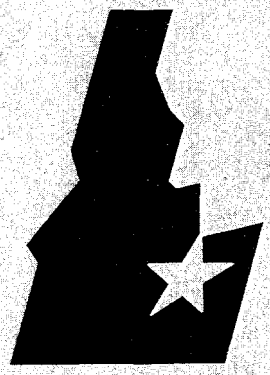


Idaho
National
Engineering
Laboratory

Managed
by the U.S.
Department
of Energy

EGG-SSRE-8137
January 1989

TECHNICAL REPORT

MIRAP, MICROCOMPUTER RELIABILITY ANALYSIS PROGRAM

J. N. T. Jehee

Prepared for the
U.S. NUCLEAR REGULATORY COMMISSION

EG&G idaho

Work performed under
DOE Contract
No. DE-AC07-76ID01570

EGG-SSRE-8137

EGG-SSRE--8137

DE89 009749

MIRAP, MICROCOMPUTER RELIABILITY ANALYSIS PROGRAM

J. N. T. Jehee

Published January 1989

EG&G Idaho, Inc.
Idaho Falls, ID 83415

Prepared under an agreement by
the United States Nuclear Regulatory Commission
and the
Netherlands Energy Research Foundation
through DOE Contract No. DE-AC07-76ID01570

MASTER

ABSTRACT

A program for a microcomputer is outlined that can determine minimal cut sets from a specified fault tree logic. The speed and memory limitations of the microcomputers on which the program is implemented (Atari ST and IBM) are addressed by reducing the fault tree's size and by storing the cut set data on disk. Extensive well proven fault tree restructuring techniques, such as the identification of sibling events and of independent gate events, reduces the fault tree's size but does not alter its logic. New methods are used for the Boolean reduction of the fault tree logic. Special criteria for combining events in the 'AND' and 'OR' logic avoid the creation of many subsuming cut sets which all would cancel out due to existing cut sets. Figures and tables illustrate these methods.

SUMMARY

A personal computer (PC) based, fault tree processing computer program is outlined that uses both new and existing data processing and management techniques. These techniques provide the program with the capability to process relatively large fault trees in the memory limited environment of the PC and in an expeditious manner.

The processing begins by restructuring the fault tree prior to Boolean reduction. This is done in two phases for maximum efficiency. The first phase, which is fast executing, entails: 1) compressing identical gate type, 2) combining events that always appear together, 3) naming gates with identical input the same, and 4) promoting events common to all the inputs to a specific 'AND' gate to their highest level. The second, more time consuming restructuring phase comprises: 1) removing events that are subsumed by events higher in the tree and 2) identifying independent subtrees.

After the restructuring is completed the Boolean reduction is performed by storing the intermediate cut set information on a disk (floppy, hard or virtual) in a Direct Access File. In this phase, the cut sets are maintained in terms of independent subtrees and pseudo-gates, which saves considerable space. Bitmaps and a binary data structure are used to allow selective searches during the reduction, rather than requiring a scan of the complete list of events. The actual cut set generation is done using the bottom-up method and program, and although it does not allow 'NOT' events, it does remove any user-defined mutually exclusive event combinations.

The final cut set list is displayed in a binned format that lets any 'outliers' to be easily recognized. The bins are defined by a matrix of the cut set order and probability (or frequency). The number of cut sets, their frequency, and their contribution to the total are all displayed.

MIRAP consists of approximately 6000 lines of code and is written in FORTRAN 77. The program requires the fault tree to be in alphanumeric form, which can be generated using an ordinary text editor.

ACKNOWLEDGMENT

This study was carried out during the author's assignment at the Idaho National Engineering Laboratory (INEL) under an agreement by the United States Nuclear Regulatory Commission (USNRC) and the Netherlands Energy Research Foundation (Energieonderzoek Centrum Nederland, ECN). These organizations are acknowledged for supporting this study. The author is particularly indebted to ECN employees Mr. H. J. van Grol for managerial support, Dr. K. Terpstra for his introduction to reliability analysis, Mr. E. van der Goot (former ECN employee) for advise on informatics, and Mr. G. van Driel for commenting this report, and INEL/EG&G employees Mr. D. L. Batt, Mr. R. J. Dallman for managerial support, Mr. N. G. Cathey, Mr. W. J. Galyean, Mr. D. W. Stillwell for comments on the applicational aspects, and finally Mr. B. W. Dixon, and Mr. K. D. Russell for commenting the applied methods.

MIRAP, MICROCOMPUTER RELIABILITY ANALYSIS PROGRAM

J. N. T. Jehee

CONTENTS

1. INTRODUCTION

1.1. General

Systems potentially capable of endangering the lives of many people, or of destructing large sums of money require a safe operation. One method to assess this safe operation is analyzing all the combinations of events, such as component failure states and human errors, which if present and/or occurred simultaneously would cause an undesirable event to occur. This list of minimal combinations is with complex systems usually obtained from a system logic model or a fault tree by means of a computer.

In this report a computer program is outlined that determines this list of minimal combinations, or minimal cut sets, by using a microcomputer. The limited memory and speed of these systems require special techniques to be applied. The Boolean cancellation of the cut sets in the final process stage is strongly enhanced by reducing the fault tree size in the early process. Existing and new techniques are implemented in this preprocessing stage. The Boolean reduction uses a direct search for specific cut sets only as opposed to scanning a complete list. Memory requirements to execute the program are reduced by storing the intermediate information regarding e.g. cut sets and event references, on disk. Simple word processor editing skills are sufficient for preparing the input file. Extensive error checking procedures are incorporated in the program.

1.2. Fault Tree Restructuring

Restructuring the fault tree prior to the complete Boolean reduction is a crucial step in obtaining an efficient cut set determination process. In this restructuring only logic criteria are applied which leaves the tree logically equivalent to the original tree. Probabilistic reduction is only considered in the cut set determination to decrease the size of the cut set list.

The applied concepts for restructuring the tree are classified into two groups with respect to their speed of execution, viz.:

- a group of relatively fast executing restructuring methods containing:

  - o compressing cascading gate events of identical type,
  - o combining events which as siblings always appear together in the tree,
  - o naming identical gate events the same, and
  - o removing first order input events from the 'AND' gate input list;

- some more time consuming methods, viz.:

  - o removing subsuming events based on first order events closer to the top event, and
  - o identifying all the independent gate events.

The relatively fast executing methods are carried out first in an iterative manner. When no further reduction can be achieved, the more time consuming methods are addressed. If subsuming events are identified and removed, the fast executing methods are applied another time. When no subsuming events are found, the independent gate events are searched for at last.

1.3. Boolean Reduction

The memory requirements and the execution time for the cut set determination procedure are significantly reduced by restructuring and simplifying the original fault tree. However the number of cut sets can still be too large to fit in the computer's memory. Therefore special software techniques and program concepts facilitate the handling of sizeable cut set lists.

The software techniques concern mainly the use of disks for storage and retrieval of cut set data. Hard, floppy, or - if the computer's memory is large enough - 'Random Access Memory (RAM)' disks can be used. A large cache memory is used to enhance the retrieval efficiency by limiting the

frequency of disk access. The cut sets are stored in binary data structures on Direct Access Files. These binary data structures enhance efficient search procedures since ideally it would split its searchable list in half at each step, requiring maximum only some log n steps for a list with n cut sets. This opposed to a sequential data structure requiring n/2 steps in average.

The selected program concepts include direct searches for specific cut sets and subsets only, and a bottom-up fault tree processing procedure. These as opposed to other feasible methods as searching a whole list for matching or subsuming cut sets stored in a bit representation, see e.g. Reference[1], and a top-down approach. To further discriminate the searches for specific cut sets or subsets, bitmaps are used showing the existence of events in the concerned lists. While applying the searches the cut set list for an 'OR' gate is assembled. The processing of an 'AND' gate requires an intermediate step to ensure the removal of all the subsuming cut sets. User-defined or default cut-off limits on order (max. 16) and probability or frequency affect the logical combination of events in the 'AND' gates. Although 'NOT' gates and complementary events cannot be processed, cut sets with user-defined mutually exclusive events can be deleted from the final list.

1.4. Computer Program

The computer program is implemented on an IBM and an Atari ST mini-computer and consists of approximately 6000 Lines of Code (6 KLOC) standard FORTRAN 77. Although hard disks and extended memory enhance the speed of the program execution considerably, only a mathematical co-processor for the IBM is essential.

The input file consists of ASCII characters and comprises the fault tree, and possibly the failure data, the cut-off criteria, and the list with mutually exclusive events. The input file can be created by users with regular word processor skills. Many input checks are performed to aid the

user and to prevent erroneous results. Error and warning messages address:

- wrong or suspicious input specifications such as:

    o case significance, which is prone to errors and therefor all lower case characters are transformed to upper case to effectively neglect case significance,

    o multiple definition of the same gate, which will cause a warning message if the gate is defined identically, and an error message if the definitions differ,

    o fault trees possessing loop structures, which are trees containing gate events that ultimately reference itself, this includes the special loop structure of a tree without an identifiable top, and

    o suspicious failure data definitions, which generate warning messages only for a large variety of conditions like multiple failure data definitions, failure data for gate events, no failure data for basic events, and data larger than 1 or negative;

- violations of the program's limits such as:

    o event names longer than 20 characters,
    o more than 40 events on one input line, and
    o more than 1500 different basic and gate events in the input file.

Although efficient methods have been applied to increase the processing speed of the program, large problems can still take a considerable amount of time. Therefore the program possesses a 'walk-away capability' meaning that after defining the in- and output files and setting the program control parameters no interaction with the program is required until it is finished.

2. DEFINITIONS

This section gives the definition of the used terminology in this document. For a more elaborate list of definitions, the reader is referred to [3].

A basic event is either a component or human fault that will not be developed in more detail. The choice of basic events should be commensurate with the required resolution of the analysis, e.g. for assessing the reliability of a power plant, the diesel generator itself without its interfacing systems can be regarded as a single basic event, while for comparing different diesel generator designs many detailed basic events relating to cooling and lubricating the engine and to its fuel supply can be distinguished.

Children of gate events are the events underneath it, and apply to gate events only.

The domain of a gate event comprises of a list of all the basic and gate events underneath it. This list only registers the occurrence of events, not the frequency of its occurrence and is stored in a so-called 'bitmap', viz.:

events	1 2 3 4 5 6 7 8 9 1500
present yes or no?	y n n y y n y n n etc.
bitmap	1 0 0 1 1 0 1 0 0 etc.

For 1500 events 1500 bits are required which translates into 1500/16 is 94 two-byte words.

Independent gate events or subtrees are parts of the fault tree that can be evaluated separately since all of its constituent events exclusively occur under the independent gate. For the boolean reduction of the remaining part of the fault tree these independent gate events are considered as basic events.

A gate event combines two or more basic or gate events according to its associated logic. Two types of gates are used, viz. an 'OR' and an 'AND' gate. Within the terminology an 'OR' gate is true or occurs if either one of its input is true, while an 'AND' gate occurs if all of its inputs occur. More special logic combinations such as exclusive 'OR' gates and priority 'AND' gates as discussed in [3] are beyond the scope of this paper.

Parents of either gate or basic events refer to the gate events above them.

A pseudo-gate event is a simple independent subtree consisting of sibling events which occur once as input to this subtree only. Pseudo-gate events are identified as such in the restructuring part and are treated as basic events in the boolean reduction.

A reverse reference list of basic and gate events contains all the gate events which has the considered event as input.

A sibling event is a basic event that share its parent gate(s) with other siblings from his group. A group of sibling events can only share parent gates of one type of gate logic, either 'AND' or 'OR'. After a group of sibling events is identified, a pseudo-gate is created with these sibling events as input.

A subsuming cut set is a cut set which is included in another cut set, e.g. cut set 'A, B, C' can be deleted since it is included in cut set 'A, B'. A subsuming event in the fault tree is an event which will result in one or more subsuming cut sets and can thus be deleted from the fault tree.

3. RESTRUCTURING THE FAULT TREE

3.1. Restructuring Concepts

As mentioned in the introduction the restructuring of the fault tree is a crucial step in simplifying the logic and obtaining an efficient cut sets cancellation process. Existing codes apply many of the discussed methods, however the beneficial effect of a concerted action of the methods, as outlined here, is not fully explored.

The restructuring methods are derived from logic criteria, probabilistic criteria are presently not applied in the fault tree restructuring. This implies that the tree remains logically equivalent to

the original tree. Probabilistic reduction is only an option in diminishing the size of the problem in the cut set determination process by using cut-off criteria for order and probability or frequency.

Two groups of restructuring methods are identified with respect to their required amount of execution time. One group of relatively fast executing methods is addressed first in an iterative manner. Only after these methods fail to achieve any further simplification more time consuming methods are applied. An interactive option allows an experienced user to modify this sequence.

The obtained reduction by restructuring the tree depends strongly on characteristics of the original tree. Methods which perform well for one tree might not be of any value for another. No general classification of trees is attempted, however a general qualification of the methods is given.

3.2. Compressing Cascading Gates

Cascading gate events of the same type or gates with one child only can be compressed without changing the logic of the tree. This method is demonstrated on the tree of Figure 3-1 resulting in Figure 3-2 fault tree. This method by itself generally does not reduce the tree, the number of levels under the top or the depth of the tree is reduced at the expense of increasing its width. However this procedure facilitates the identification of sibling events and of first order inputs to 'AND' gates.

3.3. Creating Pseudo-gates of Sibling Events

Creating pseudo-gates of sibling events as is shown in Figure 3-3 is generally the most powerful of the presented reduction techniques, see in [2]. Three pseudo-'OR'gates, each comprising two basic events, combined in a cut set expand to eight cut sets. Especially when the resolution of the system representation in the tree is large, many pseudo-gates can be identified. Pseudo-gates can contain: basic events, independent gates, and previously identified pseudo-gates.

The detection of pseudo-gates requires a reverse reference list for each potential candidate. To avoid comparing the reverse references of a large number of events, the sum of the assigned event numbers of the reverse referenced event is stored in a list and ranked. Only those candidates with the identical reverse reference number are further analyzed on pseudo-gate characteristics. Obviously only those sibling events qualify which are combined in the same logic, either all in 'AND' or in 'OR' gates.

3.4. Naming Identical Gates the Same

This method requires no further explanation. Identical gate events are found which are either specified in the input or formed in the restructuring process. Detecting similar gates is performed analog to the sibling events while using the sum of assigned events numbers of the gate input events instead.

3.5. Removing First Order Inputs to 'AND' Gates

Either the user-defined fault tree or the restructured tree can possess 'AND' gates with one or more children which are first order event(s) for that gate. The procedure addressed here will introduce an intermediate 'OR' gate with these first order event(s) and the modified 'AND' gate as inputs, see basic event H in Figure 3-4. A special case is when only one or more first order cut sets to the 'AND' gate remain in which case the modified 'AND' gate is deleted. Events C and D in Figure 3-4 qualify for this condition since modifying the 'AND' gate by removing events C and D results in an 'empty' 'OR' gate and consequently in an 'empty' set for the 'AND' gate.

An appropriate renaming of the modified 'OR' gates is performed by this program module since 'OR' gates could be used at other branches in the tree which do not qualify for this reduction. An interesting result of this 'promoting' first order inputs to 'AND' gates is that first order events for the top event will boil up to under the top of the tree assuming that the option for compressing cascading events is selected.

Although it is acknowledged that an analog procedure could be followed for 'AND' gates under an 'OR' gate with identical input, see Figure 3-5, presently this situation is left unchanged.

3.6. Removing First Order Subsuming Events

If the child event of a particular gate also occurs as input to a gate in the same branch further from the top event, this latter input of the child event can be changed, based on rules regarding the tree logic. If the concerned gate or basic event is a child of an 'AND' gate, the input of this event to other 'AND' gates downward in the branch can be removed, while the 'OR' gates with this event as input can be deleted completely as shown in Figure 3-6. If the concerned event is a child of an 'OR' gate then analogously the inputs of this event to 'OR' gates can be removed and 'AND' gates can be deleted completely. Since the affected gates, see Figure 3-6, can also be used in other branches of the tree which do not qualify for this reduction, only those gates are modified which in all cases qualify for this reduction. Although a fairly complex renaming for the affected gates could be used to apply this method also for partial modifications of the tree, presently this is not implemented since it requires additional criteria for large fault trees in order to prevent exceeding the maximum number of events.

The previously established bitmaps of the domains of the gate events and its reverse references are used to construct an efficient algorithm. After finishing this option, another pass is made through the four simpler methods.

3.7. Identifying all the Independent Gates

Independent gate events are subtrees which do not have events in common with the rest of the tree and can for this reason be evaluated separately. Existing independent gate events allow the division of the problem into a number of smaller sized subproblems with more attractive properties. Either straight inserting the subtree's cut sets in the final result or simply assigning the failure frequency of the independent subtree to the independent gate in the final cut set list will yield responsibility. The

minimal cut set list expressed in the original events, or the top event's failure frequency. The separate treatment of all the independent gate event relieves the program from all the events in those independent subtree which otherwise should have been processed until the end.

The applied fault tree construction techniques, or the 'school' of the analysts, contribute greatly to the inclusion of independent subtrees, and of the previously discussed pseudo-gate events. Especially the choice between using large fault trees and small event sequences, or small fault trees and large event sequences influence the characteristics of the fault tree. The latter method will most likely treat larger independent subtrees separately as events in the event sequences.

Not every independent subtree is as easily recognized. Although the applied method requires more time than those previously described, it recognizes all the independent subtrees, which can considerably save execution time in the final cut set cancellation process. First bitmaps for all the gate events are generated representing the occurrence of all the gate and basic events in the inputs of the concerned gate and of its children, grandchildren, etc., including the gate itself. A specific gate event is independent if the reverse references of all its bitmap events are equal to or a subset from that particular gate bitmap, see Figure 3-7. This method identifies also higher order sub-subtrees.

4. DETERMINING THE MINIMAL CUT SETS

4.1. Handling Cut Set Data

Since the memory of microcomputers is limited and the number of cut sets can be quite large, the cut sets are stored on disk (floppy, hard, or memory 'RAM' disks) by using a Direct Access file. The cut set lists with variable order cut sets are stored on this file. Apart from this the cut sets are expressed in terms of the non-expanded events, such as independent gates and pseudo-gates which saves a considerable amount of space.

Only selective searches for specific cut sets are required as opposed to scanning a complete cut set list. To enhance the efficiency of this process unnecessary searches are prevented at an as early as practical stage and the data structure of the cut set list facilitates a fast search procedure. This is accomplished by using bitmaps for presenting the occurrence of events in lists and by using a binary data structure for the cut set list organization, see annex A.

The cut set determination is a so-called bottom-up process, i.e., the top event cut sets are obtained by starting at the end of the branches and working consecutively to the top of the tree. To be able to substitute the subtree probabilities into the final tree, the independent subtrees are processed first.

In the author's opinion the applied methods are better elucidated with characteristic examples for the 'OR' and 'AND' logic than with an elaborated mathematical description. To this end lists with selected cut sets are logically combined showing examples of all the discussed techniques. The selected cut sets are supposed to be originated from processing lower region gates which are not discussed.

The next presentation of the 'OR' and 'AND' gate logic closely follows the program algorithms, which reduce the combination of more than two cut set lists to multiple combinations of two cut set lists. The simpler 'OR' gate logic is presented first in section 4.2. The 'AND' gate logic is given in section 4.3 and addresses especially the creation of large lists with non-minimal or subsuming cut sets. Section 4.4 describes the treatment of mutually exclusive events, which uses similar techniques as the 'AND' and 'OR' logic combinations. Finally, the elaborate presentation of the cut sets is presented in section 4.5.

4.2. Processing 'OR' Gates

The 'OR' gate logic requires less steps than the 'AND' gate logic and is therefor described first. As mentioned above any combination of more than two lists can be reduced to multiple combinations of two lists. Basic events will be treated as a cut set list with only one first order cut set. Table IV-1 shows two cut set lists A and B that will be combined in an 'OR'

logic and that are obtained by processing gates 1 and 2. These lists consist of selected cut sets which can illustrate all the characteristics of this process. These lists originate from processing lower region gates in the tree starting with gates with basic events as input only. A basic event in this concept is treated analog as a gate event with an elaborate cut set list containing one cut set of order one, viz. the concerned basic event. The precise creation of lists A and B is not relevant for the discussion in this section.

If list A and B do not have common events, then the lists are simply added, otherwise the cut sets are processed separately and added to the final list if they qualify. First each cut set of list A is processed and second the remaining cut sets of list B. Table IV-1 shows two lists with specially selected cut sets for illustrating the described mechanisms.

Each cut set of list A is compared with list B with respect to consecutively:

1) its intersection of the cut set with the events in the domain of list B,
2) the existence of an identical cut set in list B, and
3) the existence of any subset of the cut set considered in List B.

Based on the results of the above mentioned comparisons the following actions are taken, respectively:

1) if no intersection exists this cut set is placed in the final list,
2) if an identical cut set exists in list B this cut set is placed in the final list and the identical cut set in list B is temporarily incapacitated,
3) if a subset of this cut set exists in list B, then this cut set is not added to the final list, and
4) finally if no identical cut set or subset is found, this cut set is added to the final list.

Table IV-1 shows examples of the above mentioned procedures. Processing list B is essentially the same as processing list A, except for the search of identical cut sets since they are incapacitated already. The final list will be referenced by the concerned gate and the incapacitation in list B should be made ineffective since list B might later be compared with another cut set list. Table IV-2 shows the process for list B.

4.3. Processing 'AND' Gates

Processing the 'AND' logic requires more stages than the 'OR' logic. Any problem for the 'AND' combination is reduced to combining two cut set lists. The algorithm especially prevents the creation of too many non-minimal cut sets since processing those cut sets is a time consuming effort. Table IV-3 gives two cut set lists A and B, obtained by processing the input to gate 1 and 2. The selected cut sets will demonstrate all the mechanisms in this process.

In case lists A and B do not have common events (intersection is zero) the lists are simply combined without giving attention to subsuming events. If the lists do possess common events then first list A will be processed, second list B, and finally an intermediate list. The cut sets which qualify for the final list need no further processing, the cut sets in the intermediate list need to be compared with the intermediate list and with the final list for the existence of subsuming cut sets.

Each cut set of list A is compared with list B with respect to consecutively:

1) its intersection of the cut set with the events in the domain of list B,
2) the existence of an identical cut set in list B, and
3) the existence of any subset of the cut set considered in List B.

Based on the results of the above mentioned comparisons the following actions are taken, see Table IV-3, respectively:

1) if its intersection is empty, this cut set is not processed,
2) if an identical cut set exists in list B then this cut set is placed in the final list and the identical cut sets in list A and B are temporarily incapacitated, and
3) if a subset of this cut set exists in list B then this cut set of list A is placed in the final list and the cut set in list A is temporarily incapacitated.

While processing the cut sets of list B, essentially the same comparisons are performed except for the search for identical cut sets. However, the actions are different, see also Table IV-4, viz.:

1) if a subset of this cut set exists in list A, this cut set is placed in the final list, and
2) if this subset does not qualify for direct transfer to the final list, all the combinations of this cut set with the cut sets of list A are placed in an intermediate list.

Finally the intermediate list is compared with itself, and with the final list for occurrence of subsets. If no subsets occur, the cut sets are transferred to the final list. Identical cut sets 'collide' in the same branch position in the binary tree of the final list and don't have to be searched for.

Finally, the incapacitations in list A and B should be made ineffective at the end of the 'AND' gate processing, since lists A and B might later be compared with other cut set lists.

4.4. Processing Mutually Exclusive Events

Mutually exclusive events can be specified by the user and will be deleted from the final cut set list. The final cut sets and its subsets will be searched for in the list of mutually exclusive events. If a match is found that particular cut set is eliminated from the final list.

4.5. Presenting the Minimal Cut Sets

Since the number of cut sets expressed in basic events can be prohibitively large, the normally presented cut sets are expressed in independent subtree gates and pseudo-gates which usually comprises a much smaller number of cut sets, while the cut sets of the independent subtree gates and pseudo-gates are presented separately. A list of the cut sets expressed in basic events with the specified cut-off criteria applied can be selected as a program option.

The statistics of the cut sets are concisely presented in Table IV-5. The format of the table is chosen such that 'outlier' cut sets can be identified easily and that the information regarding top event and cut-off criteria are all included.

The cut sets are classified per order and per probability or frequency. The number of cut sets, their summarized frequency, and their fraction of the total frequency are given per order. Within each order the cut sets are distributed according to their value into 9 linear, 9 logarithmic, and 2 special bins. The bin boundaries are relative to a rounded value of the cut sets' maximum probability or frequency, as is explained in the notes of the table. The two special bins contain cut sets with extremely small, but non-zero values and cut sets with value zero, most likely caused by definition (probability = 0).

The applied cut-off criteria are shown in the table and the number of times the criteria were applied are given. The precise number should be interpreted cautiously, it does not indicate the number of discarded cut sets for the top events since that number could be larger through possible later combinations of the discarded cut sets with other events; and on the other hand could that number be smaller or zero as well since the discarded cut sets might otherwise have been subjected to reduction by subsuming events. However if a limit is never encountered, attempting to obtain more cut sets by merely changing that limit (increasing for order and decreasing for probability) has no effect. The probability cut-off value multiplied with the number of times the criterion is applied gives an upperbound of the

discarded cut sets. A more accurate approximation of this upperbound will be evaluated in the last update of the program. The cumulative probabilities of the discarded cut sets by the cut set order and by the cut set probability cut off criteria will be calculated and shown separately.

A discrepancy exists between the method of deriving the frequency per order and the more accurate method of obtaining the total frequency. Since the method per order is a straight rare event approximation, it can become larger than 1.0 while the method for obtaining the total frequency applies the inclusion-exclusion principle in a limited manner ensuring that its total never exceeds 1.0 . The algorithm for calculating the consecutive frequency uses the formula:

$$P_{\text{cut set}_I} = P_{\text{cut set}_{I-1}} + (1 - P_{\text{cut set}_{I-1}}) * P_{\text{cut set}_i}.$$

in which i refers to the particular cut set frequency, and
I indicates the sum of the frequencies of cut set_1 through cut set_i.

5. STRUCTURE OF THE COMPUTER PROGRAM

The computer program consists of approximately 6000 Lines of Code (6 KLOCs) of FORTRAN 77. The choice for this language is its superior portability to other machines and the authors familiarity with it. The realization of fancy screens and of elaborate interactive control is not supported by this language. The advantage is that no complicated menu sequences need to be learned. System and compiler dependent routines are used for obtaining the time and the date, for setting and reading bits in 2-byte integers and for writing two 2-byte integers into one 4-byte integer and vice versa.

A minimum of special knowledge is required for using the program; word processor skills are adequate to write the input fault tree logic and failure data. Knowledge of system analysis and reliability theory to define the problem and to interpret its results is obviously required.

Memory efficiency is given a great deal of attention by using 2-byte integers whenever possible, and by storing cut sets and other administrative data on disk. Time efficiency is addressed by applying the described

methods, by using available fast algorithms for sorting and searching, and by reducing the disk read operations through a large cache memory. Structured programming techniques are inevitable for this medium-sized code.

A to be published user's manual will contain example inputs and a list with the error messages.

6. DISCUSSION OF THE RESULTS

The described computer program MIRAP is capable of handling large fault trees on microcomputers since it addresses the speed and memory limitations of these machines adequately and incorporates special techniques as described in this paper regarding fault tree restructuring. The results of this program have been compared favorably with other programs during the course of its development. However, these results are not reported here since a more elaborate effort is required to appropriately compare the various existing programs. A comparison of the characteristics of other programs such are outlined in [3] and [4] is clearly beyond the scope of this paper. After extensive error checking, a user's manual will be published that will include an analysis of a problem which has similar characteristics as the benchmark fault tree of [4] and will facilitate mutual comparisons. Due to proprietary aspects dissemination of that fault tree and its data is not possible.

7. REFERENCES

[1] Terpstra, K.; Phased mission analysis of maintained systems - A study in reliability and risk analysis; Netherlands Energy Research Foundation, ECN-158(1984)

[2] Terpstra, K., Dekker, N. H., Driel, G. van; PHAMISS - A reliability computer program for phased mission analysis and risk analysis; 2 volumes, ECN-183 (1986)

[3] Roberts, N. H., Vesely, W. E., Haasl, D. F., Goldberg, F. F.; Fault tree handbook; US Nuclear Regulatory Commission, NUREG-0492 (1981)

[4] Amendola, A.; Systems reliability benchmark exercise, final report; CEC-JRC Ispra, EUR 10696 (1985)

Table IV-1: Processing the first cut set list in an 'OR' gate (list A)

Each cut set of list A will be processed and if its characteristics meet the criteria it will be transferred to the final list.

List A Gate 1	List B Gate 2	List C Final list	Remark
a,b,c	a,b,c[1]	a,b,c	Cut set found in list B and placed in the final list. This cut set in list B is temporarily incapacitated.
a,b,d	a,d	-	A subset of this cut set matches a cut set in list B and this cut set is for that reason not placed in the final list
a,e	a,b,e	a,e	Although cut set 'a,b,e' of List B is not a minimal cut set, at this point only cut sets and its subsets of List A are searched in List B.
x,y		x,y	No events in common with list B, this cut set is placed in the final list.

Note: 1) This cut set is temporarily incapacitated.

Table IV-2: Processing the second cut set list in an 'OR' gate (list B)

Each cut set of list B will be processed and if its characteristics meet the criteria it will be transferred to the final list.

List B Gate 2	List A Gate 1	List C Final list	Remark
a,b,c[1]	a,b,c	-	This cut set is incapacitated.
a,d	a,b,d	a,d	Cut set 'a,b,d' of List A didn't qualify in the first step for inclusion in the final list. Cut set 'a,d' is added.
a,b,e	a,e x,y	-	A subset of this cut set matches a cut set in list A and the cut set is for this reason not placed in the final list

Note: 1) This cut set is temporarily incapacitated.

Table IV-3: Processing the first cut set list in an 'AND' gate (list A)

List A Gate 1	List B Gate 2	List C Int.list	List D Final list	Remark
a,b,c[1]	a,b,c[1]		a,b,c	Identical cut set found in list B. Place this cut set in the final list and incapacitate both these cut sets.
a,b,d[1]	a,d		a,b,d	A subset of this cut set is found in list B. This cut set as such will be placed in the final list, incapacitated, and not combined since 'a,b,d' will be minimal.
x,a[2]				
b,d,c				
x,y				No events in common with list B, not processed in this stage

Note: 1) This cut set is temporarily incapacitated.
2) These cut sets are added to show effects in the second processing stage.

Table IV-4: Processing the second cut set list in an 'AND' gate (list B)

List B Gate 2	List A Gate 1	List C Int.list	List D Final list	Remark
	x,y			
	x,a			
	b,d,c			
a,b,c[1]	a,b,c[1]		a,b,c	This cut set is incapacitated.
a,d	a,b,d[1]	a,d,x,y[3]		No subset of this cut set are found in list A. This cut set will be combined with the remaining cut sets of list A. Its result is placed in intermediate list C.
		a,d,e		
		a,d,x		
		a,b,c,d[3]		
a,b,e[2]	a,e		a,b,e	A subset of this cut set is found in list A. This cut set as such will be placed in the final list.

Note: 1) This cut set is temporarily incapacitated.
2) Since this cut set is no more addressed, incapacitation is not required.
3) These cut sets will be deleted in the final process step

Table IV-5: Concise presentation of the cut set statistics

```
*****************************************************************************
**  C U T    S E T    S T A T I S T I C S   FOR TOPEVENT: NO_FEEDWATER_TO_SG  **
**-------------------------------------------------------------------------**
**CUT-NUMBER | FRACTION |  TOTAL   |       CUT SET CLASSIFICATION -see (1)    **
**SET|  OF   | OF TOTAL |FREQUENCY |    LINEAR BINS    |  LOGARITMIC BINS  |ZE-**
**ORD| CUTS  |----------|----------|-------------------|-------------------|RO **
** -----------                     2.0E-3  1.2E-3  4.0E-4  6.3E-6  6.3E-8  NIL**
**                                 | 1.8E-3  1.0E-3  2.0E-4  2.0E-6  2.0E-8   **
**                                 |   1.6E-3  8.0E-4  6.3E-5  6.3E-7  6.3E-9 **
**                                 |     1.4E-3  6.0E-4  2.0E-5  2.0E-7  0.0E-1*
** --------------------------------|--+-+-+-+-+-+-+-+-|-+-+-+-+-+-+-+-+-+---- **
*                                  |                  |                       *
*  2       5     .43616 1.2800E-03            1       2          1 1  |PROB. *
*  3      34     .55635 1.6328E-03          1     1   1 1 310 2   1 9 5|CUT-  *
*  4      35     .00684 2.0087E-05                    |      4 5 3 7 7 9|OFF   *
*  5      28     .00065 1.9091E-06                    |          2 6 713|-X-X- *
*  9 CUT SETS LARGER THAN ORDER  8 ARE ELIMINATED   O R D E R   C U T - O F F  *
* -- TOTAL ----------------------|--+-+-+-+-+-+-+-+-|-+-+-+-+-+-+-+-+-+----- *
*        102    1.00000 2.9348E-03 -see(3)   1 1   1   3 1 314 7 5152427|-X-X- *
** --------------------------------|-------------------------------------- **
** FREQ. ABOVE IS RARE EVENT APPROX|CUT-OFF|PROB. 1.00E-08 9598 TIMES APPLIED ** see (2)
** FREQ. BELOW IS MORE ACCURATE    |LIMITS |ORDER    8     LIMIT IS NEVER HIT **
** TOTAL FREQUENCY      2.9316E-03 |CUT-OFF NOT APPLIED TO 1-ST ORDER CUT SETS**
*****************************************************************************
** ELAPSED TIME  3634 SECONDS(TOTAL:  1 HRS, 9 MINS,36 S)** see (4)
** -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=- **
** TIME IS  12:19: 4     ON  APR 26 1988                 **
** -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=- **
```

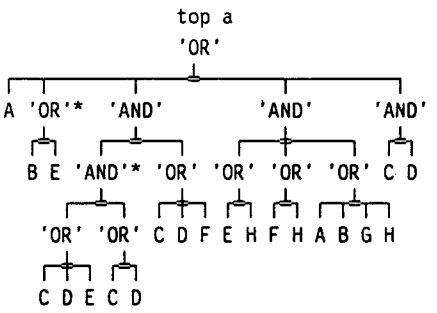
Notes:

(1) 'Outlier' cut sets are more easily recognized by this classification. The 102 cut sets are classified in 9 linear, 9 logaritmic, and 2 special bins per order. The bin boundaries are determined relative to a rounded value of the cut sets maximum probability or frequency. The 9 linear boundaries are 100%, 90%, 80%, etc. until 10% of Pmax, the 9 logaritmic boundaries are: Pmax * 10^{-1}, Pmax * $10^{-1.5}$, Pmax * 10^{-2}, etc. until Pmax * $10^{-5.5}$ and the two special bins summarize the cut sets with respectively an extremely small but non-zero probability and those with zero (NIL) probability. A more elaborate form of this table is produced seperately.

(2) This table also shows whether the used order and probability cut-off values are actually applied in the cut set determination process. Merely increasing the order cut-off would not change the results in this case since that limit is never encountered, while decreasing the probability cut-off might yield more cut sets.

(3) The discrepancy between the two presented frequencies is caused by a different numerical summation. The row total frequency is a straight rare event addition of all the cut set frequencies while the frequency in the table's bottom line is obtained by applying the inclusion-exclusion principle in a limited manner, viz. $P_{new} = P_{old} + (1 - P_{old}) * P_{cut\ set}$. This is more accurate and does not give probabilities larger than 1, but not exact.

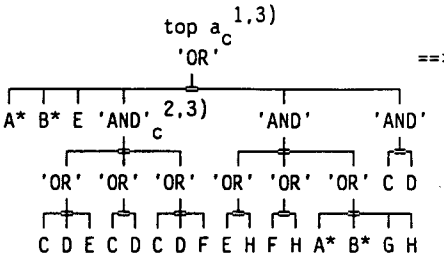
(4) The elapsed time refers to the time past since the previous time message (in this case the time required for the cut set determination), while the TOTAL (time) is relative to the start of the program.



==>

Note:
The gate logic is written in these concise fault trees and when opportune the gates are labelled. Events marked with an '*' will be changed in the next permutation.

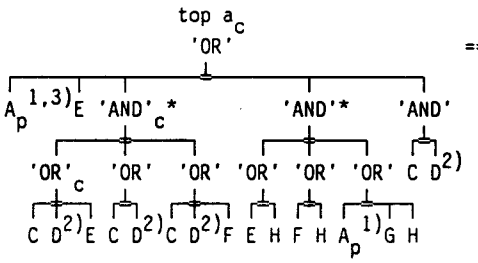
Fig. 3-1 Original fault tree



==> fig. 3-3

Notes:
1) Compressed 'OR' gate.
2) Compressed 'AND' gate.
3) Subscript $_c$ denotes compressed event.

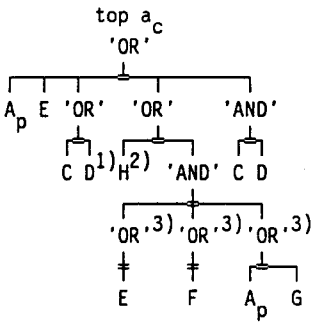
Fig. 3-2 Compressing cascading gates from fig. 3-1.



==>

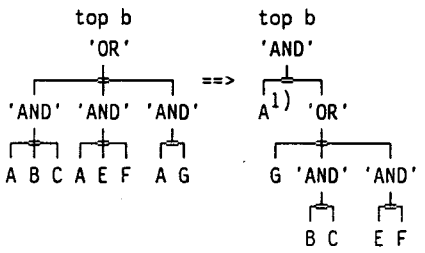
Notes:
1) Events A,B form a pseudo-gate.
2) Events C,D do not qualify for pseudo-gate due to different gate logic.
3) Subscript $_p$ denotes pseudo-gate.

Fig. 3-3 Creating pseudo-gates of sibling events from fig. 3-2.



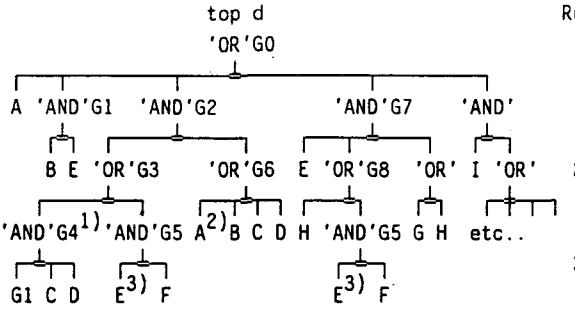
Notes:
1) Events C,D in an 'OR' combination remain from the 'AND' gate.
2) Event H is the common input to 'AND' gate.
3) These gates might appear unchanged in other parts of the tree.

Fig. 3-4 Removing first order inputs to 'AND' gates from fig. 3-3.



Notes:
1) Event A is the common input to 'OR' gate.

Fig. 3-5 Removing first order inputs to 'OR' gates (not applied!)



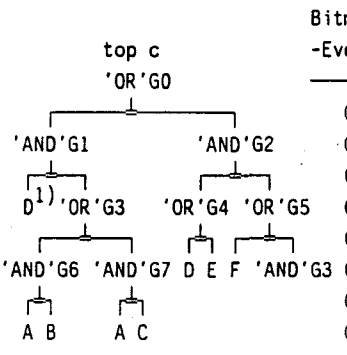
Rules for removing subsuming events

1) Events which are also inputs to ancestors will be removed if the gate logics are simular in the corresponding cases,
2) The concerned gate of an event will be removed if that event is also input to an ancestor with a different gate logic, and
3) Only those modifications are carried out which apply to all the branches of the tree.

Notes:

1) 'AND' gate G4 can be removed because one of its inputs gate G1 also appears under the top 'OR' gate, see rule 2.
2) Event A can be removed from this 'OR' gate because event A also appears under the top 'OR' gate, see rule 1.
3) Event E could be removed from 'AND' gate G5 in branch G0-G7-G8-G5 because this event also appears in 'AND' gate G7, see rule 1, however because this gate in branch G0-G2-G3-G5 does not qualify, see rule 3, this gate is not modified, rule 3.

Fig. 3-6 Removing first order subsuming events



Bitmaps -Events-	gates 01234567	basic ABCDEF
G0	.1111111	111111
G1	...1..11	1111..
G2	...11111	111111
G3	11	111...
G4	....	...11.
G5	...1..11	111..1
G6	....	11....
G7	....	1.1...

Example — 0..34..7 | A.CD.F

-Analyze the gates in a bottom-up manner:
-Start with gate G6, its bitmap, incl. G6 itself: .....1. | 11....
-Add the RRs 2) of A and B ....11 | 11....
-Conclusion for gate G6 see 4)
-Proceed with gate G3, its bitmap incl. G3 is: ...1..11 | 111...
-Add the RRs of A, B, C, G6, and G7: ...1..11 | 111...
-Conclusion for gate G3 see 5).

Notes:

1) If this events would be 'F', then gate event G1 is a second order independent subtree, assuming gate G5 is named the same as gate G1.
2) RR stands for reverse reference.
3) Since two gate events(G6 and G7) are referenced process is continued.
4) Since the determined bitmap references events outside the domain of gate G6, this gate does not qualify for independent subtree.
5) Gate G3 is independent since its bitmap equals the determined bitmap, except for its own reference.

Fig. 3-7 Identifying all the independent gates

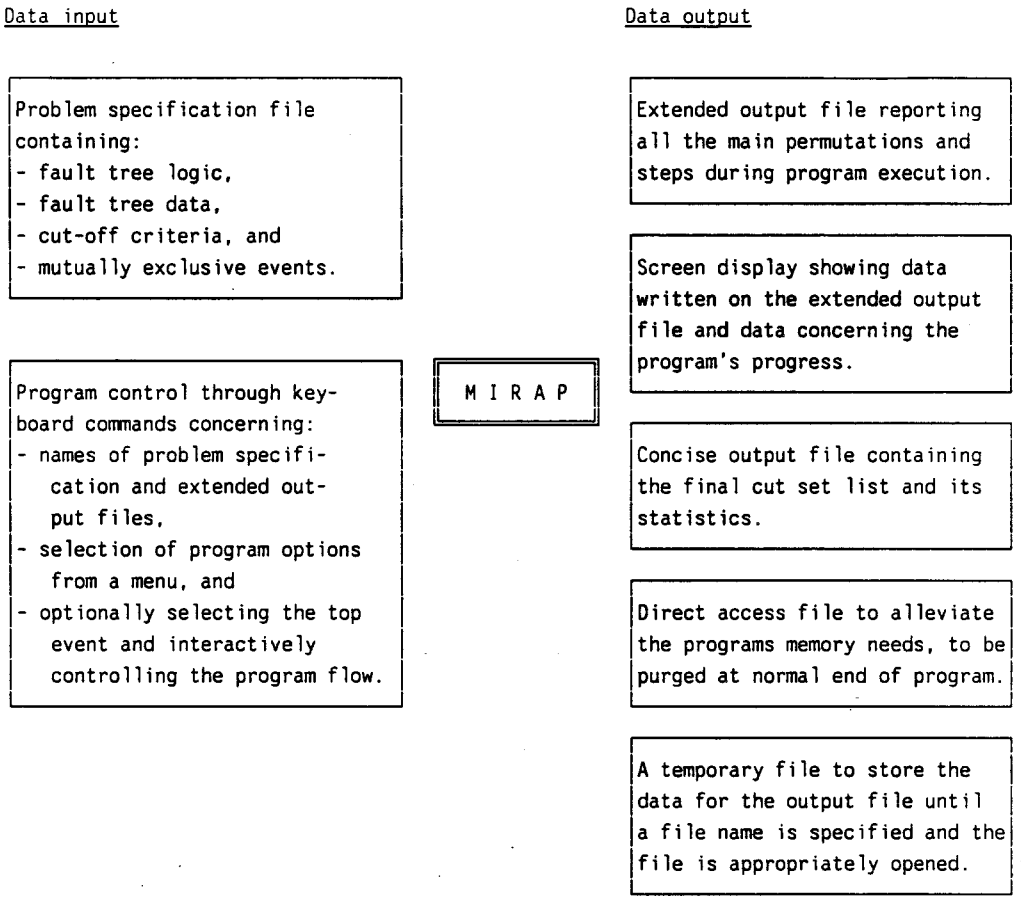


Fig. 5-1 Program structure of MIRAP

APPENDIX A. Binary data structure of the cut set list

A.1. Implemented Binary Data Structure

To facilitate an efficient search procedure the cut set list is organized in a binary data structure, which is stored on disk. The cut sets are ranked according to (1) their order, and (2) the numbers of the contained events.

A pointer to a higher and a lower ranked cut set is stored together with a backward referencing pointer as is shown below. This pointer structure supports a cut set retrieval procedure in either a ranked or a 'binary' fashion, with only the pointer to the previous cut set as input. The 'binary' cut set retrieval procedure obtains the cut sets in the same manner as the binary data structure was created initially. This prevents the destruction of the binary structure of a new cut set list, since retrieving the cut sets in a ranked fashion and creating a new list with them makes the new data structure sequential with unfavorable search properties.

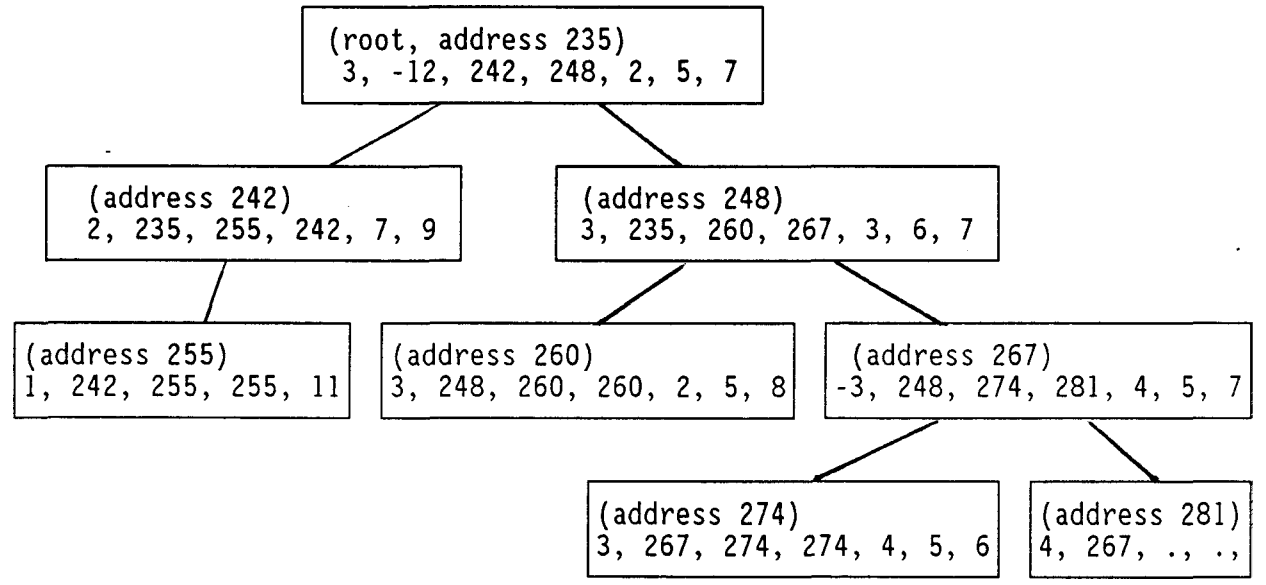
The used record structure is :

cut set order	address 'parent' cut set	address lower ranked cut set	address higher ranked cut set	event 1 of cut set	event 2 of cut set	. . . . .	event n of cut set
2 bytes	4 bytes	4 bytes	4 bytes	2 bytes	2 bytes	. . . . .	2 bytes

Used rules: - Negative back reference indicates the root of the list,
(Conventions) - Referencing itself in the higher or lower ranked exit indicates the end of that branch, and
- Negative order incapacitates this cut set definitively and an order larger than 1500 incapacitates this cut set temporarily. Eliminating a cut set from a binary tree requires a complicated recreation of a part of the tree pointers and is not performed.

A.2. Example of a binary data structure

Word lengths are assumed equal for simplicity.



A.3. Contents of the data file

addresses		last digit									
		..0	..1	..2	..3	..4	..5	..6	..7	..8	..9
First	23.	..	..	..	..	..	3	-12	242	248	2
two	24.	5	7	2	235	255	242	7	9	3	235
digits	25.	260	267	3	6	7	1	242	255	255	11
	26.	3	248	260	260	2	5	8	-3	248	274
	27.	281	4	5	7	3	267	274	274	4	5
	28.	7	4	267	..	..	etc.				

NRC FORM 335 (2-84) NRCM 1102, 3201, 3202

U.S. NUCLEAR REGULATORY COMMISSION

BIBLIOGRAPHIC DATA SHEET

SEE INSTRUCTIONS ON THE REVERSE

1. REPORT NUMBER (Assigned by TIDC, add Vol. No., if any)
EGG-SSRE-8137

2. TITLE AND SUBTITLE
MIRAP, Microcomputer Reliability Analysis Program

3. LEAVE BLANK

4. DATE REPORT COMPLETED
MONTH: January YEAR: 1989

5. AUTHOR(S)
J. N. T. Jehee

6. DATE REPORT ISSUED
MONTH: January YEAR: 1989

7. PERFORMING ORGANIZATION NAME AND MAILING ADDRESS (Include Zip Code)
U. S. Nuclear Regulatory Commission
Washington, D. C.

8. PROJECT/TASK/WORK UNIT NUMBER

9. FIN OR GRANT NUMBER

10. SPONSORING ORGANIZATION NAME AND MAILING ADDRESS (Include Zip Code)

11a. TYPE OF REPORT
Technical

b. PERIOD COVERED (Inclusive dates)

12. SUPPLEMENTARY NOTES

13. ABSTRACT (200 words or less)

A program for a microcomputer is outlined that can determine minimal cut sets from a specified fault tree logic. The speed and memory limitations of the microcomputers on which the program is implemented (Atari ST and IBM) are addressed by reducing the fault tree's size and by storing the cut set data on disk. Extensive well proven fault tree restructuring techniques, such as the identification of sibling events and of independent gate events, reduces the fault tree's size but does not alter its logic. New methods are used for the Boolean reduction of the fault tree logic. Special criteria for combining events in the 'AND' and 'OR' logic avoid the creation of many subsuming cut sets which all would cancel out due to existing cut sets. Figures and tables illustrate these methods.

14. DOCUMENT ANALYSIS - a. KEYWORDS/DESCRIPTORS

b. IDENTIFIERS/OPEN-ENDED TERMS

15. AVAILABILITY STATEMENT

16. SECURITY CLASSIFICATION
(This page) Unclassified
(This report)

17. NUMBER OF PAGES

18. PRICE