ELSEVIER

PII: S0951-8320(96)00036-1

Reliability Engineering and System Safety **58** (1997) 89–96
© 1997 Elsevier Science Limited
All rights reserved. Printed in Northern Ireland
0951-8320/97/$17.00

New approaches to evaluating fault trees

R. M. Sinnamon & J. D. Andrews

Department of Mathematical Sciences, Loughborough University of Technology, Loughborough, Leicestershire LE11 3TU, UK

(Received 12 January 1996; revised 5 March 1996; accepted 25 March 1996)

Fault Tree Analysis is now a widely accepted technique to assess the probability and frequency of system failure in many industries. For complex systems an analysis may produce hundreds of thousands of combinations of events which can cause system failure (minimal cut sets). The determination of these cut sets can be a very time consuming process even on modern high speed digital computers. Computerised methods, such as bottom-up or top-down approaches, to conduct this analysis are now so well developed that further refinement is unlikely to result in vast reductions in computer time. It is felt that substantial improvement in computer utilisation will only result from a completely new approach. 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. In particular, attention is given to the production of a minimum form of the Binary Decision Diagram by considering the ordering that has to be given to the basic events of the fault tree. © 1997 Elsevier Science Limited.

1 INTRODUCTION

Fault tree analysis is now commonly used to assess the adequacy, in reliability terms, of industrial systems. This is particularly true of safety systems such as those installed on nuclear power plants or off-shore oil platforms. The technique was first developed by H. A Watson (1961–62) during a study of the launch control system of the Minuteman, Intercontinental ballistic missile and is discussed in detail in Andrews & Moss.[1]

The fault tree provides a diagrammatic description of the way in which a system can fail in a specific mode. The importance of the fault tree for safety system analysis is that it yields a complete description of the various causes of the system failure. Hence, the engineers can identify and rectify any problem areas in the design.

Unfortunately, the analysis of the fault tree, to find the causes of system failure, has a few drawbacks. If, because of the complexity of the system under study, the fault tree is large, then finding the causes of failure, termed minimal cut sets, can require an extensive computer processing capability. Tackling this problem to improve computational efficiency has been the main concern over the years for many fault tree researchers. Semanders,[2] Fussell & Vesely,[3] Bennetts[4] and Benjiamin *et al.*[5] have all addressed this problem, usually by modifying the established, conventional approaches such as MOCUS.[3]

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. This paper is concerned with qualitative analysis. The use of Binary Decision Diagrams for the purpose of quantitative analysis has also been dealt with.[13]

The majority of the methods developed to perform fault tree analysis work directly with the system fault tree structure. Whilst this type of logic diagram provides a very good representation of the system failure logic it is not necessarily the most efficient way to manipulate the resulting Boolean function. Recent papers[6,11,12,14] have indicated that alternative approaches such as Binary Decision diagrams may provide a faster means of analysing fault trees. This alternative approach is the subject of this paper.

2 THE FAULT TREE

To construct the fault tree, the system failure mode is broken down or developed into subsystem failures, which are in turn further developed into lower resolution events or failures. This process is continued

until no further development can take place and the limit of resolution is encountered.

Those events whose causes have been further developed are termed 'intermediate events' and events terminating branches, 'basic events'. The Top Event of the fault tree is the system failure which is also an intermediate event. Events in the fault tree are combined using logic gates, in this paper only AND gates and OR gates are considered (restricting the failure diagrams to coherent structures). An example fault tree is given in Fig. 1.

In the fault tree in Fig. 1 intermediate events are represented by rectangles, which are the gates Top and G1, and terminating events or basic events are represented by circles. Boolean variables X_i are then used to represent the occurrence or non-occurrence of the basic events, i.e., $X_i = 1$ means basic event i has occurred and $X_i = 0$ means basic event i has not occurred.

3 QUALITATIVE ANALYSIS

A cut set of a fault tree is a list of basic events or component failures such that, when they all occur, the Top Event also occurs, i.e., system failure. The set of cut sets consists of all the modes in which a system can fail.

A minimal cut set is the smallest combination of components whose failure leads to system failure, i.e., removal of one of the component failures means that the system no longer fails.

It is deriving the complete list of minimal cut sets which is the main concern of the qualitative analysis. This list can be obtained directly from the logic expression for the Top Event.

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 Semanders.[2] Consider again the fault tree in Fig. 1, to obtain the s.o.p form for the Top Event, the inputs to the lowest gates are represented as logic variables. The dot or product is used to represent AND whilst the sum is used to represent OR. Once the outputs of lower gates have been expressed in this way higher gates are then treated similarly.

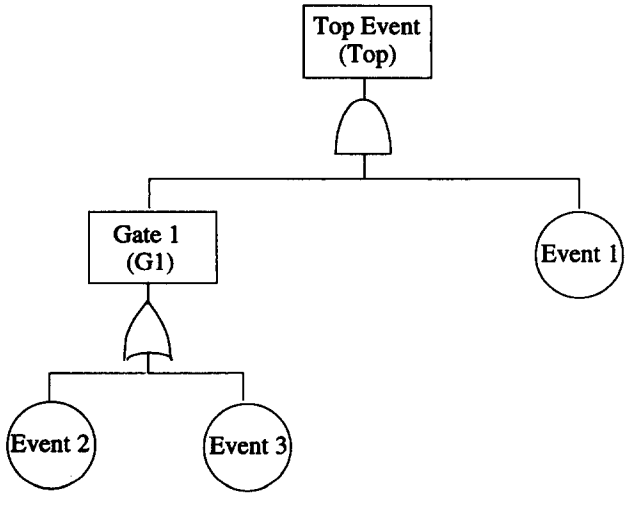

Fig. 1. Example fault tree.

$$G1 = X2 + X3$$

$$\text{Top} = G1.X1.$$

Then, substituting for $G1$ into Top gives the s.o.p form in terms of basic events.

$$\begin{aligned} \text{Top} &= (X2 + X3).X1 \\ \text{Top} &= X2.X1 + X3.X1. \end{aligned} \tag{1}$$

Therefore, the s.o.p form for the Top gate, Top, is $X2.X1 + X3.X1$.

The products in this expression then correspond to the cut sets of the fault tree, so in this case the fault tree has two cut sets, i.e., $\{X1, X2\}$ and $\{X1, X3\}$, both of order 2 (as there are 2 members in each set).

Next, to obtain the minimal cut sets from the Top Event expression, Boolean reduction rules need to be applied, which are:

(i) $A.A = A$ (removes repeated events within cut sets)
(ii) $A + A = A$ (removes repeated cut sets)
(iii) $A + A.B = A$ ($A.B$ is redundant) (removes non-minimal cut sets).

Rules (i) and (ii) are both known as the idempotence rules and (iii) as the absorption rule.

As none of these rules can be applied to eqn (1), none of the cut sets in this expression are redundant. So, the expression is minimal and therefore the two cut sets are minimal cut sets.

However, consider the non-minimal expression for the Top Event, T, of a fault tree:

$$T = X1 + X1.X2.X3 + X2.X3.X3 + X3.X4 + X3.X4.X5.$$

Applying the above rules the following minimal expression is obtained:

$$T = X1 + X2.X3 + X3.X4.$$

As stated previously, the task of obtaining the minimal cut sets of a fault tree can become computationally intensive if the logic equations produce many cut sets. If a large s.o.p expression is obtained then applying the reduction rules can take a long time on a computer due to the number of comparisons that are needed to make the expression

minimal. Also, storing all the logical expressions for each gate in the tree can make extensive demands on memory space.

To overcome this problem various techniques have been employed to reduce the number of comparisons.[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 & Marshall[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.

4 NEW APPROACH

A more recent method for minimal cut set generation called the Binary Decision Diagram (BDD) method[9,10,6] has been developed with an aim to reduce computation time and memory requirements. With this method the fault tree is first transformed to a BDD from which the minimal cut sets can be directly obtained.

A BDD is a directed acyclic graph. All paths through the BDD terminate in one of two states, either a 1 state, which corresponds to system failure, or a 0 state which corresponds to system success. All the paths terminating in a 1 state give the cut sets of the fault tree. A BDD is composed of terminal and non-terminal vertices, which are connected by branches. Terminal vertices have the value 0 or 1 and non-terminal vertices correspond to the basic events of the fault tree. Each vertex has a 0 branch which represents the basic event non-occurrence (works) and a 1 branch which represents basic event occurrence (fails). An attractive feature of the BDD technique is the use of subgraph sharing,[14] which increases computational efficiency. As an example BDD, consider Fig. 2.

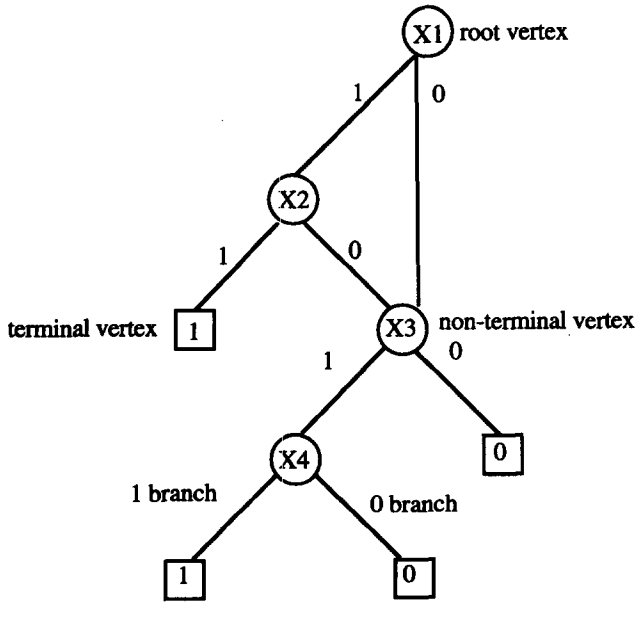

Fig. 2. Example BDD.

All the left hand branches leaving each vertex are the 1 branches and all the right hand branches are the 0 branches.

Every path starts from the root vertex, and proceeds down through the diagram to the terminal vertices. Only the vertices that lie on a 1 branch on the way to a terminal 1 vertex are included in a path which represents the cut sets. For example, the paths, or cut sets, of Fig. 2 are:

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

Each vertex in a BDD has an if-then-else (ite) structure. To illustrate this consider the if-then-else structure -ite($X1, f1, f2$), which means if $X1$ fails then consider function $f1$ else consider function $f2$. Also $f1$ lies on the 1 branch of $X1$ and $f2$ on the 0 branch. The BDD for this is the one given in Fig. 3.

Constructing the BDD from a fault tree initially requires the basic events in the fault tree to be given an ordering, such as $X1 < X2$ (normally a Top-Down ordering is used, i.e., the basic events which are placed higher up the tree are listed first and are regarded as being 'less than' those further down the tree). It also involves using ite structures and the following procedures:

(1) Taking $X < Y$;

Let $J = \text{ite}(X, F1, F2)$ and $H = \text{ite}(Y, G1, G2)$ then;

$$J < \text{op} > H = \text{ite}(X, F1 < \text{op} > H, F2 < \text{op} > H).$$

(2) Taking $X = Y$;

i.e., $J = \text{ite}(X, F1, F2)$ and $H = \text{ite}(X, G1, G2)$ then;

$$J < \text{op} > H = \text{ite}(X, F1 < \text{op} > G1, F2 < \text{op} > G2)$$

where <op> corresponds to the Boolean operation of the logic gates in the fault tree. Also, it is evident that;

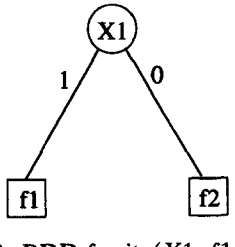

Fig. 3. BDD for ite($X1, f1, f2$).

$1 < \text{op} > H = 1$ if <op> is an OR gate
$1 < \text{op} > H = H$ if <op> is an AND gate
$0 < \text{op} > H = H$ if <op> is an OR gate
$0 < \text{op} > H = 0$ if <op> is an AND gate.

So, giving an ordering of $X1 < X2 < X3$ for the fault tree in Fig. 1, the ite structure for the Top Event is obtained as follows:

Firstly all basic events X_i, $i = 1,.....,n$, in the fault tree have the ite structure ite(X_i, 1, 0), i.e., X_i can either fail or work.

Working from the bottom to the top of the tree shown in Fig. 1 and applying the above procedures:

$$\begin{aligned} G1 &= \text{ite}(X2,1,0) + \text{ite}(X3,1,0) \\ &= \text{ite}(X2,1 + \text{ite}(X3,1,0),0 + \text{ite}(X3,1,0)) \\ &= \text{ite}(X2,1,\text{ite}(X3,1,0)) \\ \text{Top} &= \text{ite}(X2,1,\text{ite}(X3,1,0)).\text{ite}(X1,1,0) \\ &= \text{ite}(X1,1.\text{ite}(X2,1,\text{ite}(X3,1,0)),0 \\ &\quad .\text{ite}(X2,1,\text{ite}(X3,1,0))) \\ \text{Top} &= \text{ite}(X1,\text{ite}(X2,1,\text{ite}(X3,1,0)),0). \end{aligned}$$

To construct the BDD, each ite structure in Top is successively broken down into its left and right branches, resulting in the BDD in Fig. 4.

Next, the paths through the BDD are obtained, these being:

(i) $X1.X2$
(ii) $X1.X3$

which correspond to the minimal cut sets obtained using the conventional Bottom-Up procedure.

5 SPECIAL ORDERING

As discussed in Section 4, the ite structure can only be obtained once the basic event inputs have been given an ordering. Usually a 'Top-down' ordering procedure is employed where the basic events higher up the tree are 'less than' those lower down the tree.

This ordering of basic events is very important as it determines the size of the resulting BDD[9,10,6] and hence the number of cut sets. To illustrate this, consider the fault tree in Fig. 5.

The BDD for this fault tree with the conventional ordering $X1 < X2 < X3 < X4$ is given in Fig. 6 and, as a comparison, the reverse ordering $X4 < X3 < X2 < X1$, for the same fault tree, gives the BDD in Fig. 7.

Clearly, the first ordering creates a much smaller BDD, it has 4 nodes compared to the 7 nodes of Fig. 7 (here node $X1$ is shared by both $X3$ nodes). Fig. 7 gives 3 non-minimal cut sets whereas Fig. 6 creates only minimal cut sets.

Although the top down ordering produces a minimal BDD for this example which yields only minimal cut sets, this is not always the case. Obviously, if a redundant BDD is obtained then it is necessary to perform some kind of minimising procedure to obtain only the minimal cut sets. This results in greater computation time. Therefore it would be more beneficial to produce a minimal or at least a 'near' minimal BDD from the fault tree thus reducing the computation time spent minimising the resulting cut sets.

A study of 15 example fault trees indicates that using more restrictive event ordering produces a minimal BDD whereas the conventional ordering results in a redundant BDD. This restrictive ordering shall be called special ordering, and it is employed when the fault tree contains repeated events. A summary of the 15 example fault trees is given in Table 1 below.

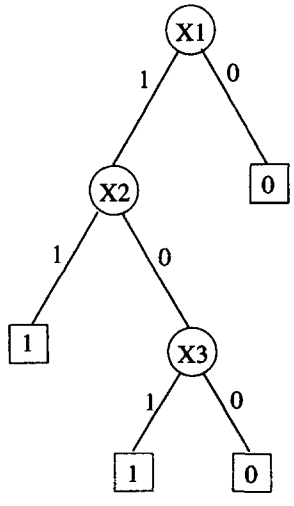

Fig. 4. BDD for example fault tree.

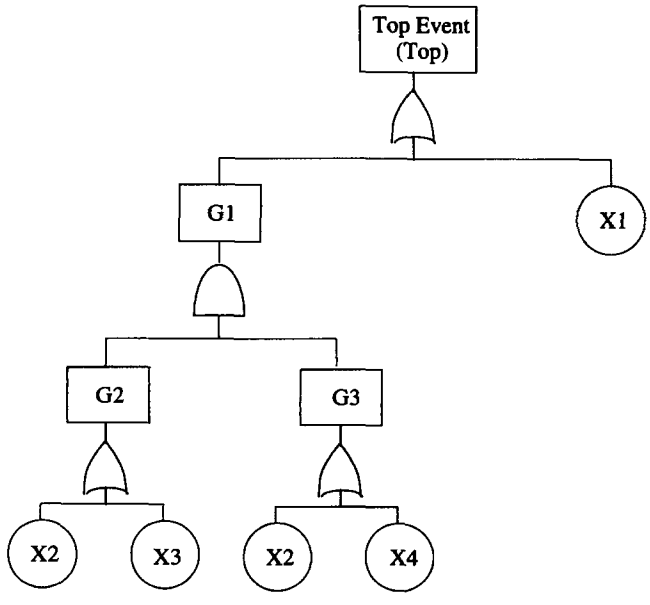

Fig. 5. Example fault tree with repeated event $X2$.

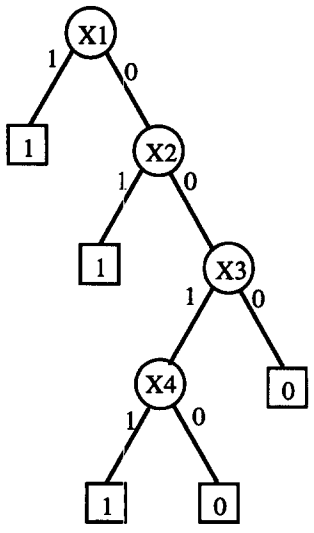

Fig. 6. BDD for ordering $X1 < X2 < X3 < X4$.

Table 1. Summary of 15 example fault trees

Fault tree	Gates	Basic events	Repeated events	Cut sets using bottom-up	Minimal cut sets from BDD
1	6	8	1	5	2
2	3	7	1	3	1
3	3	7	2	5	3
4	4	8	2	4	2
5	3	4	1	2	2
6	5	6	1	4	3
7	4	8	2	4	3
8	6	10	2	14	6
9	5	9	1	19	15
10	4	5	1	4	2
11	3	4	1	4	2
12	6	9	2	6	3
13	3	4	1	2	1
*14	4	7	1	3	3
*15	9	17	3	17	8

Relatively small fault trees were chosen to emphasise that even for small fault trees savings can be made using the special ordering. Trees 14 and 15 did not produce the absolute minimal BDD, however, from the investigation on these trees the special ordering appears to produce the nearest minimal BDD when compared to any other ordering.

The special ordering technique considers the gate events in a top down ordering similar to the conventional approach. However, at each gate the basic event inputs are listed with the repeated events first (i.e., they are 'less than' the other inputs to that gate). If the gate has more than one repeated event as an input then the most repeated event is placed first, if they occur the same number of times then the events are taken in gate list order to break the tie. Also, if an event has been ordered due to its occurrence higher up the tree then it is ignored for the ordering as an input to the gate.

To illustrate the advantages of special ordering, consider the simple fault tree in Fig. 8.

Conventional ordering would give $X1 < X3 < X2$. The ite structure calculations are given below:

$$G3 = X3 + X2$$
$$= \text{ite}(X3,1,0) + \text{ite}(X2,1,0)$$
$$= \text{ite}(X3,1,\text{ite}(X2,1,0))$$
$$G2 = X1 + X3$$
$$= \text{ite}(X1,1,0) + \text{ite}(X3,1,0)$$
$$= \text{ite}(X1,1,\text{ite}(X3,1,0))$$
$$\text{Top} = G2.G3$$
$$= \text{ite}(X1,1,\text{ite}(X3,1,0)).\text{ite}(X3,1,\text{ite}(X2,1,0))$$
$$= \text{ite}(X1,\text{ite}(X3,1,\text{ite}(X2,1,0)),\text{ite}(X3,1,0)$$
$$.\text{ite}(X3,1,\text{ite}(X2,1,0)))$$

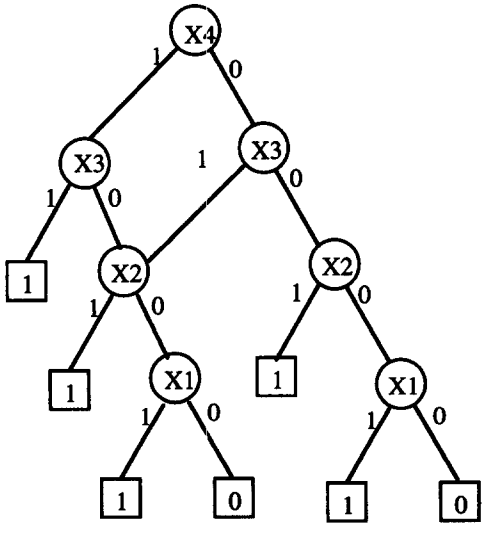

Fig. 7. BDD for ordering $X4 < X3 < X2 < X1$.

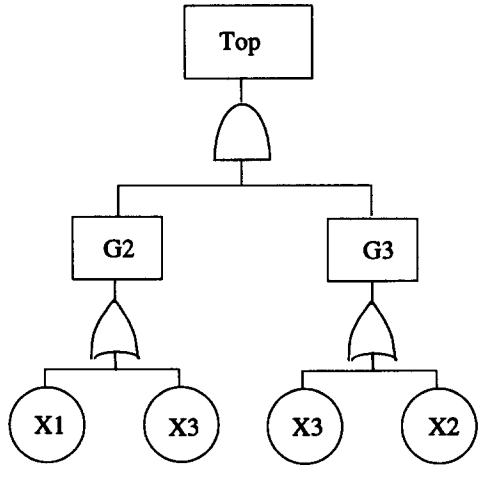

Fig. 8. Example fault tree with repeated events $X3$.

Top = ite($X1$,ite($X3$,1,ite($X2$,1,0)),ite($X3$,1,0)).

The BDD for Top is given in Fig. 9.
The paths through the BDD in Fig. 9 are:

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

Therefore, cut set $\{X1, X3\}$ is redundant which leaves the minimal cut sets:

(1) $\{X1, X2\}$
(2) $\{X3\}$.

Now, using the special ordering, one obtains $X3 < X1 < X2$. Note that $G2$ has the repeated event input $X3$ so it is placed before $X1$.
The ite structure for this ordering is calculated below:

$$G3 = X3 + X2$$
$$= \text{ite}(X3,1,0) + \text{ite}(X2,1,0)$$
$$= \text{ite}(X3,1,\text{ite}(X2,1,0))$$

$$G2 = X1 + X3$$
$$= \text{ite}(X1,1,0) + \text{ite}(X3,1,0)$$
$$= \text{ite}(X3,1,\text{ite}(X1,1,0))$$

$$\text{Top} = G2.G3$$
$$= \text{ite}(X3,1,\text{ite}(X1,1,0)).\text{ite}(X3,1,\text{ite}(X2,1,0))$$
$$= \text{ite}(X3,1,\text{ite}(X1,1,0).\text{ite}(X2,1,0))$$
$$\text{Top} = \text{ite}(X3,1,\text{ite}(X1,\text{ite}(X2,1,0),0)).$$

The BDD for this special ordering is given in Fig. 10.
The paths through the BDD in Fig. 10 are:

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

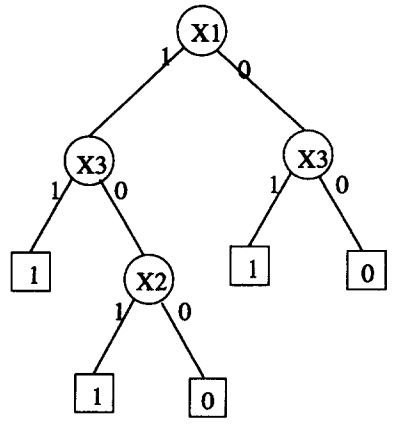

Fig. 9. BDD for fault tree in Fig.8.

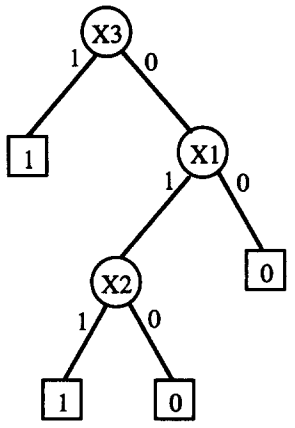

Fig. 10. BDD for 'special' ordering $X3 < X1 < X2$.

Therefore, the special ordering gives only the minimal cut sets $\{X3\}$ and $\{X1, X2\}$.

As the trees increase in size the special ordering becomes more beneficial in terms of eliminating redundant cut sets. This is the case for the slightly larger tree in Fig. 11.

The BDD for the ordering $X1 < X2 < X3 < X4 < X5 < X6 < X7 < X8$ produces 18 cut sets of which 15 are minimal. However, using the special ordering $X2 < X1 < X3 < X4 < X5 < X6 < X7 < X8$ directly gives the 15 minimal cut sets.

It has been found that, in larger trees with many repeated events, the special ordering may not give an absolute minimal BDD, however, from the research to date it appears to give the nearest minimal BDD when compared to other orderings. Thus, for some trees it may not be possible to produce a minimal BDD whatever the ordering, although using the special ordering can reduce computation time by creating a near minimal BDD. Bryant[10] recognised the problem

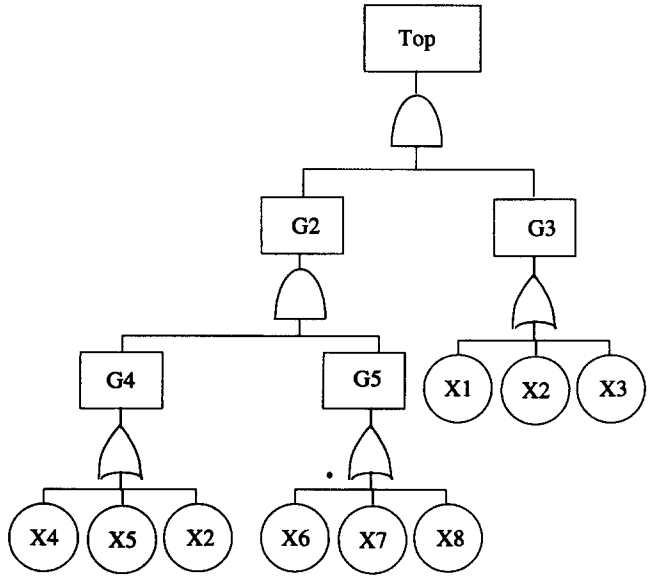

Fig. 11. Slightly larger fault tree with repeated event $X2$.

of computing an ordering that minimises the size of the BDD.

The advantage of using this simple restriction on ordering becomes more obvious when comparing it to a standard bottom-up approach. A bottom-up method is used to obtain the minimal cut sets of the fault tree in Fig. 12.

$$G5 = X7 + X3 + X6$$

$$G4 = G5.X4$$
$$= (X7 + X3 + X6).X4$$
$$= X7.X4 + X3.X4 + X6.X4$$

$$G1 = G4 + X5 + X2$$
$$= X7.X4 + X3.X4 + X6.X4 + X5 + X2$$

$$G3 = X3 + X8 + X2$$

$$G2 = G3.X1$$
$$= (X3 + X8 + X2).X1$$
$$= X3.X1 + X8.X1 + X2.X1$$

$$\text{Top} = G2.G1$$
$$= (X3.X1 + X8.X1 + X2.X1)$$
$$.(X7.X4 + X3.X4 + X6.X4 + X5 + X2)$$

$$\text{Top} = X3.X1.X7.X4 + X3.X1.X3.X4 + X3.X1.X6.X4$$
$$+ X3.X1.X5 + X3.X1.X2 + X8X1.X7.X4$$
$$+ X8.X1.X3.X4 + X8.X1.X6.X4 + X8.X1.X5$$
$$+ X8.X1.X2 + X2.X1.X7.X4 + X2.X1.X3.X4$$
$$+ X2.X1.X6.X4 + X2.X1.X5 + X2.X1.X2.$$

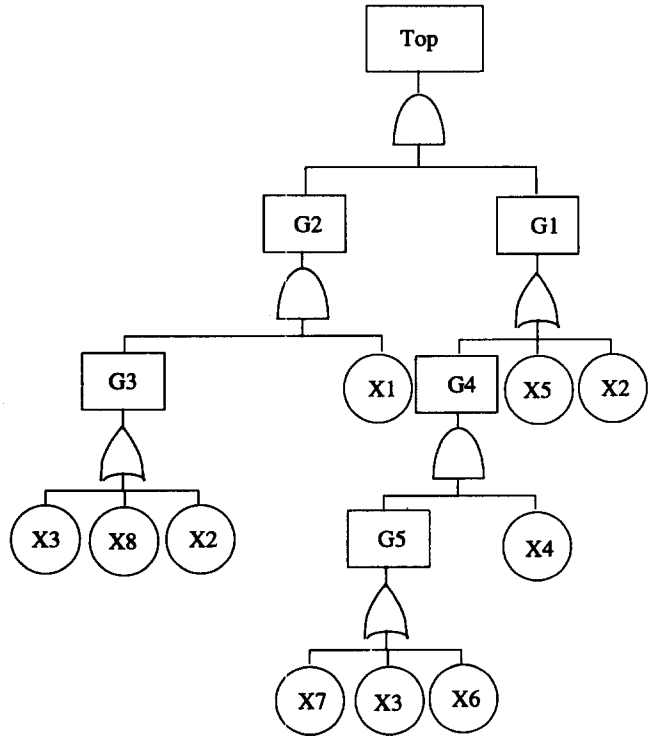

Fig. 12. Fault tree with repeated events $X2$ and $X3$.

Eliminating redundancies from the Top event expression leaves the following 6 minimal cut sets:

(1) $\{X1, X3, X4\}$
(2) $\{X1, X3, X5\}$
(3) $\{X1, X8, X4, X7\}$
(4) $\{X1, X4, X6, X8\}$
(5) $\{X1, X8, X5\}$
(6) $\{X1, X2\}$.

Comparing this method with the BDD method, the ordering $X1 < X2 < X5 < X3 < X8 < X4 < X7 < X6$ on variables directly produces the 6 minimal cut sets. Therefore in this case time is not wasted in computing and eliminating the 7 redundant cut sets that were obtained in the bottom-up method.

6 CONCLUSION

To make significant savings in the computational efficiency of the analysis of fault trees means moving away from the conventional techniques, that utilise the fault tree structure (which are now so fully developed) and representing the system failure logic in a mode which lends itself to manipulation.

The BDD form for the Boolean equation seems to provide an alternative technique to efficiently analyze fault trees. Also by considering the ordering of basic events the BDD can produce minimal cut sets without having to eliminate redundant cut sets as is often the case with conventional techniques.

The trade off for this 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.

REFERENCES

1. Andrews, J.D. & Moss, T.R., *Reliability and Risk Assessment,* Longman Scientific and Technical, Avon, UK, 1993.
2. Semanderes, S.N. 'ELRAFT,' a computer program for the efficient logic reduction analysis of fault trees,. *IEEE Trans. Nuclear Science*, **NS-18** (1971) 481–487.
3. Fussell, J.B. & Vesley, W.E., A new methodology for obtaining cut sets for fault trees. *Trans. Am. Nucl. Soc.*, **15** (1972) 262–263.
4. Bennetts, R.G., On the analysis of fault trees. *IEEE Trans. Reliab.*, **R-24** (1975) 175–185.
5. Bengiamin, N.N., Bowen, B.A. & Schenk, K.F., An efficient algorithm for reducing the complexity of computation in fault tree analysis. *IEEE Trans. Nucl. Sci.*, **NS-23** (1976) 1442–1446.
6. Rauzy, A., New algorithms for fault tree analysis. *Reliab. Engng System Safety*, **40** (1993) 203–211.
7. Liminios, N. & Ziani, R., An algorithm for reducing the minimal cut sets in fault tree analysis. *IEEE Trans. Reliab.*, **R-35** (1986) 559–561.
8. Rasmuson, D.M. & Marshall, N.H., FATRAM—a core

efficient cut-set algorithm. *IEEE Trans. Reliab.*, **R-27** (1978) 250–253.
9. Akers, S.B., Binary decision diagrams. *IEEE Trans. Computers*, **C-27** (1978) 509–516.
10. Bryant, R.E., Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Computers*, **C-35** (1986) 677–691.
11. Schneeweiss, W.G., *Boolean Functions with Engineering Applications and Computer Programs,* Springer-Verlag, Berlin, 1989.
12. Coudert, O. & Madre, J., MetaPrime: an interative fault-tree analyser. *IEEE Trans. Reliab.*, **R-43** (1994) 121–127.
13. Sinnamon, R.M. & Andrews, J.D., Fault tree analysis and binary decision diagrams. In *Proceedings of RAMS'96 Conference*, Las Vegas, Nevada, 22–25 January 1996, pp. 215–222.
14. Bouissou, M., An ordering heuristic for building binary decision diagrams from fault-trees. In *Proceedings of RAMS'96 Conference,* Las Vegas, Nevada, 22–25 January 1996, pp. 208–215.