


This item was submitted to Loughborough's Institutional Repository by the author and is made available under the following Creative Commons Licence conditions.




CC creative commons
COMMONS DEED


Attribution-NonCommercial-NoDerivs 2.5


You are free:

- to copy, distribute, display, and perform the work

Under the following conditions:

 **Attribution.** You must attribute the work in the manner specified by the author or licensor.


 **Noncommercial.** You may not use this work for commercial purposes.

 **No Derivative Works.** You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the [Legal Code \(the full license\)](#).

[Disclaimer](#) 

For the full text of this licence, please go to:
<http://creativecommons.org/licenses/by-nc-nd/2.5/>

Fault Tree Conversion to Binary Decision Diagrams

J.D. Andrews; Department of Aeronautical and Automotive Engineering
Loughborough University; Loughborough, Leicestershire, England

R. Remenyte; Department of Aeronautical and Automotive Engineering
Loughborough University; Loughborough, Leicestershire, England

Keywords: Fault Tree Analysis, Binary Decision Diagrams

Abstract

Fault Tree Analysis is a commonly used technique to predict the causes of a specific system failure mode and to then determine the likelihood of this event. Over recent years the Binary Decision Diagram (BDD) method has been developed for the solution of the fault tree. It can be shown that this approach has advantages in terms of both accuracy and efficiency over the conventional method of analysis formulated in the 1970's. The BDD expresses the failure logic in a disjoint form which gives it an advantage from the computational viewpoint. Fault Trees, however, remain the better way to represent the system failure causality. Therefore the usual way of taking advantage of the BDD structure is to construct a fault tree and then convert this to a BDD. It is on the fault tree conversion process that this paper will focus.

In order to construct a BDD the variables which represent the occurrence of the basic events in the fault tree have to be placed in an ordering. Depending on the ordering selected an efficient representation of the failure logic can be obtained or if a poor ordering is selected a less efficient analysis will result. Once the ordering is established one approach is to utilise a set of rules developed by Rauzy which are repeatedly applied to generate the BDD. An alternative approach can be used whereby BDD constructs for each of the gate types are first formed and then joined together as specified by the gates in the fault tree. Some comments on the effectiveness of these approaches will be provided.

Introduction

The binary decision diagram (BDD) method "(ref. 1)" has been developed as an alternative to conventional methods for performing qualitative and quantitative analysis of fault trees. This method appears to be a more efficient means of analysing a system and does not need to take advantage of the approximations used in the traditional approach of kinetic tree theory "(ref. 2)".

Rather than analysing the fault tree directly the BDD method first converts the fault tree to a binary decision diagram, which represents the Boolean equation for the top event. However, problems may occur with the conversion process of the fault tree to the BDD. If the ordering of the basic events is not chosen suitably, the size of the final BDD can grow exponentially. It is not possible to identify an optimum scheme for producing BDDs for all fault trees. In this paper an alternative conversion method is presented where BDDs for each of the gate types are formed and then joined together according to the type of the parent gate in the fault tree.

The effectiveness of this approach is compared with Rauzy's method using different efficiency measures, while working on the optimum connection technique.

Binary Decision Diagrams

A BDD is a directed acyclic graph, i.e. all paths through the BDD are in one direction and no loops can exist. The BDD is composed of terminal and non-terminal vertices (nodes) which are connected by branches. Terminal vertices correspond to the final state of the system, failure (1) or success (0), and non-terminal vertices correspond to the basic events of the fault tree. Each non-terminal vertex has a 1 branch, which represents basic event occurrence, and a 0 branch, which represents basic event non-occurrence. The fault tree and its equivalent BDD are presented in "figure 1". The BDD encodes system failure logic function in a disjoint form.

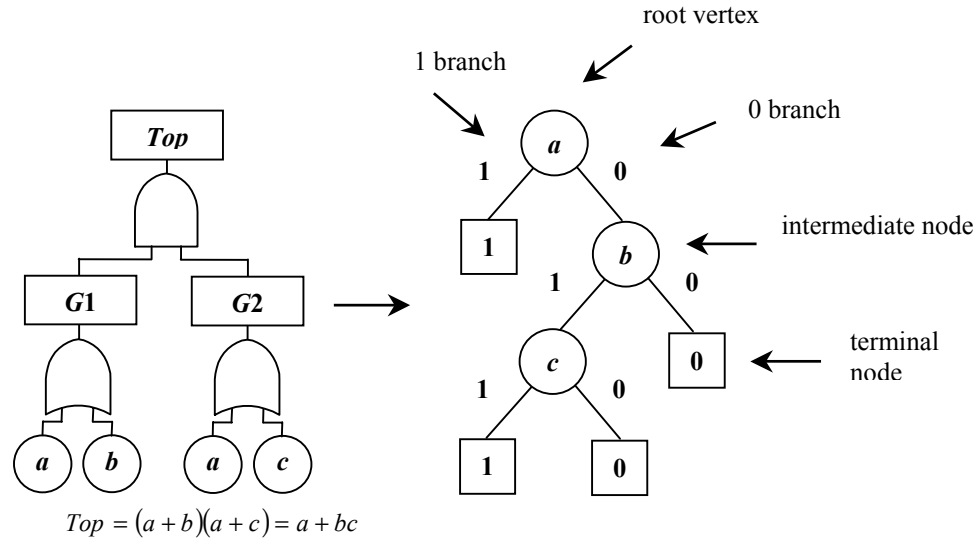


Figure 1 – Example of a Binary Decision Diagram

BDD Quantification

All paths through the diagram start at the root vertex and proceed to a terminal vertex, which marks the end of the path. Each path that terminates in a 1 state gives a cut set of the fault tree, as that particular combination of component failures which, if they all occur, will result in system failure. Only vertices that lie on the “1” branches of these paths are included in the cut sets. For example, in the BDD shown in “figure 1” there are two possible paths that terminate in 1 state. These are:

1. a
2. \bar{a}, b, c

This gives the two corresponding cut sets:

1. $\{a\}$
2. $\{b, c\}$.

In this example the BDD is in its minimal form and so generates only minimal cut sets (cut sets with both necessary and sufficient elements). However, this is not always the case. In order to obtain minimal cut sets the BDD has to undergo a minimisation procedure, introduced in “reference 1”, after which a new BDD is created that encodes only the minimal cut sets of the fault tree.

Since paths through the BDD are disjoint (mutually exclusive), the probability of occurrence of the top event, Q_{SYS} , can be expressed as the sum of the probabilities of the disjoint paths through the BDD. Each disjoint path represents a combination of working and failed components which leads to system failure. Therefore, events lying on both the 1 and 0 branches are included in the probability calculation. The probability of system failure for the BDD shown in “figure 1” is:

$$Q_{SYS} = q_a + (1 - q_a)q_bq_c \quad (1)$$

Other probabilistic properties of the system, such as the unconditional failure intensity and component importance measures, can also be calculated “(ref. 4)”.

BDD/FT Features

Fault Tree Analysis is the most widely used tool in system safety and reliability assessment. This technique analyses the causal relationships between component failures and system failure. The fault tree itself provides a visual representation of engineering failure logic and produces a complete description of the causes of system failure. However, even for moderate sized problems the calculation of minimal cut sets can be a time consuming task and the system failure probability is calculated by applying approximations.

Rather than analysing the fault tree directly the BDD method first converts the fault tree to a binary decision diagram. The BDDs are difficult to construct directly from the engineering system and they do not provide clear documentation of the system failure causes. When the quantitative analysis is performed, the BDD has the advantage that, due to the structure of the logic equation, exact probabilities can be calculated. There are no requirements to calculate minimal cut sets as an intermediate phase. However, the qualitative analysis of BDDs can be performed and minimal cut sets obtained “(ref. 5)”. The BDD method has been developed and used to overcome inaccuracy and inefficiency problems with conventional methods. It does however require an effective method to convert the fault tree structure into the BDD form. Two methods are now considered.

Construction method 1 – Rauzy

A commonly used method of constructing BDDs was developed by Rauzy “(ref. 1)” and proceeds by applying an **if-then-else (ite)** technique to each of the gates in the fault tree. The **ite** structure derives from the Shannon’s formula “(ref. 3)” such that if $f(x)$ is the Boolean equation for the top event, then by pivoting about any variable X the Shannon formula can be written as:

$$f(x) = Xf_1 + \bar{X}f_2 \tag{2}$$

where f_1 and f_2 are Boolean equations, known as the residues of f , with $X=1$ and $X=0$ respectively. The corresponding **ite** structure is $\text{ite}(X, f_1, f_2)$, which means that **if** X fails **then** consider f_1 , **else** consider f_2 . Therefore, in the BDD structure f_1 lies below the 1 branch of the node encoding X and f_2 lies below the 0 branch.

Once a variable ordering has been established, the following procedure can be implemented to construct the BDD. Let J and H be two nodes in the BDD where $J = \text{ite}(X, f_1, f_2)$ and $G = \text{ite}(Y, g_1, g_2)$.

1. if $X < Y$ (i.e. X appears before Y in the variable ordering) then
 $J < op > G = \text{ite}(X, f_1 < op > G, f_2 < op > G)$. (3)

2. if $X = Y$ then
 $J < op > G = \text{ite}(X, f_1 < op > g_1, f_2 < op > g_2)$. (4)

where $< op >$ corresponds to a Boolean operation of the gates in the fault tree.

An advantage of the **ite** algorithm is that the method automatically uses sub-node sharing. This not only reduces the computer memory requirements, as each **ite** structure is only stored once, but it also increases the efficiency, since once an **ite** structure has been calculated the process does not need to be repeated.

The **ite** method can be demonstrated by an example in “figure 2”. The ordering $c < a < d < b$ represents a simple top-down left-right traversal of the fault tree. Applying “equation 3” gives the expression for gates $G1$, $G2$ and Top :

$$\begin{aligned} G1 &= c + a + d \\ &= \text{ite}(c, 1, 0) + \text{ite}(a, 1, 0) + \text{ite}(d, 1, 0) \\ &= \text{ite}(c, 1, \text{ite}(a, 1, \text{ite}(d, 1, 0))) \end{aligned} \tag{5}$$

$$G2 = a + b$$

$$= \text{ite}(a, 1, \text{ite}(b, 1, 0)) \tag{6}$$

$$Top = G1 \cdot G2$$

$$= \text{ite}(c, \text{ite}(a, 1, \text{ite}(b, 1, 0)), \text{ite}(a, 1, \text{ite}(d, \text{ite}(b, 1, 0), 0))) \tag{7}$$

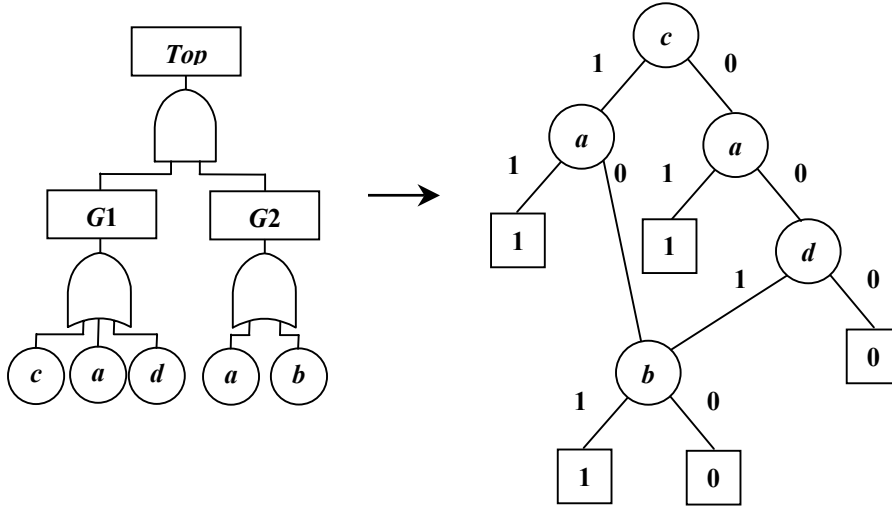


Figure 2 – BDD obtained using the *ite* technique

Construction method 2 – Component Connection method

The second method considered (“ref. 6”) uses the observed structure that results when a BDD is formed for an “AND” gate or an “OR” gate. In this way BDDs are constructed for fault trees initially without considering when basic events in the fault tree are duplicated (repeated). The resulting BDD then undergoes simplification to produce the final structure for analysis. This section of the paper describes the connection and simplification rules, with some alternative strategies, for producing a BDD for any fault tree. The basic event ordering, as required in Rauzy’s method, does not necessarily need to be established because the method can work without following any predetermined ordering scheme for the whole system. However, before the construction process can be implemented a selection scheme has to be specified which will govern the way in which gate inputs, either basic events or BDDs, are selected and combined.

In the approach presented gate inputs to any parent gate are considered in a left-right way so that this provides some ordering to consider basic events and also the BDD for a subtree of the left-most gate is built before considering the remaining gate inputs. When all BDDs, representing gate inputs of a parent gate, have been formed they are merged to obtain the BDD of the parent gate. This bottom-up process is over when the BDDs, representing gate inputs of the top-event, are combined. The following **connection rules** are used.

1. If two inputs in a fault tree are inputs to an “AND” gate, their representing nodes on a BDD are connected to each other through the 1 branch of the node. A similar statement holds true for the “OR” gate, i.e. the nodes are connected through the 0 branch.
2. If there are two BDDs, which represent two gate inputs of a parent gate, one of them is set to be the main BDD, according to the rule of selection. Then,
 - a) If two BDDs are inputs to an “AND” gate, the secondary BDD is connected to every terminal 1 node of the main BDD.
 - b) If two BDDs are inputs to an “OR” gate, the secondary BDD is connected to every terminal 0 node of the main BDD.

After every connecting operation we need to check for the repetition of basic events on any path through the BDD. If there is at least one repeated event, two **simplification rules** will be applied:

1. Each path starting at the node that represents the first occurrence of a repeated event in a path, and proceeding to a terminal vertex, must be adjusted in order to avoid the contradictory states of the repeated event in the BDD. The node, that represents the second occurrence of the event, needs to be replaced by the events below it on either its “working” branch or “failed” branch depending on the component state as specified by its first occurrence in the path. For example, if we traverse the BDD starting with the 1 branch of a node, the second appearance of that node should be replaced by the BDD structure below the 1 branch of this second node for consistency.
2. If the state of the system is the same regardless of the basic event occurrence or non-occurrence, the insignificant vertex must be removed. In other words, if the BDD structures below both branches of the node are the same, the node is irrelevant and needs to be replaced by the structure below either one of the branches.

To demonstrate this method it has been applied to the fault tree illustrated in “figure 2”. The resulting process is presented in “figure 3”. In this example the selection scheme used is that when combining BDDs representing inputs for any gates they are considered in a left-right manner and the left-most BDD is set to be the main BDD to which the other is joined. A left-right variable ordering for every gate in a fault tree is also adopted and the fault tree traversed in a bottom-up manner. First of all, gates $G1$ and $G2$ are constructed, shown in “figure 3(i)” and “figure 3(ii)” respectively, building two BDDs, which are both “OR” chains. Then the top event (“AND” gate) is considered. The left-most BDD (“figure 3(i)”) is selected as the main BDD. Then the BDD, illustrated in “figure 3(ii)”, is connected to every available 1 branch of the main BDD. The resulting BDD is presented in “figure 3(iii)”. Finally, the simplification rules are applied. The repeated *event a* is removed from the path $F1-F2-F6-F7$ replacing node $F6$ by the terminal node 1, since the path traverses the 1 branch of node $F2$, the first occurrence of the repeated event. In the same way the repeated *event a* is removed from the path $F1-F2-F3-F8-F9$, replacing node $F8$ by a direct connection to node $F9$. The final BDD is shown in “figure 3(iv)”.

No system-wide variable ordering was explicitly presented in this example, i.e. basic events were connected according to the order that they appear in the list of gate inputs. However, it is possible to apply a defined ordering scheme for the nodes which will be used in the construction method.

In forming the BDD of a parent gate the BDDs of its input events are merged together, one at a time. In this example BDDs were selected according to the order that gate inputs are listed, i.e. the BDD, presenting the left-most gate, is set to be the main BDD. Other selection schemes can be used which will, to some degree, affect the efficiency of the process. For example, BDDs can be ordered according to the position of their root vertex in an ordering scheme defined for the basic events or to minimise the number of available branches where connections will be made. The efficiency of different strategies can be analysed comparing the number of nodes in the final BDD and the processing time.

Both qualitative and quantitative analysis can be carried out on BDDs as in the first method. However, this method does not use sub-node sharing, therefore, there are some parts in the structure that are identical. This might lead to the inefficient memory usage. For example, in “figure 3(iv)” there are two identical nodes $F5$ and $F9$, which are not duplicated in Rauzy’s method.

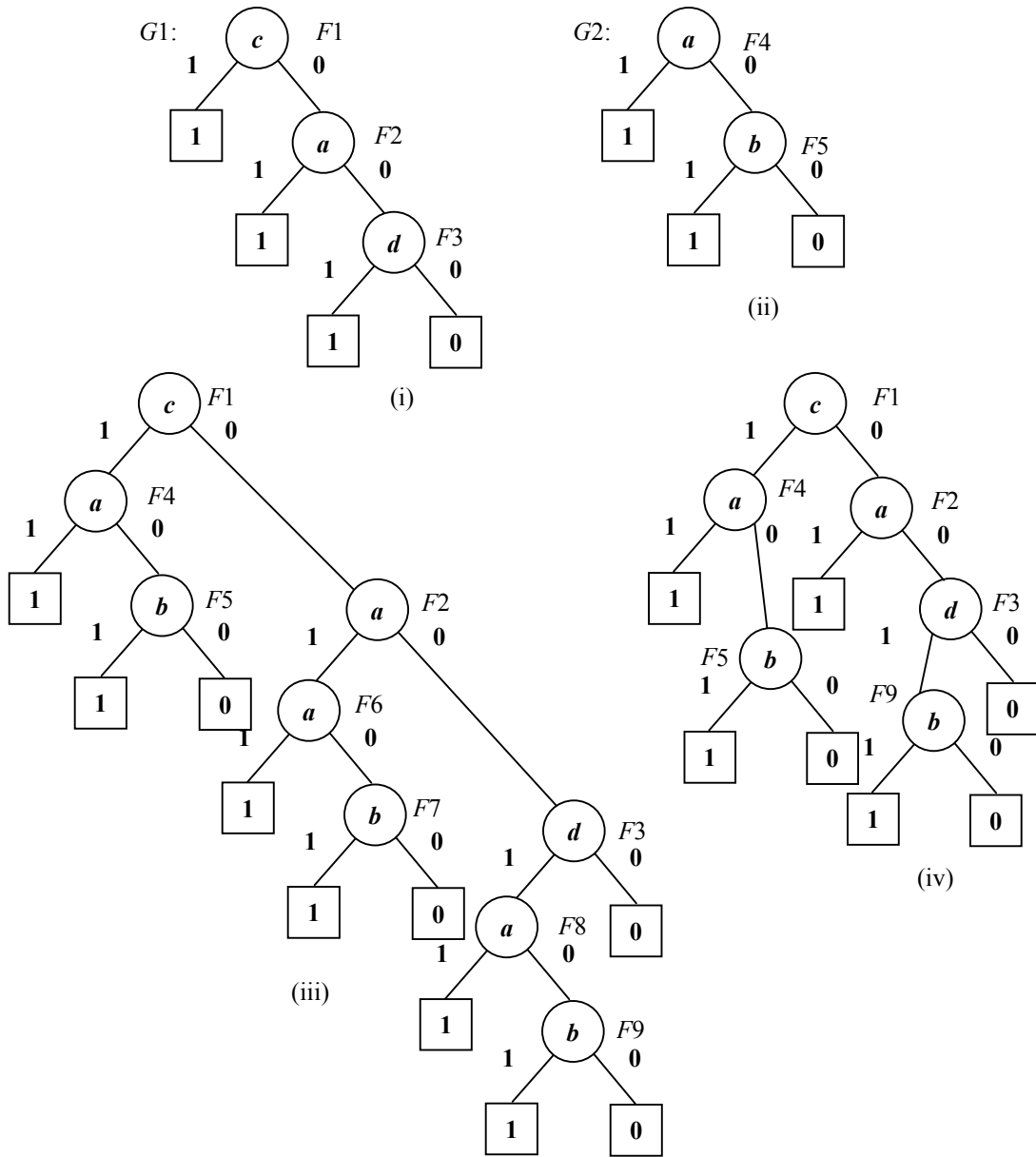


Figure 3 – BDD obtained using the Component Connection method

Comparison of methods

The performance of a method for fault tree conversion to the BDD form will be dependent upon the structure of the fault tree. An indication of the merit of a method can only be gauged over a large range of problems. As a comparison between the two BDD construction methods presented they have both been applied to a library of 12 fault trees. The characteristics of test fault trees are summarised in “table 1”. The first column is a label to identify the example fault tree, then the next three columns present the complexity of a fault tree in terms of the number of gates, the number of basic events and the number of repeated events. The last column presents the number of minimal cut sets. For each of the two methods, what are considered as effective variable ordering schemes and gate combinations input selection schemes have been used. The variable ordering that was applied for basic events is the modified top-down left-right approach “(ref. 7)”.

Number of a test FT	Number of gates	Number of basic events	Number of repeated events	Number of cut sets
1	48	94	38	6391
2	51	53	44	764
3	52	47	41	122
4	46	64	39	423
5	95	150	80	2621
6	48	114	27	66083
7	45	100	21	3344
8	46	84	40	1633
9	49	98	21	8113
10	48	72	43	493
11	38	58	26	898
12	37	77	17	45505

Table 1 – Complexity of test fault trees

The measurements that were chosen for the comparison of the two methods are the number of nodes in the final BDD and the processing time. The results obtained by applying the two methods to the fault trees in the library are shown in “table 2”.

Number of a test FT	Number of nodes			Processing time		
	Method 1	Method 2	Method 1 as a fraction of Method 2 (%)	Method 1	Method 2	Method 1 as a fraction of Method 2 (%)
1	12470	104214	12	4.125	5.563	74
2	860	2105	41	0.047	0.187	25
3	368	975	38	0.032	0.156	21
4	1472	2614	56	0.078	0.844	9
5	14109	590269	2	11.89	2018.188	1
6	18460	95650	19	9.609	1.625	591
7	16797	141588	12	8.531	3.203	266
8	1726	6650	26	0.078	1.094	7
9	1945	3850	51	0.109	0.875	12
10	1618	2921	55	0.078	0.484	16
11	1701	11751	14	0.078	0.64	12
12	1006	8629	12	0.047	0.234	20

Table 2 – Comparison of two construction methods by the number of nodes and the processing time

The first construction method resulted in smaller BDDs for all the example fault trees. The processing time was also shorter for almost all example fault trees, except two examples, (6) and (7). Therefore, Rauzy’s method has a big advantage over the Component Connection method. This is due at least partly to its capability to use sub-node sharing.

Conclusions

This paper presents two alternative techniques by which fault trees are converted to BDDs. The first method is Rauzy’s *ite* method, the second is the Component Connection method. Example fault trees have been used and the results for both methods compared. Processing time and number of nodes were used as efficiency measures while

working on what were considered efficient connection techniques for both methods. It has been shown that the Component Connection approach has a high demand for memory space since the identical parts in the BDD structure are repeated but not shared. It is shown that as a general fault tree to BDD conversion technique the method proposed by Rauzy performs best. If the Component Connection method is to compete with Rauzy's method then it must be capable of incorporating sub-node sharing.

References

1. Antoine Rauzy, "New Algorithms for Fault Tree Analysis", Reliability Engineering and System Safety, no. 40 (1993): 203-21.
2. W.E. Vesely, "A Time Dependent Methodology for Fault Tree Evaluation", Nuclear Design and Engineering, no. 13 (1970): 337-360.
3. Randale E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", IEEE Trans. Computers C-35, no. 8 (1986): 677-690.
4. R.M. Sinnamon, J.D. Andrews, "Improved Accuracy in Quantitative Fault Tree Analysis", Quality and Reliability Engineering International, no. 13 (1997): 285-292.
5. R.M. Sinnamon, J.D. Andrews, "Improved Efficiency in Qualitative Fault Tree Analysis", Quality and Reliability Engineering International, no. 13 (1997): 293-298.
6. Yuan-Shun Way, Der-Yu Hsia, "A simple component-connection method for building binary decision diagrams encoding a fault tree", Reliability Engineering and System Safety, no. 70 (2000): 59-70.
7. Karen A. Reay, "Efficient Fault Tree Analysis Using Binary Decision Diagrams", Doctoral Thesis, Loughborough University (2002).

Biography

John D. Andrews, Department of Aeronautical and Automotive Engineering, Loughborough University, Loughborough, Leicestershire, LE11 3TU, UK, telephone – +44 (0)1509 227 286, facsimile – +44 (0)1509 227 275, e-mail – J.D.Andrews@lboro.ac.uk

John Andrews is Professor of Systems Reliability in the Department of Aeronautical and Automotive Engineering. He joined Loughborough University in 1989 having previously gained nine years industrial research experience with British Gas. His current research interests concern the assessment of the safety and risk of potentially hazardous industrial activities. This research has been heavily supported by industrial funding. Over recent years grants have been secured from BAE Systems, MOD, Rolls-Royce, ExxonMobil and Bechtel. Professor Andrews has over one hundred journal/conference publications along with a jointly authored book 'Reliability and Risk Assessment' which is now in its second edition.

Rasa Remenyte, Department of Aeronautical and Automotive Engineering, Loughborough University, Loughborough, Leicestershire, LE11 3TU, UK, telephone – +44 (0)1509 227 243, facsimile – +44 (0)1509 227 275, e-mail – R.Remenyte@lboro.ac.uk

Rasa Remenyte is currently studying for a PhD at Loughborough University, having previously attained a masters degree with honours in mathematics at Kaunas University of Technology in Lithuania. During the course of these studies she attended Loughborough University as part of the Socrates\Erasmus exchange program, completing two modules of an industrial mathematical modelling course. Her current research studies are undertaken as part of the Risk and Reliability research group in the Aeronautical and Automotive Engineering Department of Loughborough University.