Fault Tree Analysis and Binary Decision Diagrams

Roslyn M. Sinnamon • Loughborough University • Loughborough
John D. Andrews • Loughborough University • Loughborough

Key Words: Fault Trees, Binary Decision Diagrams

SUMMARY & CONCLUSIONS

Fault tree analysis is now commonly used to assess the adequacy, in reliability terms, of industrial systems. For complex systems an analysis may produce thousands of combinations of events which can cause system failure (minimal cut sets). The determination of these minimal cut sets can be a very time consuming process even on modern high speed digital computers. Also if the fault tree has many minimal cut sets calculating the exact top event probability will require extensive calculations. For many complex fault trees this requirement is beyond the capability of the available machines, thus approximation techniques need to be introduced resulting in loss of accuracy.

This paper describes the use of a Binary Decision Diagram for Fault Tree Analysis and some ways in which it can be efficiently implemented on a computer. The work to date shows a substantial improvement in computational effort for large, complex fault trees analysed with this method in comparison to the traditional approach. The Binary Decision Diagram method has the additional advantage that approximations are not required, exact calculations for the top event parameters can be performed.

1. *INTRODUCTION*

The fault tree diagram itself is an excellent way of deriving the failure logic for a system and representing it in a form which is ideal for communication to other managers/ designers/operators etc. The fault tree is discussed in detail in Andrews and Moss (Ref. 1). Since the method was first conceived in the early sixties, algorithms to derive the minimal cut sets have worked directly with the fault tree diagram itself using either bottom-up, Semanderes (Ref. 2), or top-down, Fussell and Vesely (Ref. 3), approaches. Computerised methods to conduct this analysis are now so well developed that further refinement is unlikely to result in vast reductions in computer time. Tackling this problem to improve computational efficiency has been the main concern over the years for many fault tree researchers, Bennetts (Ref. 4) and Bengiamin et al. (Ref. 5) have both addressed this problem. Usually by modifying the established, conventional approaches such as MOCUS (Ref. 3).

It is felt that substantial improvement in computer utilisation will only result from a completely new approach. Such an approach would involve specifying the logic equation in a form which is easier to manipulate than a fault tree. A recent paper by Rauzy (Ref. 6) has indicated that an alternative approach using a Binary Decision Diagram may provide a faster and more efficient means of analysing fault trees.

2. *NOTATION*

$P(Top)$	– Probability of Top Event of a fault tree.
C_i	– Minimal Cut Set.
$P_{RE}(Top)$	– Rare Event Approximation of Top Event Probability.
$P_{MCSUB}(Top)$	– Minimal Cut Set Upper Bound of Top Event Probability.
X_i	– Boolean Variable.
f(x)/f1/f2	– Boolean Functions.
ite	– If-Then-Else structure for Binary Decision Diagram.
<op>	– Boolean operation (· or +).
Fi	– Nodes/Vertices in a Binary Decision Diagram.
Q_{sys}	– Probability of occurrence of top event of fault tree.
W(0, t)	– Expected number of top event occurrences.
w_{sys}	– System Unconditional Failure Intensity.
$G_i(q)$	– Criticality Function for component i

3. *ABBREVIATIONS*

s.o.p - sum of products expression.
BDD - Binary Decision Diagram.

4. *FAULT TREE ANALYSIS*

The analysis of the fault tree is generally undertaken in two stages: qualitative analysis and quantitative analysis. Qualitative analysis involves obtaining the various combinations of events which cause system failure (minimal cut sets) and quantification then deals with calculating the probability or frequency that system failure will occur.

4.1 *Qualitative Analysis*

The conventional approach to obtain the minimal cut sets is to take the Boolean logic expression for the Top Event and transform it into a sum of products (s.o.p) form. One way of doing this is to use a Bottom-Up procedure such as that of Semanderes (Ref. 2). To obtain the s.o.p form for the Top Event of the fault tree, the inputs to the lowest gates are

0-7803-3112-5/96/$5.00 © 1996 IEEE

represented as logic equations. Once the lower gates have been expressed in this way higher gates are then treated similarly. The final s.o.p form should be in terms of basic events only.

If the fault tree contains repeated events then the resulting s.o.p will not be minimal and the minimal cut sets can not be directly obtained. If this is the case Boolean Reduction Rules must first be applied to the s.o.p form to obtain the minimal cut sets. The task of obtaining the minimal cut sets of a fault tree can become computationally intensive if the logic equations produce many cut sets, due to the number of comparisons that are needed to make the expression minimal. Also the expansion procedure can make extensive demands on memory space.

To overcome these problems various techniques have been employed to reduce the number of comparisons (Ref. 7). Some methods only produce the most important minimal cut sets. One of these techniques is referred to as culling, which means that cut sets of a certain order, say 4 and above, are ignored or deleted from the expression, Rasmuson and Marshall (Ref. 8) employ this technique in their paper. The justification for doing this is that cut sets of a high order tend to have a low probability of occurrence and therefore do not make a significant contribution to the Top Event probability. However the disadvantage of this is that when common cause failures are involved this method results in considerable inaccuracies. Probabilistic culling can also be applied, in this case a cut set whose probability of occurrence is below some threshold limit will again be ignored.

4.2 *Quantitative Analysis*

The conventional approach (see Henley and Kumamoto in Ref. 9) to obtain the exact probability of the Top Event is to use the formula:

$$P(Top) = \sum_{i=1}^{nc} P(C_i) - \sum_{i=2}^{nc}\sum_{j=1}^{i-1} P(C_i \cap C_j) + \ \ldots \\ + \ \ldots(-1)^{nc-1} P(C_1 \cap C_2 \cap \ \ldots \ \cap C_{nc}) \quad (1)$$

Where C_i, i=1,.....nc are the minimal cut sets of the Top Event, i.e. product terms.

Clearly if the fault tree has many minimal cut sets calculating P(Top) will require extensive calculations to evaluate each term in the expression, for many complex fault trees the requirement is beyond the capability of the available machines. To simplify the calculation the Rare Event Approximation, $P_{RE}(Top)$, can be used which is:

$$P_{RE}(Top) = \sum_{i=1}^{nc} P(C_i) \quad (2)$$

However a more accurate approximation is the Minimal Cut Set Upper Bound, $P_{MCSUB}(Top)$, which is:

$$P_{MCSUB}(Top) = 1 - \prod_{i=1}^{nc} (1 - P(C_i)) \quad (3)$$

5. *BINARY DECISION DIAGRAM METHOD*

The Binary Decision Diagram (BDD) method, developed by Rauzy (Ref. 6), first converts the fault tree to a binary decision diagram which encodes an **If-Then-Else** (ite) structure. The attractive thing about the BDD method is that the ite structure derives from Shannons' formula (Ref. 10), such that if f(**x**) is the Boolean Function for the top event of a fault tree then the Shannon formula can be written as;

$$X1.f1 + \overline{X1}.f2 \quad (4)$$

and the corresponding ite structure is ite(X1, f1, f2), for a detailed account of this procedure refer to Ref. 11 and Ref. 12. From this diagram both the qualitative and quantitative analysis can be achieved.

The size of the resulting BDD is determined by the ordering that has to be given to the basic events in the fault tree before the BDD is constructed. This ordering has further implications for the analysis. If the BDD is not in a minimal form, then the BDD must first undergo a minimising algorithm before the minimal cut sets can be obtained, this minimising technique is discussed in section 6. The quantitative analysis must be performed on the unminimised diagram. The reason being that the minimising procedure produces a new BDD which only encodes the minimal cut sets. However if the ordering of the basic events produces a minimal BDD then both the quantitative and qualitative analysis is straight forward. It is therefore beneficial to achieve an ordering which is optimal in terms of the resulting size of the BDD. The ordering of basic events to produce a minimal diagram is considered in (Ref. 11) and discussed in section 7.

To illustrate the method of obtaining the minimal cut sets and probability of occurrence of the top event using the BDD method refer to the example fault tree in figure 1.

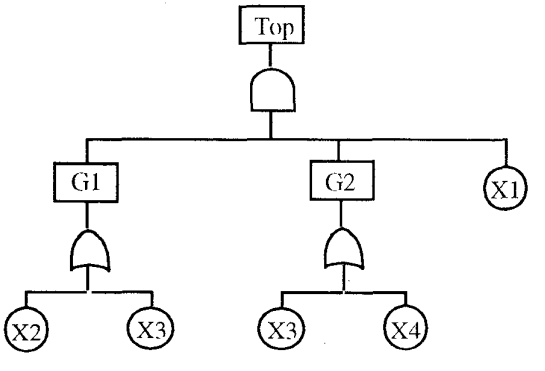

Figure 1. Example Fault Tree.

Assume an ordering for the basic events which is derived by considering those events at higher levels in the tree structure first:

X1<X2<X3<X4

To obtain the ite structures for each gate in the fault tree the following procedures are used:

(1) Taking X<Y;

Let J=ite(X, F1, F2) and H=ite(Y, G1, G2) then;

J<op>H=ite(X, F1<op>H, F2<op>H) (5)

(2) Taking X=Y;

i.e., J=ite(X, F1, F2) and H=ite(X, G1, G2) then;

J<op>H=ite(X, F1<op>G1, F2<op>G2) (6)

where <op> corresponds to the Boolean operation of the logic gates in the fault tree. For an AND gate <op> will be the dot or product symbol and for an OR gate <op> will be the addition symbol.

Also it is evident that;

1<op>H=1 if <op> is an OR gate
1<op>H=H if <op> is an AND gate
0<op>H=H if <op> is an OR gate
0<op>H=0 if <op> is an AND gate

Therefore the BDD calculations for the fault tree in figure 1 are the following:

G2 = ite(X3, 1, 0)+ite(X4, 1, 0)
= ite(X3, 1, ite(X4, 1, 0))
G1 = ite(X2, 1, 0)+ite(X3, 1, 0)
= ite(X2, 1, ite(X3, 1, 0))
Top = G1.G2.X1
= ite(X2, 1, ite(X3, 1, 0)).ite(X3, 1, ite(X4, 1, 0)).
ite(X1, 1, 0)
= ite(X2, ite(X3, 1, ite(X4, 1, 0)), ite(X3, 1, 0).
ite(X3, 1, ite(X4, 1, 0)).ite(X1, 1, 0)
= ite(X2, ite(X3, 1, ite(X4, 1, 0)), ite(X3, 1, 0))
.ite(X1, 1, 0)
Top =ite(X1, ite(X2, ite(X3, 1, ite(X4, 1, 0)), ite(X3, 1, 0)), 0)

This top event ite structure corresponds to the BDD shown in figure 2.

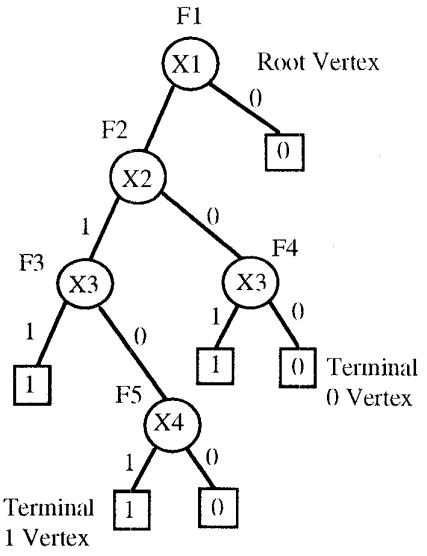

Figure 2. BDD for ite(X1, ite(X2, ite(X3, 1, ite(X4, 1, 0)), ite(X3, 1, 0)), 0).

To obtain the cut sets of the fault tree the paths through the BDD are traced from the top or root vertex to a terminal 1 vertex. Only the basic events that lie on a 1 branch (indicating the failure of that basic event) on the way to a terminal 1 vertex are included in a path. Therefore the paths through the BDD which correspond to the cut sets of the fault tree are:

(1) X1.X2.X3
(2) X1.X2.X4
(3) X1.X3

Clearly the resulting BDD for this ordering is not minimum as it produces one redundant cut set. The minimising procedure for the BDD which will produce the minimal cut sets directly is discussed in section 6.

5.1 *Top Event probability*

To obtain the probability of occurrence of the top event of the fault tree (Q_{sys}) the probability of the sum of the disjoint paths through the BDD are calculated. The disjoint paths through the BDD are found by simply including in a path the basic events that lie on a 0 branch and indicating these as $\overline{Xi}$, i.e., 'Not' Xi, meaning basic event i does not occur. Disjoint paths through the BDD are:

(1) X1.X2.X3
(2) X1.X2.$\overline{X3}$.X4
(3) X1.$\overline{X2}$.X3

Before continuing with the calculation of Q_{sys} the basic events in the fault tree need to be assigned probabilities, which for this example are given in table 1.

basic event i	q_i	λ_i	$w_i = \lambda(1-q)$
X1	0.01	1.0E-6	9.9E-7
X2	0.02	4.0E-6	3.92E-6
X3	0.03	2.0E-4	1.94E-4
X4	0.04	3.0E-5	2.88E-5

Table 1. Basic Event Data.

Where;
$\mathbf{q_i}$ – Unavailability of component i.
λ_i – Conditional failure intensity of component i.
$\mathbf{w_i}$ – Unconditional failure intensity of component i.

Since Q_{sys} can be obtained from the probability of the sum of the disjoint paths through the BDD then:

$$\begin{aligned} Q_{sys} &= P(X1.X2.X3 + X1.X2.\overline{X3}.X4 + X1.\overline{X2}.X3) \\ &= q_{X1}\cdot q_{X2}\cdot q_{X3} + q_{X1}\cdot q_{X2}\cdot(1-q_{X3})\cdot q_{X4} + \\ &\quad q_{X1}\cdot(1-q_{X2})\cdot q_{X3} \\ &= 0.01(0.02)(0.03) + 0.01(0.02)(1-0.03) \\ &\quad (0.04) + 0.01(1-0.02)(0.03) \\ Q_{sys} &= 3.0776E-4 \end{aligned}$$

The algorithm used by Rauzy for calculating the probability is given in Ref. 6.

5.2 *Unconditional System Failure Intensity*

For some systems it is the unreliability which is required for the top event i.e., the probability it will not work continuously over a given time period. An upper bound for this is the Expected number of top event occurrences W(0, t):

$$W(0,t) = \int_0^t w_{sys}\,dt \qquad (7)$$

w_{sys} is the system unconditional failure intensity:

$$w_{sys} = \sum_i G_i(q).w_i \qquad (8)$$

where $G_i(q)$ is the criticality function for each component.

The criticality function $G_i(q)$ is defined as the probability that the system is in a critical state with respect to component i and that the failure of component i will then cause the system to go from the working to the failed state, i.e., the probability that the system fails only if component i fails. Therefore:

$$G_i(q) = Q(1_i,q) - Q(0_i,q) \qquad (9)$$

Where;

$Q(1_i,q)$ - is the probability of system failure with $q_i = 1$.
$Q(0_i,q)$ - is the probability of system failure with $q_i = 0$.

Evaluating each of the two terms $Q(1_i,q)$ and $Q(0_i,q)$ for each component could be achieved by first substituting $q_i = 1$ and then $q_i = 0$, i.e., the probability that component i equals 1 and 0 respectively, and re-running the system failure probability calculations. This would require the equivalent of 2n evaluations of the top event probability to deduce all terms required in the expression for w_{sys} in eq (8).

Consider the variable Xi which occurs at two nodes in the BDD (Figure 3) then:

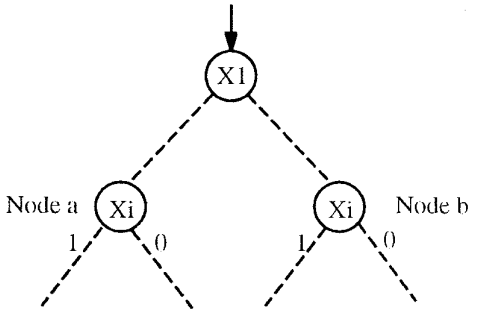

Figure 3. Considering variable Xi.

$$Q(1_i,q) = \sum_n (pr_{xi}(q).po^1_{xi}(q)) + Z(q) \qquad (10)$$

$$Q(0_i,q) = \sum_n (pr_{xi}(q).po^0_{xi}(q)) + Z(q) \qquad (11)$$

where:

$pr_{xi}(q)$ – is the probability of the path section from the root node to node xi.

$po^1_{xi}(q)$ – is the probability of the path section from node xi to the terminal 1 node after the 1 branch from node xi.

$po^0_{xi}(q)$ – is the probability of the path section from node xi to the terminal 1 node after the 0 branch from node xi.

Z(q) – is the probability of paths from the root node to the terminal 1 nodes which do not go through a node for variable xi.

n – All nodes for variable xi on the BDD.

Therefore:

$$G_i(q) = \sum_n pr_{xi}(q)\left[po^1_{xi}(q) - po^0_{xi}(q)\right] \qquad (12)$$

A more efficient way to calculate w_{sys} is to make one pass of the BDD to calculate $pr_{xi}(q)$, $po^1_{xi}(q)$ and $po^0_{xi}(q)$ for each node. With this information each $G_i(q)$ can be easily evaluated from eq (12) and w_{sys} formed.

The algorithm Probpost to calculate $po^1_{xi}(q)$ and $po^0_{xi}(q)$ is given in figure 4. The calculation of $pr_{xi}(q)$ can be achieved by the algorithm Probprev given in figure 5. The criticality function $G_i(q)$ for each basic event is calculated as shown in figure 6.

```
Probpost(F) ≡
    Do for all F, end vertices to Root Vertex
    F=ite(x, G, H)
    R←Probtable(x, prob(G), prob(H))
    Q1←p(x).prob(G)
    Q2←(1-p(x)).prob(H)
    insert - in - computation - table ({<prob, F, ->, Q1+Q2})
    return R
    return Q1 and Q2
    next F
```

Figure 4. Probpost Algorithm.

```
Probprev(F) ≡
     start at Root Vertex, F
     Probprev(F)=1
     Add Probprev(F) to Probtable, i.e.,
     Probtable(Probpost(F), Probprev(F))
     Do for all F, Root Vertex to end vertices
        F=ite(x, H1, H2)
            if H1=0 or 1 Goto [A]
            Proprev(H1)=p(x).Probprev(F)
            Add Probprev(H1) to Probtable

[A]  if H2=0 or 1 next F
     Probprev(H2)=(1-p(x)).Probprev(F)
     Add Probprev(H2) to Probtable
     next F.
```

Figure 5. Probprev Algorithm.

```
Set G(xi)=0 for all i
Do for all F
    if F=Probtable(x, q1, q2, q3)
    G(x)=G(x)+q3(q1-q2)
    insert - in Criticality table G(x)
next F.
```

Figure 6. Algorithm for Calculating the Criticality Function, G_{xi}.

Example

Applying these algorithms to the example BDD given in figure 2 illustrates the application of this method.
The ite table for the BDD in figure 2 is:

ITE

Node Label	Variable	1 branch pointer	0 branch pointer
F1	X1	F2	0
F2	X2	F3	F4
F3	X3	1	F5
F4	X3	1	0
F5	X4	1	0

Performing one pass of the BDD to evaluate $po^1_{xi}(q)$ and $po^0_{xi}(q)$ for each node using Probpost gives:

```
Probpost(F5)
    F5=ite(X4, 1, 0)
    R←Probtable(X4, 1, 0)
    Q1←p(X4)=0.04
    Q2=0
Probpost(F4)
    F4=ite(X3, 1, 0)
    R←Probtable(X3, 1, 0)
    Q1←p(X3)=0.03
    Q2=0
Probpost(F3)
    F3=ite(X3, 1, F5)
    R←Probtable(X3, 1, prob(F5))←(X3, 1, 0.04)
    Q1←p(X3)=0.03
    Q2←(1-p(X3))(0.04)=0.0388
Probpost(F2)
    F2=ite(X2, F3, F4)
    R←Probtable(X2,prob(F3), prob(F4))←(X2,0.0688, 0.03)
    Q1←p(X2)(0.0688)=1.376E-3
    Q2←(1-p(X2))(0.03)=0.0294
Probpost(F1)
    F1=ite(X1, F2, 0)
    R←Probtable(X1, prob(F2), 0)←(X1, 0.030776, 0)
    Q1←p(X1)(0.030776)=3.0776E-4
    Q2=0
```

In performing this one pass, the top event probability can be calculated by;

$$P(Top)=Q1+Q2 \tag{13}$$

for the top event node.

The values of Probpost 1 branch and Probpost 0 branch for each node are entered into the node probability table, PROBTABLE (see figure 7).

Next calculating the probability of the BDD path to each node is established using Probprev and entered into the 4th column of the PROBTABLE.

```
Probprev:

Probprev(F1)=1

F1=ite(X1, F2, 0)
    Probprev(F2)=P(X1).Probprev(F1)
                =0.01(1)=0.01
    H2=0
F2=ite(X2, F3, F4)
    Probprev(F3)=p(X2).Probprev(F2)
```

=0.02(0.01)=2.0E-4
Probprev(F4)=(1-p(X2)).Probprev(F2)
=(1-0.02)(0.01)=9.8E-3
F3=ite(X3, 1, F5)
H1=1
Probprev(F5)=(1-p(X3)).Probprev(F3)
=(1-0.03)(2.0E-4)=1.94E-4
F4=ite(X3, 1, 0)
H1=1
H2=0
F5=ite(X4, 1, 0)
H1=1
H2=0

Node Label	Variable	post '1'	post '0'	probprev
		PROBTABLE		
F1	X1	0.030776	0	1
F2	X2	0.0688	0.03	0.01
F3	X3	1	0.04	2.0E-4
F4	X3	1	0	9.8E-3
F5	X4	1	0	1.94E-4

Probtable(i, 1)=Basic event of node Fi
Probtable(i, 2)=Probability of post '1' branch
Probtable(i, 3)=Probability of post '0' branch
Probtable(i,4)=Probability of previous

Figure 7. PROBTABLE Array.

Calculation of the criticality function is then straight forward using the algorithm provided in figure 6.

Criticality Algorithm:

G(X1)=G(X2)=G(X3)=G(X4)=0

F1=Probtable(X1, 0.030776, 0, 1)
G(X1)=0+1(0.030776-0)
=0.030776
F2=Probtable(X2, 0.0688, 0.03, 0.01)
G(X2)=0+0.01(0.0688-0.03)
=3.88E-4
F3=Probtable(X3, 1, 0.04, 2.0E-4)
G(X3)=0+2.0E-4(1-0.04)
=1.92E-4
F4=Probtable(X3, 1, 0, 9.8E-3)
G(X3)=1.92E-4+9.8E-3(1-0)
=9.992E-3
F5=Probtable(X4, 1, 0, 1.94E-4)
G(X4)=1.94E-4(1-0)
=1.94E-4

Since we have calculated the criticality function for each component, w_{sys} can now be evaluated using the frequency data from table 1 using eq (8).

$$w_{sys} = G(X1)w_{X1} + G(X2)w_{X2} + G(X3)w_{X3} + G(X4)w_{X4}$$
$$= 0.030776(9.9E{-}7) + 3.88E{-}4(3.92E{-}6) + 9.992E{-}3(1.94E{-}4) + 1.94E{-}4(2.88E{-}5)$$
$$= 1.9760244E{-}6$$

Using eq (7) the expected number of top event occurrences in time, t, can be obtained.

6. *MINIMISING THE BDD*

In the example fault tree (figure 1) the resulting BDD (figure 2) was not minimum as it produced a redundant cut set. To obtain only minimal cut sets the BDD must first undergo a minimising procedure. From the unminimised BDD the minimising algorithm of Rauzy (Ref. 6) creates a new BDD that symbolises only the minimal cut sets of the fault tree. If F=ite(x, G, H) then let δ be a minimal solution of G which is not a minimal solution of H, then clearly the intersection of δ and x will be a minimal solution of F. Lastly, the set σ of all the minimal solutions of F, $sol_{\min}(F)$, will also include the minimal solutions of H so:

$$Sol_{\min}(F) = \{\sigma\}$$

where;

$$\sigma = \left[\{\delta\} \cap x\right]\left[Sol_{\min}(H)\right]$$

Rauzy (Ref. 6) has defined a 'without' operator which removes from $G_{\min}$ all the paths included in a path of H. Applying this algorithm to the BDD in figure 2 where each node is considered in turn:

F1=ite(X1, F2, 0) - Here there are no solutions on the 0 branch so the paths of F2 remain unchanged.

F2=ite(X2, F3, F4) - Here X3 is included in a path on both the 1 branch (F3) and the 0 branch (F4), therefore X3 is removed from the 1 branch by replacing the terminal 1 vertex with a 0. [Refer to figure 8]

F3=ite(X3, 0, F5) - F5 does not contain any paths that are included in the 1 branch as this is a terminal vertex.

F4=ite(X3, 1, 0) - The without operator does not apply as both 0 and 1 branches are terminal.

F5=ite(X4, 1, 0) - Same applies as F4.
The minimised BDD is drawn in figure 8.

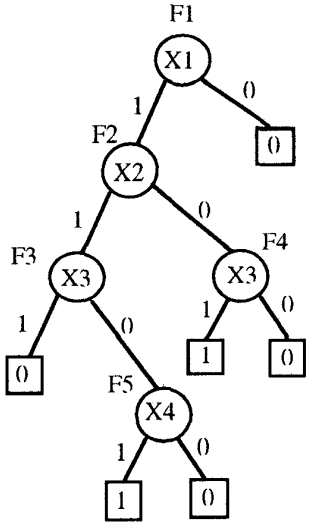

Figure 8. Minimised BDD.

Tracing the paths through the minimised BDD we obtain the minimal cut sets:

(1) X1.X2.X4
(2) X1.X3

7. *VARIABLE ORDERING SCHEME*

The ordering of basic events will determine the size of the resulting BDD. BDD's produced using a simple "top-down" ordering of the variables are frequently inefficient since they produce a large number of non-minimal cut sets. An alternative ordering scheme is presented here which focuses on those basic events which are repeated in the fault tree structure. It is the repeated events which cause the problem of non-minimal cut sets, and by considering these events first simplifies the resulting BDD structure and therefore makes it more optimal.
The alternative ordering scheme again considers the basic events in a top-down ordering (after the fault tree structure is contracted into an alternating sequence of AND and OR gates). However as each gate is considered the basic events which are inputs to the gate are taken in order of those which occur most frequently in the fault tree and placed in the ordering list. When gate input events are encountered which are already entered in the ordering list due to the occurrence at a higher level in the tree then they are ignored and the remaining input events are ordered.

Applying the new ordering here to the example fault tree (figure 1) with repeated event X3, we get the ordering X1<X3<X2<X4. The resulting BDD for this alternative ordering is minimum so the minimising technique is not needed, this is advantageous in terms of reduced computation time. Work carried out to date indicates that the new ordering appears to produce more optimal BDD's compared to other orderings. Bryant (Ref. 13) recognised the problem of computing an ordering that minimises the size of the BDD and for some trees it may not be possible to produce a minimal BDD whatever the ordering.

8. *MODULARISING*

Further improvements in terms of computational efficiency can be made for the more complex fault trees by modularising the fault tree before the analysis takes place. Khoda et al. (Ref. 14) define a module of a fault tree as having no inputs which appear elsewhere in the tree and no outputs to the rest of the tree except from its output event. For example consider the fault tree in figure 9. Modules which have the properties defined above are gates G2, G3 and Top.

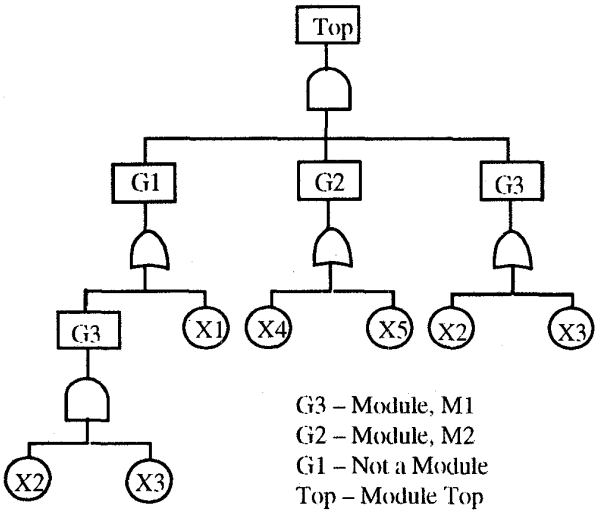

Figure 9. A Fault Tree which can be modularised.

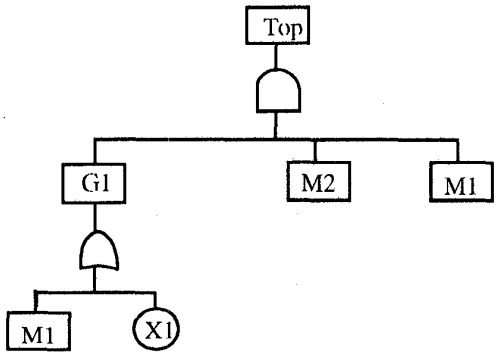

Figure 10. Modularised fault tree.

The modularised fault tree is shown in figure 10. By then using the BDD method to analyse this tree in terms of the modules and then each module in turn the results can be combined to provide an efficient means of analysing the whole fault tree.

9. *CONCLUSION*

Conventional top-down and bottom-up techniques can lead to many redundant cut sets and calculating exact top event probability can become impossible. To improve these analysis procedures the aim has been to represent the system failure logic in a mode which lends itself to the mathematical manipulation.

Representing the Boolean failure logic equation in the form of a BDD provides an alternative technique which gives significant savings in the computational efficiency and lends itself to manipulation. Also the BDD produces exact quantified results and top event parameters such as failure probability and the system unconditional failure intensity and the expected number of occurrences can be obtained with ease.

To simplify the analysis even further the fault tree may be modularised prior to the analysis. An alternative ordering of the basic event variables has also shown itself to significantly improve efficiency.

The trade off for the advantages described is the effort taken to convert the logic from the fault tree structure to the BDD form. However early work indicates that for large, complex trees this can produce a substantial reduction in computational effort.

10. *REFERENCES*

1 J.D. Andrews and T.R. Moss, *"Reliability and Risk Assessment,"* Longman Scientific and Technical, 1993.

2 S.N. Semanderes, "'ELRAFT,' A computer program for the efficient logic reduction analysis of fault trees," *IEEE Trans. Nuclear Science,* vol NS-18, 1971 Feb, pp 481-487.

3 J.B. Fussell, W.E. Vesely, "A new methodology for obtaining cut sets for fault trees," *Trans. Am. Nucl. Soc.,* vol 15, 1972 Jun, pp 262-263.

4 R.G. Bennetts, "On the analysis of fault trees," *IEEE Trans. Reliability,* vol R-24, No. 3, 1975 Aug, pp 175-185.

5 N.N. Bengiamin, B.A. Bowen, K.F. Schenk, "An efficient algorithm for reducing the complexity of computation in fault tree analysis," *IEEE Trans. Nuclear Science,* vol NS-23, No. 5, 1976 Oct, pp 1442-1446.

6 A. Rauzy, "New algorithms for fault tree analysis," *Reliability Engineering and System Safety,* vol 40, 1993, pp203-211.

7 N. Liminios, R. Ziani, "An algorithm for reducing the minimal cut sets in fault tree analysis," *IEEE Trans. Reliability,* vol R-35, No. 5, 1986 Dec, pp 559-561.

8 D.M. Rasmuson, N.H. Marshall, "FATRAM-A core efficient cut-set algorithm," *IEEE Trans. Reliability,* vol R-27, No. 4, 1978 Oct, pp 250-253.

9 E.J. Henley and H.Kumamoto, "*Reliability Engineering and Risk Assessment*," Englewood Cliffs, 1981.

10 W. G. Schneeweiss, "*Boolean Functions with Engineering Applications and Computer Programs*," Springer-Verlag, 1989.

11 R.M. Sinnamon and J.D. Andrews, "New Approaches to Evaluating Fault Trees," *Proceedings of Esrel'95 Conference,* June, 1995 pp241-254.

12 A. Rauzy et al., "Computation of prime implicants of a fault tree within Aralia," *Proceedings of Esrel'95 Conference,* June, 1995 pp190-202.

13 R. E. Bryant, "Graph-Based algorithms for Boolean function manipulation," *IEEE Trans. Computers,* vol C-35, No. 8, 1986 Aug, pp677-691.

14 T. Khoda, E. J. Henley and K. Inoue, "Finding Modules in Fault Trees," *IEEE Trans. Reliability,* vol 38, No. 2, 1989 Jun, pp165-176.

BIOGRAPHIES

Roslyn M. Sinnamon, BSc
Department of Mathematical Sciences
Loughborough University of Technology
Loughborough, Leicestershire LE11 3TU, UK.
Internet (e-mail) : R.M.Sinnamon1@lut.ac.uk

Ros Sinnamon is a second year PhD student working with Dr John Andrews on Reliability Theory, mainly Fault Tree Analysis. She also tutors in Reliability, Statistics and Mathematics for Engineers at Loughborough University. Her BSc Joint Honours is in Mathematics, Physical Education and Sports Science and was gained at Loughborough. She is a member of the Safety and Reliability Society.

John D. Andrews, PhD, BSc
Department of Mathematical Sciences
Loughborough University of Technology
Loughborough, Leicestershire, LE11, 3TU, UK
Internet (e-mail): J.D.Andrews@lut.ac.uk

Dr Andrews currently lectures in Risk and Safety Assessment Techniques in the Department of Mathematical Sciences at Loughborough University of Technology. Prior to this appointment he was a Senior Lecturer in the Department of Mechanical and Production Engineering at Birmingham Polytechnic and has also had two periods of employment as a Senior Scientist Engineer in the Research and Development Division at British Gas.

His industrial work has involved research into methods of assessing the safety and risk of potentially hazardous industrial activities. This research is now continuing at Loughborough University. Dr Andrews is currently a member of committees of the Institution of Mechanical Engineers and the Safety and Reliability Society which focus on Risk, Safety and Reliability issues.