Remenyte-Prescott, Rasa and Andrews, John (2009) An efficient real-time method of analysis for non-coherent fault trees. Quality and Reliability Engineering International, 25 (2). pp. 129-150. ISSN 1099-1638

An Efficient Real-time Method of Analysis for Non-coherent Fault Trees

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

Abstract

Fault tree analysis is commonly used to assess the reliability of potentially hazardous industrial systems. The type of logic is usually restricted to AND and OR gates which makes the fault tree structure coherent. In non-coherent structures not only components' failures but also components' working states contribute to the failure of the system. The qualitative and quantitative analyses of such fault trees can present additional difficulties when compared to the coherent versions. It is shown that the Binary Decision Diagram (BDD) method can overcome some of the difficulties in the analysis of non-coherent fault trees.

This paper presents the conversion process of non-coherent fault trees to BDDs. A fault tree is converted to a BDD that represents the system structure function (SFBDD). A SFBDD can then be used to quantify the system failure parameters but is not suitable for the qualitative analysis. Established methods, such as the meta-products BDD method, the zero-suppressed BDD (ZBDD) method and the labelled BDD (L-BDD) method, require an additional BDD that contains all prime implicant sets. The process using some of the methods can be time consuming and not very efficient. In addition, in real time applications the conversion process is less important and the requirement is to provide an efficient analysis. Recent uses of the BDD method are for real time system prognosis. In such situations as events happen, or failures occur the prediction of mission success is updated and used in the decision making process. Both qualitative and quantitative assessment are required for the decision making. Under these conditions fast processing and small storage requirements are essential. Fast processing is a feature of the BDD method. It would be advantageous if a single BDD structure could be used for both the qualitative and quantitative analyses. Therefore, a new method, the ternary decision diagram (TDD) method, is presented in this paper, where a fault tree is converted to a TDD that allows both qualitative and quantitative analyses and no additional BDDs are required. The efficiency of the four methods is compared using an example fault tree library.

1. Introduction

Fault trees were first developed in the 1960s and are commonly used for the qualitative and quantitative analyses of causes of system failure. The analysis of complex industrial systems may produce thousands of combinations of events (minimal cutsets/prime implicants) which can cause system failure. The determination of these failure combinations can be a time-consuming process even on modern high speed digital computers. If the fault tree has many failure modes, the determination of the exact top event probability also requires lengthy calculations. For many complex fault trees this requirement may be beyond the capability of the available computers.

Thus, approximation techniques have had to be introduced which resulted in loss of accuracy.

A more convenient form for the logic function from the mathematical viewpoint is that of a Binary Decision Diagram [1]. It overcomes some disadvantages of conventional FTA techniques enabling efficient and exact qualitative and quantitative analyses of both coherent and non-coherent fault trees. The BDD method is efficient for quantifying the system failure likelihood since it does not need the system failure modes as an intermediate step. It is also more accurate as there is no need for the approximations used in the traditional approach of kinetic tree theory [2]. The efficiency and the accuracy of the BDD method is discussed in [3,4].

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. The resulting SFBDD can be used in the quantitative analysis, calculating the top event probability or frequency.

An SFBDD does not provide all the information for the qualitative analysis, since not all the implicant sets of a non-coherent fault tree are necessarily prime and the list may not be complete. This means that it requires more than just a minimisation, as in the coherent case, to produce a full list of prime implicants. The full list can be obtained by applying the consensus theorem [5] to pairs of prime implicant sets involving a normal and negated literal. The first approach to this was presented by Courdet and Madre [6] and further developed by Dutuit and Rauzy [7] where a meta-products BDD is formed. Every basic event is represented by two variables, P_x and S_x, where the first of them represents the relevancy of the component (relevant or irrelevant) and the second variable represents the type of the relevancy (failure relevant or repair relevant). Since a meta-products BDD is in its minimal form, traversing its branches gives all the prime implicant sets.

The second alternative method presented by Rauzy [9] uses the system SFBDD and converts it to the zero-suppressed BDD (ZBDD), presented by Minato [10]. The 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. The resulting ZBDD is in its minimal form and traversing its paths from the root vertex to terminal vertex 1 results in all prime implicant sets.

The third alternative method investigated produces a labelled binary decision diagram (L-BDD), presented by Contini in [11] where every basic event is labelled according to its type. The additional information about the occurrence of every basic event is considered at an early stage of the algorithm, i.e. while converting a fault tree to an L-BDD. An L-BDD has two branches, therefore, it does not provide all prime implicant sets. The L-BDD obtained from the fault tree is simplified and then the determination of prime implicant sets is performed where the rules of the calculation depend on the type of a basic event. The L-BDD resulting is minimised.

The new alternative method for performing the qualitative analysis for non-coherent fault trees is presented in this paper where a fault tree is converted to a ternary decision diagram (TDD). The main concept of a TDD was presented by Sasao [8], which is expanded into an implementation methodology for fault tree analysis in this

paper. A TDD has three branches leaving any variable: the 1 branch represents the failure relevance of the component, the 0 branch represents the repair relevance of the component (so far this is a conventional BDD presentation) and the consensus branch represents the irrelevance of the component. A TDD encodes all the prime implicant sets, since while calculating the consensus branch for every node the consensus theorem is applied and all "hidden" prime implicant sets are obtained. However, the TDD can be non-minimal, therefore, the minimisation process needs to be performed removing the non-minimal paths from the 1 and 0 branches. Then the prime implicant sets can be obtained. Also, the obtained TDD can be used for the quantitative analysis as well as the qualitative analysis, therefore, no additional BDDs are required.

The four methods are described in the following sections and demonstrated throughout with the use of an example. Their efficiency is investigated using a library of large example fault trees.

2. Non-coherent fault trees

Fault tree structures can be classified according to their underlying logic. If during fault tree construction the failure logic is restricted to the use of the AND gate and the OR gate, the resulting fault tree is called coherent. If NOT logic is used or directly implied (by for example the use of exclusive OR, XOR, gates) the resulting fault tree can be non-coherent.

Define each component in the system according to an indicator x_i to show its status:

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

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

The fault tree diagram represents a logic structure which can be expressed by a structure function ϕ:

$$\phi = \begin{cases} 1 \text{ if systemis failed,} \\ 0 \text{ if systemis working.} \end{cases} \tag{2}$$

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

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

- Each component i must be relevant, i.e.

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

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

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

where

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

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

The second of these conditions means that as the component deteriorates (x fails, so $x_i = 1$) the system condition either does not change or also deteriorates. If the system is non-coherent for component i then for some state of the remaining components the system is failed then component i works and then when i fails the system is restored to the functioning (non-failed) condition. For example, system failure might occur due to the recovery of a failed component. Alternatively, during system failure, the failure of an additional component may bring the system to a good state. The fault tree is coherent if the NOT logic can be eliminated from the fault tree.

Consider an example of the leak detection system shown in Figure 1.

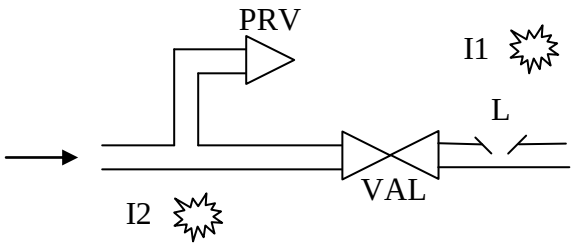

Figure 1. Leak protection system

A very high pressure gas is flowing as part of a gas transport system. If a leak (L) occurs on the section after the isolation valve (VAL), a gas detection system closes the valve. This stops a build up of the gas concentration in a location where an ignition source (I1) is possible. If the valve works and isolation occurs the high pressure gas can cause a rupture of the pipe and a second leak prior to the isolation valve. There is a permanent ignition source (I2) in this area, therefore, the problem can be avoided by a pressure relief valve (PRV) diverting the gas flow.

An explosion can occur in two ways:

- Gas is released prior to isolation valve, when the leak is detected, the isolation valve works but the pressure relief valve fails.
- Gas is released after isolation valve, when the leak is detected, the isolation valve fails and the ignition source occurs.

A fault tree representing causes of an ignition following a gas release is shown in Figure 2.

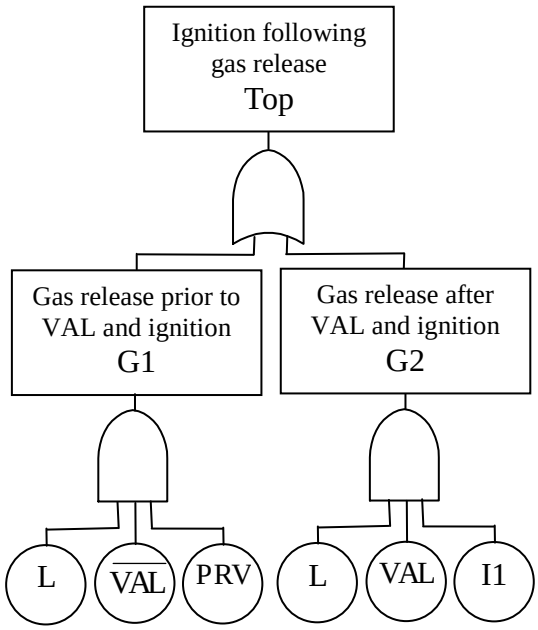

Figure 2. Explosion fault tree

Working in a bottom-up way the following logic expression is obtained.

$$\text{Top} = \text{L} \cdot \overline{\text{VAL}} \cdot \text{PRV} + \text{L} \cdot \text{VAL} \cdot \text{I1},$$

where "+" is OR, "·" is AND.

Therefore, $\left\{\text{L}, \overline{\text{VAL}}, \text{PRV}\right\}$ and $\left\{\text{L}, \text{VAL}, \text{I1}\right\}$ are prime implicants, as combinations of component conditions (working or failed) which are necessary and sufficient to cause system failure.

Also, unlike the reduction of the logic expression for a coherent fault tree the logic Equation for the top event in the non-coherent case does not produce a complete list of all prime implicants. In this example, there is one more failure mode:

$$\{\text{L}, \text{PRV}, \text{I1}\},$$

i.e. if a leak occurs and the pressure relief valve has failed and an ignition source is present on the section after the isolation valve it does not matter if the isolation valve closes or not, an ignition of the gas release will occur.

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

$$\text{Top} = \text{L} \cdot \overline{\text{VAL}} \cdot \text{PRV} + \text{L} \cdot \text{VAL} \cdot \text{I1} + \text{L} \cdot \text{PRV} \cdot \text{I1},$$

which can be obtained by applying the consensus law:

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

There are usually considerably more prime implicant sets than there are minimal cut sets for a coherent representation. Also the prime implicant sets tend to be a larger order (number of elements in a set) than the minimal cut sets. The increase in length of the logic Equation can cause difficulties for large fault trees. A coherent approximation can be obtained by assuming that all working states are TRUE, i.e. the probability of a component working is very close to 1. In the example shown in Figure 2 there are two minimal cut sets, $\left\{\text{L}, \text{PRV}\right\}$ and $\left\{\text{L}, \text{VAL}, \text{I1}\right\}$:

$$\begin{aligned} \text{Top} &= L \cdot \overline{VAL} \cdot PRV + L \cdot VAL \cdot I1 + L \cdot PRV \cdot I1 \\ &\approx L \cdot PRV + L \cdot VAL \cdot I1 + L \cdot PRV \cdot I1 \\ &= L \cdot PRV + L \cdot VAL \cdot I1. \end{aligned}$$

3. Conversion to BDDs

Binary Decision Diagrams can be used for the accurate quantification of fault trees. They do not need to evaluate the prime implicant sets as an intermediate step and do not need approximations. These advantages are critical when analysing non-coherent fault trees and overcome the difficulties encountered with conventional fault tree theory. To utilise the method, the fault tree is converted to a binary decision diagram, representing the structure function SFBDD.

Before the fault tree is converted to a BDD, the fault tree is manipulated so that the NOT logic is "pushed" down the fault tree until it is applied to 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 vertex in a SFBDD has an **ite** (if-then-else) structure. To illustrate this consider the **ite** structure **ite**(x, f_1, f_0), which means that if x fails then consider function f_1, else consider function f_0. Also, f_1 lies on the 1 branch of x and f_0 lies on the 0 branch.

Constructing the SFBDD from a fault tree initially requires the basic events in the fault tree to be given an ordering. SFBDD construction then moves through the fault tree in a bottom-up manner.

- Each basic event is assigned an **ite** structure

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

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

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

- Dealing with gate inputs:

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 $\oplus$

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

For small examples the variable ordering is largely irrelevant. Variable ordering schemes are discussed in [12,13]. Consider the variable ordering scheme L < VAL < PRV < I1. Applying the conversion rules (9)-(11) to the fault tree in Figure 2 results in a SFBDD presented in Figure 3.

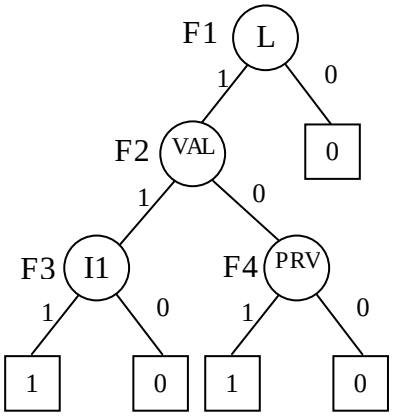

Figure 3. SFBDD for the fault tree in Figure 2

4. Calculation of prime implicant sets

In addition to quantification of the top event, knowledge of prime implicant sets can be valuable in gaining an understanding of the system. It can help to develop a repair schedule for failed components if a system cannot be taken off line for repair. The SFBDD which encodes the structure function cannot be used directly to produce the complete list of prime implicant sets of a non-coherent fault tree.

For example, consider a general component x in a non-coherent system. In a prime implicant a 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.

4.1. Ternary Decision Diagram method

A method to use a ternary decision diagram (TDD) in order to compute the prime implicant sets of a non-coherent fault tree structure 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.

4.1.1. Conversion of fault trees to TDDs

The TDD representation features a node structure with three exit branches. A new **ifre** structure is presented which distinguishes not only between relevant and irrelevant components but also it 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

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

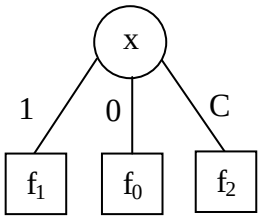

Figure 4. **ifre** structure

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" 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 x is **r**epair relevant

then consider f_0

else consider f_2

endif

Function f_2 represents prime implicant sets for which x is irrelevant, i.e. neither failure nor repair relevant combinations. The "C" branch is not obtained for all components. Therefore, this branch can be kept "empty" for components that are only failure or repair relevant but not both. If the conjunction of the two branches $f_1 \cdot f_0$ is not required, f_2 = NIL. Symbol NIL identifies cases when the "C" branch is not required and no Boolean operations that involve this branch are needed. When operating the new symbol in the Boolean algebra, it is defined that NIL $\oplus$ A= NIL.

The conversion process for computing the TDD from the non-coherent fault tree is an extension to the method used to develop the conventional BDD. Basic events of the fault tree must be ordered. Then the following process is presented:

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

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

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

If a is failure and repair relevant:

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

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

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

If $J = \textbf{ifre}(x, f_1, f_0, f_2)$ and $H = \textbf{ifre}(y, g_1, g_0, g_2)$, 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 product of the 1 and the 0 branch of every node and thus identifies the consensus of each node.

Consider the fault tree in Figure 2. Introducing the ordering of basic events L < VAL < PRV < I1 and applying the rules described in Equations (15)-(19) gives the TDD in Figure 5.

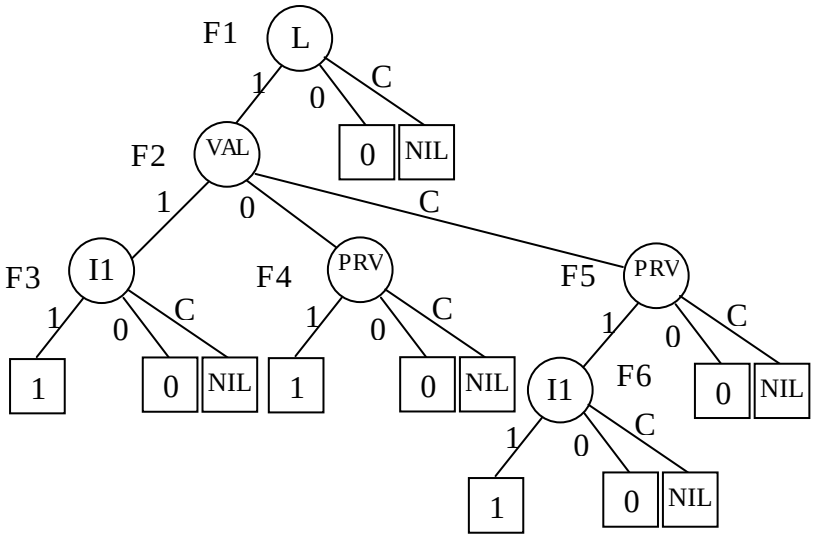

Figure 5. The TDD for the fault tree shown in Figure 2

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 F2 a new structure F5 is created. 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.

4.1.2. Minimisation of TDDs

Once the TDD has been computed there is no guarantee that the resulting structure will be minimal, i.e. produce the prime implicant sets exactly. In order to perform the qualitative analysis a minimisation procedure needs to be implemented.

The algorithm developed by Rauzy for minimising the BDD [1] can be extended to create a minimal consensus TDD that encodes only the prime implicant sets.

Consider a general node in the TDD which is represented by the function F, where

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

Since there are three types of variables in the TDD (failure relevant, repair relevant and both), the minimisation process is described for these three cases.

Minimal solutions of G, H and K are denoted as G_{min}, H_{min} and $K_{min.}$ Let δ be a minimal solution of G, which is not a minimal solution of K. Then the intersection of δ and x ($\delta \wedge x$) will be a minimal solution of F. Similarly, let γ be a minimal solution of H, which is not a minimal solution of K, and the intersection of γ and $\bar{x}$ ($\gamma \wedge \bar{x}$) will be a minimal solution of F.

Case 1 - x is failure relevant only

In Equation (20) K = NIL and the calculation of prime implicant sets is equivalent to the BDD case where the "C" branch does not exist, i.e.

$$sol_{min}(F) = (\delta \wedge x) \vee H_{min}. \tag{21}$$

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.

Case 2 - x is repair relevant only

Again, in Equation (20) K = NIL and the calculation of prime implicant sets is defined as:

$$sol_{min}(F) = (\gamma \wedge \bar{x}) \vee G_{min}. \tag{22}$$

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.

Case 3 - x is failure and repair relevant

The set of all the minimal solutions of F is given as:

$$sol_{min}(F) = (\delta \wedge x) \vee (\gamma \wedge \bar{x}) \vee K_{min}. \tag{23}$$

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.

4.1.3. Obtaining prime implicant sets from TDDs

The identification of prime implicant sets depends on the relevance of the node in the three cases, as it was given in the section above.

Case 1

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.

Case 2

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.

Case 3

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

Traversing the TDD in Figure 5 from the root vertex to terminal vertices 1 gives three prime implicant sets.

F1-F2-F3	{L, VAL, I1}
F1-F2-F4	{L, $\overline{\text{VAL}}$, PRV}
F1-F2-F5-F6	{L, PRV, I1}

The TDD method provides a good alternative technique for performing the qualitative analysis for non-coherent fault trees.

4.2. Established methods

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

4.2.1. 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.

4.2.1.1.Conversion of SFBDDs to meta-products BDDs

An alternative notation was developed in [6,7] that associates two variables with every component x. The first variable, P_x, denotes relevancy and the second variable, S_x, denotes the type of relevancy, i.e. failure or repair relevant. A meta-product, MP(π), is the intersection of all the system components according to their relevancy to the system state and π represents the prime implicant set encoded in meta-product MP(π).

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

The proposed algorithm by Dutuit and Rauzy [7] is used for calculating the meta-products BDD of a fault tree from the BDD. The meta-products BDD is always minimal, therefore it encodes the prime implicant sets exactly.

In order to present the algorithm, consider node F in a SFBDD, where F = **ite**(x, F1, F0). The meta-products BDD, that describes prime implicant sets using Equation (24), is expressed as:

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

where

$$P2 = PI(F1 \cdot F0), \quad (26)$$

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

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

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.

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

$$PI(F) = \textbf{ite}(P_{x_j}, 0, PI(F)). \quad (29)$$

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

If F is a terminal node, then

$$PI(0) = 0, \quad (30)$$

$$PI(1) = \textbf{ite}(P_{x_i}, 0, \textbf{ite}(P_{x_{i+1}}, 0, \ldots, \textbf{ite}(P_{x_n}, 0, 1))). \quad (31)$$

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

Each node in Figure 3 is considered in turn. Applying Equations (25)-(28) to node F1 gives:

$PI(F1) = \textbf{ite}(P_L, \textbf{ite}(S_L, PI(F2) \cdot 1, PI(0) \cdot 1), PI(F2 \cdot 0)) = \textbf{ite}(P_L, \textbf{ite}(S_L, PI(F2), 0), 0).$

Now consider node F2. Applying Equation (25) to node F2 gives:

$$PI(F2) = \textbf{ite}(P_{VAL}, \textbf{ite}(S_{VAL}, PI(F3) \cdot \overline{PI(F3 \cdot F4)}, PI(F4) \cdot \overline{PI(F3 \cdot F4)}), PI(F3 \cdot F4)).$$

First of all, $PI(F3 \cdot F4)$ is calculated using Equations (25)-(28):

$$PI(F3 \cdot F4) = \textbf{ite}(P_{PRV}, \textbf{ite}(S_{PRV}, \textbf{ite}(P_{I1}, \textbf{ite}(S_{I1}, 1, 0), 0), 0), 0).$$

Applying Equations (29)-(31) gives the two following expressions:

$$PI(F3) = \textbf{ite}(P_{PRV}, 0, \textbf{ite}(P_{I1}, \textbf{ite}(S_{I1}, 1, 0), 0)),$$

$$PI(F4) = \textbf{ite}(P_{PRV}, \textbf{ite}(S_{PRV}, \textbf{ite}(P_{I1}, 0, 1), 0), 0).$$

Negation $\overline{PI(F3 \cdot F4)}$ is calculated swapping terminal vertices 1 and 0.

$$\overline{PI(F3 \cdot F4)} = \textbf{ite}(P_{PRV}, \textbf{ite}(S_{PRV}, \textbf{ite}(P_{I1}, \textbf{ite}(S_{I1}, 0, 1), 1), 1), 1).$$

Since there are no repeated parts between PI(F3) and $\overline{PI(F3 \cdot F4)}$,

$$PI(F3) \cdot \overline{PI(F3 \cdot F4)} = PI(F3).$$

The same simplification is applied to PI(F4) and $\overline{PI(F3 \cdot F4)}$,

$$PI(F4) \cdot \overline{PI(F3 \cdot F4)} = PI(F4).$$

The resulting meta-products BDD for the BDD in Figure 3 is shown in Figure 6.

4.2.1.2.Obtaining prime implicant sets

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.

$P_L \wedge S_L \wedge P_{VAL} \wedge S_{VAL} \wedge \overline{P}_{PRV} \wedge P_{I1} \wedge S_{I1} =$	{L, VAL, I1}
$P_L \wedge S_L \wedge P_{VAL} \wedge \overline{S}_{VAL} \wedge P_{PRV} \wedge S_{PRV} \wedge \overline{P}_{I1} =$	$\{L, \overline{VAL}, PRV\}$
$P_L \wedge S_L \wedge \overline{P}_{VAL} \wedge P_{PRV} \wedge S_{PRV} \wedge P_{I1} \wedge S_{I1} =$	{L, PRV, I1}

For example, in the first meta-product P_L signifies that component L is relevant and S_L signifies that component L is failure relevant. The same with components VAL and I1. $\overline{P}_{PRV}$ means that component PRV is irrelevant. Hence the prime implicant set obtained is {L, VAL, I1}.

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.

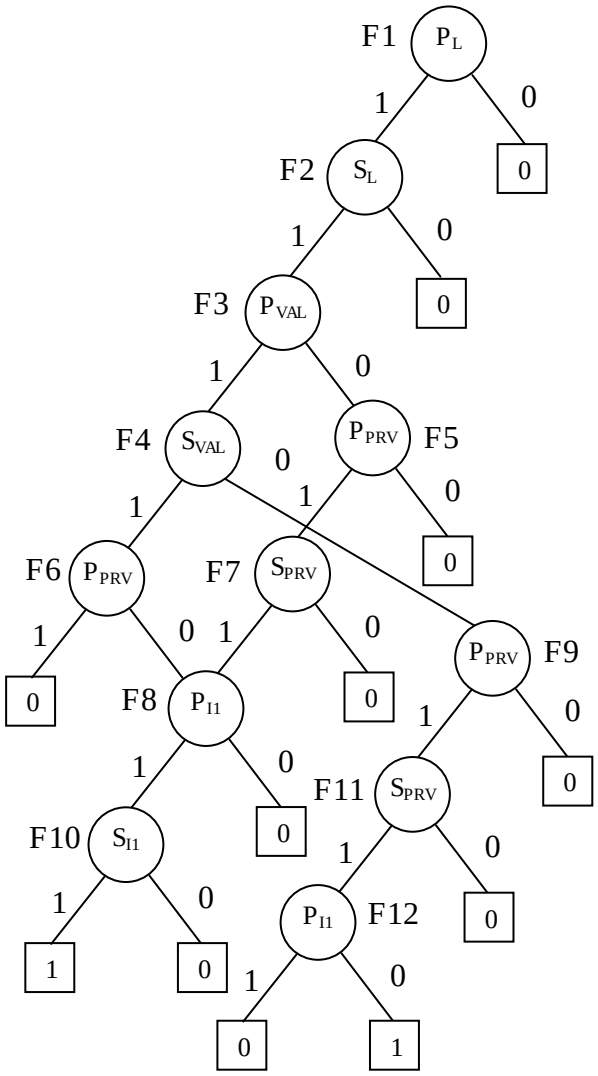

Figure 6. Meta-products BDD for calculating prime implicant sets

4.2.2. ZBDD method

An alternative method presented by Rauzy in [9] uses the idea of zero-suppressed BDDs (ZBDD) introduced by Minato in [10]. The 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.

4.2.2.1.Zero-Suppressed BDDs

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 following reduction rules for BDDs are applied:

- Eliminate all the nodes that have the 1 branch pointing to terminal vertex 0. Then connect the branch that was pointing to the eliminated node to the BDD structure beneath the 0 branch of the eliminated node as shown in Figure 7.
- Share all equivalent BDD structures as for original BDDs.

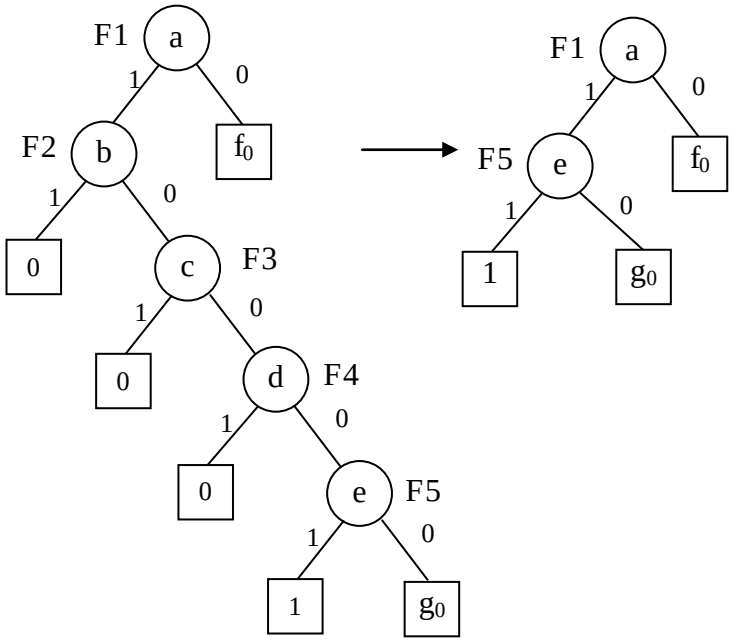

Figure 7. Elimination process

ZBDDs automatically suppress basic events that do not appear in prime implicant sets. It is very efficient when calculating sets with basic events that are far apart in the variable ordering scheme. For example, in Figure 7 the left-hand BDD contains a prime implicant set {a, e}. The established variable ordering is a<b<c<d<e. The ZBDD obtained by applying the reduction rules brings basic events close since the intermediate nodes (F2, F3, F4) can be eliminated.

4.2.2.2.Decomposition rule

The principle of this algorithm is to traverse the SFBDD that encodes structure function ϕ = **ite**(x, f_1, f_0) in a depth-first way and to build a ZBDD that encodes the prime implicant sets of ϕ in a bottom-up way.

The rule is divided in four cases:

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

$$PI(\phi) = xS_1 \cup \bar{x}S_0 \cup S_2, \tag{32}$$

where

$$S_2 = PI(f_1 \cdot f_0), \tag{33}$$

$$S_1 = PI(f_1) \setminus S_2, \tag{34}$$

$$S_0 = PI(f_0) \setminus S_2. \tag{35}$$

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

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

$$PI(\phi) = xS_1 \cup S_0, \tag{36}$$

where

$$S_0 = PI(f_0), \tag{37}$$

$$S_1 = PI(\, f_1 \,) \setminus S_0. \tag{38}$$

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

$$PI(\phi) = PI(\, f_1 + f_0 \,). \tag{39}$$

The resulting ZBDDs retain the variable ordering from the SFBDD. In addition, working states of basic events that appear in both states are incorporated in the ordering scheme, i.e. they appear after the basic event in its failed state in the ordering scheme.

4.2.2.3.ZBDD Example

Consider the SFBDD in Figure 3. Introduce the variable ordering for the ZBDD method L < VAL < $\overline{\text{VAL}}$ < PVR < I1. Applying the rules described in Equations (32)-(39) gives the ZBDD in Figure 8.

Each node is considered in the bottom-up way.
F3 = **ite**(I1, 1, 0), PI(F3) = **ite**(I1, 1, 0) - Both vertices are terminal.
F4 = **ite**(PRV, 1, 0), PI(F4) = **ite**(PRV, 1, 0) - Both vertices are terminal.

F2 = **ite**(VAL, F3, F4), PI(F2) = **ite**(VAL, S_1, **ite**($\overline{\text{VAL}}$, S_0, S_2)) - Basic event VAL appears in both failed and working states.
Here, following Equations (33)-(35) gives:
S_2 = PI(F2·F3) = PI(**ite**(PRV, **ite**(I1, 1, 0), 0)) = **ite**(PRV, **ite**(I1, 1, 0), 0) - Both basic events PRV and I1 appear only in their failed state.
S_1 = PI(F3) \ S_2 = **ite**(I1, 1, 0) - There are no repeated paths in F3 that are covered by S_2.
S_0 = PI(F4) \ S_2 = **ite**(PRV, 1, 0) - There are no repeated paths in F4 that are covered by S_2.

Therefore,
PI(F2) = **ite**(VAL, **ite**(I1, 1, 0), **ite**($\overline{\text{VAL}}$,**ite**(PRV, 1, 0), **ite**(PRV, **ite**(I1, 1, 0), 0))).

According to Equation (36)
F1 = **ite**(L, F2, 0), PI(F1) = **ite**(L, PI(F2), 0) - Basic event L appears only in its failed state.

After the last substitution we obtain:

PI(F1) = **ite**(L, **ite**(VAL, **ite**(I1, 1, 0), **ite**($\overline{\text{VAL}}$, **ite**(PRV, 1, 0),
ite(PRV, **ite**(I1, 1, 0), 0))), 0).

The ZBDD in Figure 8 is different form the SFBDD in Figure 3 because it has an additional variable $\overline{\text{VAL}}$ that indicates prime implicant sets that contain $\overline{\text{VAL}}$.

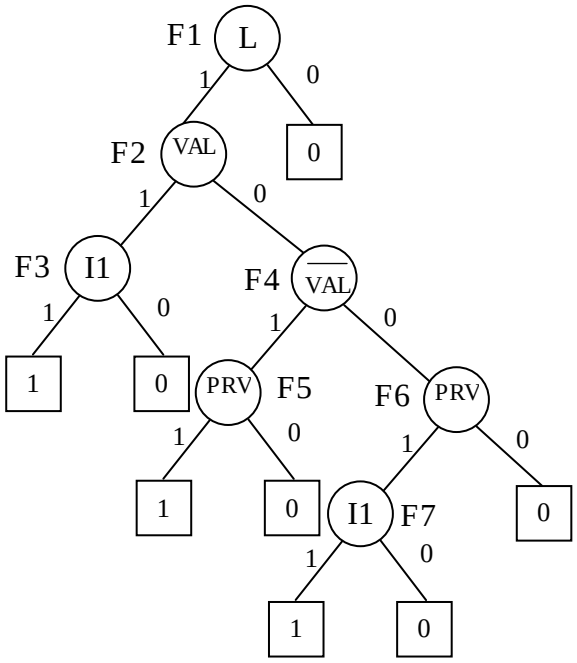

Figure 8. The ZBDD for the SFBDD in Figure 3

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

F1-F2-F3	{L, VAL, I1}
F1-F2-F4-F5	{L, $\overline{\text{VAL}}$, PRV}
F1-F2-F4-F6-F7	{L, PRV, I1}

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

4.2.3. Labelled variable method

The labelled variable method [11] provides another alternative method for constructing BDDs for non-coherent fault trees. BDDs constructed using this conversion approach consist of variables that are labelled according to their type. They are called labelled binary decision diagrams (L-BDDs).

The background for this method is that the current analysis algorithms consider the fault tree structure as a Boolean function in which all variables have the same properties, whereas the operations to be applied to determine the prime implicant sets are dependent on the variables' type. For example, variables that appear in their failure and success states can appear in combinations of events which lead to system failure in both states, and which are not covered by the SFBDD obtained. In the L-BDD method it is convenient to consider this additional information about the type of a variable during the conversion process itself. The aim of the L-BDD method is to construct and analyse a L-BDD in which all variables are labelled with their type.

4.2.3.1.Conversion of fault trees to 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 $\phi(x) = a \cdot b + \bar{a} \cdot \bar{c} + b \cdot \bar{c}$ 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. Basic events of the fault tree must be ordered. Then the following process is performed:

- By the application of De Morgan's laws push any NOT gates down through the fault tree until it reaches a basic event level.
- Each basic event is assigned an **ite** structure as in Equation (9) or Equation (10).
- Traversing the fault tree in a bottom-up manner and considering gates whose inputs have been expressed in an **ite** format using Equation (11).

The last rule Equation (11) of this algorithm is extended presenting some additional rules that incorporate labelled variables.

- Considering the ordering &x < x < $x implements the following Equations:

$$\begin{aligned} &\text{If } J = \textbf{ite}(x, f_1, f_0) \text{ and } H = \textbf{ite}(\$x, g_1, g_0), \\ &\text{then } J \oplus H = \textbf{ite}(\&x, f_1 \oplus g_0, f_0 \oplus g_1) \end{aligned} \tag{40}$$

$$\begin{aligned} &\text{If } J = \textbf{ite}(\&x, f_1, f_0) \text{ and } H = \textbf{ite}(x, g_1, g_0), \\ &\text{then } J \oplus H = \textbf{ite}(\&x, f_1 \oplus g_1, f_0 \oplus g_0) \end{aligned} \tag{41}$$

$$\begin{aligned} &\text{If } J = \textbf{ite}(\&x, f_1, f_0) \text{ and } H = \textbf{ite}(\$x, g_1, g_0), \\ &\text{then } J \oplus H = \textbf{ite}(\&x, f_1 \oplus g_0, f_0 \oplus g_1) \end{aligned} \tag{42}$$

Applying the conversion rules given in Equations (9)-(11) and Equations (40)-(42) to the fault tree in Figure 2 results in a L-BDD presented in Figure 9. The variable ordering is L < VAL < PRV < I1.

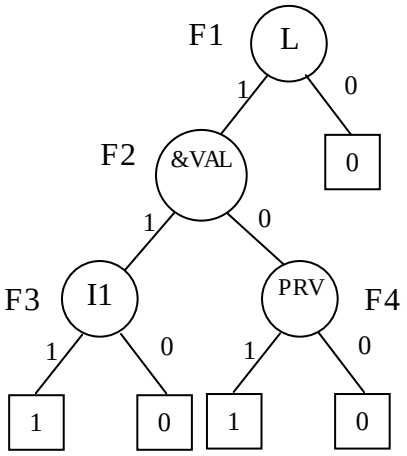

Figure 9. BDD with labelled variables for the fault tree in Figure 2

The resulting L-BDD is equivalent to its SFBDD in Figure 3, if DF variable "&VAL" is replaced by variable VAL. 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.

4.2.3.2.Simplification of L-BDD

The most complex rules of the qualitative analysis of the L-BDD are applied to nodes with DF variables. Therefore, it is convenient to reduce the number of DF variables before calculating prime implicant sets. The rules are applied on the parts of the L-BDD that have a terminal vertex. Simplification rules are shown in Figure 10.

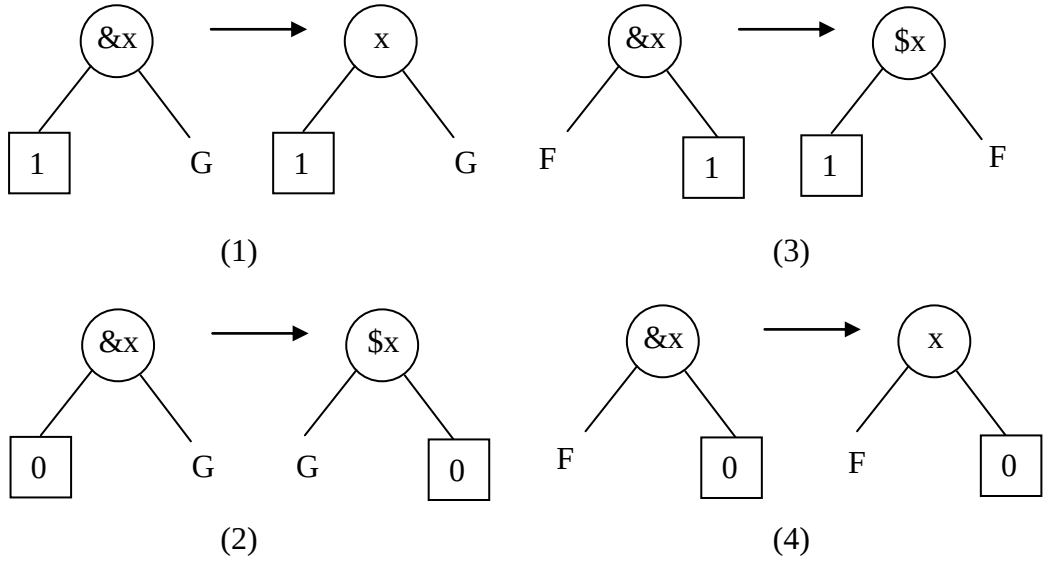

Figure 10. Simplification rules for the L-BDD

The example L-BDD in Figure 9 is in its simplest form, therefore the simplification rules are not applied.

4.2.3.3.Obtaining prime implicant sets from L-BDD

Let structure function ϕ be:

$$\phi = x_i \cdot f_1 + \overline{x_i} \cdot f_0 \tag{43}$$

where the two residues are:

$$f_1 = \phi(x_1,...,1,..., x_n) \tag{44}$$

$$f_0 = \phi(x_1,...,0,..., x_n) \tag{45}$$

Let $PI(\phi)$ be the prime implicants of ϕ. Visiting the L-BDD in the bottom-up way the procedure to be applied to the node x_i to determine the prime implicants is as follows:

- If x_i has label "&" then

$$PI(\phi) = x_i \cdot S_1 + \$x_i \cdot S_0 + S_2 \tag{46}$$

where

$$S_2 = f_1 \wedge f_0, S_1 = f_1 \setminus S_2, S_0 = f_0 \setminus S_2. \quad (47)$$

- Else

$$PI(\phi) = \alpha \cdot S_1 + f_0 \quad (48)$$

where

$$\alpha = x_i \text{ or } \$x_i, S_1 = f_1 \setminus f_0 \quad (49)$$

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

For example, $f_1 \setminus f_0$ gives the L-BDD of f_1 in which the combinations of f_1 repeated in f_0 are removed.

The heaviest operation is the intersection $S_2 = f_1 \wedge f_0$ that, however, is applied only when dealing with "&" type variables. It calculates the "hidden" prime implicant sets. If there are no "&" type variables, the L-BDD contains all prime implicant sets.

The rules for writing the prime implicant sets from an L-BDD are straightforward. On the path from the root to any terminal node:

- For variables x, "$x" do not consider the negated part,
- For variables "&x" consider the right branch as $\bar{x}$.

Consider the L-BDD shown in Figure 9. Prime implicant sets can be calculated using Equations (46)-(49).

Since the variable of node F2 is a DF variable, i.e. it appears in the fault tree in both success and failure states, the intersection of its 1 and 0 branches is calculated in order to get all prime implicant sets. This process is similar to the calculation of the "C" branch in the TDD method.

P = F3 · F4 = **ite**(I1, 1 , 0) · **ite**(PRV, 1 , 0) = **ite**(PRV, **ite**(I1, 1 , 0), 0),

PI(P) = PRV · S_1 + f_0, here S_1 = I1, f_0 = 0,

PI(P) = PRV · I1.

Since there are no repeated paths between the 1 branch and the structure P and none between the 0 branch and the structure P, prime implicant sets of F2 are as follows:

PI(F2) = VAL · I1 + $\overline{\text{VAL}}$ · PRV + PRV · I1.

Finally, the last node F1 is considered:

PI(F1) = L · S_1 + f_0, here S_1 = PI(F2), f_0 = 0,

PI(F1) = L · (VAL · I1 + $\overline{\text{VAL}}$ · PRV + PRV · I1).

The three prime implicant sets are obtained:

{L, VAL, I1}, {L, $\overline{\text{VAL}}$, PRV}, {L, PRV, I1}.

The L-BDD method provides an efficient technique that allows to calculate prime implicant sets using the prior information about the type of every variable.

5. Quantitative analysis of non-coherent

5.1. Quantification of a non-coherent system using the TDD

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.

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 (13), 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 to Equation 35 gives:

$$\phi(\mathbf{x}) = x\phi(1_i, \mathbf{x}) + \bar{x}\phi(0_i, \mathbf{x}) = x(f_1 + f_1 f_0) + \bar{x}(f_0 + f_1 f_0) = xf_1 + \bar{x}f_0. \quad (50)$$

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

$$Q_{SYS} = E(\phi(\mathbf{x})) = q_x Q(f_1) + (1 - q_x) Q(f_0). \quad (51)$$

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 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.

5.2. Quantitative analysis using other methods

Consider a node F in the SFBDD, F = **ite**(x, f_1, f_0). The structure function $\phi(\mathbf{x})$ is expressed as $\phi(\mathbf{x}) = xf_1 + \bar{x}f_0$, which is adequate to the expression in (50). Therefore, the probability of the top event, Q_{SYS}, is obtained as a sum of probabilities of the disjoint paths through the SFBDD. Quantitative analysis is performed in this way applying the meta-products BDD method and the ZBDD method, where the SFBDD is obtained after fault tree conversion prior to the qualitative analysis.

In the L-BDD method the quantitative analysis is adequate, i.e. the probability of the top event, Q_{SYS}, is the sum of the probabilities of the disjoint paths through the L-BDD. In this case each variable &x is treated as x, and $x – as $\bar{x}$.

6. Comparison of the four methods

The theoretical comparison of the four different techniques is provided in Table 1.

Method	Minimisation	Construction technique	Advantages	Disadvantages
Ternary Decision Diagram (TDD)	Required	Basic event x is coded by a node with three branches. The third branch is the conjunction of the 1 and the 0 branches and the conjunction is performed if needed	A clear representation of all implicant sets is provided.	1.Minimisation is required 2.TDD size is 3^n.
Meta-products BDD	Not required	Basic event x is coded by two variables P_x and S_x.	No minimisation is required.	1.Meta-products BDD size is 2^{2n} 2.It results in a time consuming process.
Zero-suppres-sed BDD (ZBDD)	Not required	Basic event x is coded by x and, in addition, by $\bar{x}$, if needed. The conjunction of the 1 and the 0 branches is performed only for dual state variables.	1.No minimisation is required 2.Compact representation of prime implicant sets is provided.	Dual state events are coded by two variables
Labelled BDD (L-BDD)	Required	Basic event x is labelled according to its occurrence, (&x<x<$ x). The conjunction of the 1 and the 0 branches is performed only for dual state variables.	Labels identify where the conjunction of the 1 and the 0 branches is required.	1.Minimisation is required 2.Labelling introduces some additional variables

Table 1. Comparison of the four techniques

Table 1 gives a short overview of all four construction techniques. It highlights the main advantages and disadvantages of the four approaches. The TDD method provides a clear and efficient representation of prime implicant sets when the "C" branch for a node is created if needed. Also, using the ZBDD a compact representation of prime implicant sets is obtained which does not require the minimisation process. The least favourable technique is the meta-products BDD method where two variables are assigned for every basic event. Due to this feature the size of the BDD and the processing time is increased unavoidably.

The efficiency of the four methods for calculating prime implicant sets was investigated using a benchmark of medium sized fault trees from industry. The

performance over 13 example fault trees was obtained, since each method may perform well on some fault trees dependent upon the fault tree structure. The complexity of the 13 fault trees is indicated in columns 2, 3 and 4 of Table 2, representing the number of gates, the number of events and the number of prime implicant sets in their solution.

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 5, 6, 7 and 8 respectively. The number of nodes in the TDD method describes the sum of the number of nodes in the TDD before the minimisation 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 number of nodes in the SFBDD and the number of nodes in the ZBDD is given. 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 and the number of nodes in the minimised L-BDD.

Similarly, the processing time in seconds covers the time taken to convert example fault trees to BDDs and perform the qualitative analysis. Results of processing time are shown in columns 9, 10, 11 and 12 of Table 2 for the four methods respectively.

Example	Number of gates	Number of basic events	Number of prime implicant sets	Number of nodes in TDD for the 1st (TDD) method	Number of nodes in BDD for the 2nd (MPPI) method	Number of nodes in ZBDD for the 3rd (ZBDD) method	Number of nodes in L-BDD for the 4th (L-BDD) method	Time taken for the 1st method	Time taken for the 2nd method	Time taken for the 3rd method	Time taken for the 4th method
1	31	88	41388	784	2207	580	2942	1.43	484.24	1.91	4.68
2	29	63	7433	984	1366	427	1231	0.20	10530.99	0.24	34.94
3	40	86	5447	560	2189	564	1704	0.18	526.2	0.22	21.85
4	32	83	4979	412	1854	496	1874	0.17	50.78	0.25	1.03
5	28	69	1186	1276	3356	869	2991	0.09	1590.85	0.10	26.10
6	26	50	1001	581	2692	596	2223	0.07	140.54	0.06	9.02
7	21	42	289	298	880	223	610	0.01	36.65	0.04	0.22
8	32	47	155	324	1038	309	1000	0.14	67.98	0.20	0.68
9	19	42	152	452	1309	493	732	0.04	158.84	0.05	0.48
10	34	57	71	258	816	284	708	0.02	60.07	0.04	1.00
11	25	43	43	737	877	293	913	0.04	784.89	0.05	0.74
12	41	74	35	396	741	214	264	0.08	15.05	0.21	0.50
13	36	49	24	412	613	283	685	0.14	474.99	0.20	0.53

Table 2. Calculation results for example fault trees

As it was expected from the theoretical comparison of the methods, the meta-products BDD algorithm produced a lot larger final BDDs than any other methods that were used for the calculation of prime implicant sets. Processing time was also greater. This was especially true for examples with a large number of prime implicant sets (example 2 and 5). In those situations the number of nodes and the processing time

could be decreased using any other method instead of the conventional meta-products BDD method.

The TDD method performed as well as the ZBDD method. Both methods were more efficient than the meta-products BDD method and the L-BDD method in terms of the number of nodes and the processing time. For all example fault trees the TDD method was faster than the ZBDD method. This is due to the efficient representation of prime implicant sets in the TDD method, when the third branch is created if needed. According to the number of nodes, for some examples the ZBDD method was better than the TDD method. This outcome is explained by a compact representation of prime implicant sets in the ZBDD method and no requirement of additional storage for a minimal BDD.

The second worst result was obtained using the L-BDD method. The weaknesses of the L-BDD method are the introduction of the three different types of basic events and the requirement for minimisation before obtaining prime implicant sets.

Overall, these results show that the TDD method is an efficient method to obtain prime implicant sets, when all prime implicant sets are obtained by performing the conjunction of the two branches of each node. Information about the failure and/or repair relevancy of the component is used to identify if the conjunction of the two branches is needed, which makes the process efficient. Finally, the same TDD that is used to represent prime implicant sets before the minimisation can be used for the quantitative analysis, no additional structure is required.

7. Discussion of Real-time BDD Analysis

Recent research into autonomous vehicles (such as UAVs) is using BDDs to calculate the probability of mission failure at the current point in time. At a point in time certain phases may be known to have completed successfully (i.e. NOT failed, therefore requiring a non-coherent analysis), component failures may exist (as indicated by a fault detection and identification tool) or other events may have occurred. By establishing the potential causes of mission failure (prime implicant sets) and the updated likelihood decisions are made regarding the future operation of the autonomous vehicle.

The fault tree to BDD conversion in this application is done off-line and so conversion time is important but not critical. Storing and processing the BDD on-board of the vehicle are the factors which are the more critical in determining which is the best method to use.

Since the Ternary Decision Diagram permits one representation for both qualitative and quantitative analyses and has a very good efficiency compared to the alternatives this representation is advantageous in this particular circumstance.

8. Conclusions

This paper presents procedures by which non-coherent fault trees can be examined and prime implicant sets obtained. 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 alternative 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 to calculate all prime implicants directly. Established methods - the conventional algorithm that produces a meta-products BDD, the zero-suppressed BDD method and the labelled BDD method are presented and their efficiency is compared using some example fault trees. Efficiency analysis indicates that the new proposed TDD method is the fastest method and it provides an efficient representation of prime implicant sets. Another advantage of the TDD method is that it is suitable for both qualitative and quantitative analyses of non-coherent fault trees, whereas in the existing techniques two separate BDDs need to be constructed if the full analysis is required.

9. References

1. Rauzy, A. New Algorithms for Fault Tree Analysis, Reliability Engineering and System Safety, **40**, pp203-211 (1993)
2. Vesely, W.E. A Time Dependent Methodology for Fault Tree Evaluation, Nuclear Design and Engineering, **13**, pp337-360, (1970)
3. Sinnamon, R.M. and Andrews, J.D. Improved efficiency in qualitative fault tree analysis, Quality and Reliability Engineering International, **13**, pp293-298 (1997)
4. Sinnamon, R.M. and Andrews, J.D. Improved accuracy in quantitative fault tree analysis, Quality and Reliability Engineering International, **13**, pp285-292 (1997)
5. Andrews, J.D. To Not or Not to Not!!, Proceedings of the 18th International System Safety Conference, pp267-274 (2000)
6. Courdet, O. and Madre, J.-C. A New Method to Compute Prime and Essential Prime Implicants of Boolean Functions, Advanced Research in VLSI and Parallel Systems, pp113-128 (1992)
7. Dutuit, Y. and Rauzy, A. Exact and Truncated Computations of Prime Implicants of Coherent and Non-Coherent Fault Trees with Aralia, Reliability Engineering and System Safety, **58**, pp225-235 (1997)
8. 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)
9. Rauzy, A. Mathematical Foundation of Minimal Cutsets, IEEE Transactions on Reliability, volume 50, **4**, pp389-396 (2001)
10. Minato, S. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems, Proceedings of the 30th ACM/IEEE Design Automation Conference, DAC'93, pp272-277 (1993)
11. Contini, S. Binary Decision Diagrams with Labelled Variables for Non-Coherent Fault Tree Analysis, (European Commission, Joint Research Centre) 2005
12. Bartlett, L.M. and Andrews, J.D. Comparison of Two New Approaches to Variable Ordering for Binary Decision Diagrams, Quality and Reliability Engineering International, **17**, pp151-158 (2001)
13. Bartlett, L.M. and Andrews, J.D. Selecting am ordering heuristic for the fault tree binary decision diagram conversion process using neural networks, IEEE Trans Reliab, **(51) 3**, pp344-349 (2002)