Qualitative analysis of complex modularized fault trees using binary decision diagrams

R Remenyte and **J D Andrews***
Aeronautical and Automotive Engineering, Loughborough University, Loughborough, UK

The manuscript was received on 15 June 2005 and was accepted after revision for publication on 23 February 2006.

DOI: 10.1243/1748006XJRR10

Abstract: Fault tree analysis is commonly used in the reliability assessment of industrial systems. When complex systems are studied conventional methods can become computationally intensive and require the use of approximations. This leads to inaccuracies in evaluating system reliability. To overcome such disadvantages, the binary decision diagram (BDD) method has been developed. This method improves accuracy and efficiency, because the exact solutions can be calculated without the requirement to calculate minimal cut sets as an intermediate phase. Minimal cut sets can be obtained if needed.

BDDs are already proving to be of considerable use in system reliability analysis. However, the difficulty is with the conversion process of the fault tree to the BDD. The ordering of the basic events can have a crucial effect on the size of the final BDD, and previous research has failed to identify an optimum scheme for producing BDDs for all fault trees. This paper presents an extended strategy for the analysis of complex fault trees. The method utilizes simplification rules that are applied to the fault tree to reduce it to a series of smaller subtrees whose solution is equivalent to the original fault tree. The smaller subtree units are less sensitive to the basic event ordering during BDD conversion. BDDs are constructed for every subtree. Qualitative analysis is performed on the set of BDDs to obtain the minimal cut sets for the original top event. It is shown how to extract the minimal cut sets from complex and modular events in order to obtain the minimal cut sets of the original fault tree in terms of basic events.

Keywords: fault tree analysis, binary decision diagram, minimal cut sets

1 INTRODUCTION

The binary decision diagram (BDD) method [**1**] has been developed as an alternative to conventional methods for performing qualitative and quantitative analysis of fault trees. This method appears to be more efficient for analysing a system without the need for the approximations used in the traditional approach of kinetic tree theory [**2**].

Rather than analysing the fault tree directly the BDD method first converts the fault tree to a binary decision diagram that represents the Boolean equation for the top event. It is possible that problems may occur with the conversion process of the fault tree to the BDD. If the ordering of the basic events is not chosen suitably, the size of the final BDD can grow exponentially. Previous research has failed to identify an optimum scheme for producing BDDs for all fault trees. Research has now focused on the application of alternative techniques that will facilitate the use of BDDs to solve large fault tree structures.

This paper presents an analysis approach using two simplification strategies that have been shown to be effective in reducing the complexity of the problem: reduction [**3**] and modularization [**4**]. The reduction technique simplifies the fault tree to its minimal logic form, while modularization breaks down the fault tree to independent subtrees that can be analysed separately. BDDs are obtained for each module in separate computations, culminating in a set of BDDs, which together represent the original system failure diagram. This strategy is described

**Corresponding author: Department of Aeronautical and Automotive Engineering, University of Loughborough, Loughborough, Leicestershire LE11 3TU, UK. email: J.D.Andrews@lboro.ac.uk*

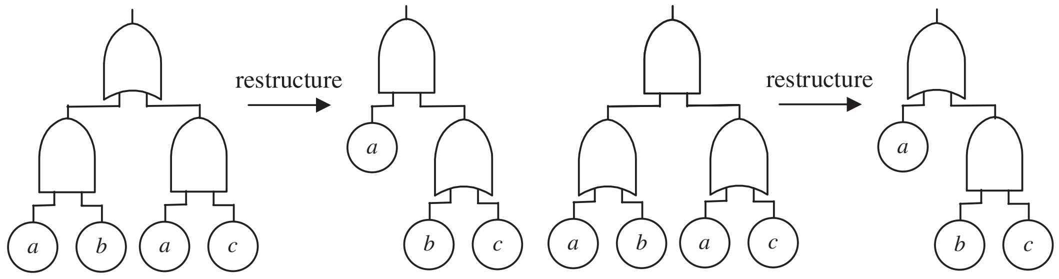


Fig. 1 Reduction, the extraction procedure

in reference [**5**] where quantitative analysis is performed on the set of BDDs to obtain the top event probability, the system unconditional failure intensity, and the criticality of the basic events.

A qualitative analysis of a fault tree produces a list of minimal cut sets. These are lists of component failures that are necessary and sufficient to cause the top event. A method of obtaining minimal cut sets is not presented in the original treatment and is the subject of this paper. Before the calculation of minimal cut sets all BDDs need to be minimized, using Rauzy's minimization procedure [**1**]. Then qualitative analysis for every module can be carried out and minimal cut sets for the whole system extracted. Each of these stages is described in detail in the following sections and demonstrated throughout with the use of an example.

2 SIMPLIFICATION OF THE FAULT TREE STRUCTURE

For complex industrial systems fault trees can be very large and their qualitative and quantitative analyses are time consuming. Therefore two pre-processing techniques can be applied to the fault tree in order to obtain the smallest possible subtrees and reduce the size of the problem. The first stage of pre-processing is a reduction technique [**3**] that restructures the fault tree to its most concise form. Once this has been applied it is possible to simplify the failure logic diagram further by identifying independent subtrees (modules) within the fault tree that can be treated separately. The linear-time algorithm is an extremely efficient method of modularization and forms the second stage of fault tree pre-processing. This results in a set of independent fault trees, each with the simplest possible structure, which together describe the original system failure causes.

2.1 Reduction

The reduction technique reduces the fault tree to its simplest form while retaining the significant features

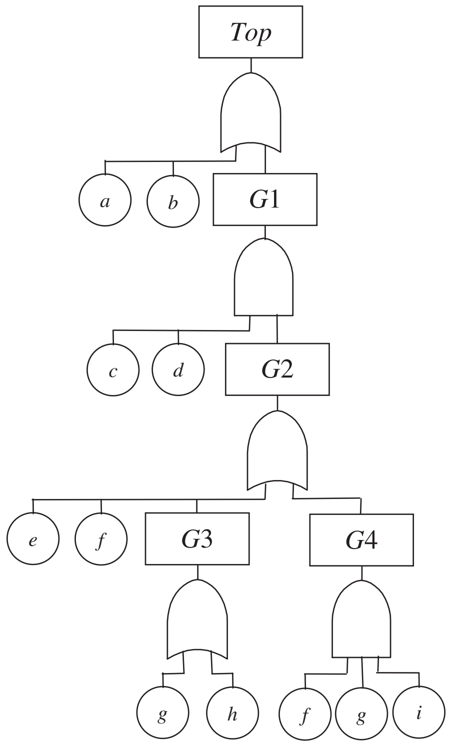


Fig. 2 Example fault tree

of the logic structure. Its effectiveness has been demonstrated with its application to a large set of fault trees, where it reduced the size of the resulting BDDs by approximately 50 per cent [**5**]. This reduction approach is applied in three stages: contraction, factorization, and extraction. First, subsequent gates of the same type are contracted to form a single gate. Second, pairs of events that always occur together in the same gate type are identified and they are combined to form a single complex event. Finally, the following two structures from Fig. 1 are identified and replaced in order to reduce the repeated occurrence of events to a single occurrence and facilitate further reduction. The above three steps are repeated until no further changes are possible in the system that would result in a more compact representation of the fault tree.

Consider the fault tree shown in Fig. 2. Using the reduction technique a smaller tree is obtained, as

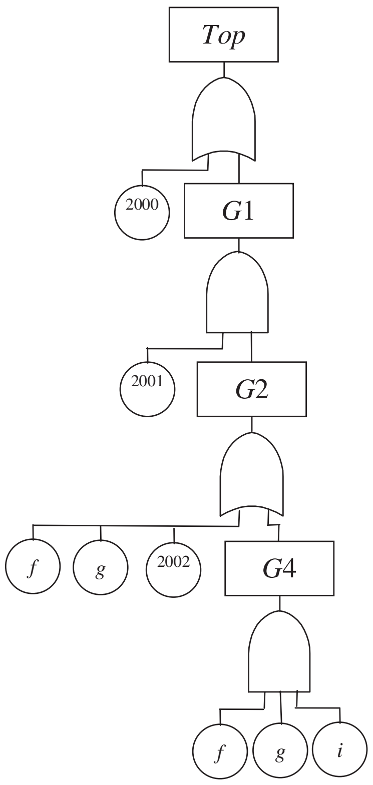


Fig. 3 Fault tree after reduction

Table 1 The complex event data

Complex event	Gate value	Event 1	Event 2
2000	OR	*a*	*b*
2001	AND	*c*	*d*
2002	OR	*e*	*h*

shown in Fig. 3. At first, two subsequent gates of the same type (*G*2 and *G*3) were contracted forming a single gate. Then the factorization procedure was performed three times: for a pair of basic events *a* OR *b*, for a pair of basic events *c* AND *d*, and for a pair of basic events *e* OR *h*, creating complex events 2000, 2001, and 2002 respectively. In this example there were no structures of the type presented in Fig. 1, therefore, the extraction procedure was not applied. The corresponding complex event data are shown in Table 1.

Reduction has simplified the example fault tree. In the original fault tree there were five gates; in the reduced fault tree there are four. In the original tree there were eleven events, nine of them different; in the reduced tree there are eight events, and six of them are different. For large systems the degree of simplification is far more significant. Having reduced the fault tree to a more concise form, the second pre-processing technique of modularization is considered.

Table 2 Step numbers for every node in the fault tree

Step number	1	2	3	4	5	6	7	8
Node	*Top*	2000	*G*1	2001	*G*2	*f*	*g*	2002
Step number	9	10	11	12	13	14	15	16
Node	*G*4	*f*	*g*	*i*	*G*4	*G*2	*G*1	*Top*

Table 3 Data for gates in the fault tree

Gate	*Top*	*G*1	*G*2	*G*4
1st visit	1	3	5	9
2nd visit	16	15	14	13
Final visit	16	15	14	13
Min	2	4	6	6
Max	15	14	13	12

Table 4 Data for events in the fault tree

Event	2000	2001	*f*	*g*	2002	*i*
1st visit	2	4	6	7	8	12
2nd visit	2	4	10	11	8	12
Final visit	2	4	10	11	8	12

2.2 Modularization

The modularization procedure identifies subtrees within the fault tree, known as modules. A module of a fault tree is a subtree that is completely independent from the rest of the tree. It contains no basic events that appear elsewhere in the fault tree. The advantage of identifying these modules is that each one can be analysed separately from the rest of the tree. The results from subtrees identified as modules are substituted into the higher-level fault trees where the modules occur.

Using the linear-time algorithm the modules can be identified after just two depth-first traversals of the fault tree. The first of these performs a step-by-step traversal recording, for each gate and event, the step number at the first, second, and final visits to that node. Each gate is visited at least twice. After the first traversal the maximum (Max) of the last visits and the minimum (Min) of the first visits of the descendants (any gates or events appearing below that gate) of each gate are calculated. Step numbers for every node in the example fault tree, Max and Min of the gates and events for the reduced tree in Fig. 3 are presented in Tables 2, 3, and 4 respectively.

The principle of the algorithm is that if any descendant of a gate has a first visit step number smaller than the first visit step number of the gate, then it must also occur beneath another gate. Also, if any descendant has a last visit step number greater than the second visit step number of the gate, then

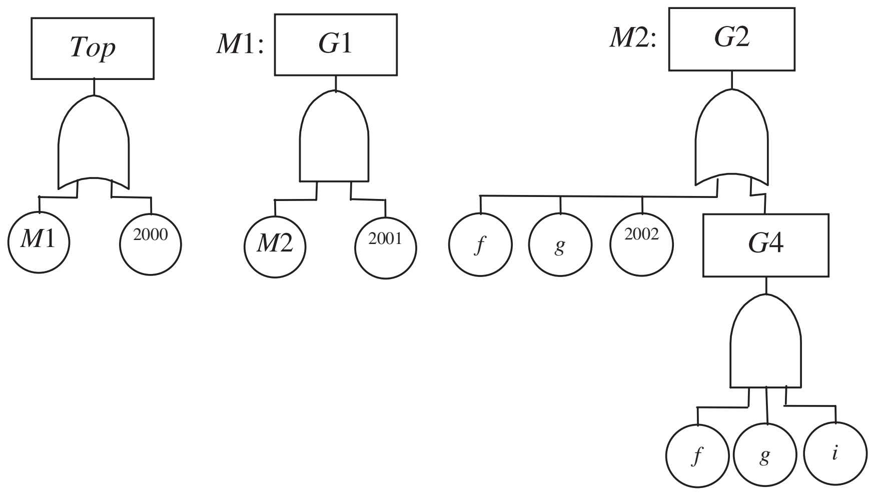


Fig. 4 The three modules obtained for the fault tree shown in Fig. 3

again it must occur elsewhere in the tree. Therefore, the rules for identifying a gate as heading a module are:

(a) the first visit to each descendant is after the first visit to the gate;
(b) the last visit to each descendant is before the second visit to the gate.

The following gates can be identified as heading modules

Top, *G*1, *G*2

*G*4 can not be a module because some of its descendants (events *f* and *g*) are visited before gate *G*4. The occurrences of these subtrees are replaced by the single modular events, which are named

$G1 - M1, G2 - M2$

Three separate fault trees, shown in Fig. 4, now replace the fault tree in Fig. 3.

Having reduced the fault tree to its minimal form and identified all the independent modules the next stage is to obtain the BDDs.

3 OBTAINING THE BINARY DECISION DIAGRAMS

A BDD must be constructed for each of the modules. In this paper the variable ordering scheme for every module is set to be left-right top-down. For examples as small as these the variable ordering is largely irrelevant. Following the chosen scheme gives the orderings of basic events

$$\begin{aligned} Top &: M1 < 2000 \\ M1 &: M2 < 2001 \\ M2 &: f < g < 2002 < i \end{aligned}$$

The BDD construction method is described in [**1**]. It works by using an ordered triple to represent each node on the BDD

$$\mathbf{ite}(x, f_1, f_0)$$

where x is the Boolean variable represented by the node and f_1 and f_0 are the logic functions on its ‘1-branch’ and ‘0-branch’ respectively. **ite** is if-then-else operation

```
If $x$ occurs
  then consider $f_1$
  else consider $f_0$
endif
```

BDD construction then moves through the fault tree in a bottom-up manner. Basic events are assigned **ite** structures. For example, a basic event *a* is expressed as

$$a = \mathbf{ite}(a, 1, 0)$$

When gates are encountered and the input events, *J* and *H*, are expressed in their **ite** form, the following rules are applied

$$\text{If } J = \mathbf{ite}(x, f_1, f_0) \text{ and } H = \mathbf{ite}(y, g_1, g_0)$$

$$\text{then } J \oplus H = \begin{cases} \mathbf{ite}(x, f_1 \oplus H, f_0 \oplus H) \\ \quad \text{if } x < y \text{ in the ordering} \\ \mathbf{ite}(x, f_1 \oplus g_1, f_0 \oplus g_0) \\ \quad \text{if } x = y \text{ in the ordering} \end{cases}$$

Applying these rules to the fault tree in Fig. 4 results in the BDDs presented in Fig. 5.

Once the complete set of BDDs have been computed, the qualitative and quantitative analyses can be carried out. This paper concentrates on the calculation of minimal cut sets using binary decision diagrams obtained from simplified trees.

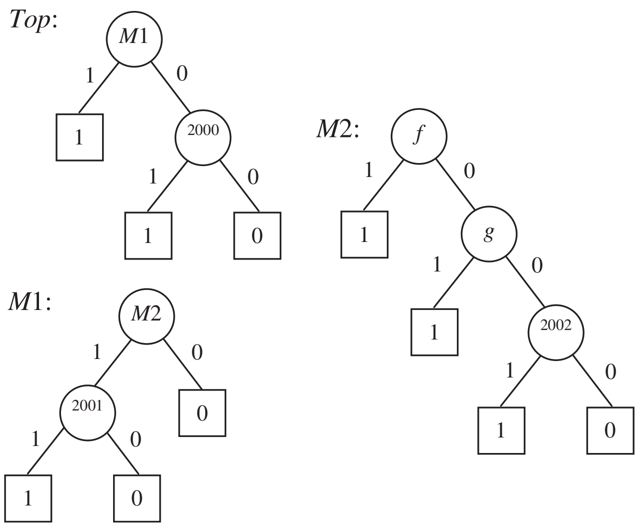


Fig. 5 The obtained BDDs for the modules presented in Fig. 4

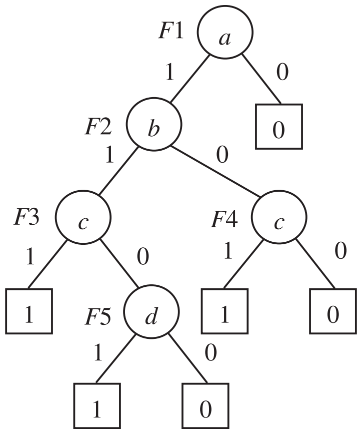


Fig. 6 Example BDD

4 COMPUTATION OF MINIMAL CUT SETS

Qualitative analysis of BDDs [**6**] produces a list of minimal cut sets of the fault tree. A minimal cut set is a list of component failure events that are both necessary and sufficient to cause the system failure mode. Every path through a BDD starts from the root vertex and proceeds down through the diagram to a terminal vertex. Paths that terminate at a 1 vertex yield a set of conditions that will result in system failure. Those components that are encountered on the path in their failure state (node exited on the 1 branch) will be members of the cut set. The task then is to remove cut sets that do not represent the minimal conditions.

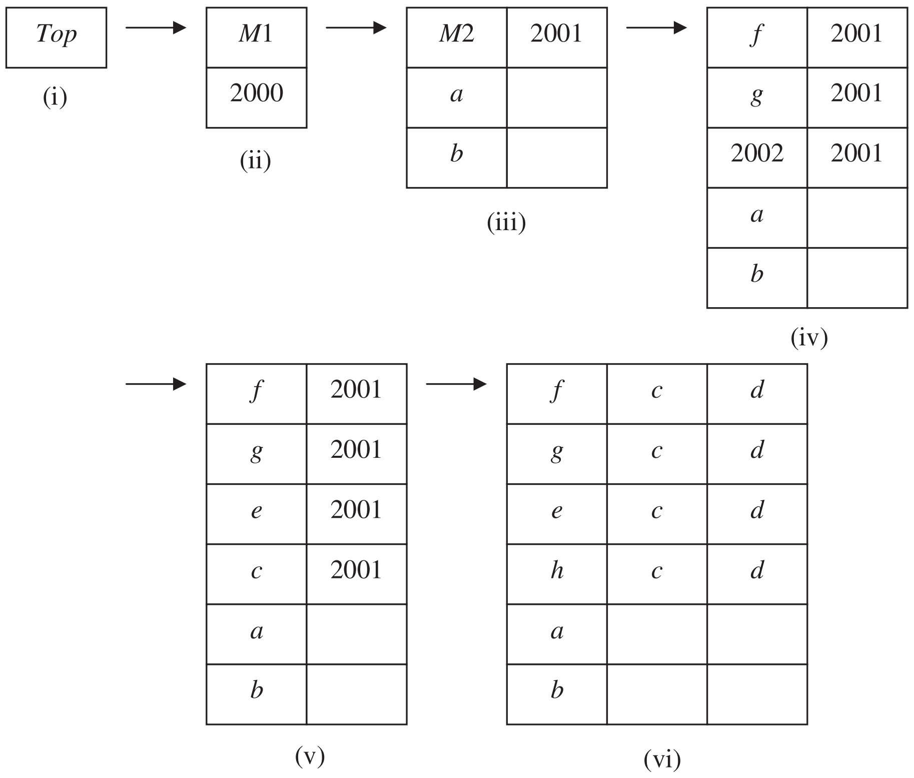
Consider the example of a BDD illustrated in Fig. 6. There are three paths through the BDD to the terminal 1 vertex passing through nodes: ($F1$, $F2$, $F3$), ($F1$, $F2$, $F3$, $F5$), and ($F1$, $F2$, $F4$). Considering the component failure events only these give three cut sets: $\{a, b, c\}$, $\{a, b, d\}$, and $\{a, c\}$. The BDD is not in its minimal form, therefore, it does not generate minimal cut sets. Since cut set $\{a, c\}$ will fail the system it does not matter if b fails or not and so the cut set $\{a, b, c\}$ needs to be removed. The structure needs to undergo the minimization procedure, presented in [**1**], after which the redundant combinations will be eliminated and the resulting BDD structure will encode the minimal cut sets. In this example, the minimization process will result in the terminal 1 vertex of node $F3$ being replaced with a terminal 0 vertex, and redundant cut set $\{a, b, c\}$ will be removed.

In the analysis strategy presented in this paper, causes of the original fault tree top event are represented by a set of modularized elements. Qualitative analysis therefore has to consider BDDs encoding complex events and/or modular events. The algorithm that performs this obtains the minimal cut sets of the system by extracting the minimal combinations of component failures from every complex and modular event. This is necessary because when reduction and modularization are used to construct the BDDs, it is essential to be able to analyse the system in terms of its original components.

The minimal cut sets for every BDD and complex event are required to represent the failure mode of the system determined by the original fault tree. The calculation process for the system level minimal cut sets then starts with the minimal cut sets produced for the primary BDD (that which represents the top event of the original fault tree). These may contain other modules or complex events. The results obtained for the modules or complex events are substituted into the list. This process continues as illustrated below until only the original basic events appear.

A key point of the algorithm, which is the same as the MOCUS method [**7**] for calculating minimal cut sets from fault trees, is that an AND gate increases the number of basic events in each minimal cut set and an OR gate increases the number of minimal cut sets in the system. A two dimensional array is created. Each line in the array represents a cut set. Each column is an element in the cut set. At the start the top event gate is located in the first row and the first column of the two-dimensional array. Then the array is scanned repeatedly replacing:

(a) each complex event that is an OR gate by a vertical expansion including the input events to the gate (duplicating all other events in this row);



Top

(i)

*M*1
2000

(ii)

*M*2	2001
a	
b	

(iii)

f	2001
g	2001
2002	2001
a	
b	

(iv)

f	2001
g	2001
e	2001
c	2001
a	
b	

(v)

f	*c*	*d*
g	*c*	*d*
e	*c*	*d*
h	*c*	*d*
a		
b		

(vi)

Fig. 7 Extracting minimal cut sets from modular and complex events

(b) each complex event that is an AND gate by a horizontal expansion including the input events to the gate;

(c) each modular event by a vertical and/or horizontal expansion including the list of minimal cut sets obtained from the BDD, that represents the modular event;

until only basic events appear in the array. Qualitative analysis using this algorithm will be performed for the example in Fig. 5 with complex events defined in Table 1.

The extraction of minimal cut sets from modular and complex events is presented in Fig. 7. First of all, the top event is modular, the primary BDD produces two minimal cut sets

$$\{M1\},\{2000\}$$

which replace the top event in the array, as shown in Fig. 7(ii). Second, performing a qualitative analysis of the BDD of module *M*1 gives the minimal cut set

$$\{M2{,}2001\}$$

This minimal cut set replaces *M*1 in a horizontal expansion.

Since complex event 2000 = *a* OR *b*, its inputs *a* and *b* replace the gate in a vertical expansion. This gives the representation shown in Fig. 7(iii). From the array in Fig. 7(iii), *M*2 can now be replaced. *M*2 produces three minimal cut sets

$$\{f\},\{g\},\{2002\}$$

They result in another vertical expansion in the array, duplicating the other elements in the row – in this case 2001, to produce the array shown in Fig. 7(iv). Finally, the inputs for complex events 2002 and 2001 are expanded to give the arrays of steps v and vi respectively.

The minimal cut sets in the array contain only basic events, therefore the calculation is finished. The minimal cut sets of the fault tree, presented in Fig. 2, are

$$\{a\},\{b\},\{f,c,d\},\{g,c,d\},\{e,c,d\},\{h,c,d\}$$

Since each of the modules and complex events (which are mini-modules) are independent the rows in the array will contain the minimal cut sets. It is recognized that the two dimensional array is an efficient representation of this information and is used mainly as a means to demonstrate the process. A practical implementation would use a single dimensional array with a more complex house-keeping routine.

5 CALCULATION OF MINIMAL CUT SETS WITH TRUNCATION APPROXIMATIONS

The computation of minimal cut sets for very large fault trees can be time-consuming. At times the computation may be too intensive or the problem too large to solve in real time. In this case the time taken to perform the analysis can be reduced by applying truncation approximations. The algorithm for calculating minimal cut sets, presented in reference [**1**], may be extended in order to obtain only truncated minimal cut sets that are the most significant ones. Truncation may be performed such that only minimal cut sets with less than or equal to a predefined order (number of events in the minimal cut set) are retained, or that only minimal cut sets whose probability is greater than a cut off are retained. If the probability of the minimal cut set, represented by a path through a BDD, is smaller than the predefined truncation value, the path corresponding to this minimal cut set does not need to be considered further. The same strategy is followed if the order of the minimal cut set is larger than the assigned maximum order.

For example, for the BDD in Fig. 8, if we are only interested in first- and second-order component failure combinations that cause the system failure mode, the calculations should be stopped before traversing the 1 branch of node *F*2, because at this point there are already two component failures on the path and the system state is still undetermined. Further failures would be required to cause system failure, which would exceed the cut off level and so a terminal 0 vertex replaces *F*3 in the minimal BDD as illustrated in Fig. 8. Therefore, the only minimal cut set obtained is {*a*, *c*}.

When all BDDs representing modules have been considered in this way, the results now need to be combined to obtain truncated minimal cut sets for the original top event. In this extraction algorithm the minimal cut sets are deleted from the list as they are being formed (even if not completely defined) as soon as the maximum order of the minimal cut set or the minimum probability value of the minimal cut set to occur are reached. If the minimal cut sets need to be truncated according to the maximum order, the array of minimal cut sets is scanned repeatedly and:

(a) each complex event that is an OR gate is replaced by a vertical expansion including the input events to the gate;
(b) each complex event that is an AND gate is replaced by a horizontal expansion including the input events to the gate under the condition that the number of events in every set does not exceed the assigned maximum order;
(c) each modular event is replaced by a vertical and/or horizontal expansion including the list of truncated minimal cut sets obtained from the BDD, which represents the modular event, under the condition, applied in case (b);
(d) each minimal cut set with an unreplaced modular or complex event is deleted.

These steps are applied until only basic events appear in the array.

A similar algorithm is applied for truncation according to the probability of a minimal cut set occurrence. In this case, the condition in the algorithm is that the minimal cut set is deleted if the probability of the basic events currently existing in the minimal cut set is smaller than the assigned value, as any other event added to the minimal cut set will reduce the probability further.

In the previous example in Fig. 7, if the maximum order is set to be to two, complex event 2001 in Fig. 7(v) is not replaced, because this would result in four minimal cut sets of order three. Therefore, the minimal cut sets with

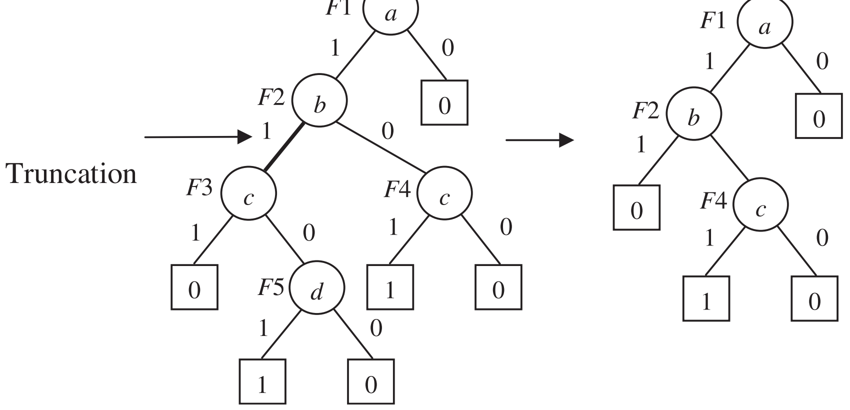


Fig. 8 Truncation of minimal cut sets of the order greater than 2

complex event 2001 are deleted. Fig. 9 represents this truncation.

6 ANALYSIS USING ORIGINAL AND SIMPLIFIED FTs

An analysis has been conducted on the fault tree-to-BDD conversion process. In this analysis some example fault trees were converted to BDDs and then qualitative and quantitative analysis performed. Seven example fault trees were analysed by applying the BDD method to both the original and the simplified fault trees. Table 5 provides a summary of the results for each fault tree.

The second and third columns of the table give some indications of the complexity of the chosen example fault trees with the number of gates and basic events. The results of the two simplification techniques are shown in the fourth and fifth columns, which represent the number of complex and modular events respectively. The reduction technique has reduced the size of the problem remarkably, especially for examples 2 and 3. The modularization technique produced two modules for each of examples 1, 3, and 5, whereas for the other examples it did not extract any modules except the module for the top event. (This is because the complex factors had already reduced the tree structure to a very efficient form.)

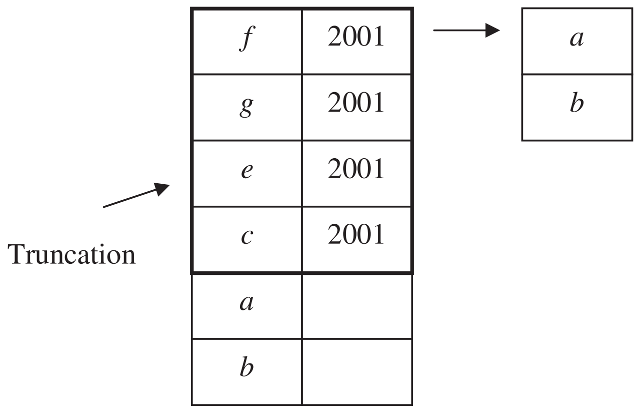


Fig. 9 Truncation of minimal cut sets of the order greater than two; the final step of the process, presented in Fig. 7

The sixth and seventh columns show the number of nodes in BDDs, which were obtained using the simplified and the original fault-tree data respectively. The simplification procedure decreased the size of the BDD remarkably. The number of nodes decreased by approximately one half (example 5 – by a factor of more than 20) when the simplification rules on the fault trees were applied. Extraction of modules and complex events had a crucial effect on the biggest trees (examples 6 and 7) because it enabled the conversion process of fault trees to BDDs, whereas owing to the size of the BDDs, the process failed if the original fault tree structures were used. (BDDs could not be formed in the memory resources available.)

The eighth column represents the number of minimal cut sets in the solution. This again indicates the complexity of the problem. The last two columns give the time taken to perform the analysis if simplified and original fault trees respectively were used. The time decreased when simplification rules were applied because smaller BDDs were obtained. Since the conversion process for example 6 and 7 failed, the entries for the time are not reported because neither quantitative, nor qualitative, analysis was able to be performed.

The computation of big fault trees can be time-consuming. In order to find out which part of the analysis utilized the most resources the analysis was performed on a library of 338 example fault trees. The results, averaged over the examples, were:

(a) reduction 1 per cent
(b) modularization 0 per cent
(c) BDD construction 14 per cent
(d) minimization 13 per cent
(e) obtaining minimal cut sets 71 per cent
(f) quantification 1 per cent

The two phases of the simplification process, the construction and minimization of BDDs, the calculation of minimal cut sets, and quantification were investigated. The most time-consuming parts of the

Table 5 Calculation results for seven example fault trees

Example	Number of gates	Number of basic events	Number of complex events	Number of modules	Number of nodes in BDD with simplifications	Number of nodes in BDD without simplifications	Number of minimal cut sets	Time taken with simplifications	Time taken without simplifications
1	25	60	33	2	116	245	79	0.156	0.172
2	31	55	15	1	566	891	262	0.156	0.359
3	32	46	4	2	1467	2056	409	1.375	2.172
4	28	65	31	1	679	1228	1112	0.484	1.047
5	40	98	66	2	731	15078	2072	2.859	-
6	50	152	151	1	1	-	14 669	4.170	-
7	56	146	145	1	1	-	2202 755	1221.160	-

analysis were the calculation of minimal cut sets and the construction and minimization of BDDs. The simplification process and quantification were the least time-consuming parts of the analysis.

The time taken to calculate the minimal cut sets and finish the analysis can be reduced if the truncation process for minimal cut sets is applied. A larger reduction is noticed when investigating large fault trees. If the order to truncate the minimal cut sets was set to be two, i.e. only the single and dual order failures were investigated, the average time taken to perform the analysis for fourteen large fault trees chosen decreased by 18 per cent. The average time distribution for the separate parts of the analysis were:

(a) reduction 0 per cent
(b) modularization 0 per cent
(c) BDD construction 48 per cent
(d) minimization 15 per cent
(e) obtaining minimal cut sets 36 per cent
(f) quantification 1 per cent

In this case, the truncation of minimal cut sets reduced the resources needed for their calculation. This truncation decreased the time taken to perform the analysis. These results are only indicative of the processing effort distribution over the analysis. It does however indicate areas in which to concentrate effort to gain further improvement in efficiency.

7 CONCLUSIONS

This paper has presented a procedure by which large fault trees can be simplified prior to conversion to their BDD form for analysis. In this procedure simplification is performed in two phases, the first reduces the fault tree to its more concise form which removes the noise from the failure logic and retains the underlying problem structure. The second phase identifies independent modules, which can be analysed separately. In doing this the problem can be solved efficiently. Having performed the simplification the problem is solved in terms of the new modular structure and complex events. A means of calculating minimal cut sets in terms of the original basic events has also been presented. The approach is capable of using truncation methods to yield only the important minimal cut sets. Finally, an assessment was performed on the analysis showing which aspects of the conversion, qualification, and quantification utilize the most resources.

REFERENCES

1 **Rauzy A.** New algorithms for fault tree analysis. *Reliab. Eng. Syst. Safety*, **40**, 1993, 203–211.

2 **Vesely, W. E.** A time dependent methodology for fault tree evaluation. *Nuc. Des. and Engng*, 1970, **13**, 337–360.

3 **Platz, O.** and **Olsen, J.-V.** *FAUNET: A program package for evaluation of fault trees and networks*, Research Establishment Risk Report, Sept. 1976, No. 348, DK-4000 Roskilde, Denmark.

4 **Dutuit, Y.** and **Rauzy, A.** A linear-time algorithm to find modules of fault trees. IEEE Trans. Reliability, 1996, **45**, No. 3, 422–425.

5 **Reay, K. A.** and **Andrews, J. D.** A fault tree analysis strategy using binary decision diagrams. *Rel. Engng and Syst Safety*, **78**, 2002, 45–56.

6 **Sinnamon, R. M.** and **Andrews, J. D.** Improved efficiency in qualitative fault tree analysis. *Qual. and Rel. Engng Int.* 1997, **13**, 293–298.

7 **Andrews, J. D.** and **Moss, T. R.** Reliability and risk assessment, 2002, Professional Engineering Publishers.

APPENDIX

Notation

a	basic event
AND	gate type AND
f_1	logic function on the 1 branch of a node
f_0	logic function on the 0 branch of a node
$F1$	number of the node
$G1$	gate
ite	if-then-else structure
$M1$	modular event
OR	gate type OR
2000	complex event
$\oplus$	operation between two gates or events