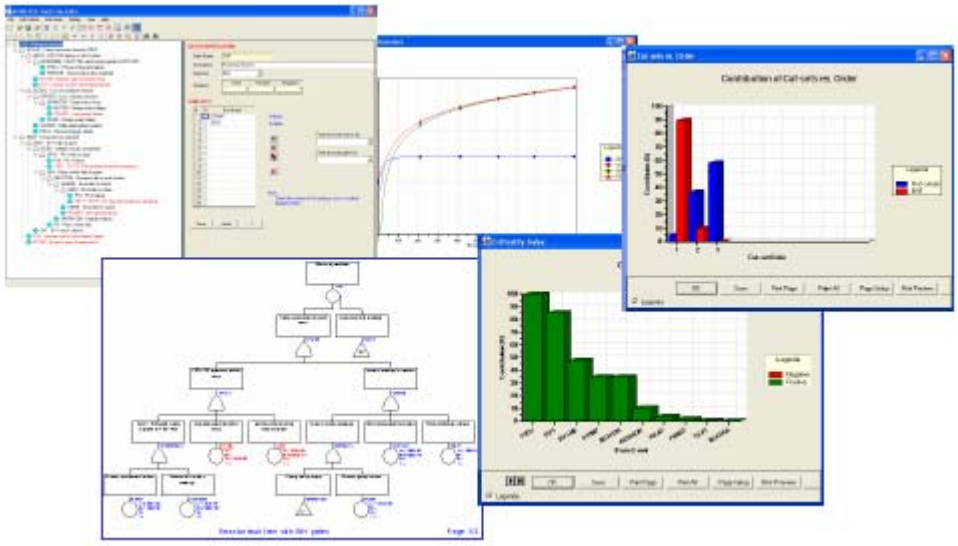
JRC Scientific and Technical Reports

ASTRA 3.0: LOGICAL AND PROBABILISTIC ANALYSIS METHODS

Description of the main phases and algorithms of the fault tree analysis procedure implemented in ASTRA 3.0

Sergio Contini and Vaidas Matuzas



EUR 24152 EN - 2010

JRC
EUROPEAN COMMISSION

ipsc
Institute for the Protection and Security of the Citizen

The mission of the JRC-IPSC is to provide research results and to support EU policy-makers in their effort towards global security and towards protection of European citizens from accidents, deliberate attacks, fraud and illegal actions against EU policies.

European Commission
Joint Research Centre
Institute for the Protection and Security of the Citizen

Contact information
Address: Sergio Contini
E-mail: Sergio.contini@jrc.ec.europa.eu
Tel.: +39 0332 789217
Fax: +39 0331 785145

http://ipsc.jrc.ec.europa.eu/
http://www.jrc.ec.europa.eu/

JRC 56318

EUR 24152 EN
ISBN 978-92-79-14857-6
ISSN 1018-5593
DOI 10.2788/55214

Luxembourg: Publications Office of the European Union

Printed in Italy

SUMMARY

This report contains the description of the main methods, implemented in ASTRA 3.0, to analyse coherent and non-coherent fault trees. ASTRA 3.0 is fully based on the Binary Decision Diagrams (BDD) approach. In case of non-coherent fault trees ASTRA 3.0 dynamically assigns to each node of the graph a label that identifies the type of the associated variable in order to drive the application of the most suitable analysis algorithms. The resulting BDD is referred to as Labelled BDD (LBDD). Exact values of the unavailability, expected number of failure and repair are calculated; the unreliability upper bound is automatically determined under given conditions. Five different importance measures of basic events are also provided. From the LBDD a ZBDD embedding all the MCS is obtained from which a subset of Significant Minimal Cut Sets (SMCS) is determined through the application of the cut-off techniques.

With very complex trees it may happen that the working memory is not sufficient to store the large LBDD structure. In these cases ASTRA 3.0 offers to the user a new method that applies the cut-off technique during the construction of the ZBDD, thus by-passing the construction of the LBDD. The result is a Reduced ZBDD (RZBDD) embedding the SMCS.

Finally, the report also contains three short tutorials on the usefulness of non-coherent fault trees, on the BDD approach, and on the determination of failure and repair frequencies.

TABLE OF CONTENTS

1. INTRODUCTION

Fault Tree Analysis (FTA) is the most popular methodology for RAMS studies of complex systems; it allows to systematically describe the system's failure logic for each system failure state and to quantify the corresponding occurrence probability / frequency. FTA is applied for design review to prove that the system is reasonably safe and that it is well protected against both internal failures and external events.

The Fault trees of real systems that contain AND, OR Boolean operators are referred to as *Coherent,* and are characterised by monotonic (non-decreasing) functions with respect to all basic events. Non monotonic logical functions, due to the presence of the NOT operator, are also of interest in system analysis. They are referred to as *not-coherent fault trees* and are very helpful in modelling e.g. the following types of problems:

- mutually exclusive events;
- event-tree sequences;
- top-events conditioned to the working state of one or more component / subsystem;
- safe maintenance procedures.

ASTRA allows the user to handle both coherent and non coherent fault trees.

The software ASTRA 3.0 is based on the state of the art approach of Binary Decision Diagrams (BDD). This approach, which was developed for the minimisation of logical circuits, was then introduced in the reliability field mainly by French researchers. Today a vast literature is available on this subject; see e.g. Ackers (1978), Bryant (1986), Brace et al. (1990), Couder-Madre (1994), Rauzy (1993), Rauzy-Dutuit (1997).

A BDD is a compact graph representation of Boolean functions. The main advantages of using the BDD approach in fault tree analysis is given by the possibility to: 1) determine the exact value of the top event unavailability and failure frequency, and 2) obtain a compact graph embedding all MCS, thus reducing the working memory space. On the graph it is possible to perform an exact probabilistic quantification. Then, from this graph, another graph can be derived embedding all Minimal Cut Sets (MCS), from which Significant MCS can easily be extracted using the classical (probabilistic / logical) cut-off techniques. The greater efficiency of the BDD approach with respect to previous approaches is somewhat surprising. Trees analysed in the past with great difficulties have been analysed with the BDD approach in few seconds and without introducing any approximation. This is due to the compact representation of the fault tree and to the high efficiency of the algorithms working on the graph.
These characteristics make the BDD approach also suitable to deal with security related applications of fault tree analysis, where high probabilities prevent the use of bounds.

In the case of non-coherent fault trees ASTRA 3.0 dynamically assigns to each node of the graph a label that identifies the "local" type of the associated variable. This is done in order to drive the application of the most suitable analysis algorithms, since the complexity of these algorithms depends on the type of variable. The resulting BDD is referred to as Labelled BDD (LBDD).

In spite of the superiority of the BDD approach over all previous approaches, it could happen that the construction of the BDD cannot be completed due to the exponential increase of the number of nodes with the complexity of the fault tree. In these cases the ASTRA package offers to the user a new module able to determine only the part of the BDD embedding the Significant MCS (SMCS) during the construction of the BDD. This method produces a compact graph embedding the SMCS having probability greater than or equal to the user defined probabilistic threshold P_{lim} and/or order less than or equal to the logical threshold n_{lim}.

This method however presents some limitations:
1) the probabilistic results are not conservative, in that they are based on a subset of MCS;
2) there is no information about the truncation error, i.e. the probability of all neglected MCS.
In these cases it is important either to estimate the truncation error or to calculate upper and lower bounds of the Top-event probability. This is still an important subject of research.

This report describes the main algorithms implemented in ASTRA 3.0 for the logical and probabilistic analysis of coherent and non-coherent fault trees. After a brief description of the implemented fault tree analysis procedure, provided in the next section, the logical analysis methods, from the construction of the LBDD to the determination of the BDD embedding all MCS up to the extraction of the significant MCS are described in sections 3 and 4. Section 5 provides the equations for the determination of the unavailability, the unconditional failure and repair frequencies, the expected number of failure and repair, the unreliability, the characteristic times (Mean Time To Failure, Mean Time To Repair and Mean Time Between Failures), and the importance measures of basic events.

Some brief tutorials, introducing those concepts that may help making the content of the report more understandable, are provided in the Appendixes. They deal with: A) the definition of coherent and non-coherent functions; B) the BDD applied to fault tree analysis; and C) the determination of failure and repair frequencies.

2. OVERVIEW OF THE ASTRA 3.0 ANALYSIS PROCEDURE

2.1 Types of operators

ASTRA allows the user to analyse fault trees containing the following set of operators: AND, OR, K/N, NOT, XOR and INH, i.e. to analyse both coherent and non-coherent fault trees. In Appendix A the concepts of coherent and non-coherent functions are described with some simple examples. Boundary Conditions (BC) can be assigned to basic events. Other operators such as NAND and NOR can be easily represented respectively as NOT-AND and NOT-OR.
The first three operators are well known and will not be discussed further, while some comments will be made for the others and for boundary conditions.

The NOT operator allows the analyst to easily model complex failure logic, e.g.:
- top-events conditioned to the good state of one or more components/subsystems;
- sequences of event trees;
- safe maintenance procedures.

The XOR operator is used to model a relationship between $n \geq 2$ events such that one occurs and n-1 do not occur. During the fault tree construction the analyst should carefully verify these types of conditions. The use of the OR operator, instead of XOR, may lead to logically impossible failure modes when input events are dependent sub-trees. Note that the representation of a mutually exclusive relationship using the XOR operator should not be confused with the mutual exclusivity of physically disjoint events, such as for instance the different failure states of a multistate component. Consider, for instance, two independent components connected in parallel. If we are interested in the probability of both components failed then we model the system using the AND operator. However, if we are interested in determining the probability of failure of only one component we use the XOR operator because components are independent. In other words both components, being independent, can also be both failed, but we are simply not interested in this state.

For non-coherent fault trees, represented as non-monotonic logical functions the concept of minimal cut set has to be replaced with that of *Prime Implicant* (PI), i.e. a combination made up by negated and not negated primary events, which are not contained in any other implicant. Non-monotonic functions are more complex to analyse than the monotonic ones.
By removing negated events from the set of PI and minimising the result, an approximated form, expressed as a set of MCS, is obtained. This implicitly means to assign unit probability to negated events, which represents a condition frequently met in practice in safety applications. Consequently, the probabilistic quantification gives conservative results.
It follows that the need to consider negated events may be limited to the logical analysis only in order to remove impossible combinations, in which an event is present in both forms, negated and not negated. The advantages of applying this simplified approach are twofold, namely:
- Clearer interpretation of system failure modes;
- Significant reduction of computation time and working memory space.

ASTRA 3.0 analyses non-coherent fault trees in two ways:
1) It performs the exact probabilistic analysis and determines the MCS;
2) It performs an approximated probabilistic analysis using the rare events approximation.

In both cases the set of Significant MCS are calculated using the logical and/or probabilistic cut offs. Hence, in this version of ASTRA, prime implicants are not determined.

The INHIBIT (INH) operator is useful to model situations in which the output event occurs when the input event occurs and a conditional event is already verified. This operator presents only two input variables: the initiating event and the enabler event. The INH gate is applied to correctly quantify the occurrence probability of sequential events.
In developing fault trees of, for instance, chemical installations, situations frequently encountered are those where the occurrence of an event (called initiating event) perturbs one or more system variables and places a demand for the protective system to intervene. The not intervention of the protection system allows the perturbation to further propagate in the plant which, eventually, may generate a dangerous situation or be the cause of an accident. Obviously, a dangerous situation occurs only if the protective system is already failed when the initiating event occurs.
Consider, for instance, a pressurised tank, and the event "tank rupture" due to overpressure and the unavailability on demand of the automatic relief system. The rupture of the tank can occur only if the relief system is already failed at the time the overpressure occurs; the opposite situation is a sequence that may lead to other undesired events, e.g. the production loss, but certainly not to the tank rupture.
In practice, the part of the plant that generates a perturbation (i.e. the control system) and the protective system may not be completely independent: a same component can indeed belong to both systems. Sequences of that kind need to be properly modelled in the fault tree for catastrophic top-events, in order to perform the probabilistic analysis correctly. Therefore the analyst must carefully identify such cases during the construction of the fault trees.
Summarising:

1. the correct modelling of these types of events requires to account for the sequence of events, since a simple AND gate would give conservative (sometimes even too conservative) results (Demichela et al. 2003);
2. In ASTRA the initiating and enabling events are modelled by means of an extended definition of the INH gate;
3. The two inputs can also be sub-trees not necessarily independent; common event are automatically identified and treated as initiators.
4. When the INH operator is used, each MCS must contain at least an initiating event; should an MCS be made up of enabler events only, an error message is generated.

The use of the INH gate dates back to 1978 when the analysis of a chemical plant highlighted the need to distinguish between events causing plant perturbations from events representing the failure of the protective system' components. This distinction was implemented in the fault tree analyser developed in that period at the JRC. The INH gate was used also for graphical purposes, in that the number of INH gates gave, at a first glance, an indication about the degree of protection against dangerous initiating events. In other words the absence of INH gates in a fault tree was considered not acceptable. Since then, this choice remained in all fault tree analysis codes developed so far at the JRC.

BOUNDARY CONDITIONS (BC) can be assigned to a subset of basic events. A boundary condition can assume only two values: good / failed, corresponding respectively to states 0, 1. Hence the analysis of a fault tree with boundary conditions allows determining the Top event occurrence probability conditioned to the state of one or more basic events. Events with BC can also be used as “House” events. Events with BC do not appear in any MCS since their value is used to properly remove them before starting the BDD construction.

2.2 Analysable fault trees

ASTRA 3.0 has been developed for the analysis of both coherent and non-coherent fault trees. Besides Appendix A, which provides the basic definition of coherence and the concept of non-coherence, a description of the possible uses of non-coherent fault trees for safety and security applications can be found in Contini et al. (2004 and 2008).

In ASTRA 3.0 basic events are classified as:

– *Positive* or normal, representing the failed state of a binary component;
– *Negated* or complemented, representing the working state of a binary component;
– *Double form*, when an event is present in the fault tree in both forms, positive and negated.

For instance, the function $\phi = \bar{a}\, b + \bar{a}\, c + b\, \bar{c}$ contains the negated variable *a*, the positive variable *b* and the double form variable *c*.

This classification allows constructing an OBDD, which is referred to as "Labelled BDD" (LBDD), in which all nodes are dynamically labelled with the variable type. This solution was adopted considering that the degree of complexity of the (logical and probabilistic) analysis algorithms depends on the type of variables.

Basic events are associated with the failed state of components that may be:

– Not repairable;
– On-line maintained;
– Tested/inspected.

Moreover, a constant probability can be associated with basic events describing items acting on-demand.
The distribution of time to failure and time to repair is exponential.

2.3 Fault tree analysis procedure

The analysis procedure implemented in ASTRA 3.0 follows the main phases pictured in Figure 2.1.

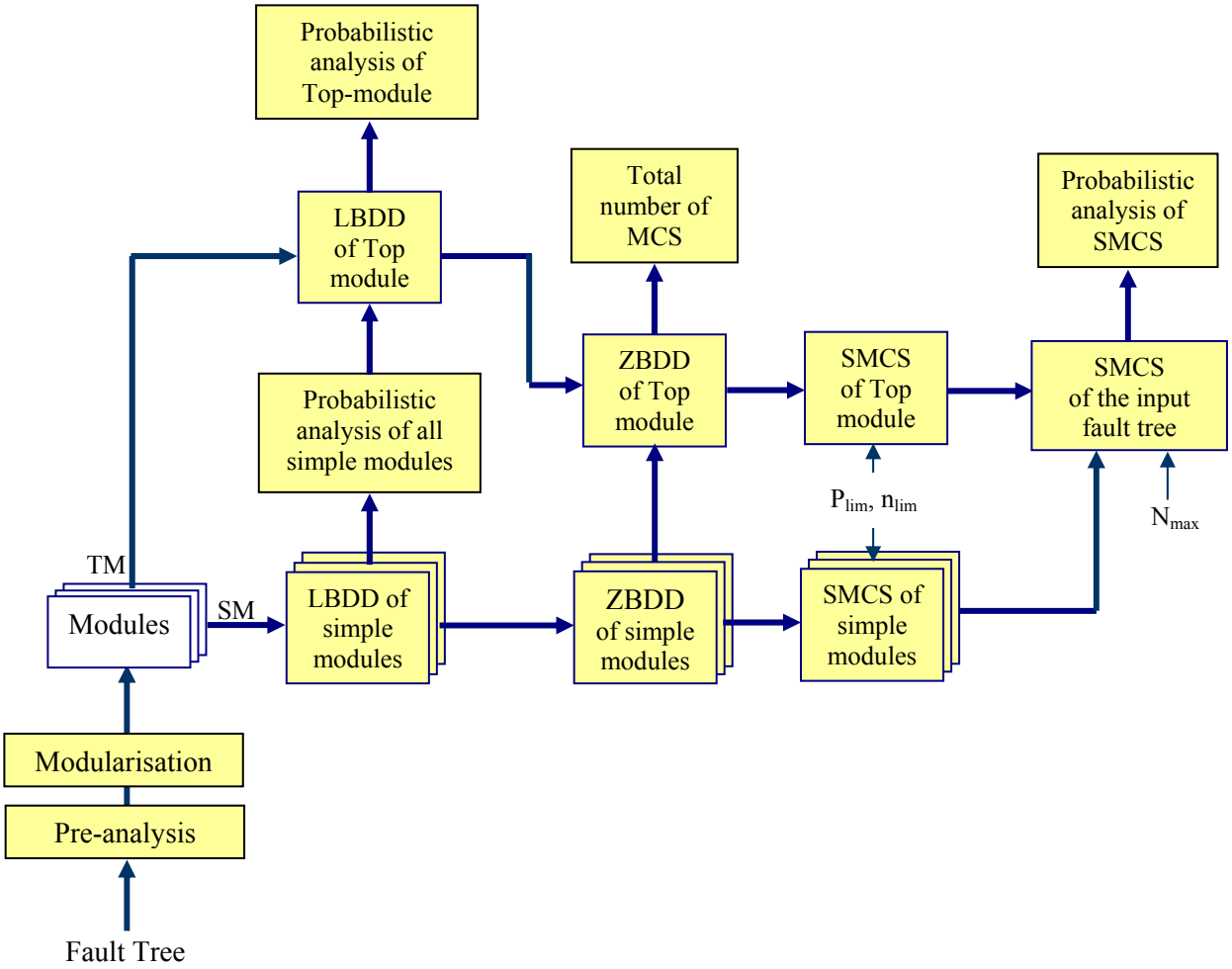


Figure 2.1 Main phases of the analysis procedure implemented in ASTRA 3.0

Initially a pre-analysis of the fault tree is performed. It consists of:
- complete check of the input data, to avoid analysing a wrong input data set;
- expansion of K/N and XOR gates into AND-OR-NOT equivalent expressions;
- transformation of INH gates into AND gates and labelling of enabling events;
- application of De Morgan rules (if any).

Moreover, if the fault tree is non-coherent and the selected analysis option is "Approximated", then unrepeated negated basic events are removed from the fault tree, since they are not necessary for deleting impossible cut sets. This option is useful when dealing with complex fault trees of safety studies containing large negated not repeated sub-trees. Indeed, removing unrepeated events allows reducing the fault tree dimension and the computational effort.

Then the input fault tree is modularised i.e. it is decomposed into a set of "Simple Modules, SM" and a "Top-module, TM".
A simple module is a sub-tree containing basic events that are not replicated in any other module; however, basic events can be replicated within a simple module.
The Top-module is the module that contains the top event definition and generally is the most complex one; it contains simple modules as fictitious basic events. If the Top-module is non-coherent, a simple module may also be of SN or DF type.

All modules, being independent, are independently analysed.

Simple modules are examined first. Each of them is stored in the form of LBDD. The method used to construct an LBDD is described in the next section. The results of the probabilistic quantification of simple modules are used to feed the Top-module, which is also stored as LBDD. The following parameters are determined for each simple module in its normal or positive form: unavailability and importance of basic events. If a simple module appears also in its negated form, then its parameters are obtained from those calculated for the positive form. The relationship applied to determine the parameters of a negated module from the parameter of the same module in positive form are described in section 5.4.2.

The quantification of the Top-module allows obtaining the exact values of the Top-event Unavailability, Expected number of failure, and various importance measures of basic events for both coherent and non-coherent fault trees. Moreover the Unreliability upper bound is determined when the failure frequency does not present a steady state behaviour. The equations applied are described in section 5.

The analysis proceeds with the independent determination of the ZBDD of simple modules and of the Top module. The ZBDD (Minato, 1990) is a compact graph embedding MCS or Prime Implicants (PI). For non coherent fault trees Prime Implicants (PI) should be determined due to the presence of negated events. Since generally a PI contains many negated events (representing working components) and few normal events (representing failed components), when the probability of negated events is very close to 1 it is more convenient to determine the MCS by removing negated events.
In practice all negated events are removed from the LBDD; the resulting MCS are stored, after minimisation, as a ZBDD.

Given that the probabilistic analysis has already been performed on the LBDD, the determination of MCS can be limited to the most important ones, which in this report are referred to as Significant MCS (SMCS). The SMCS are determined for all simple modules and for the Top module by setting up the thresholds on the order and unavailability of MCS (cut-off values):
- If the *Logical cut-off* n_{lim} is applied a MCS is retained if its order $m \leq n_{lim}$;
- According to the *Probabilistic cut-off* P_{lim} a MCS is retained if its probability $Q \geq P_{lim}$.

Remembering that simple modules are represented in the Top-module as fictitious basic events, the significant minimum failure combination of the Top-module, referred to as Macro MCS (MMCS) do not represent the SMCS of the input fault tree. A further step is necessary to expand the significant failure combinations of simple modules into the significant MMCS in which the modules appear.
For instance the j-th MMCS say C_j of the Top-module can be represented as:

$$C_j : \bigcap_{i=1}^{w} BM_i \bigcap_{k=1}^{r} BE_k$$

where w and r are respectively the number of simple modules and basic events making up the j-th MMCS; BM_i represents a generic simple module; BE_k denotes a generic basic event. Depending on w and on the number of its failure combinations, C_j may indeed contain a large number of MCS. If m_i is the number of combinations contained in the generic simple module BM_i, then the number of MCS of the input tree embedded in C_j is equal to $\prod_{i=1}^{w} m_i$

An efficient algorithm has been implemented to extract the SMCS by setting up the threshold value N_{max}. The *cut off* N_{max} defines the maximum number of SMCS to be extracted from a single MMCS. The use of this cut-off, at the end of the analysis of each MMCS, implies the automatic modification of P_{lim} to a value equal to the probability of the least important SMCS extracted. This new P_{lim} value will then be used for the analysis of the next MMCS, and so on. The result is the set of the most important MCS of the input tree: their number is generally close to N_{max}.

The final phase of the analysis concerns the determination, depending on the type of analysis, of the probabilistic parameters of interest, i.e. unavailability, unconditional failure and repair frequencies, expected number of failures and repairs and unreliability for all SMCS.

To summarise, the analysis procedure described so far can be subdivided in two parts:

1) Construction of the LBDD for all modules, exact probabilistic analysis performed on the LBDD and construction of the ZBDD;
2) Use of the cut-off technique for the determination of the SMCS.

From the tests performed on ASTRA on a very complex fault trees it appeared that the working memory was not sufficient to store the large LBDD generated. This is indeed a characteristic of the BDD approach: the number of nodes increases exponentially with the complexity of the fault tree.
In order to overcome this limitation another analysis procedure, represented in Figure 2.2, was implemented to directly determine, for each module, the ZBDD containing the SMCS, so by-passing the construction of the LBDD. The analysis modules in Figure 2.2 that also appear in Figure 2.1, are represented with dotted lines.

As shown in Figure 2.2 the cut-off values P_{lim1} and/or n_{lim1} are applied to all simple modules and to the Top-module. The probabilistic analysis is performed on the ZBDD of the Top-module in which simple modules are represented as fictitious basic events. The ZBDD of the Top-module embeds a number of minimal cut sets indicated as SMCS-1. The determination of the MCS of the input tree, i.e. the set SMCS-2, requires the application of the algorithm based on N_{max} as previously described.

The method implemented, which is described in Section 4, is applicable to both coherent and non-coherent fault trees.

This analysis procedure allows the user to solve fault trees of any practical complexity with short computation times. The obvious limitations are that:

- the probabilistic results are (not conservatively) approximated, being based on the set SMCS-1, which is generally a small percentage of the total number of MCS;
- there is no information about the truncation error, i.e. the probability of all neglected MCS;
- the total number of MCS is unknown (after all, for complex trees, this is not practically important).

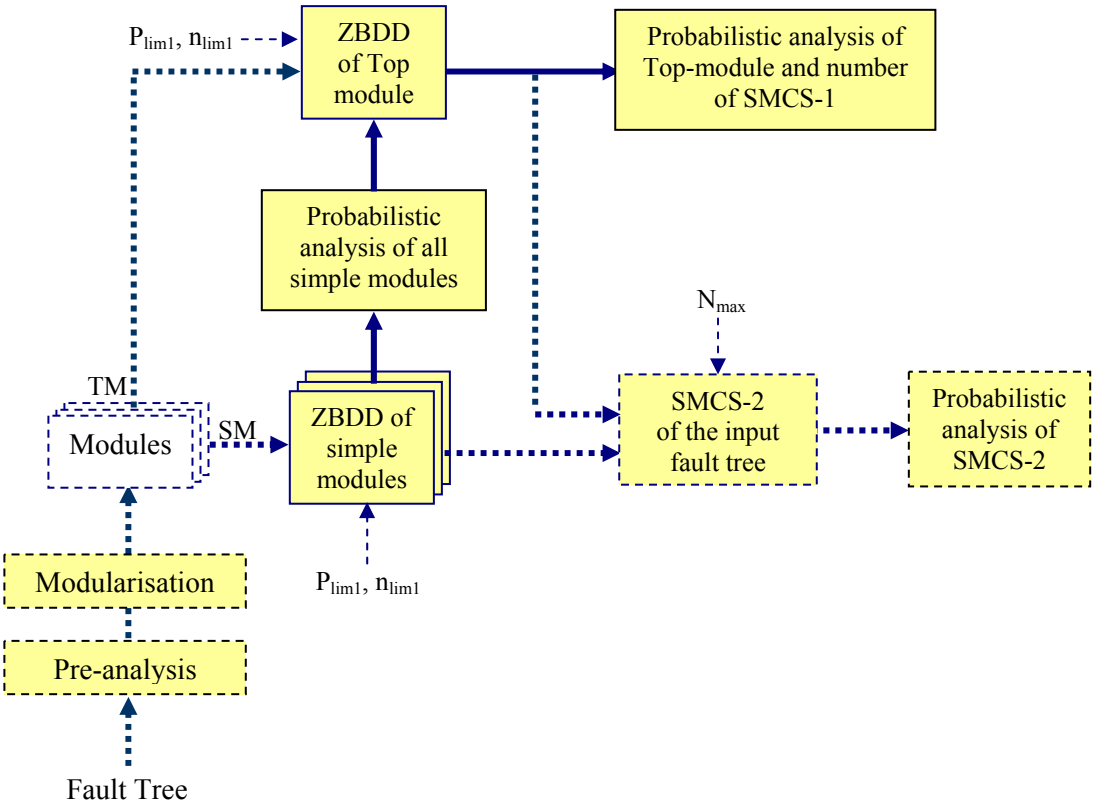


Figure 2.2 Main phases of the second analysis procedure implemented in ASTRA 3.0

Summarising, two different analysis procedures have been implemented in ASTRA 3.0, allowing: 1) the exact analysis of fault trees through the construction of the LBDD and later ZBDD, and 2) the approximated analysis through the direct construction of the Reduced ZBDD by means of the application of the cut-off techniques.

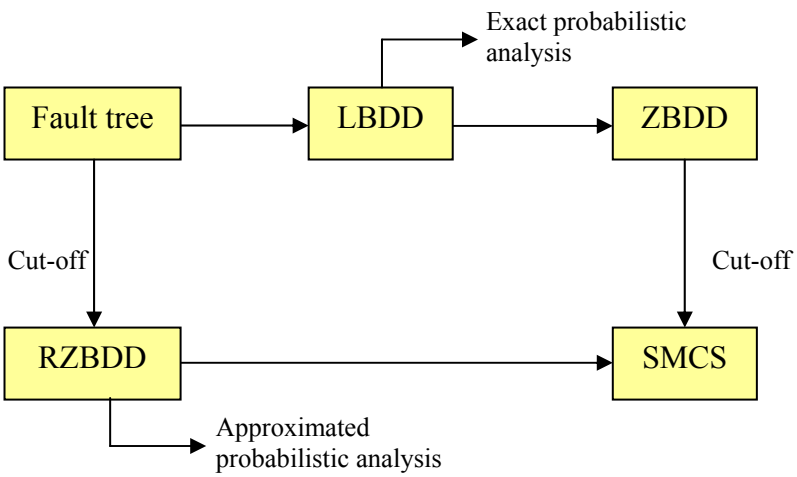


Figure 2.3 The two analysis procedures implemented in ASTRA 3.0

3. LOGICAL ANALYSIS PROCEDURE

The aim of this section is to describe the concept of LBDD, how it is generated and how a ZBDD embedding all MCS is derived.

3.1 Classification of variables in non-coherent fault trees

The Boolean function describing the logical relationships among events in fault trees can be monotonic or not monotonic, also commonly referred to as *Coherent* or *Non-Coherent*. These concepts are briefly described in Appendix A.

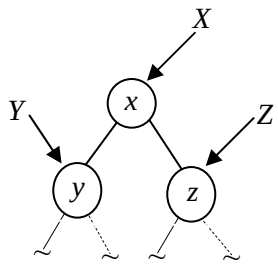
The binary function $\Phi(\mathbf{x})$, with $\mathbf{x}$ a vector of binary variables, of a non-coherent fault tree contains three different types of basic events or variables, namely:

1. normal or positive, e.g. x;
2. negated, e.g. $\bar{y}$;
3. events appearing both in positive and negated forms, e.g. z, $\bar{z}$.

In this report the following definitions are used. Variables of type 1 are referred to as *Single form Positive variables* (SP), *v*ariables of type 2 as *Single form Negated variables* (SN), whereas variables of the third type as *Double Form variables* (DF).

For instance, in the function $F = a\,\bar{b} + \bar{a}\,c$, the variable *a* is of DF type, *b* is of SN type and *c* of SP type.

Let us consider the following section of a BDD, where Y and Z are the two functions having the variables y and z as roots:



This partial BDD may represent different functions, i.e. $X = x\,Y + Z$, or $X = x\,Y + \bar{x}\,Z$ depending on the type of x. In the first function x is of SP type, whereas in the second it is of DF type.

Different types of variables require different algorithms of analysis. Indeed, on nodes with DF variables the determination of the Prime Implicants (PI) and of the failure and repair frequencies require the logical intersection between the left and right descending functions, whereas this is not needed for the other two types of variables.

The information about the type of variables can easily be extracted from the input fault tree and associated to the nodes on the BDD. We shall call this method as "Static Labelling", since the association *node-variable* with the *variable-type* is made after the construction of the BDD. Odeh-Limnios (1996) applied this technique.

However, we can observe that in a BDD a DF variable may be associated with two or more nodes in which it behaves as a positive (SP) or as a negative (SN) variable, the analysis of which require simpler algorithms.

Three different types of variables require the labelling of two out of them. In ASTRA variables of SN type are labelled with the symbol $; variables of DF type are labelled with the symbol &.
Consider for instance the exclusive OR, $F = a \oplus b = a\,\bar{b} + \bar{a}\,b$ that contains two variables of DF type.

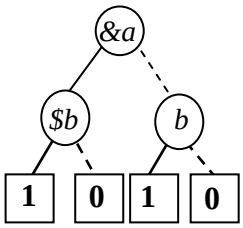


Figure 3.1 Labelled BDD of the function $F = a\,\bar{b} + \bar{a}\,b$

Figure 3.1 shows the labelled BDD in which the labels & and $ are used to characterise respectively the variables of DF and SN type. Note in fact that the variable b is differently labelled in two nodes. We can see that only *a* is represented as a DF variable, whereas the first occurrence of *b* (left descendant of *a*) behaves as an SN variables and the second occurrence of *b* (right descendant of *a*) as an SP variable. Thus, the most expensive algorithm is applied only on one node out of three.

According to Figure 3.1 the function F can be written, in terms of labelled variables, as $F = \&a\ \$b + \overline{\&a}\ b$, from which the following equivalencies can be derived: $\&a = a$, $\overline{\&a} = \bar{a}$, $\$b = \bar{b}$.

The labelling technique applied during the construction of the BDD can be used to characterise each node with the type of the associated variable. The description of the *dynamic labelling* of variables is the aim of this section, in which advantages and limitations with respect to the *static labelling* will be discussed with the help of experimental results.

Before the advent of the BDD approach in system reliability, the non-coherent functions were rarely applied, with the exception of the analysis of Event tree sequences. Indeed, the logical analysis was approximated, i.e. it was based on the determination of the MCSs rather than of the PIs. The MCSs were obtained by removing all negated variables and minimising the resulting logical expression. This was necessary since both the order (number of variables in a PI) and the number of PIs were generally much higher than the order and the number of the MCSs. The justification was based on the fact that the probabilities of negated events were very close to 1.

Rauzy & Dutuit (1997) proposed two BDD-based algorithms for efficiently determining the PIs and the MCSs (referred to as P-Cuts) of a non-coherent fault tree (see Appendix A). We show that the dynamic labelling technique also allows to efficiently calculating the OBDD embedding all MCSs.

3.2 Construction of the LBDD

The objective of this section is to present the construction of an Ordered Binary Decision Diagram of a non-coherent fault tree in which all variables are dynamically labelled with their type, so that the analysis can be performed by strictly applying the most suitable algorithms to each node. Naturally for coherent fault trees the analysis procedure is applied by skipping the unnecessary operations.

The following steps are applied in ASTRA for the logical analysis of a fault tree.

1. The fault tree is reduced to the equivalent Boolean function with AND, OR operators and positive and negated variables (basic events). This is obtained by expanding the gates XOR and K/N in their AND, OR, NOT equivalent expressions.

2. All negated variables are labelled with $, i.e. their name is preceded by the $ character. Consequently $Pr\{\$x = 1,t\} = 1 - q_x(t)$ and $Pr\{\overline{\$x} = 1,t\} = q_x(t)$.

3. The ordering of variables is defined. The user can select one of the available orderings. The ordering selected by the user is applied to all modules.

4. The fault tree modularization is applied giving a set of independent sub-trees (simple and Top modules) that are then independently analysed.

5. The logical (and probabilistic) analysis of modules is independently performed.

6. The results for the whole fault tree are obtained by re-combining the results from each module.

The main steps of the analysis of each module are as follows:

- Construction of the LBDD with labelled variables;

- Probabilistic analysis of the module performed according to the user requests (unavailability, failure and repair frequencies, importance measures, etc.);

- Determination of the ZBDD embedding all MCS.

The remainder of this section describes the algorithms implemented for the construction of both the LBDD and ZBDD.

3.2.1 Composition rules to construct the BDD with labelled variables (LBDD)

The classical OBDD is constructed in a bottom-up way by means of the composition of Boolean functions:

$$H \otimes K = y\,(H_{|\,y=1} \otimes K_{|\,y=1}) + \bar{y}\,(H_{|\,y=0} \otimes K_{|\,y=0}) \qquad (3.1)$$

where:
y is the variable selected for expansion,
$\otimes$ is the Boolean operator (AND, OR),
$H = ite(y, H_{|y=1}, H_{|y=0})$ and $K = ite(z, K_{|z=1}, K_{|z=0})$ are the two Boolean functions to be composed in the case in which $y < z$.

The two residues $(H_{|y=1} \otimes K_{|y=1})$ and $(H_{|y=0} \otimes K_{|y=0})$ are obtained by assigning respectively the values 1 and 0 to y in the Boolean expression $H \otimes K$.

From (3.1) five formulas can be derived to construct the LBDD. The first is applied to combine nodes with different variables regardless of their label, the second to combine nodes with equal variables and equal labels and the remaining deal with the same variable but differently labelled.
The first two rules are those proposed by Rauzy (1993) to construct the classical OBDD. The other three rules allow the dynamic association of the variables' type to nodes.

<u>Different variables, any label</u>

If variables are different, each one maintains its own label, i.e.
Given $H = ite(y, H_1, H_0)$ and $K = ite(z, K_1, K_0)$

If $y < z$ then

$$H \otimes K = ite(y, H|_{y=1} \otimes K), (H|_{y=0} \otimes K)) \quad (3.2)$$

In fact, $H \otimes K = ite(y, (H \otimes K)|_{y=1}, (H \otimes K)|_{y=0})$
Now, since K does not contain y, equation (3.2) follows.

<u>Same variable, same label</u>

Given $H = ite(y, H_1 , H_0)$ and $K = ite(y, K_1 , K_0)$ then

$$H \otimes K = ite(y, (H|_{y=1} \otimes K|_{y=1}), (H|_{y=0} \otimes K|_{y=0})) \quad (3.3)$$

Since H and K contain the same variable y then:
$H \otimes K = ite(y, (H \otimes K)|_{y=1}, (H \otimes K)|_{y=0}) =$
$= ite(y, (H|_{y=1} \otimes K|_{y=1}), (H|_{y=0} \otimes K|_{y=0}))$

<u>Same variable, different labels</u>

A node with a DF variable can be produced in three different ways, according to the following composition rules.

1. This rule assigns the label & to a variable - say x - by combining its positive occurrence x with its negated occurrence $x.
   Given $H = ite(x, H_1 , H_0)$ and $K = ite(\$x, K_1 , K_0)$ then

$$H \otimes K = ite(\&x, (H|_{x=1} \otimes K|_{\$x=0}), (H|_{x=0} \otimes K|_{\$x=1})) \quad (3.4)$$

   Proof.
   $H \otimes K= ite(\&x, (H \otimes K)|_{\&x=1}, (H \otimes K)|_{\&x=0}) =$
   $= ite(\&x, (H|_{\&x=1} \otimes K|_{\&x=1}), (H|_{\&x=0} \otimes K|_{\&x=0}))$
   Now, considering that:
   $\&x = 1 \Rightarrow x = 1$ and $\bar{x} = 0$, i.e. $\$x = 0$; and
   $\&x = 0 \Rightarrow x = 0$ and $\bar{x} = 1$, i.e. $\$x = 1$;
   substituting $\&x = 1$ with the couple $x = 1$; $\$x = 0$ and $\&x = 0$ with the couple $x = 0$, $\$x = 1$, we obtain
   $H \otimes K = ite(\&x, (H|_{x=1, \$x=0} \otimes K|_{x=1, \$x=0}), (H|_{x=0, \$x=1} \otimes K|_{x=0, \$x=1}))$.
   Since H does not contain $x and K does not contain x, equation (3.4) follows.

2. This rule combines an &-labelled variable with its positive part.
   Given $H = ite(\&x, H_1 , H_0)$ and $K = ite(x, K_1 , K_0)$ then

$$H \otimes K = ite(\&x, (H|_{\&x=1} \otimes K|_{x=1}), (H|_{\&x=0} \otimes K|_{x=0})) \quad (3.5)$$

   Proof.
   $H \otimes K = ite(\&x, (H \otimes K)|_{\&x=1}, (H \otimes K)|_{\&x=0}) =$
   $= ite(\&x, (H|_{\&x=1} \otimes K|_{\&x=1}), (H|_{\&x=0} \otimes K|_{\&x=0}))$
   Now, considering that:
   $\&x = 1 \Rightarrow x = 1$;
   $\&x = 0 \Rightarrow x = 0$,
   $H \otimes K = ite(\&x, (H|_{\&=1, x=1} \otimes K|_{\&=1, x=1}), (H|_{\&=0, x=0} \otimes K|_{\&x=0, x=0}))$
   Since H does not contain x and K does not contain &x, then equation (3.5) follows.

3. This rule combines an &-labelled variable with its negative part.
Given $H = ite(\&x, H_1, H_0)$ and $K = ite(\$x, K_1, K_0)$ then

$$H \otimes K = ite(\&x, (H|_{\&x=1} \otimes K|_{\$x=0}), (H|_{\&x=0} \otimes K|_{\$x=1})) \qquad (3.6)$$

Proof.
$H \otimes K = ite(\&x, (H \otimes K)|_{\&x=1}, (H \otimes K)|_{\&x=0}) =$
$= ite(\&x, (H|_{\&x=1} \otimes K|_{\&x=1}), (H|_{\&x=0} \otimes K|_{\&x=0}))$
Now, considering that:
$\&x = 1 \Rightarrow \bar{x} = 0$ i.e. $\$x = 0$;
$\&x = 0 \Rightarrow \bar{x} = 1$ i.e. $\$x = 1$;
then, substituting $\&x = 1$ with the couple $\&x = 1$; $\$x = 0$ and $\&x = 0$ with the couple $\&x = 0$, $\$x = 1$, we obtain
$H \otimes K = ite(\&x, (H|_{\&x=1, \$x=0} \otimes K|_{\&x=1, \$x=0}), (H|_{\&x=0, \$x=1} \otimes K|_{\&x=0, \$x=1}))$.
Since H does not contain $x and K does not contain &x, then (3.6) follows.

3.2.2 Example

This example is taken from Liu-Pan (1990). Let $\phi(\mathbf{x}) = x_2 (x_1 + \bar{x}_3 + \bar{x}_4) + x_3 (\bar{x}_1 + \bar{x}_2 x_4)$ be the non monotonic function, containing four variables of DF type, to be analysed. Considering the ordering $x_2 < x_1 < x_3 < x_4$ the LBDD is represented in Figure 3.2, which has been obtained as follows.
First of all negated variables are labelled with $, giving:

$\phi(\mathbf{x}) = [x_2 (x_1 + \$x_3 + \$x_4)] + [x_3 (\$x_1 + \$x_2 x_4)]$.

It can be easily seen that the application of (3.2) is sufficient to obtain the following ite structures, indicated as H and K:

$[x_2 (x_1 + \$x_3 + \$x_4)] \Rightarrow H = ite(x_2, ite(x_1, 1, ite(\$x_3, 1, ite(\$x_4, 1, 0))), 0)$

$[x_3 (\$x_1 + \$x_2 x_4)] \Rightarrow K = ite(\$x_2, ite(\$x_1, ite(x_3,1, 0), ite(x_3, ite(x_4, 1, 0),0)),$
$ite(\$x_1, ite(x_3,1, 0), 0))$

Now the combination of x_2 and $\$x_2$ needs the application of equation (3.4).
Since:
$H|_{x2=1} = ite(x_1, 1, ite(\$x_3, 1, ite(\$x_4, 1, 0)))$
$H|_{x2=0} = 0$
$K|_{\$x2=1} = ite(\$x_1, ite(x_3,1, 0), ite(x_3, ite(x_4, 1, 0),0))$
$K|_{\$x2=0} = ite(\$x_1, ite(x_3,1, 0), 0))$
$\otimes = \vee$
then:
$Top = ite(\&x_2, (H|_{x2=1} \vee K|_{\$x2=0}), (H|_{x2=0} \vee K|_{\$x2=1}))$

First residue:
$H|_{x2=1} \vee K|_{\$x2=0} = ite(x_1, 1, ite(\$x_3, 1, ite(\$x_4, 1, 0))) \vee ite(\$x_1, ite(x_3,1, 0), 0))$
The application of (3.4) gives:
$ite(\&x_1, 1, ite(\$x_3, 1, ite(\$x_4, 1, 0)) \vee ite(x_3,1, 0))$.
Now, $ite(\$x_3, 1, ite(\$x_4, 1, 0)) \vee ite(x_3,1, 0)$ expanded with respect to x_3 gives:
$ite(\&x_3, 1, 1) = 1$
Therefore, $H|_{x2=1} \vee K|_{\$x2=0} = ite(\&x_1, 1, 1) = 1$

Second residue:

$H|_{x2=0} \vee K|_{\$x2=1}$ = ite($\$x_1$, ite($x_3$,1, 0), ite($x_3$, ite($x_4$, 1, 0),0))

does not change since its Boolean expression does not contain x_2.

Also the expansion with respect to x_3 and x_4 does not change the final result, which is expressed as:

Top = ite($\&x_2$, 1, ite($\$x_1$, ite($x_3$,1, 0), ite($x_3$, ite($x_4$, 1, 0)))

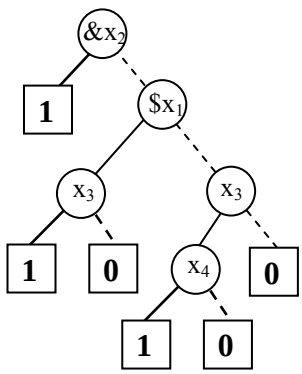


Figure 3.2 LBDD representation of Top = $x_2 (x_1 + \bar{x}_3 + \bar{x}_4) + x_3 (\bar{x}_1 + \bar{x}_2 x_4)$

Note that in this LBDD there is one node with a DF variable ($\&x_2$) and one node with an SN variable ($\$x_1$), whereas the variables associated with all other nodes have no label, i.e. they behave as SP variables. Thus, in spite of the fact that the fault tree has all four variables of DF type, the LBDD has only one.

3.2.3 Simplification of the LBDD

It is possible to further reduce the number of nodes with DF variables if the left and right descendants assume the values 1, 0.

Let &x be the DF variable under examination; $\phi(\mathbf{x})$ = &x H + $\overline{\&}$x K, where H and K are the two residues.

1. If H = 1 then $\phi(\mathbf{x})$ = &x + $\overline{\&}$x K = &x + $\overline{\&}$x K + K = &x + K = x + K. The variable &x behaves as a positive variable. Therefore, the label & can be removed.
   From the probabilistic point of view: Pr{&x + $\overline{\&}$x K } = Pr{ x + K } =
   $q_x(t) + (1- q_x(t))$ Pr(K).

2. If H = 0 then $\phi(\mathbf{x})$ = $\overline{\&}$x K. The variable &x behaves as SN variable, i.e.
   $\phi(\mathbf{x})$ = $x K + $\overline{\$}$x 0. The label & can be substituted with $, and the residues exchanged. From the probabilistic point of view:
   Pr{&x 0 + $\overline{\&}$x K } = Pr{$x K + $\overline{\$}$x 0} = $(1- q_x(t))$ Pr(K).

4. If K = 1 then $\phi(\mathbf{x})$ = &x H + $\overline{\&}$x = &x H + $\overline{\&}$x + H = $\overline{\&}$x + H = $x + H. Thus &x behaves as an SN variable.
   The label & is substituted with $ and the residues are exchanged, i.e. $\phi(\mathbf{x})$ = $x + $\overline{\$}$x H.
   From the probabilistic point of view: Pr{&x H + $\overline{\&}$x} = Pr{$x + $\overline{\$}$x H} =
   $(1- q_x(t)) + q_x(t)$ Pr(H).

4. If K = 0 then $\phi(\mathbf{x})$ = &x H. &x behaves as a positive variable, i.e. $\phi(\mathbf{x})$ = x H, and the label & is removed. From the probabilistic point of view:
   Pr{&x H + $\overline{\&}$x 0} = Pr{x H} = $q_x(t)$ Pr(H).

Summarising:

The DF label can be removed if the left descendant is equal to 1 or the right one is 0;
If the left descendant is equal to 0 or the right one is 1, the DF label becomes $ and the two descendants are exchanged.

3.2.4 Example

The application of the Simplification rules to the LBDD in Figure 3.2 gives the LBDD represented in Figure 3.3, in which $\&x_2$ has been transformed into x_2 according to the first simplification rule.

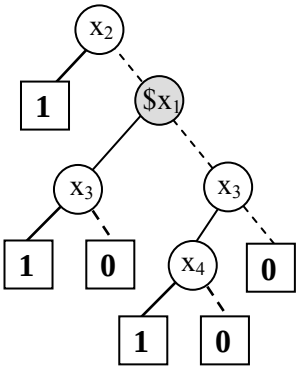


Figure 3.3. LBDD of Figure 3.2 after the Simplification phase

The Static Labelling applied to the BDD for the same function is represented in Figure 3.4 where all variables are labelled as of DF type.

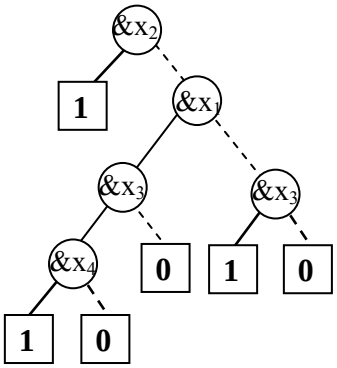


Figure 3.4. BDD with static labelling for Top = $x_2 (x_1 + \bar{x}_3 + \bar{x}_4) + x_3 (\bar{x}_1 + \bar{x}_2 x_4)$

From both Figure 3.3 (LBDD) and Figure 3.4 (classical BDD) the same set of implicants is found, on which the probabilistic quantification can be performed.

Comparing these figures the following considerations can be drawn:

- Dynamic labelling and static labelling have the same number of nodes, which depends on the variables' ordering;

- The representation of the negated $\bar{x}$ variables as \$x allows applying the same algorithms to nodes with SP and SN variable types. Hence we can call SP and SN variables as coherent (i.e. x as monotonic not decreasing; \$x as monotonic not increasing) and DF as non-coherent.

- Variables x_2, x_3 and x_4 are represented in the LBDD as SP variables and only x_1 is represented as SN variable. Therefore, in spite of the fact that all variables in the fault tree are of DF type, none of them is represented as such in the LBDD.
  The number of nodes with DF variables depends on the variable's ordering;

- The determination of the prime implicants for the BDD of Figure 3.4 requires, for each node, the intersection between the left and right descendants as described in Appendix B, whereas this is never necessary on the LBDD in Figure 3.3. The same consideration can be applied for the determination of the unconditional failure and repair frequencies.

- The absence of DF variables in Figure 3.3 assures that all implicants are embedded in the LBDD. This can easily be explained by observing that the Consensus operation, i.e. $x\,y + \bar{x}\,z = y\,z$ is never applied. The Prime Implicants set is found as if the BDD were coherent, i.e. $\{PI\} = \{(x_2)\,(\$x_1\,x_3)\,(x_3\,x_4)\}$, which is equivalent to $\{PI\} = \{(x_2)\,(\overline{x_1}\,x_3)\,(x_3\,x_4)\}$.

3.2.5 Experimental results

The LBDD construction method has been applied to the non-coherent fault trees listed in Table 3.I. Some of these trees are very simple and have been used during the testing phase of the ASTRA 3.0 software (Contini-Matuzas, 2009). Some others have been found in the literature: these are marked with an asterisk. Apart from the five fault trees 13-17, which are real applications of the non-coherence, all remaining were originally real coherent fault trees in which negated sub-trees and XOR gates have been introduced to make them non-coherent.

In Table 3.I each fault tree is characterised by the number of gates and basic events. The table also contains the total number of nodes of the BDD, the number of nodes with DF variables according to the Static Labelling (N_{SL}) and the number of nodes with DF variables obtained with the Dynamic Labelling (N_{DL}). An indicator of the gain in using the dynamic labelling over the static one is given by:

$$G\,\% = \frac{N_{SL} - N_{DL}}{N_{SL}}\,100 \qquad (3.7)$$

The last column contains the Gain factor calculated according to equation (3.7).
Note that the number of DF variables has been calculated on the modularised fault tree. This means that if a module containing e.g. K basic events appears in the modularised tree as a variable of DF type, then it is considered as a single variable.

The following considerations can be drawn from the content of Table 3.I.
- For the considered fault trees the gain ranges between 28% and 100%, but the majority of values are placed between 50% and 70%. The gain depends on the fault tree structure and on the ordering of variables.
- The gain values show the advantage of using the dynamic labelling with respect to the static labelling for the analysis of non coherent fault trees, even though the total gain in computation time can hardly be appreciated due to the intrinsic high efficiency of the BDD approach.

Table 3.I. Comparison between static and dynamic labelling

n.	Filename	Number of gates	Number of events	Number of nodes	DF in BDD (N_{SL})	DF in LBDD (N_{DL})	G (%)
1	Xor-or-xor	7	4	**9**	9	3	**66**
2	Xor+k/n	3	5	**11**	11	5	**54.6**
3	Fussell*	10	9	**13**	3	1	**66**
4	Zhang-Mei*	11	19	**22**	12	6	**50**
5	xorMVxor	9	8	**27**	27	13	**51.8**
6	Crashorg	27	23	**34**	11	3	**72.7**
7	IEEE2*	27	25	**50**	39	7	**82**
8	IEEE1*	36	25	**55**	27	11	**59.2**
9	Editor2	24	22	**57**	17	3	**82.3**
10	Sag-001	48	45	**68**	25	1	**96**
11	Spnot17	30	57	**73**	7	3	**57**
12	Sicu	41	44	**142**	83	14	**83**
13	Util01	82	121	**189**	48	17	**64.5**
14	Util6	82	121	**192**	44	22	**50**
15	Util5	82	121	**205**	57	32	**44**
16	Util02	82	121	**213**	64	35	**45.3**
17	Util0	82	121	**223**	77	38	**50.6**
18	IMM002(xor)	175	64	**243**	85	37	**56.4**
19	IE83-G3	103	190	**244**	76	31	**59.2**
20	IE83-G4	103	190	**251**	82	32	**60.9**
21	Spnot18	103	190	**279**	4	0	**100**
22	Edf2xor	132	215	**323**	106	20	**81.1**
23	Util4	82	121	**324**	63	35	**44**
24	Spnot20	102	169	**337**	90	49	**50**
25	Spnot14	52	96	**343**	6	2	**66**
26	Iveco1	165	392	**477**	43	0	**100**
27	Abtwr014-xor	88	118	**481**	54	23	**57.4**
28	Test1	86	146	**493**	149	68	**54.3**
29	Iveco01 (xor)	165	392	**587**	218	68	**68.8**
30	RIS-001*	229	585	**603**	21	7	**66**

3.3 Complementation of an LBDD

De Morgan rules are applied to complement Boolean functions. If the function is monotone (coherent fault tree) the complementation of a BDD can be obtained by complementing the terminal nodes 0, 1.
Given a function stored in the form of an LBDD (non-coherent tree) its complemented form is obtained by visiting the graph in Top-down mode to apply the following rules to each node, non terminal and terminal (the symbol ¬ means NOT):

$\neg(\&x, F, G) \Rightarrow (\&x, \neg F, \neg G)$

$\neg(x, F, G) \Rightarrow (\$x, \neg G, \neg F)$

$\neg(\$x, F, G) \Rightarrow (x, \neg G, \neg F)$

$\neg 0 = 1$

$\neg 1 = 0$

As a simple example of the application of the above rules consider the following non-coherent function: $\Phi = a\,b\,d + a\,\bar{b}\,c + \bar{c}\,d\,e + a\,d\,e + a\,c\,d$ whose LBDD is shown in Figure 3.5a. The complemented form is represented in Figure 3.5b.

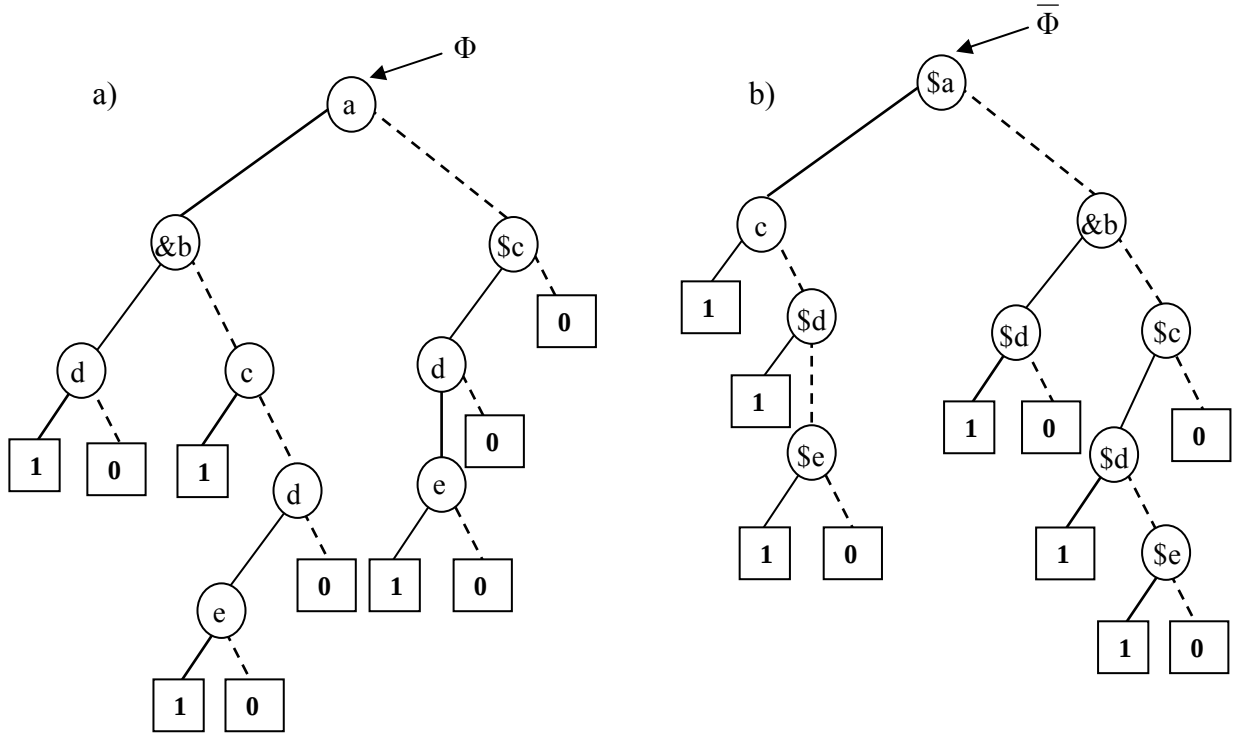


Figure 3.5 Resulting function (on the left) obtained complementing the function on the right

The implicants of the complemented LBDD are identified by applying the same rules as for the non complemented LBDD, i.e. all paths from 1 to the root represent a failure mode.

Hence, the <u>*rules for writing the Implicants*</u> from an LBDD are straightforward. On the path from any terminal node 1 to the root node:

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

The application of these rules to the LBDD in Figure 3.5b, and remembering that $\$x = \bar{x}$, the complementation results in:

$$\bar{\Phi} = \bar{a}\,c + \bar{a}\,\bar{d} + \bar{a}\,\bar{e} + b\,\bar{d} + \bar{b}\,\bar{c}\,\bar{d} + \bar{b}\,\bar{c}\,\bar{e}$$

3.4 Determination of prime implicants

In a non-coherent fault tree the concept of Minimal Cut Set is substituted by that of Prime Implicant. Prime implicants contain variables in negated form and generally, compared with MCS, have higher order.
The rules for determining the prime implicants are given in the Appendix A for the classical BDD.

Let $\phi(x_1, x_2,.., x_i,\dots , x_n) = x_i F + \bar{x}_i G$ be the function of a generic node, with

$F = \phi (x_1, x_2,..,1,\dots , x_n)$ and $G = \phi (x_1, x_2,..,0,\dots , x_n)$.

Let *Prime*( ) be the function that determines the prime implicants of the expression between brackets.

Visiting the L-BDD in bottom up mode the procedure to be applied to each node (with associated variable x_i) to determine the prime implicants from an LBDD is as follows:

If x_i has label "&" then:

$$Prime\{\phi(x_1, x_2,.., x_i,\dots , x_n)\} = x_i R + \$x_i Q + P \quad \text{where}$$
$$P = F \wedge G; \qquad R = F \setminus P; \qquad Q = G \setminus P$$

else

$$Prime\{\phi(x_1, x_2,.., x_i,\dots , x_n)\} = \alpha R + G \text{ where}$$
$$\alpha = (x_i \text{ or } \$x_i) \text{ and } R = F \setminus G$$

" \ " is the operator difference (Rauzy, 1993), i.e. F \ G gives the L-BDD of F in which the combinations of F verified by G are removed.

3.5 Construction of the ZBDD from the LBDD

Negated variables, in many applications of non-coherent fault trees, e.g. safety analysis, represent the working state of components, having success probability very close to 1.
Generally negated variables that appear in a fault tree represent plant conditions that derive from the Top-event definition (Contini et al. 2008)
In practice the reliability analyst is interested in failed components, i.e. in MCS also in the case of non coherent trees. Such MCS can be obtained by removing all negated variables from PIs.

Using the LBDD the set of MCS can be obtained in an efficient way by transforming the LBDD into a coherent OBDD by:
- Simply removing the & labels ($\&x \rightarrow x$);
- Deleting SN variables ($ labelled) by performing the logical OR between the two descending functions.

The elimination of the "$" labels can be implemented as follows.
Let ite($x, F, G) be the node under examination, $F = ite(z, F_1, F_0)$ its left descendant and $G = ite(w, G_1, G_0)$ its right descendant. The following relationships are applied:

if $z < w$ then $ite(\$x, F, G) = ite(z, F_1 \vee G, F_0 \vee G)$

if $z > w$ then $ite(\$x, F, G) = ite(w, F \vee G_1, F \vee G_0)$

if $z = w$ then $ite(\$x, F, G) = ite(z, F_1 \vee G_1, F_0 \vee G_0)$

These operations are then followed by the reduction and minimisation rules typical of monotonic functions. The result is a ZBDD embedding all minimal MCS.

Example.

Consider again the LBDD in Figure 3.3. To obtain the ZBDD embedding all MCS it is sufficient to remove the $\$x_1$ node.
In this case ($z = x_3$; $w = x_3$) we have: $F_1 = 1$; $F_0 = 0$; $G_1 = ite(x_4, 1, 0)$ and $G_0 = 0$.
Since both descending nodes have the same variable x_3, i.e. $z = w$, then:

$ite(x_3, F_1 \vee G_1, F_0 \vee G_0) = ite(x_3, 1 \vee ite(x_4,1,0), 0 \vee 0) = ite(x_3, 1, 0)$.

The resulting ZBDD is represented in Figure 3.6, in which the MCS are $\{(x_2)\ (x_3)\}$.

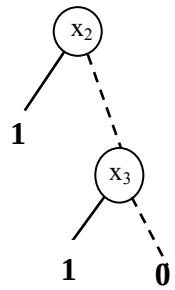


Figure 3.6. Results of the transformation of the LBDD of Figure 3.3 into the BDD embedding all MCS

4. CONSTRUCTION OF A REDUCED ZBDD USING CUT-OFF TECHNIQUES

As described in Section 2 the exact analysis of a fault tree requires the construction of the BDD (or LBDD in case of non-coherent tree) and then the determination of the ZBDD embedding all MCS, from which the SMCS are extracted using the cut-off technique. However, during the analysis of very complex fault trees it may happen that the working memory is not sufficient to store the large BDD (LBDD) structure, since the number of nodes increases exponentially with the complexity of the fault tree.
The direct construction of the reduced ZBDD using cut-off techniques (RZBDD) has been implemented in ASTRA 3.0 allowing the user to always get the results of the analysis of the fault tree even if they are not exact. This algorithm allows the direct construction of the ZBDD; to limit its size a probabilistic (P_{lim}) or/and logical (n_{lim}) truncation levels are applied during the construction process. The main advantage of this approach is much lower memory usage. On the contrary the limitations are that:

- the probabilistic results are (not conservatively) approximated, being based on the set SMCS-1 (see Figure 2.2), which is generally a small percentage of the total number of MCS;
- there is no information about the truncation error, i.e. the probability of all neglected MCS.

4.1 Coherent fault trees

A set of expansion formulas was developed by Jung et al. (2004) for determining the ZBDD embedding all MCS having probability not less than a given threshold.

Note that in the following equations the logical operator OR is represented as + whereas the AND is implied.

Let H and K be the Boolean functions described in the terms of if-then-else structure, i.e. $H = \mathrm{ite}(y, H_1, H_0)$, and $K = \mathrm{ite}(z, K_1, K_0)$. The functions H_1, H_0 and K_1, K_0 are the residues of H and K expanded respectively with respect to y and z.

If y and z are two variables the following equalities holds for coherent fault trees.

If y < z, then:

$$\begin{aligned} H \cdot K &= \mathrm{ite}(y, H_1, H_0) \cdot \mathrm{ite}(z, K_1, K_0) = \mathrm{ite}(y, H_1 K, H_0 K) \\ H + K &= \mathrm{ite}(y, H_1, H_0) + \mathrm{ite}(z, K_1, K_0) = \mathrm{ite}(y, H_1, H_0 + K) \end{aligned} \tag{4.1}$$

If $y = z$, then:

$$\begin{aligned} H \cdot K &= \mathrm{ite}(y, H_1, H_0) \cdot \mathrm{ite}(y, K_1, K_0) = \mathrm{ite}(y, H_1(K_1 + K_0) + H_0 K_1, H_0 K_0) \\ H + K &= \mathrm{ite}(y, H_1, H_0) + \mathrm{ite}(y, K_1, K_0) = \mathrm{ite}(y, H_1 + K_1, H_0 + K_0) \end{aligned} \tag{4.2}$$

In order to maintain the minimal solution in a ZBDD structure, a subsuming operation H\K is performed whenever a gate is solved. The subsuming is recursively performed from the root *ite* to the child *ite* connectives by comparing the left and right *ite* connectives: a cut set in H is deleted if K has its superset. Rauzy (1993) proposed an efficient subsuming operation:

$$H \setminus K = \begin{cases} ite(y, H_1 \setminus K, H_0 \setminus K) & y < z \\ H \setminus K_0 & y > z \\ ite(y, H_1 \setminus (K_1 \text{ or } K_0), H_0 \setminus K_0) & y = z \end{cases} \tag{4.3}$$

The term $H_1 \setminus (K_1 \; or \; K_0)$ denotes that each cut set in H_1 is tested and deleted if K_1 or K_0 has its superset.

The RZBDD construction algorithm, which has been implemented in ASTRA 3.0, was derived from the BDD construction algorithm with truncation, developed by Jung (Jung et al., 2008).
A coherent fault tree is solved with a truncation limit in a bottom-up way by using equations (4.1), (4.2) and (4.3).

4.2 The truncation algorithm

The truncation of cut sets during the construction of the ZBDD can be applied in two different ways, depending of the probability associated to negated events. Consider for instance the combination (a $\bar{b}$) having probability $Q_{a\bar{b}} = q_a\,(1 - q_b)$. Suppose that the probabilistic threshold $P_{lim} = q_a$. If the cut-off is applied on (a $\bar{b}$) then this combination is removed since $Q_{a\bar{b}} < q_a$. This combination is not removed if the basic event is given $q_a = 1$. In ASTRA the second method is applied.

The application of the RZBDD algorithm is now shown with the combination of the two ZBDDs presented in Figure 4.1. The selected ordering of variables is a<b<c<d<e.

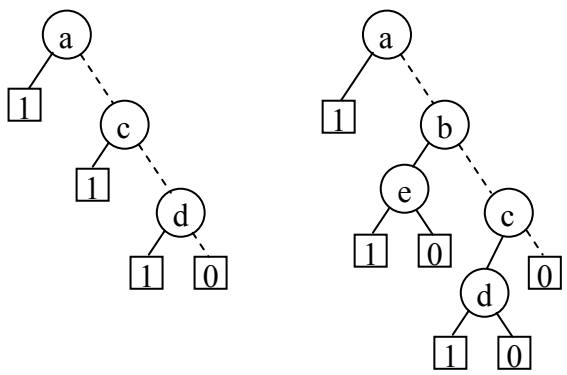


Figure 4.1. Example two ZBDDs to be combined

The above ZBDD represent respectively the function H = a +c + d and K = a+ b e +c d.
The probability 0.1 is assumed for all events. The probabilistic cut-off threshold $P_{lim} = 0.01$.

If two ZBDDs are expressed in the terms of *ite*:

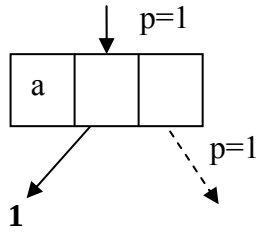
H = ite(a, 1, ite(c, 1, ite(d, 1, 0)))
K = ite(a, 1, ite(b, ite(e, 1, 0), ite(c, ite(d, 1, 0), 0)))

then the RZBDD embedding all MCS of H /\ K can be obtained directly thus avoiding the construction of the complete BDD structure and the successive application of the truncation operations.

The method consists in applying the truncation operation during the construction of the ZBDD as described below.

We start by setting p = 1. This value represents the probability of the path from the current node to the root of the RZBDD; it is determined during the construction of the ZBDD.
By applying (4.2) we initiate the recursive procedure:

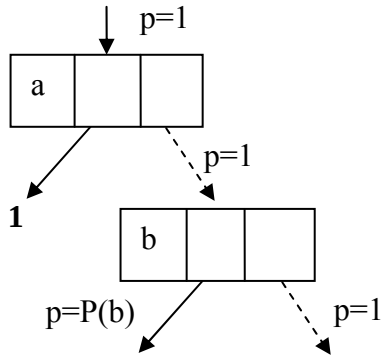
H/\K=ite(a, 1,ite(c, 1, ite(d, 1, 0)) /\ ite(a, 1, ite(b, ite(e, 1, 0), ite(c, ite(d, 1, 0), 0)))



Because the evaluation of the left branch results in a terminal node 1 (since $H_1 \wedge K_1 = 1$), we proceed with the right branch only. In order to construct the right descendant we start the recursive call with p=1 (we pass unaltered the p value in case of the right descendant). Because $p > P_{lim}$ we can proceed with the construction of the ZBDD by applying (4.1).

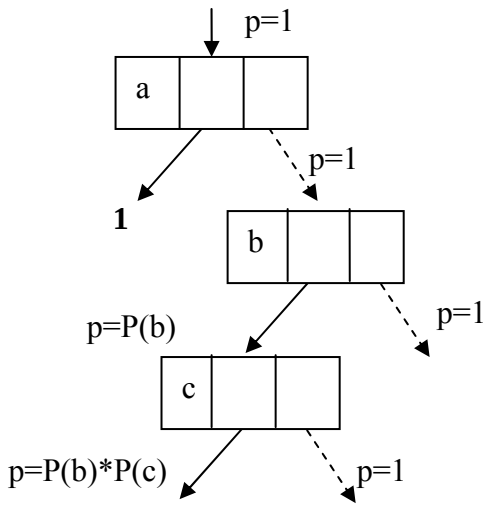
ite(b, ite(e, 1, 0), ite(c, ite(d, 1, 0), 0)) /\ ite(c, 1, ite(d, 1, 0))=

=ite(b, ite(e, 1, 0) /\ ite(c, 1, ite(d, 1, 0)), ite(c, ite(d, 1, 0), 0) /\ ite(c, 1, ite(d, 1, 0)))



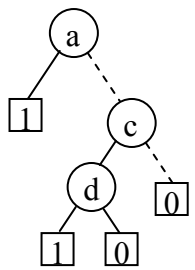
Now we proceed with the construction of the right descendant by starting the recursive call with p=P(b): $p > P_{lim}$ so again we can proceed with the construction through the application of eq. (4.1):

ite(c, 1, ite(d, 1, 0)) /\ ite(e, 1, 0)=ite(c, 1 /\ ite(e, 1, 0), ite(d, 1, 0) /\ ite(e, 1, 0))



By keeping track of p and comparing its value with P_{lim} before starting a new recursive operation (if $p < P_{lim}$ the recursive operation is cancelled and the terminal node 0 is returned) the RZBDD is constructed.

As can easily by verified the final RZBDD for the example at hand is $f = H \wedge K = a + c\ d$.



The above example shows application of probabilistic cut-off P_{lim} only. Application of logical cut-off n_{lim} during the construction of ZBDD is identical to the application of probabilistic cut-off:
- We start the ZBDD construction by setting n = 0. This value represents the depth of the path from the current node to the root of the RZBDD (i.e. MCS order); it is determined during the construction of the ZBDD;
- n value is increased by 1 and passed to the recursive call for the construction of the nodes left descendant; and in case of the right descendant n is passed unaltered;
- By keeping track of n and comparing its value with n_{lim} before starting a new recursive operation (if $n > n_{lim}$ the recursive operation is cancelled and the terminal node 0 is returned) the RZBDD is constructed.

4.3 Non-Coherent fault trees

To determine the RZBDD of a non-coherent fault tree embedding all MCS (not prime implicants) the negated events are considered for removing impossible cut sets, i.e. combinations of basic events that contain the same variable in both positive and negated forms.
Therefore the first step of the analysis procedure implemented in ASTRA is the removal from the fault tree all non repeated negated basic events. Indeed these events are not needed to delete impossible failure combinations.
Rules applied to not repeated events $x:
– If $x descends from an OR gate then y + $x = 1
– If $x descends from an AND gate then y · $x = y
Gates OR always verified and gates AND with single input are then properly removed.

On an LBDD, equations (4.1) are applicable when combining different variables; in these cases the variable's type has no effect.

The equations for combining two occurrences of the same variable with equal or different labels are listed below.

4.3.1 Same variables, same labels

1) Combination of $H = ite(x, H_1, H_0)$ with $K = ite(x, K_1, K_0)$

If variables are of type SP (not labelled) equations (4.2) are obviously applicable.

2) Combination of $H = ite(\$x, H_1, H_0)$ with $K = ite(\$x, K_1, K_0)$

If variables are of type SN equations (4.2) are still applicable since in the ite data structure $x is represented in the same way as x. Indeed, given H = ite($x, H_1, H_0) and K = ite($x, K_1, K_0), we have:

$$H \quad K = ite(\$x, H_1, H_0) \quad ite(\$x, K_1, K_0) = ite(\$x, H_1(K_1 + K_0) + H_0K_1, H_0K_0)$$
$$H + K = ite(\$x, H_1, H_0) + ite(\$x, K_1, K_0) = ite(\$x, H_1 + K_1, H_0 + K_0) \tag{4.4}$$

3) Combination of H = ite(&x, H_1, H_0) with K = ite(&x, K_1, K_0)

If variables are of type DF then, given H = ite(&x, H_1, H_0) and K = ite(&x, K_1, K_0) we have:

$$H \cdot K = ite(\&x\ H_1, H_0) \cdot ite\ (\&x\ K_1, K_0) = ite(\&x, H_1\ K_1, H_0\ K_0)$$
$$H + K = ite(\&x\ H_1, H_0) + (\&x\ K_1, K_0) = ite(\&x, H_1 + K_1, H_0 + K_0) \tag{4.5}$$

Indeed,

$$H \quad K = (x\ H_1 + \bar{x}\ H_0)\ (x\ K_1 + \bar{x}\ K_0) = x\ H_1\ K_1 + \bar{x}\ H_0\ K_0$$
$$H + K = (x\ H_1 + \bar{x}\ H_0) + (x\ K_1 + \bar{x}\ K_0) = x\ (H_1 + K_1) + \bar{x}\ (H_0 + K_0)$$

which, represented in terms of &x, give equations (4.5)

4.3.2 Same variables, different labels

The following three cases are possible.

1) Combination of H = ite(x, H_1, H_0) with K = ite($x, K_1, K_0)

$$H \cdot K = ite(x\ H_1 + H_0) \cdot ite(\$x\ K_1 + K_0) = ite(\&x, H_1\ K_0, H_0\ (K_1 + K_0))$$
$$H + K = ite(x\ H_1 + H_0) + (\$x\ K_1 + K_0) = ite(\&x, H_1, K_1 + K_0 + H_0) \tag{4.6}$$

Proof.

$$H \quad K = (x\ H_1 + H_0)\ (\bar{x}\ K_1 + K_0) = x\ H_1\ K_0 + \bar{x}\ H_0\ K_1 + H_0\ K_0 =$$
$$= x\ (H_1\ K_0 + H_0\ K_0) + \bar{x}\ (H_0\ K_1 + H_0\ K_0) = (\&x, H_1\ K_0 + H_0\ K_0, H_0\ K_1 + H_0\ K_0)$$

Since the objective of the analysis is the determination of the set of MCS the above formula can be reduced by removing the non minimal terms from the left descendant $(H_0\ K_0)$, giving the first of the equations (4.6).

An analogous procedure can be applied to prove the second of equations (4.6) for the OR operator.

2) Combination of H = ite(&x, H_1, H_0) with K = ite(x, K_1, K_0)

$$H \cdot K = ite(\&x\ H_1 + H_0) \cdot ite(x\ K_1 + K_0) = ite(\&x, H_1\ (K_1 + K_0), H_0\ K_0)$$
$$H + K = ite(\&x\ H_1 + H_0) + ite(x\ K_1 + K_0) = ite(\&x, H_1 + K_1, H_0 + K_0) \tag{4.7}$$

Proof.

$H \cdot K = (x\,H_1 + \bar{x}\,H_0)\,(x\,K_1 + K_0) = x\,H_1\,K_1 + x\,H_1\,K_0 + \bar{x}\,H_0\,K_0$ which, represented in terms of &x, gives the first of the equations (4.7).

$H + K = (x\,H_1 + \bar{x}\,H_0) + (x\,K_1 + K_0) = x\,(H_1 + K_1) + \bar{x}\,H_0 + K_0 = x\,(H_1 + K_1 + K_0) + \bar{x}\,(H_0 + K_0)$
Since the objective of the analysis is the determination of the set of MCS the above formula can be simplified by removing the non minimal term K_0 from the left descendant, giving the second of the equations (4.7).

3) Combination of H = ite(&x, H_1, H_0) with K = ite($x, K_1, K_0)

$$H \cdot K = \text{ite}(\&x\ H_1 + H_0) \cdot \text{ite}(\$x\ K_1 + K_0) = \text{ite}(\&x,\ H_1\,K_0,\ H_0\,(K_1 + K_0))$$
$$H + K = \text{ite}(\&x\ H_1 + H_0) + \text{ite}(\$x\ K_1 + K_0) = \text{ite}(\&x,\ H_1,\ H_0 + K_1 + K_0) \qquad (4.8)$$

Proof.
$H \cdot K = (x\,H_1 + \bar{x}\,H_0)\,(\bar{x}\,K_1 + K_0) = x\,H_1\,K_0 + \bar{x}\,H_0\,K_1 + \bar{x}\,H_0\,K_0 = x\,H_1\,K_0 + \bar{x}\,H_0\,(K_1 + K_0)$
which, represented in terms of &x gives the first of the equations (4.8).

$H + K = (x\,H_1 + \bar{x}\,H_0) + (\bar{x}\,K_1 + K_0) = x\,(H_1 + K_0) + \bar{x}\,(H_0 + K_1 + K_0) = x\,H_1 + \bar{x}\,(H_0 + K_1 + K_0)$

The application of the above formulas is followed by the application of the cut-off threshold and subsuming. As a consequence of this operation the resulting node of the RZBDD could contain one or both descendants equal to 0; hence the following rules are applied:
(&x, 0, G) ⇒ ($x, G, 0)
(&x, F, 0) ⇒ (x, F, 0)
(&x, 0, 0) ⇒ 0
($x, 0, G) ⇒ G
($x, F, 0) ⇒ ($x, F, 0)
($x, 0, 0) ⇒ 0
(x, 0, G) ⇒ G
(x, F, 0) ⇒ (x, F, 0)
(x, 0, 0) ⇒ 0

At the end of the analysis of the fault tree the final operation is to remove from the LBDD all nodes with negated variables. To this aim the algorithm described in section 3.5 is applied.

4.4 Application example

Filename: EDF9203 downloaded from http://iml.univ-mrs.fr/~arauzy/aralia/benchmark.html.
This coherent fault tree has 362 basic events and 707 gates. All components are assumed to be characterised by the same probabilities: q = 1.0e-03.
The total number of MCS calculated by ASTRA is equal to 20,807,446.
The Top event unavailability is equal to: Q_{UB} = 4.565136e-02.

The number of MCS obtained using different cut-off levels is provided in the following Table together with the unavailability, the percentage difference on Q_{UB}, and the peak memory used during the analysis.

Table 4.I Summary of results obtained applying the RZBDD module of ASTRA 3.0

Cut-off level	Number of MCS	Q	(Q / Q_{UB}) 100	ITE peek
1e-03	37	3.700000e-02	81.05 %	4,041
1e-06	8,368	4.533100e-02	99.30 %	20,848
1e-09	327,178	4.564981e-02	100	83,793
1e-12	1,873,598	4.565136e-02	100	221,018
1e-15	3,580,162	4.565136e-02	100	444,296
1e-18	5,413,130	4.565136e-02	100	663,475
1e-21	8,809,758	4.565136e-02	100	1,254,806
1e-24	13,381,950	4.565136e-02	100	1,922,312
1e-27	18,364,022	4.565136e-02	100	2,191,710
1e-30	20,500,566	4.565136e-02	100	3,363,473
1e-33	20,798,206	4.565136e-02	100	3,669,406
1e-36	20,807,446	4.565136e-02	100	3,673,573

The ITE peek size during the execution of the analysis without cut-off was 1,634,239. The peak values with the RZBDD module vs. the probabilistic cut off value is provided in the fifth column.

The Top-event upper bound probability converges to the exact value very fast. The exact value is reached using P_{lim} = 1e-12 cut-off value. With this cut-off level the peek size of the ITE record table is 221,018, in comparison to the 1,634,239 used by LBDD module. The plot of the Q_{UB} vs. Cut-off level showing the fast convergence is represented in the following figure.

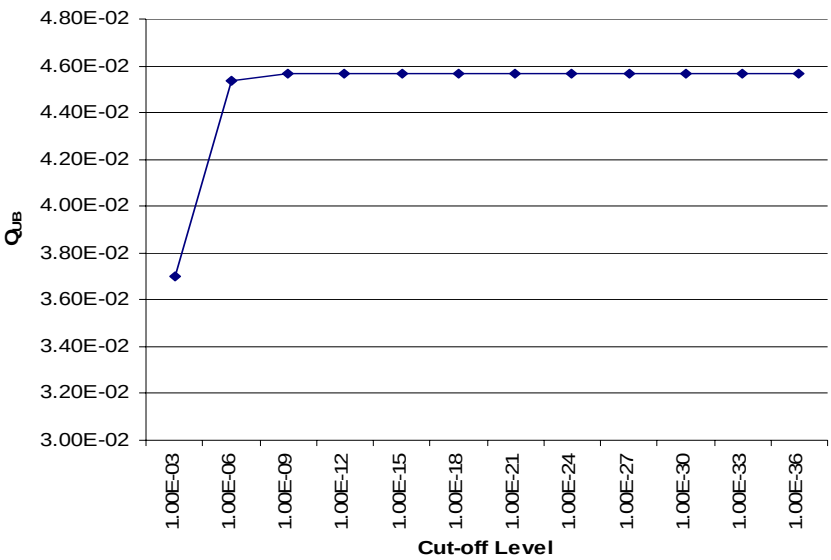


Figure 4.1 Plot of the unavailability vs. the probabilistic cut-off value for the fault tree EDF9203

5. PROBABILISTIC ANALYSIS PROCEDURE

The quantification of the LBDD allows obtaining the exact values of the RAM parameters of interest. This section provides information about the equations used together with some simple clarification examples. Other examples of application of the probabilistic analysis methods implemented in ASTRA 3.0 are described in the test report (Contini-Matuzas, 2009).
ASTRA allows the user to determine: Unavailability; Expected number of failures and repairs; Unreliability (upper bound); Importance measures of basic events.

The probabilistic analysis starts with the determination of the unavailability, the unconditional failure and repair frequencies of basic events. Then for each simple module the unavailability is calculated together with the probability of critical state for failure and repair of basic events within the module. These results are finally used for the quantification on the Top-module leading to the final results.

5.1 Notation.

λ	Failure rate (constant)
μ	Repair rate (constant)
τ	Repair time ($\tau = 1 / \mu$)
θ	Time between tests
θ_0	First time to test
$\omega(t)$	Unconditional failure frequency
$\upsilon(t)$	Unconditional repair frequency
$q(t)$	Basic event unavailability at time t
$q(0)$	Basic event unavailability at time t=0
$\Lambda_T(t)$	Top event conditional failure frequency
$Q_T(t)$	Top event Unavailability at time t
$Q_T(0)$	Top event Unavailability at t=0
$W_T(t)$	Top event Expected number of failures in 0-t
$V_S(t)$	Top event Expected number of repair in 0-t
$F_T(t)$	Top event Unreliability in 0- t
$Q_{Cj}(t)$	Top event Unavailability at time t for the j-th MCS
$Q_{Cj}(0)$	Unavailability at t=0 for the j-th MCS
$W_{Cj}(t)$	Expected number of failures in 0-t for the j-th MCS
$F_{Cj}(t)$	Unreliability in 0- t for the j-th MCS
MTBF	Mean Time Between failures
MTTR	Mean Time To Repair
MTTF	Mean Time To failure
MTTFF	Mean Time To First Failure
BE	Basic event
MCS	Minimal Cut Set
SMCS	Significant MCS
Ne	Number of basic events of the fault tree
n	Number of basic events in an MCS/SMCS
T	Mission time
$p^f_x(t)$	Probability of failure critical state for the generic event x
$p^r_x(t)$	Probability of repair critical state for generic event x
$IC_x(t)$	Criticality index at of event x time t
$RAW_x(t)$	Risk Achievement Worth of event x at time t
$RRW_x(t)$	Risk Reduction Worth of event x at time t
IS_x	Structural criticality of event x

5.2 Probabilistic quantification of basic events

The probabilistic quantification of the fault tree variables is based on the following assumptions:

– A basic event (BE) represents the failure mode of a component. Basic event and component failure are used as interchangeably. BEs are binary and statistically independent, e.g. the failure or repair of a component does not have any influence on the failure probability of any other component; the unique exception to the independence is the sequence of events that is considered in the extended INH gate.

– Failure and repair times are exponentially distributed, i.e. failure and repair rate are constant.
  Since λ is constant, then the mean time to failure MTTF = 1 / λ.
  Since μ is constant, then the mean time to repair MTTR = 1 / μ.
  The repair makes the component as good as new.

5.2.1 Unavailability of basic events

The time specific unavailability Q(t) of an item (component, subsystem, system) is the probability that the item is failed at time t.
ASTRA 3.0 allows the use of four different types of basic events:

- not repairable
- on-line maintained
- periodically tested/inspected
- acting on demand

The equations that follow are calculated by ASTRA at times t determined on the basis of the number of time points selected by the user and equally distributed in the mission time interval 0-T plus the time points corresponding to discontinuities of the unavailability function when periodically tested/inspected events are considered.

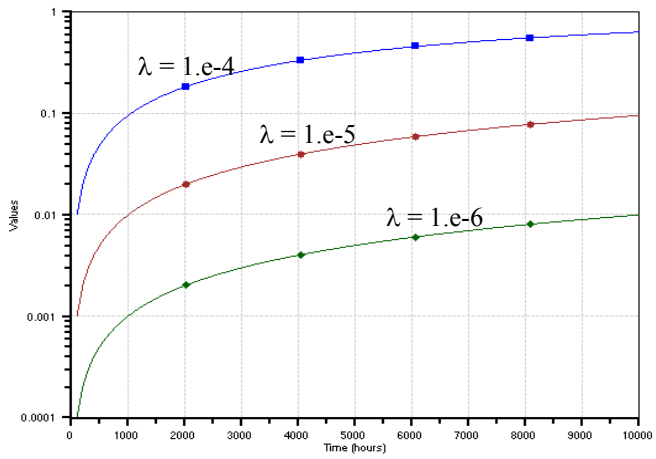
Not repairable components.

These are components that, in case of failure, cannot be repaired during the mission time interval. They are characterized by the failure rate λ.

The unavailability of the component at time t=0 may also be different from 0.

$$q(t) = 1 - e^{-\lambda t} + q(0)\, e^{-\lambda t} \qquad (5.1)$$

The following plot, produced by the ASTRA chart capability, represents the unavailability of three non repairable components with different failure rate.

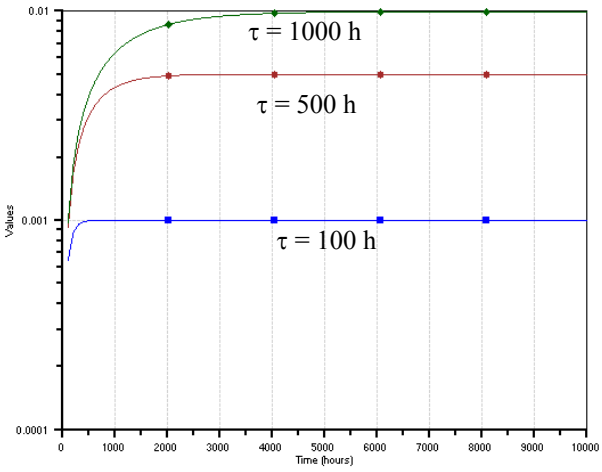


On-line maintained components

The basic hypothesis is that the repair process immediately starts as soon as the component fails. The repair, which is preformed during the mission time, i.e. while the system works, makes the component as good as new. The required parameters are λ and τ (repair time). The unavailability of the component at time t = 0 may also be different from 0.

$$q(t) = \frac{\lambda}{\lambda + \mu}(1 - e^{-\lambda t}) + q(0)\, e^{-(\lambda+\mu)t} \qquad (5.2)$$

The following plot represents the unavailability of a repairable component with λ = 1.e-5 and for different values of the mean repair time τ.



Equation 5.2 tends to the steady-state value $\omega_{ss} = \nu_{ss} = \dfrac{\lambda}{\lambda + \mu}$ after about 4 times the value τ.

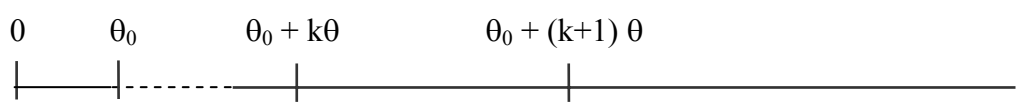
Periodically tested components

These are the typical components of safety systems whose failure can be revealed only through test/inspections. The test/inspection is performed without effects on the component state. The required parameters are λ and θ (inspection interval), θ_0 (first inspection interval), and τ (repair time).

Hypotheses adopted in the implemented model:
- the test is perfect, i.e. the test does not fail the component;
- the unavailability due to test is negligible compared with the mean unavailability between tests.

The applied unavailability equation depends on the value of the repair time τ compared with the test interval θ.

1) The repair time τ is <u>negligible</u>, i.e. $\tau < 10^{-3}\theta$ and $\theta_0 \neq \theta$



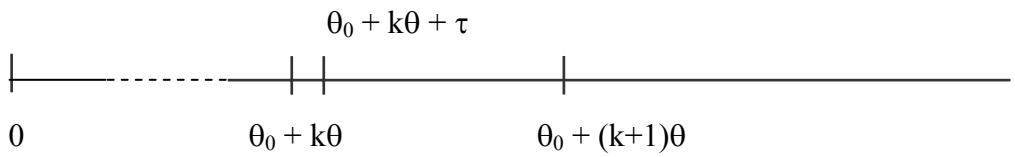
for $0 \le t < \theta_0$ then $q(t) = 1 - e^{-\lambda t}$ (5.3.a)

for $\theta_0 + k\theta \le t < \theta_0 + (k+1)\,\theta$ and k = 0,1,2,…

$$q(t) = 1 - e^{-\lambda(t-\theta_0+k\,\theta)} \qquad (5.3.b)$$

$q(t) = 1 - \exp\,[-\,\lambda\,(t - (\theta_0 + k\theta))]$

2) The repair time τ is <u>not negligible</u>, i.e. $\tau \geq 10^{-3}\,\theta$
When the test reveals that the component is failed, the basic hypothesis in this case is that the repair starts immediately after the test and lasts for τ time units. The repair makes the component as good as new.



The unavailability is given by:

for $0 \leq t < \theta_0$

$$q(t) = 1 - e^{-\lambda\,t}$$

for $\theta_k^* \leq t < \theta_k^* + \tau$

$$q(t) = \frac{\tau}{\theta}\,q(\theta_k) + (1 - q(\theta_k))\,(1 - e^{-\lambda(t-\theta_k^*)}) \qquad (5.3.c)$$

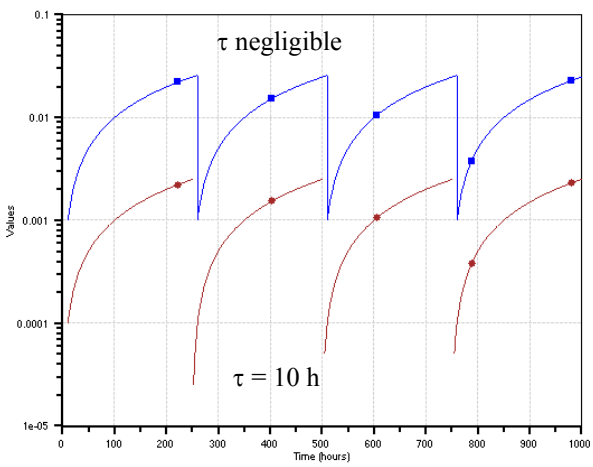
with $q(\theta_k) = 1 - e^{-\lambda\,\theta_k}$

for $\theta_k^* + \tau \leq t < \theta_{k+1}^*$ and $k = 0,1,2,\ldots$

$$q(t) = 1 - e^{-\lambda\,(t-(\vartheta k^* + \tau))} \qquad (5.3.d)$$

where $\theta_k^* = \theta_0 + k\theta$ and $\theta_{k+1}^* = \theta_0 + (\theta+1)\vartheta$

The following plot shows the unavailability of a tested component with λ = 1.e-4 test interval of 250 h, and with negligible – not negligible repair time τ.



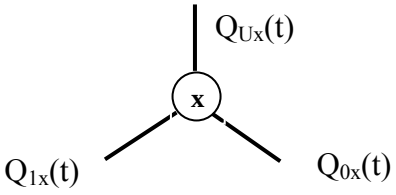
<u>Components acting on demand</u>

$$q(t) = q(0) = \text{const.} \qquad (5.4)$$

5.3 Unavailability analysis

The algorithm for determining the unavailability is applied first to all simple modules and then to the Top-module, all represented as LBDD.

The following figure represents the generic node of an LBDD.



The exact value of the unavailability $Q_{Ux}(t)$ is given by:

$$Q_{Ux}(y) = q_x(t)\, Q_{1x}(t) + [1 - q_x(t)]\, Q_{0x}(t) \qquad (5.5)$$

$Q_{1x}(t)$ and $Q_{0x}(t)$ are respectively the unavailability of the left and right branches of the node and $q_x(t)$ is the unavailability of the event x.

Equation (5.5) is recursively applied to all nodes of the LBDDs of all simple modules and of the Top-module by visiting them according to the Depth-first mode (Bottom-up approach).

Equation (5.5) is applicable for any variable type.

For terminal nodes :
- Node 1: $Q_1(t) = 1$;
- Node 0: $Q_0(t) = 0$.

NOTES:

1. The unavailability Q(t) is calculated as a function of time due to the need to correctly take into account the discontinuities of the unavailability function due to the presence of tested components.

2. Calculating Q(t) simply means that equation (5.5) is applied to all LBDD nodes as many times as the number of time points in which the unavailability function has discontinuities.

3. If all components have unavailability at t = 0 different from zero, then the top event unavailability Q(0) > 0.

4. The mean value of the unavailability is important when the system unavailability function contains discontinuities due to the presence of tested components. In these cases in fact the unavailability at the mission time T can be misleading.

5. Equation 5-5 is applicable to coherent as well as to non coherent fault trees.

6. If the "Approximate" analysis option of ASTRA is selected, which means that the cut-off is applied during the construction of the RZBDD, then the unavailability upper bound is determined as $Q_{Ux}(t) = q_x(t)\, Q_{1x}(t) + Q_{0x}(t)$.

As an example of the application of the above equations consider the determination of the unavailability the following function, Top = $[x_2 (x_1 + \bar{x}_3 + \bar{x}_4)] + [x_3 (\bar{x}_1 + \bar{x}_2 \ x_4)]$, containing all variables of DF type. For this system the prime implicants are: (x_2), $(\bar{x}_1 \ x_3)$, $(x_3 \ x_4)$.

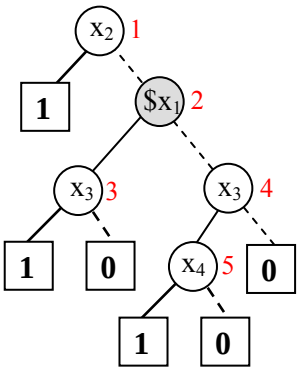


Figure 5.1 LBDD of the function Top = $[x_2 (x_1 + \bar{x}_3 + \bar{x}_4)] + [x_3 (\bar{x}_1 + \bar{x}_2 \ x_4)]$

Basic events' data are shown in Table 5.I and their plot vs. time is given in Figure 5.2.

Table 5.I. Data characterising the basic events of the LBDD represented in Figure 5.1.

Var	**Type**	**λ**	**τ**	**q**
x_1	Not repairable	1.e-5		
x_2	On-line maintained	1.e-6	200	
x_3	On-line maintained	1.e-4	20	
x_4	On demand			0.001

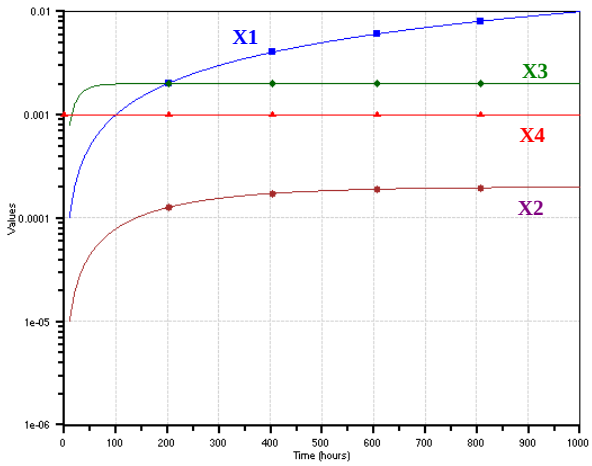


Figure 5.2 Components' unavailability for the example in Figure 5.1

The expressions of the unavailability for the different nodes of the BDD are given in Table 5.II (bottom up visit). The first column contains the number of the node represented in figure 8 on the right; the sequence from first row to the last row represents the visiting order. The last column of the last node (x_2) contains the expression of the Top event unavailability.

Table 5.II. Nodes- unavailability expression for the BDD in Figure 5.1

Node	**Var**	q_x	Q_{1x}	Q_{0x}	$Q_{tot} = q_x\, Q_{1x} + p_x\, Q_{0x}$
3	x_3	q_3	1	0	q_3
5	x_4	q_4	1	0	q_4
4	x_3	q_3	q_4	0	$q_3\, q_4$
2	$\$x_1$	p_1	q_3	$q_3\, q_4$	$p_1\, q_3 + q_1\, q_3\, q_4$
1	x_2	q_2	1	$p_1\, q_3 + q_1 q_3\, q_4$	$q_2 + p_2\,(p_1\, q_3 + q_1 q_3\, q_4)$

The following figure represents the unavailability of the function in fig. 5.1

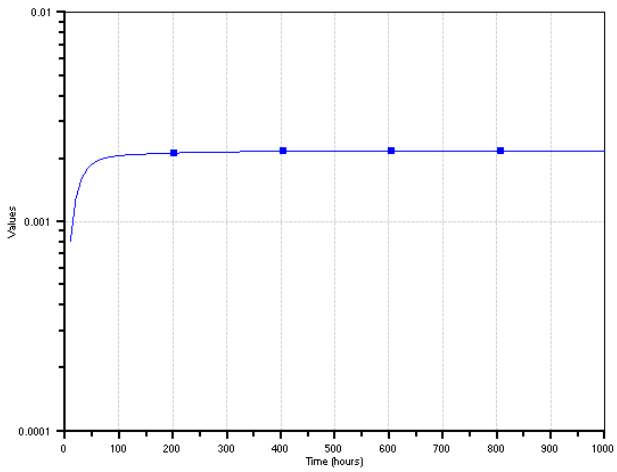


Figure 5.3 Top event Unavailability for the example in Figure 5.1.

As a second example we consider the determination of the unavailability of a 2 out of 3 system made up by equal components.
Top = a b + a c + b c
The system contains equal components characterized by:

- failure rate = 1.e-4
- repair time = 0.6 h
- test interval = 300 h

Components are tested one after the other at regular intervals of time. If θ is the test interval, then the first component is tested at t = 0 (first test at $\theta_0 = 0$), the second at $t = \theta / 3$ ($\theta_0 = 100$) and finally the third at $t = 2\,\theta / 3$ ($\theta_0 = 200$).

The plot of the unavailability of the three components is given in Figure 5.4.

The results of the Unavailability analysis of this system are provided in Figure 5.5, where both the time dependent and the mean values are represented. The mean value is given by:

$$Q_{mean} = \frac{1}{t}\int_0^t Q(x)\, dx \quad \text{for } 0 \le t \le T$$

The mean value of the system is equal to Q_{mean} = 5.14e-4.

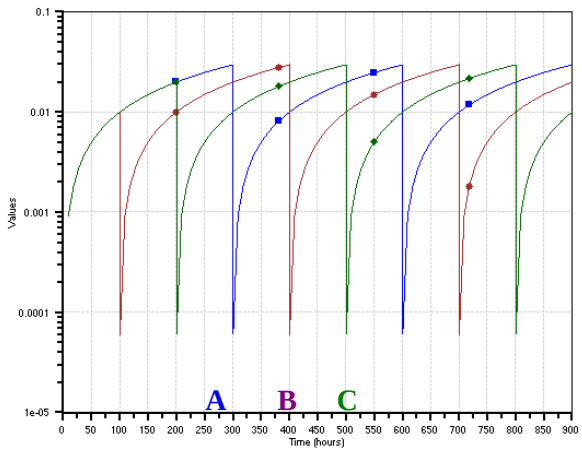


Figure 5.4 Components' unavailability for staggered testing

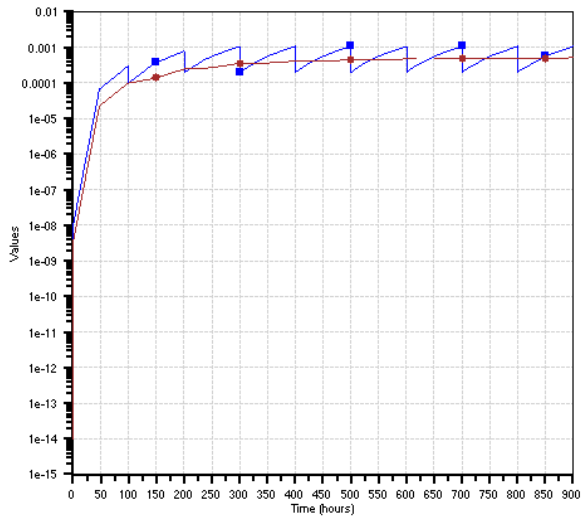


Figure 5.5 Time dependent and mean system unavailability for staggered testing of 2/3

5.4 Frequency analysis

The failure frequency is another important parameter in system analysis. It can be used together with the unavailability to determine the mean number of times the system fails within the mission time interval 0 - T.

If the Top-event describes a catastrophic situation, then the parameter of interest is the Reliability R(t), i.e. the probability that the system works from 0 to the mission time T *without* failure. In fault tree analysis we work on failure events, so the parameter of interest becomes the Unreliability F(t).
Whereas the unreliability of systems with not repairable components is equal, by definition, to the unavailability, the unreliability of systems with repairable components cannot be exactly determined using the fault tree methodology (Clarotti, 1981). Several bounds giving conservative values of the system unreliability can be found in the literature.
In ASTRA two bounds have been implemented:

- Expected Number of Failures (ENF);
- Vesely equation.

The application of these bounds requires the determination of the unconditional failure and repair frequencies.

5.4.1 Unconditional failure and repair frequencies of basic events

The determination of the unconditional failure frequency is performed for each simple module and for the Top-module.

The time specific unconditional failure frequency $\omega(t)dt$ is defined as:

$\omega(t)$ dt: *the probability that the item fails in (t,t+dt) given that it was working at time 0.*

It is important not to confuse the unconditional failure frequency with the failure rate. In the case of $\omega(t)$ the component was good at t=0 and may have failed before t, whereas $\lambda(t)$ requires that the component has never failed before t.

In the exponential case it can be proved that the following equation holds:

$$\omega(t)\, dt = [\, 1 - q(t)\,]\, \lambda\, dt$$

where q(t) is the component unavailability and λ the constant failure rate.

The failure frequency is characteristic of a component (repairable or not) whose failure represents an initiating event of an accident sequence. It could also be calculated for tested events but this is not realistic since these components perform protective actions and so they cannot be initiating events.
The time specific unconditional repair frequency $\nu(t)$ is defined as:

$\nu(t)dt$: *the probability that the item is repaired in (t,t+dt) given that it was working at time 0.*

In the exponential case (λ, μ constants) it can be proved that the following equation holds:

$$\nu(t)dt = q(t)\, \mu\, dt$$

where q(t) is the component unavailability and μ the repair rate (h^{-1}).
For non-repairable components $\nu(t) = 0$ since $\mu = 0$.

Also events characterised by means of a constant unavailability value have $\omega(t) = 0$ and $\nu(t) = 0$.

The following figure shows the failure and repair frequencies of a repairable component.

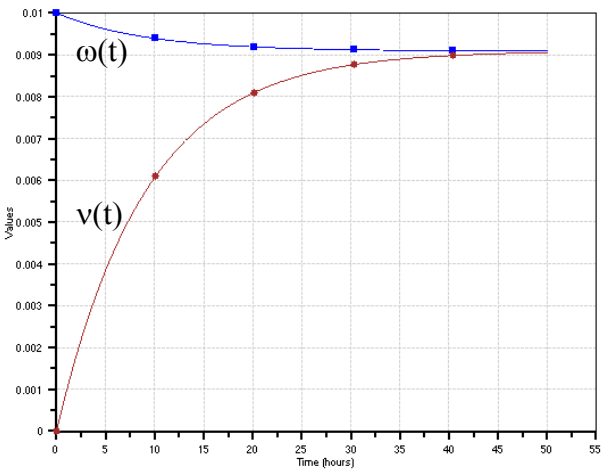


Figure 5.6 Unconditional failure and repair frequencies of a generic repairable component

The failure frequency decreases and the repair frequency increases following the variation of the unavailability with time. They tend to the constant value $\omega_\infty = \nu_\infty = \frac{\lambda\,\mu}{\lambda+\mu}$ when the unavailability reaches the steady state condition.

Note that in ASTRA in case of frequency analysis all tested components are transformed into components acting on demand, whose unavailability q(t), $0 < t \le T$, follows the discontinuities due to testing operations. Thus for these events $\omega(t) = 0$ and $\nu(t) = 0$.

Appendix C contains a brief description of the unconditional frequency calculation for the basic logical relationships between two events.

5.4.2 Unconditional frequencies of simple modules

Module in positive form

Let $\phi_K(\mathbf{x_K})$ be the logical function of the K-th simple module containing the vector of basic events $\mathbf{x_K} = \{ x_1, x_2, .., x_{nk} \}$. Each basic event is therefore characterised by its state variable. x = 0 means that the basic events is not verified (component working) and x =1 the basic events is verified.

$$p^f_{xK}(t) = \Pr\{\phi_K|_{x=1} \wedge \overline{\phi_K|_{x=0}}\} \qquad (5.6)$$

is the probability that the generic event $x \in \mathbf{x_K}$ is critical, i.e. the module K is verified if x = 1 and is not verified if x = 0;

$$p^r_{xK}(t) = \Pr\{\phi_K|_{x=0} \wedge \overline{\phi_K|_{x=1}}\} \qquad (5.7)$$

is the probability that the generic event $x \in \mathbf{x_K}$ in complemented form is critical, i.e. the module is verified if x = 0 and is not verified if x = 1;

If the event x appears in the positive form only, then $p^r_{xK}(t) = 0$
If the event x appears in the negated form only, then $p^f_{xK}(t) = 0$.
These values are calculated for each event in each simple module.

The contribution of the generic event x to the unconditional failure intensity of the K-th simple module of interest is given by:

$$\omega_{xK}(t) = p^f_{xK}(t)\,\omega_x(t)$$

For monotonic functions, summing up the contribution from all N_{ek} events of the module one gets the unconditional failure intensity $\omega_K(t)$ of $\phi_K(\mathbf{x_K})$:

$$\omega_K(t) = \sum_{x=1}^{Nek} p^f_{xK}(t)\,\omega_x(t)$$

In this case $p^f_x(t) = IB_x(t)$, i.e. the Birnbaum index of importance of x.

For non-monotonic functions also the repair of a critical component may verify the module. The contribution of the generic event x to the module failure intensity is given by $\omega_K(t) = \sum_{x=1}^{Nek} p^r_{xK}(t)\,\nu_x(t)$

Therefore the unconditional failure intensity of a non-coherent function $\phi(\mathbf{x})$ is given by:

$$\omega_K(t) = \sum_{x=1}^{Nek} [p^f_{xK}(t)\,\omega_x(t) + p^r_{xK}(t)\,\nu_x(t)] \tag{5.8}$$

Analogously, the unconditional repair intensity of a non-coherent function $\phi(\mathbf{x})$ is given by

$$\nu_K(t) = \sum_{x=1}^{Nek} [p^f_{xK}(t)\,\nu_x(t) + p^r_{xK}(t)\,\omega_x(t)] \tag{5.9}$$

Module in negated form

The Top-module of a modularised non-coherent fault tree may contain simple modules in both forms, positive and negated. In these cases ASTRA calculate the parameters of the negated occurrence of the module from the results of the analysis of the module in positive form by means of equations (5.5), (5.8) and (5.9).

Indeed, consider the following non-coherent function (x is of DF type): $\phi(\mathbf{x}) = x\,\phi_1 + \bar{x}\,\phi_0$
Since x = 1 implies $\bar{x} = 0$ and x = 0 implies $\bar{x} = 1$, it follows that $\phi_{1x} = \phi_{0\bar{x}}$ and $\phi_{0x} = \phi_{1\bar{x}}$

Let $\phi(\mathbf{x})$ be the non-coherent function of the module of interest and $\psi(\mathbf{x})$ its negation, i.e. $\psi(\mathbf{x}) = \overline{\phi(\mathbf{x})}$
Then $\psi(\mathbf{x}) = x\,\psi_1 + \bar{x}\,\psi_0$ where $\psi_1 = \overline{\phi_1}$ and $\psi_0 = \overline{\phi_0}$

It can easily be proved that:

$$Q_{\bar{\phi}}(t) = 1 - Q_\phi(t)$$

$$\omega_{\bar{\phi}}(t) = \nu_\phi(t)$$

$$\nu_{\bar{\phi}}(t) = \omega_\phi(t)$$

The first equation is obvious.

Concerning $\omega_{\bar{\phi}}(t)$ and $\nu_{\bar{\phi}}(t)$ it can be noticed that for any variable $x \in \psi(\mathbf{x})$ we have that $p^f_x(t) = \Pr\{\psi_1 \wedge \overline{\psi_0}\}$ calculated on $\psi(\mathbf{x})$ is equal to $\Pr\{\overline{\phi_1} \wedge \phi_0\} = p^r_x(t)$ calculated on $\phi(\mathbf{x})$.

Analogously, for any variable $\bar{x} \in \psi(\mathbf{x})$ we have that $p^r_x(t) = \Pr\{\overline{\psi_1} \wedge \psi_0\}$ calculated on $\psi(\mathbf{x})$ is equal to $\Pr\{\phi_1 \wedge \overline{\phi_0}\} = p^f_x(t)$ calculated on $\phi(\mathbf{x})$.

Therefore:

$$\omega_{\bar{\varphi}}(t) = \sum_{x=1}^{Nek} [p^r_{xK}(t)\,\omega_x(t) + p^f_{xK}(t)\,\nu_x(t)] = \nu_\varphi(t) \tag{5.10}$$

$$\nu_{\bar{\varphi}}(t) = \sum_{x=1}^{Nek} [p^f_{xK}(t)\,\omega_x(t) + p^r_{xK}(t)\,\nu_x(t)] = \omega_\varphi(t) \tag{5.11}$$

Hence, once the results of the analysis of a function $\phi(\mathbf{x})$ have been obtained, the determination of the parameters of interest for $\overline{\phi(\mathbf{x})}$ can be derived in a straightforward way as summarised in the table below.

Table 5.III Relationships between reliability parameters of $\phi(\mathbf{x})$ and $\overline{\phi(\mathbf{x})}$.

Parameters of $\phi(\mathbf{x})$	Parameters of $\overline{\phi(\mathbf{x})}$
$Q_\phi(t)$	$Q_{\bar{\phi}}(t) = 1 - Q_\phi(t)$
$p^f_x(t)$ for $x \in \phi(\mathbf{x})$	$p^r_x(t)$ for $x \in \overline{\phi(\mathbf{x})}$
$p^r_x(t)$ for $x \in \phi(\mathbf{x})$	$p^f_x(t)$ for $x \in \overline{\phi(\mathbf{x})}$
$\omega_\phi(t)$	$\nu_{\bar{\phi}}(t)$
$\nu_\phi(t)$	$\omega_{\bar{\phi}}(t)$

5.4.3 Probabilistic quantification of the Top-module

Top event Unavailability $Q_T(t)$

The variables in the LBDD of the Top-module can be simple modules and basic events.
The Top event unavailability is calculated on the Top-module as a function of the unavailability of its variables by applying the same algorithm used for simple modules, i.e. equation (5-5).
Besides $Q_T(t)$ for $0 \le t \le T$, if tested events are present then ASTRA also calculates the mean value

$Q_{Tmean} = \int_0^t Q_T(\tau)\, d\tau$ and the peak value Q_{Tmax}.

The next step is the determination of $p^f(t)$ and $p^r(t)$ for all basic events and simple modules.
Indicating with:
$p^f_K(t)$ and $p^r_K(t)$ the probability of critical states for the simple module K in the Top-module, and with $p^f_{xK}(t)$ and $p^r_{xK}(t)$ the probability of critical states for the event x in the simple module K, then the probabilities $p^f_x(t)$ and $p^r_x(t)$ for event x in the input tree are obtained by means of the following equations:

$$p^f_x(t) = p^f_{xK}(t)\, p^f_K(t) + p^r_{xK}(t)\, p^r_K(t) \tag{5.12}$$

$$p^r_x(t) = p^r_{xK}(t)\, p^f_K(t) + p^f_{xK}(t)\, p^r_K(t) \tag{5.13}$$

The proof of the above equation is given in Appendix D.

The knowledge of the probability of critical states determined for all basic events allow calculating the unconditional failure and repair frequencies of the Top event.

$$\omega_T(t) = \sum_{x=1}^{Ne} p^f_x(t)\, \omega_x(t) + \sum_{x=1}^{Ne} p^r_x(t)\, \nu_x(t) \tag{5.14}$$

The unconditional repair frequency is given by:

$$\nu_T(t) = \sum_{x=1}^{Ne} p^r_x(t)\, \omega_x(t) + \sum_{x=1}^{Ne} p^f_x(t)\, \nu_x(t) \tag{5.15}$$

Top event Expected number of failures $W_T(t)$

The Expected Number of Failures $W_T(t)$ is obtained as:

$$W_T(t) = \int_0^t \omega_T(\tau)\,d\tau + Q_T(0) \tag{5.16}$$

The ENF can also be interpreted as the upper bound for the unreliability $F_T(t)$.

If $\omega_T(t) \approx$ constant then $MTBF_T = \dfrac{1}{\omega_T(t)}$; (5.17)

if $Q_T(t) \approx$ constant then $MTTR_T = Q_{Tmean} * MTBF_T$ (5.18)

Top event Expected number of repairs $V_T(t)$

If there are neither INH gates nor tested events then also the Expected Number of Repair $V_T(t)$ is calculated as:

$$V_T(t) = \int_0^t \nu_T(\tau)\ d\tau \tag{5.19}$$

Note that the following relationship holds:

$$Q_T(t) = W_T(t) - V_T(t) \tag{5.20}$$

Top event Unreliability upper bound $F_T(t)$

For safety applications the Expected number of failure is generally a good upper bound for the top-event unreliability $F_T(t)$. However if the ENF is greater than a user defined threshold (e.g. 0.1) also the second bound is calculated.
To this purpose the conditional failure frequency Λ_T at Top level is determined on the basis of the unconditional failure frequency $\omega_T(t)$ and unavailability $Q_T(t)$, (Vesely, 1970) i.e.:

$$\Lambda_T(\tau) = \frac{\omega_T(\tau)}{1 - Q_T(\tau)}$$

Then,

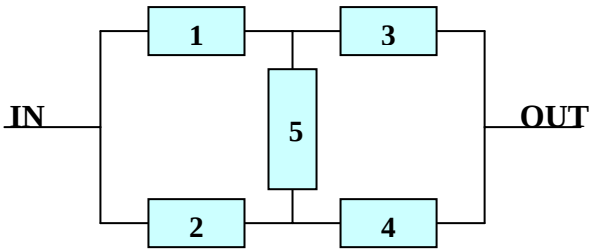
$$F_T(t) = 1 - [1 - Q_T(0)]\, e^{-\int_0^t \Lambda_T(\tau)\,d\tau} \tag{5.21}$$

Mean time to first failure is calculated, according to its definition, as:

$$MTTFF_T = \int_0^\infty (1 - F_T(t))\,dt \tag{5.22}$$

Example of application of the frequency analysis of a coherent failure function

Let us consider the bridge network. Te fault tree describes the events leading to the top-event “no output signal”.



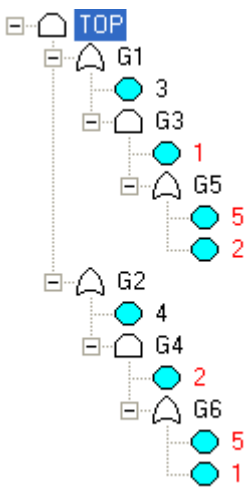


Figure 5.7 Bridge network of repairable components and associated fault tree

The analysis is performed considering $\lambda = 0.01$ and $\mu = 0.1$ for all events.

The results of the analysis are graphically represented in Figure 5.8 and Figure 5.9.

At the mission time T = 50 h the results are:

- Unavailability $Q_T(50) = 1.755459$ e-2
- Failure frequency $\omega_T(50) = 3.612036$e -3
- Repair frequency $\nu_T(50) = 3.595805$e -3
- Expected Number of Failures $W_T(50) = 0.1490059$
- Expected Number of Repair $V_T(50) = 0.1314516$
- Unreliability $F_T(50) = 0.1403110$

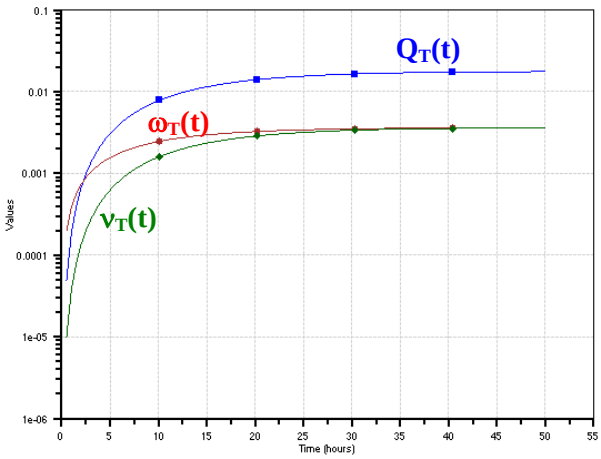


Figure 5.8 Unavailability, failure and repair frequencies for the bridge system

According to eq.(5.20): $W_T(50) - V_T(50) = 0.1490059 - 0.1314516 = 1.75543$e-2 $= Q_T(50)$.

The negligible difference between this value and the one calculated directly on the LBDD is due to the approximation in the calculation of the integrals for $W_T(50)$ and $V_T(50)$.

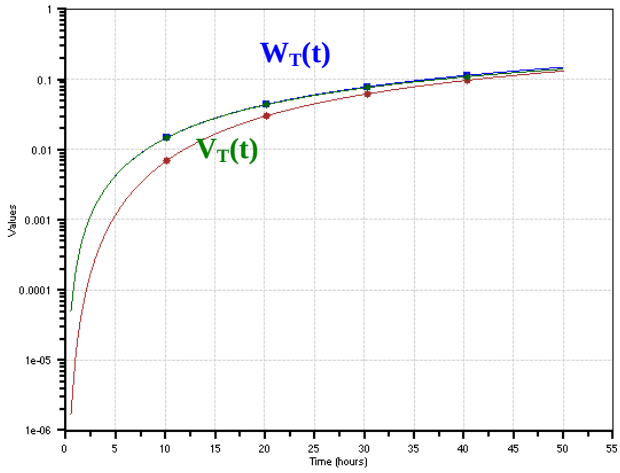


Figure 5.9 Expected number of failures, Expected number of repair and Unreliability upper bound for the bridge system

Example of frequency analysis for a non-coherent failure function

Let us consider again the function Top = $[x_2 (x_1 + \bar{x}_3 + \bar{x}_4)] + [x_3 (\bar{x}_1 + \bar{x}_2 \quad x_4)]$ previously considered for the determination of the unavailability.

The frequency analysis results are plotted in Figure 5.10 for a mission time of 10,000 h.

It can be seen that the unavailability reaches the steady state condition very rapidly and consequently Ws(t) and Vs(t) are very close each other. Their difference is the Unavailability Qs(t).
At T = 10,000 h the following values have been calculated by ASTRA:
Qs(T) = 2.005851E-03
ωs(t) = 9.129257E-05
νs(t) = 9.131060E-05
Ws(t) = 9.595718E-01
Vs(t) = 9.575631E-01
Fs(t) = 6.177130E-01

Ws(t) - Vs(t) = 2.008700e-3

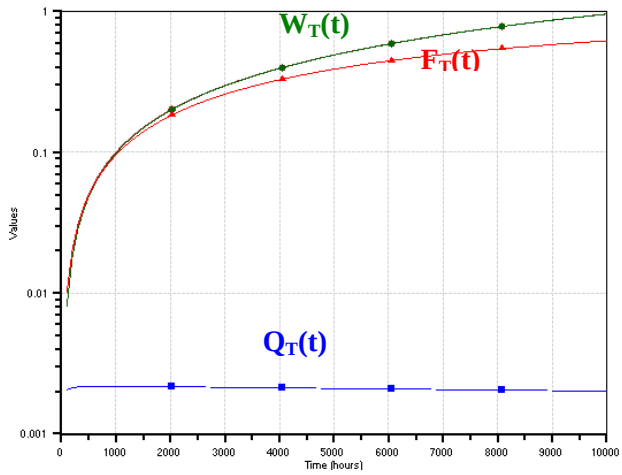


Figure 5.10 Unavailability, Expected number of failures and unreliability for the LBDD in Figure 5.1

5.4.4 Importance measures of basic events

Another useful result provided by ASTRA is the importance analysis, which is performed for all basic events of the fault tree. The following importance measures are provided:

- Probability of critical state;
- Criticality;
- Risk Achievement Worth;
- Risk Reduction Worth.
- Structural importance;

Once $Q_T(t)$, $p^f_x(t)$ and $p^r_x(t)$ are known the importance analysis can be performed. Importance measures can be determined either at a specific time (generally at the mission time) or as time dependent.
For each importance measure two contributions, positive and negative, are determined for events in double form. All importance measures are based on unavailability.

Probability of critical states $p^f_x(t)$ and $p^r_x(t)$

The equations for determining the probability of critical state $p^f_x(t)$ for positive variables and $p^r_x(t)$ for negated variables (or negated part of double form variables) have been given above.

Criticality importance measure, IC_x

This index represents the probability that the event *x* is critical and its occurrence leads to system failure. This index can also be interpreted as the relative variation of the Top-event occurrence probability vs. the relative variation of the occurrence probability of the basic event *x*.

$$IC^+_x(t) = p^f_x(t)\frac{q_x(t)}{Q_T(t)} \tag{5.23}$$

For negated variables:

$$IC^-_x(t) = p^r_x(t)\frac{1-q_x(t)}{Q_T(t)} \tag{5.24}$$

Risk Achievement Worth, RAW_x

The RAW is defined as a measure of the increase of the system failure probability when *x* is supposed failed or removed e.g. for test/maintenance operations. In calculating the RAW it is important to consider all other components that are dependent by the failure / removal of *x*. According to the definition it is proved in Contini (2005) that:

$$RAW^+_x(t) = 1 + p^f_x(t)\frac{1-q_x(t)}{Q_T(t)} \tag{5.25}$$

For negated variables:

$$RAW^-_x(t) = 1 + p^r_x(t)\frac{q_x(t)}{Q_T(t)} \tag{5.26}$$

Risk Reduction Worth RRW_x

The RRW is defined as a measure of the decrease of the system failure probability when *x* is supposed to be perfectly working:

$$RRW_x^+ = \frac{Q_T}{Q_T - p_x^f(t)\, q_x(t)} \tag{5.27}$$

For negated variables:

$$RRW_x^- = \frac{Q_T}{Q_T - p_x^r(t)[1 - q_x(t)]} \tag{5.28}$$

Structural importance IS_x

The Structural importance is determined by applying the equation $p^f_x(t)$ and $p^r_x(t)$ in which all events have probability 0.5.

5.5 Probabilistic quantification of SMCS

The determination of the generic SMCS is followed by the probabilistic quantification of each of them for the determination of: Unavailability, unconditional failure frequency, Expected number of failures, and Unreliability. An MCS of order *n* is the parallel failure configuration of *n* basic events.

Unavailability of a MCS

The unavailability of a generic minimal cut set C_j of order *n* is the probability that all *n* events are verified at time t. Since events are independent then:

$$Q_{Cj}(t) = \prod_{i=1}^{nj} q_i(t) \tag{5.29}$$

Expected number of failures of a MCS

The ENF of a MCS is obtained by integrating, over the mission time interval, the unconditional failure frequency of the SMCS given by:

$$\omega_{Cj}(t)\, dt = \sum_{i=1}^{nj} \omega_i(t) \prod_{\substack{k=1 \\ k \neq i}}^{nj} q_k(t)\, dt \tag{5.30}$$

The above equation expresses the concept that the SMCS occurs in a time interval t, t+dt if:

- *n-1* events already occurred at t, given by $\prod_{\substack{k=1 \\ k \neq i}}^{nj} q_k(t)$
- the last one occurs in *dt*, expressed as $\omega_i(t)\, dt$

The last event to occur may be the first, the second, and so on, that's why the use of the summation.

Hence: $$W_{Cj}(t) = \int_0^t \omega_{Cj}(\tau)\, d\tau + Q_{Cj}(0) \tag{5.31}$$

Unreliability upper bound of a MCS

As for the Top event, the unreliability of a MCS is calculated by means of the conditional failure frequency of the MCS `determined from $\omega_T(t)$ and $Q_T(t)$.

$$\Lambda_{Cj}(\tau) = \frac{\omega_{Cj}(\tau)}{1 - Q_{Cj}(\tau)}$$

Then,

$$F_{Cj}(t) = 1 - [1 - Q_{Cj}(0)]\, e^{-\int_0^t \Lambda_{Cj}(\tau)\, d\tau} \qquad (5.32)$$

5.6 Frequency analysis using the extended INH gate

In the analysis of catastrophic Top-events it is important to model situations in which a failure occurs only if the direct causes occur in a given sequence. Consider for instance the following example. The overpressure in a tank triggers the intervention of a shut-down system; if this system does not operate then the tank rupture occurs. Hence the event "tank rupture" is due to the occurrence of the initiating events I "overpressure" and the enabling event E "shut-down does not intervene". However, the tank rupture can occur *only if* E occurs before the occurrence of I. If E occurs before I there would be simply a spurious trip of the plant.

The simple AND of the two input variables I and E cannot represent this situation, since the sequence of occurrence cannot be taken into consideration.

These situations can be modelled in ASTRA using the Inhibit (INH) gate.

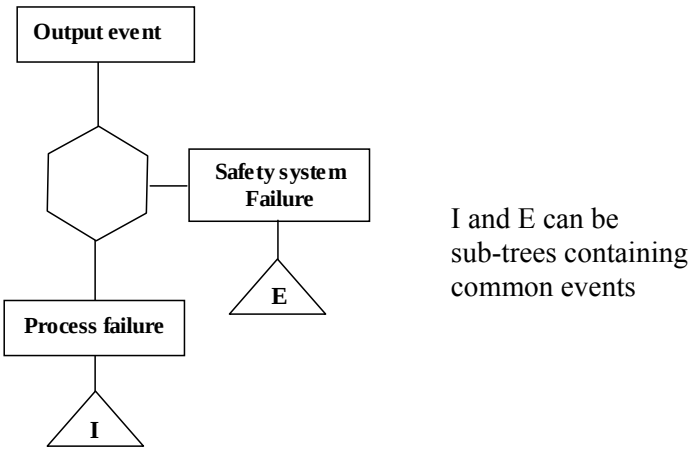


Figure 5.12 The INH gate used for modelling the relationship between initiator and enabler events

This *extended definition of the INH gate* is based on the distinction between *initiating* and *enabling* events.

An Initiating event is an event whose occurrence triggers the intervention of the Enabler event.
The output is true when, at the time the input is true, the condition defined by the enabler event is *already* true.

The method implemented in ASTRA identifies the events as either *initiator* or *enabler* depending on the sub-tree they belong to. Common events are flagged as initiators.

The differentiation of the type of events has an impact on the calculation of W(t), since initiating events are characterised by their failure frequency $\omega(t)$, whereas enabler events, associated with components of the protective system, are characterised by their on-demand unavailability q(t).

The failure and repair frequencies of the output event are given by:

$$\omega_o(t) = \omega_I(t)\, q_E(t)$$

$$\nu_o(t) = \nu_I(t)\, q_E(t)$$

Therefore in ASTRA 3.0 when the parameter of interest is the frequency of the catastrophic Top event modelled by means of the INH gate, then in equations (5-14), (5-15) and (5-30) enabler events are characterised by their on-demand unavailability $q_x(t)$ only, i.e. their unconditional failure and repair frequencies are set to zero.

Example

A system is comprised of two components: A monitors the operation of the component B. System failure occurs if both fail, but only if A fails before B.

Data about components are as follows.
A : λ_A = 1.e-6 $\mu_A = 0$
B : λ_B = 1.e-7 $\mu_B = 0$
The system is not repairable. The following table shows the comparison of the unavailability values Q(t) obtained using a Markovian approach (Ericson, 2005) with those calculated using the ASTRA method above described for different mission times. As can be seen the agreement is very good even extreme values of the mission times greater than the components failure rates.

Table 5.IV Comparison between Markov analysis and ASTRA 3.0 on sequential events

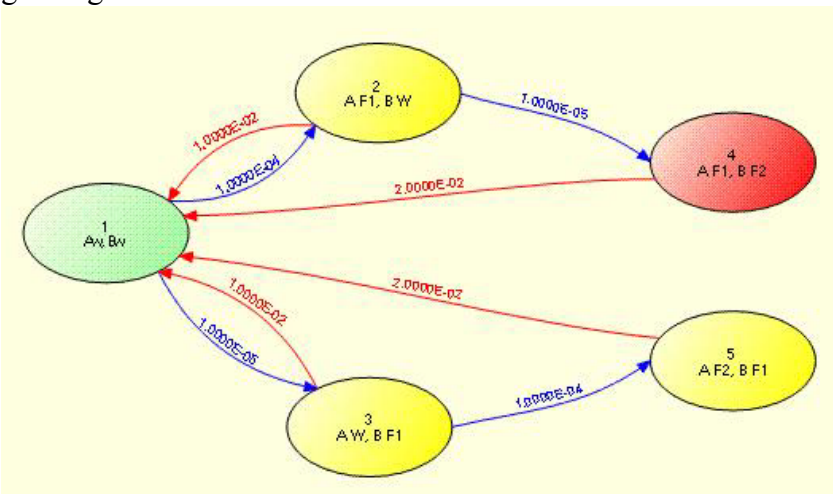
Mission time (h)	**Markov**	**ASTRA**
100	4.99980E-10	4.99980E-10
1,000	4.99800E-8	4.99800E-8
10,000	4.98006E-6	4.98005E-6
100,000	4.80542E-4	4.80542E-4
1,000,000	3.45145E-2	3.45144E-2
10,000,000	5.41213E-1	5.41208E-1

Let us consider now the same problem with the following parameters:
A : λ_A = 1.e-4 μ_A = 1.e-2
B : λ_B = 1.e-5 μ_B = 1.e-2

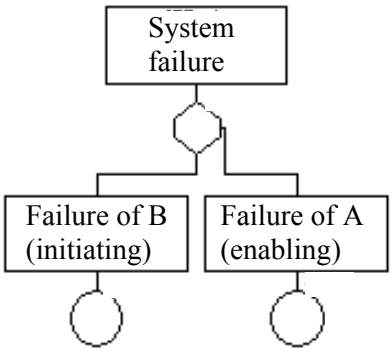
The Markov state diagram and the fault tree are as shown in Figure 5.13.

Table 5.IV contains the results of the analysis for a mission time T = 1,000 h.

The comparison of the results between the two programs XSMKA (De Cola, 2005) and ASTRA shows good agreement.



A, B components' names;
Subscripts:
W = Working
F = Failed;
N = state number from which the transition comes from.



Fault tree representation
In ASTRA

Figure 5.13 Markov state diagram and fault tree for sequential repairable events

Table 5.V Results of the analysis of the system represented in Figure 5.13

Parameter	XSMKA	ASTRA
Unreliability	8.901275E-05	8.911964E-05
Unavailability	4.940146E-06	4.96708E-06
ENF – Expected Number of Failures	8.904747E-05	8.912281E-05
ENR – Expected Number of Repairs	8.407051E-05	8.415573E-05

In practice it is common to deal with a cascade of INH operators as shown in Figure 5.14 a), in which the previously considered tank is supposed to have two levels of protection. In this case two INH gates are used to model the pressure increase. We can notice that in this case there is no need to consider the sequence of intervention of the safety devices, since both must be failed to produce the tank rupture. Therefore the cascade of INH gates is equivalent to a single INH gate in which all enabling functions are grouped under an AND gate of the enabler branch as shown Figure 5.14 b).

These considerations are applicable to the cascade of any number of INH gates.

This extended implementation of the INH gate should not be confused with the sequential AND gate that may have more than two inputs. In the cascade of INH gates the grouped enabler events are independent. In the latter we may have more than two events which must fail according to a given sequence.

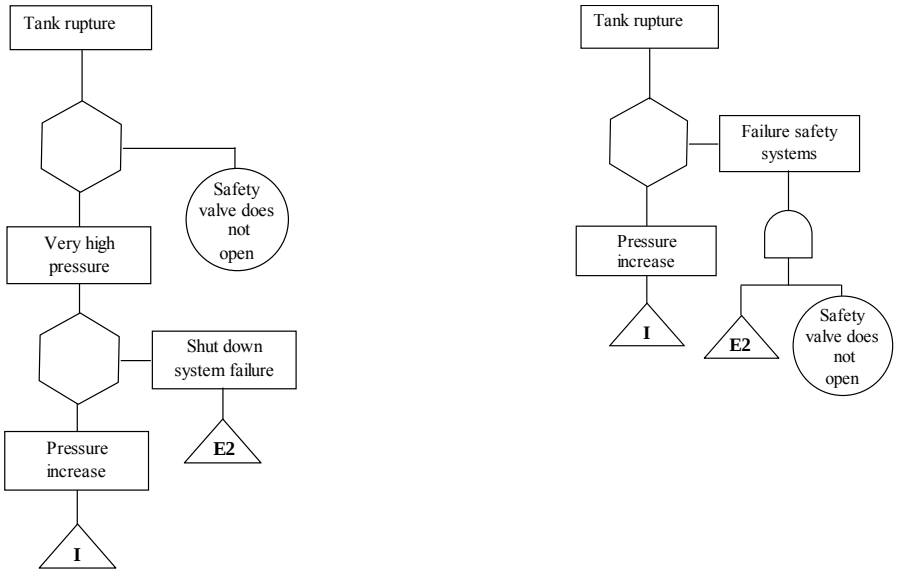


Figure 5.14. a) Cascade of INH gates; b) equivalent fault tree representation

6. CONCLUSIONS AND ON-GOING DEVELOPMENTS

In this report we have described in sufficient detail the main algorithms implemented in ASTRA 3.0 for performing the analysis of both coherent and non-coherent fault trees.
Since non-coherent fault trees contain different types of variables (SP, SN, and DF) for which algorithm of different cost (in terms of computational resources) are required, an algorithm for dynamically labelling each BDD node with the variables' type was implemented. The experimental results showed the advantage of the dynamic labelling operation. The reduction of the number of nodes with DF variables implies a reduction of the working memory due to the reduction of the number of intersections to be calculated for determining the failure and repair frequencies. Differences between the classical BDD and the LBDD have been described in detail.

From the LBDD the ZBDD embedding all MCS is obtained from which the SMCS are extracted.

Unfortunately the number of nodes of a BDD increases exponentially with the complexity of the fault tree. Therefore on very complex fault trees (i.e. sequences of event trees of nuclear power plants) it happens that the working memory is not sufficient to store the large BDD generated. In order to overcome this limitation another analysis procedure was implemented in ASTRA 3.0 to construct directly the ZBDD embedding SMCS, so by-passing the construction of the LBDD.

The new version of ASTRA 3.0 has been extensively tested on a large number of cases. This activity allowed us to identify further improvements, namely:

- Implementation of the importance measures of basic events in case of failure frequency analysis;
- Make ASTRA conform to the standard IEC 61508 *"Functional safety of electrical / electronic / programmable electronic safety-related systems"*.
- Development of a module for uncertainty analysis applied to the LBDD

The theoretical aspects of these new developments are on-going and they will be the subject of the future ASTRA upgrades.

ACKNOWLEDGEMENTS

The present work has been executed in 2009 in the framework of the Project "Systems Analysis Applied to Nuclear Safeguards and Non Proliferation", carried out in the NUSIM (Nuclear Fuel Cycle Simulations) action of the Nuclear Security unit of JRC-IPSC. The work was performed as part of the AMENUS (Assessment Methodologies for Nuclear Security) action activities until the end of 2008.

The authors wish to thanks W. Janssens, P. Peerani, and M. Sironi for the support to the activity. A particular thank is due to G.G.M. Cojazzi for the helpful discussions and advice during the whole period in which the action AMENUS was under his responsibility.

REFERENCES

Akers S. B. (1978) Binary Decision Diagrams, *IEEE Transactions on Computers,* Vol C-27.

Brace K, Rudell R, Briant R. (1990), Efficient Implementation of a BDD Package, *27th ACM/IEEE Design Automation Conference,* IEEE 0738.

Bryant R.E (1986), Graph Based Algorithms for Boolean Functions Manipulation, *IEEE Transactions on Computers*, Vol. C-35.

Clarotti C.A. (1981), Limitations of minimal cut-set approach in evaluating reliability of systems with repairable components, *IEEE Transactions on Reliability,* Vol. 30.

Contini S. Cojazzi G.G.M., De Cola G. (2004), Sull'uso degli alberi di guasto non coerenti nell'analisi dei sistemi, Conference VGR '04, Pisa, Italy.

Contini S. (2005), Measures of importance for non-coherent fault trees, JRC-working document, available from the author.

Contini S. Cojazzi G.G.M., Renda G. (2008), On the use of non-coherent fault trees in safety and security studies, *Reliability Engineering and System Safety,* Vol. 93.

Contini S. Matuzas V. (2009), "ASTRA 3.0 : Test Case Report", EUR Technical Report, EUR 24124 EN.

Coudert J.C, Madre P. (1994), Metaprime: an Interactive Fault Tree Analyser with Binary Decision Diagrams, *IEEE Transactions on Reliability*, Vol. 43.

De Cola G., 2005, XS Toolset – MKA Module for Markovian Analysis, User Guide.

Demichela M., Piccinini N., Ciarambino I., Contini S., (2003), On the numerical solution of fault trees, *Reliability Engineering and System Safety,* Vol. 82.

Ericson, 2005 C.A., Hazard Analysis Techniques for System Safety, Wiley Interscience, ISBN: 0.471-72019-4

Jung W. S., Han S. H., Ha J. (2004) A fast BDD algorithm for large coherent fault tree analysis, *Reliability Engineering and System Safety*, Vol. 83.

Jung W. S., Han S. H., Ha J. (2008) Fast BDD truncation method for efficient top-event probability calculation, *Nuclear Engineering and Design*, Vol. 40.

Liu J.C, Pan Z.J. (1990), A New Method to Calculate the Failure Frequency of Non-Coherent Systems, I*EEE Transactions on Reliability,* Vol. 39.

Minato S. (1990), Zero-suppressed BDDs for set manipulation in combinatorial problems, *Proc. ACM/IEEE Design Automation Conference*.

Odeh K, Limnios N. (1996), A New Algorithm for Fault Trees Prime Implicants Computation, Probabilistic Safety Assessment and Management, *ESREL 96 – PSAM III Conference*, P.C. Cacciabue, I.A. Papazoglou, Creete, Greece.

Rauzy (1993), A, New Algorithms for Fault Trees Analysis, *Reliability Engineering and System Safety*, Vol 40, 203-211,

Rauzy A, Dutuit Y. (1997), Exact and Truncated Computation of Prime Implicants of Coherent and Non-Coherent Fault Trees within Aralia, *Reliability Engineering and System Safety*, Vol 58.

Vesely W.E. (1970), A time dependent methodology for fault tree evaluation. *Nuclear Engineering and Design, Vol 13.*

APPENDIX A

Coherence and non-coherence

This appendix is devoted to describe the concepts of coherence and non-coherence functions representing the system failure states.

Let us consider a system composed of n interconnected components. Each component is characterised by a binary variable indicating its status at a generic moment in time. To the generic component c_i the associated binary variable x_i takes the value 0 representing the component working state and the value 1 representing the component failed state.

Definition of Coherent binary functions

Let $\Phi(\mathbf{x})$ be the function of the vector $\mathbf{x} = |x_1, x_2, \dots, x_n|$ of the status of the n system components.
Since $x_i = (0,1)$ for any i, then $\Phi(\mathbf{x}) = (0,1)$, i.e. the function $\Phi(\mathbf{x})$ is binary.
A binary function $\Phi(\mathbf{x}) = \Phi(x_1, x_2, ..., x_n)$ is monotonic (non-decreasing), also referred to as *coherent*, if the following two conditions are satisfied:

a) $\Phi(1_i, \mathbf{x}) \geq \Phi(0_i, \mathbf{x})$ for <u>any variable and for any **x**,</u> where
$\Phi(1_i, \mathbf{x}) = \Phi(x_1, x_2, .., x_{i-1}, 1, x_{i+1},..., x_n)$ and $\Phi(0_i, \mathbf{x}) = \Phi(x_1, x_2, .., x_{i-1}, 0, x_{i+1},..., x_n)$

This means in practice that the status of the system with the i-th component failed cannot be better than the status of the system with the i-th component good, for any combination of the states of the other components.

b) All components are relevant, i.e. $\Phi(x_1, x_2, .., x_{i-1}, 1, x_{i+1},..., x_n) \neq \Phi(x_1, x_2, ..., x_{i-1}, 0, x_{i+1},..., x_n)$ for <u>any variable and at least for one vector **x**.</u> In other words there are no components whose status is indifferent for the determination of the system state.

From condition a) it follows that:
$\Phi(\mathbf{1}) = 1$, i.e. if all components are failed, i.e. $\mathbf{1} = [x_1=1, x_2=1, \dots, x_n=1]$, the system is failed;
$\Phi(\mathbf{0}) = 0$, i.e. if all components are working, i.e. $\mathbf{0} = [x_1=0, x_2=0, \dots, x_n=0]$, the system is working.

As an example consider the function $\Phi(\mathbf{x}) = a\,b + a\,c$. The following table contains $\Phi(1_i, \mathbf{x})$ and $\Phi(0_i, \mathbf{x})$ for all variables, showing that the conditions for coherence are satisfied.

Variable	$\Phi(1_i, \mathbf{x})$	$\Phi(0_i, \mathbf{x})$	$\Phi(1_i, \mathbf{x}) \geq \Phi(0_i, \mathbf{x})$?	$\Phi(1_i, \mathbf{x}) \neq \Phi(0_i, \mathbf{x})$?
a	b + c	0	yes	yes
b	a + a c	a c	yes	yes
c	a b + a	a b	yes	yes

Consider now the function $\Phi(\mathbf{x}) = a\,b + a\,\bar{c}$. The following table shows that the first condition for coherence is not satisfied because of the presence of $\bar{c}$.

Variable	$\Phi(1_i, \mathbf{x})$	$\Phi(0_i, \mathbf{x})$	$\Phi(1_i, \mathbf{x}) \geq \Phi(0_i, \mathbf{x})$?	$\Phi(1_i, \mathbf{x}) \neq \Phi(0_i, \mathbf{x})$?
a	$b + \bar{c}$	0	yes	yes
b	$a + a\,\bar{c}$	$a\,\bar{c}$	yes	yes
c	a b	a b + a	**no**	yes

Note that the conditions for coherence $\Phi(1_i, \mathbf{x}) \geq \Phi(0_i, \mathbf{x})$ means that $\Phi(1_i, \mathbf{x}) = \Phi(0_i, \mathbf{x})$ or that

$\Phi(1_i, \mathbf{x}) > \Phi(0_i, \mathbf{x})$. But this is equivalent to state that $\Phi(1_i, \mathbf{x}) \wedge \Phi(0_i, \mathbf{x}) = \Phi(0_i, \mathbf{x})$. In fact the left hand side is equal to $\Phi(0_i, \mathbf{x})$, in that:

if $\Phi(1_i, \mathbf{x}) = \Phi(0_i, \mathbf{x})$ then $\Phi(0_i, \mathbf{x})\ \Phi(0_i, \mathbf{x}) = \Phi(0_i, \mathbf{x})$;

if $\Phi(1_i, \mathbf{x}) > \Phi(0_i, \mathbf{x})$ then $\Phi(1_i, \mathbf{x}) \supset \Phi(0_i, \mathbf{x})$, which means that $\Phi(1_i, \mathbf{x}) \wedge \Phi(0_i, \mathbf{x}) = \Phi(0_i, \mathbf{x})$.

This equivalence allows us to derive a more operative test for coherence than condition a).

To further clarify the concept of coherence let us consider the following function, $\Phi = a\,b + a\,c$ where a, b and c represent the status of the corresponding system components assumed, for simplicity, not repairable.
The graph in figure B.1 represents all vectors **x**, i.e. all combinations of component states.
The system evolves from the initial state a= 0, b= 0, c= 0, to the final state a= 1, b= 1, c= 1.
From any state the system can enter another state following the change of state of only one variable.

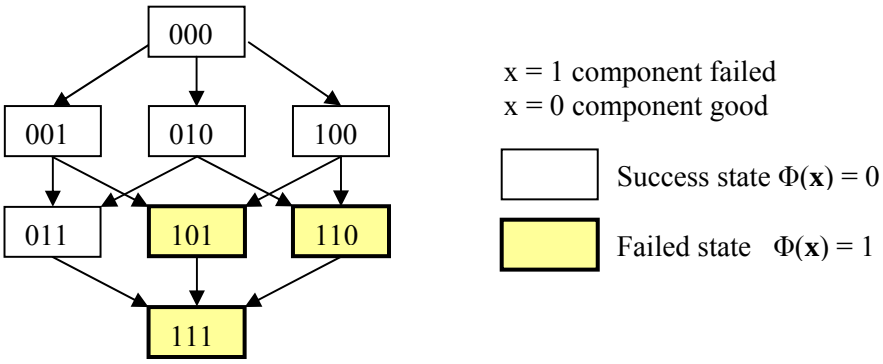


Figure B.1 State graph representation of a coherent function

We can see that from a working state, say $\Phi(010)=0$, the system may evolve either into another working state, e.g. $\Phi(011)=0$ or into a failed state, e.g. $\Phi(110)=1$ following respectively the failure of c or of a. Once a failed state is reached the system can evolve only towards failed states, i.e. it remains failed.

Consider now the not monotonic function $\Phi = a\,b + a\,\bar{c}$ and the graph in figure B.2 representing all its possible states. Here we can see that from the failed state $\Phi(100) = 1$ the system can pass to the failed state $\Phi(110)=1$ when b fails (b=1) or the working state $\Phi(101)=0$ when component c fails (c = 1, means $\bar{c} = 0$).

We speak about non-coherent system behaviour when the failure of a component makes the system to pass from a failed state ($\Phi = 1$) to a working state ($\Phi = 0$) or when the repair of a component makes the system to pass from a working state ($\Phi = 0$) to a failed state ($\Phi = 1$).

It is clear that such behaviours are unrealistic in real systems. However, we are dealing with binary functions describing the failure of a system. In these cases such functions may also be non coherent as shown in the simple example in Figure B.3.
A tank S, placed inside a bund, contains a dangerous substance xy.

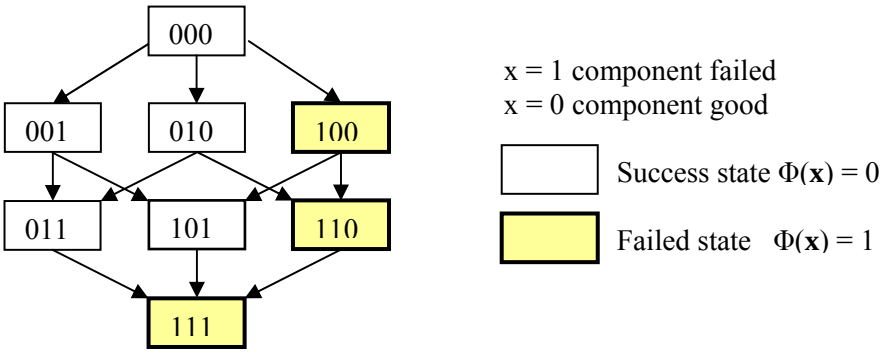


Figure B.2 State graph representation of a non-coherent function

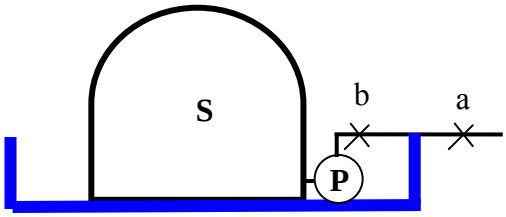


Figure B.3 Simple system showing non-coherent top events.

Consider the following Top events:

– Top.a: *Release of the substance outside the bund*
– Top.b*: Release of the substance inside the bund*

and the events:
A = pipe rupture at point a;
B = pipe rupture at point b.

If we consider, for the sake of simplicity, only these two events, the logical functions for the above defined Top events are:

$$\text{Top.a} = \overline{\text{B}} \wedge \text{A}$$

$$\text{Top.b} = \text{B}$$

If A occurs Top.a occurs; however if also B occurs then Top.a does not occur any more. This does not mean that the system has improved, but rather that the conditions that verified Top.a are no more satisfied with the occurrence of B.

When B occurs, Top.b occurs. In fact the two Top events are mutually exclusive, i.e.

$$\text{Top.a} \wedge \text{Top.b} = \overline{\text{B}} \wedge \text{A} \wedge \text{B} = 0$$

The use of non-coherent functions, i.e. fault trees containing negated events is useful in system modelling. See e.g. Contini et al. 2008.

To complete this short introduction let us see how the concept of coherence can be explained considering the different types of variables that can be found in a non-coherent fault tree.

Let $\Phi(\mathbf{x})$ be a binary function in which all variables are relevant. It is easy to realise that a coherent function contains only positive variables, whereas a non coherent function contains different types of variables. More precisely variables that appear in both forms, positive and negated, referred to as Double Form (DF) in this report, e.g. x_i, $\bar{x}_i$. Moreover it may also contain Single form Negated (SN) variables e.g. $\bar{x}_h$, as well as Single form Positive (SP) variables e.g. x_k.
Different types of variables require different algorithms of analysis.

For <u>Single form Positive variables</u> (SP) $\Phi(\mathbf{x})$ can be expressed as:
$\Phi(\mathbf{x}) = x\,A + B$
where A and B are binary functions.
In this case $\Phi(1,\mathbf{x}) = A + B$ and $\Phi(0,\mathbf{x}) = B$
The first condition for coherence $\Phi(1,\mathbf{x})\,\Phi(0,\mathbf{x}) = (A + B)\,B = B = \Phi(0,\mathbf{x})$ is satisfied.
$\Phi(1,\mathbf{x})$ is always greater than or equal to $\Phi(0,\mathbf{x})$, i.e. the function $\Phi(\mathbf{x})$ is *monotonically not decreasing*.
Thus we could also say that $\Phi(\mathbf{x})$ is (positively) coherent with respect to variables of SP type.

For <u>Single form Negative variables</u> (SN) $\Phi(\mathbf{x})$ can be expressed as:
$\Phi(\mathbf{x}) = \bar{x}\,A + B$
In this case $\Phi(1,\mathbf{x}) = B$ and $\Phi(0,\mathbf{x}) = A + B$
Therefore $\Phi(1,\mathbf{x})\,\Phi(0,\mathbf{x}) = B\,(A + B) = B = \Phi(1,\mathbf{x})$
$\Phi(1,\mathbf{x})$ is always less than or equal to $\Phi(0,\mathbf{x})$, i.e. the function $\Phi(\mathbf{x})$ is *monotonically not increasing.*
Thus we could also say that the function $\Phi(\mathbf{x})$ is (negatively) coherent with respect to variables of SN type.

For <u>Double Form variables</u> (DF) $\Phi(\mathbf{x})$ can be expressed as:
$\Phi(\mathbf{x}) = x\,A + \bar{x}\,B$
$\Phi(1,\mathbf{x}) = A$ and $\Phi(0,\mathbf{x}) = B$
$\Phi(1,\mathbf{x})\,\Phi(0,\mathbf{x}) = A\,B$ which is different from both $\Phi(1,\mathbf{x})$ and $\Phi(0,\mathbf{x})$.
The function $\Phi(\mathbf{x})$ is *not monotonic* or non-coherent with respect to variables of DF type.

As an example of application of the above classification consider the following <u>non coherent</u> function:

$$\Phi(a,b,c) = \bar{a}\,b + \bar{a}\,c + b\,\bar{c}$$

in which a, b and c are respectively of SN, SP and DF type.

For each type of variable we construct a Karnough map to show the variations of the function following the change of state of the considered variable from 0 to 1.

<u>Variable b (SP type):</u>

$\phi(b=1) = \bar{a} + \bar{a}\,c + \bar{c} = \bar{a} + \bar{c}$
$\phi(b=0) = \bar{a}\,c$
$\phi(b=1)\,\phi(b=0) = \bar{a}\,c$

b \ a c	00	01	11	10	
0	**0**	**1**	**0**	**0**	← $\Phi(b=0)$
1	**1**	**1**	**0**	**1**	← $\Phi(b=1)$

We can see that with the transition from b = 0 to b = 1 the value of the function $\Phi(a,b,c)$ either does not change (second and third column) or it increases from 0 to 1 (first and fourth column). We say that $\Phi(a,b,c)$ is monotonically not decreasing with respect to the variable b.

<u>Variable a (SN type)</u>:

$\Phi(a=1) = b\,\bar{c}$
$\Phi(a=0) = b + c + b\,\bar{c} = b + c$
$\Phi(a=1)\,\Phi(a=0) = b\,\bar{c}$

a \ b c	00	01	11	10	
0	**0**	**1**	**1**	**1**	← $\Phi(a=0)$
1	**0**	**0**	**0**	**1**	← $\Phi(a=1)$

We can see from the above Karnaugh map that with the passage from a = 0 to a = 1 the value of the function $\Phi(a,b,c)$ either does not change or it decreases from 1 to 0. We say that $\Phi(a,b,c)$ is monotonically not increasing with respect to the variable a.

<u>Variable c (DF type):</u>

$\phi(c=1) = \bar{a}\,b + \bar{a} = \bar{a}$
$\phi(c=0) = b + \bar{a}\,b = b$
$\phi(c=1)\,\phi(c=0) = \bar{a}\,b \neq \phi(c=0)$

c \ a b	00	01	11	10	
0	**0**	**1**	**1**	**0**	← $\Phi(c=0)$
1	**1**	**1**	**0**	**0**	← $\Phi(c=1)$

In this case with the transition from c = 0 to c = 1 the value of the function $\Phi(a,b,c)$ either does not change, or it increases from 0 to 1 or it decreases from 1 to 0. We say that $\Phi(a,b,c)$ is not monotonic with respect to the variable c.

APPENDIX B

Binary Decision Diagrams

The BDD representation of a fault tree (Boolean function) can be obtained through the application of the Shannon decomposition theorem with respect to all variables. The Shannon theorem states that any Boolean function can be expressed as follows:

$$F(x_1, x_2, .., x_{i-1}, x_i, x_{i+1},..., x_n) = x_i \; F_1 + \bar{x}_i \; F_0$$

where:
$F_1 = F|_{xi=1} = f(x_1, x_2, .., x_{i-1}, 1, x_{i+1},..., x_n)$ and $F_0 = F|_{xi=0} = f(x_1, x_2, .., x_{i-1}, 0, x_{i+1},..., x_n)$ are refereed to as the residues or cofactors of F with respect to x_i.
F_1 and F_0 represent the function F when the variable x_i is set to 1 and to 0 respectively;
" $\bar{\ }$ " represents the NOT logical operator, "+" the OR operator, whereas the AND operator is understood.

The above formula can easily be represented as a directed a-cyclic graph, i.e. a graph without loops.
A node of the graph represents a variable of F. A node can be either non-terminal or terminal. A non-terminal node *v* is labelled with the variable *var(v)* and has two children *l(v)* and *r(v)* which correspond, respectively, to F_1 and F_0; a terminal vertex can assume values **0**, **1** and has no children. A generic node is therefore represented by a triple [*v*, *l*(*v*), *r*(*v*)].

The BDD is constructed by adopting an ordering for variable expansions. The ordering of variables is indicated with the symbol "<". In practice, *a* < *b* means that in the BDD the node with the variable *b* descends from the node with *a*.

The idea of imposing an ordering to variables is due to Bryant (1986) and the corresponding BDD is referred to as "Ordered Binary Decision Diagram, OBDD". Bryant has shown that using an OBDD representation, Boolean function manipulations become much simpler and efficient.

The correct choice of the ordering of variables is very important, having a strong effect on the complexity of the resulting BDD and on the number of operations needed to reduce it.

As an example consider the following logical expression: $F = (a + \bar{b}\, c)\,(b + c)$. Choosing, for instance, the ordering a < b < c, the repeated application of (1) to F gives the following OBDD, where dotted lines mean v =0.

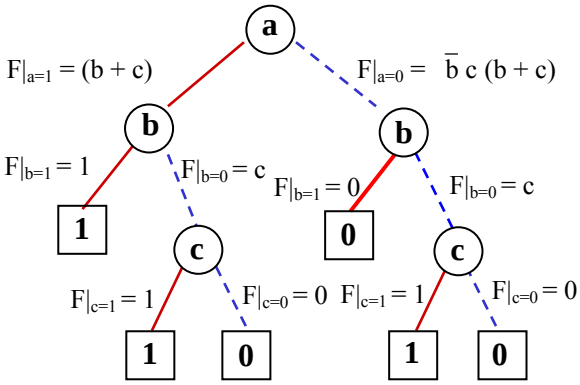
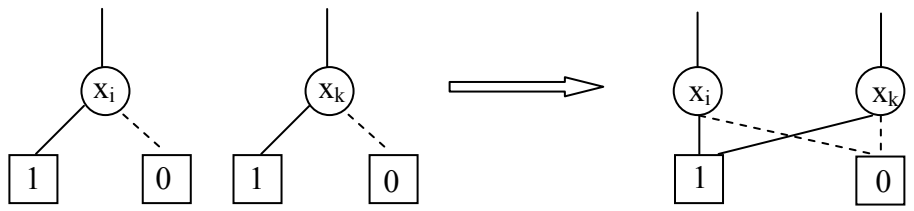


Figure B.1. OBDD representation of $F = (a + \bar{b}\, c)\,(b + c)$

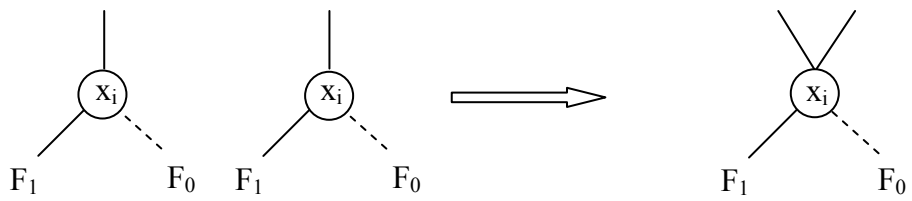
Terminal vertices **0** and **1** represent the values of the Boolean function F for different combinations of variable values. In the BDD representation, a generic path from **1** to the root is a set of variables satisfying F. For instance, in figure A.1, $a = 1$ and $b = 1$ is a set of assignments to variables *a, b* satisfying F, i.e. a path from 1 to the root.

A more compact representation of this tree can be obtained applying the following rules:

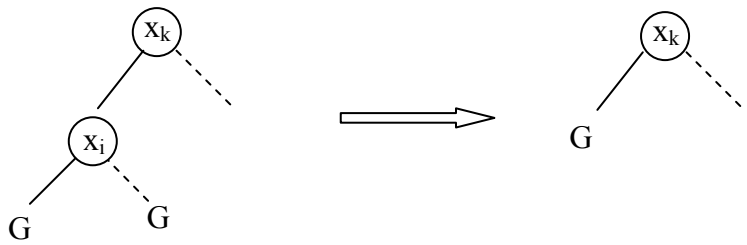
<u>Reduction Rule 1</u>: merge terminal vertices with the same value (0, 1);



<u>Reduction Rule 2</u>: merge non-terminal vertices representing the same variable and having the same descendants, i.e. *u* and *v* can be merged iff: l(u) = l(v) , r(u) = r(v);

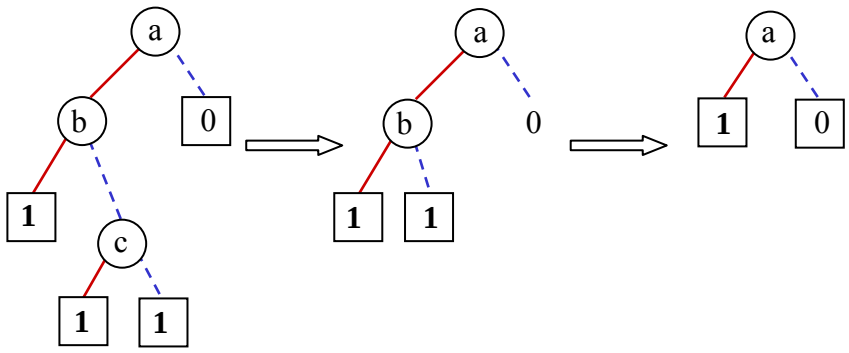


<u>Reduction Rule 3</u>: remove redundant vertices (*v* is redundant when *l(v) = r(v))*.



Rule R2 merges equal sub-graphs into a single sub-graph, whereas rule R3 applies the following logical equivalence: $(x\ G) + (\bar{x}\ G) = G$ where G is a generic sub-graph.

As a further example, the following shows the application of the third reduction rule.
Let F = a (b + c) + a c = a



The reduction rules applied to the BDD of figure A.1 gives the following graph, referred to as Reduced Ordered BDD (ROBDD).

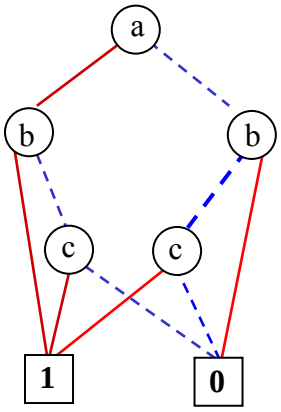


Figure B.2. ROBDD representation of $F = (a + \bar{b}\ c)\ (b + c)$

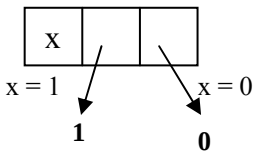
All paths from 1 to the root define the satisfying set S (dotted lines represent negated variables). For the above graph this set is $S = \{a\ b,\ a\ \bar{b}\ c,\ \bar{a}\ \bar{b}\ c\}$.

<u>The "ite" representation</u>

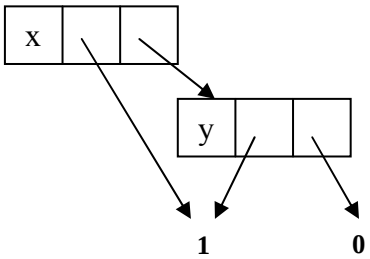
The efficiency of the BDD approach depends on the way the graph is stored and manipulated. Brace et al. (1990) suggested using a *hash table* to store the graph represented in the *"ite"* form.
The "*ite*" representation of a node (*ite* means if-then-else) is nothing but a row of a hash table containing the variable, the left descendant and the right descendant. As already mentioned, left and right descendants correspond to F_1, F_0, respectively.

Some examples of *ite* representation of simple functions are given below.

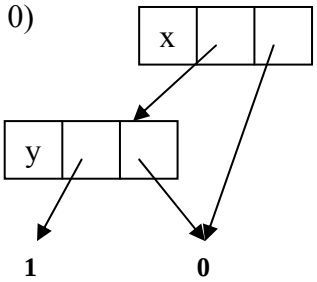
f = x *ite*(x, 1, 0)



f = x ∨ y ite(x, 1, ite(y, 1, 0)



f = x ∧ y ite(x, ite(y, 1, 0), 0)



The *ite* representation of the OBDD in Fig. B.2 is as follows:

ite(a, ite(b, 1, ite(c, 1, 0)), ite(b, 0, ite(c, 1, 0)))

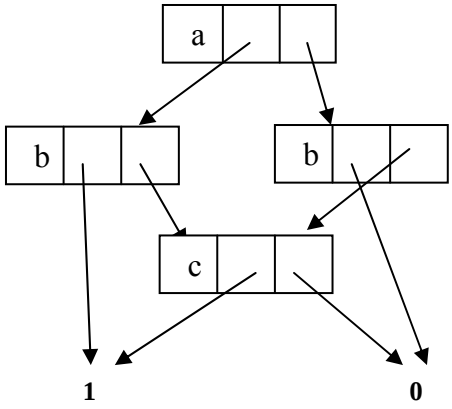


Figure B.3 "ite" representation of the OBDD in Figure B.2

BDD construction procedure

It is not efficient to generate a BDD by applying the Shannon decomposition to the function in a top-down way as shown above. It is more convenient to obtain it in a bottom up way through the composition of functions by means of the application of the following formula:

$$F \otimes G = x\,(F|_{x=1} \otimes G|_{x=1}) + \bar{x}\,(F|_{x=0} \otimes G|_{x=0})$$

where $\otimes$ is a binary Boolean operator, i.e. AND, OR, XOR, NAND, NOR.

Let $F = ite(x, F_1, F_0)$ and $G = ite(y, G_1, G_0)$, be two binary functions. In constructing the BDD the following cases are possible (Rauzy, 1993):

<u>Composition Rule 1.</u>

x < y

$ite(x, F_1, F_0) \otimes ite(y, G_1, G_0) =$

$= ite(x, F_1 \otimes ite(y, G_1, G_0), F_0 \otimes ite(y, G_1, G_0))$

<u>Composition Rule 2</u>

x = y

$ite(x, F_1, F_0) \otimes ite(x, G_1, G_0) = ite(x, F_1 \otimes G_1, F_0 \otimes G_0)$

The complementation of a function is straightforward. Indeed if $F = ite(x, F_1, F_0)$, then $\bar{F} = ite(x, F_0, F_1)$, i.e. the complementation of a function is obtained by visiting the BDD in top-down mode and inverting the two descendants of each node. This is equivalent to complement the terminal vertices, i.e. 0 to 1 and 1 to 0. As an example the BDD complementing the function in figure A.2 is represented in Figure B.4.

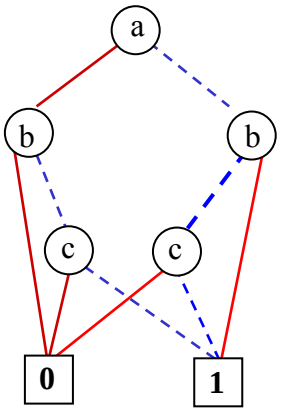


Figure B.4 OBDD representation of the complementation of the OBDD of Fig. B.2

The satisfying set is $\overline{S} = \{a\ \overline{b}\ \overline{c},\ \overline{a}\ \overline{b}\ \overline{c},\ \overline{a}\ b\}$, as can easily be verified by complementing the satisfying set of the BDD in figure B.2.

The elements of a satisfying set S are all mutually exclusive and therefore they are suitable for the probabilistic analysis but they do not represent the minimum system failure modes. When dealing with non-monotone functions the minimum failure modes are referred to as Prime Implicants (PI). An implicant is a conjunction of variables - in positive and negative form - that satisfy the top event. A Prime Implicant (PI) is an implicant that do not contain any other implicant, i.e. if one of its variables is removed it is no more an implicant.

To determine the set of PI the following algorithm was proposed by Rauzy-Dutuit (1997). It is recursively applied to all nodes of the BDD.

Let $f(x_1, x_2, \ldots, x_n)$ be a non-coherent function, $F = f(x_1, x_2,..,1,\ldots, x_n)$ be the left branch of x_i and $G = f(x_1, x_2,..,0,\ldots, x_n)$ be the right branch of x_i

$f = x_i F + \overline{x_i} G$

The set of PI of f is given by:

$\{PI\} = x_i R \cup \overline{x_i} Q \cup P$

where:

$P = F G;$
$R = F \setminus P;$
$Q = G \setminus P$

The operator \ is the operator difference (Rauzy, 1993) e.g. F \ G gives the OBDD of F in which the PI of G satisfying F are removed.

The above algorithm is based on the application of the Consensus operation, which states that:
$x\,\alpha + \beta\,\overline{x} = x\,\alpha + \beta\,\overline{x} + \alpha\,\beta$. The determination of $\alpha\,\beta$ is important being it a real failure mode.

Example

Suppose to be interested in the PI of the BDD represented in Figure A.2 representing the function
$f = (a + \bar{b}\, c)\,(b + c)$

Visiting the BDD in bottom up mode, we have:

Variable c (left and right):
$F = 1$
$G = 0$
$P = 0$
$R = 1$
$Q = 0$
$\{PI\} = x_i\, R \cup \bar{x}_i\, Q \cup P = \{c\}$

Variable b (left):
$F = 1$
$G = c$
$P = c$
$R = 1$
$Q = 0$
$\{PI\} = x_i\, R \cup \bar{x}_i\, Q \cup P = \{b, c\}$

Variable b (right):
$F = 0$
$G = c$
$P = 0$
$R = 0$
$Q = c$
$\{PI\} = x_i\, R \cup \bar{x}_i\, Q \cup P = \{\, \bar{b}\, c \,\}$

Analysis of a:
$F = \{b, c\}$
$G = \{\, \bar{b}\, c \,\}$
$P = \{\, \bar{b}\, c \,\}$
$R = \{b, c\}$
$Q = 0$
$\{PI\} = x_i\, R \cup \bar{x}_i\, Q \cup P = \{a\, b, a\, c, \bar{b}\, c\}$

From this simple example we can notice that the new set P is generated containing the result of the consensus operation.

Rauzy and Dutuit (1997) also recognised that the determination of all PI may be time consuming and in many cases practically useless. In fact PI may contain several negated events (success) and few positive (failure) events. But in practice the interest is in the failed events. Therefore they proposed an algorithm to determine the set of MCS (called p-cuts) from the OBDD.

Coherent functions

Monotone functions can be seen as a particular case of non-monotone functions.

Let f = (a + c) (b + c) be the monotone function to be analysed. Let a < b < c the chosen ordering of variables.

Since a < c, the *ite* representation of (a + c) is obtained applying Rule 1:

ite(a, 1, 0) $\vee$ ite(c, 1, 0) = *ite*(a, 1, *ite*(c, 1, 0)).

Analogously, the *ite* representation of (b + c) is:

ite(b, 1, 0) $\vee$ ite(c, 1, 0) = *ite*(b, 1, *ite*(c, 1, 0)).

Considering the adopted ordering, we get:

ite(a, 1, ite(c, 1, 0)) $\wedge$ ite(b, 1, ite(c 1, 0)) =
ite(a, 1 $\wedge$ ite(b 1, ite(c 1, 0)), ite(c, 1, 0) $\wedge$ ite(b 1, ite(c 1, 0)))

First residue:
1 $\wedge$ ite(b,1, ite(c, 1, 0)) = ite(b, 1, ite(c, 1, 0))

Second residue:
ite(c, 1, 0) $\wedge$ ite(b, 1, ite(c, 1, 0)) = ite(b, 1 $\wedge$ ite(c, 1, 0), ite(c, 1, 0) $\wedge$ ite(c, 1, 0)) = ite(b, ite(c, 1, 0), ite(c, 1, 0)) = ite(c, 1, 0)

The final graph is as follows:
Top = ite(a, ite(b, 1, ite(c, 1, 0)), ite(c, 1, 0))

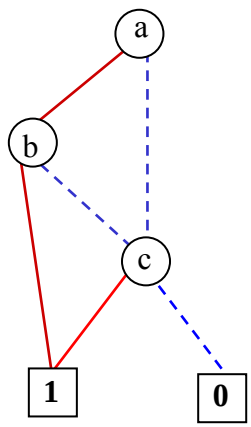


Figure A.5. OBDD representation of the coherent function f = (a + c) (b + c)

The satisfying set is S = {a b, $\bar{a}$ c, a $\bar{b}$ c}. Since all paths are disjoint the exact system failure probability is given by the sum of failure probabilities of all paths.

For coherent functions the satisfying set *S* contains all Cut Sets (CS). Since cut sets are not minimal, the graph must be minimised to remove non-minimal cut sets. It is possible to obtain a minimum OBDD, i.e. a graph embedding all MCS by means of the procedure proposed by Rauzy (1993). The calculated MCS are stored in a compact way as a ZBDD (Minato, 1990).

Let $f(x_1, x_2, \ldots, x_n)$ be a coherent function. For a generic node with the associated variable x_i let $F = f(x_1, x_2,..,1,\ldots, x_n)$ be the left branch of x_i and $G = f(x_1, x_2,..,0,\ldots, x_n)$ be the right branch of x_i

The set of MCS is given by:

$\{MCS\} = x_i\, R \cup G$
$P = F \vee G;$
$R = F \setminus P.$

Let $Z = (z, F, G)$ be the monotonic Boolean function with $F = (x, F_1, F_0)$ and $G = (y, G_1, G_0)$. Three cases are possible:
1) if $x < y$ then $F = ite(x, F_1 \setminus G, F_0 \setminus G)$
2) if $x > y$ then $F = ite(x, F_1 \setminus G_0, F_0 \setminus G_0)$
3) if $x = y$ then $F = ite(x, F_1 \setminus G_1, F_0 \setminus G_0)$

where \ is the "without" operator that removes from F all paths containing paths of G.

<u>Example</u>

Let $Z = ite(a, F, G)$ with $F = ite(b, 1, G)$ and $G = ite(c, 1, 0)$
Ordering of variables: $a < b < c$

The analysis of nodes c and b does not change the graph. Concerning a, since $b < c$, then case # 1 is applied.
$F_1 = 1$
$F_0 = ite(c, 1, 0)$
$G = ite(c, 1, 0)$

Thus, $F_1 \setminus G = 1$ and $F_0 \setminus G = 0$.

Therefore $F = (b, 1, 0)$

This means that the node c is removed, giving: $Z = ite(a, F, G)$, $F = ite(b,1,0)$ and $G = ite(c, 1, 0)$. The BDD embedding all MCS is as follows.

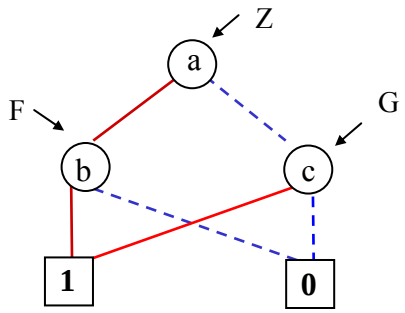


Figure A.5. ZBDD representation of the MCS of the coherent function $f = (a + c)\,(b + c)$
MCS = {c; a b}

An important problem is the selection of the best ordering, i.e. able to minimise the number of nodes of the OBDD. This is a difficult problem that is out of the scope of this brief introduction to BDD.

APPENDIX C

Unconditional Failure and Repair frequencies

In this section the basic rules for calculating the failure and repair frequencies are briefly provided based on simple coherent and not-coherent functions.

In fault tree analysis the most commonly applied logical analysis method is based on Boolean algebra. The basic operators are AND, OR and NOT. We briefly review the equations for determining the unavailability and unconditional failure and repair frequency of basic configurations.

AND

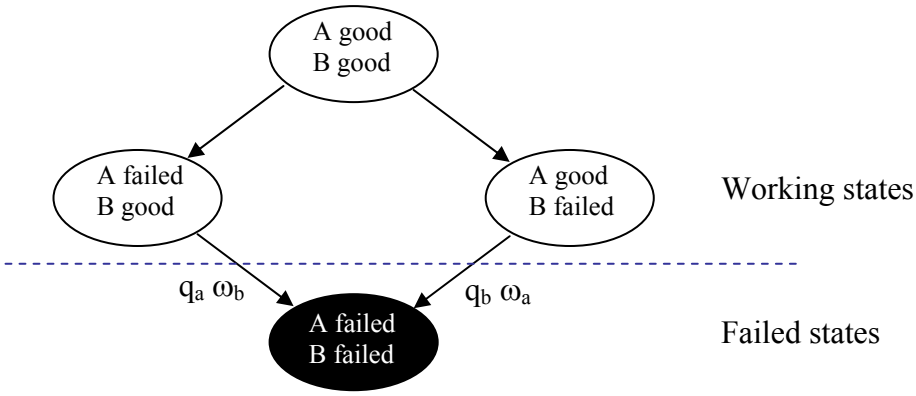
state	a	b	a b	q(t)	ω(t)	ν(t)
0	0	0	0	-	-	-
1	0	1	0	-	-	-
2	1	0	0	-	-	-
3	1	1	1	$q_a\, q_b$	$\omega_a\, q_b + \omega_b\, q_a$	$\nu_a\, q_b + \nu_b\, q_a$

The probability that the combination (a b) occurs (enter into the failed state) in the time interval dt is given by the probability that *a* occurs in t-t+dt (represented by $\omega_a(t)$ dt) with *b* failed at t (represented by $q_b(t)$) <u>or</u> that *b* occurs in t-t+dt with *a* failed at t, i.e.:
$\Omega(a\ b)\, dt = q_b(t)\ \omega_a(t)\, dt + q_a(t)\, \omega_b(t)\, dt$

The probability that the combination (a b) is repaired in the time interval dt is given by the probability that *a* is repaired in t-t+dt (represented by $\nu_a(t)$ dt) with *b* failed at t (represented by $q_b(t)$) <u>or</u> that *b* is repaired in t-t+dt with *a* failed at t, i.e.:
$V(a\ b)\, dt = q_b(t)\ \nu_a(t)\, dt + q_a(t)\, \nu_b(t)\, dt$

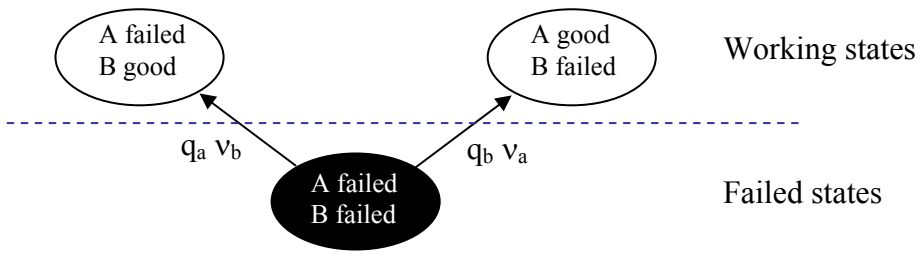
With the aim of simplifying the notation from now on the dependence of time will be omitted, implicitly meaning that the equations are applied at a generic time t.

An equivalent alternative explanation can be given considering the simplified Markov graph of the parallel of two components.



The frequency of entering the failed state is given by the sum of frequencies associated to all links from the set of good states to the set of failed states, i.e.: $\omega = q_b\ \omega_a + q_a\ \omega_b$.

The repair frequency is given by the sum of frequencies associated to all links from failed state to the working states, i.e.: $\nu = q_b\ \nu_a + q_a\ \nu_b$.

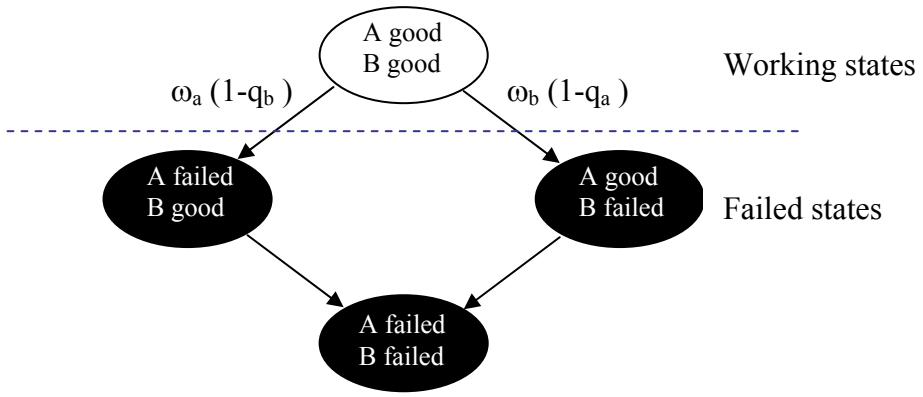


OR

state	a	b	q(t)	ωt)	ν(t)
0	0	0	-	-	-
1	0	1	$(1-q_a)\ q_b$	$\omega_b\ (1-q_a)$	$\nu_b\ (1-q_a)$
1	1	0	$q_a\ (1-q_b)$	$\omega_a\ (1-q_b)$	$\nu_a\ (1-q_b)$
1	1	1	$q_a\ q_b$	$\omega_a\ q_b + \omega_b\ q_a$	$\nu_a\ q_b + \nu_b\ q_a$

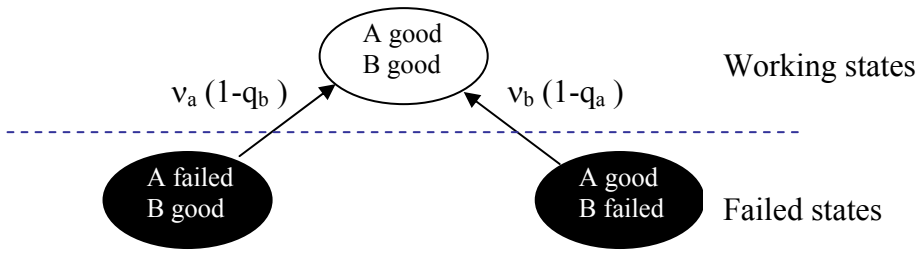
$\Omega(a + b) = \Omega(a) + \Omega(b) - \Omega(a\ b) = \omega_a + \omega_b - \omega_a\ q_b - \omega_b\ q_a$
The same result can be obtained as:
$\Omega(a + b) = \Omega(b\ \bar{a}) + \Omega(a\ \bar{b}) + \Omega(a\ b)$
Since $\Omega(b\ \bar{a}) = \Omega(b) - \Omega(a\ b)$ and $\Omega(a\ \bar{b}) = \Omega(a) - \Omega(a\ b)$ we get:
$\Omega(a + b) = \omega_a - \omega_a\ q_b - \omega_b\ q_a + \omega_b - \omega_b\ q_a - \omega_a\ q_b + \omega_a\ q_b + \omega_b\ q_a = \omega_a + \omega_b - \omega_a\ q_b - \omega_b\ q_a$

Analogously for the OR operator (series of components), we have:



The frequency of entering the set of failed states is given by: $\omega = \omega_a\ (1-q_b) + \omega_b\ (1- q_a)$.

Analogously, for the repair frequency: $\nu = \nu_a + \nu_b - \nu_a\ q_b - \nu_b\ q_a$



Note that in $\Omega(a + b) = \Omega(b\ \bar{a}) + \Omega(a\ \bar{b}) + \Omega(a\ b)$ the NOT operator is introduced to disjoint the three terms, i.e. it has been introduced by the calculation method. This means that there is no connection with non-coherence. Indeed, consider the Karnaugh map of the OR operator:

	A=0	A=1
B=0		•
B=1	•	•

The three disjoint minterms $\overline{A}\,B,\ A\,\overline{B},\ A\,B$ are covered by two MCS: A and B, i.e. the function $\overline{A}\,B + A\,\overline{B} + A\,B$ can be logically reduced to $A + B$, i.e. to a coherent function. Indeed $A\,B + A\,\overline{B} = A\,(B + \overline{B}) = A$. Moreover the consensus between A and $\overline{A}\,B$ gives $A + \overline{A}\,B + B$ which is equal to $A + B$.
This simple example shows that the presence of a negated variable in the fault tree is not sufficient to claim that the function is non-coherent. Only if the negated variables cannot be removed by means of the application of Boolean algebra, up to the determination of the minimum failure combinations, we can say that the function is not coherent.

A simple non coherent function is the XOR of two variables.

XOR

state	a	b	q(t)	ω(t)	ν (t)
0	0	0	-	-	-
1	0	1	$(1-q_a)\,q_b$	$\omega_b\,(1-q_a) + q_b\,\nu_a$	$\nu_b\,(1-q_a) + q_b\,\omega_a$
1	1	0	$q_a\,(1-q_b)$	$\omega_a\,(1-q_b) + q_a\,\nu_b$	$\nu_a\,(1-q_b) + q_a\,\omega_b$
0	1	1	-	-	-

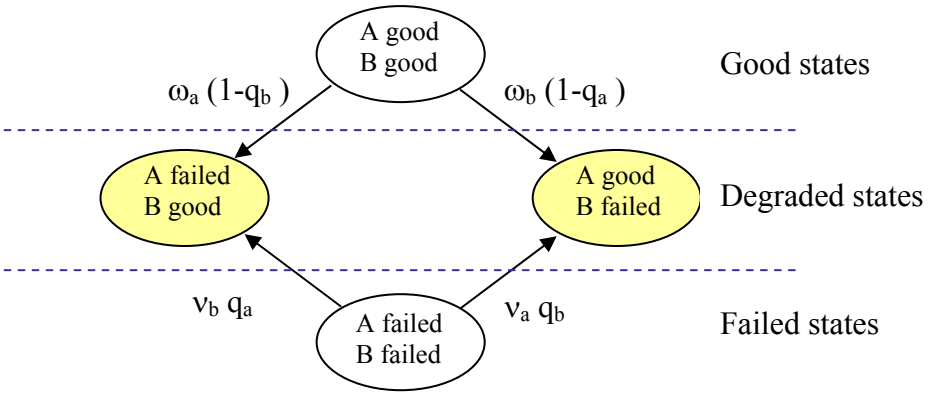
$\Omega(a\,\overline{b} + \overline{a}\,b) = \Omega(a\,\overline{b}) + \Omega(\overline{a}\,b)$
$\Omega(a\,\overline{b} + \overline{a}\,b) = \omega_b\,(1-q_a) + \omega_a\,(1-q_b) + q_a\,\nu_b + q_b\,\nu_a$
Analogously,
$V(a\,\overline{b} + \overline{a}\,b) = \nu_b\,(1-q_a) + q_b\,\omega_a + \nu_a\,(1-q_b) + q_a\,\omega_b$

In this case the NOT operators cannot be removed with logical operations as can be seen from the XOR Karnaugh map which shows that the two minterms $\overline{A}\,B$ and $A\,\overline{B}$ are disjoint.

	A=0	A=1
B=0		•
B=1	•	

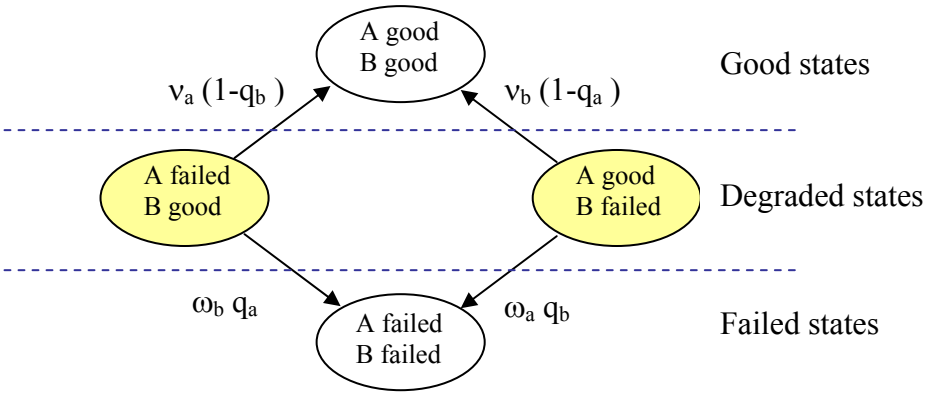
For this reason the failure frequency expression contains ν, since the function is verified not only if *a* is failed and *b* is good (first row), but also when *a* is failed and *b*, being failed, is repaired.

Note that all variables considered so far are independent, which means that the behaviour of a variable (change of state) is not influenced by the behaviour of any other variable.
For instance, consider again the simple system made up by the parallel of two independent components A, B. If we are interested in the probability of both components failed then we model the system by means of the AND operator. However, we may be interested in modelling the degraded states. The simple system has two degraded states: A failed with B working and vice versa. In practical applications the degraded states may be of particular interest e.g. due to the associated risk increase or the application of a given procedure when a component fails. This situation is modelled using the XOR operator. In other words both components, being independent, can be both failed, but we are simply not interested in the system failure state. Hence, the NOT operator has nothing to do with events that are physically disjoint, as for instance the two failed states of a three state component.
This concept is made clear by the following simplified Markov graph in which the three sets of states (good, degraded, failed) are represented.



The frequency of entering the degraded state from good states (failure frequency) is given by:
$\omega = \omega_a (1-q_b) + \omega_b (1- q_a) + \nu_b q_a + \nu_a q_b$.

The frequency of leaving the degraded state is given by:
$\nu = \nu_a (1-q_b) + \nu_b (1- q_a) + \omega_b q_a + \omega_a q_b$.



From what has been described it can be seen that the equation for determining the repair frequency can be obtained from the equation for failure frequency by changing ω with ν and ν with ω.

APPENDIX D

Determination of p_x^f *and* p_x^r *on a modularised fault tree*

Let $\Phi = M\,\Phi_1 + \overline{M}\,\Phi_0 + \Phi_1\,\Phi_0$

be the non-coherent function Φ expanded with respect to the module M where Φ_1 and Φ_0 are the residues.

Let $M = x\,M_1 + \overline{x}\,M_0 + M_1\,M_0$

be the function of the module M(**x**) expanded with respect to the variable x and $\overline{M} = x\,\overline{M}_1 + \overline{x}\,\overline{M}_0 + \overline{M}_1\,\overline{M}_0$ its complemented form.

Passing to probabilities:

$$P(\Phi) = P(M)\,[P(\Phi_1) - P(\Phi_1\,\Phi_0) + P(\overline{M})\,[P(\Phi_0) - P(\Phi_1\,\Phi_0)] + P(\Phi_1\,\Phi_0)$$

$$P(M) = P(x)\,[P(M_1) - P(M_1\,M_0)] + P(\overline{x})\,[P(M_0) - P(M_1\,M_0)] + P(M_1\,M_0)$$

$$P(\overline{M}) = P(x)\,[P(\overline{M}_1) - P(\overline{M}_1\,\overline{M}_0)] + P(\overline{x})\,[P(\overline{M}_0) - P(\overline{M}_1\,\overline{M}_0)] + P(\overline{M}_1\,\overline{M}_0)$$

The above equations can also be written as:

$$P(\Phi) = P(M)\,P(\Phi_1\,\overline{\Phi_0}) + P(\overline{M})\,P(\Phi_0\,\overline{\Phi_1}) + P(\Phi_1\,\Phi_0)$$

$$P(M) = P(x)\,P(M_1\,\overline{M_0}) + P(\overline{x})\,P(M_0\,\overline{M_1}) + P(M_1\,M_0)$$

$$P(\overline{M}) = P(x)\,P(\overline{M}_0\,M_1) + P(\overline{x})\,P(\overline{M}_0\,M_1) + P(\overline{M}_1\,\overline{M}_0)$$

Now, the importance of $x \in \Phi$ is obtained when:

$(x \in M$ and $M \in \Phi)$ or $(\overline{x} \in M$ and $\overline{M} \in \Phi)$

Analogously, the importance of $\overline{x} \in \Phi$ is obtained when:

$(x \in M$ and $\overline{M} \in \Phi)$ or $(\overline{x} \in M$ and $M \in \Phi)$

Indicating with p_x^f the probability of the system critical state for the failure of x with respect to Φ, and with p_x^r the probability of the system critical state for the repair of x with respect to Φ, we can write:

1) For the importance of $x \in \Phi$:

$$p_x^f = \frac{\partial P(\Phi)}{\partial P(M)}\frac{\partial P(M)}{\partial P(x)} + \frac{\partial P(\Phi)}{\partial P(\overline{M})}\frac{\partial P(M)}{\partial P(\overline{x})} =$$

$$= P(\Phi_1\,\overline{\Phi_0})\,P(M_1\,\overline{M_0}) + P(\Phi_0\,\overline{\Phi_1})\,P(M_0\,\overline{M_1}) = p_M^f\,p_{xM}^f + p_{\overline{M}}^f\,p_{\overline{x}M}^f$$

2) For the importance of $\overline{x} \in \Phi$:

$$p_{\overline{x}}^f = \frac{\partial P(\Phi)}{\partial P(\overline{M})}\frac{\partial P(M)}{\partial P(x)} + \frac{\partial P(\Phi)}{\partial P(M)}\frac{\partial P(M)}{\partial P(\overline{x})} =$$

$$= P(\Phi_0\,\overline{\Phi_1})\,P(M_1\,\overline{M_0}) + P(\Phi_1\,\overline{\Phi_0})\,P(M_0\,\overline{M_1}) = p_{\overline{M}}^f\,p_{xM}^f + p_M^f\,p_{\overline{x}M}^f$$

Considering that $p_{\overline{x}}^f = p_x^r$ and vice versa, the above equation can also be written as:

$$p_x^f = p_M^f \; p_{xM}^f + p_M^r \; p_{xM}^r \qquad \text{(A2.1)}$$

$$p_x^r = p_M^r \; p_{xM}^f + p_M^f \; p_{xM}^r \qquad \text{(A2.2)}$$

where

$p_M^f = P(\Phi_1 \; \overline{\Phi_0})$ is the probability of the system critical state for the failure of M in Φ;

$p_M^r = P(\overline{\Phi_1} \; \Phi_0)$ is the probability of the system critical state for the repair of $\overline{M}$ in Φ.

$p_{xM}^f = P(M_1 \; \overline{M_0})$ is the probability of the critical state of x in M.

$p_{xM}^r = P(\overline{M_1} \; M_0)$ is the probability of the critical state for the repair of x in M.

The dependence of time of the above equations is not represented for the sake of simplicity, but it is understood that p_x^f and p_x^r are calculated at a give time t.

Equations A2.1 and A2.2 are valid for DF variables; simpler relationships can be derived for SP and SN variables.

SP variables

The importance of $x \in \Phi$ is obtained when: $x \in M$ and $M \in \Phi$

Equations are derived from A2.1 and A2.2 considering that for coherent positive variables $\Phi_1 \; \Phi_0 = \Phi_0$ and $M_1 \; M_0 = M_0$; consequently $\overline{\Phi_1} \; \Phi_0 = 0$ and $\overline{M_1} \; M_0 = 0$. Hence

$$p_x^f = \frac{\partial P(\Phi)}{\partial P(M)} \frac{\partial P(M)}{\partial P(x)} = P(\Phi_1 \; \overline{\Phi_0}) \; P(M_1 \; \overline{M_0}) = p_M^f \; p_{xM}^f \qquad \text{(A2.3)}$$

$$p_{\overline{x}}^f = 0 \qquad \text{(A2.4)}$$

SN variables

Analogously, the importance of $\overline{x} \in \Phi$ is obtained when: $\overline{x} \in M$ and $M \in \Phi$

Equations are derived from A2.1 and A2.2 considering that for negated variables $\Phi_1 \; \Phi_0 = \Phi_1$ and $M_1 \; M_0 = M_1$; consequently $\overline{\Phi_0} \; \Phi_1 = 0$ and $\overline{M_0} \; M_1 = 0$. Hence

$$p_x^f = 0$$

$$p_x^r = \frac{\partial P(\Phi)}{\partial P(M)} \frac{\partial P(M)}{\partial P(\overline{x})} = P(\Phi_1 \; \overline{\Phi_0}) \; P(M_0 \; \overline{M_1}) = p_M^f \; p_{xM}^r$$

European Commission

EUR 56318 EN – Joint Research Centre – Institute for the Protection and Security of the Citizen
Title: ASTRA 3.0: Logical and probabilistic analysis methods
Author(s): Sergio Contini, Vaidas Matuzas
Luxembourg: Publications Office of the European Union
2010 – 80 pp. – 21 x 29.7 cm
EUR – Scientific and Technical Research series – ISSN 1018-5593
ISBN 978-92-79-14857-6
DOI 10.2788/55214

Abstract
This report contains the description of the main methods, implemented in ASTRA 3.0, to analyse coherent and non-coherent fault trees. ASTRA 3.0 is fully based on the Binary Decision Diagrams (BDD) approach. In the case of non-coherent fault trees ASTRA 3.0 dynamically assigns to each node of the graph a label that identifies the type of the associated variable in order to drive the application of the most suitable analysis algorithms. The resulting BDD is referred to as Labelled BDD (LBDD). Exact values of the unavailability, expected number of failure and repair are calculated; the unreliability upper bound is automatically determined under given conditions. Five different importance measures of basic events are also provided. From the LBDD a ZBDD embedding all the MCS is obtained from which a subset of Significant Minimal Cut Sets (SMCS) is determined through the application of the cut-off techniques.
With very complex trees it may happen that the working memory is not sufficient to store the large LBDD structure. In these cases ASTRA 3.0 performes the analysis by constructing a Reduced ZBDD embedding the SMCS - using cut-off techniques - thus by-passing the construction of the LBDD.
The report also contains three short tutorials on the BDD approach, on the usefulness of non-coherent fault trees and on the determination of failure and repair frequencies.

How to obtain EU publications

Our priced publications are available from EU Bookshop (http://bookshop.europa.eu), where you can place an order with the sales agent of your choice.

The Publications Office has a worldwide network of sales agents. You can obtain their contact details by sending a fax to (352) 29 29-42758.

The mission of the JRC is to provide customer-driven scientific and technical support for the conception, development, implementation and monitoring of EU policies. As a service of the European Commission, the JRC functions as a reference centre of science and technology for the Union. Close to the policy-making process, it serves the common interest of the Member States, while being independent of special interests, whether private or national.

JRC
EUROPEAN COMMISSION

LB-NA-24152-EN-C

Publications Office
Publications.europa.eu

ISBN 978-92-79-14857-6
9 789279 148576