Prime Implicants for Modularised Non-coherent Fault Trees Using Binary Decision Diagrams

Rasa Remenyte-Prescott, John D. Andrews
Aeronautical and Automotive Engineering , Loughborough University, Loughborough, UK

Keywords
Fault Tree Analysis, Binary Decision Diagrams, Non-coherent systems, prime implicants

Abstract

This paper presents an extended strategy for the analysis of complex fault trees. The method utilises simplification rules, which 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 prime implicant sets for the original top event. It is shown how to extract the prime implicant sets from complex and modular events in order to obtain the prime implicant sets of the original fault tree in terms of basic events.

1. Introduction

Fault tree analysis was first conceived in the 1960's and provides a good representation of the system from an engineering viewpoint. This form of the failure logic function does not however lend itself to easy and accurate mathematical manipulation. A more convenient form for the logic function from the mathematical viewpoint is that of a Binary Decision Diagram [1,2]. It overcomes some disadvantages of conventional FTA techniques enabling efficient and exact qualitative and quantitative analyses of both coherent and non-coherent fault trees.

The difficulty of the BDD method is with the conversion process of the fault tree to the BDD. An aspect of this is that the basic events in the fault tree have to be placed in an ordering required to construct the BDD. A good ordering gives a concise BDD form. A bad ordering may lead to an explosion in the size of the BDD used to represent the fault tree. To date an optimum strategy for producing BDDs for all fault trees has not been identified.

Non-coherency introduces difficulties in the qualitative assessment of the fault tree compared with the coherent version. The failure modes of the system, combinations of working or failed components which result in system failure, tend to be greater in number and larger in order than the coherent version. This can render the qualitative evaluation intractable. The BDD method is more efficient for analysing a system without the need for the approximations used in the traditional approach of kinetic tree theory [3].

In this paper two simplification strategies that have been shown to be effective in reducing the complexity of the problem are applied: reduction [4] and modularisation

[5]. The reduction technique simplifies the fault tree to its minimal logic form, whilst modularisation breaks down the fault tree to independent subtrees that can be analysed separately.

Then BDDs are obtained for each module in separate computations, culminating in a set of BDDs, which together represent the original system failure diagram. A qualitative analysis of non-coherent fault trees using BDDs is the subject of this paper. Meta-products BDDs [6,7] used to determine the prime implicant sets are then developed. Then prime implicant sets for every module can be calculated and extracted for the whole system. Each of these stages is described in detail in the following sections and demonstrated throughout by the use of an example.

2. Non-coherent fault tree structure

In a coherent fault tree each component in the system is relevant, and the structure function is monotonically increasing [8]. A fault tree that contains only AND gates and OR gates is always coherent. Whenever a NOT logic gate is introduced into a fault tree, it is likely to become non-coherent. A fault tree is non-coherent when both component failure and working states (positive and negative events) contribute cause of the top event. For example, system failure might occur due to the recovery of a failed component. Alternatively, during system failure, the failure of an additional component may bring the system to a good state.

Consider an example of the traffic light system [8] shown in Figure 1.

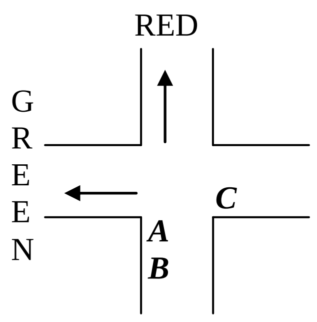

Figure 1. Traffic light system

Cars A and B are approaching the lights on RED and should stop. Car C is approaching the junction with lights on GREEN and should proceed. Given this scenario, the following failure events can occur:

- A - Car A fails to stop
- B - Car B fails to stop
- C - Car C fails to continue

An accident at this crossroads can occur in two ways:

- Car A acts properly and stops ($\overline{A}$) and car B fails to stop (B) and drives into the back of A.
- Car A fails to stop (A) and if car C continues to move into the crossing ($\overline{C}$), A hits C.

The fault tree to represent the causes of an accident situation at this crossroads is illustrated in Figure 2.

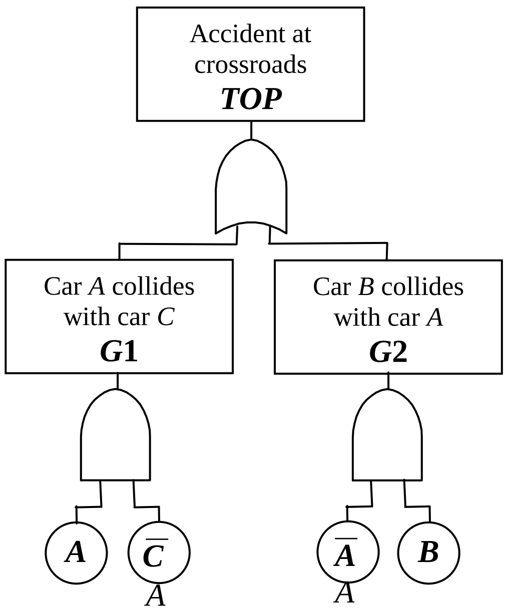

Figure 2. Fault tree for traffic light system

Working in a top-down way the following logic expression is obtained, which is in disjunctive normal form (minimal sum of products)

$$Top = A \cdot \overline{C} + \overline{A} \cdot B. \tag{1}$$

where "+" is OR, "·" is AND.
A qualitative analysis of the non-coherent fault tree will produce the system failure modes known as prime implicant sets. Prime implicants are defined as combinations of component conditions (working or failed) which are necessary and sufficient to cause system failure. Taking the produced terms from equation 1 gives prime implicant sets $\{A\overline{C}\}$ and $\{\overline{A}B\}$.

However, unlike the reduction of the logic expression for a coherent fault tree the logic equation for the top event in the non-coherent case does not produce a complete list of all prime implicants. In this example, if car *B* fails to stop and car *C* continues across the lights there will be a collision whatever *A* does. Therefore, the full logic expression for the *Top* event is:

$$Top = A \cdot \overline{C} + \overline{A} \cdot B + B \cdot \overline{C}, \tag{2}$$

which can be obtained by applying the consensus law:

$$A \cdot X + \overline{A} \cdot Y = A \cdot X + \overline{A} \cdot Y + X \cdot Y. \tag{3}$$

This example is included just as a means to demonstrate issues for non-coherent fault trees. The increase in length of the logic equation for non-coherent fault trees can cause difficulties for large fault trees and is at times necessary [8]. The inclusion of NOT logic gates is used in Event Tree Analysis where there are component failures that occur in more than one fault tree which represent the causes of the branching [9]. Also, NOT logic is important in Phased Mission Analysis [10].

It is possible to reduce the prime implicant sets to their coherent approximations prior to the fault tree quantification. This is achieved by removing all success states and re-minimising the expression (on the basis of likelihood is close to 1). This will give minimal cut sets {*A*} and {*B*}.

3. Simplification of the fault tree structure

Dealing with complex industrial systems can result in very large fault trees, whose analysis is time consuming. 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 part of the simplification process is a reduction technique [4] which resizes the fault tree to its simplest form. The second part identifies independent modules (subtrees) within the fault tree that can be dealt with separately [5]. The linear-time algorithm is applied to the second part and a set of independent fault trees in their simplest possible structure is obtained. It is equivalent to the original system failure causes and is easier to manipulate during the analysis process.

The analysis strategy of coherent fault trees incorporating the simplification process was presented in [11]. This paper provides a development of this technique in the non-coherent case.

3.1. Reduction

This technique reduces the fault tree to its simplest form while retaining its logical structure. It is applied in four stages:

- contraction,
- factorisation,
- extraction,
- absorption.

First of all, the fault tree is manipulated so that the NOT logic is "pushed" down the fault tree until it is applied to basic events using De Morgan's laws, i.e.

$$\overline{A \cdot B} = \overline{A} + \overline{B} \tag{4}$$

$$\overline{A + B} = \overline{A} \cdot \overline{B} \tag{5}$$

Then for the contraction, subsequent gates of the same type are contracted to form a single gate so that the fault tree becomes an alternating sequence of AND and OR gates.

During the factorisation, pairs of events that always occur together as inputs to the same gate type are identified and combined forming a single complex event. If events appear in their working and failed states in the fault tree, only those basic events that appear together in their negated state under the opposite gate type can be combined. (Note: in this sense the "AND" type gate is opposite to the "OR" type gate). By De Morgan's equations 4 and 5, if $a + b$ and/or $\overline{a} \cdot \overline{b}$ appear in the fault tree, then $a + b$ forms a complex event, or if $a \cdot b$ and/or $\overline{a} + \overline{b}$ appear in the fault tree, then $a \cdot b$ forms a complex event. The complex events identified are then substituted into the fault tree structure.

In the extraction stage, the two structures shown in Figure 3 are identified and replaced in order to reduce the repeated occurrence of events to a single occurrence and facilitate further reduction.

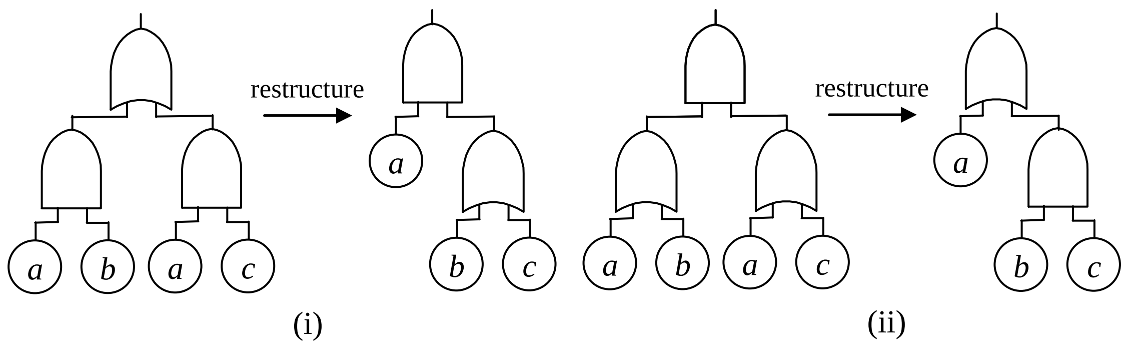

Figure 3. Reduction, the extraction procedure

If another component is repeated in the structure and it is repeated in its negated state, the structures shown in Figure 4 can be simplified even more, i.e. the whole structure is replaced by the component that appears only in one state, failed or working.

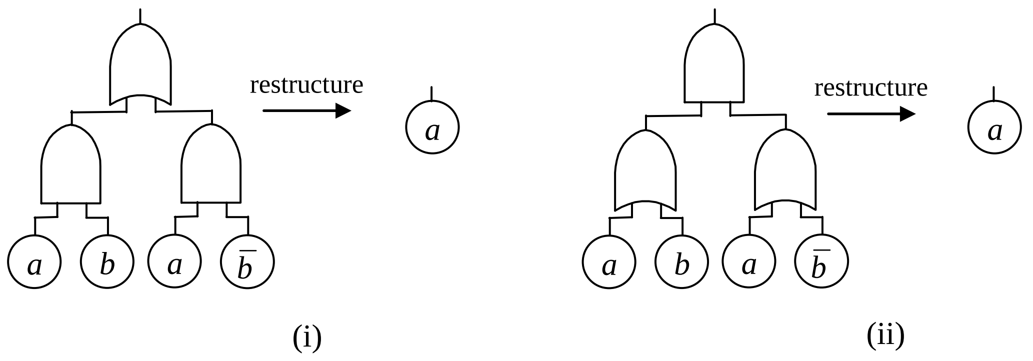

Figure 4. Reduction, the extraction procedure in non-coherent case

During absorption, structures were identified that could be further simplified through the application of the absorption and idempotent laws to the fault tree logic. The simplification process can be applied when an event is repeated in the branch of the fault tree structure. The gate with the first occurrence of the repeated event is called a primary gate and a gate with the second occurrence of the repeated event is called a secondary gate. There can be two types of the repetition in a non-coherent system, i.e. a component can be repeated in the same state or it can be repeated in its negated state.

Considering the first case, if the primary and the secondary gates with an event in common are of different type (Figure 5(i)), the structure is simplified by removing the whole secondary gate and its descendants. If the primary and the secondary gates are of the same type (Figure 5(ii)), the structure is simplified by deleting the occurrence of the event beneath the secondary gate.

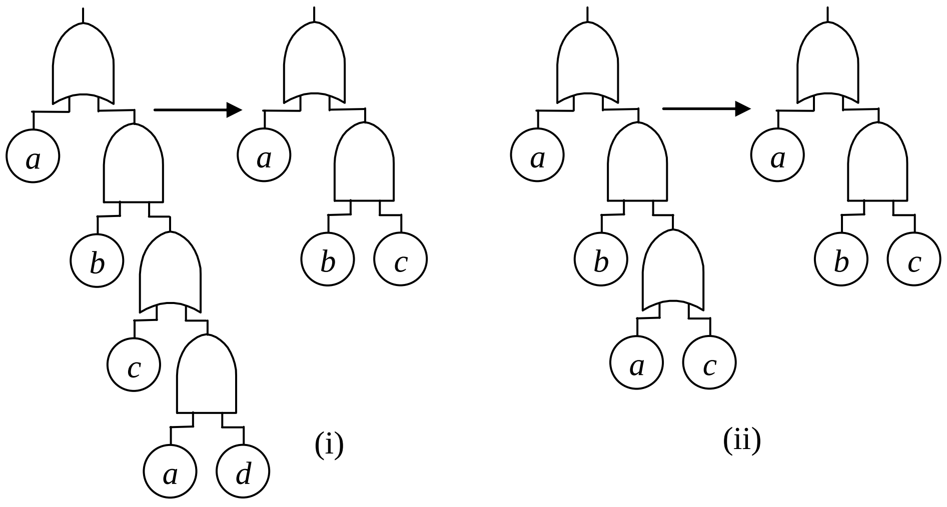

Figure 5. Reduction, the absorption procedure

If a component is repeated in its negated state, the absorption rule can again be applied. In this case the absorption cannot be applied if the primary gate is an OR gate. Therefore, if the primary gate is an AND gate and the secondary gate is an OR gate (Figure 6(i)), then the structure is simplified by deleting the occurrence of the event beneath the secondary gate. If both the primary gate and the secondary gate are AND gates (Figure 6(ii)), the whole secondary gate can be deleted. The order of appearance of positive and negative events in primary and secondary gates is irrelevant.

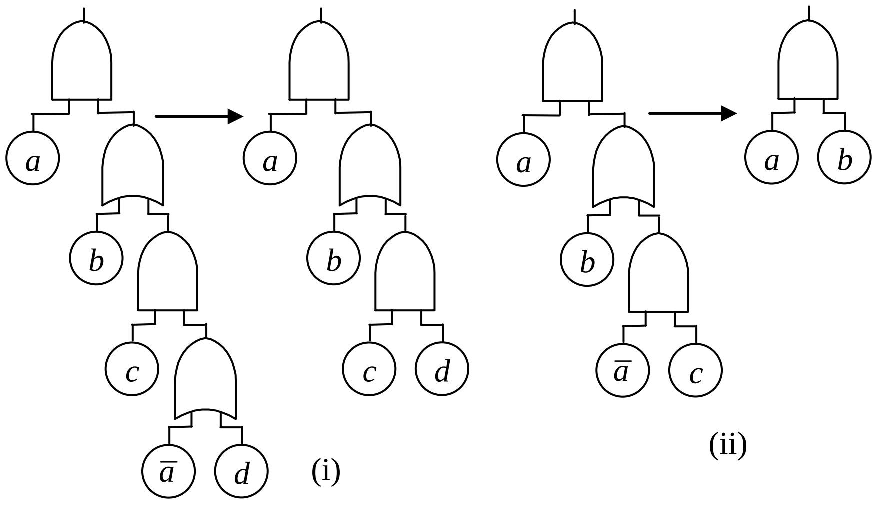

Figure 6. Reduction, the absorption procedure in non-coherent case

The above four steps are repeated until no further changes take place in the fault tree. An example of the application of these rules to a fault tree is demonstrated considering the fault tree shown in Figure 7.

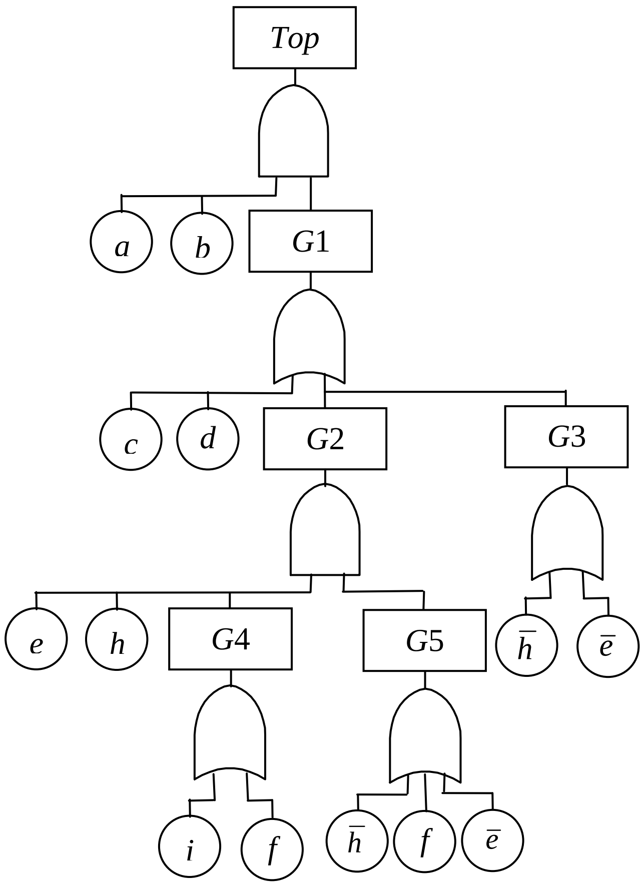

Figure 7. Example fault tree

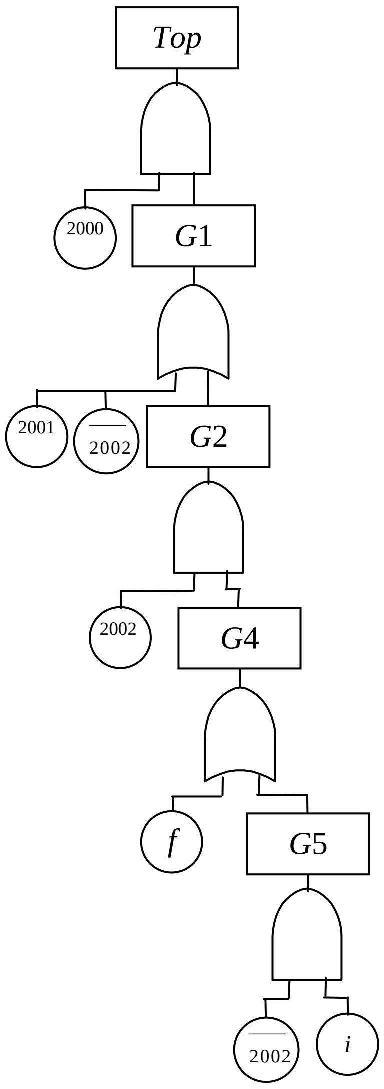

Figure 8. Reduced fault tree 1

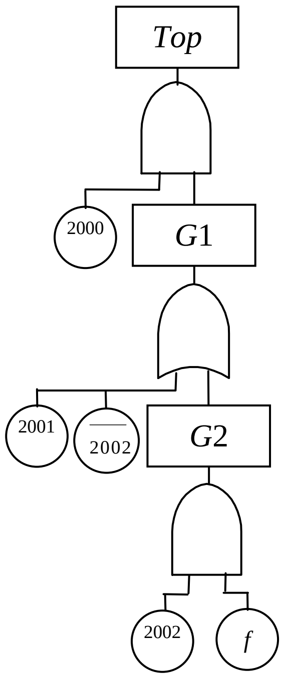

Figure 9. Reduced fault tree 2

Contraction was performed for gates *G*1 and *G*3, which are of the same type, forming a single gate. Factorisation was performed three times: for the pair of basic events *a* AND *b* for the pair of basic event *c* OR *d* and for the pair of basic events *e* AND *h*, creating complex events 2000, 2001 and 2002 respectively. Basic events $\bar{e}$ OR $\bar{h}$ form complex event $\overline{2002}$. The corresponding complex event data are shown in Table 1. The extraction procedure was applied to gates *G*4 and *G*5, extracting the repeated event *f*. The simplified fault tree is shown in Figure 8. Finally, the absorption was applied to gate *G*2 removing gate *G*5. No further simplification is possible. The final reduced tree is shown in Figure 9.

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

Table 1. The complex event data

Reduction has simplified the example fault tree. In the original tree there were six gates, in the reduced tree there are 3 gates. There were thirteen events in the original fault tree, eight of them different; in the reduced tree there are five events, and four 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 modularisation is considered.

3.2. Modularisation

The modularisation technique identifies independent subtrees within the fault tree. Every module contains no basic events that appear elsewhere in the fault tree and it can be analysed separately. The results from each module are substituted into the higher level fault trees where modules occur and further stages of the analysis are performed.

For the purposes of the process basic events in their positive or negative form are considered as the same entity as it is only the vent label that is important in identifying independent modules. Using the linear-time algorithm [5] the independent modules can be identified after just two depth-first traversals of the fault tree. During the first time of the traversal the step number at the first, second and final visits to every node (gate and event) is recorded. Then 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 Figure 9 are presented in Tables 2, 3 and 4 respectively.

Step number	1	2	3	4	5	6	7	8	9	10	11
Node	*Top*	2000	*G*1	2001	$\overline{2002}$	*G*2	2002	*f*	*G*2	*G*1	*Top*

Table 2. Step numbers for every node in the fault tree

Gate	*Top*	*G*1	*G*2
1st visit	1	3	6
2nd visit	11	10	9
Final visit	11	10	9
Min	2	4	5
Max	10	9	8

Table 3. Data for gates in the fault tree

Event	2000	2001	2002, $\overline{2002}$	*f*
1st visit	2	4	5	8
2nd visit	2	4	7	8
Final visit	2	4	7	8

Table 4. Data for events in the fault tree

The rules for identifying a gate as heading a module are:

- The first visit to each descendant of a gate is after the first visit to the gate and
- The last visit to each descendant of a gate is before the second visit to the gate.

The following gates can be identified as heading modules:

Top, *G*1.

Gate *G*2 can not be a module because one its descendant (event 2002) is visited before visiting parent gate.

Gate *G*1 is replaced by modular event M_1. Two separate fault trees, shown in Figure 10, now replace the fault tree in Figure 9.

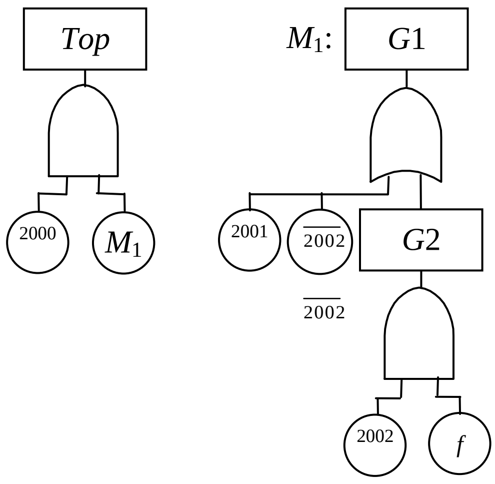

Figure 10. The two modules obtained for the fault tree shown in Figure 9

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

4. Conversion to binary decision diagrams

Every independent fault tree is converted to a BDD. The variable ordering needs to be established before the conversion process. 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:

$$\begin{aligned} &\mathit{Top}: && 2000 < M_1, \\ &M1: && 2001 < 2002 < f, \end{aligned}$$

The BDD construction algorithm is described in reference [2]. The resulting SFBDD encodes the structure function of a non-coherent fault tree. The background of it is to assign an ordered triple to each node in the SFBDD:

$$\mathbf{ite}(x,f_1,f_0), \tag{6}$$

here x is the Boolean variable representing a basic event as the node and f_1 and f_0 are the logic functions on its 1 branch and 0 branch respectively. **ite** is described as if-then-else operation, i.e.

```
If   x occurs
     then consider f_1,
     else consider f_0.
endif
```

SFBDD 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:

$$a = \mathbf{ite}(a,1,0). \tag{7}$$

Alternatively, a basic event $\bar{a}$ is assigned an **ite** structure:

$$\bar{a} = \mathbf{ite}(a,0,1). \tag{8}$$

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

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

$$\text{then } J \oplus H = \begin{cases} \mathbf{ifre}(x, K_1, K_0, K_1 \cdot K_0) & \text{if } x < y \text{ in the ordering,} \\ \mathbf{ifre}(x, L_1, L_0, L_1 \cdot L_0) & \text{if } x = y \text{ in the ordering.} \end{cases} \tag{9}$$

Applying these rules to the fault tree in Figure 10 results in the set of SFBDDs presented in Figure 11.

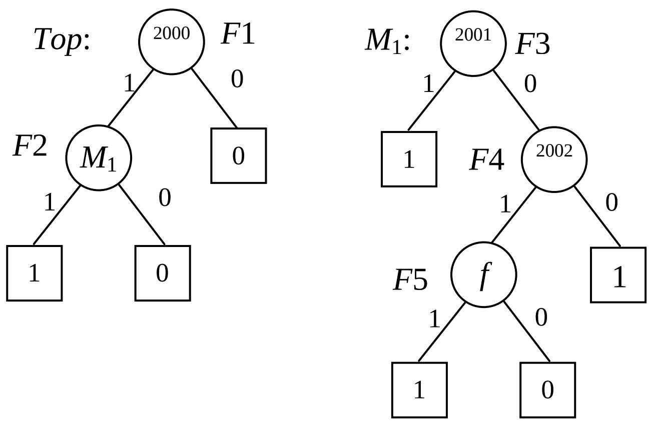

Figure 11. The obtained SFBDDs for the modules shown in Figure 10

The qualitative and quantitative analyses can be carried out since the complete set of SFBDDs have been computed. Now the calculation of prime implicant sets will be presented using the SFBDDs obtained from simplified fault trees.

5. Calculation of prime implicant sets

Knowledge of prime implicant sets can be valuable in gaining an understanding of the system. It can also help to develop a repair schedule for failed components if a system cannot be taken off line for repair. The SFBDD which encodes the structure function cannot be used directly to produce the complete list of prime implicant sets of a non-coherent fault tree. For example, consider a general component *x* in a non-coherent system. Component *x* can be in a failed or working state, or can be excluded from the failure mode. In the first two situations *x* is said to be relevant, in the third case it is irrelevant to the system state. Component *x* can be either failure relevant (the prime implicant set contains *x*) or repair relevant (the prime implicant set contains $\bar{x}$). A general node in the SFBDD, which represents component *x*, has two branches. The 1 branch corresponds to the failure of *x*; therefore, *x* is either failure relevant or irrelevant. Similarly, the 0 branch corresponds to the functioning of *x* and so *x* is either repair relevant or irrelevant. Hence it is impossible to distinguish between the two cases for each branch and the prime implicant sets cannot be identified from the SFBDD.

There is a number of techniques developed for calculating prime implicant sets. The first approach was presented by Courdet and Madre [6] and further developed by Dutuit and Rauzy [7] where a meta-products BDD is formed. This technique is used in the paper. Some other alternative techniques for calculating prime implicant sets were presented by Sasao in [12], by Rauzy in [13] and by Contini in [14]. They use alternative notations for the representation of prime implicants.

In meta-products BDD method developed an alternative notation is developed that associates two variables with every component *x*. The first variable, P_x, denotes relevancy and the second variable, S_x, denotes the type of relevancy, i.e. failure or repair relevant. A meta-product, MP(π), is the intersection of all the system components according to their relevancy to the system state and π represents the prime implicant set encoded in meta-product MP(π).

$$MP(\pi) = \begin{cases} P_x \wedge S_x \; if \;\; x \in \pi \\ P_x \wedge \overline{S}_x \; if \;\; \overline{x} \in \pi \\ \overline{P}_x \; if \;\; neither \;\; x \; nor \; \overline{x} \; belongs \; to \; \pi \end{cases} \qquad (10)$$

The proposed algorithm is used for calculating the meta-products BDD of a fault tree from the SFBDD. The meta-products BDD is always minimal, therefore it encodes the prime implicant sets exactly.

In order to present the algorithm, consider node F in a SFBDD, where F = **ite**(x, $F1$, $F0$). The meta-products BDD, that describes prime implicant sets using equation 12, is expressed as:

$$\mathrm{PI}(F) = \mathbf{ite}(P_x, \mathbf{ite}(S_x, P1, P0), P2), \qquad (11)$$

where

$$P2 = \mathrm{PI}(F1 \cdot F0), \qquad (12)$$

$$P1 = \mathrm{PI}(F1) \cdot \overline{P2}, \qquad (13)$$

$$P0 = \mathrm{PI}(F0) \cdot \overline{P2}. \qquad (14)$$

x is the first element in the variable ordering, PI(F) represents the structure of a meta-products BDD.

$P2$ encodes the prime implicants for which x is irrelevant, $P1$ encodes the prime implicants for which x is failure relevant and $P0$ encodes the prime implicants for which x is repair relevant.

If not all the basic events in the variable ordering appear on the particular path, then

$$\mathrm{PI}(F) = \mathbf{ite}(P_{x_j}, 0, \mathrm{PI}(F)). \qquad (15)$$

here x_j is before x_i and x_j does not appear on the current path from the root node to node F.

If F is a terminal node, then

$$\mathrm{PI}(0) = 0, \qquad (16)$$

$$\mathrm{PI}(1) = \mathbf{ite}(P_{x_i}, 0, \mathbf{ite}(P_{x_{i+1}}, 0, \ldots, \mathbf{ite}(P_{x_n}, 0, 1))). \qquad (17)$$

where x_i, …, x_n are basic events from the variable ordering that have not yet been considered in the process.

***

Consider the BDD for *Top* module in Figure 11. Applying equations 11-14 to node $F1$ gives:

$$\mathrm{PI}(F1) = \mathbf{ite}(P_{2000}, \mathbf{ite}(S_{2000}, \mathrm{PI}(F2) \cdot 1, \mathrm{PI}(0) \cdot 1), \mathrm{PI}(F2 \cdot 0)) = \mathbf{ite}(P_{2000}, \mathbf{ite}(S_{2000}, \mathrm{PI}(F2), 0), 0).$$

In the same way applying equations 11-14 to node $F2$ gives:

$$\mathrm{PI}(F2) = \mathbf{ite}(P_{M1}, \mathbf{ite}(S_{M1}, 1, 0), 0)).$$

The meta-products BDD is obtained for *Top* module.

Then the BDD for module M_1 is investigated. Applying equation 13 to node $F3$ gives:

$$\mathrm{PI}(F3) = \mathbf{ite}(\mathrm{P}_{2001}, \mathbf{ite}(\mathrm{S}_{2001}, \mathrm{PI}(1) \cdot \overline{PI(F4)}, \mathrm{PI}(F4) \cdot \overline{PI(F4)}, \mathrm{PI}(F4)).$$

First of all, PI(1) is calculated. Since there are two more variables in the ordering scheme that were not investigated yet, i.e. $2002 < f$, according to equation 17 we get:

$$\mathrm{PI}(1) = \mathbf{ite}(P_{2002}, 0, \mathbf{ite}(P_f, 0, 1)).$$

Then PI($F4$) is calculated applying equations 11-14:

$$\mathrm{PI}(F4) = \mathbf{ite}(\mathrm{P}_{2002}, \mathbf{ite}(\mathrm{S}_{2002}, \mathrm{PI}(F5) \cdot \overline{PI(F5)}, \mathrm{PI}(1) \cdot \overline{PI(F5)}, \mathrm{PI}(F5)).$$

PI(1) is calculated in the same. Since there is only one variable left in the ordering scheme, i.e. variable f, according to equation 17 it gives:

$$\mathrm{PI}(1) = \mathbf{ite}(P_f, 0, 1).$$

PI($F5$) is obtained applying equations 11-14 to node $F5$. i.e.

$$\mathrm{PI}(F5) = \mathbf{ite}(P_f, \mathbf{ite}(S_f, 1, 0), 0).$$

During the negation of PI($F5$) terminal vertices 1 and vertices 0 are swapped in the meta-products BDD. Therefore,

$$\overline{PI(F5)} = \mathbf{ite}(P_f, \mathbf{ite}(S_f, 0, 1), 1).$$

Since there are no repeated parts between PI(1) and $\overline{PI(F5)}$,

$$\mathrm{PI}(1) \cdot \overline{PI(F5)} = \mathrm{PI}(1).$$

Then the calculation of PI($F4$) is completed:

$$\mathrm{PI}(F4) = \mathbf{ite}(\mathrm{P}_{2002}, \mathbf{ite}(\mathrm{S}_{2002}, 0, \mathbf{ite}(P_f, 0, 1)), \mathbf{ite}(P_f, \mathbf{ite}(S_f, 1, 0), 0)).$$

$\overline{PI(F4)}$ is obtained in the same way, swapping terminal vertices 1 and 0. Because there are no repeated parts between PI(1) and $\overline{PI(F4)}$,

$$\mathrm{PI}(1) \cdot \overline{PI(F4)} = \mathrm{PI}(1).$$

This assures that the meta-products BDD produced is in its minimal form. The calculation of the meta-products BDD for module M_1 is completed. The set of resulting meta-products BDDs for two fault trees in Figure 11 is shown in Figure 12.

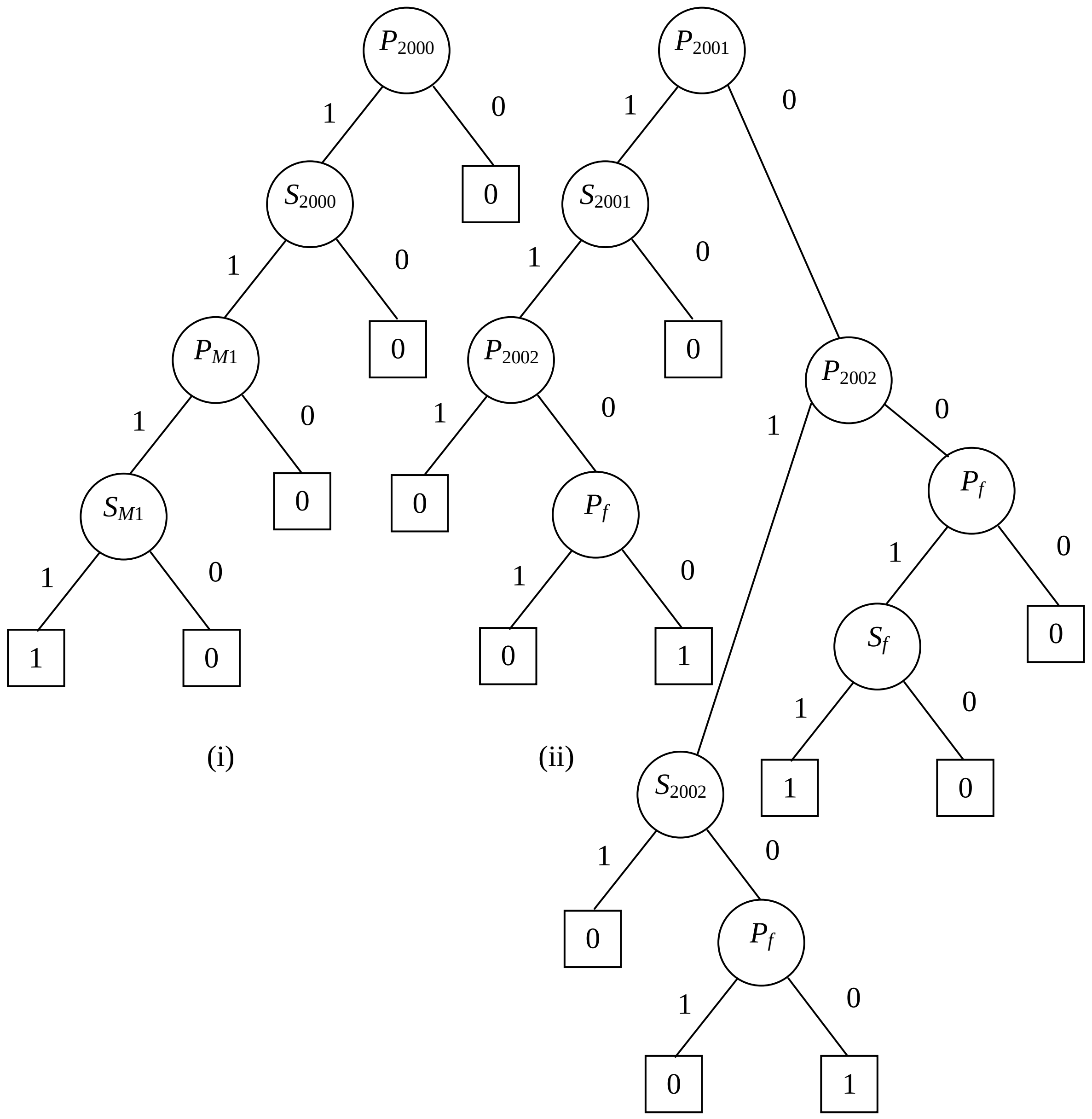

Figure 12. Meta-products BDDs for calculating prime implicant sets

Now it is possible to obtain the meta-products and identify the prime implicant sets.

First module *Top*	$P_{2000} \wedge S_{2000} \wedge P_{M_1} \wedge S_{M_1} = \{2000, M_1\}$
Second module M_1	$P_{2001} \wedge S_{2001} \wedge \overline{P}_{2002} \wedge \overline{P}_f = \{2001\}$
	$\overline{P}_{2001} \wedge P_{2002} \wedge \overline{S}_{2002} \wedge \overline{P}_f = \{\overline{2002}\}$
	$\overline{P}_{2001} \wedge \overline{P}_{2002} \wedge P_f \wedge S_f = \{f\}$

For example, in the second meta-product P_{2001} signifies that component 2001 is relevant and S_{2001} signifies that component 2001 is failure relevant. $\overline{P}_{2002}$ and $\overline{P}_f$ mean that components 2002 and f are irrelevant. Hence the prime implicant obtained is {2001}.

The prime implicant sets have been produced for every independent module, that contain modular and complex events. It is essential to be able to analyse the system in terms of its original components, therefore, the next stage of the qualitative analysis has to consider the extraction of the combinations of component failures from every complex and modular event.

A key point of the expansion algorithm, which is the same as the MOCUS method [15] for calculating minimal cut sets from fault trees, is that an AND gate increases the number of basic events in each prime implicant set and an OR gate increases the number of prime implicant sets in the system. A two dimensional array is created. Each line in the array represents a prime implicant set. At the start the top event gate is located in the first row and the first column of the two-dimensional array. Then repeatedly the array is scanned replacing:

1. Each complex event which is an OR gate by a vertical expansion including the input events to the gate (duplicating all other events in this row)
2. Each complex event which is an AND gate by a horizontal expansion including the input events to the gate
3. Each modular event (original or negated) by a vertical and/or horizontal expansion including the list of prime implicant sets obtained from the meta-products BDD, which represents the modular event (original or negated

until only basic events appear in the array.

If prime implicant sets need to be produced for the negated modular event, the original BDD for this event need to be negated, replacing its terminal vertices 1 to vertices 0 and the other way round. Then the meta-products BDD for the negated modular event can be computed using the algorithm presented in references [6,7].

Calculation of prime implicant sets using this algorithm was performed and shown in Figure 13.

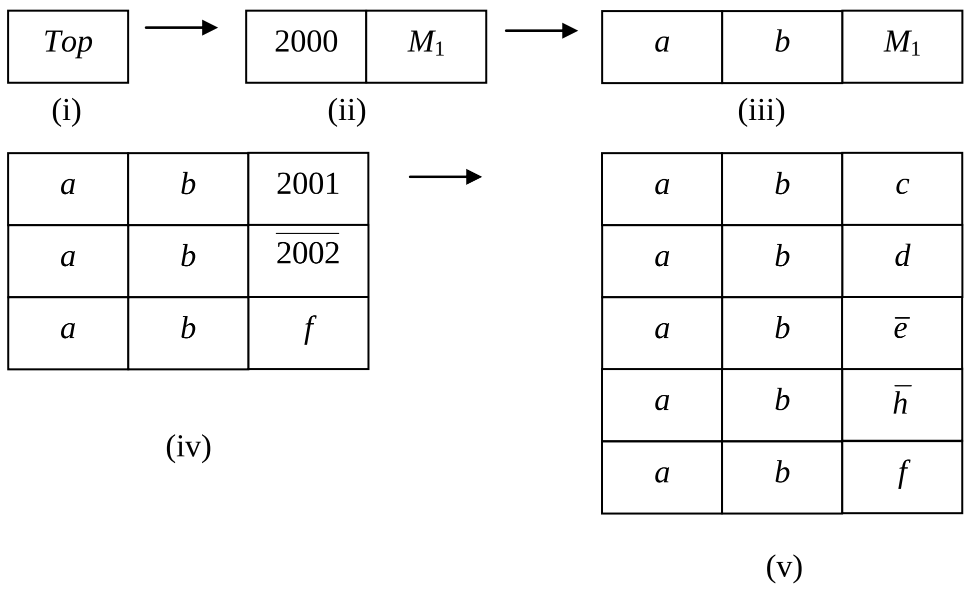

Figure 13. Extracting prime implicant sets from modular and complex events

First module – *Top* – has produced one prime implicant set

$$\{2000, M_1\},$$

which replace the top event in the array, as shown in Figure 13(ii). Since complex event 2000 = *a* AND *b*, its inputs *a* and *b* replace the gate in a horizontal expansion. The obtained array is shown in Figure 13(iii). Also, module M_1 can be replaced. M_1 produces three prime implicant set

$$\{2001\}, \{\overline{2002}\}, \{f\}.$$

This set results in a vertical expansion in the array. The resulting array is shown in Figure 13(iv).

Then the expansion of the inputs for complex event 2001 and event $\overline{2002}$ give two more horizontal expansions, shown in Figure 13(v).

The prime implicant sets in the array contain only basic events, therefore the calculation is finished. The prime implicant sets of the example fault tree, presented in Figure 7 are:

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

6. Comparison of qualitative analyses using original and simplified fault trees

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 analysis performed. Sixteen 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.

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 prime implicant sets	Time taken with simplifications	Time taken without simplifications
1	13	46	23	5	1188	81155	41310	0.172	66.86
2	17	58	39	1	5303	172219	42318	0.64	434.782
3	17	63	35	1	13738	105184	53217	1.828	109.729
4	15	69	44	1	31824	594918	1357	7.562	1573.391
5	21	16	1	1	11281	14769	20	0.687	0.938
6	27	20	2	1	4452	17459	38	0.172	1.609
7	39	27	1	1	43300	66166	105	8.25	19.734
8	27	61	31	1	50765	-	373	9.219	-
9	32	66	19	1	66406	-	67	18.421	-
10	22	55	27	1	10583	683695	339	0.859	4130.425
11	26	48	8	1	94827	336024	47	33.295	630.488
12	32	38	4	1	13656	30671	50	0.844	3.297
13	27	57	28	1	31123	533913	203	4.421	1796.779
14	21	29	13	1	13464	109767	46	1.063	61.653
15	25	41	17	1	10270	77215	74	0.547	18.921
16	20	35	9	1	34633	134464	64	6.64	99.027

Table 5. Calculation results for example fault trees

The second and the third columns give some indications of the complexity of the example fault trees, presenting 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, 3 and 4. The modularisation technique extracted five independent modules for example 1. No more modules were extracted for other examples since the complex factors had already reduced the tree structure to its most efficient form.

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. A sum of number of nodes in SFBDDs and meta-products BDDs is examined since both types of BDDs are used in the qualitative analysis of non-coherent fault trees. The simplification procedure decreased the size of the BDDs significantly.

The number of nodes decreased by approximately one half at least 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 8 and 9) because it enabled the conversion process of fault trees to BDDs, whereas due 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 prime implicant sets in the solution. This again indicates the complexity of the problem. The last two columns respectively give the time taken to perform the analysis if simplified and original fault trees were used.

The time decreased when simplification rules were applied because smaller BDDs were obtained. Since the conversion process for example 8 and 9 failed, the entries for the time are not reported because analysis was unable to be performed.

7. Conclusions

This paper presents a procedure by which large fault trees can be simplified prior to conversion to their Binary Decision Diagram form for analysis. Non-coherent fault trees are examined. Simplification is performed in two phases, the first reduces the fault tree to its more concise form 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 prime implicant sets in terms of the original basic events is presented. The efficiency of the simplification process is analysed using some example fault trees and it is compared with the analysis of fault trees that were not simplified. A comparison of two methods illustrates a favourable degree of efficiency in the simplification of fault trees prior to conversion to their BDD form for analysis.

8. References

1. Bryant, R.E. *Graph-Based Algorithms for Boolean Function Manipulation*', IEEE Trans. Computers, C-35, No.8, pp677-690, (1986)
2. Rauzy, A. *New Algorithms for Fault Tree Analysis*, Reliab Eng Syst Safety, **40**, pp203-211 (1993).
3. Vesely, W.E. *A Time Dependent Methodology for Fault Tree Evaluation*, Nuclear Design and Engineering, **13**, pp337-360, (1970).
4. Platz, O. and Olsen, J.V. *FAUNET: A program Package for Evaluation of Fault Trees and Networks*, Research Establishment Risk Report, No. 348, DK-4000 Roskilde, Denmark, Sept. (1976)
5. Dutuit, Y. and Rauzy, A. *A Linear-Time Algorithm to Find Modules of Fault Trees*, IEEE Trans. Reliability, **45**, No.3, pp422-425, (1996)
6. Courdet, O. and Madre, J.-C. *A New Method to Compute Prime and Essential Prime Implicants of Boolean Functions*, Advanced Research in VLSI and Parallel Systems, pp113-128 (1992)
7. Dutuit, Y. and Rauzy, A. *Exact and Truncated Computations of Prime Implicants of Coherent and Non-Coherent Fault Trees with Aralia*, Reliability Engineering and System Safety, **58**, pp225-235 (1997)
8. Andrews, J.D. *To Not or Not to Not!!*, Proceedings of the 18th International System Safety Conference, pp267-274 (2000)
9. Andrews, J.D. and Dunnett, S.J. *Improved Accuracy in Event Tree Analysis*, Foresight and Precaution. Cottam, Harvey, Pape and Tate (eds). Proceedings of ESREL 2000, SARS and SRA-EUROPE annual conference, pp1525-1532, (2000)
10. La Band, R.A. and Andrews, J.D. *Phased Mission Modelling Using Fault Tree Analysis*, Proceedings of the IMechE Part E Journal of Process Mechanical Engineering, **218**, pp83-91, (2004)
11. Reay, K.A. and Andrews, J.D. *A Fault Tree Analysis Strategy Using Binary Decision Diagrams*, Reliability Engineering and System Safety, **78**, pp45-56, (2002)

12. Sasao, T. *Ternary Decision Diagrams and their Applications*, Representations of Discrete Functions, Chapter 12, pp269-292 (1996)
13. Rauzy, A. *Mathematical Foundation of Minimal Cutsets*, IEEE Transactions on Reliability, volume 50, **4**, pp389-396 (2001)
14. Contini, S. *Binary Decision Diagrams with Labelled Variables for Non-Coherent Fault Tree Analysis*, European Commission, Joint Research Centre (2005)
15. Andrews, J.D. and Moss, T.R. *Reliability and Risk Assessment*, Professional Engineering Publishers, (2002)