets for the accident sequence.

E1 * E2 * E3

E2 * E5 * E7

E1 * E2 * E5

Let the following fault tree represent the successful-systems fault tree.

```
TOP OR SYS1 SYS2 SYS3
SYS1 AND E1 E6
SYS2 AND E1 E5
SYS3 AND E3 E4
```

Since events E4 and E6 do not appear in the list of failed-systems cut sets, we can set them to house event FALSE and prune the fault tree, resulting in the following fault tree.

```
TOP OR SYS1 SYS2 SYS3
SYS1 AND E1 FALSE
SYS2 AND E1 E5
SYS3 AND E3 FALSE
```

Pruning this tree gives the following reduced fault tree.

```
TOP AND E1 E5
```

Solving this fault tree results in the following single cut set

```
E1 * E5
```

This cut set is used to reduce the failed-systems cut sets as follows.

```
E1 * E2 * E3
E2 * E5 * E7
E1 * E2 * E5
```

Whether specified externally by the user or internally by SAPHIRE, before the fault tree is solved, it is pruned depending on the structure of the tree and the house event setting. In order to do this, SAPHIRE again traverses the fault tree checking for house events. At each gate the algorithm checks each of the inputs to the gate to see if it has been set to any one of the three house event settings, "T," "F," or "I." If so then the logic for that gate is modified as follows. If the gate is an AND gate, then an input set to T or I is removed from the gate input list, while an input set to F causes the gate to be set to F. If the gate is an OR gate, then an input set to F or I is removed from the gate input list, while an input set to T causes the gate to be set to T. For example, we show in Figure 11 that the LOP house event is turned on (TRUE) for sequences 3 and 4. For Sequences 1 and 2 the LOP house event is left off (FALSE). The setting of the house event is dependent upon the success or failure of recovering the offsite ac power.

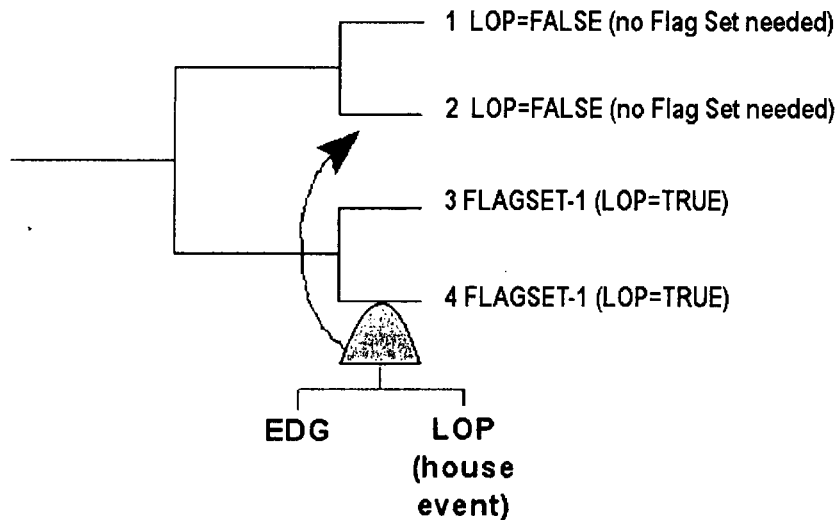


Figure 11. Event tree with fault tree linked to sequences where house events are used

The routine to check for house events and prune the logic of the fault tree is a recursive routine. Using the fault tree logic defined previously, along with the house event information and starting at the top gate in the fault tree, SAPHIRE checks each of the inputs to the current gate. If the input is a gate and the gate has not been previously checked, then the recursive routine calls itself to check this gate. The recursive routine returns a value of T, F, or I for each gate that is processed and it processes each gate only once. If a house event value is returned for the top gate, then there is no need to solve the fault tree and a message is displayed. If the value returned is T, the message "The TOP event has occurred (TRUE)!" will be displayed. If the value is F, then the message "The TOP event cannot occur (FALSE)!" will be displayed. If the value returned is I, then the message "No logic to solve!" will be displayed.

5.8 Coalescing Like Gates

The next step in the fault tree solution is to coalesce like gates. This process combines those gates that are input to other gates of the same type. Specifically, AND gates that are input to AND gates are combined and OR gates that are input to OR gates are combined. The following fault tree is an example of the coalescing of both an AND gate and an OR gate.



GATE2	AND	EVENT2	EVENT3
GATE3	OR	EVENT4	EVENT5

After coalescing, GATE2 is consumed by the TOP gate and GATE3 is combined with GATE1. The following fault tree is the result of these modifications.

TOP	AND	GATE1	EVENT2	EVENT3
GATE1	OR	EVENT1	EVENT4	EVENT5

In the above example, both gates that were coalesced were referenced only by gates of the same type. This resulted in the removal of both of these gates from the logic. The following example shows a case where the coalesced gate is not removed.

TOP	AND	GATE1	GATE2
GATE1	OR	GATE2	EVENT1
GATE2	AND	EVENT2	EVENT3

After coalescing, the following tree is generated:

TOP	AND	GATE1	EVENT2	EVENT3
GATE1	OR	GATE2	EVENT1	
GATE2	AND	EVENT2	EVENT3	

By coalescing the fault tree, the number of gates is reduced and the number of inputs to a gate is maximized. This process can substantially reduce the processing time as well as provide for better optimization later in the fault tree restructuring process. Note, however, that the total amount of space required to store the inputs to the fault tree can grow significantly as a result of coalescing the tree. The amount of additional space required depends on the number of gates that can be coalesced, the number of times a coalesced gate is referenced in the tree, and the number of inputs to the coalesced gate. This increased space requirement will usually be recovered during module and independent subtree processing later.

To perform the coalescing step, SAPHIRE starts with the TOP gate of the fault tree and recursively checks the list of inputs to the current gate. Any duplicate inputs in the list are removed. If the input is a gate and it is the same type as the current gate, then the list of inputs to this gate is added to the current gate input list. The gate reference is then removed from the list. If the input is a gate with a single input then the gate reference is replaced by its input. Once all inputs to all gates have been processed, then SAPHIRE makes a pass through the current gate list and eliminates any gates that are no longer needed due to any of the previous restructuring steps.

5.9 Modules versus Independent Subtrees

SAPHIRE uses two methods of optimization that are similar and should be clarified. These optimization methods are independent subtrees and modules. Before solving a fault tree, SAPHIRE converts all the logic into a logically equivalent form in terms of AND gates, OR gates, and basic events. The following discussion assumes this form of fault tree logic. In SAPHIRE, an *independent event* is defined as an event that is input to only one gate. An *independent gate* is a gate that is input to only one other gate and contains as inputs only independent events.

An *independent subtree* is a gate that has as inputs only independent events or independent gates. The inputs to an independent subtree can occur only once in a fault tree, however, an independent subtree may be input to many other gates. Note, the independence defined here is logical independence.

In SAPHIRE a set of events $M=\{E_1,E_2,\dots,E_n\}$ is defined to be a *module* of a fault tree if the following two conditions are met:

- For every occurrence of E as input to a gate, the other events in M also occur as input to the same gate.
- Every occurrence of M is an input to the same gate type, either an AND or an OR gate.

These events can be combined under a single gate called a module. All references to these events are converted to reference the module. Once a module is created, all of the events input to it occur only as inputs to a single gate. Since a module may appear multiple times in a fault tree, it is usually not an independent gate, however, it is always an independent subtree. A gate that has a module as one of its inputs is only an independent subtree if the module is an independent gate.

In the fault tree reduction process, independent subtrees need not be expanded until the very end of the process. Once a fault tree is solved in terms of independent subtrees, it is a simple expansion process to convert the minimal cut sets to their basic event representation. Since a reduced number of tokens needs to be analyzed in the fault tree solution process, independent subtrees save large amounts of processing time. Figure 12 shows an example fault tree with a module and an independent subtree. In the example, Gate-3 also happens to be an independent gate.

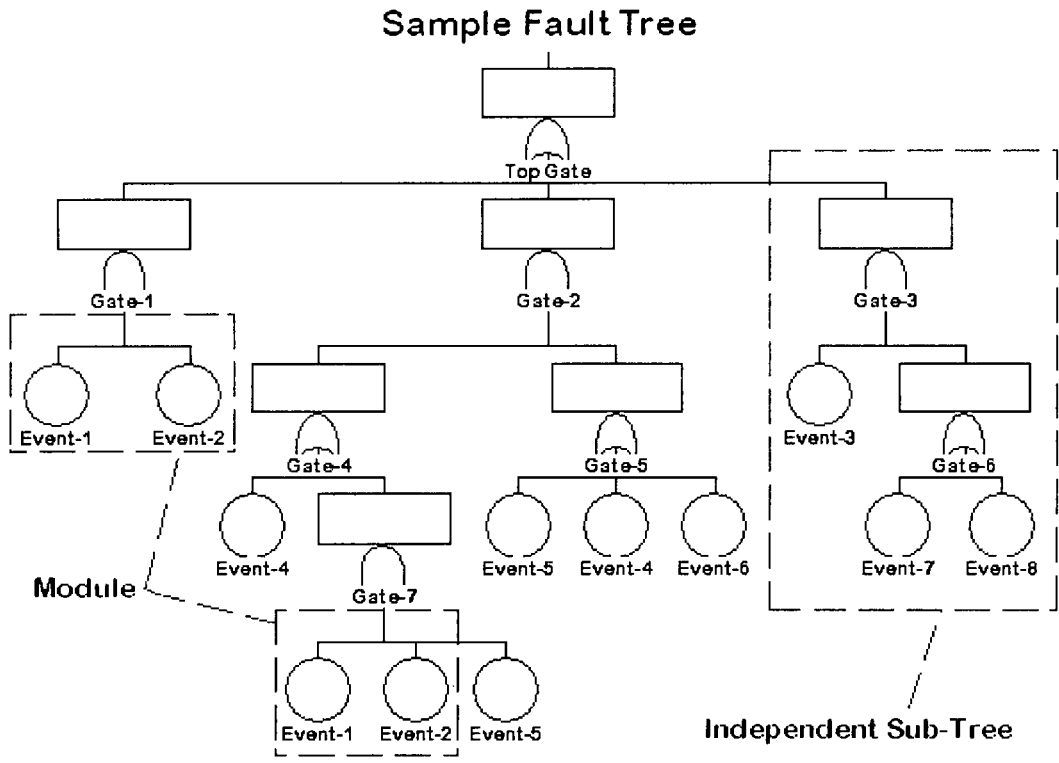


Figure 12 Examples of independent a subtree and module fault tree

5.10 Module Determination and Creation

The next step in the restructuring process is to find all modules in the fault tree. To perform this step, SAPHIRE uses a temporary bit vector. The bit vector contains one bit for each event in the fault tree. The first of these bit vectors keeps track of the events that are used in the fault tree. If complemented events are used, then a second bit vector is allocated for the complemented events.

A vector is also created for each gate currently defined. These vectors will contain, in bit format, the events used by each gate. We also define two vectors, TMP1 and TMP2, which hold intermediate results. Finally, we define an array containing one number for each event. This number is a count of the number of times each event is used in the fault tree.

Once the data arrays are created, we initialize the TMP1 vector and the event count array by traversing the input list. For each input, we check to see if it is an event, and if so, we set its bit in the TMP1 vector and increment the count for this event. If the event is complemented, then its bit is set in the complemented vector. When all inputs have been processed, we eliminate any event that occurs as both a complemented and a non-complemented event from the event vector list. These events cannot be included in modules. Next, we process each gate and set the appropriate bits in each gate's bit vector to reflect the events used by that gate. When this process is complete, we are ready to find the modules in the fault tree. Using the fault tree shown in Figure 12, the following initial data structures would be defined:

	Event-1	Event-2	Event-3	Event-4	Event-5	Event-6	Event-7	Event-8
Used?	1	1	1	1	1	1	1	1

5.11 Independent Gate, Subtree, and Event Determination

The next step in the fault tree restructuring process is to determine which events are independent. For this purpose SAPHIRE defines "independent" as only occurring once in the fault tree. This step is performed by defining two bit vectors. Each time an event is encountered, a bit is set in the first vector. If the bit is already set, then the corresponding bit in the second vector is also set. When complete, the second bit vector represents the list of basic events that occur more than once. The events not in this list are independent.

The next step in the restructuring of the fault tree is to determine the independent gates and subtrees in the fault tree. Independent subtrees are much easier to solve since they generate only minimal cut sets. SAPHIRE processes independent subtrees separately from the rest of the fault tree.

To find the independent gates and subtrees, SAPHIRE again uses a recursive routine to traverse the fault tree. SAPHIRE uses the data structures defined previously to check the inputs to each gate. If all the inputs to the gate are independent events and the gate occurs only once, then it is marked as an independent gate. If the input is a gate and has not been processed, then the routine calls itself to check this gate. If all inputs to the gate are independent events or gates, then the gate is flagged as an independent subtree. This results in a fault tree that has all independent subtrees identified.

5.12 Determining Gate Levels

The last step in the fault tree restructuring process is to determine the gate levels. The TOP gate is defined to have level 0. Its inputs have level 1, the inputs to those gates have level 2, and so forth. The *level* of a gate is the number of gates one encounters after the TOP in going from the TOP to the gate of interest. If a gate appears more than once in a tree, define the gate's level as the largest of the levels corresponding to the various places where the gate occurs. To determine the level of each gate, a recursive routine is used. This routine keeps track of the level for each gate. Each time the gate is encountered in the traversal of the fault tree, its level is checked against the current level. If the current level is greater than the gate's assigned level, then the gate's level is set to the current level. The routine exits early if a gate's level is greater than or equal to the current level. This process continues until the entire tree has been processed.

This information is used later in determining the *expansion path* for the fault tree. The expansion path for a fault tree is the order in which the gates for a fault tree are solved. This expansion path can significantly affect the time it takes to solve a fault tree. SAPHIRE attempts to determine the optimal expansion path.

5.13 Fault Tree Reduction and Truncation

Once the fault tree is loaded and restructured, it is ready to be solved. This process consists of a number of steps that convert the Boolean logic representing the fault tree to its expanded form representing the desired minimal cut sets for the tree. In SAPHIRE, a fault tree may represent either a system equation or a sequence equation. In either case, the same algorithm is used to solve the tree.

The exact solution of many large fault trees using cut set-based methods can prove to be prohibitive; therefore, various methods have been developed to reduce the time required to solve a fault tree. SAPHIRE allows the user to specify that a number of these methods be used in the fault tree solution. The first and most common method is to eliminate any cut set whose probability falls below a specified truncation value. The second method is to eliminate any cut set that has more than a specified number of unique events in it. The third method is to eliminate any cut set that has more than a specified number of zone flagged events in it. A *zone flagged event* is an event that has been marked as representing a zone (location or area). In a facility, a fire zone may represent a room with fire barriers around it. A security zone may represent an area with certain security characteristics. This method is used in location analysis to allow for the truncation on the number of zone events in a cut set. The last method provided in SAPHIRE for cut set truncation is typically used in seismic analysis and allows the user to combine the first truncation method with another criterion that checks to see if any event in the cut set is below a specified probability before it is truncated.

All of the above truncation methods are supported by SAPHIRE. The user may also choose to solve the fault tree exactly. No matter which methods are used, SAPHIRE attempts to take advantage of whatever it can to simplify and reduce the amount of work required to solve a tree. The ways each of these truncation methods is implemented will be discussed in detail as the process for the fault tree solution is described.

5.14 Intermediate Result Caching and Initialization

Fault tree solutions can easily generate enough intermediate cut sets to fill up all available computer memory. Therefore, a method is required to allow this data to be written out to a secondary data storage area. SAPHIRE uses a disk caching technique to store the intermediate data. This allows for the processing of large amounts of intermediate data. The limit is the amount of available disk space on the computer being used. SAPHIRE does, however, allow the user with a more powerful computer and additional extended memory to create a virtual disk and direct the intermediate information that would have resided on the hard disk to the virtual disk.

The first step in the fault tree reduction process is to take the fault tree logic that has been loaded and restructured and store this logic in a format for efficient use and retrieval by the fault tree reduction software. This process includes the creation and initialization of certain data structures containing information that is used during the solution process to simplify and speed up the fault tree reduction process. By including this data in a data structure and updating it as the fault tree is solved, SAPHIRE is able to avoid many additional calculations.

Using the gate level information determined previously, SAPHIRE creates an ordered table such that all gates for a given level appear before any gates for the next larger level. Any independent subtrees appear after all nonindependent gates for the fault tree. This ordering defines the expansion path to be used for solving the fault tree. As mentioned previously, the SAPHIRE algorithm is essentially a top-down approach, but strictly speaking, the algorithm processes the fault tree first from the bottom up, then from the top down. The algorithm is bottom up because we treat each OR gate as a mini fault tree and solve them starting with the last gate or the bottom of the fault tree. When all OR gates up to the TOP gate have been solved, SAPHIRE expands the TOP gate from the top down.

As the fault tree logic table is being created, SAPHIRE generates information to be used during the expansion process to help in cut set truncation. A bound can be calculated on the contribution of the independent subtrees to the cut set probabilities. If the user has specified truncation on probability, this bound can be used to eliminate cut sets earlier than otherwise possible. For now, let BPC denote this Bound on the Probability Contribution (BPC). Calculate the BPC for any gate as follows. The BPC for a basic event is its probability. The BPC for an AND gate is the product of the BPC's of the inputs. The BPC for an OR gate is the largest BPC of the inputs. Since the gate table is ordered by level, these calculations can be performed one gate at a time, starting with the last gate and proceeding to the top of each independent subtree.

To see how this works, suppose first that S is an independent subtree with only two inputs, A and B , both basic events. Because S is independent, each of its basic events appears only once, so A and B do not appear in any other part of the fault tree. Because basic events are assumed to be independent in the statistical sense, A and B are statistically independent of each other and of the rest of the tree.

Any cut set that S contributes to will have the form (S AND other terms). If S is an AND gate, this form is (A AND B AND other terms), and the probability of the cut set is $P(A)P(B)P(\text{other terms})$, by independence. This equals $\text{BPC}(S) \times P(\text{other terms})$, by the definition of BPC for an AND gate. If instead S is an OR gate, any cut set that S contributes to will have the form (A and other terms) or else (B and other terms). The cut set probabilities are bounded by

$$\max[P(A), P(B)] \times P(\text{other terms})$$

which equals $\text{BPC}(S) \times P(\text{other terms})$, by the definition of BPC for an OR gate.

In either case, any cut set that S contributes to have probability bounded by the value of BPC for S . The same idea is true if S has more than two inputs, and if they are not necessarily basic events but may be independent gates instead. Therefore, if BPC for S is less than the

truncation value, S can be eliminated from the tree. In any case, the BPC is calculated and stored so that it can be used to eliminate cut sets earlier than otherwise possible.

If the user has chosen to truncate on size or zones a similar calculation can be performed on independent subtrees to get a size contribution of the subtree to each cut set it appears in. If size truncation is selected, then all basic events are counted. If zone truncation is selected, then only events that are zone flagged are counted. At each AND gate, the size contributions of the inputs are added together. For a qualified basic event the size is one. For a gate, however, the size may be larger than one. At each OR gate, the size of the smallest input is used as the size contribution of the gate. Once these values are calculated, they are stored in the gate table for future use. The tree is now ready to be expanded.

5.15 Fault Tree Gate Expansion

The process of solving a fault tree involves three basic steps. These steps are gate expansion, Boolean absorption, and cut set truncation. In the first step, the gates of the tree are expanded by replacing them with their inputs. In the second step, the first four of the following identities are applied to the cut sets:

1. $A * A = A$
2. $A + A * B = A$
3. $A * B * /A = \emptyset$
4. $//A = A$
5. $A * B + A * /B = A$ (not applied).

The first identity (idempotent relationship) prevents two identical events from appearing in the same cut set. The second one (absorption relationship) is the most computationally difficult to apply. In terms of set theory it consists of eliminating subsets, because $A*B$ is a subset of A . Computer programmers, on the other hand, tend to think of the identity as eliminating supersets; $A*B$ is regarded as a larger entity than A because it has more tokens to manipulate. Both the subset and superset terminology can be found in the literature, but this document will use only the term "absorption." The absorption identity is used to eliminate cut sets that are not minimal. The basis for using the Law of Absorption is that the top gate has become a giant OR gate with the cut sets as inputs. If A and $A*B$ are cut sets, the top gate contains $A + A*B$, which can be simplified to A . The third identity (exclusion relationship) implies that no cut set will contain both the failure and the success of an event. The fourth identity (double negation relationship) states that the complement of a complemented event is the event itself. Identity number five (exhaustion relationship) is not currently performed by SAPHIRE. It is important to note that SAPHIRE does not calculate prime implicants (Quine 1959). Complemented events appear in the cut sets with a "/" in front of the event name.

The final step, cut set truncation, involves the elimination of cut sets that fall outside user specified truncation limits. There have been many different methods applied to performing these three steps. Some codes use a top-down approach, while others use a bottom-up

approach. Both approaches have their strong points. SAPHIRE uses some features from each approach to optimize the fault tree solution process.

Using the fault tree logic definition generated previously, SAPHIRE begins expanding the tree. Since OR gates increase the number of cut sets, the algorithm treats all OR gates in the fault tree as mini fault trees. These trees are solved first, starting with the last nonindependent OR gate and proceeding to the TOP gate of the fault tree. All absorption and truncation techniques are applied on these small trees, eliminating cut sets as soon as possible. When the TOP gate is encountered, it is solved using as input all the cut sets generated by solving the mini fault trees described above. The result of this approach is to partition the large fault tree into many smaller subtrees that are easier to solve. The fewer cut sets generated for the smaller trees will also tend to require less time to apply the absorption identities and to truncate.

Note that the cut sets generated by the above process are in terms of independent subtrees. When the TOP gate has been solved and all absorption has been performed, the independent subtrees are expanded. This step requires no absorption; independent subtrees can only generate cut sets that are minimal.

5.16 Cut Set Absorption

As the fault tree expansion occurs, cut sets are checked at each gate to see if they can be eliminated. There are several ways a cut set may be eliminated during the expansion process. SAPHIRE maintains the current bound on the probability contribution (BPC defined in Section 5.17) and size for each cut set throughout the fault tree expansion. These contributions are updated depending on the type of expansion being performed. By keeping current BPC values, SAPHIRE does not need to recalculate these values each time the cut set is modified or expanded, thus saving time.

If the gate to be expanded is an OR gate, then SAPHIRE also compares the inputs to the OR gate against the inputs of the cut set containing the OR gate. If there is a common event, then the reference to the OR gate can be removed and the cut set need not be expanded further. The reason for this is that any cut sets generated from an OR gate of this type will be absorbed later in the process anyway. The following example demonstrates this process.

The cut set

$$\text{GATE1} * \text{EVENT1} * \text{EVENT2}$$

and the following definition of GATE1 as an OR gate with three inputs

$$\text{GATE1 OR EVENT1 EVENT3 EVENT4}$$

will generate the following cut sets when expanded.

EVENT1 * EVENT2
EVENT1 * EVENT2 * EVENT3
EVENT1 * EVENT2 * EVENT4

Notice that the second and third cut sets are absorbed by the first.

5.17 Boolean Absorption

The process of performing the Boolean absorption reduction can be a time-consuming operation. The methods used in SAPHIRE are described in Corynen (1988). This method uses a set of bit tables to determine those cut sets that can be absorbed by a given cut set. For a detailed description of the process, refer to the indicated document. This method is very powerful and has good run-time characteristics. In order to be most effective with this algorithm or any other one used for the Boolean absorption process, the number of cut sets compared must be minimized. The expansion approach described previously tends to generate smaller numbers of intermediate cut sets, minimizing the amount of time spent on absorption.

5.18 Data Storage Considerations

Given the task to be performed in solving a fault tree, an optimal format for storage and retrieval of the intermediate cut set data must be determined. Two obvious methods were considered in SAPHIRE. First, since a large amount of time can be spent in the determination of sets to be absorbed, one option is to store the intermediate data in a format that can be directly used by the absorption routine. This format would be an array of bit vectors with each row of the array representing an event and each column representing a cut set. This format was used in the first version of SAPHIRE and worked well for small problems because the bit vector arrays could be easily contained in the computer's fast memory. As problem size increased and it became necessary to shift these arrays to disk, this method of storage became difficult to manage efficiently.

The second alternative is to store the cut sets as an array of numbers representing the events in each cut set. The first number is a count representing the number of events in the cut set. This number would be followed by a probability value, a size value, and a list of numbers representing the gates or events contained in the cut set. The list is terminated by a zero count number. This format is the one used in the current version of SAPHIRE. It is simple and easy to store and retrieve from intermediate storage. The process of gate expansion is also easily handled with this format. When absorption is performed, SAPHIRE creates the array of bit vectors. As problem size increases, this format has proven to be much more flexible and easy to manage than the first. Also, modifications in the storage routines have led to efficiently using the cut set lists, resulting in an improvement in analysis speed.

5.19 Sequence Cut Set Generation

Another area that must be considered when developing a risk assessment code is the accident sequence analysis. Accident sequences are defined in SAPHIRE by developing event trees. SAPHIRE provides a graphical editor to use in developing event trees. Figure 13 shows an example of an event tree developed in SAPHIRE. Once the user has developed the event tree, SAPHIRE generates the sequence logic from the graphical event tree (via the event tree "link" option). The sequence logic is the list of systems that succeed or fail during this accident sequence. These system failures and successes are top events of fault trees. This logic is used by SAPHIRE to generate the cut sets for the sequence.

There are two methods that can be used to generate sequence cut sets. First, the cut sets generated by solving the system fault trees can be used as input to the accident sequence algorithm. This method simply combines the cut sets for each system as defined by the sequence logic and is largely deprecated within the risk analysis community. The second method is to create a fault tree for a sequence by combining the fault trees corresponding to system failures and successes for the sequence. The fault tree reduction algorithms can then be used to solve the accident sequence. SAPHIRE allows the user to select either method, but only the latter method will be discussed here.

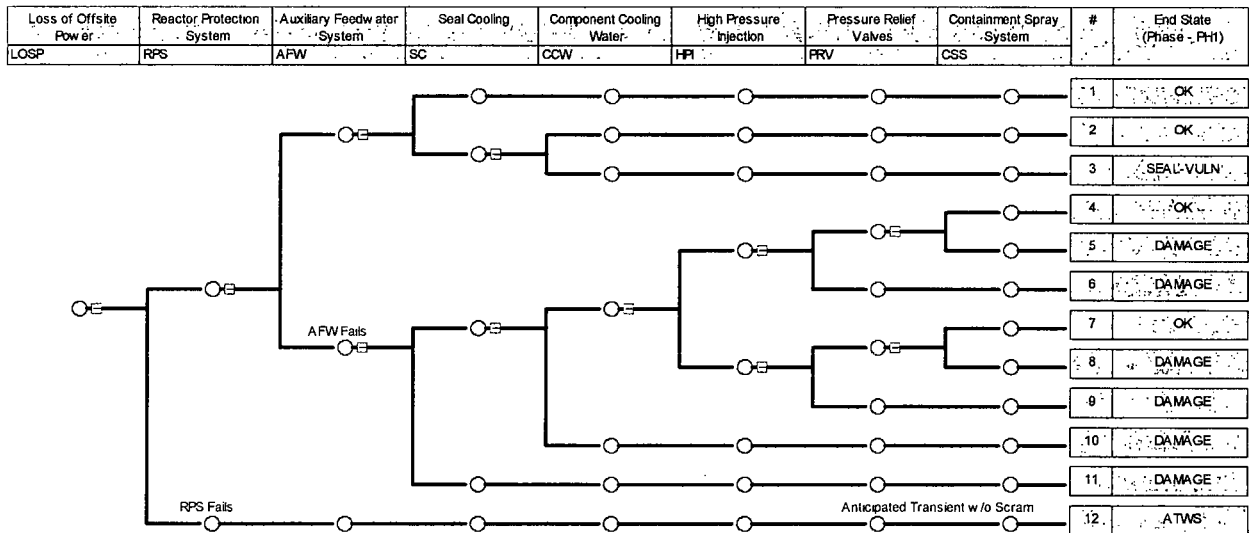


Figure 13 Example SAPHIRE event tree

In SAPHIRE, accident sequences are defined using an event tree to indicate the failure or success of top events. Each top event in the event tree is associated with a system fault tree. To solve the accident sequence, SAPHIRE constructs a fault tree for those systems that are defined to be failed in the sequence logic by creating a dummy AND gate with these systems as inputs. In Figure 13, the accident sequence logic for sequence 9 is

LOSP * /RPS * AFW * /SC * /CCW * HPI * PRV

Therefore, SAPHIRE creates (internally) the following failed systems fault tree

FAILED	AND	AFW	HPI	PRV
AFW	TRAN			
HPI	TRAN			
PRV	TRAN			

where AFW, HPI, and PRV represent the fault tree logic for Auxiliary Feedwater System, High Pressure Injection, and Pressure Relief Valves, respectively, and TRAN denotes a transfer to the system fault tree.

SAPHIRE then solves this fault tree using the specified truncation values. This process results in a list of cut sets for the failed systems in the accident sequence. SAPHIRE then uses the "delete term" technique to further reduce this list of failed-system cut sets. This technique uses the cut sets determined from solving the successful-system fault trees in the accident sequence logic to eliminate cut sets from the list of failed-system cut sets. To do this, SAPHIRE first scans the list of failed-system cut sets and assigns a value of FALSE to any basic event that does not appear in this list. Once this is done, the fault tree representing the successful systems in the accident sequence logic is constructed, pruned by the house events, and solved. The successful systems fault tree for accident sequence 9 is

SUCCESS	OR	RPS	SC	CCW
RPS	TRAN			
SC	TRAN			
CCW	TRAN			

where RPS, SC, and CCW represent the fault tree logic for the Reactor Protection System, Seal Cooling, and Component Cooling Water, respectively. This fault tree models failure of the RPS system, the SC system, or the CCW system. The top event of the tree does not occur as part of accident sequence 9. That is, none of the cut sets in the tree occur.

The minimal cut sets for the sequence remain after the successful-system cut sets terms are deleted. There are a couple of points to note in this process.

First, each sequence has an initiating event frequency associated with it. If the user specifies a probability truncation value, SAPHIRE divides this value by the initiating event frequency. This eliminates the need to handle the initiating event during the fault tree reduction phase.

Second, during the processing of an accident sequence, certain pieces of equipment or trains of a system may need to be either failed or ignored. SAPHIRE allows the user to specify a set of house event flags to be associated with a particular sequence. These flags, assigned by using *flag sets*, allow the user to automatically prune the fault tree logic before it is solved by

setting basic events to house events and reducing. The result is a fault tree with the specified components in the specific state required by the sequence. Note that while in theory each sequence may have unique flag sets, common elements are shared among typical sequences. Consequently, SAPHIRE sorts sequences (prior to solving) based upon flag sets in order to solve common-parts of the overall event tree as efficiently as possible.

New in SAPHIRE 8 is the ability for sequence flag sets to automatically adjust common-cause failure (CCF) basic events. This change was made to ensure that PRA models that use fault tree and sequence flag sets have the applicable adjustments made to any impacted CCF basic events. In older versions of SAPHIRE, flag sets are processes after basic event data changes, which implies that CCF probability modifications may not be changed for a specific fault tree or sequence flag set.

Additional details on the general solution methods used by SAPHIRE may be found in Russell and Rasmuson (1993).

5.20 Parallel Processing During Cut Set Generation

SAPHIRE 8 has been modified to use threaded calculations that will be pushed to the processors that are available within a single computer. By using multiple threads, the calculation of cut sets is changed from serial processes to parallel processes. During the cut set analysis, the user may select the number of threads to be used, up to a maximum of 32 concurrent processes.

6. QUANTIFICATION TOOLS FOR PROBABILITIES AND FREQUENCIES

This section provides an overview of fault tree and accident sequence quantification using minimal cut sets. Vesely et al. (1981) and Fussell (1975) contain additional details and references for the interested reader. The section is written in terms of failure probabilities, but is also correct if the term "probability" or "failure probability" is replaced everywhere by "unavailability."

6.1 Quantifying Minimal Cut Sets

The individual cut set probabilities are determined by multiplying the probabilities of the applicable basic events.

$$C_i = q_1 q_2 \dots q_n \quad (6-1)$$

where

$$\begin{aligned} C_i &= \text{probability of cut set } i, \text{ and} \\ q_k &= \text{probability of the } k\text{-th basic event in the } i\text{-th cut set.} \end{aligned}$$

This follows from Equation (4-8) and the assumed statistical independence of the basic events.

6.2 Quantifying Fault Trees

The fault tree quantification process is performed in two steps: (1) calculation of individual cut set probabilities, which were described above in Section 6.1, and (2) combining the cut set probabilities. The exact probability of the union of the cut sets can be found, in principle, by Equation (4-6), where each A_i is a cut set. This is normally an intractable problem for large problems. Therefore, two approximations are often used by PRA software, the rare event approximation and the minimal cut set upper bound. Examples are calculated in Sections A-4 and A-5 of Appendix A.

Rare Event Approximation

A common approach to calculate the probability for a top event is to add together the probabilities for the cut sets, where the cut set probability is given by Equation (6-1). Thus, the rare event approximation is

$$S = \sum_{i=1}^m C_i \quad (6-2)$$

where

$$\begin{aligned} S &= \text{minimal cut set upper bound for the system unavailability,} \\ C_i &= \text{probability of the } i\text{th cut set, and} \\ m &= \text{number of minimal cut sets in the fault tree.} \end{aligned}$$

This approximation is accurate when the cut set probabilities are small and the cut sets do not share many of the same basic events. In screening analyses, when relatively large screening values are used to bound the component failure probabilities, the rare event approximation can exceed 1.

Minimal Cut Set Upper Bound

The minimal cut set upper bound calculation is an approximation to the probability of the union of the minimal cut sets for the fault tree. The equation for the minimal cut set upper bound is

$$S = 1 - \prod_{i=1}^m (1 - C_i) \quad (6-3)$$

where

S = minimal cut set upper bound for the system unavailability,

C_i = probability of the i -th cut set, and

m = number of minimal cut sets in the fault tree.

The minimal cut set upper bound is always less than or equal to 1. The input values for the minimal cut set upper bound are probabilities. Barlow and Proschan (1981) showed that Equation (6-3) gives an upper bound on the exact probability of the top event. SAPHIRE uses the minimal cut set upper bound approximation by *default* for its quantification of fault tree, event tree cut sequence, and end state cut sets.

The minimal cut set upper bound works well with fault trees containing only AND and OR gates without complemented events or NOT gates. With noncoherent fault trees, that is, trees that contain NOT gates and/or complemented events, the minimal cut set upper bound can produce results that are conservative. The magnitude of the overestimation will depend upon the structure of the tree and the values of the basic events in the tree. In such cases, other calculation techniques should be used such as the SIGPI algorithm (Patenaude 1987) or binary decision diagrams (Bedford and Cooke, 2001). In many cases, however, the minimal cut set upper bound will produce accurate results.

Note that the default quantification method in SAPHIRE 8 is the minimal cut set upper bound.

Warning: When C_i is very small (on the order of $1E-15$), $1 - C_i$ is rounded off to 1.0. If this happens for most or all of the C_i 's, the product in Equation (6-3) will be too large, and the bound S will be too small. Although S is an upper bound in theory, in practice it is not computed to sufficient accuracy when the C_i 's are extremely small. In such a case the rare event approximation, given by Equation (6-2), is better.

Min-Max

In some cases, both the rare event and minimal cut set upper bound provide too conservative a result. In these cases, it is desirable to have a more accurate quantification calculation. In SAPHIRE, the inclusion-exclusion rule (i.e., the min-max option) is used to calculate the exact

solution when the number of cut sets is small (generally less than 50). Basically, it is the sum of the probability of the individual sets, minus the sum of the probability of all possible pairs, plus the sum of the probabilities of all possible combinations of three, minus the probabilities of all possible combinations of four, plus the probability of intersection of all five minimal cut sets, etc. However, since all combinations of cut sets are evaluated, the min-max approach becomes intractable for calculations involving a many cut sets.

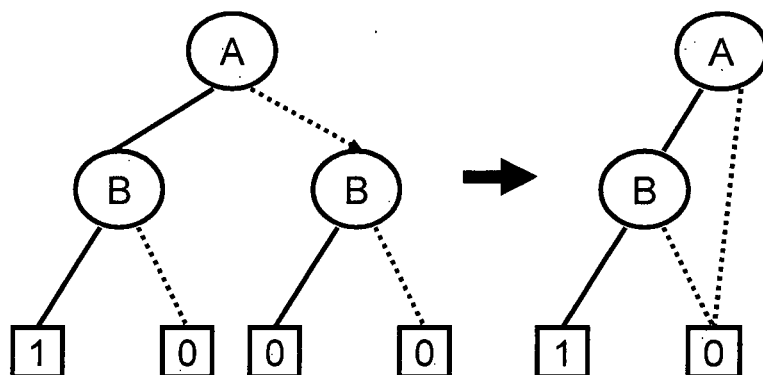
Binary Decision Diagrams

The default quantification method in SAPHIRE 8 is to use minimal cut sets with the minimal cut set upper bound to quantify an overall risk result. Typically sized models can produce millions of minimal cut sets. After determining these cut sets via Boolean algebra (with results typically truncated), the analyst may adjust the cut sets to enhance the realism and accuracy of the results (e.g., by appending recovery actions). Following this “post-processing” step, the cut sets are used to quantify the overall risk measure – however this last step is usually performed via approximations (e.g., minimal cut set upper bound).

Binary Decision Diagram (BDD)-based methods quantify the overall probability directly from the BDD and avoid truncation and the use of approximations seen in cut set-based methods. However, since these methods use the BDD model directly, there is no easy way to adjust failure scenarios by applying recovery, for example. Further, since the model is evaluated via the BDD, it is possible to have complex models that cannot be solved using this technique.

A BDD is a compact way of representing a Boolean logic equation. BDDs were originally developed as a way of representing switching circuits, see (Lee, 1959). Any logic function using AND, OR, NOT, etc., can be represented in a BDD using the “if-then-else” notation. In this notation, every basic event in the Boolean structure is represented as a node in the BDD. Each node in the BDD has two branches from it, one branch if the basic event node is set to a present and the other if the event is set to its complement. When a terminal node is reached, the ending point is either a zero or a one, indicating the presence of the path through the diagram (one) or not (zero).

To demonstrate a BDD, assume we have a single cut set with two events (A, B) representing core damage: $CD = A \cdot B$. The BDD for this case is shown at right. The left part of the figure shows the full BDD, while the right part is an equivalent simpler BDD obtained by applying reduction rules to the first.



In order to demonstrate the quantification using a BDD, let us assume the following probabilities:

$$P(A) = 0.2$$

$$P(B) = 0.5$$

To quantify the BDD, one traces each path through the tree, multiplying the event probabilities as the path is followed to the terminal "1" nodes. Each path probability is then added to obtain the overall probability. In the first case, the only path is:

$$A*B$$

Therefore, the probability of this cut set is:

$$P(CD)_{bdd} = P(A)P(B) = 0.2 * 0.5 = 0.1$$

assuming A and B are independent.

The min-cut upper bound approximation in this case is:

$$P(CD)_{mcutb} = 1 - [1 - P(A)P(B)] = 1 - [1 - 0.1] = 0.1$$

For a second example, assume we have two cut sets: $CD = (A B C)_1 \cup (A D)_2$

Again, to demonstrate the quantification, assume:

$$P(A) = 0.2$$

$$P(B) = 0.5$$

$$P(C) = 0.7$$

$$P(D) = 0.2$$

For the second case, the paths through the BDD are:

$$A*B*C*D$$

$$A*B*C*/D$$

$$A*B*/C*D$$

$$A*/B*C*D$$

$$A*/B*/C*D$$

Therefore, the probability for CD is:

$$\begin{aligned}
 P(CD)_{\text{bdd}} &= P(A)P(B)P(C)P(D) + P(A)P(B)P(C)P(/D) + P(A)P(B)P(/C)P(D) + \\
 &\quad P(A)P(/B)P(C)P(D) + P(A)P(/B)P(/C)P(D) \\
 &= 0.2*0.5*0.7*0.2 + 0.2*0.5*0.7*0.8 + 0.2*0.5*0.3*0.2 + \\
 &\quad 0.2*0.5*0.7*0.2 + 0.2*0.5*0.3*0.2 \\
 &= 0.014 + 0.056 + 0.006 + 0.014 + 0.006 \\
 &= 0.096
 \end{aligned}$$

The min-cut upper bound in this case is:

$$\begin{aligned}
 P(CD)_{\text{mcub}} &= 1 - [1 - P(A)P(B)P(C)][1 - P(A)P(D)] \\
 &= 1 - [1 - 0.07][1 - 0.04] \\
 &= 1 - [0.93][0.96] \\
 &= 0.107
 \end{aligned}$$

The probability using the SAPHIRE min-max approach is:

$$\begin{aligned}
 P(CD)_{\text{min-max}} &= P(A)P(B)P(C) + P(A)P(D) - P(A)P(B)P(C)P(D) \\
 &= 0.2*0.5*0.7 + 0.2*0.2 - 0.2*0.5*0.7*0.2 \\
 &= 0.07 + 0.04 - 0.014 \\
 &= 0.096
 \end{aligned}$$

One problem that is apparent is that BDD size grows exponentially with the number of events in the model. Note that there is not generally a unique BDD corresponding to a given Boolean function. Rather, one must choose an ordering of the nodes in the BDD, and the size and complexity of the BDD can vary significantly with the chosen ordering. To complicate the BDD analysis, it is not generally possible to choose the best ordering (i.e., that leading to the smallest BDD) for any given problem.

In SAPHIRE 8, INL has included a BDD solver that currently operates only on fault tree logic. To use this option, the "BDD solve" method must be enabled in the user settings (Project → User Settings → General Analysis). Once enabled, this analysis method appears as a new Project option.

6.3 Quantifying Sequences

An accident sequence begins with an initiating event, which traditionally has a frequency f . The units of the frequency are 1/time, and there is no theoretical upper bound on its possible value. This distinguishes a frequency from a probability, which is unitless and bounded by 1.0.

The actual units of the initiator frequency are specified and stored in the SAPHIRE relational database.

After the initiating event, various systems in the plant are required to function. Depending on whether they function or not, the sequence can proceed to different possible plant states. Consider one of these systems. Given the assumed initiating event and the success or failure of the systems that were invoked earlier in the sequence, the probability of the system's failure is quantified by a fault tree for the system. For each such sequence of interest, SAPHIRE constructs and simplifies the fault tree for the entire sequence, by combining the fault trees for the failed systems and the negation of the fault trees for the successful systems, as described in Section 5.

Let S be the probability of the sequence fault tree(s) (which are conditional upon the initiating event and specific sequence), evaluated using the minimal cut set upper bound or the rare event approximation. It is a probability calculated assuming that the initiating event has occurred. Then, the frequency of the sequence is the product $f \times S$. In this way, sequence frequencies are found. SAPHIRE treats the initiating event as being separate from the cut sets generated from the associated accident sequence fault trees.

7. EVENT PROBABILITY CALCULATION TYPES

The calculation type specifies the method to be used to calculate the basic event probability. Thirteen types are available in SAPHIRE, and they are summarized in Table 1. The resulting probability for Types 1 through 7 will be the mean used in the uncertainty analysis described in Section 9.

Table 1. SAPHIRE Calculation Types.

Type	Calculation Method
1	Probability (bounded between 0 and 1)
3	Operating Component with no Repair: $1 - \text{Exp}(-\text{Lambda} * \text{Mission Time})$
5	Operating Component with Repair
7	Standby Component: $1 + (\text{Exp}(-\text{Lambda} * \text{Tau}) - 1.0) / (\text{Lambda} * \text{Tau})$
T	Set to House Event (Failed, Probability = 1.0)
F	Set to House Event (Successful, Probability = 0.0)
I	Ignore this Event (Remove it from logic)
C	Compound Event (see Section 11)
S	Set to System Min Cut Upper Bound
E	Set to End State Min Cut Upper Bound
G	Enter a Ground Acceleration for Screening
H	Use the maximum value on the Hazard Curve for a screening G-Level
V	Value (any real number, positive or negative)
N	Initiating event frequency (any positive real number)
X	Human Factor Event (uses performance shaping factors to calculate a probability)
R	RASP common cause failure
U	Indicates a component is failed (unknown type), used with Calculation Type R
D	Indicates a component is failed (dependent type), used with Calculation Type R
Y	Indicates a component is failed (independently), used with Calculation Type R

7.1 Calculation Type 1

Probability

Calculation Type 1 takes the number specified by the user in the Probability field as the basic event failure probability. This is the type used for demand probabilities.

SAPHIRE will accept numbers in scientific or decimal format. For example, 1.E-4 and 0.0001 are both valid inputs.

NOTE: SAPHIRE will accept an uppercase E or a lowercase e. Also, note that 1.0E- 020 is not the same as 1.0E- 02.

7.2 Calculation Type 3

1 - Exp (- Lambda * Mission Time)

Calculation Type 3 uses the equation for failure probability for an operating component without repair,

$$q = 1 - e^{-\lambda t} \quad (7-1)$$

where

q = failure probability of the basic event,
 λ = failure rate per hour, and
 t = mission time hours.

7.3 Calculation Type 5

Operating Component with Repair (Full Equation)

Calculation Type 5 is the equation for the failure probability of an operating component with repair. The equation is

$$q = \frac{\lambda \tau}{1 + \lambda \tau} (1 - e^{-(\lambda + 1/\tau)t})$$

where (7-2)

q = failure probability of the basic event,
 λ = failure rate per hour,
 t = mission time hours, and
 τ = average time to repair hours.

7.4 Calculation Type 7

1 + (Exp(- Lambda*Tau)- 1.0) / (Lambda * Tau)

Calculation Type 7 is the equation for the failure probability of a standby component with a surveillance test interval. The equation is

$$q = 1 + \frac{e^{-\lambda T} - 1}{\lambda T} \quad (7-3)$$

where

q = failure probability of the basic event,
 λ = standby failure rate per hour, and
 T = surveillance test interval hours.

7.5 Calculation Types T, F, and I

Calculation Types T, F, and I are used to set basic events to house events.

Set to House Event TRUE (Failed, Prob=1.0)

Calculation Type T turns the basic event into a house event that always occurs (probability 1.0). Note that a TRUE event will not appear in minimal cut sets unless it is the only event in a cut set (i.e., the entire cut set is TRUE).

Set to House Event FALSE (Successful, Prob=0.0)

Calculation Type F turns the basic event into a house event that never occurs (probability 0.0). If the event states that a component fails, T forces the component to fail while F forces it to succeed. Note that if truncation is used, basic events set to FALSE will not appear in minimal cut sets.

Ignore this Event (Remove it from logic)

Calculation Type I indicates that the basic event is to be treated as if it did not exist in the logic for the fault tree.

Setting an event to a house event actually changes the logic of the fault tree, pruning appropriate branches and events from the fault tree. Therefore, the flags on the affected fault trees will indicate a need to generate new cut sets rather than just requantifying existing cut sets.

7.6 Calculation Type C

Use compound event

Calculation Type C indicates that the event will use the SAPHIRE compound event feature. Compound events are basic events that utilize an external (to SAPHIRE) calculation to determine its probability. These external calculations are contained within a Windows-

compatible library, called a dynamic link library (DLL). A basic event that has been set to “C” utilizes a DLL, where this DLL can be contained either in the project folder or the SAPHIRE tools folder. Typically, the common DLLs are stored in the SAPHIRE tools folder and include calculations such as common cause failure probabilities and general calculations (via the “utility” library). SAPHIRE will first check the project folder for the DLL and, if not found there, will search the SAPHIRE tools folder.

The "utility" compound event library has procedures for general operations such as adding basic events together, multiplying events, dividing events, etc. In other words, it enables a handful of general purpose mathematical calculations to be performed on any basic events. Included in this library is the “min-cut” calculation which will take the minimal cut set upper bound calculation on the specified basic events (up to 20 per compound event). One could utilize this calculation to make a model via the “supercomponent” approach where a basic event is represented by other lower-level basic events. The minimal cut sets would only contain the higher-level event (the supercomponent) but the event’s probability would automatically be determined by the compound DLL calculation.

7.7 Calculation Type S

Set to System Min Cut Upper Bound

Calculation Type S indicates that the probability of the basic event is to be determined by finding a system (i.e., fault tree) with the same name as the basic event. Then, use the minimal cut set upper bound (or default quantification method specified for the system) for this system as the failure probability for the basic event.

7.8 Calculation Type E

Set to End State Min Cut Upper Bound

Calculation Type E indicates that the probability of the basic event is to be determined by finding an end state with the same name as the basic event. Then, use the minimal cut set upper bound for this end state as the failure probability for the basic event.

7.9 Calculation Type G

Enter Ground Acceleration for Screening

Calculation Type G indicates that the basic event is to be treated as a seismic event. The Screening G-Value field will be the ground acceleration to be used to determine the probability of failure given that specified ground acceleration level and the applicable seismic hazard curve.

7.10 Calculation Type H

Use Hazard Curve for screening G-Level

Calculation Type H indicates that the probability for this event is to be determined from the hazard curve histogram by taking the highest ground acceleration level from the hazard curve.

7.11 Calculation Type V

Value

Calculation Type V takes the real number specified by the user in the Probability field as the basic event numerical value. This value could be a real number, either positive or negative. Use of this calculation type includes storing consequence-types estimates (e.g., person-rem) to be multiplied on accident sequences.

7.12 Calculation Type N

Initiating Event frequency

Calculation Type N takes the real, positive number specified by the user in the Frequency field as the basic event numerical value. In addition to the value, the units (e.g., per hour, per year) must be specified in the Frequency Units field.

7.13 Calculation Type X

Human Error Event

Calculation Type X indicates that the event will use the human error probability (HEP) worksheets based on the Standardized Plant Analysis Risk (SPAR) Human Reliability Analysis (HRA) methodology (Gertman, *et al*, 2004). Events with a Calculation Type X use a built-in "worksheet" to calculate the HEP. The HEP is calculated based on whether the operator requires an action only, an action with diagnosis, and if there is dependence between other operator actions. For each calculation, Performance Shaping Factors (PSFs) can be applied to adjust the HEP value. PSFs include: available time, stress/stressors, complexity, experience/training, procedures, ergonomics, fitness for duty, and work processes.

7.14 Calculation Types R, U, D, and Y

In SAPHIRE 8, four new calculation types will be created for the CCF modifications:

- R to indicate a CCF object (this replaces the old CCF plug-in approach)
- U to indicate component is failed (unknown type)
- D to indicate a component is failed (dependent type)
- Y component is failed (independently)

When the CCF adjustments are being made via the Boolean reduction, SAPHIRE will key on any independent events that are set to a calculation type of U, Y, or D. Details of the new CCF calculations are described in Appendix B.

8. IMPORTANCE MEASURES

8.1 Types of Importance Measures

Early work in the development of importance measures is best represented by the IMPORTANCE code developed by Lambert in the early 1970s. (Lambert, 1975) This analysis tool built upon the theoretical development completed by early pioneers such as Barlow, Birnbaum, Fussell, Proschan, and Vesely. The IMPORTANCE software calculated results for different types of importance measures, ranging from Fussell-Vesely to Birnbaum to Barlow-Proschan metrics. The SAPHIRE calculates seven different basic event importance measures. These are:

- Fussell-Vesely
- Risk reduction ratio
- Risk increase ratio
- Birnbaum (the so-called first derivative importance)
- Uncertainty Importance
- Risk reduction difference
- Risk increase difference

These importance measures are calculated for each basic event for the respective fault tree, accident sequence, or end state.

The ratio importance measures are dimensionless and consider only relative changes. The difference definitions account for the actual risk levels that exist and are more appropriate when actual risk levels are of concern, such as comparisons or prioritizations across different plants. For purely relative evaluations, such as prioritizations within a plant, the ratios sometime give more graphic results.

The main importance measures are:

- Fussell-Vesely importance, an indication of the percentage of the minimal cut set upper bound contributed by the cut sets containing the basic event
- Risk reduction, an indication of how much the minimal cut set upper bound would decrease if the basic event never occurred (typically, if the corresponding component never failed)
- Risk increase, an indication of how much the minimal cut set upper bound would go up if the basic event always occurred (typically, if the corresponding component always failed)

Also available on each of the importance measure reports is an indication of another type of measure, the structural importance, which is the number of cut sets that contain the basic event. For example, if a total of 1,000 cut sets exist for a fault tree and basic event XYZ is in 300 cut sets, then SAPHIRE would indicate that that event appears 300 times.

In SAPHIRE, the Basic Event Importance display lists the basic event name, its failure probability, the number of cut sets in which the basic event occurs, and three of the six importance measures. The user can choose to display either ratios or differences. If the user selects ratios then the Fussell-Vesely importance, risk reduction ratio, and risk increase ratio are displayed together. Otherwise, the Birnbaum importance, risk reduction difference, and risk increase difference are displayed together. The list can be sorted on any column in the display.

The exposition below is written in terms of fault trees and event probabilities. However, SAPHIRE also can calculate importances for events in sequences or end states. Recall that a sequence is simply a fault tree preceded by an initiating event with frequency f , where f usually has units of 1/time. The frequency of any event in the fault tree is f times the probability of the event. Therefore, the ratio importances are unchanged whether the event is part of a fault tree or a sequence. A difference importance for an event in a sequence is f times the importance of the event in the fault tree. The maximum possible value of a difference importance is 1.0 if the event is in a fault tree and f if the event is in a sequence. This alternative formulation is indicated below by phrases in parentheses.

8.2 Calculation Details

This section contains the calculation definition of the importance measures. Examples are given in Section A.6 of Appendix A. Both the ratio and the difference are discussed in the appropriate sections. For the basic event under consideration, several notations are used repeatedly.

$F(x)$ = minimal cut set upper bound (sequence frequency) evaluated with the basic event probability at its mean value.

$F(i)$ = minimal cut set upper bound (sequence frequency) evaluated for all cut sets containing the i 'th basic event

$F(0)$ = minimal cut set upper bound (sequence frequency) evaluated with the basic event probability set to zero.

$F(1)$ = minimal cut set upper bound (sequence frequency) evaluated with the basic event failure probability set to 1.0.

Fussell-Vesely Importance

The Fussell-Vesely importance is an indication of the fraction of the minimal cut set upper bound (or sequence frequency) that involves the cut sets containing the basic event of concern. Or, alternatively, the Fussell-Vesely is "the probability that event i is contributing to system failure, given that system failure occurred." (Henley and Kumamoto, 1981). It is

calculated by finding the minimal cut set upper bound of those cut sets containing the basic event of concern and dividing it by the minimal cut set upper bound of the top event (or of the sequence). Prior to SAPHIRE version 7.27, this calculation was performed by determining the minimal cut set upper bound (sequence frequency) with the basic event failure probability at its mean value and again with the basic event failure probability set to zero. The difference between these two results is divided by the base minimal cut set upper bound to obtain the Fussell-Vesely importance. The Fussell-Vesely importance FV was approximated by the equation

$$FV = F(i)/F(x)$$

Prior to SAPHIRE version 7.27, the FV equation used the expression $FV \approx 1 - F(0)/F(x)$.

Risk Reduction

The risk reduction importance measure is an indication of how much the results would be reduced if the specific event probability equaled zero, normally corresponding to a totally reliable piece of equipment. The risk reduction ratio is determined by evaluating the fault tree minimal cut set upper bound (or the sequence frequency) with the basic event probability set to its true value and dividing it by the minimal cut set upper bound (sequence frequency) calculated with the basic event probability set to zero. In equation form, the risk reduction ratio RRR is

$$RRR = F(x)/F(0)$$

The risk reduction difference indicates the same characteristic as the ratio, but it reflects the actual minimal cut set upper bound (sequence frequency) levels instead of a ratio. This is the amount by which the failure probability or sequence frequency would be reduced if the basic event never failed.

The risk reduction difference (RRD) is calculated by taking the difference between the mean value and the function evaluated at 0. In equation form, the risk reduction difference RRD is

$$RRD = F(x) - F(0) .$$

Risk Increase

The risk increase ratio is an indication of how much the top event probability (frequency) would go up if the specific event had probability equal to 1.0, normally corresponding to totally unreliable equipment. The risk increase ratio is determined by evaluating the minimal cut set upper bound (sequence frequency) with the basic event probability set to 1.0 and dividing it by the minimal cut set upper bound evaluated with the basic event probability set to its true value. In equation form, the risk increase ratio RIR is

$$RIR = F(1)/F(x)$$

The risk increase difference RID is calculated by taking the difference between the function evaluated at 1.0 and the nominal value. In equation form, the risk increase difference RID is

$$RID = F(1) - F(x)$$

Birnbaum Importance

The Birnbaum importance measure is calculated in place of the Fussell-Vesely importance measure when differences are selected instead of ratios. The Birnbaum importance is an indication of the sensitivity of the minimal cut set upper bound (or sequence frequency) with respect to the basic event of concern. It is calculated by determining the minimal cut set upper bound (or sequence frequency) with the basic event probability of concern set to 1.0 and again with the basic event probability set to 0.0. The difference between these two values is the Birnbaum importance. In equation form, the Birnbaum importance B is

$$B = F(1) - F(0)$$

The Birnbaum importance can be interpreted as follows as an approximation of a derivative. If basic event i has probability p_i , the rare-event approximation says that the top event probability, F , can be written approximately as a linear function of p_i , and therefore, $\partial F/\partial p_i$ is approximately equal to $[F(1)-F(0)]/(1-0)$ which equals the Birnbaum importance.

Uncertainty Importance

The uncertainty importance of basic event i is defined by Iman and Shortencarrier (1986) as $\sigma_i \partial F/\partial p_i$, where p_i is the probability of event i and σ_i is the standard deviation of p_i , reflecting uncertainty in p_i , and F is defined above. SAPHIRE calculates the uncertainty importance as $F_i B_i$, where B_i is the Birnbaum importance of event i . By the above discussion of the Birnbaum importance, the SAPHIRE calculation uses the rare-event approximation of the derivative.

The uncertainty in each input parameter, as expressed through its probability distribution, contributes to the uncertainty in the output parameter of interest (e.g., core damage frequency, loss of mission). The uncertainty importance measure in SAPHIRE quantifies the contribution of each individual basic event's uncertainty to this total output uncertainty. The measure used in SAPHIRE is based on a first-order Taylor series expansion of the variance of the output of interest. The equation used by SAPHIRE is:

$$Var(R) = \sum_{i=1}^n \left(\frac{\partial R}{\partial p_i} \right)^2 \sigma_i^2$$

where R is the output of interest, p_i is the probability of the i^{th} basic event, and σ_i^2 is the variance of the uncertainty distribution for the i^{th} event, as listed in Table 2. This approximation, which hinges upon the basic events being mutually statistically independent, says that the variance of the output is approximately the sum of n separate contributions, one from each basic event. The magnitude of each contribution (each contribution is positive) measures how much of the output variance is contributed by each basic event. Because it is more convenient, SAPHIRE uses the square root of each individual contribution as the uncertainty importance:

$$I_{unc} = \frac{\partial R}{\partial p_i} \sigma_i$$

where σ_i is the standard deviation of the uncertainty distribution of the i^{th} basic event. Note that the partial derivative in the above equation for the uncertainty importance is, by definition, the Birnbaum importance of that event. Therefore, no new calculations are needed; the uncertainty importance is the Birnbaum importance multiplied by the standard deviation of the input probability distribution. The standard deviation, σ_i , for each basic event is calculated depending upon its defined uncertainty distribution. Table 2 outlines the standard deviation calculations.

Table 2. Standard deviation calculations for the supported uncertainty distributions.

Distribution Type	Parameter 1	Parameter 2	Standard Deviation (<i>dev</i>)
Normal	Standard deviation	None	<i>dev</i> = standard deviation
Lognormal	EF = Error factor	None	$\sigma = \ln(EF/1.645)$ $dev = \mu * \sqrt{\exp(\sigma^2) - 1}$
Beta	B in Beta(a,b)	None	$dev = (1 - \mu) * \sqrt{\mu / ((1 - \mu) + B)}$
Chi-Squared	D - Degrees of freedom	None	$dev = \mu * \sqrt{2/D}$
Gamma	r in $\Gamma(r)$	None	$dev = \mu / \sqrt{r}$
Exponential	None	None	$dev = \mu$
Uniform	U = Upper end point	None	$dev = (U - \mu) / \sqrt{3}$
Max Entropy	L-Lower end point	U- Upper end point	$dev = \sqrt{\text{variance}}$ See note below for variance information
Dirichlet	B in Beta (a,b)	None	See, for example, www.wikipedia.com for information on the standard deviation.
Triangular	Mode	Upper End	$dev = \frac{1}{18} (a^2 + b^2 + c^2 - ab - ac - bc)$
Histogram	Histogram identifier	None	Depends on the histogram
Seismic Lognormal	Beta r	Beta u	$dev = 0.0$

NOTE: For the maximum entropy distribution, the variance is found from:

```

IF (U < 1.5E+306) THEN
  beta = BetaFromMaxEntMu(mean,L,U,error);
    (* Interpolated from mean and U and L *)
  g = beta * (U-L);
  IF ABS(g3) < (10.0*1.0e-13) THEN
    (* use 2nd order Taylor expansion *)
    variance = (U - L)2 / 12 * (1 - g2 / 20)
  ELSE
    variance = (U - L)2 / 4 * (1 + 4 / g2 - (exp(g) + 1) / exp(g) - 1)2)
  END;
ELSE
  variance := mean*mean;
END;

```

It should be pointed out that SAPHIRE 8 can also calculate the uncertainty on any of the importance measures, including the “uncertainty” importance measure. The “uncertainty” importance measure indicates the contribution to the overall uncertainty from a single basic event. The uncertainty on the importance measures (e.g., Fussell-Vesely, Birnbaum, RRR, RIR) indicates the uncertainty in the importance measure estimate. For example, we may calculate a RIR for a basic event as 4.3. However this value is a point estimate. Performing the uncertainty analysis on the RIR calculation may indicate that the 5th percentile for the basic event is 2.5, the mean value is 5.1, and the 95th percentile is 6.8. Further, the “uncertainty” importance measure for this same event may indicate that it has a low contribution to the overall fault tree, sequence, or end state uncertainty (depending on what is being analyzed). A basic event may have a large uncertainty itself and its importance measures may be quite uncertain, but its contribution to the overall uncertainty can be negligible.

In general, in SAPHIRE 8, the uncertainty may be estimated on any of the resultant risk measures, including:

- Fault tree cut set
- Fault tree importance measures
- Sequence cut sets
- Sequence importance measures
- End state cut sets
- End state importance measures
- ECA Workspace results
- General Analysis Workspace results

9. UNCERTAINTY AND MONTE CARLO

9.1 Uncertainty and Monte Carlo

The uncertainty analysis allows the user to calculate the uncertainty in the top event probability resulting from uncertainties in the basic event probabilities. To use this option, the user must have previously loaded or generated the cut sets and loaded the component reliability information and distribution data. Bohn et al. (1988) contains an excellent discussion of uncertainty analysis. A very brief overview is given here, with elaborations in the subsequent sections.

In an uncertainty analysis, SAPHIRE already has the top event expressed in terms of minimal cut sets, either generated earlier or loaded from some other source. These cut sets depend on many basic events, each of which has a probability described in terms of some parameter(s). For definiteness in this explanation, suppose that a basic event probability depends on the parameter λ . The value of λ for each basic event is not known exactly, but is estimated based on data or on expert opinion. The uncertainty in λ is quantified by a probability distribution: the mean of the distribution is the best estimate of λ , and the dispersion of the distribution measures the uncertainty in λ , with a large or small dispersion reflecting large or small uncertainty, respectively, in the true value of λ . This distribution is the *uncertainty distribution of λ* .

SAPHIRE treats an initiating event of a sequence as a basic event. It differs from the other basic events in only two ways: it can have a frequency instead of a probability, and every cut set contains exactly one initiating event.

For all the basic events, SAPHIRE randomly samples the parameters from their uncertainty distributions, and uses these parameter values to calculate the probability of the top event. This sampling and calculation are repeated many times, and the uncertainty distribution for the probability of the top event is thus found empirically. The mean of the distribution is the best estimate of the probability of the top event, and the dispersion quantifies the uncertainty in this probability. For an accident sequence the process is the same, except the sequence fault tree is preceded by an initiating event, whose frequency is also quantified by an uncertainty distribution. The term *Monte Carlo* is used to describe this analysis by repeated random sampling. Two kinds of Monte Carlo sampling are simple Monte Carlo sampling and Latin Hypercube sampling; they are described and compared in Sections 9.6 through 9.8.

9.2 Basic Uncertainty Output

The Monte Carlo procedure computes the probability distribution of a fault tree top event or accident sequence using the assigned probability distributions for each basic event contained in the minimal cut sets. By using the probability distributions for the basic events, the uncertainty in the system unavailability can be calculated.

The first step in the process of computing the uncertainty in the minimal cut set upper bound is to provide a measure of the uncertainty for each basic event contained in the minimal cut sets. SAPHIRE then computes the minimal cut set upper bound for a set of random samples from the uncertainty distributions of the basic events. After calculating the minimal cut set upper bound, SAPHIRE computes the first four moments of the distribution and the 5th, 50th, mean, and 95th percentile values.

The moments are calculated as a basis for comparison of the calculated distribution with other distributions (McGrath and Irving 1975). From the first four moments, the sample mean, sample variance, coefficient of skewness, and coefficient of kurtosis can be calculated. To establish some standard notation, the following symbols are used:

- n = the number of samples calculated.
- x_i = i th data value for $i = 1, 2, 3, \dots n$.

The sample mean, given as \bar{x} , can be defined as

$$\bar{x} = \sum_{i=1}^n \frac{x_i}{n} \tag{9-1}$$

and the sample variance, given as

$$s^2 = \sum_{i=1}^n \frac{(x_i - \bar{x})^2}{n-1} \tag{9-2}$$

The k -th sample moment about the mean is next defined in general as

$$m_k = \sum_{i=1}^n \frac{(x_i - \bar{x})^k}{n-1} \tag{9-3}$$

Thus, from the third moment, the coefficient of skewness, $\beta_1^{1/2}$, is

$$\beta_1^{1/2} = \frac{m_3}{s^3} \tag{9-4}$$

and from the fourth moment, the coefficient of kurtosis, β_2 , is

$$\beta_2 = \frac{m_4}{s^4} \quad (9-5)$$

where

s = the square root of the variance.

The coefficient of skewness and the coefficient of kurtosis are generally used as measures for comparison with the normal distribution. If the skewness is close to zero while the kurtosis is approximately three, the normal distribution is a good approximation. A zero skewness value indicates a symmetric distribution; a negative skewness indicates a long left tail, while a positive value indicates a long right tail. If the kurtosis is greater than three, the distribution is more peaked than the normal distribution, and has more weight in the tails. However, if the value is less than three, the distribution is flatter than the normal, and has less weight in the tails.

9.3 Uncertainty Analysis Input Data

The parameters for the probability of a basic event, discussed in Section 7, are input in the Failure Data area of the SAPHIRE input screen. Now we move to the Uncertainty Data area using the arrow keys or the tab key. The fields in this area that can be accessed from this menu are the current case distribution type, a distribution parameter value, and a correlation class.

Currently, SAPHIRE supports 13 distributions

- Beta
- Chi-squared
- Constrained Noninformative
- Dirichlet
- Exponential
- Gamma
- Histogram distributions
- Lognormal
- Maximum Entropy
- Normal
- Seismic Lognormal
- Triangular
- Uniform

Most distributions can be defined with two statistical parameters, although some take more. The first parameter is the mean failure probability and the second parameter (and third if applicable) is specific to the particular uncertainty distribution. The mean failure probability is calculated from the data input in the Failure Data area just discussed. For more clarity,

SAPHIRE allows the user to input the parameters of the distribution directly. It will check them for consistency with the mean.

Correlation classes, as explained in Section 9.6, are used to identify basic events whose failure data are derived from the same data source. This information is used in the uncertainty analysis. Correlation classes consist of up to a 24 character string. A blank correlation class indicates that there are no data dependencies. When running the uncertainty analyses, the same sample value will be used for all basic events with the same correlation class.

NOTE: The user must set up a correlation class labeling scheme for the basic events in the database. For example, correlation class MDPFTS may be assigned to motor- driven pumps fail to start, correlation class MDPFTR to motor- driven pumps fail to continue to run, correlation class CKVFTC to check valves fail to close, and so on.

SAPHIRE provides more sophisticated ways of entering failure and uncertainty data that reduce the amount of data input required and ensure consistency among like basic events. These techniques utilize the "template" feature in SAPHIRE. A basic event can be identified as being a template, and then other events may "borrow" information from the template event. For example, if a database has 20 basic events for different valves, rather than entering the same failure data 20 times, the user can enter the data once and denote this basic event as a "template." Then for the 20 basic events, the user would indicate that each valve event should use the template event containing the failure data of interest. In total, the database would contain 21 basic events (20 valves and one valve template), but the amount of data entry is greatly reduced.

9.4 Distributions

At the present time, as mentioned above, SAPHIRE supports 13 uncertainty distributions. Within these distributions, only the histogram distribution requires detailed information to be fully specified. Each of the other distributions is described by its mean and typically one (or at most two) additional parameter. Table 3 summarizes this information for each of the supported distributions except for the histogram distribution, which is explained separately in Section 9.5. These distributions in Table 3 are described in more detail in this section. Additional detail about these distributions can be found in Mood et al. (1974) and Hahn and Shapiro (1967).

Table 3. Uncertainty distributions supported in SAPHIRE.

Distribution	Code	Parameter
beta	B	b in beta(a, b)
chi-squared	C	Degrees of freedom
constrained noninformative	O	None
Dirichlet	D	B
Exponential	E	None
gamma	G	r in gamma(r)
lognormal	L	95% error factor
maximum entropy	M	a = lower end range b = upper end range
normal	N	Standard deviation
Seismic lognormal	S	None
Triangular	T	Mode (highest point) Upper end point
Uniform	U	Upper end point

Beta Distribution

The general parameters of the beta distribution are a and b . The probability density function is given by

$$f(x) = \left[\frac{1}{B(a, b)} \right] x^{a-1} (1-x)^{b-1}$$

for $0 \leq x \leq 1$, where $B(a, b)$ is the beta function. In SAPHIRE, the parameter b is used in addition to the mean to define the distribution. Then, the parameter a is calculated from the mean value by the formula

$$a = \mu * b / (1 - \mu)$$

where $\mu = a/(a+b)$ is the mean of the beta distribution. Note that the mean of the beta distribution is between 0 and 1.

SAPHIRE generates a beta random variable using the fact that if X is $\chi^2(2a)$ and Y is $\chi^2(2b)$ and X and Y are independent then $X/(X + Y)$ has a beta(a, b) distribution. See Section 24.2 of Johnson and Kotz (1970).

For LHS sampling, the inverse cumulative distribution function (CDF) method is used, with the inverse of the CDF computed by numerical iteration (with the method of false position) on the

beta CDF. The beta CDF is evaluated using the BETAI function of Press et al. (1986). Note, this way of generating the LHS sample is not fast, and simple Monte Carlo sampling with a larger sample may be more efficient than LHS sampling when many beta distributions must be sampled.

Chi-Squared Distribution

The chi-squared distribution is directly related to the gamma distribution, as follows. Let X have a $\text{gamma}(\lambda, r)$ distribution. Then $2\lambda X$ has a chi-squared distribution with $2r$ degrees of freedom, denoted $\chi^2(2r)$. For this reason, the chi-squared distribution is an option in SAPHIRE only as a convenience to the user. Anything that requires a chi-squared distribution can be accomplished using a gamma distribution.

The mean of a $\chi^2(k)$ distribution equals k and the variance equals $2k$, for degrees of freedom $k > 0$. Note that the mean of a chi-squared distribution determines the variance. This is not flexible enough for most uncertainty analyses. Therefore, when SAPHIRE is asked for a chi-squared random variable with k degrees of freedom, it generates a *multiple* of a chi-squared random variable, $Y = aX$, where X is $\chi^2(k)$ and $a = \mu/k$. This results in a random variable with mean μ and variance $2\mu^2/k$. Exactly the same distribution would be obtained by specifying a gamma distribution with mean μ and $r = k/2$.

For simple Monte Carlo sampling, SAPHIRE generates a multiple of a chi-squared random variable with mean μ and k degrees of freedom by generating a gamma random variable with mean μ and with $r = k/2$.

For LHS sampling, SAPHIRE considers several special cases, and uses the inverse CDF method in every case. Let k be the degrees of freedom. When $k = 1$, the random variable is the square of a normal random variable, and SAPHIRE finds the inverse CDF based on the inverse of a normal CDF. When $k = 2$, the distribution is exponential, and SAPHIRE finds the inverse CDF explicitly. For all other values of k , SAPHIRE begins with the Wilson-Hilferty approximation of the chi-squared inverse CDF. (Section 5.10.2. of Thisted 1988), or 0 if the Wilson-Hilferty approximation is negative. It then refines this approximation by number i iteration (the method of false position), obtaining values for which the chi-squared CDF is ever closer to the specified value of the CDF. The chi-squared CDF is computed as the equivalent gamma CDF, calculated using the algorithm GAMMP given by Section 6.2 of Press et al. (1986). When the Wilson-Hilferty approximation is good, typically in the right tail of the distribution with k not very small, very little time is required for the refinement.

Constrained Noninformative Distribution

The constrained noninformative prior distribution is a diffuse distribution with a specified mean. The amount of diffuseness is set so that the distribution is noninformative except for the information given by the specified mean, neither too concentrated nor too diffuse. The precise definition is given by Atwood (1994). The parameter of interest is either an initiating event frequency or a basic event probability. We proceed by transforming the model so that the

parameter of interest is approximately a location parameter. In this transformed model, we find the maximum entropy distribution constrained by the specified mean for the original parameter. This distribution is as flat as possible subject to the constraint. Then we transform back to the original model. The resulting distribution is the constrained noninformative distribution on the parameter.

The constrained noninformative distribution is a special type of maximum entropy distribution. As Atwood (1996) points out, if the uncertainty distribution has a finite range (which, for probabilities, it does, bounded between 0 and 1), then the function that maximizes entropy is a uniform distribution. A limitation in *unconstrained* noninformative distributions is that the mean value of a uniform 0-to-1 distribution is 0.5. Consequently, the prior distribution in any Bayesian calculation, having a mean of 0.5, would tend to pull the posterior distribution toward a mean value of 0.5. It was this limitation that motivated Atwood to develop the constrained noninformative distribution, where the constraint is that the prior distribution has a user-specified mean rather than a mean of 0.5. Once the mean is specified, the analyst may use Atwood (1996) to determine an approximate distribution based on a beta distribution. The beta distribution requires two parameters, α and β . Atwood (1996) supplies a table of applicable α parameters (as a function of the mean), but numerical values of α as a function of the mean are shown in Figure 16. The second parameter, β , is found via the equation:

$$\beta = \frac{\alpha(1 - \text{mean})}{\text{mean}}$$

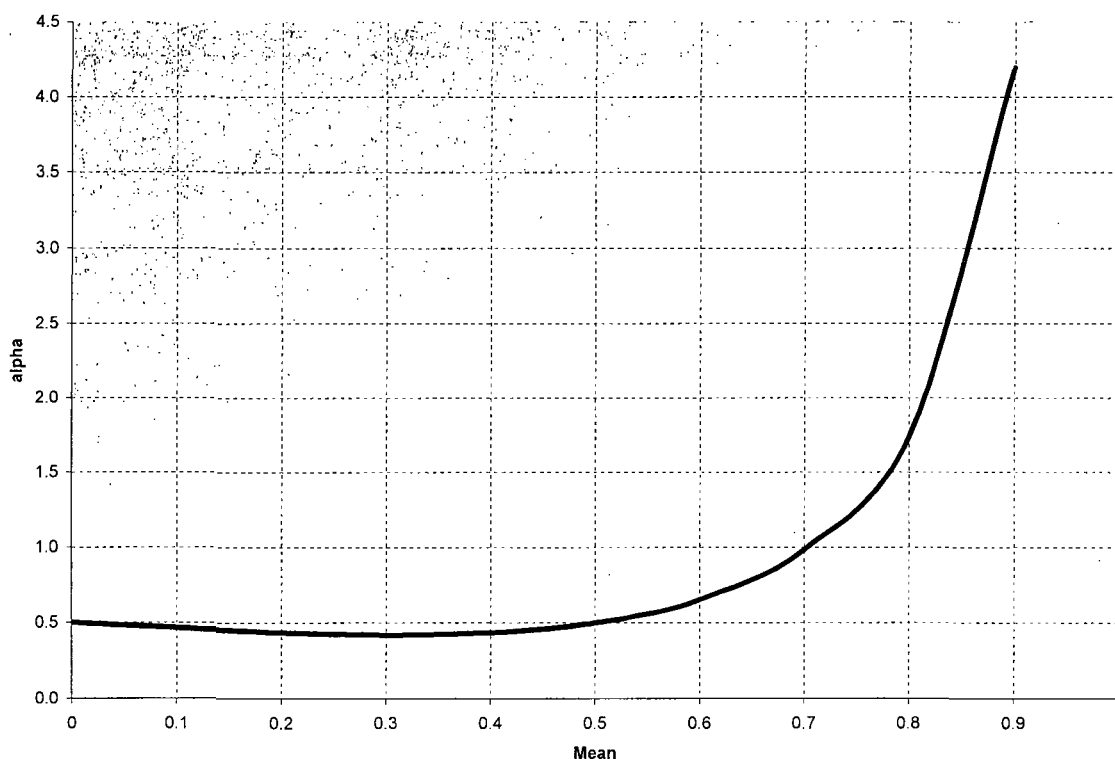


Figure 14. Constrained noninformative alpha parameter as a function of the mean value

Dirichlet Distribution

A k -variate random vector, (X_1, X_2, \dots, X_k) is *Dirichlet* distributed with parameters $\alpha_1, \alpha_2, \dots, \alpha_k$ and α_{k+1} if it has joint probability density function

$$f(x_1, x_2, \dots, x_k) = \frac{\Gamma(\alpha_1 + \alpha_2 + \dots + \alpha_{k+1})}{\Gamma(\alpha_1)\Gamma(\alpha_2) \dots \Gamma(\alpha_{k+1})} x_1^{\alpha_1-1} x_2^{\alpha_2-1} \dots x_k^{\alpha_k-1} (1 - x_1 - x_2 - \dots - x_k)^{\alpha_{k+1}-1}$$

for $0 < x_i$ and $x_1 + x_2 + \dots + x_k < 1$.

The Dirichlet distribution is a generalization of the Beta distribution with parameters a and b if we let $k = 1$, $a = \alpha_1$ and $b = \alpha_2$. The mean of the i th Dirichlet variable, X_i , is

$$\mu_i = \alpha_i / D$$

where the denominator D is defined as

$$D = \alpha_1 + \alpha_2 + \dots + \alpha_{k+1}.$$

Thus, an alternative parameterization is in terms of the means $\mu_1, \mu_2, \dots, \mu_k$ and either D or $\beta = \alpha_{k+1}$.

The marginal distributions of the individual Dirichlet variables are Beta distributions. In particular, the i th variable, X_i , has the Beta distribution with parameters $a = \alpha_i$ and $b = D - \alpha_i$. A set of k -variate Dirichlet variables are dependent random variables, but it is possible to transform from a set of k independent Beta distributed random variables into dependent Dirichlet random variables. Specifically, if Y_1, \dots, Y_k are independent random variables, such that the i th variable, Y_i , has a Beta distribution with parameters $a = \alpha_i$ and $b = D - \alpha_1 - \dots - \alpha_{i-1}$, then the vector of k random variables (X_1, X_2, \dots, X_k) defined by $X_1 = Y_1$ and

$$X_i = (1 - X_1 - \dots - X_{i-1})Y_i \text{ for } 2 \leq i \leq k,$$

are distributed as k -variate Dirichlet with parameters $\alpha_1, \alpha_2, \dots, \alpha_k$ and β .

It is also true that a subset of Dirichlet variables is also Dirichlet distributed. For example, if $1 \leq i < j \leq k$, then the pair (X_i, X_j) has the 2-variate Dirichlet distribution with parameters α_i, α_j and $D - \alpha_i - \alpha_j$.

As an example, suppose a branch in an event tree has three mutually exclusive failure events A_1, A_2, A_3 and the complement (success) event B . Suppose the uncertainty in the probabilities of these events is modeled using a 3-variate Dirichlet distribution. In other words, $P(A_1), P(A_2)$ and $P(A_3)$ are modeled as Dirichlet with nominal values μ_1, μ_2 and μ_3 . The remaining parameter β is the nominal value of $P(B)$. The alternate parameterization in terms of D is useful, because it is involved in the variances of the individual Dirichlet variables. Specifically, the i th variable has variance $\mu_i(1 - \mu_i)/(D + 1)$, so a smaller value of D corresponds to a large variance, which in turn means greater uncertainty about $P(A_i)$. Similarly, a large value of D corresponds

to a small variance or less uncertainty. Variables with this distribution can be generated by first generating independent Beta variables, Y1 with parameters $a = \alpha_1$ and $b = D - \alpha_1$, Y2 with parameters α_2 and $b = D - \alpha_1 - \alpha_2$, and Y3 with parameters α_3 and $D - \alpha_1 - \alpha_2 - \alpha_3 = \beta$. Then, the required Dirichlet variables would be $X_1 = Y_1$, $X_2 = (1 - X_1)Y_2$ and $X_3 = (1 - X_1 - X_2)Y_3$.

If only one branch (say the first) is of interest, then the uncertainty can be given in terms of a Beta variable with mean $\mu_1 = \alpha_1/D$ and $b = \alpha_2 + \alpha_3 + \beta = D - \alpha_1$. Suppose it is required to consider jointly two of the events (say A1 and A2). The "joint" uncertainty is modeled by a 2-variate Dirichlet distribution with parameters α_1 , α_2 and $\beta = D - \alpha_1 - \alpha_2 = \alpha_3 + \beta$.

Exponential Distribution

The exponential distribution is commonly used for modeling a time to failure, but it is generally not used for modeling uncertainties. One reason for its use in modeling failures and its disuse in modeling uncertainties is that it has only one parameter. Therefore the mean determines the variance. The exponential density is

$$f(x) = \lambda e^{-\lambda x}$$

where the parameter λ and the mean μ are related by $\mu = 1/\lambda$. Note that the exponential density is a special case of the gamma density, with the gamma parameter $r = 1$. Alternatively, if Y is $\chi^2(2)$, then $X = Y/(2\lambda)$ has a gamma distribution with $r = 1$ and mean $\mu = 1/\lambda$, i.e. an exponential(λ) distribution. Therefore, anything that can be simulated with an exponential distribution can also be simulated with a gamma or chi-squared distribution.

An exponential(λ) random variable is generated by the inverse CDF method, as explained at the beginning of Section 9.3. This method is recommended in Section 6.5.2 of Kennedy and Gentle (1980) for the gamma distribution with $r = 1$.

Gamma Distribution

The parameters of the gamma distribution are λ and r . The probability density function is given by

$$f(x) = \frac{\lambda^r}{\Gamma(r)} x^{r-1} e^{-\lambda x}$$

for $x > 0$, where $\Gamma(r)$ is the gamma function. In SAPHIRE, the value in the uncertainty distribution is r . The parameter λ is calculated from the mean value by the formula $\lambda = r/\mu$, since the mean is $\mu = r/\lambda$.

SAPHIRE generates a gamma random variable in two stages. First it generates a random variable Y from a gamma distribution with the desired r and with $\lambda = 1$. The algorithm used depends on the value of r .

if $r < 1$, SAPHIRE uses Algorithm GS of Ahrens and Dieter (1974),

if $r > 1$, SAPHIRE uses Algorithm GA of Cheng (1977),

if $r = 1$, SAPHIRE uses the inverse CDF method.

Once Y has been generated, the gamma random variable with parameter r and with the desired mean μ is defined as $X = Y/\lambda$, with $\lambda = r/\mu$.

For LHS sampling, SAPHIRE uses the fact that the gamma and the chi-squared distributions are different parameterizations of the same distribution. SAPHIRE uses the inverse CDF method to generate LHS samples from a gamma distribution.

Lognormal Distribution

X has a lognormal distribution if $\ln X$ has a normal distribution. The parameters used in SAPHIRE to describe the lognormal distribution are the mean of the lognormal distribution and the upper 95% error factor. The mean value of the lognormal distribution, μ_{\ln} , can be expressed as:

$$\mu_{\ln} = e^{\frac{\mu + \sigma^2}{2}}$$

where μ is the mean and σ is the standard deviation of the *underlying* normal distribution. Likewise, the 95% error factor (*EF*) for the lognormal distribution is given by:

$$EF = e^{1.645\sigma}$$

where 1.645 is the 95th percentile of the standard normal distribution. The density of the lognormal distribution is

$$f(x) = \frac{1}{x\sqrt{2\pi\sigma^2}} e^{-\frac{(\ln(x)-\mu)^2}{2\sigma^2}}$$

for $x > 0$.

In SAPHIRE, a random variable X is sampled from the lognormal distribution as follows. A random variable Y is generated from a normal distribution with mean μ and standard deviation σ . Then X is defined as $X = \exp(Y)$. This is the procedure for simple Monte Carlo sampling and for Latin Hypercube sampling.

Since SAPHIRE requires both the mean and EF, occasionally one will need to convert information into the required input parameters. For example, if a probability distribution is known, but the median and standard deviation is available, these two inputs would need to be

converted into the associated mean and EF. To assist in this conversion, we provide common translation equations for a set of lognormal parameters in Table 4.

Table 4. Conversion equations for various lognormal distribution parameters.

Known parameters				Equation
Mean (μ_{ln})	Standard deviation (σ_{ln})	Error factor (EF)	Median (\tilde{x}_{ln})	
✓	✓			$EF = e^{1.645\sqrt{\ln(1 + (\sigma_{ln}/\mu_{ln})^2)}}$
	✓		✓	$\mu_{ln} = e^{\frac{\ln(\tilde{x}_{ln})}{2} - \frac{1}{2}\ln(0.5 - 0.5\sqrt{1 - 4(\sigma_{ln}/\tilde{x}_{ln})^2})}$ $EF = e^{1.645\sqrt{\ln((\mu_{ln}/\tilde{x}_{ln})^2)}}$
		✓	✓	$\mu_{ln} = \tilde{x}_{ln} (EF)^{0.185 \ln(EF)}$
✓			✓	$EF = e^{1.645\sqrt{\ln((\mu_{ln}/\tilde{x}_{ln})^2)}}$

Maximum Entropy Distribution

The maximum entropy distribution is specified by its mean value, μ , and by a , the lower end of the range, and b , the upper end of the range. Section C1 of Appendix C shows that if μ is not at the midpoint of the range, the density has the form

$$f(x) = \beta e^{\beta x} / (e^{\beta b} - e^{\beta a})$$

for $a \leq x \leq b$, with β a non-zero parameter. If β is negative, the distribution is a truncated exponential distribution with parameter $-\beta$. If β is positive, the density is of exponential form, an increasing function of x . If instead μ is the midpoint of the range, $\mu = (a + b)/2$, then the maximum entropy density is not of the form above, but instead is flat, a uniform density. This is the limiting value for as $\beta \rightarrow 0$.

The density $f(x)$ corresponds to a CDF of the form

$$F(x) = (e^{\beta x} - e^{\beta a}) / (e^{\beta b} - e^{\beta a}) .$$

The parameter β and the user-input mean μ are related by

$$\mu = (be^{\beta b} - ae^{\beta a}) / (e^{\beta b} - e^{\beta a}) - 1/\beta , \text{ for } \beta \neq 0 .$$

Section C-1 of Appendix C shows that μ is a monotonic function of β . As $\beta \rightarrow -4$, μ approaches its minimum possible value, a , and as $\beta \rightarrow 4$, μ approaches its maximum possible value, b . This monotonicity is the basis for the algorithm given in Section C2 of Appendix C for finding β from the user inputs of μ , a , and b .

When the user specifies a maximum entropy distribution with mean μ and range $[a, b]$, SAPHIRE first uses a numerical search to obtain the value of β satisfying the user-input mean equation above. It then inverts the CDF explicitly and uses the inverse CDF method, by generating a random number U from a uniform distribution between 0 and 1, and calculating

$$X = \{ \ln[(e^{\beta b} - e^{\beta a})U + e^{\beta a}] \} / \beta ,$$

a random number from the maximum entropy distribution. This method is used both for simple Monte Carlo sampling and for LHS sampling.

Normal Distribution

The additional parameter to describe the normal distribution in SAPHIRE is the standard deviation of the distribution, Φ . The density function is given by

$$f(x) = \frac{1}{\sqrt{2\pi} \sigma} e^{-(x-\mu)^2/2\sigma^2}$$

where

$$-4 < x < +4.$$

SAPHIRE uses the Marsaglia-Bray algorithm, described on p. 203 of Kennedy and Gentle (1980), to generate a normal(0, 1) random variable Z . Then X , a normal random variable with mean μ and standard deviation Φ , is defined as $X = \mu + \Phi Z$.

For LHS sampling from a normal distribution, the inverse CDF method is used, with the inverse of the normal CDF $F^{-1}(U)$ computed as follows. For $0.1 \leq U \leq 0.9$, F^{-1} is found by the algorithm of Beasley and Springer (1977). For $U < 0.1$ or $U > 0.9$, F^{-1} is approximated by Algorithm 5.10.1 of Thisted (1988), due to Wichura. The approximation is then refined by one application of Equation (5.9.2) of Thisted.

Triangular Distribution

The additional parameters to describe the triangular distribution in SAPHIRE are the mode (most likely point on the curve) and the upper end point (b). The density function is given by

$$f(x) = \frac{2(x-a)}{(b-a)(\text{mode}-a)} \text{ for } a < x < \text{mode}$$
$$\frac{2(b-x)}{(b-a)(b-\text{mode})} \text{ for } \text{mode} < x < b$$

where a is the lower end point and b is the upper end point.

The mean is specified by the user, but in general can be found as $\mu = (1/3)(a + b + \text{mode})$. The triangular distribution is found in SAPHIRE 7.0 or higher. A triangular distribution can be viewed as a mixture distribution and, as such, the inverse CDF method can be employed to sample from it. The algorithm to sample the triangular distribution is from Butler (1970). For LHS sampling, SAPHIRE uses the inverse CDF method to generate LHS samples from a triangular distribution.

Uniform Distribution

The mean of this distribution is $M = (a+b)/2$. The value in the uncertainty distribution in SAPHIRE is b , the right (upper) endpoint of the distribution. The value for a is calculated by the equation $a = 2*M - b$. The density function for this distribution is

$$f(x) = \frac{1}{b-a}$$

for $a \leq x \leq b$.

SAPHIRE generates a uniformly distributed random number using the prime modulus multiplicative linear congruential generator approach. The modulus m for the generator is $2^{31}-1 = 2,147,483,647$ and the multiplier is 397,204,094. (Fishman and Moore, 1982) This generates a sequence of $m - 1$ distinct integers before repeating, in an order that appears random. To obtain real numbers between 0 and 1, the integer obtained in this way is divided by m .

Having generated a random variable Y uniform between 0 and 1, SAPHIRE obtains a random number uniform between a and b as $X = a + (b-a)Y$. This is used for both simple Monte Carlo sampling and for LHS sampling.

9.5 Histograms

SAPHIRE allows for either a discrete or a continuous distribution under this option. The modeled quantity is a probability or a frequency. When the PERCENT option is selected, the distribution is discrete on up to 100 values; the percents, giving the degree of belief for each value, must sum to 100. If the RANGE or AREA option is selected, the density is a step function covering up to 100 adjacent intervals. The function is constant within each interval, and the area under the entire function must equal 1.0.

The histogram feature allows the user to approximate analytical distributions types not currently available in SAPHIRE or other distributions not fitting a common analytical form (e.g., multi-modal). For a continuous random variable, the histogram is an approximation to the probability density function in which area is equivalent to frequency, thus maintaining the continuous characteristic of the variable. In other words, the bins of a histogram have an area (normalized to 1.0) representing the proportion of the density function lying in the interval given by the bin. An example of a "range type" histogram is shown in Figure 15. With this option, the starting probability is 0.0036 (and height of 15.38) while its end probability is 0.036. The second bin has an end probability of 0.36 (and height of 1.538). The area of each bin is 0.5. When this distribution is used, SAPHIRE will randomly sample values from the histogram bars (for example, it is equally likely, with overall probability of 0.5, to have a probability value between 0.0036 and 0.036).

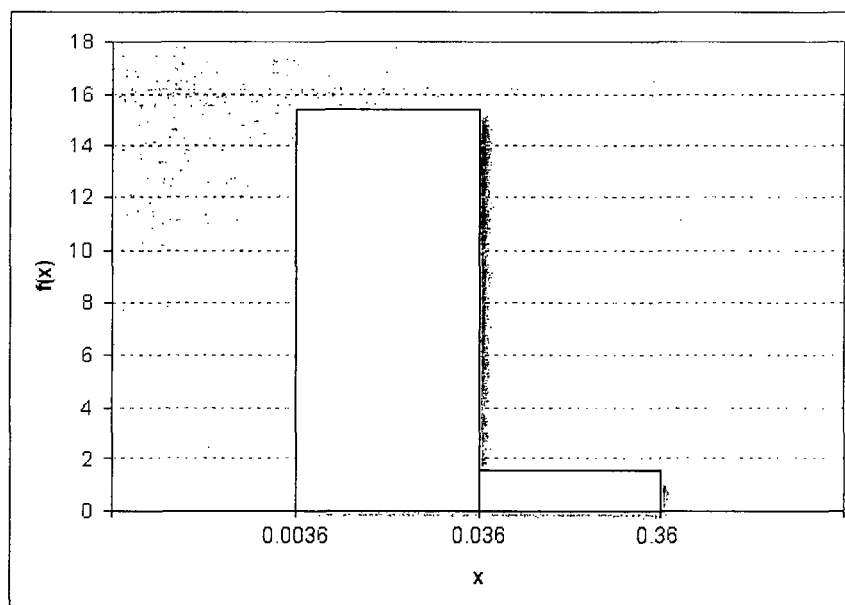


Figure 15. Example of range histogram

A second histogram type, "area," is closely related to the "range" type. The area histogram type is like the range type in that it represents a bin (instead of a discrete point) and the

starting and ending probability values are required. But, unlike the range histogram, the area histogram asks the user to specify the area of the bin directly rather than the magnitude of the bin. Thus, in our case, the area of each bin is 0.5. Consequently, the two histograms, the range and area, will be equivalent.

The last histogram type is a discrete version of the bin-type histogram. The "percentage" histogram requires a discrete point and the corresponding percent associated with that point (in %). The cumulative percent for all the points must total 100%.

9.5 Correlation Classes

The practice of using the same uncertainty distribution for a group of similar components has been common since the Reactor Safety Study (NRC 1975). The PRA Procedures Guide (Hickman 1983) recommends this practice as well. Philosophical arguments have been given to support this practice or used to give it credence. Apostolakis and Kaplan (1981) discuss this issue from a Bayesian perspective, and they call it a "lack of knowledge" dependency. This dependency is present whenever the same data set is used for several components. It is not a Bayesian or classical statistical phenomenon, but it is induced because of the way the data are used.

For example, suppose that a plant has two motor-driven AFW pumps. These pumps are virtually identical, and therefore are modeled as having the same unavailability, q . The uncertainty distribution for q is taken from some database, and describes our best belief about the true value of q . Because the two components have uncertainty distributions taken from the same source, if our estimate of q is too high (say) for one pump, it will be also be too high for the other pump, by the same amount. Similarly, if our estimate is too low for one, it will be too low for the other by the same amount. The uncertainty distributions for the two unavailabilities are perfectly correlated.

This correlation of the uncertainties must be distinguished from the independence of the basic events. The two basic events (failures of the pumps to be available) are independent; that is, the probability that one pump is unavailable is some number q , unaffected by whether the other pump is available or not. However, our uncertainty about the value of q is totally correlated for the two basic events.

The user tells SAPHIRE of this uncertainty correlation by putting the two basic events in a single *correlation class*. When q is sampled from its uncertainty distribution, that same value of q is assigned to all the basic events in the correlation class. After the probability of the top event has been calculated, on the next Monte Carlo pass a new (presumably different) value of q is drawn from the uncertainty distribution, and is assigned to all the basic events in the class.

Let us now examine the effect of total correlation in accident sequence analysis. Consider a simple example involving a cut set with two components. Let q_1 and q_2 denote the unavailability of the two components in the cut set. If the components are independent, then

$$Q = q_1 q_2 \quad (9-1)$$

is the cut set unavailability.

As we begin the analysis, we can make one of two assumptions. First, we can assume that the unavailability of each component is estimated from independent data sources. For example, if the first basic event is failure of a pump and the second basic event is failure of a valve, the probabilities of these basic events will be estimated from independent sources, and therefore the two probabilities have independent uncertainty distributions. The expected value and variance of Q are given by

$$E(Q) = E(q_1)E(q_2) \quad (9-2)$$

and

$$\text{var}(Q) = E(q_1^2)E(q_2^2) - [E(q_1)E(q_2)]^2 \quad (9-3)$$

These equations follow from the independence of the uncertainty distributions.

If instead, the components are identical, then $q_1 = q_2 = q$, and Equations (9-2) and (9-3) reduce to

$$E(Q) = E(q_1)E(q_2) = [E(q)]^2 \quad (9-4)$$

and

$$\text{var}(Q) = [E(q^2)]^2 - [E(q)]^4 \quad (9-5)$$

However, when the components are identical, Equations (9-4) and (9-5) are probably not correct. The same source would presumably be used to obtain the uncertainty distribution for both unavailabilities. Therefore, any value q that is used for one basic event should also be used for the others. Equation (9-1) reduces to

$$Q = q^2 \quad .$$

so the expected value is

$$E(Q) = E(q^2) \quad (9-6)$$

and

$$\text{var}(Q) = E(q^4) - [E(q^2)]^2 \quad (9-7)$$

A standard identity from statistics says that

$$E(q^2) = [E(q)]^2 + \text{var}(q) > [E(q)]^2 .$$

Therefore, Equation (9-6), the correct one, is larger than Equation (9-4), the incorrect one. This is why the point estimate and the mean of the uncertainty distribution are not equal in PRAs. The point estimate for the example cut set is the product of the basic event means, given by Equation (9-4), whereas the mean of the cut set uncertainty distribution is given by the larger value in Equation (9-6). Similarly, the variance should be calculated from Equation (9-7), not Equation (9-5). In typical cases, including any case in which q is lognormally distributed, Equation (9-7) gives a larger value than Equation (9-5). The effects are most pronounced when the distributions are highly skewed.

It is suggested to use the following steps for grouping basic events into correlation classes:

- Group all basic events by component type (e.g., MOV, AOV, MDP),
- Within each component group, organize events into subgroups by failure mode (e.g., fail-to-start, fail-to-run),
- For time related basic events, group all events from each component failure mode group into sets according to the time parameter value used to quantify the event probability (e.g., 6 hours, 720 hours), and
- For demand related failures, no further grouping is necessary beyond the component failure model level.

If different estimates are developed for components within the same component group (e.g., service water motor-driven pump, residual heat removal motor-driven pump) then these components should be treated in separate component groups.

9.6 Sampling Techniques

The Monte Carlo approach is the most fundamental approach to uncertainty analysis. Simple Monte Carlo simulation consists of making repeated quantifications of the top event value using values selected at random from the uncertainty distributions of the basic events. For each iteration of the Monte Carlo run, each basic event uncertainty distribution is sampled using a random number generator to select the failure probability of the basic event. The top event probability or accident sequence frequency is calculated. When this procedure has been repeated a predetermined number of times, the top event or accident sequence results

are sorted to obtain empirical estimates of the desired top event attributes such as the mean, median, 5th percentile, and 95th percentile. A plot of the empirical uncertainty distribution is often obtained. Figure 16 contains an example of an uncertainty distribution for an accident sequence. For more information about the Monte Carlo technique the reader is referred to Hahn and Shapiro (1967).

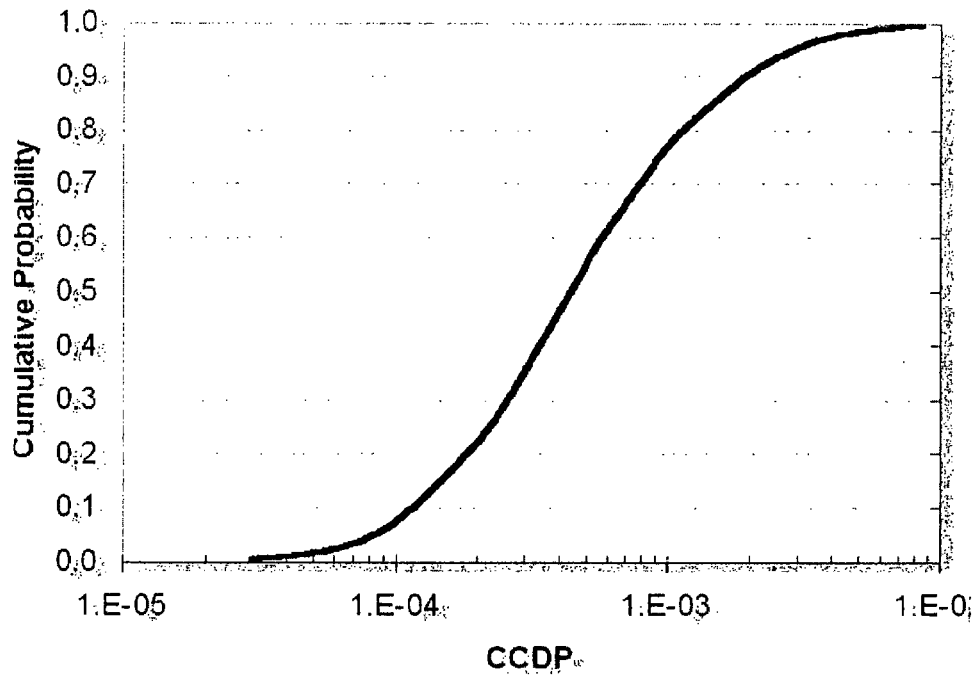


Figure 16. Illustration of the uncertainty on accident sequence probability results

To illustrate the Monte Carlo technique, consider a system with two components in series. Let A denote failure of the first component and B failure of the second. The cut sets for the system are A and B , so the equation for the top event (system) is

$$S = A \cup B$$

Let A and B have mean failure probabilities of 0.001 and 0.005, respectively. Also assume that the uncertainty distribution for A is uniform from 0 to 0.002 and the distribution for B is normal with standard deviation of 0.001. These distributions are shown in Figure 17 and Figure 18, respectively.

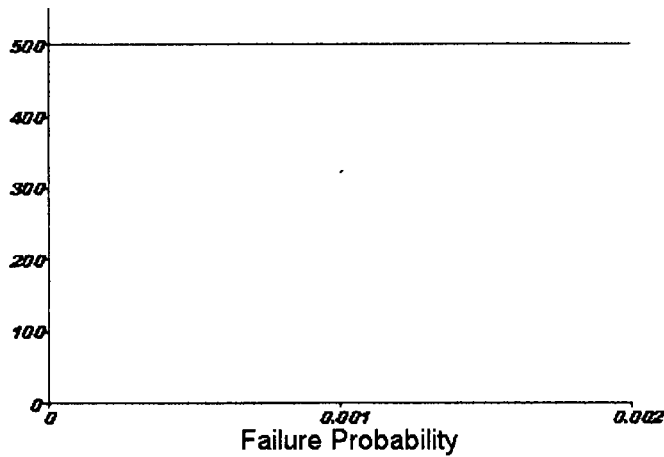


Figure 17. Uncertainty distribution for Component A

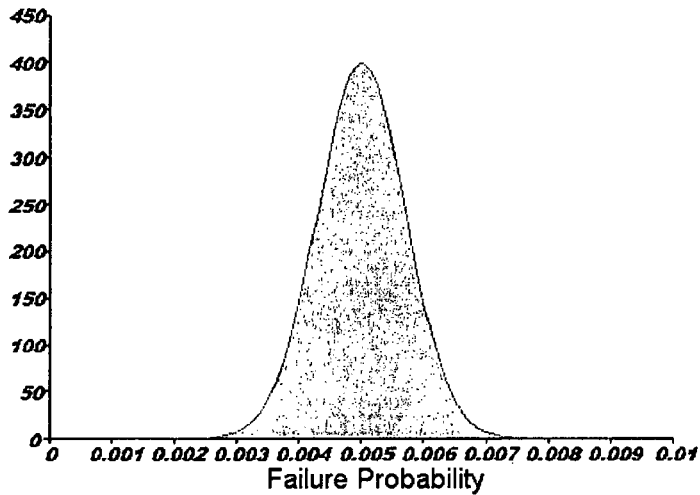


Figure 18. Uncertainty distribution for Component B

The point estimate for S is 0.006. Table 5 contains a random sample of size 10 for this example. Within this table, column 1 contains the sample for component A which has a uniform uncertainty distribution. Column 2 contains the sample for failure of component B, and column 3 contains the sum of columns 1 and 2 which is the minimal cut set upper bound for the probability of failure of the system. The bottom row is the average (mean) of the columns.

Table 5. Random samples for the S system example.

Iteration	A	B	$A \cup B$
1	0.00042	0.00500	0.00542
2	0.00086	0.00661	0.00747
3	0.00149	0.00570	0.00719
4	0.00109	0.00605	0.00714
5	0.00066	0.00420	0.00487
6	0.00024	0.00609	0.00633
7	0.00066	0.00396	0.00462
8	0.00075	0.00293	0.00368
9	0.00037	0.00500	0.00537
10	0.00127	0.00597	0.00724
<i>mean</i>	<i>0.00078</i>	<i>0.00515</i>	<i>0.00593</i>

Overview of Latin Hypercube Sampling

Latin Hypercube Sampling (LHS) selects n different values from each of the k variables X_1, \dots, X_k in the following manner. The range of each variable is divided into n nonoverlapping intervals on the basis of equal probabilities for the intervals. The n values thus obtained for X_1 are paired in a random manner with the n values of X_2 . These n pairs are combined in a random manner with the n values of X_3 to form n triplets, and so on, until n k -tuplets are formed. This is the Latin Hypercube sample. It is convenient to think of the LHS, or a random sample of size n , as forming an $n \times k$ matrix of inputs where the i th row contains specific values for each of the k input variables to be used on the i th evaluation of the cut sets.

To help clarify how intervals are determined in the LHS, consider the simple example used in the previous section. We want to generate an LHS sample of size 5. The first step is to divide the uncertainty distributions of A and B into 5 equal probability areas each containing an area of 0.2. For A this is easy since it has a uniform uncertainty distribution. The points separating the cells are 0.0004, 0.0008, 0.0012, and 0.0016. The areas are shown in Figure 19. The uncertainty distribution for B is a normal distribution; it is harder to find the points that divide the areas into equal probability areas. Probability tables or a calculator with an inverse normal calculation routine is needed. The four points which define the 5 equal probability areas are 4.16E-3, 4.75E-3, 5.25E-3, and 5.84E-3. These are shown in Figure 20.

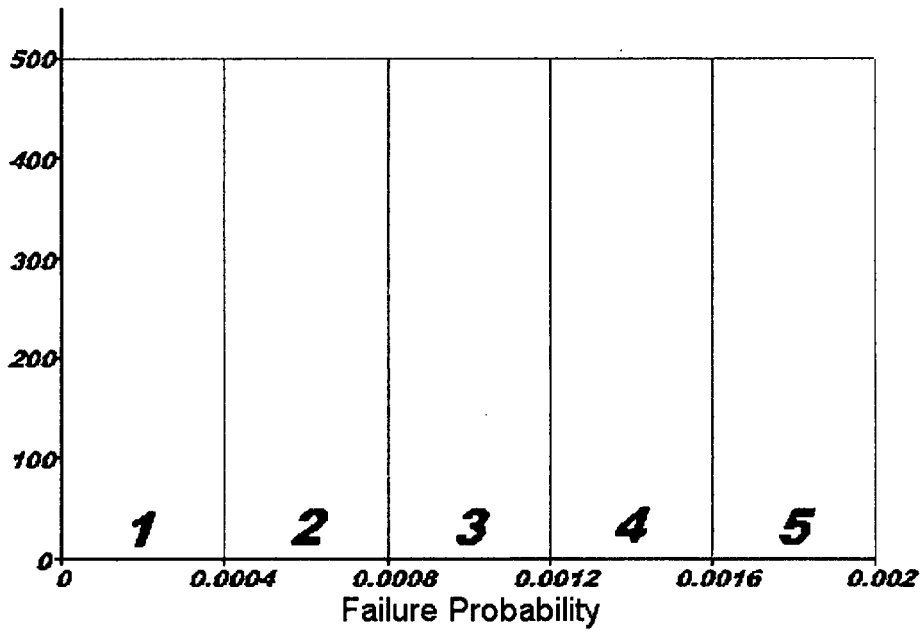


Figure 19. Latin hypercube sample for Component A

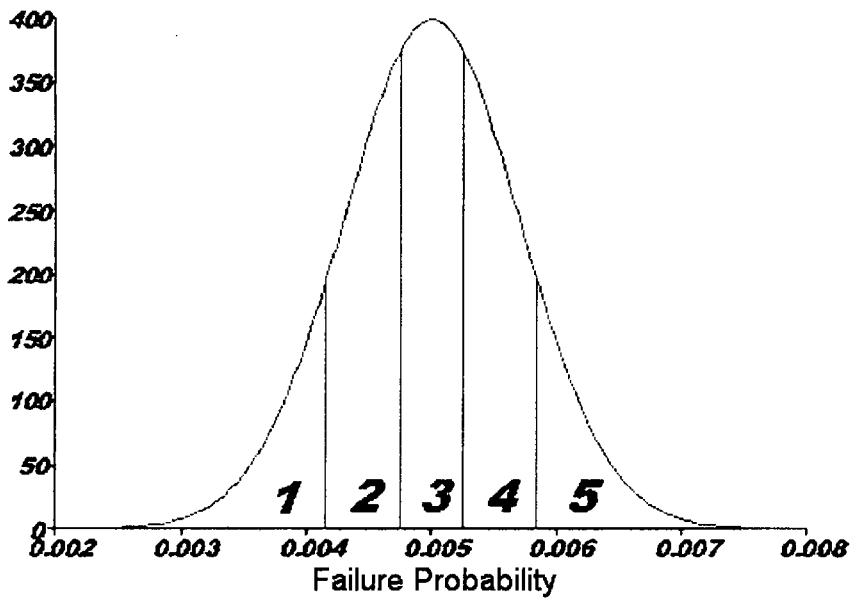


Figure 20. Latin hypercube sample for Component B

The next step is to generate a random permutation of the integers (1, 2, 3, 4, 5). In this example we get (2, 3, 4, 1, 5). Now we choose a random offset for each basic event, 2 for A and 3 for B. We start the basic permutation for A at the second element, yielding (3, 4, 1, 5, 2), and we start the basic permutation for B at the third element, yielding (4, 1, 5, 2, 3). Combining these values yields Table 6.

Table 6. Randomly selected intervals for the LHS example cells.

Computer Run	Interval for A	Interval for B
1	3	4
2	4	1
3	1	5
4	5	2
5	2	3

These five cells are shown in Figure 21. The next step is to obtain random values for A and B for each of the intervals. The first value for A lies in interval 3; thus, the value must be between 0.0008 and 0.0012. To select a value in the cell, a random number U is generated from a uniform distribution between 0 and 1. Then A is defined as $0.0008 + 0.0004U$. The corresponding value for B lies in interval 4; thus the value for B must lie between the 60th and 80th percentiles of the normal distribution. A new random number U is generated from a uniform distribution between 0 and 1, and $V = 0.6 + 0.2U$ is therefore uniform between 0.6 and 0.8. Let F denote the standard normal CDF. Then $Y = F^{-1}(V)$ is sampled from between the 60th and 80th percentiles of the standard normal CDF. Finally $B = 0.005 + 0.001Y$ is sampled from between the 60th and 80th percentiles of a normal distribution with mean 0.005 and standard deviation 0.001. Table 7 summarizes the random numbers in this case.

Table 7. Randomly selected values for A and B using the LHS example.

Computer Run	Value for A	Value for B	Value for A∪B
1	9.454E-4	5.398E-3	6.343E-3
2	1.512E-4	3.862E-3	5.374E-3
3	6.102E-5	6.684E-3	6.745E-3
4	1.827E-3	4.504E-3	6.331E-3
5	7.068E-4	4.898E-3	5.605E-3
Mean	1.010E-3	5.069E-3	6.080E-3

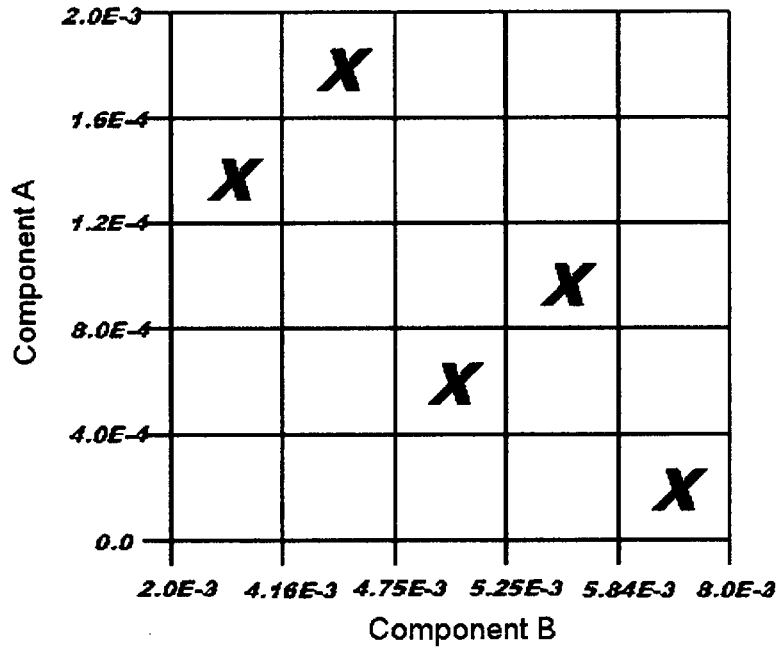


Figure 21. Cells sampled in the LHS example

This method of generating a permutation for each basic event, using a random offset of a single permutation, is less general than choosing a random permutation for each basic event. It is used in SAPHIRE to save storage and, secondarily, to save execution time.

Comparison of Simple Monte Carlo and Latin Hypercube Sampling

The following information is a comparison of Simple Monte Carlo simulation and LHS. Table 8 contains output from SAPHIRE for the sample problem in the previous section and also the exact values, which should be obtained if the sample sizes were infinite. Figure 22 contains a plot of the cumulative distribution function for each sample. The results are very similar for these two methods. Notice the size of the samples for each. The LHS method requires only a quarter of the sample size of ordinary Monte Carlo, for similar accuracy. This must be balanced against the fact that for some distributions it takes longer to generate a random number for an LHS sample than for a simple Monte Carlo sample. Nevertheless, LHS sampling can often substantially reduce the time required for an analysis, while obtaining similar accuracy.

Table 8. Comparison of Monte Carlo and LHS for sample problem.

	Monte Carlo	LHS	Exact
Random Seed	51530	27290	-
Sample Size	200	50	-
Point estimate	5.995E-003	5.995E-003	6.00E-003
Mean Value	6.008E-003	5.994E-003	6.00E-003
5th Percentile Value	3.890E-003	3.876E-003	4.10E-003
Median Value	6.103E-003	6.320E-003	6.00E-003
95th Percentile Value	7.783E-003	7.816E-003	7.90E-003
Minimum Sample Value	2.798E-003	2.789E-003	-
Maximum Sample Value	8.944E-003	8.605E-003	-
Standard Deviation	1.163E-003	1.245E-003	1.16E-003
Skewness	-1.973E-001	-3.071E-001	0.000
Kurtosis	2.860E+000	2.747E+000	3.09

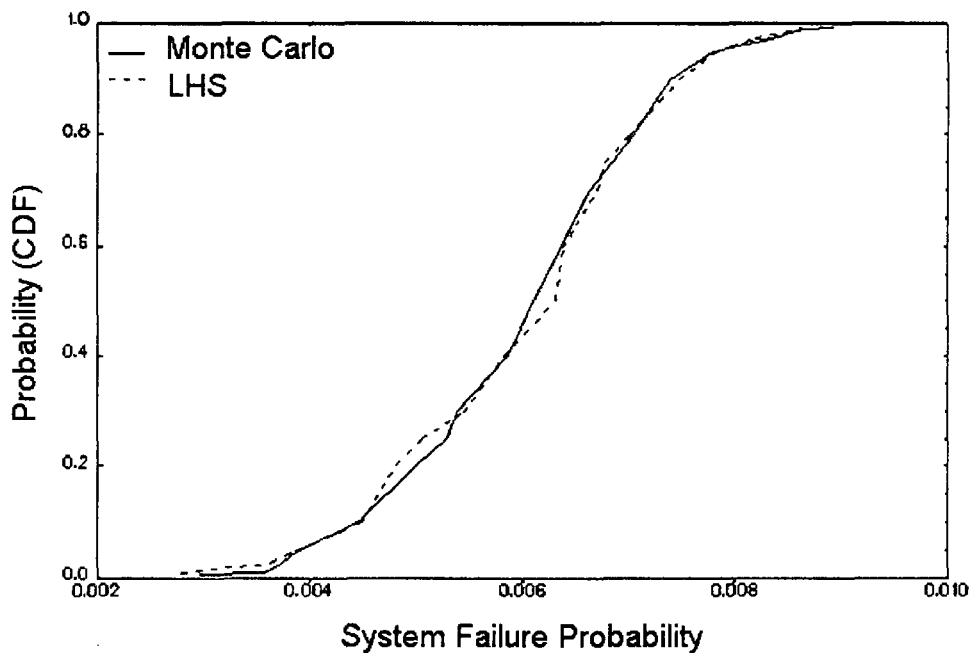


Figure 22. Cumulative distribution plots for example using Monte Carlo and LHS

9.7 Inverse C.D.F Method

One method for generating random numbers, called the *inverse CDF method*, is used for several distributions in SAPHIRE. Let X denote a random variable, let x denote a number, and let F denote the cumulative distribution function (CDF) of X . It follows directly from the definition

$$F(x) = P(X \leq x)$$

that $F(X)$ is a uniformly distributed random variable between 0 and 1. Therefore, generate U from a uniform distribution between 0 and 1, and solve $F(X) = U$ for $X = F^{-1}(U)$.

For example, if X is exponentially distributed with mean μ , the CDF is

$$F(x) = 1 - e^{-x/\mu}$$

To generate an exponentially distributed random variable X , first generate a uniformly distributed random variable U and let $X = F^{-1}(U) = \ln(1-U)$. Actually $\ln(U)$ can be used instead of $\ln(1-U)$, because if U is uniformly distributed between 0 and 1, then so is $1-U$.

The inverse CDF method is only one of many methods of generating random numbers from a specified distribution. For some distributions it is natural and fast, and for other distributions a different method may be quicker. If the inverse CDF is hard to compute, for example if it must be found at any point by numerical iteration on the (non-inverse) CDF, then the inverse CDF method is not a fast way to generate random numbers.

There is one application where the inverse CDF method is very natural. This is in Latin Hypercube Sampling (LHS), where stratified portions of the distribution must be sampled. For example, if 20 points are to be sampled, one point must be below the 5th percentile, one must be between the 5th and the 10th percentiles, one between the 10th and 15th, and so forth. It is easy to sample in this way from a uniform distribution: For example, to sample a uniform (0, 1) distribution between its 10th and 15th percentiles, we must sample it and obtain a number between 0.10 and 0.15. Do this by letting U be uniform between 0 and 1. Then let Y equal $0.10 + 0.05U$, which is between 0.10 and 0.15. Then use the relationship $X = F^{-1}(Y)$ where X is between the 10th and 15th percentiles of F , as required. For this reason, all Latin Hypercube samples are generated in SAPHIRE using the inverse CDF method.

Whenever a distribution is specified, for either the frequency of an initiating event or the probability of a basic event, SAPHIRE performs some preliminary checks, to help the user avoid entering illegal or unwise values. Further, the parameters of the distribution are always checked for legality. For example, the mean must be nonnegative and the other parameter(s) must be nonnegative or positive, for all of the distributions used in SAPHIRE when representing probabilities. When a violation is found, SAPHIRE prints a message on the screen stating what is wrong. When a parameter is legal but degenerate, such as a zero

standard deviation or a zero range, SAPHIRE prints a warning message, but allows the value to be used.

The term *parameter* refers not to the numbers that define a distribution (such as a mean and variance) but to the quantity that has the uncertainty distribution, namely, the frequency of an initiating event or the probability of a basic event. SAPHIRE looks at the probability that the uncertain parameter is outside of its allowed range. That is, the probability of a basic event must lie between 0 and 1, and the frequency of an initiating event must be greater than 0. These are the allowed ranges of the uncertain parameters. If, for example, the parameter is assigned a normal distribution, it is possible for the parameter to take a negative value, which is not allowed for probabilities. If the parameter is a basic event probability, it must not exceed 1.0, but the normal distribution (and some others) allows this with positive probability. SAPHIRE prints a warning message on the screen if the probability of exceeding the maximum legal value is greater than 5E-4, and a second warning message if the probability of being less than the minimum legal value is greater than 5E-4. The user may then change the distribution.

Later, if an illegal value is generated in the course of Monte Carlo simulation, the value is discarded and a new value is generated. If LHS is performed instead of simple Monte Carlo simulation, the range is restricted to the legal portion of the distribution. Thus, in either case the distribution is truncated to its legal portion. When the truncated portion has a very small probability, the effect of the truncation is negligible. When a substantial fraction of original distribution is illegal, however, the effect of the truncation is to change the distribution, including its mean, from what the user specified. Therefore, it is unwise to use a distribution that puts substantial probability outside the allowed range of the parameter.

Whenever a distribution is specified, SAPHIRE compares it to the constrained noninformative distribution having the same mean. As described earlier, the constrained noninformative prior is a diffuse distribution with the specified mean, and with the amount of diffuseness corresponding to ignorance of everything except the mean. If the user specifies a distribution having a variance larger than that of the constrained noninformative distribution, SAPHIRE prints a warning message on the screen. The warning states that the specified distribution is very diffuse, even more diffuse than a model of ignorance. The user has put more weight in the tails of the distribution than the constrained noninformative distribution does, and so the user may be overly pessimistic.

There can be valid reasons for choosing a distribution that is more diffuse than the constrained noninformative distribution. For example, the mean, treated as known for the constrained noninformative distribution, may not be known very well, and the user may desire to model this uncertainty by adding some extra spread to the distribution. Or the user may be duplicating portions of published earlier work, where a highly diffuse distribution was used. If, however, the user simply picked a distribution with a large variance, in an attempt to model great uncertainty, it may be preferable to use the constrained noninformative distribution or one with a comparable variance.

10. SEISMIC EVENTS

10.1 Fragility and Component Failure Probabilities

The severity of an earthquake is measured by its peak ground acceleration, denoted g . The *fragility distribution* of a component is the probability that the component fails, expressed as a function of g . The *fragility curve* is a graph of this function, plotting the probability of failure against the peak ground acceleration. For simplicity, the fragility is assumed to be based on a lognormal distribution. The approach is based on Kennedy et al. (1980).

For a particular component, let A_f denote the "failure acceleration," the peak ground acceleration that is just sufficient to cause a component to fail. It is random, because sometimes a component will fail when subjected to a certain acceleration and sometimes an apparently identical component will survive when subjected to the same acceleration. The randomness is a really a property of the components. A_f is random in the frequentist sense; that is, it is the random outcome of a hypothetical repeatable experiment in which we are assumed able to measure the acceleration that is just enough to cause the particular component to fail. This distribution is another way of expressing the fragility distribution, with large A_f corresponding to small fragility.

Assume for the moment that there is no uncertainty, only randomness as described above. Uncertainty will be modeled later. Model the failure acceleration as a lognormal random variable:

$$A_f = \alpha \varepsilon_R \quad (10-1)$$

Here ε_R is a lognormal random variable such that $\ln(\varepsilon_R)/\beta_R$ is normal(0,1), and α is the median failure acceleration. For now, α is treated as being perfectly known. The quantity β_R is a parameter of the model. The subscript R stands for "random," and is a reminder that this probability distribution refers to random differences between nominally like components, not to subjective uncertainty. Another way to write (10-1) and the explanatory text is

$$\frac{\ln(A_f / \alpha)}{\beta_R} \approx \text{normal}(0,1) \quad (10-2)$$

Consider some specific acceleration g . The probability of a component's failure, given that the component is subjected to this g , is:

$$\begin{aligned} P[g \geq A_f] &= P[A_f \leq g] \\ &= P[\ln(A_f / \alpha) / \beta_R \leq \ln(g / \alpha) / \beta_R] \\ &= \Phi[\ln(g / \alpha) / \beta_R] \end{aligned} \quad (10-3)$$

where Φ is the standard normal cumulative distribution function, and the last step follows from Equation (10-2). Equation (10-3) gives the fragility curve for the component, relating the assumed peak ground acceleration g to the probability that the component fails.

For a PRA without an uncertainty analysis, Equation (10-3) is used for the basic event failure probability, conditional on the acceleration g . The frequency of g is treated like the frequency of an initiating event.

Now consider uncertainty. Let a be the best estimate of the uncertain α . This is the quantity entered by the user in the field for median failure acceleration. Model the uncertainty of α in the usual Bayesian way by treating α as a random variable with median a :

$$\alpha = a \varepsilon_U$$

where ε_U is a lognormal random variable such that $\ln(\varepsilon_U)/\beta_U$ is normal(0,1). This can be rewritten as

$$\frac{\ln(\alpha / a)}{\beta_R} \approx normal(0,1)$$

Note, the meaning of "best estimate" is different from the use in the non-seismic portions of SAPHIRE. Here, the best estimate is the median of the uncertainty distribution, whereas in the non-seismic analysis the best estimate is the mean of the uncertainty distribution. There are two reasons for this. (1) It follows Kennedy et al. (1980), and so is consistent with customary use by seismic analysts. (2) It is more conservative than using the mean. The mean is larger than the median, and corresponds to a larger failure acceleration, hence a smaller fragility. Use of the mean failure acceleration as a nominal value would result in a larger nominal failure acceleration, and a smaller nominal fragility. This would lead to more severe truncation of cut sets, and also a smaller nominal top event probability. This is the reverse of the usual, non-seismic, case. There, the failure probability, not a failure acceleration, has an uncertainty distribution, and use of the mean instead of the median results in a larger nominal failure probability, less severe truncation of cut sets, and a larger nominal top event probability.

In summary, the user inputs a , β_R , and β_U , where a estimates α . Here α is the median failure acceleration (the median of the distribution whose dispersion is determined by β_R), and a is the median of the distribution quantifying uncertainty about α , the distribution whose dispersion is determined by β_U . The uses by SAPHIRE of the input quantities are discussed next.

For many purposes, the nominal basic event probabilities are used. These include truncating cut sets, calculating the nominal top event probability, and calculating basic event importances. Conditional on a peak ground acceleration g , the nominal probability of any basic event is given by Equation (10-3) with a substituted for α .

When performing an uncertainty analysis in a PRA, it is customary to quantify the uncertainties in the failure probabilities by Monte Carlo simulation. This approach is followed in SAPHIRE.

First, α is generated by Monte Carlo sampling, by letting z be generated by a standard normal random number generator, and letting

$$\alpha = a \exp(\beta_U z) . \quad (10-4)$$

For a given g , the failure probability for a component is given by substituting this value of α into Equation (10-3).

A quantity of some interest is denoted HCLPF (pronounced hick-clip or hick-cliff), the maximum acceleration that corresponds to High Confidence of Low Probability of Failure. The quantity is calculated as

$$\text{HCLPF} = a \exp[-1.645(\beta_R + \beta_U)] ,$$

which is derived as follows. HCLPF requires both high confidence and low probability. The words high and low are taken to mean 0.95 and 0.05, respectively. Consider first the low-probability portion: In terms of the true, but unknown, median failure acceleration, α , the probability of a component's failure is given by Equation (10-3). The failure probability is low, that is, $p \leq 0.05$, if $\ln(g/\alpha)/\beta_R$ at most the 5th percentile of the standard normal distribution. This says that

$$\ln(g/\alpha)/\beta_R \leq -1.645 ,$$

from which we obtain

$$g \leq \alpha \exp(-1.645\beta_R) . \quad (10-5)$$

That is the low-probability portion of the definition. For the high-confidence portion, recall that α is not known, but has an uncertainty quantified by Equation (10-4). Therefore, with high (95%) confidence we have

$$\ln(\alpha/a)/\beta_U \geq -1.645$$

or, equivalently,

$$a \exp(-1.645\beta_U) \leq \alpha .$$

Substitute this bound on α into Equation (10-5). It follows that we have high confidence that Equation (10-5) is true if

$$g \leq a \exp(-1.645\beta_U) \exp(-1.645\beta_R) / a \exp[-1.645(\beta_R + \beta_U)] .$$

The largest such g is the expression on the right, the HCLPF value.

10.2 Frequencies of Seismic Events

For cut set truncation, the user enters a single peak ground acceleration, g . Equation (10-3) is used with this g and the best estimate of α to determine the nominal basic event probabilities. Cut sets are truncated based on these probabilities. It is a mistake to enter a small value of g , because too many cut sets will be truncated.

For all other portions of the analysis, the user may enter up to 20 peak ground accelerations g_i that could be produced by an earthquake, and up to 20 corresponding frequencies, with units 1/yr. This assumes that the user knows the frequencies. To account for uncertainty in the frequencies in a simple way, three sets of accelerations and frequencies can be entered, low, medium, and high. The analysis can be based on the medium (best estimate) set, or on the low set (mild earthquake assumptions) or the high set (severe earthquake assumptions).

Recall that non-seismic analyses have initiating events, with event frequencies. Examples are loss of off-site power and a large LOCA. A seismic analysis differs, because each postulated g_i is an initiator with its corresponding frequency. The events such as loss of off-site power and large LOCA are consequences of the earthquake. We will call them "induced initiators," but mathematically, they are events just like basic events. They have probabilities, not frequencies, and these probabilities are conditional on the value of g_i . Similarly, the basic events have probabilities that depend on g_i , found using Equation (10-3) above. In an accident sequence, the frequency of g_i is multiplied by the probability of the induced initiator and by the probability of failure of each other component or system in the sequence, to yield the frequency of the sequence, with units 1/yr. The frequency of a plant state is the sum of the frequencies of the sequences (with various levels of g_i) corresponding to that plant state, plus the frequency of the plant state found by a non-seismic analysis assuming no earthquake.

After setting up the initiators and the events as described above, SAPHIRE performs its seismic calculations in the same way that it performs calculations for non-seismic sequences.

10.3 SAPHIRE Seismic Implementation

The following discusses the features and use of the seismic module in SAPHIRE. This is intended to introduce users who are already familiar with seismic and PRA analyses to the SAPHIRE seismic module interface, including how to input data and perform seismic calculations. This is not intended to teach users how to do seismic-PRA.

The discussion assumes the availability of internal-events PRA. Specifically, random-failure composed system-models, accident sequence progression, and initiating events have all been defined and developed for the engineered system of interest (e.g., nuclear power plant). The seismic analysis is being factored into that engineered system, which is already well understood and comprehensively modeled. Therefore, functional vulnerabilities have been identified and the seismic analysis consists of converting to and adding in the seismic-induced failures.

SAPHIRE provides the flexibility to construct seismic risk analysis models by either (or a combination) of two methods. First, seismic-specific event tree and fault tree models can be developed via the graphical interface. Second, SAPHIRE contains a provision for performing transformations in the form of Boolean identities (i.e., $A=A+B$, $A=B$, or $A=A*B$). This allows the user to build on an internal events analysis when developing a seismic model. More specifically, after site-specific seismic vulnerabilities have been identified (through plant walk-downs or some other site-specific review), they can be incorporated into an existing internal events analysis using a set of basic-event transformations that either replace or add the seismic failure events to the existing basic events.

10.3.1 The Hazard Curve

The hazard curve is the representation of the range of possible earthquakes. It is commonly found in the form of a probability of exceedence curve, with the earthquake ground acceleration on the horizontal and the probability of exceeding that acceleration on the vertical axes. (One source for this information is NUREG-1488.) However, SAPHIRE utilizes this information in the form of a histogram (or more precisely, a discrete probability density distribution). Specifically, the density needs to be arranged into a maximum of 20 ground acceleration bins with each one assigned a yearly frequency of occurrence.

To input this information into SAPHIRE, the user selects "Modify " from the SAPHIRE menu. Selecting "Histograms" from the sub-menu brings up a dialog box containing the list histograms. Click the right mouse button for the pop-up menu containing options to be executed. Selecting "Add" allows the user to actually create the histogram. Each bin (numbered 1-20) is associated with an event name (e.g., HAZARDEVENT01), which is how that specific earthquake event (magnitude and frequency) is identified in the analysis. Note that these event names are generic. That is, only a single set of event names, corresponding to the 20 bins of a single histogram, are defined. The data associated with a particular event name is taken from the histogram that is selected for use with a particular family. Which histogram will be used in the analysis is identified by selecting the Project → Modify option.

10.3.2 Modeling Seismic Event and Fault Trees

The most straightforward approach (at least with respect to using SAPHIRE) for creating a seismic analysis model involves the development of a seismic event tree that prioritizes and links the seismic-induced internal events initiators with the earthquake (the true initiating event). This single seismic event tree begins with a generic seismic-initiating event set to a value of 1.0. [The actual magnitude (g- level) and frequency of the earthquake of interest are identified by the user and factored into the analysis when the cut sets are generated and quantified.] The event tree top- events are those internal events initiators that have the potential to be induced by an earthquake. They are listed in order of severity (in terms of challenging plant safety systems), with the more severe induced-initiators listed first. This addresses the potential pitfall of over-counting core damage sequences (i.e., a single earthquake inducing both a large LOCA and a small LOCA at the same time). However, these event tree top- events are treated as seismic basic- events (or fault trees) with associated

seismic fragility data. The resulting end states are therefore the frequencies of seismically induced challenges (i.e., internal- events type initiators) to the plant. These in turn can be identified as transfers to the systems analysis (internal-events accident sequences) event trees. This linking will automatically replace the internal events analysis initiator on the systems analysis event tree with the transferred information from the seismic event tree. This is how the linking between the earthquake, induced initiating event, and the system models are made.

The actual system models are commonly fault trees. As mentioned above, the seismic system models can be created as independent, stand-alone fault trees. However, using SAPHIRE, they can also be integrated with the internal events analysis.

SAPHIRE allows seismic failures to be identified with a specific Model Type. A basic event will be able to “cloned” so that one failure model is used for one model type (e.g., internal events) while a different failure model (including failure parameters) is used in a different model type (e.g., seismic). SAPHIRE will automatically make these cloned events in the database at the time the model type information is assigned to the basic event. When editing a basic event, the dialog will have a “Applicability” option via the event attributes link (the other information for the attributes is to be determined). The model type applicability attributes will define the event’s specific instance to be used for each model type. The available model types are user defined and are available from the Project → Model Type option.

10.3.3 Basic Event Data - Fragilities and Uncertainty Data

Basic events are defined in the basic event database. Seismic failure data is usually characterized by a median fragility and two uncertainty terms representing the random uncertainty and the confidence uncertainty (β_R and β_U , respectively). There is also an added factor that might or might not be included in seismic failure data called the structural response factor (SRF). The SRF quantifies the amount of amplification or dampening of ground motion a particular piece of equipment experiences during an earthquake, by virtue of its location. For example, during a postulated earthquake, a relay on the fourth floor of a building would likely experience a different magnitude of shaking compared to a relay on the first floor. The SRF accounts for this difference. SAPHIRE does not maintain the SRF information separately. Before entering the seismic fragility data, the SRF needs to be factored in, and then the SRF-adjusted fragility data is entered into the database.

To enter seismic data into a seismic basic event record, go to the Failure Model for a selected basic event. Selecting "G" or "H" defines the basic event as a seismic basic event. The "G" and "H" simply identify the basis for the assumed magnitude of the peak ground acceleration (PGA) or g-level, for initially generating cut sets. If the "G" option is selected, the user will need to specify a particular g level to be used (with the fragility curve) to calculate a point-estimate probability for the cut set generation process. The "H" option tells SAPHIRE to utilize the highest g level found on the user-specified hazard curve.

10.3.4 Generating and Quantifying Cut Sets

SAPHIRE requires a failure probability for each basic event to generate cut sets. For the seismic basic events, this requires an assumed earthquake magnitude to combine with the basic event fragility data to produce a failure probability. The user can choose from two options, which are set individually for each basic event, the "G" or "H" option described in Section 10.6. Note that if the user expects to enable the cut set truncation feature when generating cut sets, the most prudent approach is to specify the highest g-level (i.e., PGA in units of gravity) that might be considered in subsequent calculations. Otherwise, cut sets that might be important in later analyses would be truncated because their seismic-induced failure probabilities fall below the truncation limit.

Seismic cut set quantification is performed similar to a regular analysis. The only difference is that the specific Model Type to be evaluated needs to be indicated during the analysis. An important feature to keep in mind is that only a single cut set list is maintained for each system, sequence, or end state in SAPHIRE. This limit also applies to seismic Model Type calculations.

11. COMPOUND EVENTS

In SAPHIRE, a compound event is a basic event that has parameters that are basic events and constants, where these parameters are utilized by way of an external calculation. Further, these parameters or sub events may themselves be other compound events, allowing a nested modeling approach. Each of these compound events combines the sub-events based upon the selected function to determine its value. These functions are called plug-in functions or just plug-ins. A plug-in is software that follows an algorithm that combines the sub events and constants. Currently there are eight plug-in libraries included in SAPHIRE. These eight plug-in libraries and their included functions are explained briefly below. Four of these plug-in libraries deal with the calculation of common-cause failure and the other three plug-in libraries are more general purpose in their nature.

Common-Cause Plug-Ins

- PlugCCFMGL – Performs Common Cause Failure calculations based on the Multiple Greek Letter methodology.
- PlugCCFAlpha - Performs Common Cause Failure calculations based on the Alpha Factor with the non-staggered testing methodology.
- PlugCCFStag - Performs Common Cause Failure calculations based on the Alpha Factor with the staggered testing methodology.
- PlugCCFBeta - Performs Common Cause Failure calculations based on the Beta Factor methodology.

General Purpose Plug-Ins

- PlugUtil - Utility library to perform general mathematical operations on basic events (e.g., add two events together, calculate the minimal cut set upper bound of a group of basic events, and divide two basic events).
- PlugTimeSeries - Library to determine the probability that a process (a series of events sequential in time) spans a duration longer than the mission time. Each of the sequential events (up to 19) should be represented by other basic events with their own uncertainty distributions.

PlugCapacityLoad - Library to determine the chance that a load on a component exceeds its capacity (for example, stress versus strength). The load-capacity plug-in take applicable input values for only two parameters, the load and the capacity, and then calculates the probability that the load will be larger than the capacity. This probability is the component's failure probability because if the load on a component is higher than its actual capacity, the component will fail (by definition).

Plug_LoopRec - Library sums the frequency (λ) for the five classifications: Plant Centered, Grid Related, Switchyard Related, Severe Weather, and Extreme Weather. The event record containing the calculated probability is output. The library also has a procedure that calculates plant specific diesel generator nonrecovery probabilities.

For a basic event, the "C" calculation type is used to indicate that the basic event is a "compound event." Compound events are basic events that utilize an external (to SAPHIRE) calculation to determine its probability. These external calculations are contained within a Windows-compatible library, called a dynamic link library (DLL). Typically, the common DLLs are stored in the SAPHIRE tools folder and include calculations such as common cause failure probabilities and general calculations (via the "utility" library). SAPHIRE will first check the project folder for the DLL and, if not found there, will search the SAPHIRE tools folder.

The utility compound library has procedures for general operations such as adding basic events together, multiplying events, dividing events, etc. In other words, it enables a handful of general purpose mathematical calculations to be performed on any basic events. Included in this library is the "min-cut" calculation which will take the minimal cut set upperbound calculation on the specified basic events (up to 20 per compound event). One could utilize this calculation to make a model via the "supercomponent" approach where a basic event is represented by other lower-level basic events. The minimal cut sets would only contain the higher-level event (the supercomponent) but the event's probability would automatically be determined by the compound DLL calculation. Using these compound events allow the user to customize their risk and reliability assessment to handle complex calculations specific to their analysis needs. The remainder of this section will cover, in detail, the calculations provided by the common-cause failure plug-in.

11.1 General Structure of the Common Cause Failure Plug-in

The older common-cause failure (CCF) plug-in libraries are based on work by Rasmuson (1998). The actual functions of the CCF plug-ins will be explained below. A group of four components with a failure of all four components necessary was selected to demonstrate the

functionality, but other group sizes (up to six) may be specified in SAPHIRE. Note that the CCF plug-in has been superseded by the CCF R Calculation Type basic event.

The event failures are noted as follows:

- I_1, I_2, I_3, I_4 - Indicates the independent failure of the subscripted component.
- $C_{12}, C_{13}, C_{14},$
- C_{23}, C_{24}, C_{34} - Indicates the common-cause failure of the 2 subscripted components.
- $C_{123}, C_{124}, C_{134},$
- C_{234} - Indicates the common-cause failure of the 3 subscripted components.
- C_{1234} - Indicates the common-cause failure of the 4 subscripted components.

The Basic Parameter Model for CCF analysis defines the following:

$$1_T = I_1 \cup C_{12} \cup C_{13} \cup C_{14} \cup C_{123} \cup C_{124} \cup C_{134} \cup C_{1234}$$

$$2_T = I_2 \cup C_{12} \cup C_{23} \cup C_{24} \cup C_{123} \cup C_{124} \cup C_{234} \cup C_{1234}$$

$$3_T = I_3 \cup C_{13} \cup C_{23} \cup C_{34} \cup C_{123} \cup C_{134} \cup C_{234} \cup C_{1234}$$

$$4_T = I_4 \cup C_{14} \cup C_{24} \cup C_{34} \cup C_{124} \cup C_{134} \cup C_{234} \cup C_{1234}$$

where the subscript T denotes total failure from all causes.

The failure probability of $1_T, 2_T, 3_T,$ or 4_T is given by the following equation:

$$Q_T = Q_1 + 3Q_2 + 3Q_3 + Q_4$$

where $Q_1 = P[I_1] = P[I_2] = P[I_3] = P[I_4], Q_2 = P[C_{12}] = P[C_{13}] = P[C_{14}] = P[C_{23}] = P[C_{24}] = P[C_{34}], Q_3 = P[C_{123}] = P[C_{124}] = P[C_{134}] = P[C_{234}],$ and $Q_4 = P[C_{1234}].$

The definitions of the Q Values, $Q_1, Q_2, Q_3, Q_4, \dots, Q_n,$ are where the CCF plug-in libraries differ. We will highlight the differences for the different methods.

Multiple Greek Letter methodology defines the Q values in the following manner.

$$Q_1^{(4)} = (1-\beta)Q_T$$

$$Q_2^{(4)} = 1/3 \beta(1-\gamma)Q_T$$

$$Q_3^{(4)} = 1/3 \beta\gamma(1-\delta)Q_T$$

$$Q_4^{(4)} = \beta\gamma\delta Q_T$$

The generalization of this definition is given by

$$Q_k^{(m)} = \frac{1}{\binom{m-1}{k-1}} \prod_{i=1}^k p_i (1 - p_{k+1}) Q_T \quad (k = 1, \dots, p_{m+1} = 0)$$

where

$$p_1 = 1, p_2 = \beta, p_3 = \gamma, p_4 = \delta, \dots, p_{m+1} = 0$$

and

$$\binom{m-1}{k-1} = \frac{(m-1)!}{(k-1)!(m-k)!}$$

Alpha-Factor Model (Non –staggered) methodology defines the Q values in the following manner.

$$Q_1^{(4)} = \frac{\alpha_1^{(4)}}{\alpha_t} Q_T$$

$$Q_2^{(4)} = \frac{2\alpha_2^{(4)}}{3\alpha_t} Q_T$$

$$Q_3^{(4)} = \frac{\alpha_3^{(4)}}{\alpha_t} Q_T$$

$$Q_4^{(4)} = 4 \frac{\alpha_4^{(4)}}{\alpha_t} Q_T$$

The generalization of this definition is given by

$$Q_k^{(m)} = \frac{k}{\binom{m-1}{k-1}} \left(\frac{\alpha_k^{(m)}}{\alpha_t} Q_T \right)$$

where

$$\alpha_t = \sum_{k=1}^m k \alpha_k^{(m)}$$

and

$$\binom{m-1}{k-1} = \frac{(m-1)!}{(k-1)!(m-k)!}$$

Alpha-Factor Staggered Testing methodology defines the Q values in the following manner:

$$Q_1^{(4)} = \alpha_1^{(4)} Q_T$$

$$Q_2^{(4)} = \frac{1}{3} \alpha_2^{(4)} Q_T$$

$$Q_3^{(4)} = \frac{1}{3} \alpha_3^{(4)} Q_T$$

$$Q_4^{(4)} = \alpha_4^{(4)} Q_T$$

The generalization of this definition is given by

$$Q_k^{(m)} = \frac{1}{\binom{m-1}{k-1}} (\alpha_k^{(m)} Q_T)$$

where

$$\binom{m-1}{k-1} = \frac{(m-1)!}{(k-1)!(m-k)!}$$

Beta-Factor methodology defines the Q values in the following manner.

$$Q_1^{(4)} = (1-\beta) Q_T$$

$$Q_2^{(4)} = 0$$

$$Q_3^{(4)} = 0$$

$$Q_4^{(4)} = \beta Q_T$$

The generalization of this definition is given by

$$Q_k^{(m)} = (1-\beta) Q_T \quad \text{for } k = 1$$

$$\text{for } m > k > 1$$

$$\beta Q_T \quad \text{for } k = m$$

There are equations to transform the alpha factors to the MGL factors and vice-versa. These are explained here to allow the example to be worked in those three functions, but they are not

used in any of the CCF plug-ins. Since our example is for a group of four components, we will assume all four components must fail. Another way of stating this is a system of four components with a success criterion of 1-of-4. Further, assume the value of $Q_T = 0.01$.

The MGL parameters are:

$$\begin{aligned}\beta &= 0.01 \\ \gamma &= 0.4 \\ \delta &= 0.4\end{aligned}$$

These factors are converted to alpha-staggered by the following transformations, which are specific to a group of four components.

$$\begin{aligned}\alpha_1 &= 1.0 - \beta = 0.99 \\ \alpha_2 &= (1.0 - \gamma)\beta = 0.006 \\ \alpha_3 &= (1.0 - \delta)\beta\gamma = 0.0024 \\ \alpha_4 &= \beta\gamma\delta = 0.0016\end{aligned}$$

The MGL factors are converted to alpha (non-staggered) by the following transformations, which are specific to a group of four components.

$$\begin{aligned}\alpha_1 &= \frac{12(1 - \beta)}{6(2 - \beta) - \beta\gamma(2 + \delta)} = 0.9958 \\ \alpha_2 &= \frac{6\beta(1 - \gamma)}{6(2 - \beta) - \beta\gamma(2 + \delta)} = 0.0030 \\ \alpha_3 &= \frac{4\beta\gamma(1 - \delta)}{6(2 - \beta) - \beta\gamma(2 + \delta)} = 0.0008 \\ \alpha_4 &= \frac{3\beta\gamma\delta}{6(2 - \beta) - \beta\gamma(2 + \delta)} = 0.0004\end{aligned}$$

A key to understanding these CCF plug-in functions is to work with minimal cut sets. The following list contains the 15 cut sets (think in terms of combinations of failures for the four components) that describe the failure of these components.

$[I_1, I_2, I_3, I_4], [I_1, I_2, C_{34}], [I_1, I_3, C_{24}], [I_1, I_4, C_{23}], [I_2, I_3, C_{14}], [I_2, I_4, C_{13}],$

$[I_3, I_4, C_{12}], [C_{12}, C_{34}], [C_{13}, C_{24}], [C_{14}, C_{23}], [I_1, C_{234}], [I_2, C_{134}], [I_3, C_{124}],$

[I₄, C₁₂₃], and [C₁₂₃₄].

The failure probability for the undesired event $S = 1_T \cap 2_T \cap 3_T \cap 4_T$ which we denote by Q_s is given, in terms of the basic parameter model by:

$$Q_s = Q_1^4 + 6Q_1^2Q_2 + 3Q_2^2 + 4Q_1Q_3 + Q_4$$

Q Values from MGL

Calculating the Q values from the MGL factors yields the following:

$$Q_1^{(4)} = (1-\beta)Q_T = (1-0.01)*0.01 = 9.9E-3$$

$$Q_2^{(4)} = 1/3 \beta(1-\gamma)Q_T = 1/3*0.01*(1-0.4)*0.01 = 2.0E-5$$

$$Q_3^{(4)} = 1/3 \beta\gamma(1-\delta)Q_T = 1/3*0.01*.4*(1-0.4)*0.01 = 8.0E-6$$

$$Q_4^{(4)} = \beta\gamma\delta Q_T = 0.01*0.4*0.4*0.01 = 1.6E-5$$

Calculating value of Q_s from the above equation yields the following:

$$\begin{aligned} Q_s &= Q_1^4 + 6Q_1^2Q_2 + 3Q_2^2 + 4Q_1Q_3 + Q_4 \\ &= 9.9E-3^4 + 6*(9.9E-3^2 * 2.0E-5) + 3*(2.0E-5)^2 + 4*(9.9E-3)*8.0E-6 + 1.6E-5 \\ &= 1.633E-5 \end{aligned}$$

Q Values from Alpha Non-Staggered

Calculating the Q values from the Alpha Non-Staggered methodology factors yields the following:

$$\begin{aligned} \alpha_i &= \sum_{k=1}^m k\alpha_k^{(m)} = 1*0.9958 + 2*0.0030 + 3*0.0008 + 4*0.0004 \\ &= 1.0058 \end{aligned}$$

$$Q_1^{(4)} = \frac{\alpha_1^{(4)}}{\alpha_i} Q_T = 0.9958/1.0058*0.01 = 9.9E-3$$

$$Q_2^{(4)} = \frac{2\alpha_2^{(4)}}{3\alpha_i} Q_T = 2*0.0030/3*1.0058*0.01 = 1.99E-5$$

$$Q_3^{(4)} = \frac{\alpha_3^{(4)}}{\alpha_1} Q_T = 0.0008/1.0058*0.01 = 7.95E-6$$

$$Q_4^{(4)} = \frac{4\alpha_4^{(4)}}{\alpha_1} Q_T = 4*0.0004/1.0058*0.01 = 1.59E-5$$

Calculating value of Q_s from the above equation yields the following:

$$\begin{aligned} Q_s &= Q_1^4 + 6Q_1^2Q_2 + 3Q_2^2 + 4Q_1Q_3 + Q_4 \\ &= 9.9E-3^4 + 6*(9.9E-3^2 * 1.99E-5) + 3*(1.99E-5)^2 + 4*(9.9E-3)*7.95E-6 + 1.59E-5 \\ &= 1.593E-5 \end{aligned}$$

Q Values from Alpha Staggered

Calculating the Q values from the Alpha Staggered methodology factors yields the following:

$$Q_1^{(4)} = \alpha_1^{(4)} Q_T = 0.99*0.01 = 9.9E-3$$

$$Q_2^{(4)} = \frac{1}{3}\alpha_2^{(4)} Q_T = 0.006*0.01/3 = 2.0E-5$$

$$Q_3^{(4)} = \frac{1}{3}\alpha_3^{(4)} Q_T = 0.0024*0.01/3 = 8.0E-6$$

$$Q_4^{(4)} = \alpha_4^{(4)} Q_T = 0.0016*0.01 = 1.6E-5$$

Calculating value of Q_s from the above equation yields the following:

$$\begin{aligned} Q_s &= Q_1^4 + 6Q_1^2Q_2 + 3Q_2^2 + 4Q_1Q_3 + Q_4 \\ &= 9.9E-3^4 + 6*(9.9E-3^2 * 2.0E-5) + 3*(2.0E-5)^2 + 4*(9.9E-3)*8.0E-6 + 1.6E-5 \\ &= 1.633E-5 \end{aligned}$$

Q Values from Beta Factor

Calculating the Q values from the Beta factor methodology yields the following:

$$Q_1^{(4)} = (1-\beta)Q_T = (1-0.01)*0.01 = 9.9E-3$$

$$Q_2^{(4)} = 0$$

$$Q_3^{(4)} = 0$$

$$Q_4^{(4)} = \beta Q_T = 0.01 * 0.01 = 1.0E-4$$

Calculating value of Q_s from the above equation yields the following:

$$\begin{aligned} Q_s &= Q_1^4 + 6Q_1^2Q_2 + 3Q_2^2 + 4Q_1Q_3 + Q_4 \\ &= 9.9E-3^4 + 6*(9.9E-3^2 * 0.0) + 3*(0.0)^2 + 4*(9.9E-3)*0.0 + 1.0E-4 \\ &= 1.0001E-4 \end{aligned}$$

Note that the beta factor is conservative since it assumes that a conditional failure will fail all redundant trains in the CCF group.

11.2 Using the Common Cause Failure Plug-in

This section will describe how the older CCF plug-in calculates probabilities during an event assessment. This is what the CCF value is, given that one or more of the input events has failed or is out of service. SAPHIRE can use two treatments to calculate the CCF value. The first is based on the work of Rasmuson (1998) and the principle that the potential exists for common-cause failures to occur in any situation. The second is based on the concept of reducing component count whenever an independent event fails or is taken out of service (not for a failure). We will look at each in turn.

Using the same group of four components with a failure of all four components necessary, let us examine the impact of one component in a failed state, where we consider the possibility of the remaining components failing conditional on the first failure. Next we'll examine two components in a failed state, then three components in a failed state. It is important to work this through in terms of cut sets. For the example, we will utilize the Q values from the MGL methodology.

The One Component Failure Case

First assume that component 1 has failed, so the conditional failure probability of S given 1_T is given by:

$$\begin{aligned} P[S | 1_T] &= \frac{P[1_T \cap 2_T \cap 3_T \cap 4_T]}{P[1_T]} \\ &= Q_s / Q_T \\ &= 1.633E-5 / 1.0E-2 = 1.633E-3 \end{aligned}$$

The Two Component Failure Case

Next assume that components 1 and 2 have failed, so the conditional failure probability of S given 1_T and 2_T is given by:

$$P[S | 1_T \cap 2_T] = \frac{P[1_T \cap 2_T \cap 3_T \cap 4_T]}{P[1_T \cap 2_T]}$$

The numerator remains the same. It is the calculation of the denominator that is vital. Lets go back to the Basic Parameter Model for CCF analysis definition of 1_T and 2_T :

$$1_T = I_1 \cup C_{12} \cup C_{13} \cup C_{14} \cup C_{123} \cup C_{124} \cup C_{134} \cup C_{1234}$$

$$2_T = I_2 \cup C_{12} \cup C_{23} \cup C_{24} \cup C_{123} \cup C_{124} \cup C_{234} \cup C_{1234}$$

The intersection of 1_T and 2_T yields:

$$\begin{aligned} 1_T \cap 2_T &= (I_1 \cup C_{12} \cup C_{13} \cup C_{14} \cup C_{123} \cup C_{124} \cup C_{134} \cup C_{1234}) \cap \\ &\quad (I_2 \cup C_{12} \cup C_{23} \cup C_{24} \cup C_{123} \cup C_{124} \cup C_{234} \cup C_{1234}) \\ &= C_{12} \cup C_{123} \cup C_{124} \cup C_{1234} \end{aligned}$$

Using the Q values derived from the MGL methodology, we can calculate

$$\begin{aligned} P[1_T \cap 2_T] &= Q_2 + 2Q_3 + Q_4 \\ &= 2.0E-5 + 2 * 8.0E-6 + 1.6E-5 \\ &= 5.2E-5 \end{aligned}$$

and

$$\begin{aligned} Q_s &= Q_1^4 + 6Q_1^2 Q_2 + 3Q_2^2 + 4Q_1 Q_3 + Q_4 \\ &= 9.9E-3^4 + 6*(9.9E-3^2 * 2.0E-5) + 3*(2.0E-5)^2 + 4*(9.9E-3)*8.0E-6 + 1.6E-5 \\ &= 1.633e-5 \end{aligned}$$

Therefore

$$\begin{aligned} Q_s / P[1_T \cap 2_T] &= 1.633e-5 / 5.2E-5 \\ &= 0.3140 \end{aligned}$$

The Three Component Failure Case

Next assume that components 1 and 2 and 3 have failed, so the conditional failure probability of S given 1_T and 2_T and 3_T is given by:

$$P[S | 1_T \cap 2_T \cap 3_T] = \frac{P[1_T \cap 2_T \cap 3_T \cap 4_T]}{P[1_T \cap 2_T \cap 3_T]}$$

The numerator again remains the same. It is the calculation of the denominator that is vital. Lets go back to the Basic Parameter Model for CCF analysis definition of 1_T , 2_T and 3_T :

$$1_T = I_1 \cup C_{12} \cup C_{13} \cup C_{14} \cup C_{123} \cup C_{124} \cup C_{134} \cup C_{1234}$$

$$2_T = I_2 \cup C_{12} \cup C_{23} \cup C_{24} \cup C_{123} \cup C_{124} \cup C_{234} \cup C_{1234}$$

$$3_T = I_3 \cup C_{13} \cup C_{23} \cup C_{34} \cup C_{123} \cup C_{134} \cup C_{234} \cup C_{1234}$$

The intersection of 1_T and 2_T and 3_T yields:

$$\begin{aligned} 1_T \cap 2_T \cap 3_T &= (I_1 \cup C_{12} \cup C_{13} \cup C_{14} \cup C_{123} \cup C_{124} \cup C_{134} \cup C_{1234}) \cap \\ &\quad (I_2 \cup C_{12} \cup C_{23} \cup C_{24} \cup C_{123} \cup C_{124} \cup C_{234} \cup C_{1234}) \cap \\ &\quad (I_3 \cup C_{13} \cup C_{23} \cup C_{34} \cup C_{123} \cup C_{134} \cup C_{234} \cup C_{1234}) \\ &= C_{123} \cup C_{1234} \end{aligned}$$

Using the Q values derived from the MGL methodology, we can calculate

$$P[1_T \cap 2_T \cap 3_T] = Q_3 + Q_4 = 8.0E-6 + 1.6E-5 = 2.4E-5$$

and

$$\begin{aligned} Q_s &= Q_1^4 + 6Q_1^2Q_2 + 3Q_2^2 + 4Q_1Q_3 + Q_4 \\ &= 9.9E-3^4 + 6*(9.9E-3^2 * 2.0E-5) + 3*(2.0E-5)^2 + 4*(9.9E-3)*8.0E-6 + 1.6E-5 \\ &= 1.633e-5 \end{aligned}$$

Therefore

$$\begin{aligned} Q_s / P[1_T \cap 2_T \cap 3_T] &= 1.633e-5 / 2.4E-5 \\ &= 0.6804 \end{aligned}$$

The One Component Outage Case

First assume that component 1 has failed, so the conditional failure probability of S given 1_M is given by:

$$P[S | 1_M] = \frac{P[1_T \cap 2_T \cap 3_T \cap 4_T]}{P[1_M]}$$

Here are the fifteen cut sets that represent the possible failures of the four components.

$[I_1, I_2, I_3, I_4], [I_1, I_2, C_{34}], [I_1, I_3, C_{24}], [I_1, I_4, C_{23}], [I_2, I_3, C_{14}], [I_2, I_4, C_{13}],$

$[I_3, I_4, C_{12}], [C_{12}, C_{34}], [C_{13}, C_{24}], [C_{14}, C_{23}], [I_1, C_{234}], [I_2, C_{134}], [I_3, C_{124}],$

$[I_4, C_{123}],$ and $[C_{1234}].$

Let's modify the cut sets based on the knowledge that I_1 is out.

$[I_2, I_3, I_4], [I_2, C_{34}], [I_3, C_{24}], [I_4, C_{23}], [I_2, I_3, C_{14}], [I_2, I_4, C_{13}],$

$[I_3, I_4, C_{12}], [C_{12}, C_{34}], [C_{13}, C_{24}], [C_{14}, C_{23}], [C_{234}], [I_2, C_{134}], [I_3, C_{124}],$

$[I_4, C_{123}],$ and $[C_{1234}].$

The probability then becomes:

$$\begin{aligned} Q_S | 1_M &= Q_1^3 + 3Q_1Q_2 + 3Q_1^2Q_2 + 3Q_2^2 + Q_3 + 3Q_1Q_3 + Q_4 \\ &= 9.9E-3^3 + 3*(9.9E-3 * 2.0E-5) + 3*(9.9E-3^2 * 2.0E-5) + 3*(2.0E-5)^2 + 9.9E-3 + \\ &\quad 3*(9.9E-3)*8.0E-6 + 1.6E-5 \\ &= 2.58E-5 \end{aligned}$$

The Two Component Outage Case

First assume that component 1 and 2 are out for preventive maintenance, so the conditional failure probability of S given 1_M and 2_M is given by:

$$P[S | 1_M \cap 2_M] = \frac{P[1_T \cap 2_T \cap 3_T \cap 4_T]}{P[1_M \cap 2_M]}$$

Here are the fifteen cut sets that represent the possible failures of the four components.

$[I_1, I_2, I_3, I_4], [I_1, I_2, C_{34}], [I_1, I_3, C_{24}], [I_1, I_4, C_{23}], [I_2, I_3, C_{14}], [I_2, I_4, C_{13}],$

$[I_3, I_4, C_{12}], [C_{12}, C_{34}], [C_{13}, C_{24}], [C_{14}, C_{23}], [I_1, C_{234}], [I_2, C_{134}], [I_3, C_{124}],$

$[I_4, C_{123}],$ and $[C_{1234}].$

Let's modify the cut sets based on the knowledge that I_1 and I_2 are out.

$[I_3, I_4], [C_{34}], [I_3, C_{24}], [I_4, C_{23}], [I_3, C_{14}], [I_4, C_{13}],$

$[C_{13}, C_{24}], [C_{14}, C_{23}], [C_{234}], [C_{134}], [I_3, C_{124}],$

$[I_4, C_{123}],$ and $[C_{1234}].$

The probability then becomes:

$$\begin{aligned}
 Q_S | P[1_M \cap 2_M] &= Q_1^2 + Q_2 + 4Q_1Q_2 + 2Q_2^2 + 2Q_3 + 2Q_1Q_3 + Q_4 \\
 &= 9.9E-3^2 + 2.0E-5 + 4*(9.9E-3 * 2.0E-5) + 2*(2.0E-5)^2 + 2*8.0E-6 + \\
 &\quad 2*(9.9E-3)*8.0E-6 + 1.6E-5 \\
 &= 1.51E-4
 \end{aligned}$$

The Three Component Outage Case

First assume that component 1, 2, and 3 are out for preventive maintenance, so the conditional failure probability of S given $1_M, 2_M$ and 3_M is given by:

$$P[S | 1_M \cap 2_M \cap 3_M] = \frac{P[I_T \cap 2_T \cap 3_T \cap 4_T]}{P[1_M \cap 2_M \cap 3_M]}$$

Here are the fifteen cut sets that represent the possible failures of the four components.

$[I_1, I_2, I_3, I_4], [I_1, I_2, C_{34}], [I_1, I_3, C_{24}], [I_1, I_4, C_{23}], [I_2, I_3, C_{14}], [I_2, I_4, C_{13}],$

$[I_3, I_4, C_{12}], [C_{12}, C_{34}], [C_{13}, C_{24}], [C_{14}, C_{23}], [I_1, C_{234}], [I_2, C_{134}], [I_3, C_{124}],$

$[I_4, C_{123}],$ and $[C_{1234}].$

Let's modify the cut sets based on the knowledge that $I_1, I_2,$ and I_3 are out.

$[I_4], [C_{34}], [C_{24}], [C_{14}], [C_{234}], [C_{134}], [C_{124}],$ and $[C_{1234}].$

The probability then becomes:

$$\begin{aligned}
 Q_s|P[1_M \cap 2_M \cap 3_M] &= Q_1 + 3Q_2 + 3Q_3 + Q_4 \\
 &= 9.9E-3 + 3*2.0E-5 + 3*8.0E-6 + 1.6E-5 \\
 &= 1.00E-2
 \end{aligned}$$

Using the same group of four components with a failure of all four components necessary, let us examine the impact of components that are either failed or out for preventive maintenance. The potential exists for common-cause failures to occur. The independent failure of the component out for maintenance cannot occur, but the common-cause failure can occur for others in the group. First, we will examine the one component that has failed and then the other component that is out for preventive maintenance. Then, the case of two components that fail with one more component that is out for maintenance will be explored. Lastly, we will discuss the case where one component is failed with two more components out for maintenance.

Again, we list the fifteen cut sets that represent the possible failures of the four components.

$$\begin{aligned}
 &[I_1, I_2, I_3, I_4], [I_1, I_2, C_{34}], [I_1, I_3, C_{24}], [I_1, I_4, C_{23}], [I_2, I_3, C_{14}], [I_2, I_4, C_{13}], \\
 &[I_3, I_4, C_{12}], [C_{12}, C_{34}], [C_{13}, C_{24}], [C_{14}, C_{23}], [I_1, C_{234}], [I_2, C_{134}], [I_3, C_{124}], \\
 &[I_4, C_{123}], \text{ and } [C_{1234}].
 \end{aligned}$$

One Component Failed and another Out for Preventive Maintenance

First assume that component 1 has failed and component 2 is out for maintenance, so the conditional failure probability of S given 1_T and 2_M is given by:

$$P[S|1_T \cap 2_M] = \frac{P[1_T \cap 2_T \cap 3_T \cap 4_T] / P[2_M]}{P[1_T]}$$

Here are the fifteen cut sets that represent the possible failures of the four components.

$$\begin{aligned}
 &[I_1, I_2, I_3, I_4], [I_1, I_2, C_{34}], [I_1, I_3, C_{24}], [I_1, I_4, C_{23}], [I_2, I_3, C_{14}], [I_2, I_4, C_{13}], \\
 &[I_3, I_4, C_{12}], [C_{12}, C_{34}], [C_{13}, C_{24}], [C_{14}, C_{23}], [I_1, C_{234}], [I_2, C_{134}], [I_3, C_{124}], \\
 &[I_4, C_{123}], \text{ and } [C_{1234}].
 \end{aligned}$$

Let's modify the cut sets based on the knowledge that I_2 is out for maintenance.

$$\begin{aligned}
 &[I_1, I_3, I_4], [I_1, C_{34}], [I_1, I_3, C_{24}], [I_1, I_4, C_{23}], [I_3, C_{14}], [I_4, C_{13}], \\
 &[I_3, I_4, C_{12}], [C_{12}, C_{34}], [C_{13}, C_{24}], [C_{14}, C_{23}], [I_1, C_{234}], [C_{134}], [I_3, C_{124}],
 \end{aligned}$$

$[I_4, C_{123}]$, and $[C_{1234}]$.

The probability then becomes:

$$\begin{aligned}
 Q_S | 2_M | 1_T &= (Q_1^3 + 3Q_1Q_2 + 3Q_1^2Q_2 + 3Q_2^2 + Q_3 + 3Q_1Q_3 + Q_4) / Q_T \\
 &= (9.9E-3^3 + 3*(9.9E-3 * 2.0E-5) + 3*(9.9E-3^2 * 2.0E-5) + 3*(2.0E-5)^2 + \\
 &\quad 9.9E-3 + 3*(9.9E-3)*8.0E-6 + 1.6E-5) / 1.0E-2 \\
 &= 2.58E-3
 \end{aligned}$$

Two Components Failed and One Component Out for Preventive Maintenance

First assume that component 1 and 2 have failed and 3 is out for preventive maintenance, so the conditional failure probability of S given 1_T and 2_T and 3_M is given by:

$$P[S | 1_T \cap 2_T \cap 3_M] = \frac{P[I_T \cap 2_T \cap 3_T \cap 4_T] / 3_M}{P[I_T \cap 2_T]}$$

First the numerator is changed to reflect the impact of 3_M . Here are the fifteen cut sets that represent the possible failures of the four components.

$[I_1, I_2, I_3, I_4]$, $[I_1, I_2, C_{34}]$, $[I_1, I_3, C_{24}]$, $[I_1, I_4, C_{23}]$, $[I_2, I_3, C_{14}]$, $[I_2, I_4, C_{13}]$,
 $[I_3, I_4, C_{12}]$, $[C_{12}, C_{34}]$, $[C_{13}, C_{24}]$, $[C_{14}, C_{23}]$, $[I_1, C_{234}]$, $[I_2, C_{134}]$, $[I_3, C_{124}]$,
 $[I_4, C_{123}]$, and $[C_{1234}]$.

Let's modify the cut sets based on the knowledge that component I_3 is out.

$[I_1, I_2, I_4]$, $[I_1, I_2, C_{34}]$, $[I_1, C_{24}]$, $[I_1, I_4, C_{23}]$, $[I_2, C_{14}]$, $[I_2, I_4, C_{13}]$,
 $[I_3, I_4, C_{12}]$, $[C_{12}, C_{34}]$, $[C_{13}, C_{24}]$, $[C_{14}, C_{23}]$, $[I_1, C_{234}]$, $[I_2, C_{134}]$, $[C_{124}]$,
 $[I_4, C_{123}]$, and $[C_{1234}]$.

$$\begin{aligned}
 Q_S | P[3_M] &= Q_1^3 + 3Q_1Q_2 + 3Q_1^2Q_2 + 3Q_2^2 + Q_3 + 3Q_1Q_3 + Q_4 \\
 &= 9.9E-3^3 + 3*(9.9E-3 * 2.0E-5) + 3*(9.9E-3^2 * 2.0E-5) + 3*(2.0E-5)^2 + \\
 &\quad 9.9E-3 + 3*(9.9E-3)*8.0E-6 + 1.6E-5
 \end{aligned}$$

$$= 2.58E-5$$

Then the denominator is calculated and applied to the numerator. Lets go back to the Basic Parameter Model for CCF analysis definition of 1_T and 2_T :

$$1_T = I_1 \cup C_{12} \cup C_{13} \cup C_{14} \cup C_{123} \cup C_{124} \cup C_{134} \cup C_{1234}$$

$$2_T = I_2 \cup C_{12} \cup C_{23} \cup C_{24} \cup C_{123} \cup C_{124} \cup C_{234} \cup C_{1234}$$

The intersection of 1_T and 2_T yields:

$$\begin{aligned} 1_T \cap 2_T &= (I_1 \cup C_{12} \cup C_{13} \cup C_{14} \cup C_{123} \cup C_{124} \cup C_{134} \cup C_{1234}) \cap \\ &\quad (I_2 \cup C_{12} \cup C_{23} \cup C_{24} \cup C_{123} \cup C_{124} \cup C_{234} \cup C_{1234}) \\ &= C_{12} \cup C_{123} \cup C_{124} \cup C_{1234} \end{aligned}$$

Using the Q values derived from the MGL methodology, we can calculate

$$\begin{aligned} P[1_T \cap 2_T] &= Q_2 + 2Q_3 + Q_4 \\ &= 2.0E-5 + 2 * 8.0E-6 + 1.6E-5 \\ &= 5.2E-5 \end{aligned}$$

Combining the numerator and the denominator, the answer is

$$\begin{aligned} Q_s | P[2_M \cap 1_T \cap 2_T] &= (Q_1^3 + 3Q_1Q_2 + 3Q_1^2Q_2 + 3Q_2^2 + Q_3 + 3Q_1Q_3 + Q_4) / Q_2 + 2Q_3 + Q_4 \\ &= 2.58E-5 / 5.2E-5 \\ &= 4.96E-1 \end{aligned}$$

One Component Failed and Two Components Out for Preventive Maintenance

First assume that component 1 has failed and components 2 and 3 are out for preventive maintenance, so the conditional failure probability of S given 1_T , 2_M and 3_M is given by:

$$P[S | 1_T \cap 2_M \cap 3_M] = \frac{P[1_T \cap 2_T \cap 3_T \cap 4_T]}{P[1_T]}$$

Here are the fifteen cut sets that represent the possible failures of the four components.

$$[I_1, I_2, I_3, I_4], [I_1, I_2, C_{34}], [I_1, I_3, C_{24}], [I_1, I_4, C_{23}], [I_2, I_3, C_{14}], [I_2, I_4, C_{13}],$$

[I₃, I₄, C₁₂], [C₁₂, C₃₄], [C₁₃, C₂₄], [C₁₄, C₂₃], [I₁, C₂₃₄], [I₂, C₁₃₄], [I₃, C₁₂₄],
 [I₄, C₁₂₃], and [C₁₂₃₄].

Let's modify the cut sets based on the knowledge that I₂ and I₃ are out.

[I₁, I₄], [I₁, C₃₄], [I₁, C₂₄], [C₁₄], [I₄, C₁₃], [I₄, C₁₂], [C₁₂, C₃₄], [C₁₃, C₂₄],
 [I₁, C₂₃₄], [C₁₃₄], [C₁₂₄], [I₄, C₁₂₃], and [C₁₂₃₄].

The probability then becomes:

$$\begin{aligned}
 Q_s | P[1_T \cap 2_M \cap 3_M] &= (Q_1^2 + Q_2 + 4Q_1Q_2 + 2Q_2^2 + 2Q_3 + 2Q_1Q_3 + Q_4) / Q_T \\
 &= (9.9E-3^2 + 2.0E-5 + 4*(9.9E-3 * 2.0E-5) + 2*(2.0E-5)^2 + 2*8.0E- \\
 &\quad 6 + 2*(9.9E-3)*8.0E-6) + 1.6E-5) / 1.0E-2 \\
 &= 1.51E-2
 \end{aligned}$$

11.3 General Structure of the LOOP Recovery Plug-in

SAPHIRE version 8 is equipped with a loss of offsite power (LOOP) recovery plug-in for calculating LOOP initiating event frequencies and offsite power recovery failure probabilities. The plug-in also calculates plant specific diesel generator nonrecovery probabilities. The following sections describe the plug-in calculations that must be performed and the parameters that must be stored to make the plug-in work. A section is also provided that describes how to configure a SAPHIRE basic event to use the plug-in to calculate loss of offsite power initiating event frequency, or offsite power recovery failure probability.

SAPHIRE assumes the LOOP initiating event frequency (λ_T) is the sum from all the individual LOOP frequency subclasses combined, or

$$\lambda_T = \sum_{i=1}^4 \lambda_i \tag{11-1}$$

The four classes of LOOP initiating event identified are:

- Plant-centered. A LOOP event in which the design and operational characteristics of the nuclear power plant itself play the major role in the cause and duration of the loss of offsite power. The line of demarcation between plant-centered and switchyard-centered LOOP events is the nuclear power plant main and station auxiliary transformers high-voltage terminals. Both transformers are considered to be part of the switchyard.

- Switchyard-centered. A LOOP event in which the equipment, whether human-induced or actual equipment failure, in the switchyard play the major role in the loss of offsite power.
- Grid-related. A LOOP event in which the initial failure occurs in the interconnected transmission grid that is outside the direct control of plant personnel.
- Weather-related. A LOOP event caused by weather, in which the weather is widespread, not just centered at the site, and capable of major disruption. Weather is defined to be weather with forceful and non-localized effects.

The LOOP frequency and nonrecovery probability calculations that follow must be performed for each of the above LOOP subcategories separately, and must be performed for a composite representing all categories together. The LOOP plug-in must perform the calculation described by Equation 11-1 using specified parameters in such a way that both the correct point estimate and correct uncertainty distribution of δ_T is obtained from the uncertainty distributions of the δ_i . Given the use of SAPHIRE basic events to represent the δ_i , this should follow from the standard SAPHIRE sampling procedures.

LOOP Recovery Failure Probabilities

The expression used for calculating the probability of failing to recover offsite power is given by:

$$P_{OPRF}(t_{long} | t_{short}) = P(L > t_{long} | L > t_{short}) \quad (11-2)$$

where L is the duration of a LOOP, and t_{long} is a sequence-dependent time requirement that is greater than t_{short} . The interpretation of t_{long} and t_{short} are model and sequence specific. The most common application is to station blackout sequences where t_{long} will correspond to either a battery depletion time or a core uncover time and t_{short} will correspond to a short-term recovery interval based on the time to uncover the reactor core if no safety systems function. In the current generation of SPAR models t_{short} is most often zero unless there are multiple failure to recover offsite power events in a given sequence. In these sequences the first event calculation would use a t_{short} of zero and the remaining power recovery failure probabilities would be conditional on the previous failure event.

The probability that offsite power will not be recovered at time t is the fraction of all LOOP events with duration L greater than t , or

$$P(L > t) = \int_t^{\infty} f_L(l) dl = 1 - F_L(t) \quad (11-3)$$

where f_L is the density function for the distribution of observed LOOP durations, and F_L is the cumulative distribution form of f_L . Combining Equations 11-2 and 11-3 gives the general expression for offsite power recovery failure probabilities.

$$P_{OPRF}(t_{long} | t_{short}) = \frac{\int_{t_{short}}^{\infty} f_L(t) dt}{\int_{t_{short}}^{\infty} f(t) dt} = \frac{1 - F_L(t_{long})}{1 - F_L(t_{short})} \quad (11-4)$$

Equation 11-4 can be generalized so that recovery failure probabilities can be calculated when LOOP frequency and LOOP recovery information is divided into plant, switchyard, grid, and weather subclasses by frequency-weighting the class probabilities as follows

$$P_{OPRF}(t_{long} | t_{short}) = \frac{1}{\lambda_T} \sum_{i=1}^4 \lambda_i \frac{(1 - F_{L_i}(t_{long}))}{(1 - F_{L_i}(t_{short}))} \quad (11-5)$$

Equation 11-5 is the most general form for calculating LOOP nonrecovery probabilities that are consistent with past and anticipated future LOOP modeling methods.

Eide, et al provides a Weibull-based form of f_L as

$$f_L(t) = \frac{\alpha}{t} \left(\frac{t}{\beta} \right)^{\alpha} e^{-(t/\beta)^{\alpha}} \quad (11-6)$$

where

α = the distribution shape parameter, and

β = the distribution scale parameter.

Using Equation 11-4 and Equation 11-6 gives the following general expression for failure to recover offsite power at time t_{long} for LOOP class i , conditional on failure to recover offsite power at time t_{short}

$$P_{OPRF_i}(t_{long} | t_{short}) = \frac{\int_{t_{short}}^{\infty} f_{L_i}(l) dl}{\int_{t_{short}}^{\infty} f_{L_i}(l) dl} = \frac{e^{-(t_{long} / \beta_i)^{\alpha_i}}}{e^{-(t_{short} / \beta_i)^{\alpha_i}}} \quad (11-7)$$

Equation 11-7 is presently applied when evaluating a specific class of LOOP, as in initiating event assessments where the class of initiating event is known. For a frequency-weighted average recovery failure probability is required and Equation 11-7 becomes

$$P_{OPRF}(t_{long} | t_{short}) = \frac{1}{\lambda_T} \sum_{i=1}^4 \lambda_i \frac{e^{-(t_{long} / \beta_i)^{\alpha_i}}}{e^{-(t_{short} / \beta_i)^{\alpha_i}}} \quad (11-8)$$

Diesel Nonrecovery Probabilities

The Plug_LoopRec library provides a method to produce nonrecovery probabilities of an emergency diesel generator. By selecting the procedure type "DG_RECOVERY and providing SAPHIRE with the appropriate Recovery Time, Alpha, and Beta parameters, SAPHIRE will calculate a DG nonrecovery probability. Equation 11-6 is the general Weibull-based form for calculating DG nonrecovery probabilities.

12. PHASE MISSION CALCULATIONS

New to the SAPHIRE 8 design related to phased-mission analysis are the following attributes:

- General phase information (names, descriptions, time, character identifier) are stored at the project level and are user definable.
- Basic events are extended to allow for multiple phases (and associated failure models) to be specified.
- Event trees allow for phase information to be indicated as a function of moving through the event tree scenario.
- The cut set solving methods are phase-aware in order to provide correct analysis results.

SAPHIRE allows modeling system configuration and associated failure (or success) as a function of the mission phase. At the basic event level, the various SAPHIRE failure models (e.g., fails on demand, fails to run without repair) will be selectable for *each* phase to determine the basic events phase susceptibility. The analysis methods will modify cut sets that have phase information, for example a component that fails in an early phase will also be failed in a later phase unless it has been repaired between phases.

12.1.1 General Phase Information

A SAPHIRE project allows the user to specify the general phase and analysis type applicability information. This information will include:

Item	Data Structure	Description
Phase (or applicability) name	24 characters	A general (short) name for the phase, for example "Phase 1" or "Seismic"
Phase ID	2 characters	A (short) identifier used to append to a basic event name in order to generate phase mission cut sets
Phase description	120 characters	General descriptive text
Phase duration	Time variable	The duration of the phase
Phase units	Time units	The time units (hr, min, sec, etc.) on the phase duration
Preceding phase	Name	The name of the preceding phase, if applicable
Phase color identifier	RGB Color	A color setting for graphical display use.

The user has access to the general phase information via the "Modify → Project → Phase" option. Note that the phase duration should be an actual duration value. The "preceding phase" indicates which phase precedes the phase. For example, a "phase 1" might have duration of

100 hours, with no preceding phase while "phase 2" has duration of 100 hours with a preceding phase of "phase 1." A total of 100 phases are available to the user within a single project.

12.1.2 Event Tree Phase or Applicability Modifications

Event trees are tied to one or more phases. For each top event in the event tree graphic, an "editing" node is available so the user may modify the nodal information (event tree text; assignment of fault trees or split fraction probabilities; phase information). Each node may be assigned to a single phase if desired—however, the phase assignment must proceed chronologically through the accident sequence. If a node does not have Phased-Mission information assigned to it, the last phase found in a preceding node is assumed to be applicable.

12.1.3 Analysis Capabilities Phase Modifications

Additional processing is required to ensure that the correct minimal cut sets are generated. For example, if a cut set is found where a component is failed during an early phase and is successful during a later phase, then this cut set would be removed. In the case where a component is failed during an early phase and is also failed in a later phase, the conditional probability of the component failing in the later phase given it was already failed is 1.0 (in other words, the later failure is a TRUE house event, or can simply be removed from the cut set leaving the initial failure event). In the case where a component is successful in an early phase but failed in a later phase, this cut set would be left as-is since this is a valid scenario. Lastly, in the case where a component is successful in an early phase and successful in a later phase, this cut set would be left as-is since this is also a valid scenario. Three different situations are used when evaluating the cut sets:

1. A combination of failure and success resulting in an impossible scenario
2. A combination of failures where basic event probabilities may need to be adjusted
3. A combination of successes and/or failures resulting in a valid scenario.

Complicating the cut set analysis will be the evaluation of sequences, where components may fail or succeed, depending on the particulars of the sequence. Processing success logic or evaluating the "delete term" method will require checks on Phased-Mission enabled events. Further complicating the cut set evaluation will be possible inclusion of recovery basic events and common-cause failure basic events.

To illustrate the processing steps that must be accomplished during the cut set generation, two example problems are described. The first uses a single component type and factors in possible recovery. The second uses two redundant components, includes potential recovery, and models common-cause failures. Note that SAPHIRE currently does not include any method to identify a specific non-recovery basic event to it applicable failure events.

12.1.3.1 Phase Analysis Case 1

For Case 1, a single component is called to function in two different parts of the Phased-Mission, both in Phase 1 and in Phase 2. The structure of the logic models used for this example is shown in Figure 23. In the fault trees, consider the component (A) recoverable in Phase 1 – thus an AND of the non-recovery probability (event NRA1) to the failure of A in the X fault tree. The failure of A is not recoverable in phase 2.

The cut sets that appear from the event tree sequences are:

Sequence 1:	$/A1 * /A2 + /A2 * /NRA1$	(two cut sets)
Sequence 2:	$A2 * /NRA1 + /A1 * A2$	(two cut sets)
Sequence 3:	$A1 * NRA1 * /A2$	(one cut set)
Sequence 4:	$A1 * NRA1 * A2$	(one cut set)

where “/” indicates the success of the event, “*” indicates AND operation, and “+” indicates OR operations.

Evaluate these cut sets considering that component **A** (described by **A1** and **A2**) is evaluated in different phases. Evaluate each cut set for the four sequences.

Sequence 1

$/A1 * /A2$

This cut set represent the success of A in both phases. Consequently, this is a valid cut set.

$/A2 * /NRA1$

This cut set represents the successful recovery of A in phase 1 and then success of A in phase 2. However, there is no successful recovery of A in phase 1 (in this cut set) since there is no failure of A (in phase 1) in the cut set. Consequently, this cut set is not valid and must be removed from the cut set list.

The resulting (correct) cut set for this sequence should be:

$/A1 * /A2$

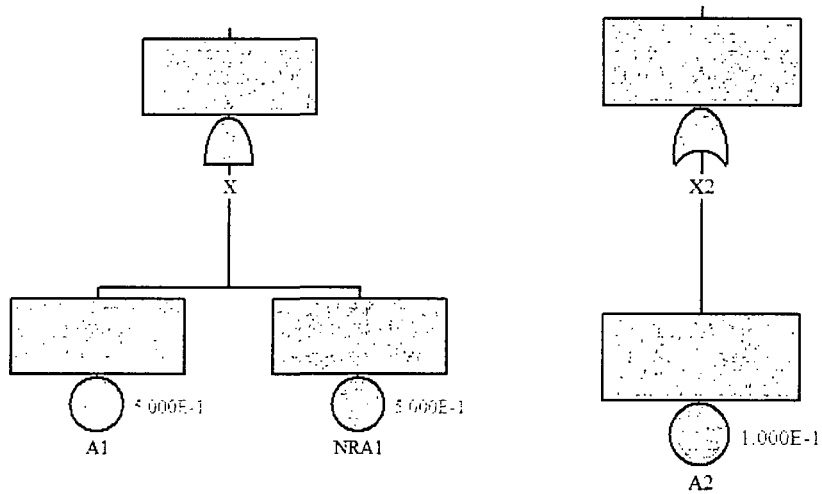
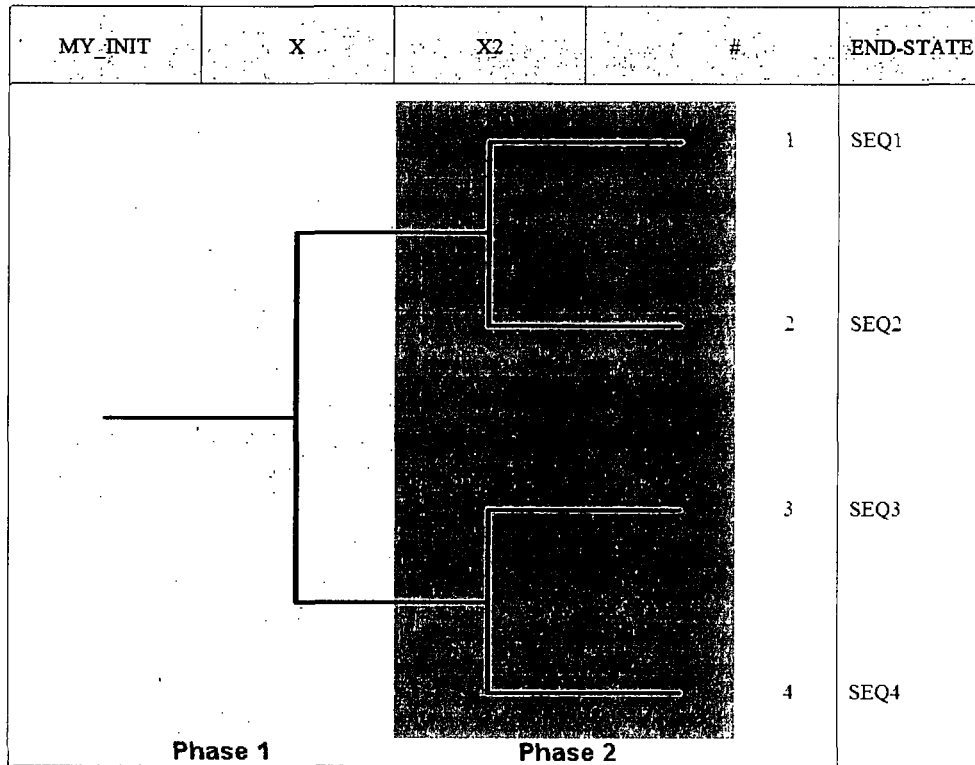


Figure 23. Event and fault trees used for Case 1 calculation

Sequence 2

$$A2 * /NRA1$$

This cut set represents the successful recovery of A in phase 1 and then failure of A in phase 2. However, there is no successful recovery of A in phase 1 since there is not a failure of A (in

phase 1) in the cut set. Consequently, this cut set is not valid and must be removed from the cut set list.

$/A1 * A2$

This cut set represents the success of A in phase 1 and the failure of A in phase 2. Consequently, this is a valid cut set.

The resulting (correct) cut set for this sequence should be:

$/A1 * A2$.

Sequence 3

$A1 * NRA1 * /A2$

This cut set represents the failure of A in phase 1, the failure to recover A in phase 1, and success of A in phase 2. However, there cannot exist component A both failed early in the accident scenario and successful later in the accident scenario [$P(/A2 | A1 * NRA1) = 0$]. Consequently, this cut set is not valid and must be removed from the cut set list. Note that this assumes that, in time, phase 1 precedes phase 2.

The resulting (correct) cut set for this sequence should be:

<FALSE> .

Sequence 4

$A1 * NRA1 * A2$

This cut set represents the failure of A in phase 1, the failure to recover A in phase 1, and failure of A in phase 2. However, if A is failed in an early phase, it is guaranteed failed in a later phase in the accident scenario [$P(A2 | A1 * NRA1) = 1$]. Consequently, this cut set is must be adjusted by removing the basic events representing failures of A in the later phase(s).

The resulting (correct) cut set for this sequence should be:

$A1 * NRA1$.

12.1.3.2 Phase Analysis Case 2

For Case 2, two components are called to function in two different parts of the Phased-Mission, both in Phase 1 and in Phase 2. The structure of the logic models used for this example is

shown in Figure 24. In the fault trees, we consider the components (**A** and **B**) recoverable in Phase 1 and Phase 2 – thus we AND the non-recovery probability (event **NRA1** and **NRA2**) to the component failure in the S (either S1 or S2) fault tree. In addition, the example considers the possibility for common-cause failure. Note though that SAPHIRE does not currently track common-cause failure adjustments through the phase-based cut sets.

The cut sets that appear from the event tree sequences are:

Sequence 1

/A2 * /CCF1 * /CCF2 * /NRB1 +
 /A2 * /CCF1 * /CCF2 * /NRA1 +
 /B2 * /CCF1 * /CCF2 * /NRB1 +
 /B2 * /CCF1 * /CCF2 * /NRA1 +
 /A2 * /B1 * /CCF1 * /CCF2 +
 /A1 * /B2 * /CCF1 * /CCF2 +
 /B1 * /B2 * /CCF1 * /CCF2 +
 /A1 * /A2 * /CCF1 * /CCF2 +
 /B1 * /CCF1 * /CCF2 * /NRB2 +
 /A1 * /CCF1 * /CCF2 * /NRB2 +
 /A1 * /CCF1 * /CCF2 * /NRA2 +
 /B1 * /CCF1 * /CCF2 * /NRA2 +
 /CCF1 * /CCF2 * /NRB1 * /NRB2 +
 /CCF1 * /CCF2 * /NRA1 * /NRB2 +
 /CCF1 * /CCF2 * /NRA2 * /NRB1 +
 /CCF1 * /CCF2 * /NRA1 * /NRA2

(16 cut sets)

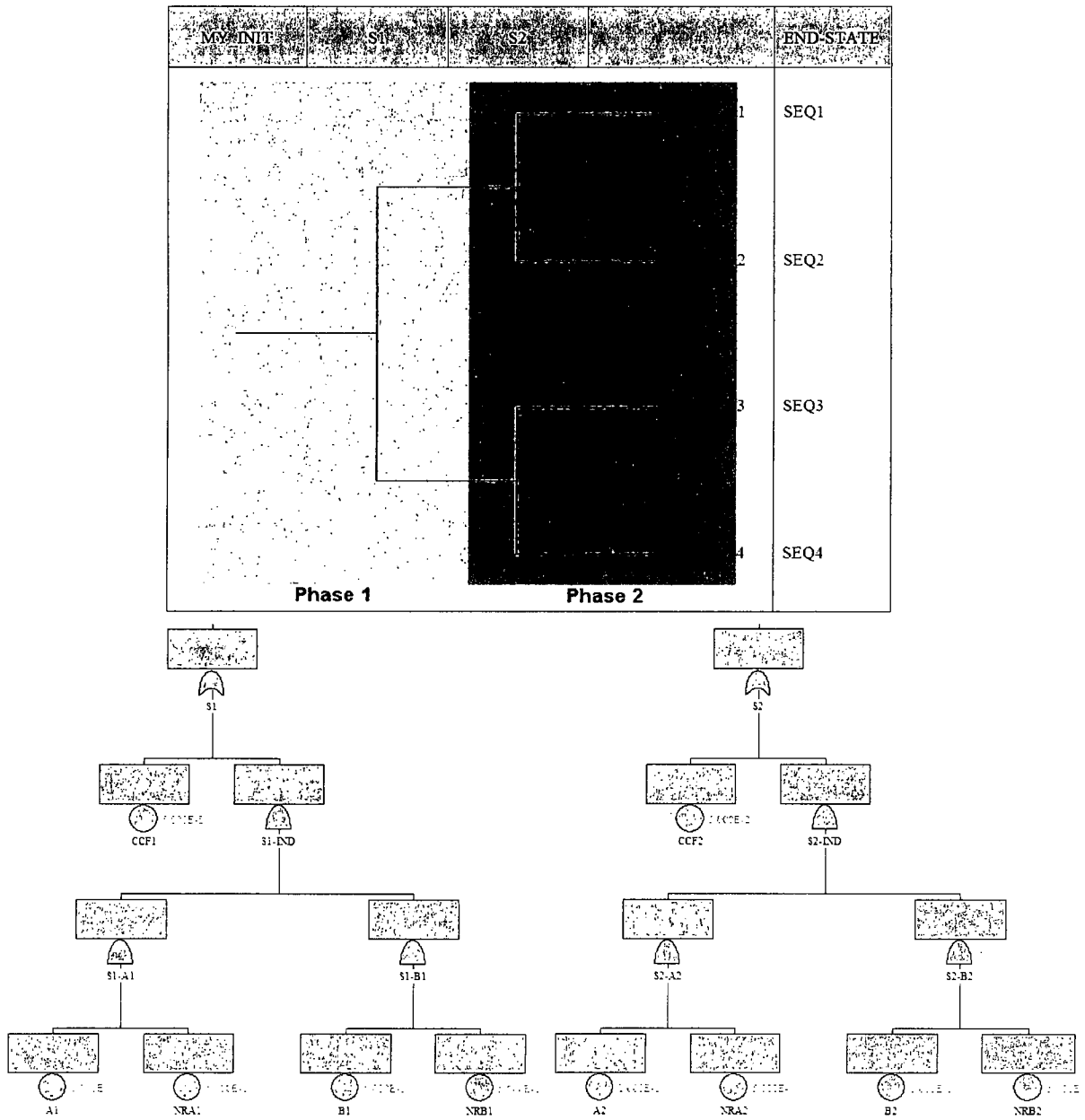


Figure 24. Event and Fault Trees used for Case 2 Calculation

Sequence 2

$/A1 * /CCF1 * CCF2 +$
 $/B1 * /CCF1 * CCF2 +$
 $/CCF1 * CCF2 * /NRA1 +$
 $/CCF1 * CCF2 * /NRB1 +$
 $/CCF1 * NRA2 * /NRB1 * NRB2 +$
 $A2 * /B1 * B2 * /CCF1 * NRA2 * NRB2 +$
 $/A1 * A2 * B2 * /CCF1 * NRA2 * NRB2 +$
 $A2 * B2 * /CCF1 * /NRA1 * NRA2 * NRB2$ (eight cut sets)

Sequence 3

$A1 * /A2 * B1 * /CCF2 * NRA1 * NRB1 +$
 $A1 * B1 * /B2 * /CCF2 * NRA1 * NRB1 +$
 $/B2 * CCF1 * /CCF2 +$
 $/A2 * CCF1 * /CCF2 +$
 $A1 * B1 * /CCF2 * NRA1 * /NRA2 * NRB1 +$
 $A1 * B1 * /CCF2 * NRA1 * NRB1 * /NRB2 +$
 $CCF1 * /CCF2 * /NRB2 +$
 $CCF1 * /CCF2 * /NRA2$ (eight cut sets)

Sequence 4

$A1 * B1 * CCF2 * NRA1 * NRB1 +$
 $CCF1 * CCF2 +$
 $A1 * A2 * B1 * B2 * NRA1 * NRA2 * NRB1 * NRB2 +$
 $A2 * B2 * CCF1 * NRA2 * NRB2$ (four cut sets)

where “/” indicates the success of the event, “*” indicates AND operation, and “+” indicates OR operations.

Evaluate these cut sets considering that component A (described by A1 and A2) and component B (described by B1 and B2) is evaluated in different phases.

Sequence 1

$/A2 * /B1 * /CCF1 * /CCF2$

This cut set represents the success of A in phase 2 and B in phase 1. Note that success of A implies that the CCF is successful in phase 1 (which is seen) and success of B implies that the CCF is successful in phase 2 (which is also seen). This is a valid cut set.

$/A2 * /CCF1 * /CCF2 * /NRA1$

This cut set represents the successful recovery of A in phase 1 and then success of A in phase 2. However, we cannot have a successful recovery of A in phase 1 (in this cut set) since we do not see a failure of A (in phase 1) in the cut set. Consequently, this cut set is not valid and must be removed from the cut set list.

The resulting (correct) cut set for this sequence should be:

$(/A2 * /B1 * /CCF1 * /CCF2) + (/A1 * /B2 * /CCF1 * /CCF2) + (/B1 * /B2 * /CCF1 * /CCF2) + (/A1 * /A2 * /CCF1 * /CCF2).$

Sequence 2

$/CCF1 * CCF2 * /NRA1$

This cut set represents the successful recovery of A in phase 1 and then failure of A and B in phase 2. However, we cannot have successful recovery of A in phase 1 since we do not see a failure of A (in phase 1) in the cut set. Consequently, this cut set is not valid and must be removed from the cut set list.

$/A1 * /CCF1 * CCF2$

This cut set represent the success of A in phase 1 and the failure of A and B in phase 2. Consequently, this is a valid cut set.

$/CCF1 * NRA2 * /NRB1 * NRB2$

This cut set represents the successful recovery of A in phase 2 and B in phase 2. However, we can not a successful recovery if we do not also see the failure of the corresponding component. Consequently, this cut set is not valid and must be removed from the cut set list.

$A2 * /B1 * B2 * /CCF1 * NRA2 * NRB2$

This cut set represents the success of B in phase 1 then the failure of both A and B (and the failure to recover both A and B) in phase 2. Consequently, this is a valid cut set.

The resulting (correct) cut set for this sequence should be:

$(/A1 * /CCF1 * CCF2) + (/B1 * /CCF1 * CCF2) + (A2 * /B1 * B2 * /CCF1 * NRA2 * NRB2) + (/A1 * A2 * B2 * /CCF1 * NRA2 * NRB2).$

Sequence 3

$A1 * /A2 * B1 * /CCF2 * NRA1 * NRB1$

This cut set represents the failure of A in phase 1, the failure to recover A in phase 1, and success of A in phase 2 (in addition to B failing in phase 1). However, component A both failed early in the accident scenario and is successful later in the accident scenario [$P(/A2 | A1 * NRA1) = 0$]. Consequently, this cut set is not valid and must be removed from the cut set list.

$/B2 * CCF1 * /CCF2$

This cut set represents the CCF of A and B in phase 1 and the success of B in phase 1. However, component B cannot be successful in a later phase once it is failed in an early phase. Consequently, this cut set is not valid and must be removed from the cut set list.

$CCF1 * /CCF2 * /NRA2$

This cut set represents the CCF of A and B in phase 1 and the success of either A or B in phase 2. However, A or B cannot be successful in a later phase once it is failed in an early phase. Consequently, this cut set is not valid and must be removed from the cut set list.

The resulting (correct) cut set for this sequence should be:

<FALSE>

Sequence 4

$A1 * B1 * CCF2 * NRA1 * NRB1$

This cut set represents the failure of A and B in phase 1 (and not recovered) and CCF in phase 2. However, if both A and B are failed in an early phase, they are guaranteed failed in a later phase in the accident scenario [$P(CCF2 | A1 * NRA1 * B1 * NRB1) = 1$]. Consequently, this cut set is must be adjusted by removing the basic events representing failures of A and B (CCF2) in the later phase(s).

$CCF1 * CCF2$

This cut set represents the CCF of A and B in phase 1 and in phase 2. However, $P(CCF2 | CCF1) = 1$. Consequently, use Check 3 to adjust this cut set.

$A2 * B2 * CCF1 * NRA2 * NRB2$

This cut set represents the CCF of A and B in phase 1 and failure of A and B in phase 2. However, failure of a component in an initial phase implies failure of that component in later phase(s). Consequently, use Check 3 to adjust this cut set.

The resulting (correct) cut set for this sequence should be:

$(A1 * B1 * NRA1 * NRB1) + (CCF1) + (A1 * B1 * NRA1 * NRA2 * NRB1 * NRB2)$

13. INTEGRATED MODEL SOLVING USING MODEL TYPES

Some risk analysis approaches require modeling failures due to causes other than “random” types of failures, such as fires, floods, or seismic events. Historically, model types other than “Random” were created using basic event transformations in SAPHIRE, and were run as separate models. Now, SAPHIRE Version 8 uses the Model Type definitions to create models such as “Random,” as well as other user-specified types.

In Version 8 a virtual basic event will allow the user to specify a basic event’s model type(s) as well as different reliability models for *each* applicable model type, and will display this information in one “virtual” basic event. In effect, SAPHIRE is extending a single basic event into multiple failure types, one for each Model Type. This capability allows for combinations of model types to be run at the same time, and presents cut sets which can contain mixtures of basic events of different model types (e.g., Fire + Random).

The solving of the fault trees and event trees is transparent to the user, which improves user checks of the logic and model solving. For example, in the expanded view of the fault tree, SAPHIRE will display the logical relation of system basic events which may be subject to random failure or other conditions as indicated by their respective Model Type.

If the Version 8 “Model Type” feature is used to build integrated models, then basic events following different initiators (as part of an accident sequence) may have different unreliability models and probabilities assigned. Consider, for example, a fire event tree which transfers to a loss of offsite power event tree. System failures in the loss of offsite power event tree could be due to fire-related only, random, or fire-related + random events. The random failure of a component could be given one type of unreliability model, and the fire-related failure of the same component could be assigned a different unreliability model. It may also be desired to model fire sequence-specific component failure probabilities by assigning different fault trees to different sequences. This approach would result in assigning sequence specific fire-related component failure probabilities prior to solving. Post-processing rules could also be used to change the basic event’s probability.

To evaluate models using the Version 8 Model Type feature, one of the options available during the “solve” step is to choose which Model Type(s) to be evaluated. For additional details on the solve option, refer to Volume 3 (Users’ Guide) of this NUREG.

14. POST-PROCESSING RULES

The SAPHIRE "post-processing rules" are textual logic rules that allow for the alteration or deletion of fault tree or sequence cut sets. Although previously called "recovery rules," the post-processing rules have evolved from the simple inclusion of recovery events into a powerful rule-based system for cut set manipulation. The post-processing rules can be used which for probabilistic risk assessment techniques include;

- The automated inclusion of sequence recovery events
- The inclusion of common-cause failure cut sets
- The elimination of mutually-exclusive events (e.g., impossible combinations of events).

The rules follow a format similar to the structure that is found in traditional programming languages (e.g., BASIC or PASCAL). As such, the ability exists to define "macros" and "if...then" type of structures. The rules may be developed for a particular fault tree, all fault trees, a particular sequence, a single event tree, or all sequences. In Table 9 we list the objects which may be manipulated by way of recovery rules.

Table 9. Applicable objects with post-processing rules.

Item	Menu	Name of post-processing rules
Specific fault tree	Fault Trees	Fault Trees Rules
All fault trees	Fault Trees	Project Rules
Specific sequence	Sequences	Sequence Rules
Single event tree	Sequences	Event Tree Rules
All sequences	Sequences	Project Rules

Post-processing rules follow a typical "if... then" type of structure. The rules are entered in a free-form text editor within SAPHIRE. However, the rules can be exported and loaded through MAR-D. Use of the recovery rules could result in non-minimal cut sets (for example, one could add event X to a cut sets already containing X. Consequently, the typical steps one would take in performing an analysis when using the recovery rules are

1. Solve fault tree or sequence cut sets
2. Apply post-processing rules to applicable fault trees or sequences
3. Perform a "Cut Set Update" to fault tree or sequence cut sets in order to remove any non-minimal cut sets from Step 2.
4. Display or report results

The recovery rule editor searches existing fault tree or sequence cut sets for cut sets matching the search criteria defined in the rule. The post-processing rule is used to modify the cut sets matching the search criteria. As such, a nomenclature has been defined to allow users to specify exactly what is to be done on the cut sets generated by SAPHIRE. The general symbols used in the post-processing rules include:

14.1.1.1 Symbols

	Denotes a comment line	~	Operator for or "not present"
*	Logical AND operator	+	Logical OR operator
/	Complement	()	Parentheses

Example of the search criteria (for basic events X, Y, and Z) are shown in Table 10.

Table 10. Examples of search criteria used in the SAPHIRE post-processing rules.

Search Criteria	Meaning of the Search Criteria
X	Basic event X appears in the cut set
~X	Basic event does not occur in the cut set
/X	Success of event X appears in the cut set
X * Y	Both events X and Y appear in the cut set
X + Y	Either event X or Y appear in the cut set
X*(Y + Z)	Either X and Y or X and Z appear in cut set in the cut set
~X*Y	Y does appear and X does not appear
always	Pre-defined macro-name means the criteria is always met.

To illustrate the use of post-processing rules, several examples are provided. Each example has comments to clarify what the rule does.

14.1.1.2 Post-processing Rule Structure (Example 1 – if-then)

| The "if-then" Rule Structure:
| This rule adds a recovery action BUSREC when electric bus B or C is failed if EL-BUS-B + EL-BUS-C then recovery = BUSREC;
| This keyword line must end with a semicolon.
endif

14.1.1.3 Post-processing Rule Structure (Example 2 – if-then-else)

| The "if-then-elsif" Structure:
| This rule deletes the cut set if both diesel generators are out for maintenance.
| If the two DGs fail randomly, add a common cause event.
if (DG-1-MAINT * DG-2-MAINT) then
DeleteRoot;
elsif (DG-1-RAND * DG-2-RAND) then
| Copy the original cut set, remove the two failure events, then add CC
CopyRoot;
DeleteEvent = DG-1-RAND;
DeleteEvent = DG-2-RAND;
AddEvent = DG-CCF-1AND2;
endif

14.1.1.4 Post-processing Rule Structure (Example 3 – appending recovery actions)

| The rule attaches the recovery action NRAC-12HR to every cut set for a particular sequence.
| This rule would be typed into the event tree sequence rule editor for the sequence of interest.
| A rule to apply NRAC-12HR recovery event to all cut sets in the sequence.
if always then
recovery = NRAC-12HR;
endif

14.1.1.5 Post-processing Rule Structure (Example 4 – mutually exclusive event removal)

```
| This rule could be placed in either (or both) the fault tree project rules or the event  
| tree project rules.  
| Define a macro to get those cut sets that have combinations of two motor driven  
| pumps out for maintenance.  
PUMPS-IN-MAINT = MDP-A-MAINT * MDP-B-MAINT;  
| Search for the maintenance events and then delete cut set.  
if PUMPS-IN-MAINT then  
    | Delete the cut set  
    DeleteRoot;  
endif
```

14.1.1.6 Post-processing Rule Structure (Example 5 – including common-cause failure events)

```
| The search criteria identifies the failure combination of two auxiliary feedwater pumps.  
| If these two basic events are found in a cut set then a new cut set will be created that  
| replaces the independent failures of the two pumps with a single common-cause basic event.  
| This rule could be placed in either (or both) the fault tree project rules or the event tree project  
| rules.  
| Define a macro to only pick up those cut sets that have  
| combinations of AFW-PUMP-A and AFW-PUMP-B.  
CCF-AFW-PUMPS = AFW-PUMP-A * AFW-PUMP-B;  
| Search for the AFW pump basic events and make a new  
| cut set with the CCF event.  
if CCF-AFW-PUMPS then  
    | First make a copy of the original cut set  
    CopyRoot;  
    | Now remove the two independent failure events  
    DeleteEvent = AFW-PUMP-A;  
    DeleteEvent = AFW-PUMP-B;  
    | Now add the CCF event  
    AddEvent = AFW-PUMP-CCF;  
endif
```

Lastly, we close this section by listing, in Table 11, all of the keywords available to the user when entering post-processing rules in SAPHIRE.

Table 11. Keywords utilized in the post-processing rule process in SAPHIRE.

Keyword or symbol	Definition	Usage
if then	Keyword that indicates a search criteria is being specified.	if "search criteria" then perform some action on each cut set; endif
endif	Keyword that indicates the end of a particular rule.	if "search criteria" then perform some action on each cut set; endif
else	Keyword that specifies some action to be taken if all the search criteria(s) are not met. The else should be the last condition in the recovery rule.	if "search criteria" then perform some action on each cut set; else perform some other action on each cut set if search criteria not met; endif
elsif	Keyword that specifies an alternative search criteria. Any number of elsifs can be used within a recovery rule.	if "search criteria" then perform some action on each cut set; elsif "2nd search criteria" then perform some other action on each cut set; elsif "3rd search criteria" then perform some other action on each cut set; endif
Always	Keyword that indicates that every cut set that is being evaluated satisfies the search criteria.	if always then perform some action on each cut set; endif
init()	Keyword used in the search criteria to indicate that a sequence cut set has a particular initiating event.	if init(INITIATOR-NAME) * "other search criteria if needed" then perform some action on each cut set; endif

Keyword or symbol	Definition	Usage
~	Symbol used in the search criteria to indicate that a particular event will not be in the cut set that is being evaluated.	<p>if (~SEARCH-CRITERIA) * "other search criteria if needed" then</p> <p>...</p> <p>The search criteria will be satisfied for all cut sets that do not contain SEARCH-CRITERIA (and also contains the optional "other search criteria"). SEARCH-CRITERIA may be either an initiating event, basic event, macro, or logic expression.</p>
/	Symbol used to represent a complemented event (i.e., the success of a failure basic event).	<p>if (/BASIC-EVENT) * "other search criteria" then</p> <p>The search criteria will be satisfied for all cut sets that contain the complement of BASIC-EVENT (and also contains the optional "other search criteria").</p>
	Symbol used to represent a comment contained in the rules. Everything on a line to the right of this symbol will be ignored by the rule compiler.	<p> Place your comments here!</p> <p> Note that blank lines are also permissible!</p>
;	Symbol to indicate the end of a macro line or a line that modifies the cut set being evaluated.	<p> usage for a macro command MACRO-NAME = "search criteria" ;</p> <p> usage for a cut set modification line recovery = RECOVERY-EVENT ;</p>
*	Symbol to indicate the logical AND command.	<p>if SEARCH-CRITERIA1 * SEARCH-CRITERIA2 then</p> <p>The search criteria will be satisfied for all cut sets that match SEARCH-CRITERIA1 and SEARCH-CRITERIA2. The SEARCH-CRITERIA# may be either an initiating event, basic event, macro, or logic expression.</p>

Keyword or symbol	Definition	Usage
+	Symbol to indicate the logical OR command.	if SEARCH-CRITERIA1 + SEARCH-CRITERIA2 then The search criteria will be satisfied for all cut sets that match either SEARCH-CRITERIA1 or SEARCH-CRITERIA2. The SEARCH-CRITERIA# may be either an initiating event, basic event, macro, or logic expression.
()	Symbols to indicate a specific grouping of items.	if (A + B) * (C + D) then The search criteria above would return all cut sets that contain: [A * C], [A * D], [B * C], or [B * D].
system()	Keyword used in the search criteria to indicate that a fault tree contributes to the existence of the cut set that is being evaluated.	if system(ECS) then perform some action on each cut set endif
Recovery =	Keyword that indicates that a recovery event is going to be added to the cut set being evaluated (SAPHIRE keeps record of all recovery events).	if "search criteria" then recovery = NAME-OF-RECOVERY; endif
AddEvent =	Keyword that indicates that an event will be added to the cut set being evaluated.	if "search criteria" then AddEvent = EVENT-NAME; Endif
DeleteEvent=	Keyword that indicates that an event will be deleted from the cut set being evaluated.	if "search criteria" then DeleteEvent = EVENT-NAME; Endif
NewCutset;	Keyword that indicates that a new, empty cut set will be added to the list of cut sets. This new cut set then becomes the cut set that is being evaluated.	if "search criteria" then NewCutset; now make additions to the empty cut set... endif
DeleteRoot;	Keyword that indicates that the original cut set (i.e., that cut set that satisfied the search criteria) will be deleted.	if "search criteria" then DeleteRoot; Endif

Keyword or symbol	Definition	Usage
CopyCutset;	Keyword that indicates that the cut set being evaluated will be copied and added to the list of cut sets. This copied cut set then becomes the cut set that is being evaluated.	if "search criteria" then CopyCutset; now make modification to a copy of the cut set... endif
CopyRoot;	Keyword that indicates that the original cut set (i.e., that cut set that satisfied the search criteria) will be copied. This copied cut set will then become the cut set that is being evaluated.	if "search criteria" then CopyRoot; now make modifications to a copy of the original cut set... endif
MACRO	A macro is a user-definable keyword that specifies a search criteria. The macro name must be all upper-case, must be 16 characters or less, and must not include any of the restricted characters (e.g., a space, *, ?, \, /). The macro line can wrap around to more than one line, but must end with a semicolon.	MACRO-NAME = SEARCH-CRITERIA; if MACRO-NAME "and other search criteria" then perform some action on each cut set...; endif Macros are only applicable in the particular rule they are entered into

15. DEFINING END STATES USING PARTITIONING RULES

In standard application of PRA models, accident sequences are defined via event tree logic structures. Each accident sequence indicates a string of events that, if they occur, may lead to an undesired outcome. For each of these sequences, different outcomes may be specified in order to group or analyze like outcomes (i.e., end state aggregation). In order to assist analysts in assigning end states to specific sequences, SAPHIRE includes the ability to define outcome states via programmatic rules. These rules, called "partition rules," are similar in structure to the "recovery rules" described in Section 12.

In general, the partitioning rule editor tests the existing sequence cut sets for the presence or absence of specific combinations of basic events or initiating events, and assigns characters in the end state name when the criteria are met. This allows end state names to be built as the rules are applied. For example, if we wanted to define a rule that looked for cut sets containing an initiating event called "LOSP" and assigned those cut sets to an end state called "ES_LOSP," we would construct a partition rule like:

```
If init(LOSP) then
  Partition = "ES_LOSP";
endif
```

SAPHIRE would then search through the list of cut sets (when the partition rules are applied) to find those that meet the search criteria. Each cut set may be assigned an end state, where the end state text is up to 24 characters long. Also, partition rules may be defined sequentially so that the first rule can set just a portion of the end state text (say, for example, the first two characters) while other later rules set other parts of the end state text (say, for example, the third and fourth characters).

Other examples of the partition rules are listed below.

Partition Rule Structure (Example 1 – if-then)

```
| The "if-then" Rule Structure:
| This rule adds -SBO as characters 4 through 7 of the end state name
| when both DG-A and DG-B are present in the cut sets.
| The ??? are placeholders in the end state name. (The end state name is initially blank.)
if DG-A * DG-B then
  partition = "???-SBO";
  endif
| Note that the partition statement must end with a semicolon.
| The end state name must be ≤ 24 characters.
| The end state characters are enclosed in quotation marks.
```

15.1.1.1 Partition Rule Structure (Example 2 – if-always)

| The "if-always" Rule Structure. This rule adds END as the first 3 characters in every cut set.
if always then partition = "END";
endif

15.1.1.2 Partition Rule Structure (Example 3 – if-then-elsif)

| The "if-then-elsif" Structure:
| This rule adds characters 4 through 7 to the end state name.
| When both DG-A and DG-B are failed, -SBO is added.
| When DG-A is failed (but not DG-B), characters -DGA are added.
| When DG-B is failed (but not DG-A), characters -DGB are added.
if DG-A * DG-B then
 partition = "???-SBO";
elsif DG-A then
 partition = "???-DGA";
elsif DG-B then
 partition = "???-DGB";
endif

15.1.1.3 Partition Rule Structure (Example 4 – Current Partition)

| The "Current Partition" Rule Structure:
| This rule uses end state created by a rule to gather multiple end states into single end state.
| This rule can use wild cards as part of its search criteria.
| The rule below first creates two end states, one containing cut sets with DG-A and one containing cut sets with DG-B.
| The rule then reassigns the end states for those cut sets containing
| both DG-A and DG-B to the end state "A-AND-B".
 if DG-A then
 partition = "A?";
 endif
 if DG-B then
 partition = "?B";
 endif
| Demonstrate use of "wild cards (cannot use *)" with CurrentPart() keyword.
 if CurrentPart(A?) + CurrentPart(?B) then
 partition = "A-AND-B";
 endif

15.1.1.4 Partition Rule Structure (Example 5 – Global Partition)

```
| The "GlobalPartition" Rule Structure:  
| This rule globally partitions all cut sets in a sequence to an end state.  
| This option is activated by using the keyword "GlobalPartition" instead  
| of the normal "partition" keyword.  
| This example "GlobalPartition" rule will gather all sequence cut sets that  
| pertain to specified sequence logic.  
| Cut sets will be put into an endstate called CD-SEQ2 if they are found in  
| sequences that contain the following sequence logic  
|     LOSP * ECS * /CCS.  
| Cut sets will be put into an endstate called CD-SEQ3 if they are found in  
| sequences that contain the following sequence logic  
|     LOSP * ECS * CCS.  
|if INIT(LOSP) * SYSTEM(ECS) * SYSTEM(/CCS) then  
|   GlobalPartition = "CD-SEQ2";  
|elseif INIT(LOSP) * SYSTEM(ECS) * SYSTEM(CCS) then  
|   GlobalPartition = "CD-SEQ3";  
|endif
```

To finish this section, we list all of the available keywords utilized in the partition rules in Table 12.

Table 12. Keywords utilized in the SAPHIRE end state partition rules.

Keyword or symbol	Definition	Usage
if then	Keyword that indicates a search criteria is being specified.	if "search criteria" then perform some action on each cut set; endif
endif	Keyword that indicates the end of a particular rule.	if "search criteria" then perform some action on each cut set; endif
else	Keyword that specifies some action to be taken if all the search criteria(s) are not met. The else should be the last condition in the recovery rule.	if "search criteria" then perform some action on each cut set; else perform action on each cut set if search criteria not met; endif
elsif	Keyword that specifies an alternative search criteria. Any number of elsifs can be used within a recovery rule.	if "search criteria" then perform some action on each cut set; elsif "2nd search criteria" then perform action on each cut set; elsif "3rd search criteria" then perform action on each cut set; endif
always	Keyword that indicates that every cut set that is being evaluated satisfies the search criteria.	if always then perform some action on each cut set; endif
INIT()	Keyword used in the search criteria to indicate that a sequence cut set has a particular initiating event.	if INIT(INITIATOR-NAME) * "other search criteria if needed" then perform some action on each cut set; endif
system()	Keyword used in the search criteria to indicate that the sequence logic contains the particular top event.	if system(TOP EVENT) * "other search criteria if needed" then perform action on each sequence; endif

Keyword or symbol	Definition	Usage
~	Symbol used in the search criteria to indicate that a particular event will not be in the cut set that is being evaluated.	if (~SEARCH-CRITERIA) * "other search criteria if needed" then ... The search criteria will be satisfied for all cut sets that do not contain SEARCH-CRITERIA (and also contains the optional "other search criteria").
/	Symbol used to represent a complemented event (i.e., the success of a system or basic event).	if (/BASIC-EVENT) then The search criteria will be satisfied for all cut sets that contain the complement of BASIC-EVENT.
;	Symbol to indicate the end of a macro line or a line that modifies the cut set being evaluated.	usage for a macro command MACRO-NAME = "search criteria" ; usage for a cut set modification line partition = ENDSTATE ;
*	Symbol to indicate the logical AND command.	if SEARCH-CRITERIA1 * SEARCH-CRITERIA2 then The search criteria will be satisfied for all cut sets that match SEARCH-CRITERIA1 and SEARCH-CRITERIA2.
+	Symbol to indicate the logical OR command.	if SEARCH-CRITERIA1 + SEARCH-CRITERIA2 then The search criteria will be satisfied for all cut sets that match either SEARCH-CRITERIA1 or SEARCH-CRITERIA2.
partition =	Keyword that indicates the end state characters for cut sets meeting the search criteria will be modified according to the text after the equal sign.	if "search criteria" then partition = "END_STATE_NAME"; endif

Keyword or symbol	Definition	Usage
CurrentPart()	Keyword that searches for cut sets that have already been assigned to the endstate indicated.	if CurrentPart(CORE-DAMAGE) then partition = "NEW-CORE-DAMAGE"; endif
GlobalPartition=	Keyword to indicate that all cut sets in a particular sequence will be assigned to the end state identified after the equal sign.	if "search criteria" then GlobalPartition = "MY-END-STATE"; Endif
transfer =	Keyword to indicate the event tree to be created and transferred to for the sequence meeting the search criteria. The sequence end state frequency will be used as the initiating event frequency for the new event tree.	if "search criteria" then GlobalPartition = "CORE-DAMAGE"; transfer = LEVEL-2-TREE; endif

16. WORKSPACE ANALYSIS ROUTINES

16.1 Multi-Pass Solution

In previous versions of SAPHIRE, the generation of minimal cut sets was a linear three-step process. First, one generated the failure data. Second, the minimal cut sets were generated. Third, any post-processing via “recovery rules” was performed.

This linear analysis process is prone to error and analysis limitations. For example, if a basic event is set to TRUE to indicate a failure, then any applicable recovery rules using that basic event will never be applied since the basic event does not appear in the cut sets.

In addition to the cut set solution methods described earlier in this report, SAPHIRE 8 has an improved analysis method described as “multi-pass.” The older method of analysis is called “single-pass.”

In SAPHIRE 8 for analysis that compares a nominal against a changed case, SAPHIRE uses the steps below for the “**single-pass**” solution:

1. Recalculate the “nominal case” to be able to determine the increase in risk.
 - a. Generate basic event data with no change sets marked. If the no test/maintenance (T&M) option is used, all T&M events will be set to zero.
 - b. Solve all sequence cut sets using a predetermined (but low) truncation.
 - c. Apply recovery rules to all sequence cut sets.
 - d. If identified in the project constants, perform a cut set update to ensure non-minimal cut sets are removed.
 - e. Re-quantify the cut sets.
 - f. Store the cut set results via a base case update.
2. Calculate the risk increase.
 - a. Generate basic event data with changes indicated by the analyst. If the no test/maintenance option is used, all T&M events will be set to zero.
 - b. Solve sequence cut sets using the same truncation for the “nominal” case.
 - c. Apply recovery rules to all sequence cut sets.
 - d. If identified in the project constants, perform a cut set update to ensure non-minimal cut sets are removed.
 - e. Re-quantify the cut sets.
 - f. Store the cut set results for use in determining the change.

In SAPHIRE 8 for analysis that compares a nominal against a changed case, SAPHIRE uses the steps below for the “**multi-pass**” solution:

1. Recalculate the “nominal case” to be able to determine the increase in risk.
 - a. Generate basic event data with no change sets marked. If the no test/maintenance option is used, all T&M events will be set to zero.
 - b. Solve all sequence cut sets using a predetermined (but low) truncation.

- c. Apply recovery rules to all sequence cut sets.
 - d. Perform a cut set update to ensure non-minimal cut sets are removed.
 - e. Re-quantify the cut sets to ensure they have the proper values.
 - f. Store the cut set results via a base case update.
2. Calculate the risk increase.
- a. Generate basic event data with changes indicated by the analyst and store these values for use in Step 2g. If the no test/maintenance option is used, all T&M events will be set to zero.
 - b. Change TRUE events to 1.0 events (to ensure application of recovery rules).
 - c. Solve sequence cut sets using the same truncation for the “nominal” case.
 - d. Apply recovery rules to all sequence cut sets.
 - e. Change events having a probability of 1.0 to TRUE events (to ensure that non-minimal cut sets will be removed during the next step).
 - f. Perform a cut set update to ensure non-minimal cut sets are removed.
 - g. Re-quantify the cut sets so they have the proper values stored from step 2a.
 - h. Store the cut set results for use in determining the change.

The user has the option to select either the single-pass option or multi-pass option. However, the single-pass option will only be displayed if indicated via the Project User Settings. Since the multi-pass option provides an improved analysis approach (over the older single-pass option), it is the default analysis method.

16.2 Events and Condition Assessment Uncertainty Analysis

Two general calculations are performed in the ECA Workspace, initiating event assessment and condition assessment. For the former, a single metric, the conditional core damage probability (CCDP), is calculated. This metric is used in an absolute sense in that its numerical value is not subtracted from a nominal value to obtain an increase in risk. However, the *condition assessment* calculation requires two metrics, a CCDP and a core damage probability (CDP). In this calculation, the risk increase (over the time of the condition) is calculated.

Historically, only a point estimate value was calculated for the condition assessment CCDP and CDP. These two values were then subtracted to find the risk increase. However, a variety of complications arise when this approach is used, specifically on the change in risk measure, or “event importance” (I_e) which is given by the relationship

$$\text{Importance}_{\text{event}} = \text{CCDP} - \text{CDP} = I_e$$

where: CCDP = the conditional core damage frequency (CCDF)* duration of the condition

$$\text{CDP} = \text{the core damage frequency (CDF)* duration of the condition}$$

The uncertainty results are based on sampling from the component's variability and then solving for the CCDF and CDF based on the sampled probability. SAPHIRE then subtracts CDF from CCDF to obtain a sampled Δ CCDF, which is multiplied by the duration. SAPHIRE stores this calculated value and continues this process for the number of samples noted. Once SAPHIRE has performed the number of specified samples, it orders the Δ CCDF values and pulls out the 5th, 50th, and 95th, then calculates the mean by summing up the Δ CCDFs and divides it by the number of samples.

The algorithm used to determine the uncertainty on the l_e using either Monte Carlo or Latin Hypercube sampling is given below:

1. Solve all cut sets (to the project truncation level) for all sequences. Two lists are solved and stored, one for the nominal CDP case and one for the condition CCDF case.
2. Create a list of all basic events appearing in cut sets for either the nominal or condition cut sets. There will be a total of J number of basic events.
3. Start the uncertainty sampling loop.

For I = 1 to N, where N is the total number of iterations specified by the user.

- a. Go through the list of J events. For each event, obtain a random sample. If events are correlated, obtain a single value for all events in that correlation group. If an event is modified for the condition case, it will appear in the list twice, once to be used for the nominal (CDP) case and once for the condition (CCDF) case, with its values set for the respective case.
- b. Calculate CDP_I , where $CDP = 1 - \exp(-\lambda_{CDF} * \text{Duration})$. The λ_{CDF} is the value of the core damage frequency from the nominal case cut sets quantified using the samples obtained in Step 3a.
- c. Calculate $CCDF_I$, where $CCDF = 1 - \exp(-\lambda_{CCDF} * \text{Duration})$. The λ_{CCDF} is the value of the core damage frequency from the condition case cut sets quantified using the samples obtained in Step 3a.
- d. Calculate l_e , where $l_e = CCDF_I - CDP_I$.
- e. Store l_e , $CCDF_I$, and CDP_I into their own respective array, where each array is N elements long.

Repeat the **For I** loop N times.

4. Once the **For I** loop is complete and the three results arrays (l_e , CCDF, and CDP) are populated, sort each array.
5. From each result array, determine the moments and percentiles.

The graph of the uncertainty analysis, as shown in Figure 25, is added to Version 8 reports in the ECA interface. It color codes the percentages of the probability distribution function between the orders of magnitudes.

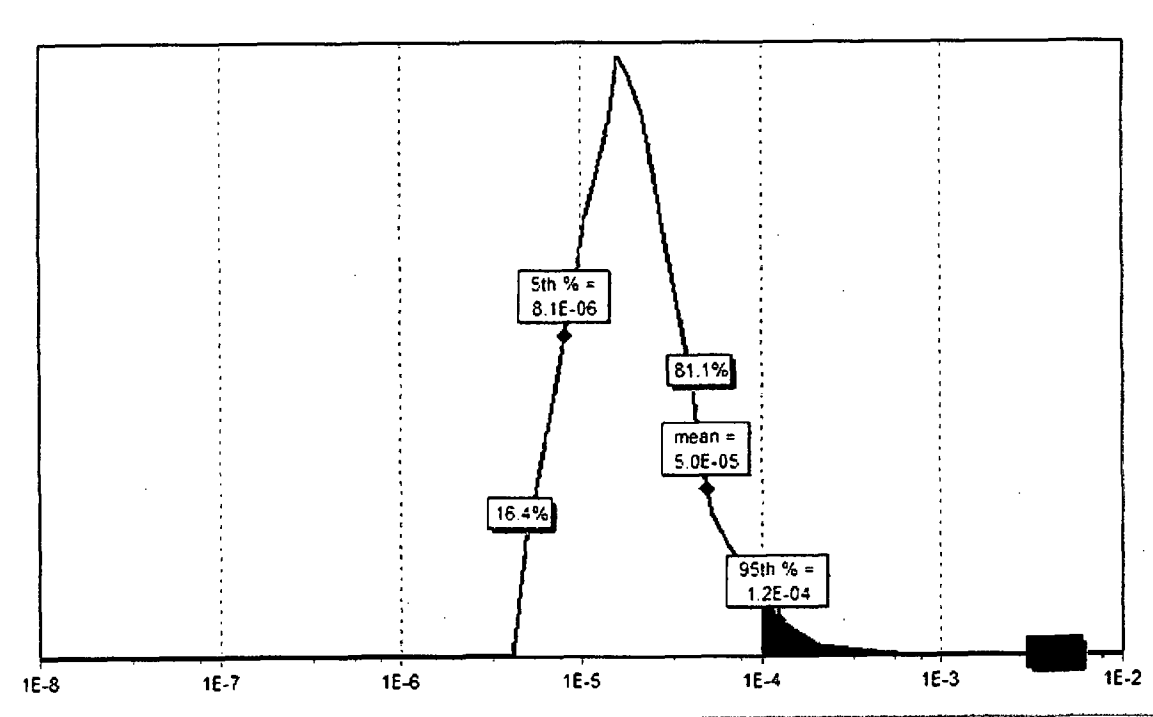


Figure 25. Graph of ECA uncertainty analysis

17. USING MACROS IN SAPHIRE

SAPHIRE version 8 is equipped with a feature that allows for the usage of macros to run specified menu or analysis steps in a medium outside of SAPHIRE. This feature is useful for tedious and repetitive analysis procedures in databases or several databases. Macros in SAPHIRE are different than the scripts used in the linking, recovery and partition rule editors. Those scripts are used to define a combination of rule based operands, systems, or basic events for linking event trees or adding basic event recovery actions or end state partitions. Conversely, macros are keywords in a text file format that SAPHIRE uses to automatically perform "analysis menu" functions. Though not directly designed to perform model modifications (i.e. modify basic event data, fault tree logic and event tree logic), most "modify" functions which can be used via macros are implemented by calling the MAR-D extract and load feature.

Macros are "programmed" using a standard text editor. The macros are coded with keywords that SAPHIRE recognizes as a specific menu or analysis functions. These keywords are listed in Table 13.

The embedded macro code uses tags, similar to those in an "HTML" or "XML" type file to run the actual associated macro. The tags are classified as verbs or "actions" which are the main actions or operations performed on a plant model; classes of "objects" which are acted upon by the verbs; and "parameters" which are passed to the actions. Each action, class, and parameter has opening and closing tags that mark the series of commands performed during a macro scenario. This code contains the declarations for all the tags used by the macro. A macro also contains functions and procedures that parse the steps into its actions and verbs; initializes the routines and parameters in the tool from the macro specified inputs; runs the code; and outputs results into a user-defined text file. The macros themselves can be run in normal or a "debug" mode. All the commands executed can be submitted into an output ASCII text log file. Results are output into a separate ASCII text file. Time, date and user identification information is logged along with the test results. Comment capability is available in the macro language.

To develop a macro, the "macro manager" should be access which provides a text editor (to create the macro) and a save option. Once a macro is saved using the macro manager, it is stored in the project's Shared\Macros folder. A macro can then be executed by selecting it in the list of macros by using the **File → Macro** option.

Table 13. Keywords for SAPHIRE Macros.

Keyword	Use/Example	Notes
<add>	<add> ... </add>	This keyword will add a new item to a list. Using this keyword, a basic event could be added to a MAR-D cut set listing or a change set added to a database.
<analysis type>	<analysis type>4 </analysis type>	This keyword is used to specify analysis type that SAPHIRE is to perform calculation in. Random = 1, Fire = 2, Flood = 3, Seismic = 4, etc. For this example the analysis type would be "seismic".
<analysis>	<analysis>random </analysis>	This keyword is used to specify specific analysis type during a base case update. For this example, during the base case update, the analysis type would be "random."
<base case update>	<base case update> (analysis type specified) </base case update>	This keyword is used to specify the start and end a base case update action.
<basic event>	<basic event> ... </basic event>	This keyword is used to specify the start and end of a basic event action. Generally, basic event actions involves extracting or loading MAR-D files or adding/deleting a basic event from a MAR-D file.
<calc type>	<calc type>1</calc type>	This keyword is used to specify the calculation type. For this example, calculation type 1 would be use in the change set.
<case>	<case>base</case>	This keyword is used to specify the base or current case. For this example, the base case would be invoked.
<change set>	<change set> ... </change set>	This keyword is used to specify the start and end of change set operations. For example, a change set can be added, deleted, marked or unmarked.
<class>	<class> ... </class>	This keyword is used to specify the start and end of an event class operation (i.e., class type basic event change in a change set). Within this keyword function, the class of the basic events to be changed and their desired changes is specified.
<comment>	<comment> information </comment>	This keyword is used to specify the start and end of a macro comment block. Any text can be located within the comment block.

Keyword	Use/Example	Notes
<compare file>	<pre><compare file> <input 1>FILE1 </input1> <input 2>FILE2 </input2> </compare file></pre>	This keyword is used to specify the start and end of a file comparison. Used with <input 1>, <input 2> tags to specify the two files to compare (report files). The results of the comparison can be viewed in a file using the <output>filename</output> keyword.
<condition>	<pre><condition> ... </condition></pre>	This keyword is used to specify the start and end of a selected GEM condition assessment process. Within the condition block, a GEM condition assessment can be added, deleted, marked or unmarked.
<delete>	<pre><delete> ... </delete></pre>	This keyword is used to specify the start and end of a list item to be deleted. Using this keyword, a basic event could be deleted to a MAR-D cut set listing and then reloaded or a change set deleted from a database.
<description>	<pre><description> text description </description></pre>	This keyword is used to specify a text description to change set, flag set, or scenario. For example if a change set is created, <description>Class change - All events</description>, the text between the description block would be added as the change set's description.
<duration>	<pre><duration>72 </duration></pre>	This keyword is used to specify a duration time for a GEM condition assessment. For this example, the duration time would be 72 hours.
<end state>	<pre><end state> ... </end state></pre>	This keyword is used to specify the start and end of an end state operation. Within the end state keyword blocks, end states can be marked, gathered, solved, and an uncertainty analysis performed.
<end>	<pre><end> </end></pre>	This keyword is used to specify the end of a scenario.
<event name>	<pre><event name> BE NAME </event name></pre>	This keyword is used to specify an event name to be adjusted during a change set operation. For this example, the basic event BE NAME, could be adjusted within a change set block.
<event tree>	<pre><event tree> ...</pre>	This keyword is used to specify the start and end of an event tree operation. Within the event tree keyword block, an

Keyword	Use/Example	Notes
	<code></event tree></code>	event tree can be marked, unmarked, and linked.
<code><exclude></code>	<code><exclude> FT-1 </exclude></code>	This keyword is used to specify the name of a event tree, fault tree, or sequence to be excluded in an operation. For this example, fault tree FT-1, would be excluded from any operation.
<code><extract></code>	<code><extract> . . . </extract></code>	This keyword is used to specify the start and end of a MAR-D extract operation. Within the extract keyword block, the MAR-D file type and file name is specified.
<code><fault tree></code>	<code><fault tree> . . . </fault tree></code>	This keyword is used to specify the start and end of a fault tree operation. Within the fault tree keyword block, cut sets can be solved, flag sets applied, uncertainty analysis and importance measures can be performed.
<code><file name></code>	<code><file name> TEMP.BEI </file name></code>	This keyword is used to specify a file name for MAR-D extract and load functions. For this example, the file name is MAR-D file with a name "TEMP.BEI". This keyword is also used to specify report output files names.
<code><flag set></code>	<code><flag set> . . . </flag set></code>	This keyword is used to specify the start and end of flag set operations. For example, a flag set can be added, deleted, and applied to fault trees and sequences.
<code><gather method></code>	<code><gather method>by sequence</gather method></code>	This keyword is used to specify the specific end state gather operation. For this example, the cut sets would be gathered "by sequence."
<code><generate></code>	<code><generate> </generate></code>	This keyword is used to specify the generate option. The generate command implements the changes in a change set or flag set and moves from the base case to the current case for an analysis.
<code><group></code>	<code><group>yes</group ></code>	This keyword is used to specify the use of group importance measures or not. For this example, the group importance is "on".
<code><ie name></code>	<code><ie name>IE-LOSP </ie name></code>	This keyword is used to specify the initiating event name. For this example, the initiating event "IE-LOSP" would be selected during a GEM initiating event assessment.
<code><importance></code>	<code><importance> . . . </importance></code>	This keyword is used to specify the start and end of an importance measure operation.

Keyword	Use/Example	Notes
<include>	<include>FT-1 </include>	A keyword used to specify the name of a event tree, fault tree, or sequence to be included in an operation. For this example, fault tree FT-1, would be included in the operation.
<init event>	<init event> . . . </init event>	This keyword is used to specify the start and end of a GEM initiating event assessment..
<initial prompt>	<initial prompt> yes </initial prompt>	This keyword is not necessary to run a macro. However, if “yes” is specified (as in this example) in the initial prompt block, a confirmation box is displayed that ask the analyst do confirm the operation of the macro-script.
<input 1>	<input 1> FILENAME1</input 1>	This keyword is used to designate the first file in a compare operation. For this example, the first file is called FILENAME1.
<input 2>	<input 2> FILENAME2</input2 >	This keyword is used to designate the second file in a compare operation. For this example, the second file is called FILENAME2.
<lambda>	<lambda> 1.0E-2 </lambda>	This keyword is used to specify the value. For this example, the lambda value would be 1.0E-2.
<link>	<link> </link>	This keyword is used to link event trees.
<load>	<load> . . . </load>	This keyword is used to specify the start and end of a MAR-D load operation. This operation can load in additional MAR-D files or files that have been modified
<logic show save>	<logic show save> </logic show save>	This keyword is used to show and save fault tree logic .
<loop class>	<loop class> grid </loop class>	This keyword is used to specify the type of loop classification being used during an analysis. For this example, the loop class would be “Grid-related”.
<mark event tree mask>	<mark event tree mask> LOSP </mark event tree mask>	This keyword is used to specify the specific event tree mask and is used during sequence analysis. For this example, all sequences generated with the event tree LOSP would be marked.
<mark logic fault tree>	<mark logic fault tree>*</mark logic fault tree>	This keyword is used to select a specific sequence having a specific fault tree logic during sequence analysis.

Keyword	Use/Example	Notes
<mark mask>	<mark mask> * </mark mask>	This keyword is used to specify the specific fault trees or event trees to mark. For this example, a "*" would mark all fault or event trees in the list (depending on if you were under a <fault tree> or <event tree> function block).
<mark name>	<mark name> NAME1</mark name>	This keyword is used to specify the specific condition name in GEM. For this example, the condition assessment NAME1 would be marked.
<mark sequence mask>	<mark sequence mask> ATWS </mark sequence mask>	This keyword is used to specify the specific sequence mask. For this example, all ATWS sequences would be selected for analysis.
<mark sequence name>	<mark sequence name> LOSP2 </mark sequence name>	This keyword is used to mark a specific sequence for sequence analysis. For this example, the sequence named LOSP2 would be marked..
<mask operation>	<mask operation>and </mask operation>	This keyword is used to specify the specific mask operation for masking/marking sequences. The logic mask operator can either be "OR" or "AND". This function block allows more flexibility in selecting event tree names, sequence names and fault tree names for sequence analysis.
<mask>	<mask> </mask>	This keyword is used in conjunction with others.
<method>	<method> mcs </method>	This keyword is used to specify the specific uncertainty method. For this example, the uncertainty method would be Monte Carlo. For Latin Hypercube, "lhs" would be used.
<mission time>	<mission time> 10 </mission time>	This keyword is used to specify the specific mission time value. Mission time can be in scientific notation i.e. <mission time>1.0E+1</mission time>. For this example the mission time would be 10 hours.
<name>	<name> test name </name>	This keyword is used to specify the name of a specific test passed in by a DOS script file, i.e. <name>%P-11</name>. It is also used to specify the name of a change set, flag set, fault tree, event tree, etc to be added or deleted during an operation.
<output>	<output> file name	This keyword is used to specify the output to a specific file name. For example, <output>compare.rpt</output>, would

Keyword	Use/Example	Notes
	<code></output></code>	output to a text file called "compare.rpt".
<code><page></code>	<code><page> </page></code>	This keyword is used to turn on the MAR-D paging option.
<code><partition></code>	<code><partition> </partition></code>	This keyword is used to turn on the partition option for event trees.
<code><probability></code>	<code><probability> 0.02 </probability></code>	This keyword is used to specify a specific probability. For this example the probability of 0.02 would be used in a change set.
<code><process></code>	<code><process> </process></code>	This keyword is used to specify the start and end of a condition assessment timed test.
<code><program exit></code>	<code><program exit> </program exit></code>	This keyword is used to specify the end of the macro script.
<code><project></code>	<code><project> </project></code>	This keyword is used to specify a "project level" uncertainty operation.
<code><recover></code>	<code><recover> </recover></code>	This keyword is used to start the cut set recovery process.
<code><report destination></code>	<code><report destination>F </report destination></code>	This keyword is used to specify the destination of the report output. There are three options: "F" is a file, "V" is a screen view, and "P" is a default printer. If the report destination is not specified, the default is to a file.
<code><report format></code>	<code><report format>A </report format></code>	This keyword is used to specify the type of report format that is outputted. There are three options: "A" is text file, "R" is RTF format, and "H" is HTML format. The default is text format.
<code><report></code>	<code><report> .. </report></code>	This keyword is used to specify a report creating operation.
<code><sample></code>	<code><sample> 5000 </sample></code>	This keyword is used to specify the number of samples in an uncertainty calculation. For this example, 5000 samples would be ran for the uncertainty operation.
<code><scenario></code>	<code><scenario> ... </scenario></code>	This keyword is used to specify a specific scenario sequence of events. This keyword is used to help identify various macro-script functions in a file.
<code><seed></code>	<code><seed>1234</seed></code>	This keyword is used to specify the random number seed in an uncertainty measure calculation. If a "0" is used, then the

Keyword	Use/Example	Notes
		clock number is implemented. For this example, the seed number would be "1234".
<seismic bin>	<seismic bin> 9 </seismic bin>	This keyword is used to specify the histogram bin number for a seismic analysis.
<sequence>	<sequence> ... </sequence>	This keyword is used to specify the start and end of a sequence operation.
<set units>	<set units> PPerYear </set units>	This keyword is used to specify the project units. For this example "per year" would be used. Other units include per month, per week, and per day.
<setup>	<setup> ... </setup>	This keyword is used to specify the test scenario state of operation. This keyword is used to during the validation and verification macro-script tests.
<show save>	<show save></show save>	This keyword displays a fault tree graphic and saves it.
<single>	<single> </single>	This keyword is used to specify a single event operation in a change set operation. Two types of change sets are available, single and class changes.
<slice save>	<slice save> ... </slice save>	This keyword is used to specify a cut set slice save operation. This feature allow for specific cut sets to be saved in a file (or end state).
<solve>	<solve> ... </solve>	This keyword is used to specify a fault tree/sequence solve operation (i.e. solve for cut sets).
<sort order>	<sort order> name </sort order>	This keyword is used to specify the type of sorted order of the output. This is used for sorting using a specified importance measure. This function sorts the basic events in alphabetical order.
<start>	<start> </start>	This keyword is used to specify the start of a new validation and verification test.
<sub type>	<sub type>base only </sub type>	This keyword is used to specify a report sub-type. For this example, the sub-type report is for base case results only.

Keyword	Use/Example	Notes
<suscept>	<suscept> 4 </suscept>	This keyword is used to specify an event susceptibility value. 1=random, 2=fire, 3=flood, 4=seismic, etc.
<tau>	<tau>4</tau>	This keyword is used to specify a specific tau value in a change set.
<truncation>	<truncation> 1.0E-12 </truncation>	This keyword is used to specify a probability cut off level, a truncation. This function block is used within the <solve> block.
<type>	<type>cut set</type>	This keyword is used to specify the report type. For this example, the report type would be for a cut set output report.
<uncertainty>	<uncertainty> </uncertainty>	This keyword is used to specify an uncertainty measure operation.
<unmark>	<unmark></unmark>	This keyword is used to clear any marked change sets in a plant model prior to additional analysis.
<update>	<update></update>	This keyword is used to specify an update operation for a cut set.
<verbose>	<verbose> </verbose>	This keyword is used to specify the verbose option on for the verification and validation test procedure of SAPHIRE. This keyword is used to setup the level of detail in the test results output files.
affected	<sub type> affected </sub type>	Parameter setting for a change set report that shows affected events adjusted in the change set (i.e. shows current case data)
all	<sub type> all </sub type>	Parameter setting for a change set report that shows both affected and unaffected basic events (i.e. shows both current and base case).
and	<mask operation>and </mask operation>	Parameter setting for setting for filtering masking/marking selections for fault trees and sequences.
ascii	<report format> ascii </report format>	Parameter setting for setting the report file out format to delimited ascii text.

Keyword	Use/Example	Notes
average	<loop class> average </loop class>	Parameter setting for use with GEM initiating event operations. Set the loop class to be a combined average of Grid related, Plant centered, severe weather, and extremely severe weather classes.
base	<case>base</case>	Parameter setting to specify the base case is to be used.
by partition	<gather method> by partition </gather method>	Parameter setting for selection of an end state gather technique to be "gather cut sets by partition".
by sequence	<gather method>by sequence</gather method>	Parameter setting for selection of an end state gather technique to be "gather cut sets by sequence".
combination	<type> combination </type>	Parameter setting to specify a combination report type (summary report) for fault tree, sequences and end states reports.
current	<case>current</cas e>	Parameter setting to specify the current case is to be used.
current base	<sub type>current base</sub type>	Parameter setting for selection of a report sub type to have output for both the current case and the base case.
current only	<sub type>current only</sub type>	Parameter setting for selection of a report sub type to be current case only.
cut set	<type>cut set</type>	Parameter setting for selection of a report type to be for "cut sets." i.e. the output file will contain cut sets.
debug		Parameter setting for selecting the mode of the log file output used for validation and verification testing.
detail		Parameter setting for selecting the mode of the log file output in a high detail mode.
esw	<loop class> esw </loop class>	Parameter setting for use with GEM initiating event operations. Sets the loop class to "extreme severe weather."
file		Parameter setting for TBD
grid	<loop class> grid	Parameter setting for use with Initiating Event operations.

Keyword	Use/Example	Notes
	<code></loop class></code>	Set the loop class to "grid related."
group	<code><type>group</type></code>	Parameter setting for selection of an importance measure calculation type "group."
interval	<code><type>interval</type></code>	Setting for selection of an importance measure calculation, "interval." The other option is "ratio".
lhs	<code><method>lhs</method></code>	Parameter setting for Latin Hypercube Simulation method in an uncertainty calculation.
mcs	<code><method>mcs</method></code>	Parameter setting for Monte Carlo Simulation method in an uncertainty calculation.
name	<code><name>%P-10</name></code>	Parameter setting for assigning the name of the database (or project) that the macro-script is being used on to the test scenario. For this example, if the project name was "TEST", then the name of the scenario would be "TEST-10".
no	<code><initial prompt>no</initial prompt></code>	Parameter setting for turning off a feature. For this example, the conformation prompt would not be implemented.
none		Parameter setting for TBD
or	<code><mask operation>or</mask operation></code>	Parameter setting for setting for filtering masking/marking selections for fault trees and sequences.
PPerDay	<code><set units> PPerDay</set units></code>	Parameter setting for <code><set units></code> to be "per day."
PPerDemand	<code><set units> PPerDemand</set units></code>	Parameter setting for <code><set units></code> to be "per demand."
PPerHour	<code><set units> PPerHour</set units></code>	Parameter setting for <code><set units></code> to be "per hour."
PPerMinute	<code><set units> PPerMinute</set units></code>	Parameter setting for <code><set units></code> to be "per minute."
PPerMonth	<code><set units> PPerMonth</set units></code>	Parameter setting for <code><set units></code> to be "per month."

Keyword	Use/Example	Notes
PPerWeek	<set units> PPerWeek </set units>	Parameter setting for <set units> to be "per week."
PPerYear	<set units> PPerYear </set units>	Parameter setting for <set units> to be "per year."
plant centered	<loop class>plant centered</loop class>	Parameter setting for use with an Initiating Event operation. Sets the loop class to "plant centered."
print	<report destination> print </report destination>	Parameter setting for selecting a report output option to be to a system default printer.
project	<type>project</type >	Parameter setting for selection of an uncertainty calculation type to "project".
random	<analysis>random</ analysis>	Parameter setting for an analysis type to random.
ratio	<type>ratio</type>	Parameter setting for selection of an importance measure calculation to be "ratio". The other option is "interval"
results	<type>results</type >	Parameter setting for selection of a report type for analysis results.
rtf	<report format> rtf </report format>	Parameter setting for setting the file output format of a report to be RTF format.
severe weather	<loop class>severe weather</loop class>	Parameter setting for use with initiating event operations. Set loop class to severe weather."
single	<type>single</type>	Parameter setting for selection of an importance measure calculation to single.
unaffected	<sub type> unaffected </sub type>	Parameter setting for a change set report that shows unaffected events not adjusted in the change set (i.e. shows base case data)
uncertainty	<type>uncertainty </type>	Parameter setting for selection of an importance measure calculation to be by "uncertainty."

Keyword	Use/Example	Notes
view	<pre><report destination> view </report destination></pre>	Parameter setting for selecting a report output option to be to screen for viewing.
yes	<pre><group>yes</group ></pre>	Parameter setting for turning on a feature.

The macro shown in Table 14 is designed for the DEMO project that accompanies the SAPHIRE program. The macro can be created using any standard text editor. The following macro, when run, creates a change set called "Example" and adds a description to it. The change set adjusts the probability of basic event C-PUMP-A to 1.0E-1. The macro then marks the change set and generates the new data. Next, all the sequences are marked and new cut sets are solved. Finally, the new cut set listing is displayed on the screen.

Table 14. Example macro using the Demo project.

<comment>

The following is an example SCRIPT that can be run in the DEMO project of SAPHIRE.
This simple SCRIPT performs the following functions automatically when implemented using the "Run Macro..." command:

Creates a Change Set named "EXAMPLE".

Adds a description to the change set.

Changes the probability of basic event C-PUMP-A to 1.0E-1.

Generates the new data.

Marks all Sequences.

Solves for cuts sets.

Generates a report of the new cut sets to the screen.

</comment>

<change set>

<add>

<name>EXAMPLE</name>

<description>Changes C-PUMP-A probability to 1.0E-1</description>

<single>

<event name>C-PUMP-A</event name>

<probability>1.0E-1</probability>

</single>

</add>

<mark name>EXAMPLE</mark name>

<generate></generate>

</change set>

```
<sequence>
  <unmark></unmark>
  <include>
    <mark event tree mask>*</mark event tree mask>
    <mask operation>or</mask operation>
    <mark sequence mask>*</mark sequence mask>
    <mask operation>or</mask operation>
    <mark logic fault tree>*</mark logic fault tree>
  </include>
  <solve></solve>
  <report>
    <type>cut set</type>
    <report destination>view</report destination>
  </report>
</sequence>
```


18. REFERENCES

- Ahrens, J. H., and U. Dieter, 1974, "Computer Methods for Sampling from Gamma, Beta, Poisson and Binomial Distributions," *Computing*, 12, pp. 223-246.
- Andrews, J. and T. Moss, 2002, *Reliability and Risk Assessment*, ASME Press, Second Edition.
- Apostolakis, G. and S. Kaplan, 1981, "Pitfalls in Risk Calculations," *Reliability Engineering*, 2, pp 135-145.
- Barlow, R. P. and F. Proschan, 1981, *Statistical Theory of Reliability and Life Testing*, Silver Springs, MD, p. 32.
- Beasley, J. D. and S. G. Springer, 1977, Algorithm AS 111, The Percentage Points of the Normal Distribution," *Applied Statistics*, 26, pp. 118-120.
- Bedford, T. and R. Cooke, 2001, *Probabilistic Risk Analysis: Foundations and Methods*, Cambridge University Press.
- Bohn, M. P., T. A. Wheeler, G. W. Parry, 1988, *Approaches to Uncertainty in Probabilistic Risk Assessment*, NUREG/CR-4836, January 1988.
- Butler, E. L., 1970, "Algorithm 370: General Random Number Generator", *Communications of the ACM*, 13, pp. 49-52.
- Cheng, R. C. H., 1977, "The Generation of Gamma Variables with Non-integral Shape Parameter," *Applied Statistics*, 26, pp. 71-75.
- Corynen, G. C., 1988, *A Fast Bottom up Algorithm for Computing the Cutsets of Non-Coherent Fault Trees*, NUREG/CR-5242, October 1988.
- Eide, S. A., et al., 2005, "Reevaluation of Station Blackout Risk at Nuclear Power Plants," NUREG/CR-6890, Vol.1, November 2005.
- Fishman, G. S. and Moore, L. R., 1982, "A Statistical Evaluation of Multiplicative Congruential Generators with Modulus (231 - 1)," *Journal of the American Statistical Association*, 77, 1 29-136
- Fussell, J. B., 1975, "How to Hand Calculate System Reliability and Safety Characteristics," *IEEE Transactions on Reliability*, R-24, 3, August 1975.
- Gertman, D., Blackman, H., Marble, J., Byers, J., Haney, L., Smith, C., *The SPAR-H Human Reliability Analysis Method*, INEEL/EXT-02-01307, 2004.

Hahn, G. J. and S. S. Shapiro, 1967, *Statistical Models in Engineering*, New York: John Wiley & Sons.

Henley, E. J. and H. Kumamoto, 1981, *Reliability Engineering and Risk Assessment*, Prentice-Hall.

Henley E. and H. Kumamoto, 1985, *Designing for Reliability and Safety Control*, Prentice-Hall, Inc.

Hickman, J. W., 1983, *PRA Procedures Guide: A Guide to the Performance of Probabilistic Risk Assessments for Nuclear Power Plants*, American Nuclear Society and Institute of Electrical and Electronic Engineers, NUREG/CR-2300, Volumes 1 and 2, January.

Jaynes, E., 2003, *Probability Theory*, Cambridge University Press.

Johnson, Norman L. and Samuel Kotz, 1970, *Continuous Distributions - 2*, New York: John Wiley & Sons.

Kennedy, R. P. et al., 1980, "Probabilistic Seismic Safety Study of an Existing Nuclear Power Plant," *Nuclear Engineering and Design*, 59, pp. 315-337.

Kennedy, William J., Jr. and James E. Gentle, 1980, *Statistical Computing*, New York: Marcel Dekker, Inc.

Lambert, H. E., "Measure of Importance of Events and Cut Sets in Fault Trees," *Reliability and Fault Tree Analysis*, SIAM, 1975.

Lee, C. Y. Representation of Switching Circuits by Binary Decision Diagrams, *Bell Systems Technology Journal*, 1959, Vol. 38, pp. 985-999.

Lindley, D. V., 1985, *Making Decisions*, New York: Wiley.

McGrath, E. J. and D. C. Irving, 1975, "Variance Reduction," *Techniques for Efficient Monte Carlo Simulation, III*, ORNL-RSIC-38, April.

McKay, M. K, N. L. Skinner, S. T. Wood, 1994, *Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Version 5.0, Fault Tree, Event Tree, and Piping & Instrumentation Diagram (FEP) Editors Reference Manual*, NUREG/CR-6116, Vol. 7, July 1994.

Mood, A. M., F. A. Graybill, D. C. Boes, 1974, *Introduction to the Theory of Statistics*, Third Edition, New York: McGraw-Hill.

NASA, 2009, *Bayesian Inference for NASA Probabilistic Risk and Reliability Analysis*, NASA/SP-2009-569, June.

NRC (U. S. Nuclear Regulatory Commission), 1975, *Reactor Safety Study--An Assessment of Accident Risks in U.S. Commercial Nuclear Power Plants*, WASH-1400 (NUREG/75-014), October 1975.

Park, Stephen K. and Keith W. Miller, 1988, "Random Number Generators: Good Ones Are Hard to Find," *Communications of the ACM*, 31, October 1988, pp. 1192-1201.

Patenaude, C. J., 1987, *SIGPI: A User's Manual for Fast Computation of Probabilistic Performance of Complex Systems*, NUREG/CR-4800, Lawrence Livermore National Laboratory, May 1987.

Press, S., 1989, *Bayesian Statistics: Principles, Models, and Applications*, New York: John Wiley & Sons.

Press, William H. et al., 1986, *Numerical Recipes: The Art of Scientific Computing*, Cambridge, UK: Cambridge University Press.

Quine, W. V., 1959, "On Cores and Prime Implicants of Truth Functions," *American Mathematical Monthly*, 66, Nov. 1959, pp. 755-760.

Rasmuson, D. M., 1998, "Treatment of Common-Cause Failures in Event Assessment," *Proceedings of the 4th International Conference on Probabilistic Safety Assessment and Management*, September, New York City, USA, Volume 2, Pages 1183 -1188.

Russell, K. D. and D. M. Rasmuson, 1993, "Fault Tree Reduction and Quantification – An Overview of IRRAS Algorithms," *Reliability Engineering and System Safety*, **40**, pp. 149-164.

Russell, K. D., K. J. Kvarfordt, N. L. Skinner, S. T. Wood, 1994, *Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Version 5.0, System Analysis and Risk Assessment (SARA) System Reference Manual*, NUREG/CR-6116, Vol. 4, July 1994.

Russell, K. D., K. J. Kvarfordt, N. L. Skinner, S. T. Wood, D. M. Rasmuson, 1994, *Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Version 5.0, Integrated Reliability and Risk Analysis System (IRRAS) Reference Manual*, NUREG/CR-6116, Vol. 2, July 1994.

Russell, K. D. and N. L. Skinner, 1994, *Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Version 5.0, Models and Results Database (MAR-D) Reference Manual*, NUREG/CR-6116, Vol. 8, July 1994.

Sattison, M. B, K. D. Russell, N. L. Skinner, 1994, *Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Version 5.0, System Analysis and Risk Assessment (SARA) System Tutorial Manual*, NUREG/CR-6116, Vol. 5, July 1994.

Singpurwalla, N., 1988, *Foundational Issues in Reliability and Risk Analysis*, Siam Review, Volume 30, No. 2, June, pp. 264-282.

Smith, C., J. Knudsen, and M. Calley, 1999, "Calculating and addressing uncertainty for risk-based allowable outage times," *Reliability Engineering and System Safety*, **66** (1999) 41-47.

Thisted, Ronald A., 1988, *Elements of Statistical Computing*, New York: Chapman and Hall.

US NRC, 2003, *Handbook of Parameter Estimation for Probabilistic Risk Assessment*, NUREG/CR-6823, September.

VanHorn, R. L, K. D. Russell, N. L. Skinner, 1994, *Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Version 5.0, Integrated Reliability and Risk Analysis System (IRRAS) Tutorial Manual*, NUREG/CR-6116, Vol. 3, July 1994.

Venn, J., 1880, *On the Diagrammatic and Mechanical Representation of Propositions and Reasonings*, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 9 (1880), pp. 1-18.

Vesely, W. E. et al., 1981, *Fault Tree Handbook*, NUREG-0492, January 1981.

Wilks, S. S., 1962, *Mathematical Statistics*, John Wiley and Sons: New York.

Appendix A

Fault Tree Quantification Example

Appendix A

Fault Tree Quantification Example

A1. INTRODUCTION

This appendix contains a worked example of the reduction and quantification of a simple fault tree. The minimal cut sets are obtained using a cut set algorithm and also using Boolean equations. The minimal cut sets are then quantified using the rare event approximation; the minimal cut set upper bound, and the inclusion-exclusion rule to obtain the exact solution. These quantification steps are worked out in detail. Finally, basic event importance measures are calculated to show how the calculations are done.

This appendix uses the notation + for \cup and * for \cap .

A2. FAULT TREE INPUT

The fault tree for this example is shown in Figure A-1. It contains a 2/3 combination gate. The alphanumeric input for the fault tree is shown in the following:

Alphanumeric Fault Tree (Shown in Figure A-1)

TOP	AND	GATE1	GATE2	
GATE1	2/3	GATE3	GATE4	B1
GATE2	OR	B1	B3	B4
GATE3	OR	B2	B4	
GATE4	AND	B3	B5	

Each row corresponds to a gate in the fault tree. The first entry is the gate name. The next entry is the gate type. The remaining entries are the inputs to the gate.

Next, we show the fault tree logic when the 2/3 combination gate (GATE1) is expanded into AND and OR gates. The new gates are FT-N/M-1, FT-N/M-2, and FT-N/M-3. The alphanumeric coding of the fault tree is shown below:

Alphanumeric Fault Tree with Expanded Gates

TOP	AND	GATE1	GATE2		
GATE1	OR	FT-N/M-1	FT-N/M-2	FT-N/M-3	
GATE2	OR	B1	B3	B4	
GATE3	OR	B2	B4		
FT-N/M-1	AND	GATE3	B3	B5	
FT-N/M-2	AND	GATE3	B1		
FT-N/M-3	AND	B3	B5	B1	

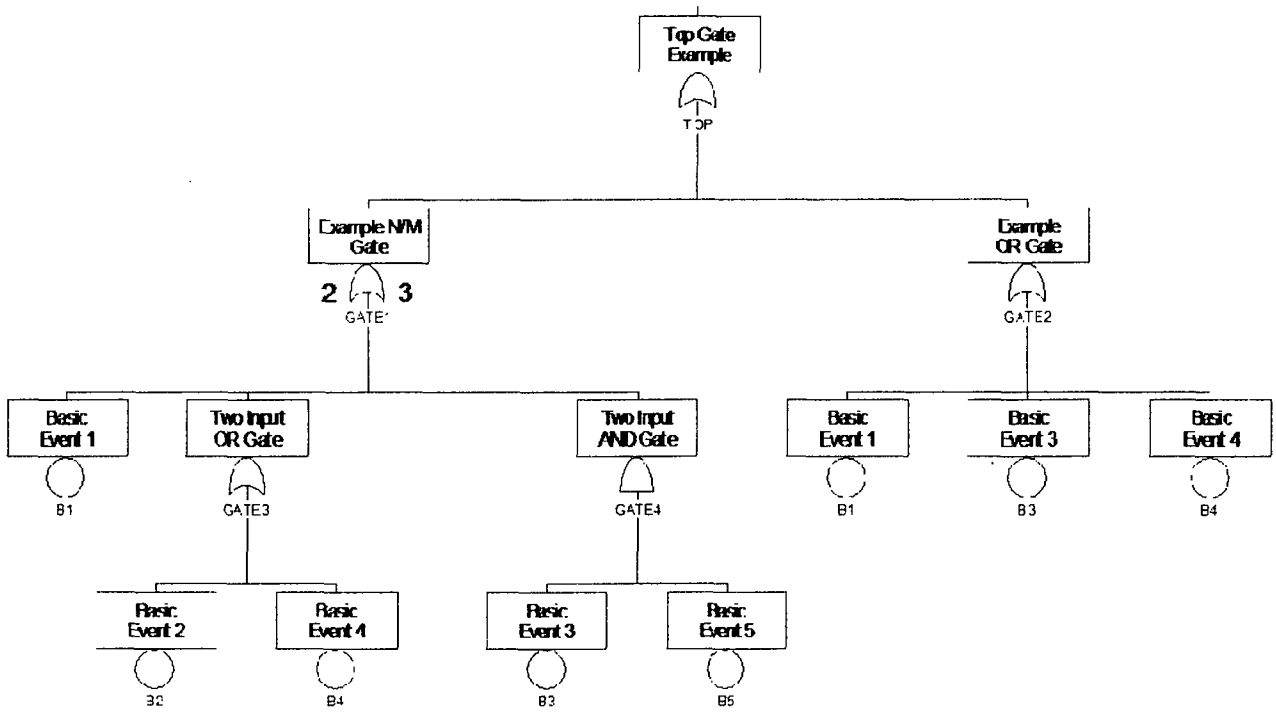


Figure A-1. Example fault tree graphic

CUT SET GENERATION (Top-down approach)

In this section the minimal cut sets are obtained using a top-down approach. The steps are illustrated in detail so that the reader can understand all of the calculation details. In practice, several of the steps can be performed together.

Step 1 (TOP)

To start the algorithm the TOP gate is replaced by its inputs. If the TOP gate is an OR gate, then each input becomes a row. If the TOP gate is an AND gate, the inputs are placed in the same row. Thus, the first step is the following:

GATE1 GATE2

Step 2 (GATE1)

In this step, GATE1 is replaced by its three inputs. Since GATE1 is an OR gate each input becomes a row. This results in the following:

FT-N/M-1 GATE2

FT-N/M-2 GATE2

FT-N/M-3 GATE2

Step 3 (FT-N/M-1)

In this step, FT-N/M-1 is replaced by its inputs GATE3, B3 and B5. Only the first row was modified since the gate is an AND gate. The results are:

B3 B5 GATE2 GATE3

FT-N/M-2 GATE2

FT-N/M-3 GATE2

Step 4 (FT-N/M-2)

Next, FT-N/M-2 is expanded. It is an AND gate so it is replaced by its inputs in every row that contains it. The results of this step are:

B3 B5 GATE2 GATE3

B1 GATE2 GATE3

FT-N/M-3 GATE2

Step 5 (FT-N/M-3)

Gate FT-N/M-3 is selected to process. It is also an AND gate and appears in only one row of the table in step 4. Thus, no rows are added in this step. The gate is replaced by its inputs. The results are:

B3	B5	GATE2	GATE3
B1	GATE2	GATE3	
B1	B3	B5	GATE2

Step 6 (GATE3)

GATE3 is selected to be expanded next. GATE3 is an OR gate with two inputs. For the first row in the table in step 5, GATE3 is replaced by one of its inputs. The row is then repeated and the gate name replaced by its other inputs. The results of this step are:

B2	B3	B5	GATE2	(Replace GATE3 by B2.)
B1	B2		GATE2	(Replace GATE3 by B2.)
B3	B4	B5	GATE2	(Replace GATE3 by B4.)
B1	B4		GATE2	(Replace GATE3 by B4.)
B1	B3	B5	GATE2	(Does not involve GATE3.)

Notice that two new rows were added in this step.

Step 7 (GATE2)

In this step, GATE2 is processed. Notice that GATE2 appears in every row of the table in step 6. GATE2 is an OR gate with 3 inputs. Thus, the number of rows will increase, but the number of entries in each row will remain the same. The number of rows will be three times the number in the table of step 6. That is, the table for this step will consist of 15 rows. The table for this step is the following:

B2	B3	B5	B1	
B1	B2	B1		
B3	B4	B5	B1	(Replace GATE2 by B1.)
B4	B1	B1		
B1	B3	B5	B1	
B2	B3	B5	B3	

B1	B2	B3	
B3	B4	B5	B3 (Replace GATE2 by B3.)
B1	B4	B3	
B1	B3	B5	B3

B2	B3	B5	B4
B1	B2	B4	
B3	B4	B5	B4 (Replace GATE2 by B4.)
B1	B4	B4	
B1	B3	B5	B4

Step 8 (Idempotence $A*A=A$)

At this point, all of the gates have been resolved so that only basic events occur in the table. The next step is to apply the Law of Idempotence, $A*A = A$. The results are:

B2	B3	B5	B1	=	B1	B2	B3	B5
B1	B2	B1		=	B1	B2		
B3	B4	B5	B1	=	B1	B3	B4	B5
B4	B1	B1		=	B1	B4		
B1	B3	B5	B1	=	B1	B3	B5	
B2	B3	B5	B3	=	B2	B3	B5	
B1	B2	B3		=	B1	B2	B3	
B3	B4	B5	B3	=	B3	B4	B5	
B1	B4	B3		=	B1	B3	B4	
B1	B3	B5	B3	=	B1	B3	B5	
B2	B3	B5	B4	=	B2	B3	B4	B5
B1	B2	B4		=	B1	B2	B4	
B3	B4	B5	B4	=	B3	B4	B5	
B1	B4	B4		=	B1	B4		
B1	B3	B5	B4	=	B1	B3	B4	B5

Step 9 (Absorption $A+(A*B)=A$)

The next step is the absorption step. That is, nonminimal cut sets must be eliminated, as well as duplicate rows. In the following table, the rows that are eliminated have a line through them and the reason it is eliminated is provided to the left. The results are:

B1	B2			
B1	B2	B3	Eliminated by B1 B2	
B1	B2	B3	B5	Eliminated by B1 B2
B1	B2	B4	Eliminated by B1 B2	
B1	B3	B4	Eliminated by B1 B4	
B1	B3	B4	B5	Eliminated by B1 B4
B1	B3	B4	B5	Eliminated by B1 B4
B1	B3	B5		
B1	B3	B5	Repeated cut set	
B1	B4			
B1	B4		Repeated cut set	
B2	B3	B4	B5	Eliminated by B2 B3 B5
B2	B3	B5		
B3	B4	B5		
B3	B4	B5	Repeated cut set	

Step 10 (Final minimal cut sets)

The remaining 5 sets are the minimal cut sets for this example. They are:

B1	B2	
B1	B4	
B1	B3	B5
B2	B3	B5
B3	B4	B5

A3. BOOLEAN EQUATION FOR THE FAULT TREE

In this section the Boolean equation form of the fault tree is used to obtain the minimal cut sets. The steps below are not the only way the equations can be combined and reduced. Many of the steps illustrated below can be combined and performed simultaneously. These steps are

presented to illustrate the various concepts and show how they parallel the cut set algorithm illustrated in the previous section.

The equation form of the fault tree is:

$$\begin{aligned} \text{TOP} &= \text{GATE1} * \text{GATE2} \\ \text{GATE1} &= \text{FT-N/M-1} + \text{FT-N/M-2} + \text{FT-N/M-3} \\ \text{GATE2} &= \text{B1} + \text{B3} + \text{B4} \\ \text{GATE3} &= \text{B2} + \text{B4} \\ \text{FT-N/M-1} &= \text{GATE3} * \text{B3} * \text{B5} \\ \text{FT-N/M-2} &= \text{GATE3} * \text{B1} \\ \text{FT-N/M-3} &= \text{B1} * \text{B3} * \text{B5} \end{aligned}$$

Step 1

The first step is to start with the TOP equation:

$$\text{TOP} = \text{GATE1} * \text{GATE2}.$$

Step 2

In this step GATE1 and GATE2 are replaced by their inputs. This results in the following equation:

$$\text{TOP} = (\text{FT-N/M-1} + \text{FT-N/M-2} + \text{FT-N/M-3}) * (\text{B1} + \text{B3} + \text{B4}).$$

Step 3

In this step the three expanded gates (FT-N/M-1, FT-N/M-2, and FT-N/M-3) are replaced by their inputs

$$\text{TOP} = (\text{GATE3} * \text{B3} * \text{B5} + \text{GATE3} * \text{B1} + \text{B1} * \text{B3} * \text{B5}) * (\text{B1} + \text{B3} + \text{B4}).$$

Step 4

Next GATE3 is replaced by its inputs to obtain

$$\text{TOP} = (\text{B1} + \text{B3} + \text{B4}) * [(\text{B2} + \text{B4})(\text{B3} * \text{B5}) + (\text{B2} + \text{B4}) * \text{B1} + \text{B1} * \text{B3} * \text{B5}].$$

At this point all gates have been replaced by their inputs, and the equation consists of basic events only.

Step 5

The next step is to expand and combine the terms in the square brackets. This yields

$$\text{TOP} = (B1 + B3 + B4) * (B2*B3*B5 + B3*B4*B5 + B1*B2 + B1*B4 + B1*B3*B5).$$

Step 6

The terms in the first set of parentheses are distributed across the second set to yield

$$\begin{aligned} \text{TOP} = & B1 * (B2*B3*B5 + B3*B4*B5 + B1*B2 + B1*B4 + B1*B3*B5) \\ & + B3 * (B2*B3*B5 + B3*B4*B5 + B1*B2 + B1*B4 + B1*B3*B5) \\ & + B4 * (B2*B3*B5 + B3*B4*B5 + B1*B2 + B1*B4 + B1*B3*B5). \end{aligned}$$

Step 7

Each term is now expanded to yield

$$\begin{aligned} \text{TOP} = & B1*B2*B3*B5 + B1*B3*B4*B5 + B1*B1*B2 + B1*B1*B4 + B1*B1*B3*B5 \\ & + B3*B2*B3*B5 + B3*B3*B4*B5 + B1*B2*B3 + B1*B3*B4 + B1*B3*B3*B5 \\ & + B2*B3*B4*B5 + B3*B4*B4*B5 + B1*B2*B4 + B1*B4*B4 + B1*B3*B4*B5. \end{aligned}$$

Step 8 (Idempotence)

The Law of Idempotence ($A*A=A$) is now applied. This produces

$$\begin{aligned} \text{TOP} = & B1*B2*B3*B5 + B1*B3*B4*B5 + B1*B2 + B1*B4 + B1*B3*B5 \\ & + B2*B3*B5 + B3*B4*B5 + B1*B2*B3 + B1*B3*B4 + B1*B3*B5 \\ & + B2*B3*B4*B5 + B3*B4*B5 + B1*B2*B4 + B1*B4 + B1*B3*B4*B5. \end{aligned}$$

Step 9 (Absorption)

Finally, the nonminimal cut sets are eliminated. The terms that are eliminated are shown with a line through them.

$$\begin{aligned} \text{TOP} = & \cancel{B1*B2*B3*B5} + \cancel{B1*B3*B4*B5} + B1*B2 + B1*B4 + B1*B3*B5 \\ & + B2*B3*B5 + B3*B4*B5 + \cancel{B1*B2*B3} + \cancel{B1*B3*B4} + \cancel{B1*B3*B5} \\ & + \cancel{B2*B3*B4*B5} + \cancel{B3*B4*B5} + \cancel{B1*B2*B4} + B1*B4 + \cancel{B1*B3*B4*B5} \end{aligned}$$

Minimal Cut Set Equation

The final minimal cut set equation is

$$\text{TOP} = B1*B2 + B1*B4 + B1*B3*B5 + B2*B3*B5 + B3*B4*B5.$$

These are exactly the same minimal cut sets that were obtained in Section A2.

A4. CUT SET QUANTIFICATION

In this section the different ways of quantifying the minimal cut sets are compared. Numerical results are treated in the next section. The objective is to illustrate the complexity of the exact solution and also the Boolean algebra required in calculating it.

The minimal cut set equation is the starting point for the calculations. From Section A2 or A3, we have

$$P[\text{TOP}] = P[B1*B2 + B1*B4 + B1*B3*B5 + B2*B3*B5 + B3*B4*B5]$$

Exact Solution

The inclusion-exclusion rule (i.e., the min-max option) is used to calculate the exact solution. Basically, it is the sum of the probability of the individual sets, minus the sum of the probability of all possible pairs, plus the sum of the probabilities of all possible combinations of three, minus the probabilities of all possible combinations of four, plus the probability of intersection of all five minimal cut sets. This calculation is shown in Table A-1.

Table A-1 Exact solution, Step 1.

$$\begin{aligned}
 P[\text{TOP}] &= P\{\{B1*B2\}\} \\
 &+ P\{\{B1*B4\}\} \\
 &+ P\{\{B1*B3*B5\}\} \\
 &+ P\{\{B2*B3*B5\}\} \\
 &+ P\{\{B3*B4*B5\}\} \\
 \\
 &- P\{\{B1*B2\} * \{B1*B4\}\} \\
 &- P\{\{B1*B2\} * \{B1*B3*B5\}\} \\
 &- P\{\{B1*B2\} * \{B2*B3*B5\}\} \\
 &- P\{\{B1*B2\} * \{B3*B4*B5\}\} \\
 &- P\{\{B1*B4\} * \{B1*B3*B5\}\} \\
 &- P\{\{B1*B4\} * \{B2*B3*B5\}\} \\
 &- P\{\{B1*B4\} * \{B3*B4*B5\}\} \\
 &- P\{\{B1*B3*B5\} * \{B2*B3*B5\}\} \\
 &- P\{\{B1*B3*B5\} * \{B3*B4*B5\}\} \\
 &- P\{\{B2*B3*B5\} * \{B3*B4*B5\}\} \\
 \\
 &+ P\{\{B1*B2\} * \{B1*B4\} * \{B1*B3*B5\}\} \\
 &+ P\{\{B1*B2\} * \{B1*B4\} * \{B2*B3*B5\}\} \\
 &+ P\{\{B1*B2\} * \{B1*B4\} * \{B3*B4*B5\}\} \\
 &+ P\{\{B1*B2\} * \{B1*B3*B5\} * \{B2*B3*B5\}\} \\
 &+ P\{\{B1*B2\} * \{B1*B3*B5\} * \{B3*B4*B5\}\} \\
 &+ P\{\{B1*B2\} * \{B2*B3*B5\} * \{B3*B4*B5\}\} \\
 &+ P\{\{B1*B4\} * \{B1*B3*B5\} * \{B2*B3*B5\}\} \\
 &+ P\{\{B1*B4\} * \{B1*B3*B5\} * \{B3*B4*B5\}\} \\
 &+ P\{\{B1*B4\} * \{B2*B3*B5\} * \{B3*B4*B5\}\} \\
 &+ P\{\{B1*B3*B5\} * \{B2*B3*B5\} * \{B3*B4*B5\}\} \\
 \\
 &- P\{\{B1*B2\} * \{B1*B4\} * \{B1*B3*B5\} * \{B2*B3*B5\}\} \\
 &- P\{\{B1*B2\} * \{B1*B4\} * \{B1*B3*B5\} * \{B3*B4*B5\}\}
 \end{aligned}$$

From Table A-1 we see that the intersection of most of the sets contain common terms, e.g., B1 B2 and B1 B4 have B1 in common. The intersections must be reduced to simplest form by use of the Law of Idempotence ($A * A = A$). The results of this are shown in Table A-2.

In most situations, the basic events are assumed to be statistically independent. That is, $P[AB] = P[A]P[B]$. The results of this step are shown in Table A-3.

Minimal Cut Set Upper Bound

The minimal cut set upper bound for our example is shown in Table A-4.

Rare Event Approximation

The first term of the inclusion-exclusion rule is an upper bound for the probability of the TOP event. For our example the rare event approximation is

$$P[\text{TOP}] = P[B1 * B2] + P[B1 * B4] + P[B1 * B3 * B5] + P[B2 * B3 * B5] + P[B3 * B4 * B5].$$

Table A-2. Exact solution after applying Law of Idempotence.

$$\begin{aligned}
 P[\text{TOP}] &= P[B1*B2] \\
 &+ P[B1*B4] \\
 &+ P[B1*B3*B5] \\
 &+ P[B2*B3*B5] \\
 &+ P[B3*B4*B5] \\
 \\
 &- P[B1*B2*B4] \\
 &- P[B1*B2*B3*B5] \\
 &- P[B1*B2*B3*B5] \\
 &- P[B1*B2*B3*B4*B5] \\
 &- P[B1*B3*B4*B5] \\
 &- P[B1*B2*B3*B4*B5] \\
 &- P[B1*B3*B4*B5] \\
 &- P[B1*B2*B3*B5] \\
 &- P[B1*B3*B4*B5] \\
 &- P[B2*B3*B4*B5] \\
 \\
 &+ P[B1*B2*B3*B4*B5] \\
 &+ P[B1*B2*B3*B4*B5] \\
 &+ P[B1*B2*B3*B4*B5] \\
 &+ P[B1*B2*B3*B4*B5] \\
 &+ P[B1*B2*B3*B4*B5] \\
 &+ P[B1*B2*B3*B4*B5] \\
 &+ P[B1*B2*B3*B4*B5] \\
 &+ P[B1*B2*B3*B4*B5] \\
 &+ P[B1*B2*B3*B4*B5] \\
 &+ P[B1*B2*B3*B4*B5] \\
 &- P[B1*B2*B3*B4*B5]
 \end{aligned}$$

Table A-4. Minimal cut set upper bound calculations for example.

$$P[\text{TOP}] = 1 - (1 - P\{\{B1*B2\}\}) * (1 - P\{\{B1*B4\}\}) * (1 - P\{\{B1*B3*B5\}\}) * (1 - P\{\{B2*B3*B5\}\}) * (1 - P\{\{B3*B4*B5\}\})$$

$$= P\{\{B1*B2\}\}$$

$$+ P\{\{B1*B4\}\}$$

$$+ P\{\{B1*B3*B5\}\}$$

$$+ P\{\{B2*B3*B5\}\}$$

$$+ P\{\{B3*B4*B5\}\}$$

$$- P\{\{B1*B2\}\} * P\{\{B1*B4\}\}$$

$$- P\{\{B1*B2\}\} * P\{\{B1*B3*B5\}\}$$

$$- P\{\{B1*B2\}\} * P\{\{B2*B3*B5\}\}$$

$$- P\{\{B1*B2\}\} * P\{\{B3*B4*B5\}\}$$

$$- P\{\{B1*B4\}\} * P\{\{B1*B3*B5\}\}$$

$$- P\{\{B1*B4\}\} * P\{\{B2*B3*B5\}\}$$

$$- P\{\{B1*B4\}\} * P\{\{B3*B4*B5\}\}$$

$$- P\{\{B1*B3*B5\}\} * P\{\{B2*B3*B5\}\}$$

$$- P\{\{B1*B3*B5\}\} * P\{\{B3*B4*B5\}\}$$

$$- P\{\{B2*B3*B5\}\} * P\{\{B3*B4*B5\}\}$$

$$+ P\{\{B1*B2\}\} * P\{\{B1*B4\}\} * P\{\{B1*B3*B5\}\}$$

$$+ P\{\{B1*B2\}\} * P\{\{B1*B4\}\} * P\{\{B2*B3*B5\}\}$$

$$+ P\{\{B1*B2\}\} * P\{\{B1*B4\}\} * P\{\{B3*B4*B5\}\}$$

$$+ P\{\{B1*B2\}\} * P\{\{B1*B3*B5\}\} * P\{\{B2*B3*B5\}\}$$

$$+ P\{\{B1*B2\}\} * P\{\{B1*B3*B5\}\} * P\{\{B3*B4*B5\}\}$$

$$+ P\{\{B1*B2\}\} * P\{\{B2*B3*B5\}\} * P\{\{B3*B4*B5\}\}$$

$$+ P\{\{B1*B4\}\} * P\{\{B1*B3*B5\}\} * P\{\{B2*B3*B5\}\}$$

$$+ P\{\{B1*B4\}\} * P\{\{B1*B3*B5\}\} * P\{\{B3*B4*B5\}\}$$

$$+ P\{\{B1*B4\}\} * P\{\{B2*B3*B5\}\} * P\{\{B3*B4*B5\}\}$$

$$+ P\{\{B1*B3*B5\}\} * P\{\{B2*B3*B5\}\} * P\{\{B3*B4*B5\}\}$$

A5. NUMERICAL CALCULATIONS

This section contains numerical calculations illustrating the formulas developed in the previous section. The basic event probabilities for our example problem are the following:

$$P(B1) = q_1 = 0.01$$

$$P(B2) = q_2 = 0.02$$

$$P(B3) = q_3 = 0.03$$

$$P(B4) = q_4 = 0.04$$

$$P(B5) = q_5 = 0.05$$

The cut set unavailabilities, denoted by C_i , are calculated below:

$$C_1 = P(B1*B2) = P(B1)*P(B2) = q_1q_2 = 0.01 * 0.02 = 2.0E-4$$

$$C_2 = P(B1*B4) = P(B1)*P(B4) = q_1q_4 = 0.01 * 0.04 = 4.0E-4$$

$$C_3 = P(B1*B3*B5) = P(B1)*P(B3)*P(B5) = q_1q_3q_5 = 0.01 * 0.03 * 0.05 = 1.5E-5$$

$$C_4 = P(B2*B3*B5) = P(B2)*P(B3)*P(B5) = q_2q_3q_5 = 0.02 * 0.03 * 0.05 = 3.0E-5$$

$$C_5 = P(B3*B4*B5) = P(B3)*P(B4)*P(B5) = q_3q_4q_5 = 0.03 * 0.04 * 0.05 = 6.0E-5$$

Using the cut set unavailabilities, the rare event approximation and the minimal cut set upper bound can be calculated. The rare event approximation is:

$$\text{Rare Event Approximation} = C_1 + C_2 + C_3 + C_4 + C_5 = 7.050E-4$$

The minimal cut set upper bound is:

$$\begin{aligned} \text{Min Cut Upper Bound} &= 1 - (1-C_1) * (1-C_2) * (1-C_3) * (1-C_4) * (1-C_5) \\ &= 1 - 0.9998 * 0.9996 * 0.999985 * 0.99997 * 0.99994 \\ &= 1 - 0.99929515 = 7.0485386E-4 \end{aligned}$$

The exact solution calculations are shown in Table A-6. Table A-5 compares the results of the three calculation formulas.

Table A-7 shows the probabilities of the contributors (listed in Table A-4) for the minimum cut set upper bound. A line-by-line examination shows that some lines of Table A-7 have certain

basic event probabilities repeated and that this is the only difference between Tables A-6 and A-7. A corresponding comparison can be made of Tables A-3 and A-4.

Table A-5 Comparison of Results

Type of Calculation	Unavailability
Min Cut Upper Bound	7.04854E-4
Rare Event Approximation	7.05000E-4
Sum of 1st and 2nd order terms ^a	6.93076E-4
Sum of 1st* 2nd and 3rd order terms ^a	6.93196E-4
Sum of 1st* 2nd* 3rd* and 4th order terms ^a	6.93136E-4
Sum of all terms (Exact answer) ^a	6.93148E-4

a. See Table A-6 for details.

Table A-6. Calculations for exact solution.

<u>Basic Events in Term</u>	<u>Unavailability</u>
+ B1 B2	2.000E-04
+ B1 B4	4.000E-04
+ B1 B3 B5	1.500E-05
+ B2 B3 B5	3.000E-05
+ B3 B4 B5	6.000E-05
- B1 B2 B4	-8.000E-06
- B1 B2 B3 B5	-3.000E-07
- B1 B2 B3 B5	-3.000E-07
- B1 B2 B3 B4 B5	-1.200E-08
- B1 B3 B4 B5	-6.000E-07
- B1 B2 B3 B4 B5	-1.200E-08
- B1 B3 B4 B5	-6.000E-07
- B1 B2 B3 B5	-3.000E-07
- B1 B3 B4 B5	-6.000E-07
- B2 B3 B4 B5	-1.200E-06
+ B1 B2 B3 B4 B5	1.200E-08
+ B1 B2 B3 B4 B5	1.200E-08
+ B1 B2 B3 B4 B5	1.200E-08
+ B1 B2 B3 B4 B5	1.200E-08
+ B1 B2 B3 B4 B5	1.200E-08
+ B1 B2 B3 B4 B5	1.200E-08
+ B1 B2 B3 B4 B5	1.200E-08
+ B1 B2 B3 B4 B5	1.200E-08
+ B1 B2 B3 B4 B5	1.200E-08
+ B1 B2 B3 B4 B5	1.200E-08
+ B1 B2 B3 B4 B5	1.200E-08
- B1 B2 B3 B4 B5	-1.200E-08

Table A-7. Probabilities of contributors to minimal cut set upper bound.

<u>Basic Events in Term</u>	<u>Unavailability</u>
+ B1 B2	2.000E-04
+ B1 B4	4.000E-04
+ B1 B3 B5	1.500E-05
+ B2 B3 B5	3.000E-05
+ B3 B4 B5	6.000E-05
- B1 B2 B1 B4	-8.000E-08
- B1 B2 B1 B3 B5	-3.000E-09
- B1 B2 B2 B3 B5	-6.000E-09
- B1 B2 B3 B4 B5	-1.200E-08
- B1 B4 B1 B3 B5	-6.000E-09
- B1 B4 B2 B3 B5	-1.200E-08
- B1 B4 B3 B4 B5	-2.400E-08
- B1 B3 B5 B2 B3 B5	-4.500E-10
- B1 B3 B5 B3 B4 B5	-9.000E-10
- B2 B3 B5 B3 B4 B5	-1.800E-09
+ B1 B2 B1 B4 B1 B3 B5	1.200E-12
+ B1 B2 B1 B4 B2 B3 B5	2.400E-12
+ B1 B2 B1 B4 B3 B4 B5	4.800E-12
+ B1 B2 B1 B3 B5 B2 B3 B5	9.000E-14
+ B1 B2 B1 B3 B5 B3 B4 B5	1.800E-13
+ B1 B2 B2 B3 B5 B3 B4 B5	3.600E-13
+ B1 B4 B1 B3 B5 B2 B3 B5	1.800E-13
+ B1 B4 B1 B3 B5 B3 B4 B5	3.600E-13
+ B1 B4 B2 B3 B5 B3 B4 B5	7.200E-13
+ B1 B3 B5 B2 B3 B5 B3 B4 B5	2.700E-14
- B1 B2 B1 B4 B1 B3 B5 B2 B3 B5	-3.600E-17

A6. IMPORTANCE MEASURES

Basic Event Probabilities

$$P(B1) = q_1 = 0.01$$

$$P(B2) = q_2 = 0.02$$

$$P(B3) = q_3 = 0.03$$

$$P(B4) = q_4 = 0.04$$

$$P(B5) = q_5 = 0.05$$

$$C_1 = P(B1*B2) = P(B1)*P(B2) = q_1q_2 = 0.01 * 0.02 = 2.0E-4$$

$$C_2 = P(B1*B4) = P(B1)*P(B4) = q_1q_4 = 0.01 * 0.04 = 4.0E-4$$

$$C_3 = P(B1*B3*B5) = P(B1)*P(B3)*P(B5) = q_1q_3q_5 = 0.01 * 0.03 * 0.05 = 1.5E-5$$

$$C_4 = P(B2*B3*B5) = P(B2)*P(B3)*P(B5) = q_2q_3q_5 = 0.02 * 0.03 * 0.05 = 3.0E-5$$

$$C_5 = P(B3*B4*B5) = P(B3)*P(B4)*P(B5) = q_3q_4q_5 = 0.03 * 0.04 * 0.05 = 6.0E-5$$

$$Q = C_1 + C_2 + C_3 + C_4 + C_5 = 7.050E-4$$

Fussell-Vesely Importance Measure

$$B1 - FV(B1) = (C1 + C2 + C3)/Q = (2.0E-4 + 4.0E-4 + 1.5E-5)/7.05E-4 = 0.8723$$

$$B2 - FV(B2) = (C1 + C4)/Q = (2.0E-4 + 3.E-5) / 7.05E-4 = 0.3262$$

$$B3 - FV(B3) = (C3+C4+C5)/Q = 1.05E-5/7.05E-5 = 0.1489$$

$$B4 - FV(B4) = (C2+C5)/Q = 4.0E-4 + 4.6E-4/7.05E-4 = 0.6525$$

$$B5 - FV(B5) = (C3+C4+C5)/Q = 1.05E-4 / 7.05E-4 = 0.1489$$

Risk Reduction Importance

For B1, set $q_1 = 0.0$. Then we get

$$C_1 = P(B1*B2) = P(B1)*P(B2) = q_1q_2 = 0.01 * 0.02 = 0$$

$$\begin{aligned}
C_2 = P(B1*B4) &= P(B1)*P(B4) &= q_1q_4 &= 0.01 * 0.04 &= 0 \\
C_3 = P(B1*B3*B5) & &= P(B1)*P(B3)*P(B5) &= q_1q_3q_4 &= 0.01 * 0.03 * \\
0.05 &= 0 & & & \\
C_4 = P(B2*B3*B5) & &= P(b2)*P(B3)*P(B5) &= q_2q_3q_5 &= 0.02 * 0.03 * \\
0.05 &= 3.0E-5 & & & \\
C_5 = P(B3*B4*B5) & &= P(B3)*P(B4)*P(B5) &= q_3q_4q_5 &= 0.03 * 0.04 * \\
0.05 &= 6.0E-5 & & &
\end{aligned}$$

Using these results, the risk reduction ratio is

$$RRR(B1) = 7.05E-4/(3.0E-5+6.0E-5) = 7.05E-4/9.0E-5 = 7.833,$$

and the risk reduction difference is

$$RRD(B1) = 7.05E-4 - 9.0E-5 = 6.15e-4.$$

Risk Increase Importance

For B1, set $q_1 = 1.0$. Then we get

$$\begin{aligned}
C_1 = P(B1*B2) &= P(B1)*P(B2) &= q_1q_2 &= 1.0 * 0.02 &= 0.02 \\
C_2 = P(B1*B4) &= P(B1)*P(B4) &= q_1q_4 &= 1.0 * 0.04 &= 0.04 \\
C_3 = P(B1*B3*B5) & &= P(B1)*P(B3)*P(B5) &= q_1q_3q_4 &= 1.0 * \\
0.03 * 0.05 &= 1.5E-3 & & & \\
C_4 = P(B2*B3*B5) & &= P(b2)*P(B3)*P(B5) &= q_2q_3q_5 &= 0.02 \\
* 0.03 * 0.05 &= 3.0E-5 & & & \\
C_5 = P(B3*B4*B5) & &= P(B3)*P(B4)*P(B5) &= q_3q_4q_5 &= 0.03 \\
* 0.04 * 0.05 &= 6.0E-5 & & &
\end{aligned}$$

Using these results, the risk increase ratio is

$$RIR(B1) = 6.159E-2/7.05E-4 = 86.36,$$

and the risk increase difference is

$$RID(B1) = 6.159E-2 - 7.05E-4 = 6.089E-2.$$

Birnbaum Importance

$$B(B1) = 6.159E-2 - 9.0E-5 = 6.15E-5.$$

Structural Importance

B1 appears in three cut sets

Table A-8 Ratio importance measures

Name	Num of Occ.	Probability of Failure	Fussell- Vesely Importance	Risk Reduction Ratio	Risk Increase Ratio
----	----	-----	-----	-----	-----
B1	3	1.000E-2	8.723E-1	7.832	8.611E+1
B4	2	4.000E-2	6.524E-1	2.877	1.664E+1
B2	2	2.000E-2	3.261E-1	1.484	1.696E+1
B3	3	3.000E-2	1.489E-1	1.175	5.809E+0
B5	3	5.000E-2	1.489E-1	1.175	3.827E+0

Table A-9 Difference importance measures

Name	Num. of Occ.	Probability of Failure	Birnbaum Importance Measure	Risk Reduction Difference	Risk Increase Difference
----	----	-----	-----	-----	-----
B1	3	1.000E-2	6.061E-2	6.149E-4	5.999E-2
B4	2	4.000E-2	1.148E-2	4.599E-4	1.102E-2
B2	2	2.000E-2	1.148E-2	2.299E-4	1.125E-2
B3	3	3.000E-2	3.494E-3	1.049E-4	3.389E-3
B5	3	5.000E-2	2.097E-3	1.049E-4	1.993E-3

Appendix B

Common Cause Failure Calculations

Appendix B

Common Cause Failure Calculations

B.1 Common Cause Failure Adjustments

SAPHIRE will allow for common cause failure (CCF) adjustments by using formal Boolean reduction. Assume we have a system S of:

S AND A_TRAIN B_TRAIN
A_TRAIN OR Ai CCF SUPPORT_1
B_TRAIN OR Bi CCF SUPPORT_2
SUPPORT_1 OR p1
SUPPORT_2 OR p2

So the system cut sets are:

$$S = CCF + A_i * B_i + A_i * p_2 + p_1 * B_i$$

where CCF is common-cause failure of components A and B

If we fail component A, so we need to calculate:

$$P[S|A_t] = P[S \text{ and } A_t] / P[A_t]$$

where $A_t = A_i$ or CCF

Thus,

$$[S \text{ and } A_t] = (CCF + A_i * B_i + A_i * p_2 + p_1 * B_i) \text{ and } (A_i \text{ or CCF})$$

$$= (CCF * A_i + A_i * B_i * A_i + A_i * p_2 * A_i + p_1 * B_i * A_i + CCF * CCF + A_i * B_i * CCF + A_i * p_2 * CCF + p_1 * B_i * CCF)$$

However, $CCF * A_i = CCF * B_i = 0$, so we can reduce this down to:

$$[S \text{ and } A_t] = (A_i * B_i + A_i * p_2 + CCF)$$

Now, assuming we are using the alpha factor method

$$\begin{aligned} P[S|A_t] &= P[S \text{ and } A_t] / P[A_t] \\ &= P[A_i * B_i + A_i * p_2 + CCF] / P[A_t] \\ &= P[A_i * B_i] / P[A_t] + P[A_i * p_2] / P[A_t] + P[CCF] / P[A_t] \\ &= (\alpha_1 Q_t * \alpha_1 Q_t) / Q_t + \alpha_1 Q_t * p_2 / Q_t + \alpha_2 Q_t / Q_t \\ &= (\alpha_1)^2 * Q_t + \alpha_1 * p_2 + \alpha_2 \end{aligned}$$

where $Q_t = P[A_t]$

SAPHIRE 8 is aware of any situation where one or more of the independent events are modified from its nominal condition. Included in this determination will be changes that affect CCF objects in:

- Fault tree logic or cut sets
- Sequence logic or cut sets
- End state cut sets

For example, say the user modifies A_FTS from the fault tree above when using a change set. In this change set, the user indicates that A_FTS was failed (unknown failure). When the change set is invoked and the user then goes to solve all of the fault trees, SAPHIRE will check each fault tree to determine if the modified A_FTS is part of a CCF module in the i'th fault tree logic. If it is, then SAPHIRE will solve the fault tree **conditional** upon the failure of A_FTS as indicated. If it is not, then SAPHIRE will solve the fault tree using the traditional solution method.

Note that end state cut sets will be evaluated slightly different than either fault tree or sequence logic. In SAPHIRE, end state cut sets are not "solved," they are gathered from already solved logic models. Consequently, the end state cut sets will need to be re-evaluated in a fashion similar to solving a logic model. The conditionality will need to be imposed on the cut sets, but in this case cut sets that are not affected may simply be ignored from modification. Conversely, any cut set containing elements of an affected CCF module will need to be evaluated as noted above.

It may not be possible to treat CCF objects as independent subtrees, whereby these trees will be solved last and the condition (if any) will be imposed directly on this subtree.

For example, if we revisit the example above, where we condition on A_t , where $A_t = A_i$ or CCF, and only apply the condition to the CCF event in the logic model:

$$[S \text{ and } A_t] = [CCF \text{ and } (A_i \text{ or } CCF)] + A_i * B_i + A_i * p_2 + p_1 * B_i$$

$$= [CCF \cdot A_i + CCF \cdot CCF] + A_i \cdot B_i + A_i \cdot p_2 + p_1 \cdot B_i$$

However, $CCF \cdot A_i = 0$, so we can reduce this down to:

$$[S \text{ and } A_t] = [CCF] + A_i \cdot B_i + A_i \cdot p_2 + p_1 \cdot B_i$$

Note that this result is different than imposing the condition onto the entire logic structure (as done above) where we found $[S \text{ and } A_t] = (A_i \cdot B_i + A_i \cdot p_2 + CCF)$. If we only condition the CCF event, we see that an additional term $p_1 \cdot B_i$ remains as part of the calculation. However, this term is not correct. Looking at the system logic, we see that:

A_TRAIN OR A_i CCF p₁

When this gate is conditioned by A_t occurring, then we have

$(A_i \text{ or } CCF)$ and $(A_i \text{ or } CCF \text{ or } p_1)$

$$= A_i \cdot A_i + CCF \cdot A_i + p_1 \cdot A_i + A_i \cdot CCF + CCF \cdot CCF + p_1 \cdot CCF$$

$$= A_i + p_1 \cdot A_i + CCF + p_1 \cdot CCF$$

$$= A_i + p_1 \cdot A_i + CCF$$

In other words, once either A_i or CCF occurs, any cut set with the event p_1 from this gate is non-minimal and should not appear in the list of minimal cut sets.

B.2 Detailed Example Calculation

For this section, first assume we have a three-component common-cause group where only one component is needed for success. The Boolean expression for this system is:

$$S = A_T \cap B_T \cap C_T.$$

The minimal cut sets and Basic Parameter Model (BPM) expression for this group are the following:

Cut Sets	BPM
$\{A_i, B_i, C_i\}$	
$\{A_i, C_{BC}\}$	$Q_1^3 +$
$\{B_i, C_{AC}\}$	$3 * Q_1 * Q_2 +$
$\{C_i, C_{AB}\}$	Q_3
$\{C_{ABC}\}$	

The potential cut sets $\{C_{AB}, C_{AC}\}$, $\{C_{AB}, C_{BC}\}$, and $\{C_{AC}, C_{BC}\}$ are not evaluated since the intersection of the events in each of these potential cut set is assumed to be the null set. In other words, we do not make a distinction from $\{C_{AB}, C_{AC}\}$ and $\{C_{ABC}\}$ and simply subsume that term with multiple separate CCF events.

Table B-1 provides a summary of the CCF modifications to be made for the various independent event modifications. A few comments on these adjustments include:

1. Dependent failures condition on the specific dependent failures (e.g., A and B) and any additional dependent failures ALSO containing those two failures.
2. The T/M conditions on the complement of the independent failure (e.g., /Ai).
3. A success event conditions on the complement of the total failure (e.g., /At).
4. The Ignore condition attempts to "map down" the system since the level of redundancy is reduced. The proposed approach takes the two highest alpha factors (e.g., α_2 and α_3 in the first example), adds them together as a new α_{k-1} , and then uses that new alpha factor as if the CCF had one less level of redundancy. This approach conserves CCF probabilities since the alpha factors still sum to 1.0.
5. The "virtual" event part of the CCF cut sets should be modified to reflect the conditional probability associated with that cut set. For example, in the case of an "unknown" failure, the cut set MOV-CC-A * A-RCCF-3-3-BC has a value of 6.370E-6. However, the event MOV-CC-A has a probability of 1.0 (it was assumed that A has the unknown failure) – consequently the A-RCCF-3-3-BC virtual event will have a probability of 6.370E-6.

Table B-1. CCF Analysis Table for the three-component (1-of-3 success criteria) example.

Note 1: $Q_t = A_t = 1E-3$, $\alpha_1 = 0.980$, $\alpha_2 = 0.0130$, $\alpha_3 = 0.0070$, assumed staggered testing model.

Note 2: The case of all independent failures $\{A_i, B_i, C_i\}$ is ignored in the table and the SAPHIRE CCF module.

Modification Type	Conditional Calculation	SAPHIRE Treatment	Result																								
Nominal	<p>Baseline calculation</p> <p>Cut sets</p> <p>$\{A_i, C_{BC}\}, \{B_i, C_{AC}\}, \{C_i, C_{AB}\}, \{C_{ABC}\}$</p> <p>$P(S) = (3/2) \alpha_1 Q_t \alpha_2 Q_t + \alpha_3 Q_t$</p>	<p>No modification from baseline:</p> <p>$1 * Q_3 + 3 * Q_1 * Q_2$</p>	<p>Total = 7.019E-6</p> <table border="1"> <thead> <tr> <th>#</th> <th>Terms</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td># 1</td> <td>MOV-CC-C</td> <td>6.370E-09</td> </tr> <tr> <td></td> <td>A-RCCF-3-3-AB</td> <td></td> </tr> <tr> <td># 2</td> <td>MOV-CC-B</td> <td>6.370E-09</td> </tr> <tr> <td></td> <td>A-RCCF-3-3-AC</td> <td></td> </tr> <tr> <td># 3</td> <td>MOV-CC-A</td> <td>6.370E-09</td> </tr> <tr> <td></td> <td>A-RCCF-3-3-BC</td> <td></td> </tr> <tr> <td># 4</td> <td>A-RCCF-3-3-ABC</td> <td>7.000E-06</td> </tr> </tbody> </table>	#	Terms	Value	# 1	MOV-CC-C	6.370E-09		A-RCCF-3-3-AB		# 2	MOV-CC-B	6.370E-09		A-RCCF-3-3-AC		# 3	MOV-CC-A	6.370E-09		A-RCCF-3-3-BC		# 4	A-RCCF-3-3-ABC	7.000E-06
#	Terms	Value																									
# 1	MOV-CC-C	6.370E-09																									
	A-RCCF-3-3-AB																										
# 2	MOV-CC-B	6.370E-09																									
	A-RCCF-3-3-AC																										
# 3	MOV-CC-A	6.370E-09																									
	A-RCCF-3-3-BC																										
# 4	A-RCCF-3-3-ABC	7.000E-06																									
Unknown failure	<p>Condition on an unknown failure, A_T</p> <p>$S \cap A_T = \{A_i, C_{BC}\}, \{B_i, C_{AC}\}, \{C_i, C_{AB}\}, \{C_{ABC}\}$</p>	<p>$1 * Q_3/Q_t + 3 * Q_1 * Q_2/Q_t$</p>	<p>Total = 7.019E-3</p> <table border="1"> <thead> <tr> <th>#</th> <th>Terms</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td># 1</td> <td>MOV-CC-C</td> <td>6.370E-06</td> </tr> <tr> <td></td> <td>A-RCCF-3-3-AB</td> <td></td> </tr> <tr> <td># 2</td> <td>MOV-CC-B</td> <td>6.370E-06</td> </tr> </tbody> </table>	#	Terms	Value	# 1	MOV-CC-C	6.370E-06		A-RCCF-3-3-AB		# 2	MOV-CC-B	6.370E-06												
#	Terms	Value																									
# 1	MOV-CC-C	6.370E-06																									
	A-RCCF-3-3-AB																										
# 2	MOV-CC-B	6.370E-06																									

Modification Type	Conditional Calculation	SAPHIRE Treatment	Result															
	$P(S A_T) = P(S \cap A_T) / P(A_T) =$ $(3/2) \alpha_1 Q_t \alpha_2 + \alpha_3$		<table> <tr> <td># 3</td> <td>A-RCCF-3-3-AC MOV-CC-A</td> <td>6.370E-06</td> </tr> <tr> <td># 4</td> <td>A-RCCF-3-3-BC A-RCCF-3-3-ABC</td> <td>7.0000E-03</td> </tr> </table>	# 3	A-RCCF-3-3-AC MOV-CC-A	6.370E-06	# 4	A-RCCF-3-3-BC A-RCCF-3-3-ABC	7.0000E-03									
# 3	A-RCCF-3-3-AC MOV-CC-A	6.370E-06																
# 4	A-RCCF-3-3-BC A-RCCF-3-3-ABC	7.0000E-03																
Two unknown failures	<p>Condition on two unknown failures, A_T and B_T</p> $S \cap (A_T \cap B_T) = \{A_i, C_{BC}\},$ $\{B_i, C_{AC}\}, \{C_i, C_{AB}\}, \{C_{ABC}\}$ $P(S A_T \cap B_T) = P(S \cap A_T \cap B_T) /$ $P(A_T \cap B_T) =$ $[(3/2) \alpha_1 Q_t \alpha_2 Q_t] / [\varphi] +$ $[\alpha_3 Q_t] / [\varphi]$ <p>where $\varphi = (\alpha_1 Q_t)^2 + \alpha_1 Q_t \alpha_2 Q_t +$ $(1/2) \alpha_2 Q_t + \alpha_3 Q_t$</p>	$1 * Q3 / \varphi +$ $3 * Q1 * Q2 / \varphi$	<p>Total = 4.850E-1</p> <table> <thead> <tr> <th>#</th> <th>Terms</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td># 1</td> <td>MOV-CC-C</td> <td>4.401E-04</td> </tr> <tr> <td># 2</td> <td>A-RCCF-3-3-AB MOV-CC-B</td> <td>4.401E-04</td> </tr> <tr> <td># 3</td> <td>A-RCCF-3-3-AC MOV-CC-A</td> <td>4.401E-04</td> </tr> <tr> <td># 4</td> <td>A-RCCF-3-3-BC A-RCCF-3-3-ABC</td> <td>4.8365E-1</td> </tr> </tbody> </table>	#	Terms	Value	# 1	MOV-CC-C	4.401E-04	# 2	A-RCCF-3-3-AB MOV-CC-B	4.401E-04	# 3	A-RCCF-3-3-AC MOV-CC-A	4.401E-04	# 4	A-RCCF-3-3-BC A-RCCF-3-3-ABC	4.8365E-1
#	Terms	Value																
# 1	MOV-CC-C	4.401E-04																
# 2	A-RCCF-3-3-AB MOV-CC-B	4.401E-04																
# 3	A-RCCF-3-3-AC MOV-CC-A	4.401E-04																
# 4	A-RCCF-3-3-BC A-RCCF-3-3-ABC	4.8365E-1																
Dependent failure	Condition on two (A and B) dependent failures, C_{AB} or C_{ABC}	$1 * Q1 * Q2 / (Q2 + Q3) +$ $1 * Q3 / (Q2 + Q3)$	<p>Total = 5.190E-1</p> <table> <thead> <tr> <th>#</th> <th>Terms</th> <th>Value</th> </tr> </thead> </table>	#	Terms	Value												
#	Terms	Value																

Modification Type	Conditional Calculation	SAPHIRE Treatment	Result									
	$S \cap (C_{AB} \text{ or } C_{ABC}) = \{C_i, C_{AB}\}, \{C_{ABC}\}$ $P(S C_{AB} \text{ or } C_{ABC}) = P(S \cap C_{AB} \text{ or } C_{ABC}) / P(C_{AB} \text{ or } C_{ABC}) =$ $(\alpha_1 Q_1 \alpha_2 Q_1 + \alpha_3 Q_1) / (\alpha_2 Q_1 + \alpha_3 Q_1)$		<table> <tr> <td># 1</td> <td>MOV-CC-C</td> <td>4.7185E-04</td> </tr> <tr> <td># 2</td> <td>A-RCCF-3-3-AB</td> <td>5.1852E-1</td> </tr> <tr> <td></td> <td>A-RCCF-3-3-ABC</td> <td></td> </tr> </table>	# 1	MOV-CC-C	4.7185E-04	# 2	A-RCCF-3-3-AB	5.1852E-1		A-RCCF-3-3-ABC	
# 1	MOV-CC-C	4.7185E-04										
# 2	A-RCCF-3-3-AB	5.1852E-1										
	A-RCCF-3-3-ABC											
Independent failure	Condition on an independent failure, A_i $S \cap A_i = \{A_i, C_{BC}\}$ $P(S A_i) = P(S \cap A_i) / P(A_i) =$ $(1/2) \alpha_2 Q_1$	$1 * Q_1 * Q_2 / Q_1$	Total = 6.5E-6 <table> <thead> <tr> <th>#</th> <th>Terms</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td># 1</td> <td>MOV-CC-A</td> <td>6.500E-06</td> </tr> <tr> <td></td> <td>A-RCCF-3-3-BC</td> <td></td> </tr> </tbody> </table>	#	Terms	Value	# 1	MOV-CC-A	6.500E-06		A-RCCF-3-3-BC	
#	Terms	Value										
# 1	MOV-CC-A	6.500E-06										
	A-RCCF-3-3-BC											
Two independent failures	Condition on two independent failures, A_i and B_i $S \cap (A_i \cap B_i) = \emptyset$ $P(S A_i \cap B_i) = 0.0$	<FALSE> In this case, there is no CCF contribution since there is only a single component remaining.	Total = 0 <table> <thead> <tr> <th>#</th> <th>Terms</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td># 1</td> <td><FALSE></td> <td>0.0</td> </tr> </tbody> </table>	#	Terms	Value	# 1	<FALSE>	0.0			
#	Terms	Value										
# 1	<FALSE>	0.0										

Modification Type	Conditional Calculation	SAPHIRE Treatment	Result
New Probability (higher 1E-2)	Redo the baseline calculation, but use the <i>smallest</i> value for Qt (in this case 1E-3) $P(S) = (3/2) \alpha_1 Q_t \alpha_2 Q_t + \alpha_3 Q_t$	1 * Q3 + 3 * Q1 * Q2	Total = 7.019E-6 # Terms Value # 1 MOV-CC-C 6.370E-09 A-RCCF-3-3-AB # 2 MOV-CC-B 6.370E-09 A-RCCF-3-3-AC # 3 MOV-CC-A 6.370E-09 A-RCCF-3-3-BC # 4 A-RCCF-3-3-ABC 7.000E-06
New Probability (lower 1E-4)	Redo the baseline calculation, but use the <i>smallest</i> value for Qt (in this case 1E-4) $P(S) = (3/2) \alpha_1 Q_t \alpha_2 Q_t + \alpha_3 Q_t$	1 * Q3 + 3 * Q1 * Q2	Total = 7.002E-7 # Terms Value # 1 MOV-CC-C 6.370E-11 A-RCCF-3-3-AB # 2 MOV-CC-B 6.370E-11 A-RCCF-3-3-AC # 3 MOV-CC-A 6.370E-11 A-RCCF-3-3-BC # 4 A-RCCF-3-3-ABC 7.000E-07
Test and Maintenance	Find new cut sets by ANDing with NOT an independent failure $/A_1$	1 * Q3 + 2 * Q1 * Q2	Total = 7.013E-6 # Terms Value

Modification Type	Conditional Calculation	SAPHIRE Treatment	Result									
	$S \cap /A_I = \{/A_I, B_I, C_{AC}\}, \{/A_I, C_I, C_{AB}\}, \{/A_I, C_{ABC}\}$ $P(S) = (2/2) \alpha_1 Q_t \alpha_2 Q_t + \alpha_3 Q_t$	Steps: 1. Condition on /AI 2. Remove all of the /AI parts of the remaining cut sets 3. Do not divide the remaining cut sets by any value.	<table> <tr> <td># 1</td> <td>MOV-CC-C</td> <td>6.3100E-09</td> </tr> <tr> <td># 2</td> <td>A-RCCF-3-3-AB MOV-CC-B</td> <td>6.3100E-09</td> </tr> <tr> <td># 3</td> <td>A-RCCF-3-3-AC A-RCCF-3-3-ABC</td> <td>7.0000E-06</td> </tr> </table>	# 1	MOV-CC-C	6.3100E-09	# 2	A-RCCF-3-3-AB MOV-CC-B	6.3100E-09	# 3	A-RCCF-3-3-AC A-RCCF-3-3-ABC	7.0000E-06
# 1	MOV-CC-C	6.3100E-09										
# 2	A-RCCF-3-3-AB MOV-CC-B	6.3100E-09										
# 3	A-RCCF-3-3-AC A-RCCF-3-3-ABC	7.0000E-06										
Not Failed	Condition on NOT failure /A _T $S \cap /A_T = \emptyset$ $P(S /A_I) = 0.0$	<FALSE> (if one component works, system is not failed in this example)	Total = 0 <table> <thead> <tr> <th>#</th> <th>Terms</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td># 1</td> <td><FALSE></td> <td>0.0</td> </tr> </tbody> </table>	#	Terms	Value	# 1	<FALSE>	0.0			
#	Terms	Value										
# 1	<FALSE>	0.0										
Ignore	Map the system down to two component group.	1 * Q2' Where Q2' = $(\alpha_2 + \alpha_3)Q_t$	Total = 2.000E-5 <table> <thead> <tr> <th>#</th> <th>Terms</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td># 1</td> <td>A-RCCF-3-3-BC</td> <td>2.000E-5</td> </tr> </tbody> </table>	#	Terms	Value	# 1	A-RCCF-3-3-BC	2.000E-5			
#	Terms	Value										
# 1	A-RCCF-3-3-BC	2.000E-5										
TRUE house event	This case will be treated as an unknown failure.											
Probability of 1.0	This case will be treated as an independent failure.											

For the second example, assume we have a three-component common-cause group where two components are needed for success (see Table B-2 for details). The Boolean expression for this system is:

$$S = (A_T \cap B_T) \cup (B_T \cap C_T) \cup (A_T \cap C_T)$$

The minimal cut sets and Basic Parameter Model (BPM) expression for this group are the following:

Cut Sets	BPM
{A _i ,B _i }	
{A _i ,C _i }	
{B _i ,C _i }	3 * Q1 ² +
{C _{AB} }	3 * Q2 +
{C _{AC} }	Q3
{C _{BC} }	
{C _{ABC} }	

B.3 Flag Set Adjustment for CCF Basic Events

An additional complication related to CCF events is related to sequence flag sets. Flag sets are a special type of change set which are applied to specified sequences after solving fault trees and event tree sequences. Flag sets will modify a basic event to be True, False, or Ignore.

When a flag set is applied to a sequence and one (or more) component is set to TRUE, system redundancy is reduced – however the associated CCF probability is not automatically adjusted in many analysis tools for the new common cause failure group (which would be one less if one component is set to TRUE). In SAPHIRE Version 8, CCF probabilities are automatically adjusted in the affected sequence when a flag set reduces redundancy. This capability removes the potential to underestimate CCF probabilities when using sequence flag sets. Note, however, that the CCF probability adjustment can be applied only for the RASP CCF method for basic events specified with calculation type R (not the compound CCF basic events which are of calculation type C). To invoke this capability in SAPHIRE, mark the checkbox under the Project → User Settings → RASP CCF option.

Table B-2. CCF Analysis Table for the three-component (2-of-3 success criteria) example

Note 1: $Q_t = A_t = 1E-3$, $\alpha_1 = 0.980$, $\alpha_2 = 0.0130$, $\alpha_3 = 0.0070$, assumed staggered testing model.

Note 2: The case of all independent failures $\{A_i, B_i, C_i\}$ is ignored in the table and the SAPHIRE CCF module.

Modification Type	Conditional Calculation	SAPHIRE Treatment	Result																		
Nominal	<p>Baseline calculation</p> <p>Cut sets</p> <p>$\{C_{BC}\}, \{C_{AC}\}, \{C_{AB}\}, \{C_{ABC}\}$</p> <p>$P(S) = (3/2) \alpha_2 Q_t + \alpha_3 Q_t$</p>	<p>No modification from baseline:</p> <p>$3 * Q_2 +$</p> <p>Q_3</p>	<p>Total = 2.650E-5</p> <table border="1"> <thead> <tr> <th>#</th> <th>Terms</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td># 1</td> <td>A-RCCF-3-3-AB</td> <td>6.500E-06</td> </tr> <tr> <td># 2</td> <td>A-RCCF-3-3-AC</td> <td>6.500E-06</td> </tr> <tr> <td># 3</td> <td>A-RCCF-3-3-BC</td> <td>6.500E-06</td> </tr> <tr> <td># 4</td> <td>A-RCCF-3-3-ABC</td> <td>7.000E-06</td> </tr> </tbody> </table>	#	Terms	Value	# 1	A-RCCF-3-3-AB	6.500E-06	# 2	A-RCCF-3-3-AC	6.500E-06	# 3	A-RCCF-3-3-BC	6.500E-06	# 4	A-RCCF-3-3-ABC	7.000E-06			
#	Terms	Value																			
# 1	A-RCCF-3-3-AB	6.500E-06																			
# 2	A-RCCF-3-3-AC	6.500E-06																			
# 3	A-RCCF-3-3-BC	6.500E-06																			
# 4	A-RCCF-3-3-ABC	7.000E-06																			
Unknown	<p>Condition on an unknown failure, A_T</p> <p>$S \cap A_T = \{A_i, C_{BC}\}, \{C_{AC}\}, \{C_{AB}\}, \{C_{ABC}\}$</p> <p>$P(S A_T) = P(S \cap A_T) / P(A_T) =$</p>	<p>$1 * Q_3 / Q_t +$</p> <p>$2 * Q_2 / Q_t +$</p> <p>$1 * Q_1 * Q_2 / Q_t$</p>	<p>Total = 2.001E-2</p> <table border="1"> <thead> <tr> <th>#</th> <th>Terms</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td># 1</td> <td>A-RCCF-3-3-AB</td> <td>6.500E-03</td> </tr> <tr> <td># 2</td> <td>A-RCCF-3-3-AC</td> <td>6.500E-03</td> </tr> <tr> <td># 3</td> <td>MOV-CC-A</td> <td>6.370E-06</td> </tr> <tr> <td></td> <td>A-RCCF-3-3-BC</td> <td></td> </tr> <tr> <td># 4</td> <td>A-RCCF-3-3-ABC</td> <td>7.0000E-03</td> </tr> </tbody> </table>	#	Terms	Value	# 1	A-RCCF-3-3-AB	6.500E-03	# 2	A-RCCF-3-3-AC	6.500E-03	# 3	MOV-CC-A	6.370E-06		A-RCCF-3-3-BC		# 4	A-RCCF-3-3-ABC	7.0000E-03
#	Terms	Value																			
# 1	A-RCCF-3-3-AB	6.500E-03																			
# 2	A-RCCF-3-3-AC	6.500E-03																			
# 3	MOV-CC-A	6.370E-06																			
	A-RCCF-3-3-BC																				
# 4	A-RCCF-3-3-ABC	7.0000E-03																			

Modification Type	Conditional Calculation	SAPHIRE Treatment	Result																					
	$(1/2) \alpha_1 Q_t \alpha_2 + (2/2) \alpha_2 + \alpha_3$																							
Two unknown failures	<p>Condition on two unknown failures, A_T and B_T</p> <p>$S \cap (A_T \cap B_T) = \{A_i, C_{BC}\}, \{B_i, C_{AC}\}, \{C_{AB}\}, \{C_{ABC}\}$</p> <p>$P(S A_T \cap B_T) = P(S \cap A_T \cap B_T) / P(A_T \cap B_T) =$</p> <p>$[(2/2) \alpha_1 Q_t \alpha_2 Q_t] / [\varphi] +$</p> <p>$[(1/2) \alpha_2 Q_t] / [\varphi] +$</p> <p>$[\alpha_3 Q_t] / [\varphi]$</p> <p>where $\varphi = (\alpha_1 Q_t)^2 + \alpha_1 Q_t \alpha_2 Q_t + (1/2) \alpha_2 Q_t + \alpha_3 Q_t$</p>	<p>$1 * Q3/\varphi +$</p> <p>$2 * Q1 * Q2/\varphi +$</p> <p>$1 * Q2/\varphi$</p>	<p>Total = 9.336E-1</p> <table border="1"> <thead> <tr> <th>#</th> <th>Terms</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td># 1</td> <td>A-RCCF-3-3-AB</td> <td>4.491E-01</td> </tr> <tr> <td># 2</td> <td>MOV-CC-B</td> <td>4.401E-04</td> </tr> <tr> <td></td> <td>A-RCCF-3-3-AC</td> <td></td> </tr> <tr> <td># 3</td> <td>MOV-CC-A</td> <td>4.401E-04</td> </tr> <tr> <td></td> <td>A-RCCF-3-3-BC</td> <td></td> </tr> <tr> <td># 4</td> <td>A-RCCF-3-3-ABC</td> <td>4.8365E-1</td> </tr> </tbody> </table> <p>Note that the 0.9336 value represents the system failure probability contribution from just CCF. The portion of the failure probability from "independent" terms is 0.06642.</p>	#	Terms	Value	# 1	A-RCCF-3-3-AB	4.491E-01	# 2	MOV-CC-B	4.401E-04		A-RCCF-3-3-AC		# 3	MOV-CC-A	4.401E-04		A-RCCF-3-3-BC		# 4	A-RCCF-3-3-ABC	4.8365E-1
#	Terms	Value																						
# 1	A-RCCF-3-3-AB	4.491E-01																						
# 2	MOV-CC-B	4.401E-04																						
	A-RCCF-3-3-AC																							
# 3	MOV-CC-A	4.401E-04																						
	A-RCCF-3-3-BC																							
# 4	A-RCCF-3-3-ABC	4.8365E-1																						
Dependent	Condition on multiple dependent failures, C_{AB} or C_{ABC}	<p>$1 * Q2/(Q2+Q3) +$</p> <p>$1 * Q3/(Q2+Q3)$</p>	<p>Total = 1.0000</p> <table border="1"> <thead> <tr> <th>#</th> <th>Terms</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td># 1</td> <td>A-RCCF-3-3-AB</td> <td>4.8148E-1</td> </tr> <tr> <td># 2</td> <td>A-RCCF-3-3-ABC</td> <td>5.1852E-1</td> </tr> </tbody> </table>	#	Terms	Value	# 1	A-RCCF-3-3-AB	4.8148E-1	# 2	A-RCCF-3-3-ABC	5.1852E-1												
#	Terms	Value																						
# 1	A-RCCF-3-3-AB	4.8148E-1																						
# 2	A-RCCF-3-3-ABC	5.1852E-1																						

Modification Type	Conditional Calculation	SAPHIRE Treatment	Result									
	$S \cap (C_{AB} \text{ or } C_{ABC}) = \{C_{AB}\}, \{C_{ABC}\}$ $P(S C_{AB} \text{ or } C_{ABC}) = P(S \cap C_{AB} \text{ or } C_{ABC}) / P(C_{AB} \text{ or } C_{ABC}) =$ $(\alpha_2 Q_t + \alpha_3 Q_t) / (\alpha_2 Q_t + \alpha_3 Q_t)$											
Independent	Condition on an independent failure, A_i $S \cap A_i = \{A_i, C_{BC}\}$ $P(S A_i) = P(S \cap A_i) / P(A_i) =$ $(1/2) \alpha_2 Q_t$	1 * Q1 * Q2/Q1	Total = 6.5E-6 <table border="0"> <thead> <tr> <th>#</th> <th>Terms</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td># 1</td> <td>MOV-CC-A</td> <td>6.500E-06</td> </tr> <tr> <td></td> <td>A-RCCF-3-3-BC</td> <td></td> </tr> </tbody> </table>	#	Terms	Value	# 1	MOV-CC-A	6.500E-06		A-RCCF-3-3-BC	
#	Terms	Value										
# 1	MOV-CC-A	6.500E-06										
	A-RCCF-3-3-BC											
Two independent failures	Condition on two independent failures, A_i and B_i $S \cap (A_i \cap B_i) = \emptyset$ $P(S A_i \cap B_i) = 0.0$	<FALSE> In this case, there is no CCF contribution since there are no components remaining.	Total = 0 <table border="0"> <thead> <tr> <th>#</th> <th>Terms</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td># 1</td> <td><FALSE></td> <td>0.0</td> </tr> </tbody> </table> <p>Note that the portion of the system failure probability due to independent terms is 1.0.</p>	#	Terms	Value	# 1	<FALSE>	0.0			
#	Terms	Value										
# 1	<FALSE>	0.0										
New Probability (higher 1E-2)	Redo the baseline calculation, but use the <i>smallest</i> value for Q_t (in this case 1E-3)	3 * Q2 + Q3	Total = 2.650E-5 <table border="0"> <thead> <tr> <th>#</th> <th>Terms</th> <th>Value</th> </tr> </thead> </table>	#	Terms	Value						
#	Terms	Value										

Modification Type	Conditional Calculation	SAPHIRE Treatment	Result															
	$P(S) = (3/2) \alpha_2 Q_t + \alpha_3 Q_t$		<table border="0"> <tr> <td># 1</td> <td>A-RCCF-3-3-AB</td> <td>6.500E-06</td> </tr> <tr> <td># 2</td> <td>A-RCCF-3-3-AC</td> <td>6.500E-06</td> </tr> <tr> <td># 3</td> <td>A-RCCF-3-3-BC</td> <td>6.500E-06</td> </tr> <tr> <td># 4</td> <td>A-RCCF-3-3-ABC</td> <td>7.000E-06</td> </tr> </table>	# 1	A-RCCF-3-3-AB	6.500E-06	# 2	A-RCCF-3-3-AC	6.500E-06	# 3	A-RCCF-3-3-BC	6.500E-06	# 4	A-RCCF-3-3-ABC	7.000E-06			
# 1	A-RCCF-3-3-AB	6.500E-06																
# 2	A-RCCF-3-3-AC	6.500E-06																
# 3	A-RCCF-3-3-BC	6.500E-06																
# 4	A-RCCF-3-3-ABC	7.000E-06																
New Probability (lower 1E-4)	Redo the baseline calculation, but use the <i>smallest</i> value for Q_t (in this case 1E-4) $P(S) = (3/2) \alpha_2 Q_t + \alpha_3 Q_t$	3 * Q2 + Q3	Total = 2.650E-6 <table border="0"> <tr> <td>#</td> <td>Terms</td> <td>Value</td> </tr> <tr> <td># 1</td> <td>A-RCCF-3-3-AB</td> <td>6.500E-07</td> </tr> <tr> <td># 2</td> <td>A-RCCF-3-3-AC</td> <td>6.500E-07</td> </tr> <tr> <td># 3</td> <td>A-RCCF-3-3-BC</td> <td>6.500E-07</td> </tr> <tr> <td># 4</td> <td>A-RCCF-3-3-ABC</td> <td>7.000E-07</td> </tr> </table>	#	Terms	Value	# 1	A-RCCF-3-3-AB	6.500E-07	# 2	A-RCCF-3-3-AC	6.500E-07	# 3	A-RCCF-3-3-BC	6.500E-07	# 4	A-RCCF-3-3-ABC	7.000E-07
#	Terms	Value																
# 1	A-RCCF-3-3-AB	6.500E-07																
# 2	A-RCCF-3-3-AC	6.500E-07																
# 3	A-RCCF-3-3-BC	6.500E-07																
# 4	A-RCCF-3-3-ABC	7.000E-07																
Test and Maintenance	Find new cut sets by ANDing with NOT an independent failure $/A_i$ $S \cap /A_i = \{/A_i, C_{BC}\}, \{/A_i, C_{AC}\}, \{/A_i, C_{AB}\}, \{/A_i, C_{ABC}\}$ $P(S) = (3/2) \alpha_2 Q_t + \alpha_3 Q_t$	1 * Q3 + 3 * Q2 Steps: 1. Condition on $/A_i$ 2. Remove all of the $/A_i$ parts of the remaining cut sets 3. Do not divide the cut sets by any value.	Total = 2.650E-5 <table border="0"> <tr> <td>#</td> <td>Terms</td> <td>Value</td> </tr> <tr> <td># 1</td> <td>A-RCCF-3-3-AB</td> <td>6.500E-06</td> </tr> <tr> <td># 2</td> <td>A-RCCF-3-3-AC</td> <td>6.500E-06</td> </tr> <tr> <td># 3</td> <td>A-RCCF-3-3-BC</td> <td>6.500E-06</td> </tr> <tr> <td># 4</td> <td>A-RCCF-3-3-ABC</td> <td>7.000E-06</td> </tr> </table>	#	Terms	Value	# 1	A-RCCF-3-3-AB	6.500E-06	# 2	A-RCCF-3-3-AC	6.500E-06	# 3	A-RCCF-3-3-BC	6.500E-06	# 4	A-RCCF-3-3-ABC	7.000E-06
#	Terms	Value																
# 1	A-RCCF-3-3-AB	6.500E-06																
# 2	A-RCCF-3-3-AC	6.500E-06																
# 3	A-RCCF-3-3-BC	6.500E-06																
# 4	A-RCCF-3-3-ABC	7.000E-06																
Not Failed	Condition on NOT failure $/A_T$ $S \cap /A_T = \{/A_i, /C_{AB}, /C_{AC}, /C_{ABC}, C_{BC}\}$	1 * Q2	Total = 6.5000E-6 <table border="0"> <tr> <td>#</td> <td>Terms</td> <td>Value</td> </tr> <tr> <td># 1</td> <td>/MOV-CC-A</td> <td>6.5000E-6</td> </tr> </table>	#	Terms	Value	# 1	/MOV-CC-A	6.5000E-6									
#	Terms	Value																
# 1	/MOV-CC-A	6.5000E-6																

Modification Type	Conditional Calculation	SAPHIRE Treatment	Result						
	$P(S A_T) = \varphi [(1/2)\alpha_2 Q_i] / \varphi$ <p>where $\varphi = (1 - \alpha_1 Q_i)(2)[1 - (1/2)\alpha_2 Q_i] (1 - \alpha_3 Q_i)$</p>		/A-RCCF-3-3-AB /A-RCCF-3-3-AC /A-RCCF-3-3-ABC A-RCCF-3-3-BC Note that the portion of the system failure probability due to independent terms is 9.604E-7. Also, the event A-RCCF-3-3-BC should be assigned the value of 6.5000E-6. The other terms (e.g., /MOV-CC-A) are equal to 1.0.						
Ignore	Map the system down to two component group.	<FALSE> In this case, there is no CCF contribution since only single failures remain (B or C).	Total = 0 <table> <thead> <tr> <th>#</th> <th>Terms</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td># 1</td> <td><FALSE></td> <td>0.0</td> </tr> </tbody> </table>	#	Terms	Value	# 1	<FALSE>	0.0
#	Terms	Value							
# 1	<FALSE>	0.0							
TRUE house event	This case will be treated as an unknown failure.								
Probability of 1.0	This case will be treated as an independent failure.								

Appendix C

Maximum Entropy Distribution

Appendix C

Maximum Entropy Distribution

C.1 Facts About the Maximum Entropy Distribution

C.1.1 Form of the Distribution

Consider a distribution with density f defined on a finite range $[a, b]$. The *entropy* of the distribution is defined as

$$-\int [\ln f(x)]f(x)dx . \tag{C-1}$$

This can be approximated by

$$-\sum (\ln p_i)p_i , \tag{C-2}$$

by dividing the x -axis into equally spaced intervals with midpoints x_i and width Δx , and letting $p_i = f(x_i)\Delta x$. The approximation can be made arbitrarily good by making Δx arbitrarily small, and the number of terms in the sum correspondingly large.

Suppose that the distribution is required to have mean μ . To find the values of p_i that maximize (C-2) subject to the constraints $\sum p_i = 1$ and $\sum x_i p_i = \mu$, use the method of Lagrange multipliers: differentiate

$$-\sum (\ln p_i)p_i + \lambda_1(\sum p_i - 1) + \lambda_2(\sum x_i p_i - \mu)$$

with respect to each p_i and with respect to λ_1 and λ_2 , and set each of these first derivatives to zero. This results in the original constraints, and equations of the form

$$-(p_i/p_i) - \ln p_i + \lambda_1 + \lambda_2 x_i = 0 .$$

These equations imply that

$$\ln p_i = \lambda_2 x_i + (\lambda_1 - 1)$$

for all i . This is equivalent to

$$p_i = (C \exp(\lambda_2 x_i))$$

for some constants C and λ_2 , which may depend on the width Δx . Therefore, Expression (C-1) is maximized, subject to the constraint $\int f(x)dx = 1$, by making

$$f(x) = ce^{\beta x}$$

for some constants c and β .

If β is zero, then f is constant, a uniform density. Consider now the case with $\beta \neq 0$. Because f integrates to 1, it follows that f must be of the form

$$f(x) = \beta e^{\beta x} / (e^{\beta b} - e^{\beta a}) \quad \text{for } \beta \neq 0 . \quad (\text{C-3})$$

Therefore, the cumulative distribution function is

$$F(x) = (e^{\beta x} - e^{\beta a}) / (e^{\beta b} - e^{\beta a}) \quad \text{for } \beta \neq 0 .$$

If β is negative, the distribution given by (C-3) is a truncated exponential distribution with parameter $-\beta$, a decreasing function of x . If β is positive, the density is of exponential form, an increasing function of x . If β is 0, the density is flat, neither increasing nor decreasing.

C.1.2 μ as a Function of β

First, if β is zero, then f is a uniform density, so the mean is the midpoint of the range:

$$\mu = (a + b)/2 \quad \text{for } \beta=0 . \quad (\text{C-4})$$

If instead the density is given by Equation (B-3), the mean is $\mu = \int xf(x)dx$. To evaluate this, integrate $\int xf(x)dx$ by parts, obtaining

$$\mu = (be^{\beta b} - ae^{\beta a}) / (e^{\beta b} - e^{\beta a}) - 1/\beta , \quad \text{for } \beta \neq 0 . \quad (\text{C-5})$$

C.1.3 Continuity of μ as a Function of β

Equations (C-4) and (C-5) define μ as a function of β for all β . To show that the function is continuous at $\beta = 0$, rewrite (B-5) as a single fraction. Now use Taylor's Theorem with the remainder term to write

$$e^{a\beta} = 1 + a\beta + \frac{1}{2} a^2 \beta^2 + \beta^3 c(\beta; a)$$

where $c(\beta; a)$ is bounded and continuous as $\beta \rightarrow 0$. Write $e^{b\beta}$ similarly, work out the algebra, and cancel terms to show that the limit of (B-5) as $\beta \rightarrow 0$ is (C-4). Therefore, Equations (C-4) and (C-5) together define μ as a continuous function of β .

C.1.4 Monotonicity of μ as a Function of β

The numerical problem for software like SAPHIRE is to solve Equation (C-5) for β in terms of μ . Before developing a numerical method, we first show some facts about the relation between β and μ , culminating with the fact that μ is a monotone increasing function of β .

First, let us use a more convenient parameterization. Let c (for "center") be defined by

$$c = (a + b)/2$$

and let r (for "range") be defined by

$$r = (b - a)/2,$$

so that $a = c - r$, and $b = c + r$. In this parameterization, Equation (B-5) becomes

$$\mu = c + rg(\beta) ,$$

with

$$g(\beta) = [(e^{\beta r} + e^{-\beta r}) / (e^{\beta r} - e^{-\beta r}) - 1 / \beta r] \text{ for } \beta \neq 0 \tag{C-6}$$

and $g(\beta)$ defined by continuity at $\beta = 0$.

From Equation (C-6) it is clear that $g(-\beta) = -g(\beta)$, that is, g is an odd function. By continuity, therefore, $g(0) = 0$. This value of 0 could also have been obtained directly from Equation (C-4). Note also that $g(\beta) \rightarrow 1$ as $\beta \rightarrow \infty$, and similarly $g(\beta) \rightarrow -1$ as $\beta \rightarrow -\infty$. These translate to the following facts about μ .

$$\mu(\beta) \rightarrow c - r = a \text{ as } \beta \rightarrow -\infty$$

$$\mu(0) = c$$

$$\mu(\beta) \rightarrow c + r = b \text{ as } \beta \rightarrow \infty .$$

The first and third cases correspond to distributions that are as asymmetrical as possible, and the second case, with $\beta = 0$, corresponds to a uniform distribution. Therefore, as β ranges over the real line μ attains any value between the minimum possible value of a and the maximum possible value of b . It remains to be shown that μ is monotone in β , so that every value of μ corresponds

to only one value of β . It is enough to show that μ is monotone in β for $\beta > 0$; the result for $\beta < 0$ follows from the antisymmetry of g . To do this, we calculate the first derivative of μ , and examine it for $\beta > 0$.

To reduce the number of occurrences of β in the expression for μ , for $\beta > 0$ multiply the numerator and denominator in the large fraction in Equation (C-6) by $e^{\beta r}$, resulting in

$$\begin{aligned}\mu &= c + rg(\beta) \\ &= c + r[(e^{2\beta r} + 1)/(e^{2\beta r} - 1) - 1/\beta r] \\ &= c + r[(e^\gamma + 1)/(e^\gamma - 1) - 2/\gamma]\end{aligned}\tag{C-7}$$

with γ defined as $2\beta r$. Therefore, by the usual rules for differentiation and a bit of algebra,

$$(\partial\mu/\partial\gamma) = 2r[-e^\gamma/(e^\gamma-1)^2 + 1/\gamma^2],$$

and

$$\begin{aligned}(\partial\mu/\partial\beta) &= (\partial\mu/\partial\gamma)\times(\partial\gamma/\partial\beta) \\ &= 4r^2[-e^\gamma/(e^\gamma-1)^2 + 1/\gamma^2].\end{aligned}\tag{C-8}$$

It must be shown that Expression (C-8) is positive for $\gamma > 0$. This is equivalent to showing that

$$(e^\gamma - 1)^2 > \gamma^2 e^\gamma,$$

which is equivalent, when $\gamma > 0$, to

$$e^\gamma - 1 > \gamma e^{\gamma/2}.$$

Expand both sides in their Taylor series. The inequality is equivalent to

$$\sum_{j=1}^{\infty} \frac{\gamma^j}{j!} > \gamma \sum_{i=0}^{\infty} \frac{\gamma^i}{2^i i!} = \sum_{j=1}^{\infty} \frac{\gamma^j}{2^{j-1} (j-1)!}$$

The corresponding terms in the leftmost and rightmost summations are equal if $j \leq 2$, while each term on the left is larger than the term on the right if $j > 2$ and $\gamma > 0$. Therefore the inequality is true, and equation (C-8) is positive for $\gamma > 0$, and μ is monotone increasing in β , as was to be shown.

C.2 Method for Finding β Corresponding to μ

C.2.1 Conceptual Method

The monotonicity proved above suggests the following natural method for finding the value of β corresponding to a specified μ . Suppose, for example, that the specified μ is less than c . Then β must be less than 0. Beginning with $\beta_0 = 0$, try successively smaller values (values farther to the left of 0) of β . Call them β_1, β_2 , and so forth, and let μ_i be the value from Equation (C-5) corresponding to β_i . Let μ_k be the first such value found that is less than the required μ . Then the required β must be in the interval from β_k to β_{k-1} . Try a value between β_k and β_{k-1} , and decide whether it is too large or too small by whether the corresponding μ is too large or too small. Keep repeating this step, each time shrinking the interval in which the required β must lie, until β is determined to the desired accuracy. The formal implementation of the algorithm for shrinking the interval is called the "method of false position" and is discussed in books on numerical analysis.

As implemented in SAPHIRE, the parameter μ is considered primary, the parameter that the user enters, and β is considered an intermediate means to obtain μ . Therefore the search for β continues until $\mu(\beta)$ differs from the originally input μ by less than $r \times 10^{-5}$. This is considered at least as accurate as the normal user's knowledge of μ .

In rare cases, r can be so small that $r \times 10^{-5}$ would underflow. In these cases the search continues until the interval for β has relative length ($= \text{length}/\beta$) $< 5 \times 10^{-5}$ or absolute length $< (5 \times 10^{-8})/r$. This results in possible inaccuracy in β of at most 1 in the fifth digit, except when $|\beta r| < 1 \times 10^{-3}$. This is considered sufficient for the intended applications.

C.2.2 Numerical Computation of μ in Terms of β

The computation of μ uses Expression (C-7), with $\gamma = 2\beta r$. When γ approaches 0, the expressions in the two denominators in Expression (C-7) both approach 0, and the computer is unable to evaluate Expression (C-7) as it is written. Therefore, an approximation is used when γ is near 0, based on the Taylor series for the exponential function:

$$e^\gamma = 1 + \gamma + \gamma^2/2 + \gamma^3/6 + \gamma^4/24 + \dots$$

$$= \frac{\gamma(e^\gamma + 1) - 2(e^\gamma - 1)}{\gamma(e^\gamma - 1)} - \frac{\gamma^3/6 + \gamma^4/12 + 3\gamma^5/120}{\gamma(\gamma + \gamma^2/2 + \gamma^3/6)} = \frac{\gamma}{6} \left[\frac{1 + \gamma/2 + 3\gamma^2/20}{1 + \gamma/2 + \gamma^2/6} \right].$$

In this way the expression in square brackets in Expression (C-7) can be rewritten as follows.

$$\frac{e^\gamma + 1}{e^\gamma - 1} - \frac{2}{\gamma}$$

If, to the numerical accuracy of the computer, $1 + \gamma^2/6$ is equal to 1, then the numerator and the denominator in the square brackets are each equal to $1 + \gamma/2$, and the entire expression equals $\gamma/6$. Therefore, SAPHIRE evaluates μ as

$$c + r\gamma/6$$

if γ is so small that $1 + \gamma^2/6 = 1$, and evaluates μ by Expression (C-7) otherwise.

BIBLIOGRAPHIC DATA SHEET

(See instructions on the reverse)

NUREG/CR-7039, Vol. 2
INL/EXT-09-17010

2. TITLE AND SUBTITLE

Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Version 8

Volume 2: Technical Reference

3. DATE REPORT PUBLISHED

MONTH

YEAR

June

2011

4. FIN OR GRANT NUMBER

N6423

5. AUTHOR(S)

C. L. Smith, S. T. Wood, W. J. Galyean, J. A. Schroeder, M. B. Sattison

6. TYPE OF REPORT

Technical

7. PERIOD COVERED (Inclusive Dates)

8. PERFORMING ORGANIZATION - NAME AND ADDRESS (If NRC, provide Division, Office or Region, U.S. Nuclear Regulatory Commission, and mailing address; if contractor, provide name and mailing address.)

Idaho National Laboratory
Battelle Energy Alliance
P.O. Box 1625
Idaho Falls, ID 83415-3850

9. SPONSORING ORGANIZATION - NAME AND ADDRESS (If NRC, type "Same as above", if contractor, provide NRC Division, Office or Region, U.S. Nuclear Regulatory Commission, and mailing address.)

Division of Risk Analysis
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001

10. SUPPLEMENTARY NOTES

D. O'Neal, NRC Project Manager

11. ABSTRACT (200 words or less)

SAPHIRE 8 is a software application developed for performing a complete probabilistic risk assessment (PRA) using a personal computer (PC) running the Microsoft Windows operating system. This report summarizes the fundamental mathematical concepts of sets and logic, fault trees, and probability. This volume then describes the algorithms used to construct a fault tree and to obtain the minimal cut sets. It gives the formulas used to obtain the probability of the top event from the minimal cut sets, and the formulas for probabilities that apply for various assumptions concerning reparability and mission time. It defines the measures of basic event importance that SAPHIRE 8 can calculate. This volume gives an overview of uncertainty analysis using simple Monte Carlo sampling or Latin Hypercube sampling, and states the algorithms used by this program to generate random basic event probabilities from various distributions. Also covered are enhance capabilities such as seismic analysis, cut set "recovery," end state manipulation, and use of "compound events."

12. KEY WORDS/DESCRIPTORS (List words or phrases that will assist researchers in locating the report.)

SAPHIRE 8, software, reliability, risk, safety, PRA, fault tree, event tree

13. AVAILABILITY STATEMENT

unlimited

14. SECURITY CLASSIFICATION

(This Page)

unclassified

(This Report)

unclassified

15. NUMBER OF PAGES

16. PRICE



Federal Recycling Program



UNITED STATES
NUCLEAR REGULATORY COMMISSION
WASHINGTON, DC 20555-0001

OFFICIAL BUSINESS