Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Non-coherent Modelling in Compositional Fault Tree Analysis

Septavera Sharvia, Yiannis Papadopoulos

Department of Computer Science, University of Hull, UK
{s.sharvia, y.i.papadopoulos} @ dcs.hull.ac.uk

Abstract: The inclusion of NOT gates in a fault tree creates a "non-coherent" structure in which not only the failure of a component but also the negation of failure, i.e. the working state of the component, can contribute to the undesirable effects on a system. This type of non-coherent modelling remains controversial; its usefulness is still debated among academics, which explains why NOT gates have not been included in the *Fault Tree Handbook*. In this paper, we review work on non-coherent fault trees and highlight circumstances where non-coherent modelling is appropriate and useful. We then describe an extension to HiP-HOPS (Hierarchically Performed Hazard Origin and Propagation Studies), a recently proposed compositional safety analysis method, that enables model-based synthesis and analysis of non-coherent fault trees. A small example is given to illustrate application of the extended method and demonstrate how this type of non-coherent modelling can give a more precise and ultimately more correct insight into failure behaviour. *Copyright © 2008 IFAC.*

1. INTRODUCTION AND BACKGROUND

Fault Tree Analysis (FTA) is undoubtedly a useful safety analysis technique. However, in industrial practice, the construction of fault trees remains a manual process which in the context of large and complex systems becomes laborious, expensive and error prone. Recently, a number of compositional FTA techniques have emerged in which fault trees are automatically produced from system models that contain information about component failures and their effects. Techniques include HiP-HOPS (Papadopoulos, et al, 2001), Components Fault Trees (CFT) (Kaiser et al, 2003), and State-Event Fault Trees (SEFT) (Grunske et al, 2005). The role of safety analysts in compositional FTA is redefined as people are only required to provide the local failure behaviour of components, while an automated tool takes that information and determines how local failures propagate through a system model to cause system-level effects. This approach simplifies the assessment, and makes it possible to perform multiple iterations of safety analysis, for example to examine the effects of design modifications on system safety.

This new work on compositional FTA has so far been limited to the synthesis and analysis of fault trees with coherent structure, i.e. fault trees in which the failure logic is composed of classical Boolean AND and OR gates. While recent work has looked into extending this concept to support state-based (Kaiser et al, 2004) and temporal safety analysis (Walker and Papadopoulos, 2006) (Merle and Roussel, 2007) the potential for synthesis and analysis of non-coherent fault trees, i.e. fault trees which also contain NOT gates, has not yet been explored. A possible explanation for this is that the inclusion of NOT gates in fault trees implies that both the failed and working state of components can contribute to the failure of the system, a feature that traditionally has been seen as an indication of poor design. In such circumstances, the prevalent advice has been to revise the design rather than resort to the need for non-coherent modelling. Beyond conceptual objections to the use of NOT gates, it has also been argued that the inclusion of NOT logic increases the complexity of both qualitative and quantitative analysis of the fault tree while providing little additional information.

Despite those strong objections, researchers have begun to recognize the importance of incorporating NOT logic in fault trees and a body of work has emerged in support of non-coherent modelling. In section 2, we review arguments for and against non-coherent fault trees highlighting cases where the inclusion of NOT gates benefits failure modelling. In section 3, we present an extension to HiP-HOPS that enables compositional synthesis and analysis of non-coherent fault trees. We discuss the type of non-coherent modelling enabled within HiP-HOPS as well as state-of the art techniques for analysis of non-coherent fault trees that have underpinned the tool extension. In section 4, we give an example of application of this approach and demonstrate how compositional non-coherent failure modelling can help analysts improve their insight into the behaviour of a system. Finally in section 5 we draw conclusions.

2. NON-COHERENT MODELLING

The traditional view that NOT gates are not required in FTA is largely based on the notion that the probability of a negation of a failure event is always close to one which means that such conditions can be safely ignored in quantitative FTA. This view is challenged in (Johnston and Matthews, 1983) where circumstances are presented in which the assumption $P(\neg A) \approx 1$, *where* A *= failure event,* does not always hold. It is clear, for example, that this assumption does not hold in conditions where the failure probability of a component becomes significant enough and this is often the case in conditions that exceed the operating specifications of a component. In stormy weather, for instance, the probability of an electrical power supply working is substantially lower than what would be required for that condition to be ignored in quantitative FTA. In such cases, the omission of NOT gates results in incorrect quantitative evaluations.

(Andrews, 2000) argues that the use of NOT gates is especially important in the failure modelling of multitasking systems. He discusses an example of such a system where a coherent simplification of a fault tree could result in the generation of misleading "false" *minimal cut sets*[1]. Andrews shows that these "false" cut sets could be avoided with the inclusion of NOT gates, which in this case are needed for correct modelling.

Another argument against non-coherent modelling is that if we allow the negation of component failures to cause system failure, then we must also accept that the occurrence of component failures can somehow prevent system failure. The Fault Tree Handbook *(*Vesley, 1981*)* takes the view that situations where component failures miraculously help prevent system failure should not be included in fault trees as they are very unlikely. However, Johnston and Matthews (1983*)* show that this type of inclusion could be beneficial in helping designers to identify potential preventive measures. Take for example the following two conditions:

$$\text{OilLeakage} = \text{PipeImpaired} \qquad (1)$$

$$\text{OilLeakage} = \text{PipeImpaired} \,.\, \neg\, \text{FailureToSupplyOil} \qquad (2)$$

Compared to its coherent simplification in (1), the non-coherent representation in (2) helps a designer to identify that, in circumstances where a pipe is damaged, shutting off the oil supply to the pipe could prevent oil leakage. The system could then be enhanced with an automatic mechanism that shuts down the oil supply when an impaired pipe is detected.

The need for negative failure logic also arises in circumstances where an XOR gate is required, e.g. to indicate that the co-existence of two failures is prohibited by system boundaries. An XOR gate by definition implies a NOT gate (because $A \oplus B = A \,.\, \neg\, B + B \,.\, \neg\, A$), and therefore indirectly leads to non-coherent modelling.

Another case for non-coherent modelling is that of a phased mission system, or a system that goes through a sequence of operational modes. In such a system, the success of a given phase clearly depends on the success of the precursor phases. Therefore the failure of a system (S) in a phase (A) can only occur if failure (P) have been avoided in all precursor phases (i.e. $S=A. \neg P$).

Beeson *(*2002*)* shows that the prime implicant sets generated from non-coherent fault trees can help to develop a repair schedule for failed components if a system cannot be taken off line for repair. For example, suppose that prime implicant set $A.B.\neg C$ causes a catastrophic system failure. The set shows that, in circumstances where A, B and C have failed, C should be the last to be repaired in order to avoid system failure. Note that the set also identifies potential design improvements. It shows, for example, that additional measures can be incorporated to prevent system failure, by forcing failure of component C when A and B have failed. This can be generalised as a particular type of hazard mitigation strategy, where the act of forcing an additional component failure makes it possible to prevent a system failure, a condition known as achieving an "Island of Success" (Johnston and Matthews, 1983). For example, if we assume that HAZARD is caused by $\neg A.B$, then the condition A.B represents an island of success that can be achieved by forcing failure of A in conditions of failure of B. This knowledge can only be gained via non-coherent modelling, and is useful, because it suggests a (temporary at least) mitigation measure against HAZARD. In the absence of this in coherent modelling, a designer must simply assume that failure of B by itself is sufficient to cause system failure, and that nothing can be done in those circumstances.

Finally, there is a common misconception that the INHIBIT gate can be used as a substitute of NOT. The INHIBIT gate propagates an input failure (I) to its output only in the presence of condition (C) and effectively yields a logical result which is equivalent to [I.C]. Such a gate could indeed be used in situations where C represents the working state or negation of failure (F) of a component or subsystem. However, in such cases the logical result of an INHIBIT gate is simply equivalent to $(I.\neg F)$. Clearly, the problem remains: any logical processing of such gates would still require non-coherent modelling and calculation of prime implicants.

3. SYNTHESIS AND ANALYSIS OF NON-COHERENT FAULT TREES

As the value of non-coherent failure modelling is increasingly understood, it becomes important to enable this type of modelling in contemporary developments in safety analysis, such as the compositional FTA techniques which have emerged since the late 90s.

HiP-HOPS is one of the first proposals for compositional FTA which introduces a degree of automation and reuse in safety analysis to address problems arising from the increasing complexity of systems. In HiP-HOPS, the topology of a system together with reusable local failure specifications at component level are used to automatically produce a set of fault trees and an FMEA (Failure Modes and Effects Analysis) for the system. The technique is supported by an automated tool which currently works in conjunction with modelling tools like Matlab Simulink and ITI's Simulation X – but can also be interfaced to other modelling packages.

To achieve synthesis of system level safety analyses, HiP-HOPS defines a language for the description of failure behaviour at component level. Using this language, the failure behaviour of a component can be specified as a list of

[1] A minimal cut set of a fault tree (or "*prime implicant"* for a non-coherent fault tree) is the smallest combination of *basic events* that results in system failure represented as top event in the tree. A basic event is a leaf node, which represents a component failure or the negation of failure (in prime implicants), i.e. a component in working state.

declarations of internal failure modes of the component (*internal malfunctions*) and a list of declarations of deviations of parameters as they can be observed at component outputs *(output deviations).* Each internal malfunction is optionally accompanied by quantitative data, e.g. a failure and a repair rate if these are known. For each output deviation an expression is given to describe causes as a logical combination of internal malfunctions of the component and similar deviations of parameters at component inputs (*input deviations*). Analysts decide which input and output deviations should be examined during the analysis. Although these deviations will tend to vary across applications, they would typically represent conditions such as the "omission" or "commission"[2] of an input or output parameter and conditions such as the parameter being delivered at a higher or lower value ("value failures") or earlier or later than expected ("timing failures").

This specification of component failures and their local effects can be stored in a library and is reusable for other components of the same type. For the purposes of the work describe here, the grammar of this specification has been extended to include a NOT operator, which makes it possible to perform non-coherent failure modelling at component level. Negation can be applied either to internal malfunctions or input deviations to suggest the complement of those conditions in expressions that define the causes of output deviations.

Using these new capabilities it is possible to express the working state of a component as a negation of the disjunction of all its failure modes. E.g. if a component has two failure modes FM1 and FM2, its working state can be represented as $\neg$ (FM1 + FM2), or with the equivalent expression $\neg$ FM1. $\neg$ FM2. Similarly, if a component has a single failure mode FM and handles two inputs IN1 and IN2 which we know that by design cannot fail simultaneously, then it is possible to specify that an omission of component output O1 is caused by FM or an omission of exactly one input, i.e. that

O-O1 = FM + O-IN1. $\neg$ O-IN2 + O-IN2. $\neg$ O-IN1

where "O-"stands for "Omission of".

Once these local analyses (coherent or not) have been inserted into the model, the structure of the model is then used to automatically determine how local conditions specified in the analyses propagate through connections in the model and cause functional failures at the outputs of the system. The global view of failure in the system is captured in a set of non-coherent - if necessary - fault trees which are automatically constructed by traversing the model of the system backwards moving from the final elements of the design, i.e. the actuators, towards system inputs and by evaluating the failure expressions of the components encountered during this traversal.

[2] An "omission" of output is a condition in which output is not provided when there should be output according to the design intention, while a "commission" of an output is a condition is which output is generated inadvertently.

To enable synthesis of non-coherent fault trees, the original fault tree synthesis algorithm was extended with the ability to handle not only conditions that represent failures but also conditions that represent the negation of such failures. The fault trees synthesized using this approach show how functional failures or malfunctions at the outputs of the system are caused by logical combinations of component failures or component working states. These fault trees may share branches and basic events in which case they record dependencies in the model and common causes of failure, i.e. component failures that contribute to more than one system failures. Thus, in general, the result of the fault tree synthesis process is a network of interconnected fault trees which record logical relationships between component states and system failures.

In HiP-HOPS, the fault tree synthesis is followed by qualitative analysis of the fault trees in which each fault tree is logically reduced to sets of prime implicants. The objective of this analysis is to remove the intermediate logic that connects leaf nodes of the trees (basic events) to top events in order to establish the direct contribution of component states represented as basic events to the system failures represented as top events. To achieve this, HiP-HOPS was extended with an algorithm for qualitative analysis of non-coherent fault trees.

3. 1. Analysis of Non-coherent Fault Trees

NOT gates can be located anywhere in the synthesized fault trees: they can directly complement leaf nodes representing component failure modes or intermediate events representing deviations of control parameters; in the latter case each deviation is underpinned by a branch recording the causes of the deviation further upstream in the model as captured by the fault tree synthesis algorithm. Before a fault tree can be analyzed, NOT gates need to be "pushed" down the tree to complement only component failure modes. This is achieved via, recursive if necessary, application of De Morgan's law, as illustrated in Figure 1 below:

De Morgan's law: $\neg$ (A.B) = $\neg$ A + $\neg$ B

$\neg$ (A + B) = $\neg$ A . $\neg$ B

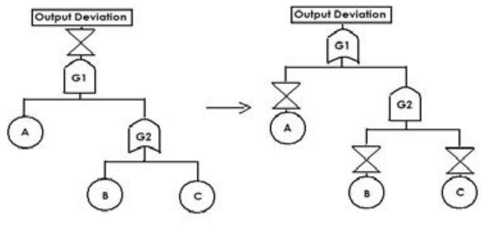

Figure 1: Pushing NOT gates down

When the above is achieved, the fault tree is ready for further analysis and generation of prime implicant sets. Perhaps the simplest algorithm for qualitative analysis of non-coherent trees is a modification of a minimal cut sets algorithm known as the "the simple prime implicant set algorithm" which is described in (Worrel, 1981)). Although the algorithm does not always produce complete prime implicant sets, the problem can be corrected and "hidden"[3] sets can be identified via application of the *Consensus* method as described in (Quinne, 1955). Other algorithms for prime implicant generation include the Set Equation Transformation System (SETS) (Worrell and Stack, 1978), a top-down algorithm employing dual fault trees proposed by (Kumamoto and Henley, 1978) and the Binary Decision Diagram (BDD) algorithm (Rauzy, 1993). The use of BDDs in particular is one of the latest advancement in FTA, and one that has received much research interest. In the analysis of coherent models using BDDs, a fault tree is first converted into a BDD, where each of the basic events of the tree is represented as a node with two branches (branch 1 and 0, corresponding to the component failure and working state respectively). This traditional structure (known as structure-function BDD), however, cannot be used in non-coherent analysis. This is because a component x in a non-coherent fault tree can contribute to a particular failure in its failed state (referred to as "failure-relevant", x) or working state (referred to as "repair-relevant", ¬x); or be excluded as a cause of failure (irrelevant to the system failure). Hence the two branches (failure and success) in a BDD node are no longer sufficient to represent these three states.

To solve this problem, an extended Meta-products BDD was introduced by (Rauzy and Dutuit, 1997) which associates two variables (Px for "presence" and Sx for "Sign") with every component x. "Presence" of the variable x encodes its relevancy (relevant or irrelevant) while "sign" of the variable x encodes its type of relevancy (failure or repair relevant).

Another method for non-coherent fault tree analysis, called the Consensus BDD, has been proposed in (Beeson, 2002). It employs the Consensus law to encode "hidden" prime implicant sets and introduces three branches to every BDD node to distinguish the three possible states of a component The Consensus law is given below:

Consensus Law: $A.B + \neg A.C = A.B + \neg A.C + B.C$ [1]

And simply states that: if B causes system failure when A fails and C causes system failure when A works, then the combination of B and C inevitably cause system failure regardless of the state of A. In such circumstances B.C is known as the "hidden" prime implicant set that can be identified by application of Consensus.

Research on analysis of coherent trees has shown that the BDD approach can be more efficient than conventional FTA methods. Hence the development of extensions for non-coherent fault trees (including proposals such as the MPBDD and Consensus BDD) is highly justified and still represents an open area for research. For the purposes of this work, an alternative approach that combines a modified classical BDD structure and iterated consensus (Sharvia, 2007) was developed.

In its core, this new approach is an adaptation of the classic coherent BDD analysis to non-coherent systems. Non-coherent fault trees are still translated to BDDs. But unlike in a structure-function BDD where a single node is used to represent both the failed state of a component and its negation (working state), here these two states are represented in two separate nodes which are initially treated as independent (as shown in figure 2). This enables the three states (i.e. failed, working and irrelevant) to be distinguished and a classic BDD structure with two branches in each node to be retained. Although the "irrelevant" branch is duplicated, the node representing the negated basic event (working state) can be used where and when necessary without the need to modify the classic BDD structure as in the MP-BDD or Consensus BDD. The conversion from the fault tree to the proposed BDD structure is done in the classical way of coherent BDD analysis and the known BBD algorithm used for minimal cut-sets still applies and can be used for prime implicant generation.

One consequence of this approach is the possibility of hidden prime implicant sets to arise as a result of treating the failed and working state of a component as separate independent nodes. This problem is overcome by extending this approach with effective iterated consensus (see equation 1) which is applied once initial sets are generated from the BDD. By applying consensus and by removing any contradictions that may arise in the process (of the type $x . \neg x = 0$) the full complete prime implicant sets can be obtained.

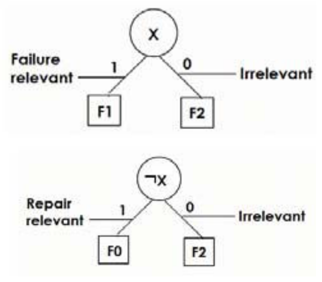

Figure 2 : Proposed BDD node structure

As an example, Figure 3 shows an example fault tree and Figure 4 shows its equivalent BDD according to the proposed conversion. From the BDD, prime implicant sets of "B.A + ¬A.C" can first be obtained using the algorithm for coherent BDD analysis and then the hidden prime implicant "BC" is added to the set by application of Consensus.

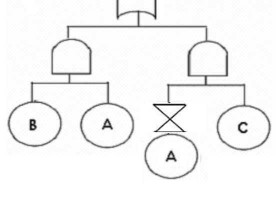

Figure 3: Example of non-coherent fault tree

[3] The presence of biform events in non-coherent structure potentially leads to the creation of "hidden"prime implicant sets. An event is said to be monoform in a logical expression written as sum of products if the event X occurs while the negated event (¬ X) does not occur. Otherwise it is said to be biform.

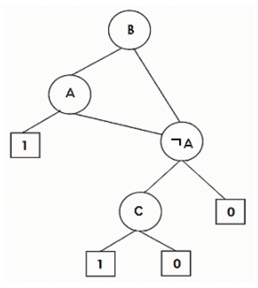

Figure 4 : Representation of sample fault tree in BDD

A detailed description of the algorithm is beyond the scope of this paper. For the purposes of this work it suffices to describe the principle and note that the algorithm has been implemented and integrated in the HiP-HOPS tool which now supports a new type of non-coherent compositional failure modelling. An example follows which demonstrates application of the approach to a small system.

4. EXAMPLE

Figure 5 illustrates a generic pattern of a primary-standby configuration in which initially the function of the system is provided by the primary component (**A1**) and when this fails function is provided by the standby (**A2**). Using a generic example means that the results of analysis shown here are applicable to a wider class of systems that follow this particular pattern. It can easily be imagined how specific components and the failure modes of these components can substitute the generic references made in the example.

Out is the system output, and is initially the output of component A1. A1 performs a function on the outputs provided by components **S1** and **S2**, which in turn perform functions on a common input **In**. **M** is a monitor that detects an omission of output from A1 (O-A1) and activates the standby component A2. In that case, the system output is provided by A2 instead.

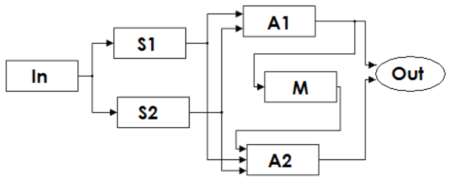

Figure 5: Simplified standby system

In this design, any single value failure or omission at the output of S1 or S2 (V-S1, V-S2, O-S1, O-S2) causes a value failure at the output of A1 or A2. On the other hand, omission of both outputs from S1 and S2 causes omission of A1 or A2. Two failures can be observed at the output of the system:

1. Omission failure (O-out): caused by omissions of output of both A1 and A2 (O-A1.O-A2).
2. Value failure (V-out): caused by value failure of either A1 or A2 (V-A1+O-A1.V-A2) depending on mode

The internal malfunctions of components are as follows:

S1 : Component S1 failed causing omission of output
V1 : Component S1 biased causing incorrect value at output
S2 : Component S2 failed causing omission of output
V2 : Component S2 biased causing incorrect value at output
A1 : Component A1 failed causing omission of output
A2 : Component A1 failed causing omission of output
In : Input generator failed causing Omission of input
M : Monitor M failed causing omission of output

In coherent analysis, the failure logic that links output failures to internal malfunctions and input deviations for each component can be described as follows:

Out:	O-out = O-A1.O-A2
	V-out = V-A1 + O-A1.V-A2
A1:	O-A1 = A1 + O-S1.O-S2
	V-A1 = V-S1+ V-S2 + O-S1 + O-S2
A2:	O-A2 = O-M + A2 + O-S1.O-S2
	V-A2 = V-S1+ V-S2 + O-S1 + O-S2
S1:	O-S1 = S1 + O-I
	V-S1 = V1
S2:	O-S2 = S2 + O-I
	V-S2 = V2
M:	O-M = M
Input:	O-I = In

By starting from the output of the system and by progressively substituting in the expressions above conditions that represent deviations of parameters with their causes, coherent fault trees can be created that explain the causes of two system failures: omission of output (O-out) and incorrect system output (V-out)

O-out	= O-A1.O-A2
	= (A1 + O-S1.O-S2).(O-M + A2 + O-S1.O-S2)
	= (A1 + S1.S2+ In).(M+A2+ S1.S2+ In)
	= A1.M + A1.A2 + In + S1.S2
V-out	= V-A1 + O-A1.V-A2
	= V-S1+ V-S2 + O-S1 + O-S2
	= V1 + V2 + S1 + S2 + In

The result for O-out are as expected, i.e. omission of output is caused by failure of both components A1 and A2, both component S1 and S2, failure of primary and monitor in which case the standby is not started, or omission of input. The results for V-out are more interesting. They state, for example, that a failure of either S1 or S2 (S1 + S2) will cause an incorrect output V-out. Indeed, if **one** of these components fails causing an omission of its output (i.e. O-S1, O-S2) this in turn should cause a value failure because A1 and A2 rely on both inputs and inevitably produce incorrect results (in applications such as vehicle braking applying incorrect braking on demand may still be better than failing to brake). However, if **both** S1 and S2 fail causing omission of their outputs, this will cause omission of system output. The problem is precisely this, that the current causes of V-out indirectly include the possibility of S1 and S2 failures occurring together (because S1 + S2 contains S1.S2) and this is wrong. Furthermore, this cannot be corrected using coherent modelling; to distinguish between the causes of an omission (S1.S2) and the causes of a value failure we need to express the latter by using a XOR gate and by making references to the working states of components (S1⊕S2 = S1.¬S2+ S2.¬S1). Of course using XOR gates means using NOT gates and non-coherent modelling.

By applying non-coherent modelling in the failure logic of A1 and A2, the fault tree for V-out becomes:

V-out = V-A1 +O-A1.V-A2
= O-S1. ¬ O-S2 + ¬ O-S1.O-S2 + V-S1 + V-S2
= (S1 + In).¬ (S2+In) + ¬ (S1+In). (S2+ In) + V1 + V2
= S1.¬S2.¬In + S2.¬S1.¬In + V1 + V2

This accurate representation gives a correct and clear view of the contribution of component failures to the system, as well as helps to eliminate spurious causes. Note, for example, that in coherent analysis, omission of input (In) is recorded as a cause of value failure at the output of the system because it causes omission of component S1,S2 output which in turn can (but not always) cause such value failure. However, what is overlooked here is that the omission of input can only cause simultaneous omission of both S1 and S2, and as such, it can only cause an omission of system output (O-out) and NOT a value failure. This further inaccuracy is also corrected in non-coherent analysis.

5. CONCLUSIONS

In this paper, we have given evidence in support of non-coherent modelling in FTA and argued that this type of modelling is important in the context of the compositional FTA techniques which have emerged since the late 90s. We have also described an extension that enables non-coherent modelling in the context of HiP-HOPS, a technique that enables a form of compositional, semiautomatic, model-based FTA on complex systems. We have shown that, using this extension, it is possible to analyse the failure behaviour of a system more precisely, and obtain more accurate results than those gained via coherent analysis. Through this extension, the benefits of a more precise non-coherent analysis can be combined with the benefits of automation and potential reuse of component failure models inherent in HiP-HOPS. In certain cases, non-coherent modelling will not only indicate potential weaknesses in system design, but also suggest subtle remedial measures.

Ultimately, the decision as to whether or not to use NOT gates should strike a balance, among the need for precision and correctness, the increased complexity in failure modelling and the greater computational expenses involved. However, in any case, the extension described in this paper will enable a wider range of failure modelling capabilities to be explored within compositional FTA techniques which will further contribute, we hope, to the debate on non-coherent modelling in the future.

ACKNOWLEDGEMENTS

This work was supported by the European Framework 6 Integrated Project SAFEDOR (IP-516278). The authors would like to thank Martin Walker for his help.

REFERENCES

Andrews, J.D. (2000). To not or not to not. *Proceedings of the 18th international system safety conference.* Forte Worth, pp 267-275.

Beeson, S.C. (2002). *Non-coherent fault tree analysis.* Loughborough University. UK.

Grunske, L., B. Kaiser, Y. Papadopoulos (2005). Model-driven safety evaluation with state-event-based component failure annotation. *Lecture notes in computer science* **3489**: pp 33-48, Springer. Berlin.

Johnston, B.D. and R.H. Matthews (1983). *Non-coherent structure theory: A review and its role in fault tree analysis.* United Kingdom Atomic Energy Authority.

Kaiser, B. (2004). A fault-tree semantics to model software-controlled systems *Softwaretechnik-Trends* **23(3),** Gesellschaft für Informatik (Hg.).

Kaiser, B., P.Liggesmeyer, O.Mackel (2003). A new component concept for fault trees. *Conferences in research and practice in information technology,* **33**. P.Lindsay & T. Cant, Eds. Canberra.

Kumamoto, H. and A. Henley (1978) Top-down algorithms for obtaining prime implicant sets of non-coherent fault tree. *IEEE Transactions on reliability*, **R-27/4**: pp 249.

Merle G. and J.-M. Roussel (2007). Algebraic modelling of Fault Trees with Priority AND gates, *DCDS'07*, Paris, France, pp 175-180.

Papadopoulos, Y., J.A. McDermid, R.Sasse, and G.Heiner (2001). Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions. *Reliability engineering and system safety* **71(3):** pp 229-247**.**

Quine, W.V (1955). A way to simplify truth function. *American mathematical monthly* **62**: pp 627-631

Rauzy, A., Y.Dutuit (1993). New algorithms for fault tree analysis. *Reliability engineering and system safety* **40:** pp 203-211.

Rauzy, A., Y.Dutuit (1997). Exact and truncated computations of prime implicants of coherent and non-coherent fault tree. *Reliability engineering and system safety* **58:** pp 127-144.

Sharvia, S. (2007). *Extending fault tree synthesis with negative logic operator.* MSc Thesis, University of Hull, UK.

Vesely, W.E., F.F. Goldberg, N.H. Roberts, D.F. Haasl (1981). *Fault tree handbook*, Washington D.C., USA. US NRC.

Walker, M. and Y.Papadopoulos (2006). PANDORA: The time of Priority-AND gates. INCOM 2006, France: pp 237-242.

Worrell, R.B and D.W. Stack (1978). *A SETS user manual for the fault tree analyst.* NUREG CR_04651, Washington D.C., US NRC.

Worrell, R.B., D.W. Stack, B.L.Hulme (1981). Prime implicants of non-coherent fault trees. *IEEE Transaction on reliability* R-**30**/2 : pp 98-100.