Analysis of Non-coherent Fault Trees Using Ternary Decision Diagrams

Rasa Remenyte-Prescott

Dep. of Aeronautical and Automotive Engineering

Loughborough University,

Loughborough, LE11 3TU, England

R.Remenyte-Prescott@lboro.ac.uk

John Andrews

Dep. of Aeronautical and Automotive Engineering

Loughborough University,

Loughborough, LE11 3TU, England

J.D.Andrews@lboro.ac.uk

Abstract

Risk and safety assessments performed on potentially hazardous industrial systems commonly utilise Fault Tree Analysis (FTA) to forecast the probability of system failure. The type of logic for the top event is usually limited to AND and OR gates which leads to a coherent fault tree structure. In non-coherent fault trees components' working states as well as components' failures contribute to the failure of the system. The qualitative and quantitative analyses of non-coherent fault trees can introduce further difficulties over and above those seen in the coherent case. It is shown that the Binary Decision Diagram (BDD) method can be used for this type of assessment. The BDD approach can improve the accuracy and efficiency of the quantitative analysis of non-coherent fault trees. This article demonstrates the value of the Ternary Decision Diagram method (TDD) for the qualitative analysis of non-coherent fault trees. Such analysis can be used to provide information to a decision making process for future actions of an autonomous system and therefore it must be performed in real time. In these circumstances fast processing and small storage requirements are very important. The TDD method provides a fast processing capability and small storage is achieved when a single structure is used for both qualitative and quantitative analyses. The efficiency of the TDD

method is discussed and compared to the performance of the established methods for analysis of non-coherent fault trees.

Keywords: fault tree analysis, binary decision diagrams, non-coherent fault trees, ternary decision diagrams

1. Introduction

Fault Tree Analysis (FTA) was first introduced in the 1960s and it is commonly used for the reliability assessment of complex industrial systems. Causes of system failure are analysed by performing qualitative and quantitative analyses. A large number of combinations of events which can cause system failure may be produced for real systems (minimal cut sets/prime implicant sets) and the calculation of these failure combinations can be time-consuming. Also, the determination of the exact top event probability requires lengthy calculations. For real systems this demand may exceed the capability of the available computers, introducing approximations into the analysis with the resulting loss of accuracy.

The Binary Decision Diagram (BDD) method **[1]** provides a more concise form for the logic function of a fault tree. It overcomes some disadvantages of conventional FTA techniques and provides an efficient and exact analysis of coherent and non-coherent fault trees. The BDD method is efficient for quantifying the likelihood of system failure occurrence because it does not require system failure modes as an intermediate step. It is also more accurate since approximations used in the traditional approach of kinetic tree theory **[2]** are not applied. Previous work on the efficiency and the accuracy of the BDD method is presented in **[3, 4]**.

Instead of analysing the fault tree directly, the BDD method first converts the fault tree to a binary decision diagram, which encodes the Boolean equation for the top event. The resulting structure function BDD (SFBDD) can be used in the quantitative analysis to calculate the top event probability

or frequency. An SFBDD is not of the correct form for the qualitative analysis and further processing is required. In the coherent case a list of minimal cut sets is obtained by using the minimisation technique **[1]**. In the non-coherent case a full set of prime implicants is determined by applying the consensus theorem **[5]** to pairs of prime implicant sets involving a normal and negated literal. There are several methods for the calculation of prime implicant sets proposed in the literature. A meta-products BDD method, the first approach to this problem, was presented in **[6]** and further developed in **[7]**. It was followed by a zero-suppressed BDD method (ZBDD), presented in **[8]**. The third alternative method was developed in **[9]** and it uses a labelled binary decision diagram (L-BDD). These methods produce prime implicant sets and have their advantages and disadvantages in the conversion and representation techniques.

A new alternative method for performing the qualitative analysis of non-coherent fault trees is proposed in this paper. In this approach a fault tree is converted to a ternary decision diagram (TDD). The main concept of a TDD was presented in **[10]**, which is expanded into an implementation methodology for fault tree analysis in this paper. Every node in a TDD has three branches: the 1 branch which represents the failure relevance of the component, the 0 branch which represents the repair relevance of the component (so far this is a conventional BDD presentation) and the consensus branch which represents the irrelevance of the component to the system failure. A TDD encodes all prime implicant sets, because the consensus branch for a node is calculated by applying the consensus theorem which gives all "hidden" prime implicant sets. However, the TDD can be non-minimal, thus, the minimisation process is performed to remove non-minimal paths from the 1 and 0 branches. The obtained TDD can be used for the quantitative analysis as well as the qualitative analysis.

2. Non-coherent fault trees

Fault trees are classified according to their logic function. If during fault tree construction only AND gates and OR gates are used, the resulting fault tree is defined as coherent. If NOT logic is used or directly implied the resulting fault tree can be non-coherent.

Introduce each component in the system by an indicator x_i to show the status of the component:

$$x_i = \begin{cases} 1 \text{ if component } i \text{ is failed,} \\ 0 \text{ if component } i \text{ is working.} \end{cases} \quad (1)$$

where $i = 1,2,\ldots n$, and n is the number of components in the system.

The logic structure of the fault tree can be expressed by a structure function ϕ :

$$\phi = \begin{cases} 1 \text{ if system is failed,} \\ 0 \text{ if system is working.} \end{cases} \quad (2)$$

$\phi = \phi(x)$, where $x = (x_1, x_2, \ldots, x_n)$.

According to the requirements of coherency **[5]**, a structure function $\phi(x)$ is coherent if:

1. Each component i is relevant to the system, i.e.

$$\phi(1_i, \mathbf{x}) \neq \phi(0_i, \mathbf{x}) \text{ for some } \mathbf{x} \; \forall i \, . \quad (3)$$

2. $\phi(x)$ is increasing (non-decreasing) for each x_i, i.e.

$$\phi(1_i, \mathbf{x}) \geq \phi(0_i, \mathbf{x}) \; \forall i \, . \quad (4)$$

where

$$\phi(1_i, \mathbf{x}) = \phi(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \, , \quad (5)$$

$$\phi(0_i, \mathbf{x}) = \phi(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n). \quad (6)$$

The second condition means that the system condition does not change or deteriorates as the component deteriorates. If the system is non-coherent for component i then for a particular state of the rest of components the system is failed when component i works and when component i fails the system is restored to the non-failed condition. As a consequence of this property, system failure might occur due to the repair of a failed component or for a failed system the failure of an additional

component may give a successful outcome of system performance. The fault tree becomes coherent if the NOT logic can be eliminated from the fault tree structure.

Consider a simple example in Figure 1. Cars *A* and *B* are approaching the junction with lights on red and should stop. Car *C* has the right of way and should proceed through the junction. Three basic events are considered:

A – Car *A* fails to stop,

B – Car *B* fails to stop,

C – Car *C* fails to continue.

The collision at the crossroads can happen in two ways:

- Car *A* fails to stop and hits car *C* which is moving
- Car *A* stops but car *B* drives into the back of it.

A fault tree representing causes of failure of the collision is shown in Figure 2. Working in a bottom-up way the following logic expression is obtained.

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

where "+" is OR, "·" is AND.

Therefore, $\{A, \overline{C}\}$ and $\{\overline{A}, B\}$ are prime implicants, as combinations of component conditions (working or failed) which are necessary and sufficient to cause system failure. This list is incomplete because there is on more failure mode for the system:

$$\{B, \overline{C}\}$$

i.e. if *B* fails to stop and *C* continues across the lights it does not matter what *A* does there will be a collision.

Therefore, the full logic expression for the Top event is:

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

which can be obtained by applying the consensus law:

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

3. Fault tree conversion to Binary Decision Diagram

For the fault tree to be converted to a BDD it first needs to be prepared so that in the non-coherent case the NOT logic is pushed down to the level of basic events by using De Morgan's laws, i.e.

$$\overline{A \cdot B} = \overline{A} + \overline{B} , \tag{7}$$

$$\overline{A + B} = \overline{A} \cdot \overline{B} . \tag{8}$$

Each node in a SFBDD is defined by an **ite**(if-then-else) structure. The **ite** structure **ite**(x, f_1, f_0) means that if x fails then consider function f_1, else consider function f_0. So, f_1 lies on the 1 branch of x and f_0 lies on the 0 branch in the diagram.

Before the conversion process takes place basic events in the fault tree are ordered. SFBDD construction then moves through the fault tree in a bottom-up manner applying the variable ordering in the conversion process.

Each basic event in the system is assigned an **ite** structure:

$$a = \textbf{ite}(a,1,0). \tag{9}$$

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

$$\overline{a} = \textbf{ite}(a,0,1). \tag{10}$$

For gates whose inputs have already been defined as an **ite** structure the main rule of the conversion process is applied, i.e. if J = **ite**(x, f_1, f_0) and H = **ite**(y, g_1, g_0) represent two inputs to a gate of logic type $\oplus$, then:

$$J \oplus H = \begin{cases} \mathbf{ite}(x, f_1 \oplus H, f_0 \oplus H) \text{ if } x < y \text{ in the ordering,} \\ \mathbf{ite}(x, f_1 \oplus g_1, f_0 \oplus g_0) \text{ if } x = y \text{ in the ordering.} \end{cases} \quad (11)$$

For small examples the variable ordering is largely irrelevant. Variable ordering schemes are discussed in **[11, 12]**. For the fault tree example in Figure 2 consider the variable ordering scheme $A < B < C$. Applying the conversion rules (9) - (11) to the fault tree results in a SFBDD presented in Figure 3.

4. Calculation of prime implicant sets

Knowledge of prime implicant sets can be valuable in gaining an understanding of the system and the causes of system failure. It can help to develop a repair schedule for failed components if a system cannot be taken off line for repair. For systems whose state has all the failed components in any prime implicant care should be taken to ensure that the repair of other components does not then cause the remaining functioning events in the prime implicant. 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 and a conversion process is usually performed to produce a different form of a BDD which encodes only the prime implicants.

Consider a general component x in a non-coherent system. In a prime implicant set component x can appear 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 directly from the BDD. Therefore, additional methods for encoding prime implicant sets are required.

5. TDD method

An approach to build a Ternary Decision Diagram (TDD) for the analysis of non-coherent fault trees is proposed in this section. It employs the consensus theorem and creates, in addition to the two branches of the BDD, a third branch for every node, called the consensus branch. This third branch encodes the "hidden" prime implicant sets. The minimisation algorithm **[1]** is applied to remove non-minimal paths and obtain prime implicant sets only.

5.1 Conversion

Every node in the TDD has three exit branches. A new **ifre** structure is defined which separates relevant and irrelevant components and also distinguishes between the type of relevancy, i.e. failure relevant and repair relevant. The **ifre** structure for a node x is given in Figure 4. So, if:

$$\phi = \textbf{ifre}(x, f_1, f_0, f_2), \tag{12}$$

then

$$\phi = x\, f_1 + \bar{x}\, f_0 + f_2, \tag{13}$$

where

$$f_2 = f_1 \cdot f_0. \tag{14}$$

The 1 branch encodes prime implicant sets for which component x is failure relevant, the 0 branch encodes prime implicant sets for which component x is repair relevant, and the "C" (consensus) branch encodes prime implicant sets for which component x is irrelevant. The **ifre** structure shown in Figure 4 can be interpreted as follows:

If x is **f**ailure relevant

then consider f_1,

else if x is **r**epair relevant

then consider f_0

else consider f_2

endif

Function f_2 encodes prime implicant sets for which x is irrelevant, but this branch is not important for all components. For components that are only failure or repair relevant but not both this branch can be kept "empty". In our method we assign f_2 = NIL, if the conjunction of the two branches $f_1 \cdot f_0$ is not required. While operating the new symbol in the Boolean algebra, it is defined that NIL $\oplus$ A= NIL. Symbol NIL is used to identify cases when the "C" branch is not required and no Boolean operations that involve this branch are needed.

The conversion technique to compute the TDD from the non-coherent fault tree is an extension of the method used to develop the conventional BDD. First of all, basic events of the fault tree must be ordered. Then the following process is presented:

- By the application of De Morgan's laws push NOT logic down through the fault tree until the basic event level is reached.
- Each basic event is assigned an **ifre** structure:
    - If a is only failure or repair relevant:

$$a = \textbf{ifre}(a,1,0,\text{NIL}), \tag{15}$$

$$\bar{a} = \textbf{ifre}(a,0,1,\text{NIL}). \tag{16}$$

    - If a is failure and repair relevant:

$$a = \textbf{ifre}(a,1,0,0) \tag{17}$$

$$\bar{a} = \textbf{ifre}(a,0,1,0) \tag{18}$$

- Traversing the fault tree in a bottom-up manner and considering gates whose inputs have been expressed in an **ifre** format gives:

If J = **ifre**(x, f_1, f_0, f_2) and H = **ifre**(y, g_1, g_0, g_2),

$$\text{then } J \oplus H = \begin{cases} \textbf{ifre}(x, K_1, K_0, K_1 \cdot K_0) & \text{if } x < y \text{ in the ordering,} \\ \textbf{ifre}(x, L_1, L_0, L_1 \cdot L_0) & \text{if } x = y \text{ in the ordering.} \end{cases} \quad (19)$$

here $K_1 = f_1 \oplus H$, $K_0 = f_0 \oplus H$, $L_1 = f_1 \oplus g_1$, $L_0 = f_0 \oplus g_0$, $K_1 \cdot K_0$ – consensus of K_1 and K_0, $L_1 \cdot L_0$ – consensus of L_1 and L_0.

If component x is failure or repair relevant, $K_1 \cdot K_0$ = NIL, $L_1 \cdot L_0$ = NIL in equation 19.

Within each **ifre** calculation an additional consensus calculation is performed to ensure all the "hidden" prime implicant sets are encoded in the TDD. It calculates the conjunction of the 1 and the 0 branch of every node and thus identifies the consensus of each node. If a node in the TDD encodes component which is only failure or repair relevant the conjunction of the 1 and 0 branch for the node is not required, because there are no "hidden" prime implicant sets associated with this component. This property makes the TDD method an efficient technique for performing the qualitative analysis of non-coherent fault trees.

Consider the fault tree in Figure 2. Introducing the ordering of basic events $A < B < C$ and applying the rules described in (15)-(19) gives the TDD in Figure 5.

It can be seen that the TDD in Figure 5 is different from the SFBDD in Figure 3 only with its "C" branch that represents the intersection of the 1 and 0 branches. Only for node F1 there is a new structure F4 created as the "C" branch. The other nodes have the "C" branch leading to value NIL, since they encode variables that only appear as failure or repair relevant. To obtain prime implicant sets non-minimal combinations from every path need to be removed.

5.2 Minimisation

Once a fault tree is converted to a TDD there is no guarantee that the resulting structure will be minimal and give exact prime implicant sets. In order to perform the qualitative analysis a minimisation procedure needs to be implemented.

The algorithm developed by Rauzy for minimising the BDD **[1]** was extended to create a minimal TDD. Consider a general node in the TDD which is represented by the function *F*, where

$$F = \textbf{ifre}(x, G, H, K). \tag{20}$$

The process of minimisation is described in three cases:

1. Component *x* is failure and repair relevant
2. Component *x* is failure relevant
3. Component *x* is repair relevant

In case 1, the set of all minimal solutions of *F* is minimal solutions of *G* and *H* ($G_{\min}$ and $H_{\min}$), that are not minimal solutions of *K*, and also all minimal solutions of *K* ($K_{\min}$). Then if δ is a set of minimal solutions of *G*, which are not a minimal solution of *K*, then the intersection of δ and *x* ($\delta \wedge x$) will be minimal solutions of *F*. Similarly, let γ be a set of minimal solutions of *H* which are not minimal solutions of *K*, then the intersection of γ and $\bar{x}$ ($\gamma \wedge \bar{x}$) will be minimal solutions of *F*.

The set of all the minimal solutions of *F* ($sol_{\min}(F)$) will also include the minimal solutions of *K*, so:

$$sol_{\min}(F) = (\delta \wedge x) \vee (\gamma \wedge \bar{x}) \vee K_{\min}\,. \tag{21}$$

The set $sol_{\min}(F)$ represents the minimal solutions of *F* by removing any minimal solutions of *G* and *H* that are also minimal solutions of *K*.

In case 2, where x is failure relevant, K = NIL and the calculation of prime implicant sets is equivalent adequate to the BDD case where the "C" branch does not exist, i.e.

$$sol_{\min}(F) = (\delta \wedge x) \vee H_{\min} . \quad (22)$$

The set $sol_{\min}(F)$ represents the minimal solutions of F by removing any minimal solutions of G that are also minimal solutions of H.

In case 3, where x is repair relevant, K = NIL and the calculation of prime implicant sets is defined as:

$$sol_{\min}(F) = (\gamma \wedge \bar{x}) \vee G_{\min} . \quad (23)$$

The set $sol_{\min}(F)$ represents the minimal solutions of F by removing any minimal solutions of H that are also minimal solutions of G.

5.3 Obtaining prime implicant sets

Traversing the TDD in Figure 5, which is already in its minimal form, from the root vertex to terminal 1 vertices give all three prime implicant sets. Again, the algorithm depends on the relevance of node variable and the value of "C" branch.

1. If $K \neq$ NIL, traversing the 1 branch of node x results in a failed state of a component in a particular failure mode. Traversing the 0 branch of node x results in a working state of a component in a particular failure mode. Finally, traversing the "C" branch of node x does not include that component in a particular failure mode at all.
2. If K = NIL and x is failure relevant, traversing the 1 branch of node x results in a failed state of a component in a particular failure mode. Traversing the 0 branch of node x does not include that component in a particular failure mode.
3. If K = NIL and x is repair relevant, traversing the 0 branch of node x results in a repaired state of a component in a particular failure mode. Traversing the 1 branch of node x does not include that component in a particular failure mode.

The three paths obtained give the three prime implicant sets:

$F1$-$F2$	$\{A, \overline{C}\}$
$F1$-$F3$	$\{\overline{A}, B\}$
$F1$-$F4$-$F5$	$\{B, \overline{C}\}$

This method provides an advanced technique for encoding prime implicant sets.

5.4 Established methods for qualitative analysis

This section presents the existing methods for converting non-coherent fault trees to BDDs and obtaining prime implicant sets. In the later sections the efficiency of all methods, including the TDD method, will be investigated and compared using some example fault trees.

5.4.1 Meta-products BDD method

This method converts a SFBDD to a meta-products BDD which produces all prime implicant sets. A meta-products BDD obtained is in its minimal form. The method was developed in **[6,7]** where two variables are associated 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, $MP(\pi)$, 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 $MP(\pi)$:

$$MP(\pi) = \begin{cases} P_x \wedge S_x \ \textit{if} \ \ x \in \pi, \\ P_x \wedge \overline{S}_x \ \textit{if} \ \ \overline{x} \in \pi, \\ \overline{P}_x \ \text{if neither } x \text{ nor } \overline{x} \text{ belongs to } \pi. \end{cases} \tag{24}$$

Consider node F in a SFBDD, where F = **ite**(x, $F1$, $F0$). The meta-products BDD, that describes prime implicant sets using equation (20), is expressed as:

$$PI(F) = \textbf{ite}(P_x, \textbf{ite}(S_x, P1, P0), P2), \tag{25}$$

where

$$P2 = PI(F1 \cdot F0), \tag{26}$$

$$P1 = PI(F1) \cdot \overline{P2}\,, \tag{27}$$

$$P0 = PI(F0) \cdot \overline{P2}\,. \tag{28}$$

x is the first element in the variable ordering, $PI(F)$ represents the structure of a meta-products BDD, PI is used to denote the prime implicants. $P2$ 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.

The SFBDD in Figure 3 has been converted to a meta-products BDD, shown in Figure 6.

Now it is possible to obtain the meta-products and identify the prime implicant sets. Every path from the root node to a terminal 1 gives a prime implicant set.

$P_A \wedge S_A \wedge \overline{P}_B \wedge P_C \wedge \overline{S}_C =$	$\{A, \overline{C}\}$
$P_A \wedge \overline{S}_A \wedge P_B \wedge S_B \wedge \overline{P}_C =$	$\{\overline{A}, B\}$
$\overline{P}_A \wedge P_B \wedge S_B \wedge P_C \wedge \overline{S}_C =$	$\{B, \overline{C}\}$

The number of nodes in a meta-products BDD increases largely since every basic event x is presented by two nodes, P_x and S_x. The process can be time-consuming.

5.4.2 ZBDD method

An alternative method presented by Rauzy in **[8]** uses the idea of zero-suppressed BDDs (ZBDD). This method requires to label nodes with failed and/or working states of basic events and to decompose prime implicant sets according to the presence of a given state of a basic event. Zero-suppressed BDDs are BDDs based on a reduction rule. This data structure provides a unique and compact representation which is more efficient and simpler than the usual BDDs when manipulating sets in combinatorial problems.

The principle of this algorithm is to traverse the SFBDD that encodes structure function ϕ in a depth-first way and to build a ZBDD that encodes the prime implicant sets of ϕ in a bottom-up way. The conversion rule is divided in four cases. Consider node F in a SFBDD, where F = **ite**(x, $F1$, $F0$).

Case 1: if basic event x appears in its failed and working states then:

$$PI(F) = xP1 + \bar{x}P0 + P2, \tag{29}$$

where

$$P2 = PI(F1 \cdot F0), \tag{30}$$

$$P1 = PI(F1) \setminus P2, \tag{31}$$

$$P0 = PI(F0) \setminus P2. \tag{32}$$

Here "\" is operator "without" **[1]** that is used minimising conventional BDDs.

Case 2: if basic event x appears in its failed state only then

$$PI(F) = xP1 + P0, \tag{33}$$

where

$$P0 = PI(F0), \tag{34}$$

$$P1 = PI(F1) \setminus P0. \tag{35}$$

Case 3: if basic event x appears in its working state only then it is considered in a similar way to case 2.

Case 4: if basic event x does not appear in the system then

$$PI(F) = PI(F1 + F0). \tag{36}$$

Applying this method to the SFBDD in Figure 3 gives the ZBDD in Figure 7.

Every path from the root vertex to terminal vertex 1 presents a prime implicant set. Therefore, this ZBDD contains three prime implicant sets:

$A \wedge \overline{C} =$	$\{A, \overline{C}\}$
$\overline{A} \wedge B =$	$\{\overline{A}, B\}$

$B \wedge \overline{C} =$	$\{B, \overline{C}\}$

The ZBDD is an efficient technique where all prime implicant sets are described by a compact and easy handling structure.

5.4.3 Labelled variable method

The labelled variable method **[9]** provides another alternative method for constructing BDDs for non-coherent fault trees. BDDs constructed using this approach consist of variables that are labelled according to their type. They are called labelled binary decision diagrams (L-BDDs). The structure function $\phi(\mathbf{x})$ of a non-coherent fault tree may contain three different types of basic events. For example, the function $\phi(x) = a \cdot b + \overline{a} \cdot \overline{c} + b \cdot \overline{c}$ contains a double form (DF) variable *a* that appears in both states, a single form positive (SFP) variable *b* and a single form negative (SFN) variable *c*. In the further presentation the SFP variable *x* will be simply presented by *x*, the SFN variable *x* will be labelled as "$*x*" and the DF variable *x* will be labelled as "&*x*".

The conversion process for computing the L-BDD from the non-coherent fault tree is an extension to the method used to develop the SFBDD. Considering the ordering &*x* < *x* < $*x* implements the additional equations:

$$\text{If } J = \textbf{ite}(x, f_1, f_0) \text{ and } H = \textbf{ite}(\$x, g_1, g_0), \quad \text{then } J \oplus H = \textbf{ite}(\&x, f_1 \oplus g_0, f_0 \oplus g_1) \tag{37}$$

$$\text{If } J = \textbf{ite}(\&x, f_1, f_0) \text{ and } H = \textbf{ite}(x, g_1, g_0), \quad \text{then } J \oplus H = \textbf{ite}(\&x, f_1 \oplus g_1, f_0 \oplus g_0) \tag{38}$$

$$\text{If } J = \textbf{ite}(\&x, f_1, f_0) \text{ and } H = \textbf{ite}(\$x, g_1, g_0), \quad \text{Then } J \oplus H = \textbf{ite}(\&x, f_1 \oplus g_0, f_0 \oplus g_1) \tag{39}$$

Applying the conversion rules to the fault tree in Figure 2 results in a L-BDD presented in Figure 8 (the top BDD). The L-BDD does not provide all the information for the qualitative analysis, therefore some additional calculations are performed in order to get all prime implicant sets.

Visiting the L-BDD in the bottom-up way the procedure to be applied to the node F = **ite**(x, $F1$, $F0$) to determine the prime implicants is as follows:

If x has label "&", then:

$$PI(F) = xP1 + \$xP0 + P2 \tag{40}$$

where

$$P2 = F1 \cdot F0, P1 = F1 \setminus P2, P0 = F0 \setminus P2 . \tag{41}$$

Else:

$$PI(F) = \alpha \cdot P1 + F0 \tag{42}$$

where

$$\alpha = x \; or \; \$x, P1 = F1 \setminus F0 . \tag{43}$$

"\" is the operator "without" proposed by Rauzy **[1]**. Some extra rules are applied in the cases with labelled variables.

The heaviest operation is the intersection $P2 = F_1 \wedge F_0$, shown in Figure 8, the bottom BDD. The three prime implicant sets are obtained, tracing all paths from root vertex to terminal vertex 1 and taking into account the results of intersection:

$\& A \wedge \$C =$	$\{A, \overline{C}\}$
$\overline{\& A} \wedge B =$	$\{\overline{A}, B\}$
$B \wedge \$C =$	$\{B, \overline{C}\}$

The L-BDD method uses the prior information about the type of every variable, however, the labelling introduces some additional variables and increases the size of the structure.

The three established methods for the calculation of prime implicant sets will be considered for the efficiency test of the TDD method.

5.5 Quantitative analysis using TDDs

In order to perform the quantitative analysis for non-coherent fault trees using the BDD method, a non-coherent fault tree is converted to a SFBDD that represents the structure function of the fault tree. In the TDD method the non-coherent fault tree is converted to the TDD that has three branches from each node. The third branch is created to encode all prime implicants of the system. However, the TDD can be used not only for the qualitative analysis but also for the quantitative analysis.

5.5.1 Top event probability

Consider node F in the TDD, F = **ifre**(x, f_1, f_0, f_1 f_0). The structure function $\phi(\mathbf{x})$ was expressed in (12), i.e. $\phi(\mathbf{x}) = xf_1 + \bar{x}f_0 + f_1 f_0$. Using the pivotal decomposition to the structure function of order n it is possible to express it in terms of structure functions that are of order n-1. Pivoting $\phi(\mathbf{x})$ about variable x and applying the absorption law gives:

$$\phi(\mathbf{x}) = x\phi(1_i, \mathbf{x}) + \bar{x}\phi(0_i, \mathbf{x}) = x\left(f_1 + f_1 f_0\right) + \bar{x}\left(f_0 + f_1 f_0\right) = xf_1 + \bar{x}f_0. \quad (44)$$

Then the expectation of $\phi(\mathbf{x})$ is obtained and the top event probability is calculated:

$$Q_{SYS} = E\left(\phi(\mathbf{x})\right) = q_x Q(f_1) + (1 - q_x) Q(f_0). \quad (45)$$

where q_x is the failure probability of component x.

Therefore, the probability of the top event, Q_{SYS}, is the sum of the probabilities of the disjoint paths through the TDD. The disjoint paths, that are taken into account, can be found by tracing all paths from the root vertex via the 1 and 0 branches to terminal 1 vertices. The disjoint paths via the “C” branch are not included in the quantification process.

If $f_1\, f_0$ = NIL, $\phi(\mathbf{x}) = xf_1 + \bar{x}f_0$ which allows to calculate the Q_{SYS} in the same way.

5.5.2 Birnbaum's measure of importance

The probability that component *i* is critical to system failure can be expressed as the probability that component *i* is failure critical, $G_i^F(q)$, or the probability that component *i* is repair critical, $G_i^R(q)$, **[15]**:

$$G_i(q) = G_i^F(q) + G_i^R(q). \tag{46}$$

Beeson and Andrews **[16]** showed how to define Birnbaum's measure of component failure importance as the probability that component i is failure relevant to the system state given by

$$G_i^F(q) = E[\phi_{i=1}] - E[\phi_{i='-'}]. \tag{47}$$

where $E[\phi_{i=1}]$ is the probability that component *i* is either failure relevant or irrelevant to the state of the system; and $E[\phi_{i='-'}]$ is the probability that component *i* is irrelevant to the state of the system. Similarly is defined Birnbaum's measure of component repair importance:

$$G_i^R(q) = E[\phi_{i=0}] - E[\phi_{i='-'}]. \tag{48}$$

where $E[\phi_{i=0}]$ is the probability that component *i* is repair failure relevant or irrelevant to the state of the system.

It is possible to calculate $E[\phi_{i=1}]$, $E[\phi_{i=1}]$ and $E[\phi_{i=1}]$ from the ternary decision diagram. The procedure for calculating the failure and repair criticality of component *i* is outlined below:

$$E[\phi_{i=1}] = \sum_{x_i} pr_{x_i}(q)\, po_{x_i}^{1,C}(q). \tag{49}$$

$$E[\phi_{i=0}] = \sum_{x_i} pr_{x_i}(q)\, po_{x_i}^{0,C}(q). \tag{50}$$

$$E[\phi_{i='-'}] = \sum_{x_i} pr_{x_i}(q)\, po^{C}_{x_i}(q). \tag{51}$$

where:

$pr_{x_i}(q)$ - the probability of the path section from the root vertex to node x_i,

$po^{1,C}_{x_i}(q)$ - the probability of the path section from the 1 branch of node x_i to a terminal 1 vertex via 1 or 0 branches of non-terminal nodes,

$po^{0,C}_{x_i}(q)$ - the probability of the path section from the 0 branch of node x_i to a terminal 1 vertex via 1 or 0 branches of non-terminal nodes,

$po^{C}_{x_i}(q)$ - the probability of the path section from the "C" branch of node x_i to a terminal 1 vertex via 1 or 0 branches of non-terminal nodes.

Therefore, the failure and repair criticality of component *i* using the TDD are expressed as:

$$G^{F}_{i}(q) = \sum_{x_i} pr_{x_i}(q)\left(po^{1,C}_{x_i}(q) - po^{C}_{x_i}(q)\right). \tag{52}$$

$$G^{R}_{i}(q) = \sum_{x_i} pr_{x_i}(q)\left(po^{0,C}_{x_i}(q) - po^{C}_{x_i}(q)\right). \tag{53}$$

These expressions are true for every component *i* that is failure and repair relevant **[17]**.

For the other two cases, i.e. when component *i* is either failure or repair relevant and the "C" branch is "empty", Birnbaum's measure of importance is expressed in the following equations. If component *i* is only failure relevant, then:

$$G^{F}_{i}(q) = \sum_{x_i} pr_{x_i}(q)\left(po^{1,C}_{x_i}(q) - po^{0,C}_{x_i}(q)\right). \tag{54}$$

$$G^{R}_{i}(q) = 0. \tag{55}$$

If component *i* is only repair critical, then:

$$G_i^F(q) = 0. \tag{56}$$

$$G_i^R(q) = \sum_{x_i} pr_{x_i}(q)\left(po_{x_i}^{0,C}(q) - po_{x_i}^{1,C}(q)\right). \tag{57}$$

Using the TDD in Figure 5 and applying the above equations, the Birnbaum's measure of importance can be calculated for all components in the system.

For component *A*: $G_A^F(q) = p_C - q_B p_C = p_B p_c$, $G_A^R(q) = q_B - q_B p_C = q_B q_c$.

For component *B*: $G_B^F(q) = p_A$, $G_B^R(q) = 0$.

For component *C*: $G_C^F(q) = 0$, $G_C^R(q) = q_A$.

Summarising, if the quantitative analysis is required as well as the qualitative analysis, the TDD before the minimisation can be used for the quantification process. Additional calculations for obtaining the SFBDD are not required as it is required in some of the established methods. This property makes the TDD method an efficient approach for full analysis of non-coherent fault trees

5.6 Efficiency comparison between the TDD method and the established methods

The efficiency of the TDD method and the established methods for calculating prime implicant sets were investigated and compared using a benchmark set of medium sized fault trees for engineering systems from several industries. The performance over 16 example fault trees was obtained, since each method may perform well on some fault trees dependent upon the fault tree structure. The performance of each method over a range of test cases is monitored. The complexity of the 16 fault trees is indicated in columns 2, 3 and 4 of Table 1, representing the number of gates, the number of events and the number of prime implicant sets in their solution. Example fault trees were simplified prior to the conversion process, using the reduction **[13]** and modularisation **[14]** techniques. The number of complex and modular events are shown in columns 5 and 6.

The number of nodes using the TDD method, the meta-products BDD method, the ZBDD method and the L-BDD method are presented in columns 7-10. The number of nodes in the TDD method describes the sum of the number of nodes in the TDD before the minimisation (which is also used for the quantitative analysis) and the number of nodes in the TDD after the minimisation. The number of nodes for the second method covers the number of nodes in the SFBDD and the meta-products BDD. For the ZBDD method the sum of the number of nodes in the SFBDD and the number of nodes in the ZBDD is presented. The number of nodes in the L-BDD method contains the sum of the number of nodes in the L-BDD before applying the minimisation, the number of nodes of the additional structures after applying the conjunction and the number of nodes in the minimised L-BDD. Similarly, the processing time covers the time taken to convert example fault trees to BDDs and perform the qualitative analysis. Results of processing time are shown in columns 11-14 of Table 1 for the four methods respectively. Total number of nodes and processing time for the four methods are shown in Table 2.

As it is shown in Table 2 the TDD method performed as well as the ZBDD method. Both methods out performed the meta-products and L-BDD methods resulting in the smallest final BDDs in shortest calculation time. The L-BDD method gave the second worst result and the meta-products BDD method required the longest processing time since the size of the problem increased marginally.

These results showed that the TDD method provides an efficient way to represent prime implicant sets, where “hidden” sets are obtained by applying the conjunction of the two branches. It also has an ability to do so only if it is required, avoiding the generation of a structure that is not needed. This is achieved using the information about the failure/repair relevance of the component which determines if the conjunction of the two branches is performed or not. The final advantage of this technique is the fact that the quantitative analysis can be also performed using the TDD before the minimisation.

6. Conclusions

This paper presents a new technique which is developed for application to finding prime implicant sets of non-coherent fault trees. Since the introduction of NOT logic to the logic function expands the calculation time and increases the size of the problem, the BDD method can be used as an efficient way for qualitative and quantitative analyses of non-coherent fault trees. This paper proposes a new alternative technique that produces a ternary decision diagram, which allows the calculation of all prime implicants directly. Its efficiency is analysed and compared with the established methods - the conventional algorithm that produces a meta-products BDD, the zero-suppressed BDD method and the labelled BDD method - using some example fault trees. Efficiency analysis indicates that the new proposed TDD method provides as good a representation of prime implicant sets as other methods and has the advantage of being suitable for both qualitative and quantitative analyses of non-coherent fault trees.

References

1 **Rauzy, A.** New Algorithms for Fault Tree Analysis, *Reliability Engineering and System Safety*, 1993, **40**, pp203-211.

2 **Vesely, W.E.** A Time Dependent Methodology for Fault Tree Evaluation, *Nuclear Design and Engineering*, 1970, **13**, pp337-360.

3 **Sinnamon, R.M.** and **Andrews, J.D.** Improved Efficiency in Qualitative Fault Tree Analysis, *Quality and Reliability Engineering International*, 1997, **13**, pp293-298.

4 **Sinnamon, R.M.** and **Andrews, J.D.** Improved Accuracy in Quantitative Fault Tree Analysis, *Quality and Reliability Engineering International*, 1997, **13**, pp285-292.

5 **Andrews, J.D.** To Not or Not to Not!!, *Proceedings of the 18th International System Safety Conference*, 2000, pp267-274.

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*, 1992, pp113-128.

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*, 1997, **58**, pp225-235.

8 **Rauzy, A.** Mathematical Foundation of Minimal Cutsets, *IEEE Transactions on Reliability*, 2001, **50**(4), pp389-396.

9 **Contini, S.** Binary Decision Diagrams with Labelled Variables for Non-Coherent Fault Tree Analysis, European Commission, Joint Research Centre, 2005.

10 **Sasao, T.** Ternary Decision Diagrams and their Applications, Chapter 12, pp269-292, Representations of Discrete Functions, Edited by Sasao, T. and Fujita, M. Kluwer Academic Publishers, 1996.

11 **Bartlett, L.M.** and **Andrews, J.D.** Comparison of Two New Approaches to Variable Ordering for Binary Decision Diagrams, *Quality and Reliability Engineering International*, 2001, **17**, pp151-158.

12 **Bartlett, L.M.** and **Andrews, J.D.** Selecting an Ordering Heuristic for the Fault Tree Binary Decision Diagram Conversion Process Using Neural Networks, *IEEE Transactions on Reliability*, 2002, **51**(3), pp344-349.

13 **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, 1976.

14 **Dutuit, Y.** and **Rauzy, A.** A Linear-Time Algorithm to Find Modules of Fault Trees, *IEEE Trans. Reliability*, 1996, **45**(3), pp422-425.

15 **Andrews, J.D.** and **Beeson, S.** Birnbaum's Measure of Component Importance for Non-coherent Systems, *IEEE Trans. Reliability*, 2003, **52**(2), pp213-219.

16 **Beeson, S.** and **Andrews, J.D.** Calculating the Failure Intensity of a Non-coherent Fault Tree Using the BDD Technique, *Quality and Reliability Engineering International*, 2004, **20**, pp225-235

17 **Remenyte-Prescott, R.** System Failure Modelling Using Binary Decision Diagrams, *Doctoral Thesis*, Loughborough University, 2007