Dynamic Model-based Safety Analysis:
From State Machines to Temporal Fault Trees

By

Nidhal Mahmud

Thesis submitted for the degree of Doctor of Philosophy

Submitted on:	19 June 2012
Viva Voce Defence on:	4 September 2012
Revised on:	29 October 2012

Externally examined by: Professor Jean-Marc FAURE

Laboratoire Universitaire de Recherche en Production Automatisée (LURPA)
École Normale Supérieure (ENS) de Cachan

Internally examined by: Dr Bing Wang

The University of Hull
Department of Computer Science

Rapporteur of the jury: Mr Warren Viant

The University of Hull
Department of Computer Science

بسم آلله آلرحمن آلرحيم

It is not arduous to return to the same place, but not in the same state of fineness in this machine of brutes, where destiny may be chosen at each join state, to the final state…

Nidhal Ali Mahmud نضال علي محمود

To the memory of my father Ali Ahmad Mahmud ألمرحوم علي أحمد محمود

Abstract

Finite state transition models such as State Machines (SMs) have become a prevalent paradigm for the description of dynamic systems. Such models are well-suited to modelling the behaviour of complex systems, including in conditions of failure, and where the order in which failures and fault events occur can affect the overall outcome (e.g. total failure of the system). For the safety assessment though, the SM failure behavioural models need to be converted to analysis models like Generalised Stochastic Petri Nets (GSPNs), Markov Chains (MCs) or Fault Trees (FTs). This is particularly important if the transformed models are supported by safety analysis tools.

This thesis, firstly, identifies a number of problems encountered in current safety analysis techniques based on SMs. One of the existing approaches consists of transforming the SMs to analysis-supported state-transition formalisms like GSPNs or MCs, which are very powerful in capturing the dynamic aspects and in the evaluation of safety measures. But in this approach, qualitative analysis is not encouraged; here the focus is primarily on probabilistic analysis. Qualitative analysis is particularly important when probabilistic data are not available (e.g., at early stages of design). In an alternative approach though, the generation of combinatorial, Boolean FTs has been applied to SM-based models. FTs are well-suited to qualitative analysis, but cannot capture the significance of the temporal order of events expressed by SMs. This makes the approach potentially error prone for the analysis of dynamic systems. In response, we propose a new SM-based safety analysis technique which converts SMs to Temporal Fault Trees (TFTs) using Pandora — a recent technique for introducing temporal logic to FTs. Pandora provides a set of temporal laws, which allow the significance of the SM temporal semantics to be preserved along the logical analysis, and thereby enabling a true qualitative analysis of a dynamic system. The thesis develops algorithms for conversion of SMs to TFTs. It also deals with the issue of scalability of the approach by proposing a form of compositional synthesis in which system large TFTs can be generated from individual component SMs using a process of composition. This has the dual benefits of allowing more accurate analysis of different sequences of faults, and also helping to reduce the cost of performing temporal analysis by producing smaller, more manageable TFTs via the compositionality.

The thesis concludes that this approach can potentially address limitations of earlier work and thus help to improve the safety analysis of increasingly complex dynamic safety-critical systems.

List of abbreviations

AADL	Architecture Analysis and Design Language
ABS	Anti-lock Braking System
BBW	Brake By Wire
CFT	Component Fault Tree
DCCA	Deductive Cause Consequence Analysis technique
DFOA	Deductive Failure Order Analysis
DFT	Dynamic Fault Tree
EAST-ADL	Electronic Architecture and Software Technology — Architecture Description Language
ECU	Electronic Control Unit
FBD	Functional Block Diagram
FFA	Functional Failure Analysis
FHA	Functional Hazard Assessment
FMEA	Failure Modes and Effects Analysis
FMECA	FMEA extension for criticality analysis
FPM	Failure Propagation Model
FPTN	Failure Propagation & Transformation Notation
FSAP/NuSMV-SA	Formal Safety Analysis Platform equipped with the NuSMV model checker for Safety Analysis
FTA	Fault Tree Analysis
GSPN	Generalised Stochastic Petri Net
HiP-HOPS	Hierarchically Performed Hazard Origin and Propagation Studies
IF-FMEA	Interface Focused FMEA
MC	Markov Chain
OSATE	Open Source AADL Tool Environment
PANDORA	Hour or “time” (ORA in Greek) of Priority-AND gates
PASA	Preliminary Aircraft Safety Assessment
PSSA	Preliminary System Safety Assessment
SAFORA	State Automata to Fault-trees extended with temporal information (ORA in Greek)
SEFT	State-Event Fault Tree

SM	State Machine
SSA	System Safety Assessment
TFT	Temporal Fault Tree

Contents

List of tables

List of figures

Acknowledgements

Several persons have accompanied me in the long journey of this PhD project. My grateful thanks go to Professor Yiannis Papadopoulos who supervised my studies, gave me support and invaluable advice throughout, and I am deeply grateful to him for the opportunity to enter the world of safety analysis. However this project, simply, would not have even started without Professor Abdelrahim A. Hunaiti, Professor Said Ghoul and Dr Mourad Maouche, my superior colleagues of Philadelphia University Jordan who recommended me and tremendously encouraged me, and there are no words that can express my deepest and grateful thanks to them. I am also very grateful to Dr Leonardo Bottaci and Dr Bing Wang for all the meetings and discussions shared throughout this project. Also, I pay my sincere respects to the authors of the journal "Efficient reachability analysis for Time Petri Nets", Dr Rachid Hadjidj and Professor Hanifa Boucheneb. The representation using Latex of the algorithms specific to their journal impressed me very much and highly encouraged me to convert my own algorithms into Latex, in this thesis and in the papers listed in the next declaration. Moreover, my grateful thanks go to Professeur Marc Bouissou and to Dr Max Walter for our fruitful discussions about Markov models in Vienna and for the exchange of emails afterwards. I am also very thankful to all those who contributed immensely in the work on severity and effects of the brake-by-wire failures, including Dr Martin Walker who has made the study of the effects of different sequences of failures available. I evaluated the concepts and methods presented in this thesis in reference to this study. I am also thankful to all my colleagues, particularly those who started their studies in the same period as me, Dr Amer Dheedan and Dr Shawulu Nggada as well as all those who demonstrated in the labs with me, particularly Dr Septavera Sharvia, Dr Ian Wolforth, Dr Zurinahni Zainol, Zhibao Mian, Nabil Abu Hashish, Julius Nganji and Lisa Moore. Finally, my thanks go to the Computer Science department personnel for always being wonderful in sorting out the administrative and practical issues that I encountered in my work, particularly Amanda Millson, Helen El-Sharkawy, Colleen Nicholson, Joan Hopper and Jo Clappison; for the computer, printer and devices my thanks go particularly to Adam Hird and to the memory of Debbie Clayton for always preserving a high quality of her work despite the struggle with her health condition.

Deepest thanks go to my old good friends and much more than that, but I cannot find the word that can express it, particularly Kamel Kadri, Moumen Agoulmine, Nacéra Madani, Dalila Allam, Saïd Abdeddaïm, Mourad and Farid Naït-Abdesselam, Tarek

Branki, Nacéra Bensaou, Linda Demil, Salima and especially Fariza Tahi whose distinguished achievements remain always a constant source of inspiration, I am always indebted and so grateful to them.

Above all, my highest respects and most grateful thanks go to my parents, sisters and brothers; my heart has always been with them. Likewise, there are no words that can express my sincere respects and admiration to my uncles, especially Wassef, Medhat and Jawdat who are the distinguished, elite professors of the “state machine” of life. Nevertheless, I cannot finish without expressing my huge grateful thanks and respects to all my former teachers, especially my mathematic teachers in Algiers from my early childhood to University time.

Finally, I acknowledge that this PhD research was almost completely funded by the projects: Model-based Analysis & Engineering of Novel Architectures for Dependable Electric Vehicles EU FP7 project (MAENAD www.maenad.eu), Advancing Traffic Efficiency and Safety through Software Technology EU FP7 project phase 2 (ATESST2 www.atesst.org), the Safety Design Operation and Regulation EU FP6 project (SAFEDOR www.safedor.org) and the University of Hull.

Author’s declaration

I declare that the material contained in this thesis represents original work undertaken solely by the author. The various aspects of the work covered in this material have been presented in a number of international conferences and scientific publications.

The work on the conversion of state machines to temporal fault trees (chapter four of the thesis) was presented in (Mahmud, Papadopoulos & Walker, 2010). The complexity analysis together with the formal version of the conversion algorithm (same chapter of the thesis) was presented in a journal in (Mahmud, Walker & Papadopoulos, 2012). The work on the compositional synthesis of temporal fault trees from state machines (chapter five) was presented in (Mahmud, Walker & Papadopoulos, 2011) and in a journal extension in (Mahmud, Walker & Papadopoulos, 2012). Parts of the literature chapter of the thesis (chapter two) contributed to some European project deliverables in [(Törngren *et al.*, 2008) and (Walker *et al.*, 2010)].

1 Introduction

1.1 Setting the scene:

Sophisticated computerised distributed engineering systems become increasingly more prevalent. Yet our dependence on, for example, systems for transport, production plants or medical devices comes with a potential of harm should any of these systems fail. Such systems are commonly called *safety-critical* systems, and they also include power plants, air traffic control systems, spaceships and many other complex systems.

Several definitions of the term *safety* appear in the literature. It is outlined by one of these as an extension of *reliability* (Avižienis *et al.*, 2001), but from a standpoint of possible catastrophic failures[1]; and where reliability is defined as a measure of the continuous delivery of correct service. Thus, a system is in a safe state when it is in any state of correct service, or when it is in a state of incorrect service but due to non-catastrophic failures. For example, an aircraft is in a safe state while it is correctly taxiing or flying; and the same applies as long as it is flying safely with one engine only due to a failure. In (Avižienis *et al.*, 2001), reliability and safety are both defined as attributes of a wider scope concept called *dependability*. The dependability of a computing system can be defined as "the ability to deliver service that can justifiably be trusted". Avižienis (Avižienis *et al.*, 2001) defines dependability as "the ability of a system to avoid failures that are more frequent or more severe, and outage durations that are longer, than is acceptable to the user(s)" where a user can be another interacting system. In general dependability is seen as an umbrella term that includes safety, reliability, availability, security and maintainability.

A key objective in the design of safety-critical systems is the identification of potential hazards posed by such systems and the minimisation of the likelihood of these hazards. To achieve this, analysts need to understand how systems may fail by investigating potential causes and to estimate the probability of hazardous system failures, so that a system's corresponding design can be adjusted with overall failure preventive measures which render the system safer — this process is known as *safety analysis.* The analysis can be *qualitative* by attempting to determine the necessary and

[1] For studies on basic dependability concepts, taxonomy of faults and fault tolerance, the reader is referred to (Laprie, 1985) and (Laprie, 1998) and (Avižienis *et al.*, 2004).

sufficient causes, or combinations of such causes, that lead the system to a hazardous state. On the other hand, safety analysis can be *quantitative* and the objective is to estimate the probabilities of hazardous system failures from the estimated probabilities of component failures. Both types of analyses are often done using a Boolean combinatorial model called the *fault tree (FT)*[2] — logical model which shows how low level failures logically combine and propagate to cause system failures. Although fault tree analysis has been proved to be a robust technique, the application of this technique is challenged by the dynamic nature of modern systems. Safety-critical systems indeed become increasingly *dynamic*, i.e. exhibit *dynamic behaviour* undergoing a plethora of mode and state transitions in the context of operation.

A simple example of dynamic behaviour that has implications for safety analysis is given in Figure 1—1 (Mahmud, Papadopoulos, & Walker, 2010). The system illustrated in the figure represents a Primary-Standby (PS) redundant system. It is generic in the sense that components A (i.e. the primary) and B (i.e. the backup) can be any sensing, control or actuating device. S is a monitoring sensor whose role is to activate B upon detection of an output deviation from A (e.g. omission of output). I represents the input to the system – input to each of the two redundant components, and Out is the output of the system. Out must receive input from either A or B for the PS system to function.

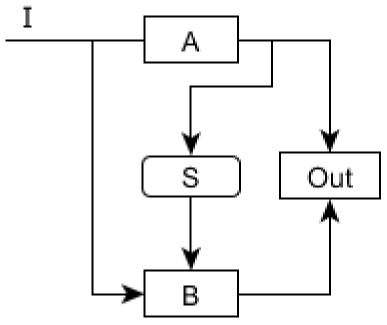


Figure 1—1. Simple example of a primary standby system

This example system exhibits dynamic behaviour; for instance it can switch from primary component A to backup B during its operation — i.e., the state of the system moves from

[2] Direct acyclic graphs which represent how several logical combinations of component failures (events) lead a composed system to a total failure. Graphically, failure events are interconnected through Boolean gates, mainly conjunction (AND gates) and disjunction (OR gates), and as illustrated with an example later in this chapter.

state1 = [A active, B OFF] to state2 = [A failed, B ON]. This behaviour renders the system as fault tolerant since a failure of A can be tolerated without loss of system function. Looking at scenarios of system failure in more detail, we can observe that the supposedly fault tolerant system is not always fault tolerant. For instance, if the system is operating with its primary component but with a failed monitoring sensor, so it is in a state such as state3 = [A active, S failed, B OFF], then a failure of A will immediately cause a total failure of the PS example system (B will not be activated since S has already failed). It is clear that the order of failure of A and S is important for the system, since the outcomes are so different when A fails before or after S. This creates implications for fault tree analysis, a technique unable to capture the ordering of events but also creates difficulties in the analysis of dynamics systems in general. We shall return to this example later on when we lay out in more detail the motivation of this work.

For better capturing behaviour in dynamic systems, states-transition formalisms are increasingly being used within system design languages. Descriptions can be from a functional viewpoint – e.g. to show how an aircraft system moves from taxiing to flying mode, or from the viewpoint of representing possible dysfunctional aspects which degrade systems or cause total failure. This latter kind of dynamics' aspects, from this point onwards, will be designated by the name of *failure behaviour* — behaviour of systems under failure conditions — and is a central issue in this thesis. Recently proposed system description languages which incorporate states-transitions formalisms for describing dynamic behaviour include but are not limited to:

(1) AltaRica[3] — a formal verification technique, which has demonstrated its value in formally specifying the behaviour of complex systems when faults occur (Griffault A. , 2003). The language underlying formalism is a state-transition based system (Rauzy, 2002).

(2) FSAP/NuSMV-SA[4] — a Formal Safety Analysis Platform equipped with a model checker, i.e. NuSMV, for Safety Analysis. Like AltaRica, and formal verification

[3] The language AltaRica was issued in 1996 from an industrial collaboration with an academic and research institution – LABRI (Laboratoire Bordelais de Recherche en Informatique). For methodological studies on AltaRica, the reader is referred to (Arnold *et al.*, 2005).

[4] FSAP/NuSMV has also been developed in collaboration between research institutions with industry. This was under the auspices of the Enhanced Safety Assessment for Complex Systems (ESACS) European project. For methodological studies on the NuSMV model checker, the reader is referred to (Cavada *et al.*, 1998).

techniques in general, this language has the potential of dealing with complex behavioural aspects of dynamic systems. FASP/NuSMV descriptions have a state-transition based representation (Bozzano & Villafiorita, 2003).

(3) AADL[5] — an Architecture Analysis and Design Language intended to be the aerospace standard. This description system has also a state-transition formalism for modelling the behaviour of systems[6], including in conditions of failure (Feiler & Rugina, 2007).

Furthermore, representations such as *Generalised Stochastic Petri Nets (GSPNs)* and *Markov Chains (MCs)* are also state-transition formalisms. It is emphasised, however, in [(Codetta-Raiteri, 2005), (Rugina, 2007) and (Walker, 2009)] that GSPNs and MCs have demonstrated their importance in quantitative analysis (i.e. to compute probabilistic measures of failures); in addition, these models present rather low-level formalisms (Rauzy, 2002) – higher level descriptions are often converted into GSPNs and MCs for quantitative analysis, and thus these models are not well suited to capture complex systems' dynamics, and whose representations often need to be abstracted.

In (Papadopoulos Y. , 2000) for instance, it is emphasised that complex system behaviour can rather be expressed as a hierarchy of *State Machines (SMs)* or mode charts[7], in order to break the corresponding description down into smaller streamlined constituents – an abstract state represents a set of states at a lower level of the hierarchy. Furthermore, we find the SM abstraction concept in AADL as well; a SM representation of the failure behaviour of a component can also represent its subcomponents in the

[5] AADL has been developed under the auspices of the International Society of Automotive Engineers (SAE) – Avionics Systems Division (ASD). For studies on architectural (and behavioural) descriptions, the reader is referred to (Feiler et al, 2006). For studies on dependability modeling and error annex, the reader is referred to (Feiler & Rugina, 2007) and (AADL-Committee-As2cAnnexE, 2006), respectively.

[6] See the behaviour annex in (AADL-Subcommittee, 2007) and (Gaufillet et al, 2006).

[7] A mode being defined in (Papadopoulos, 2000) as an abstract functional state in which the system maintains a stable functional profile – e.g. a flying mode for an aircraft is the set of states in which the system maintains a stable flying profile. Thus, a system's dynamic behaviour is expressed as a set of different functional states (i.e. a set of modes) plus a set of modes' transitions. Degraded modes are those where there is partial loss of functionality, but in which the system is still operational. Failed modes are those in which there is total loss of the intended functionality, and thus rendering the situation hazardous.

presence of failures as an abstraction (Feiler & Rugina, 2007). Therefore, SMs are an expressive, high-level form of representation well-suited for modelling the *nominal* as well as the failure behaviour of safety-critical systems; the nominal behaviour of a system is collectively known as the behaviour of the system without failure conditions.

This thesis aims to improve dynamic safety analysis, and the emphasis is therefore placed on descriptions of failure behaviour. Such descriptions have in the past provided input for safety analysis in techniques for example where state automata describing failure behaviour have been converted into analysis models — e.g., GSPNs, MCs or FTs.

The SM of Figure 1—2 for instance (originally used in Mahmud, Papadopoulos, & Walker, 2010) represents the failure behaviour of the primary-standby example system introduced earlier. This SM depicts the significance of the temporal sequencing of failures.

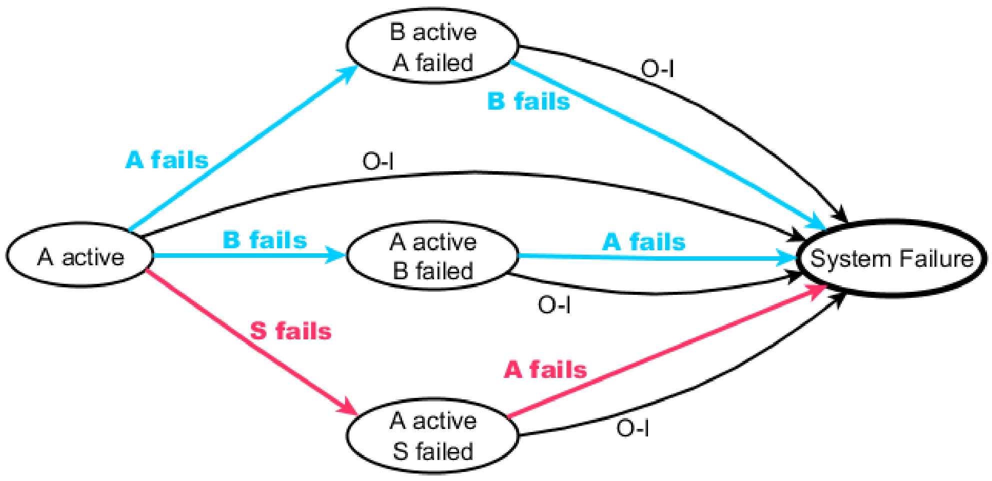


Figure 1—2. Failure behaviour SM of the PS example system

For instance, the sequences [A fails before B] or [B fails before A] both lead the system to a total failure; hence the temporal order is insignificant in this case. On the other hand the sequence [S fails before A] leads to total failure, while [S fails after A] does not cause a failure and therefore is not represented in the SM. If the SM was to be converted in a model suitable for safety analysis (for example a fault tree), then for the accuracy of results it would be important to preserve the significance of the temporal ordering so that only one sequence [S fails before A] is accounted as cause of system failure.

1.2 Research Motivation

One attempt to perform safety analysis on systems modelled with SMs went through transforming their descriptions into GSPNs or MCs. However, this was limited to quantitative analysis as identified earlier in [(Codetta-Raiteri, 2005) and (Rugina, 2007)], and noted in (Walker, 2009) as well. For instance in (Rugina, 2007), this has been applied to AADL models (effectively SMs) by transforming these to GSPNs. This was automated using a transformation tool called ADAPT[8] (from AADL Architectural models to stochastic Petri nets through model Transformation). The tool interfaces from the AADL side with OSATE[9] (an Open Source AADL Tool Environment) and, from the GSPN side, with a dependability evaluation tool Surf-2 (Béounes *et al.*, 1993).

However, performing a qualitative analysis before quantification appears to be beneficial. A central issue is that qualitative analysis can be applied to early models of system design, and where no probabilistic data are available; and then analysis models (e.g., FTs) can be interpreted stochastically to perform probabilistic analysis when failure data becomes available, e.g. in the refined models. Another central issue which favours a prior qualitative analysis is that removing redundancies, or insignificant component failures — those which are irrelevant to a system failure behaviour — can make quantitative analysis more tractable. Something else again which recommends a prior qualitative analysis is that Boolean model assessment tools are much more efficient than those of state-transition models (Rauzy, 2002).

One of the premier and rigorous analysis techniques with an underlying Boolean formalism is *Fault Tree Analysis (FTA)*[10]. FTA has demonstrated its value in a variety of contexts over the years[11], and it is still widely practised in reliability engineering (and in many safety-critical industries such as automotive, aerospace and nuclear). Besides, FTA is employed in conjunction with influential formal languages – e.g. qualification of AltaRica modelling and analysis platform [with a computer performed generation of fault

[8] For details on the tool, the reader is referred to (Rugina *et al.*, 2008).

[9] http://www.aadl.info/aadl/currentsite/tool/toolsets.html#OSATE

[10] For studies on the technique, the reader is referred to the fault tree handbook in (Vesely, 1981).

[11] In the aerospace industry for instance, FTA has been recognised as a significant system safety analysis technique since 1963 (Ericson, 1999).

tree exhaustive *Minimal Cut Sets (MCSs)*[12]] as a validation tool[13], as well as its use (FTA capability within AltaRica) as part of several aerospace projects on, for instance, Airbus civil aircraft programs (Pouzolz, 2010). As for FTA employment with recent *Architecture Description Languages (ADLs)*, several research efforts have been directed towards using the technique to analyse complex systems by automatically generating fault trees from models like in AADL (Joshi *et al.*, 2007).

FTA postulates a catastrophic situation (i.e. a hazard or *top event* in the fault tree vocabulary) which must be avoided. It then reasons backward to identify all scenarios[14] which could conduct the system to that hazard. It thus deduces a hazard cause, or contributory causes (i.e. their logical combinations), and a cause can be for instance an initiating failure of a component of the system under analysis. FTA can be *qualitative* by attempting to minimise the deduced cause combinations, i.e. reducing them to logical expressions that consist of the necessary failures, and which are sufficient to cause the hazard. The technique can also be *quantitative* by combining figures for component failure rates to calculate overall probabilities of system hazards.

For instance, performing FTA on the PS example system of Figure 1—1 will yield the below results, where each is a necessary and sufficient cause, or combination of causes, for the system to fail as a whole.

(1) Omission of Input at I (O-I)

(2) Both A and B fail

(3) Both A and S fail (as B will not be activated)

Thus, these results are called *Minimal Cut Sets (MCSs)* of the fault tree – i.e. disjunction of branches individually leading the system to the top event, and which are graphically represented in Figure 1—3 – i.e. fault tree of the fault tolerant example

[12] A MCS is a conjunct of the necessary and sufficient individual component failures which cause a system hazard, and as illustrated later in this section.

[13] AltaRica has been used in a certification process, and in a first industrial application of *Model Based Safety Analysis (MBSA)*, with Dassault aviation in 2007. For further details about AltaRica employment in recent industrial projects, the reader is referred to (Pouzolz, 2010) and (Bernard & Pouzolz, 2010).

[14] Causal relationships branches of the fault tree.

system. Failures A, B, S and O-I are circled[15] which means they are basic or initiating events (i.e. they cannot be further developed). The gate in the centre, which is used twice, is the logical AND gate and the one above is the logical OR gate. The fault tree top event is the total failure of the PS example system.

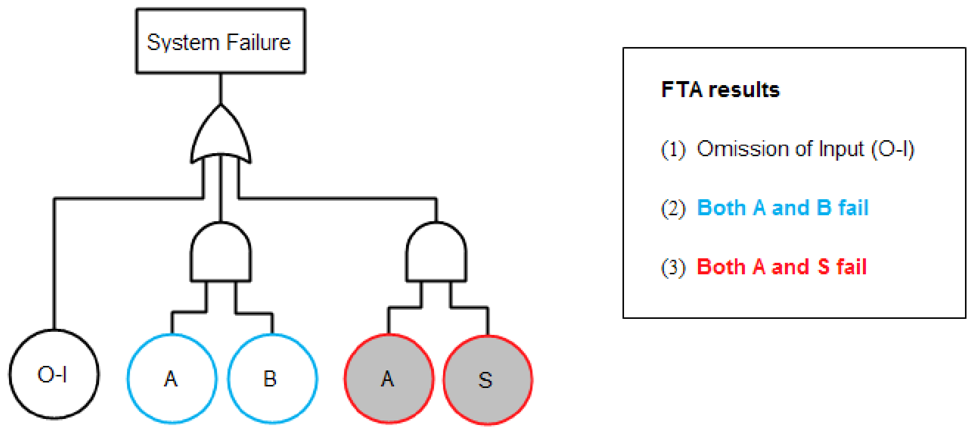


Figure 1—3. Fault tree[16] of the PS example

O-I is a single point of failure (a one-event MCS), and the two other analysis results are also minimal logical combinations of initiating failures of individual components.

MCS (3) [rightmost branch: both A and S fail], however, does not appear to be an accurate analysis result: the appearing conjunct will lead the system to a total failure only if S fails before A, and which means that S will be unable to activate backup B upon omission of output from A (since it has already failed). But, if S fails after A then it has already served its purpose and activated B; thus the failure conjunct will not be a sufficient combination of causes to fail the system as a whole, rendering hence MCS (3) unduly pessimistic. This example shows, therefore, that a fault tree model is not able to cope accurately with systems exhibiting dynamic behaviour; and thus performing analysis on such models will put the obtained results in doubt. These can be dangerously erroneous such as the case of an optimistic analysis result (e.g. where a system fails when

[15] A, B and S are abbreviations for ‘A failure’, ‘B failure’ and ‘S failure’, respectively.

[16] Although shading is meaningless in a fault tree model; the purpose is to distinguish MCS (3) i.e. [A and S fail] from MCS (2) in the centre of the tree, and in term of the significance of the temporal order of failures.

in fact this wasn't predicted), rather than an unduly pessimistic result (a system predicted to fail but it hasn't e.g. the PS example system). The sequence [S fails before A] is a significant temporal order of failures for the example system; hence it needs to be expressed accurately in the analysis model as a sufficient failure cause. Therefore, there is a need to extend fault trees with, at least for the case presented in this example, a modelling capability to express that two failures should occur, but one before another in order to cause a total failure.

This modelling shortage (and any consequent analysis results with the corresponding repercussions on the design) is due to, at the origin, the nature of 'pure' Boolean logic based combinatorial models e.g. conventional fault trees which from this point onwards we will call *static*[17] *fault trees*. This poses a significant problem when it comes to analyse dynamic systems, and where it is imperative to preserve the significance of such dynamic aspects throughout the analysis.

To compensate this shortage, there have been many important attempts to increment the modelling capacity of fault trees, and thence to bridge the gap between FTA modelling tools and the systems being modelled. One of these is the *State-Event Fault Tree (SEFT)*. It is designed to extend the static fault tree with representation capacities for states and events (Kaiser *et al.*, 2007). It, therefore, allows a better capturing of systems state-based descriptions by the analysis model, and hence easily preserving the significance of the sequence-dependant behaviour (which is typically associated with fault tolerant systems). A limitation in using SEFTs as analysis models, however, is that these are not designed for qualitative analysis; they are more suitable for quantitative analysis instead. This is by converting them to other state-transition formalisms such as GSPNs or MCs, and to be then passed to assessment tools e.g. Surf-2 (Béounes *et al.*, 1993) from the GSPN side, or a Markov chain generation algorithm (Vesely *et al.*, 2002) depending on the formalism chosen.

Another attempt in extending fault trees is the *Dynamic Fault Tree (DFT)*. This one adds dynamic gates to the static fault tree (Vesely *et al.*, 2002). The added ingredients allow a more accurate representation of systems where the order in which events occur affects the outcome. The priority AND gate, for instance, and whose output evaluates to true if all of its input events occur on a one-at-a-time basis (one failure occurs before another), solves the problem found in the fault tree of Figure 1—3. This added gate

[17] Its use is more expressive in this thesis, since fault trees capture only the 'static' aspects of dynamic systems.

represents well a much better alternative to the rightmost branch of the fault tree – i.e. MCS (3), such that the ordered sequence [S fails before A] leads the system to a total failure (but not necessarily the inverse). However, and like SEFTs, DFTs are also designed for quantitative analysis. The DFT methodology[18] was introduced to provide a Markov analysis of fault trees (Vesely *et al.*, 2002), fault trees being extended with dynamic gates (hence becoming DFTs) to capture the significance of the temporal sequencing of events, and thereby making it possible to automatically generate the equivalent Markov models, and without a loss of the event-order semantics. As regards the Markov models, however, despite their straightforward representation of the sequence-dependant behaviour in systems, they have their own known flaws (Vesely *et al.*, 2002). They grow easily large and cumbersome, and thus impractical for large systems[19].

An attempt to overcome this weakness in a solution of a Markov model[20] is to provide modular DFT analysis for a system. This means if a system-level fault tree can be divided into independent modules (effectively individual branches leading the system, on their own, to the top event), and if these can be solved separately (through analysis of their corresponding Markov models), then the corresponding results can be combined to achieve a full analysis (Vesely *et al.*, 2002). However, even for a simple example DFT — for a Vehicle Management System (VMS)[21] with just three components mainly (depicted by Figure 1—4), the Markov model is complex and presents several redundant paths. In the VMS example, each one of the three vehicle management components A, B and C is sharing the same spare S with the other components, and through its corresponding Warm SPare gate (WSP)[22]. If a component fails, then S (if it has not already failed and is available) is supposed to take over that failed component's job.

---

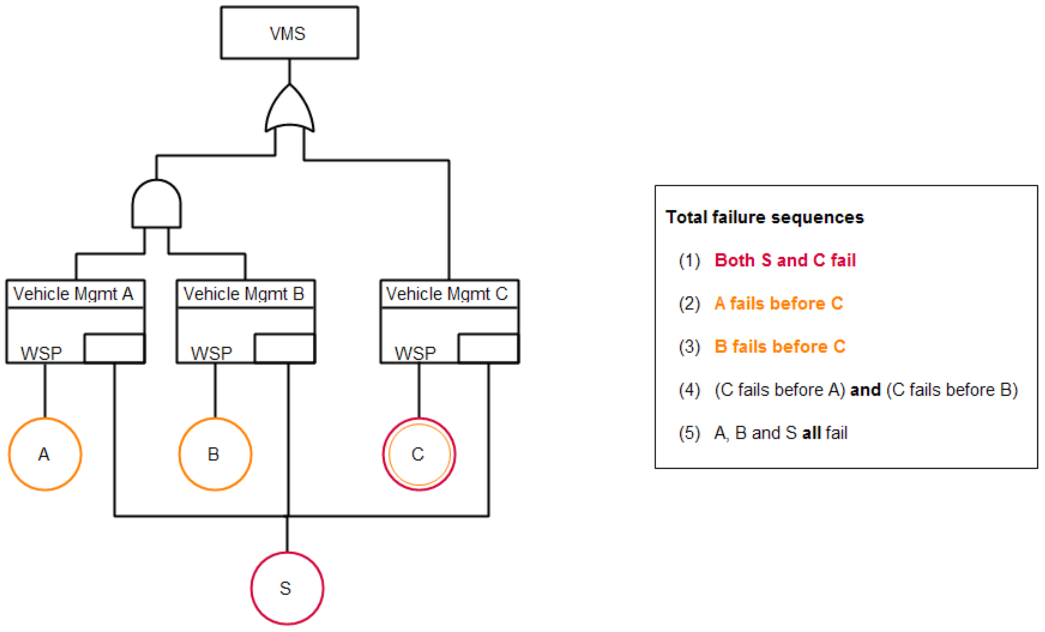
[18] The analysis technique performed on a DFT model.

[19] The number of states and transitions in a Markov model grows exponentially with the number of components in the modelled system (Vesely *et al.*, 2002).

[20] Some quantitative assessments may pass through higher level formalisms (e.g. GSPN) for a following Markov analysis (Rugina, 2007). But, these will also end up being confronted with the disadvantage of a Markov model (state explosion).

[21] The original version of the DFT (and the corresponding Markov model) can be found in (Vesely *et al.*, 2002).

[22] A warm spare has a reduced failure rate before being switched into active use. The other types of spares are cold (do not fail dormant) and hot (maintain their same failure rates) (Vesely *et al.*, 2002).



Total failure sequences

(1) Both S and C fail

(2) A fails before C

(3) B fails before C

(4) (C fails before A) **and** (C fails before B)

(5) A, B and S **all** fail

Figure 1—4. An example DFT for VMS revised from (Vesely *et al.*, 2002)

For the example system to fail as whole, it is sufficient that one of the two DFT disjunction branches results in the top event, e.g. (i) the single WSP branch (rightmost one) evaluates to true, which means that C fails and cannot be replaced by S (e.g. S has failed, or is already replacing either A or B whichever has failed), or C has been replaced by S which has failed in its turn afterwards. Alternatively, and for a total failure situation, it is also sufficient that (ii) the WSP conjunct branch (leftmost one) evaluates to true, and which means that A and B fail together, but none of these can be replaced by S (S has already failed, or is replacing C in response to a corresponding failure); or simply S is already replacing either A or B (whichever failed first), but S has failed in its turn afterwards.

The conversion of a DFT to an equivalent Markov model starts from a state where all the system components are operational, then repeatedly generates a set of target states (and associated transitions) by considering the effect of failure of every component. The same applies to every new state by considering the effects of the remaining failures (not already treated in a same path from the initial state). Any new state can be either an operational or a failed state. The former means that the system is still functioning, despite some component failures. The latter is a state of a total failure of the system. The type of a state (operational or failed) is determined by being checked against the fault tree.

For the fault tree of the given VMS example, its equivalent Markov model is depicted by Figure 1—5 (for clarity, state transitions are shown labelled with individual component failures, rather than failure rates). States are labelled with three consecutive symbols linked to the components A, B and C (in this order). Each symbol represents the status of its corresponding component, for instance, the first symbol can be A (component A is functioning), S (component A is failed but S has taken over), or X (component A is severely failed, meaning that S is also failed or is not available to take over). Therefore if a state is labelled with e.g. SBX, this means that the VMS system is in a state where A has failed and then switched to spare S, B is functioning and C, however, is severely failed (S is no longer available to take over since it is already replacing A). This state i.e. SBX (when checked against the DFT) is a state of a total failure of the system; in Figure 1—4, a severe failure of C – without a possibility to switch to S – is sufficient to cause the top event. To be in this specific state, A has to fail first (S takes over) and C fails afterwards. A reversed order of these failures has a clearly different effect on the system, such that C fails first (then a switch to S) and A fails afterwards (severe failure of A since S is no longer available), but the system can still operate in the state XBS – the DFT shows that also B has to severely fail, so that the AND gate delivers an output which evaluates to true, hence the top event of the fault tree.

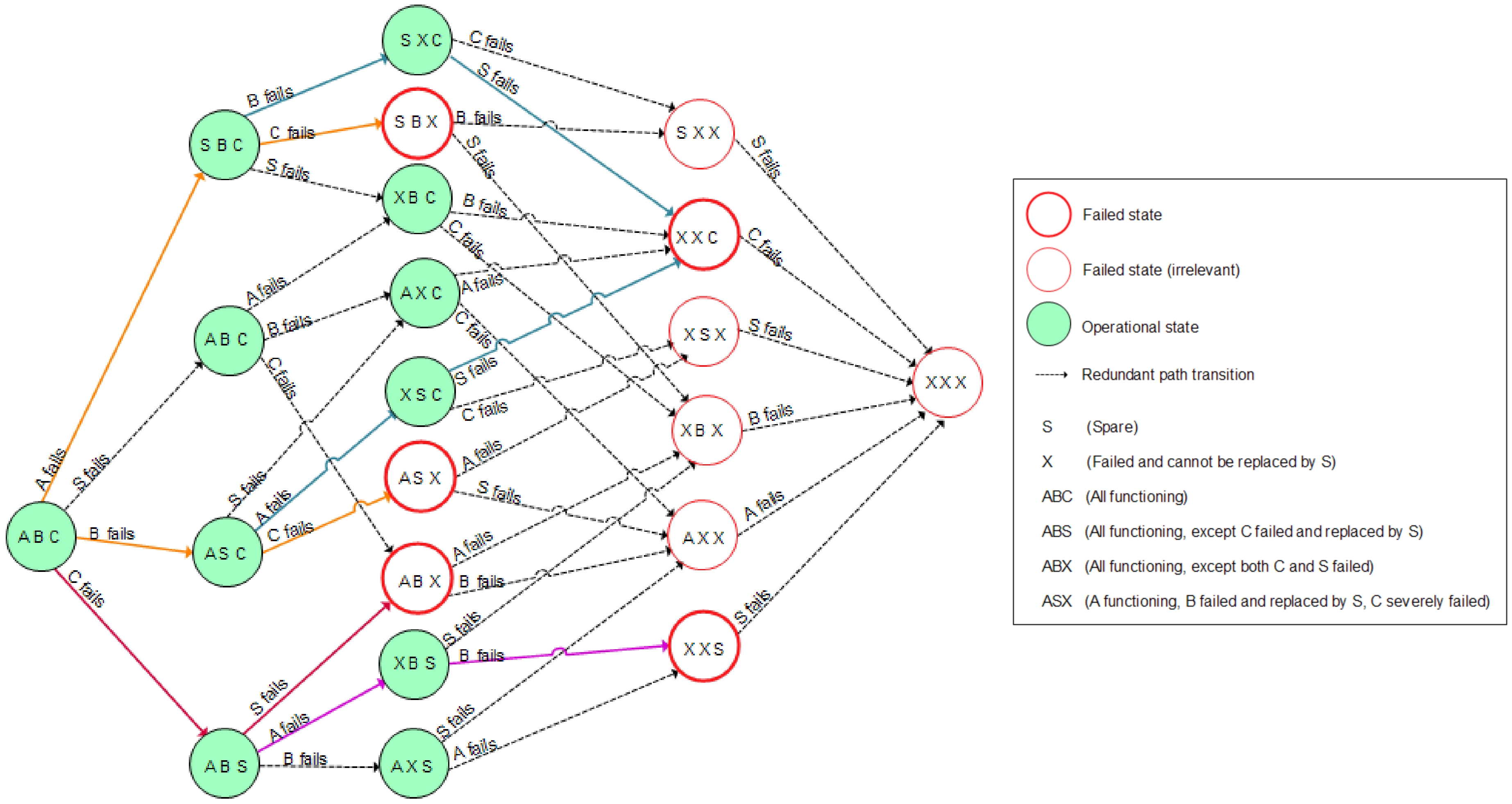
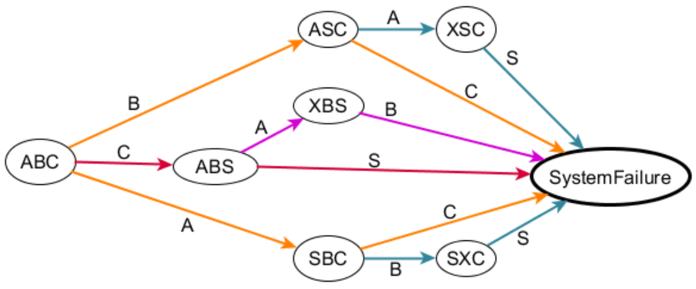

A B C
S B C
A B C
A S C
A B S
S X C
S B X
X B C
A X C
X S C
A S X
A B X
X B S
A X S
S X X
X X C
X S X
X B X
A X X
X X S
X X X
A fails
S fails
B fails
C fails
Failed state
Failed state (irrelevant)
Operational state
Redundant path transition
S (Spare)
X (Failed and cannot be replaced by S)
ABC (All functioning)
ABS (All functioning, except C failed and replaced by S)
ABX (All functioning, except both C and S failed)
ASX (A functioning, B failed and replaced by S, C severely failed)

Figure 1—5. Markov model for the DFT of VMS example

Markov models are solved using ordinary differential equations (Vesely *et al.*, 2002), with one time-dependant probability associated with each state (which represents the probability that the system is in that state) and states are mutually exclusive. However, like the Markov model in Figure 1—5 which corresponds to a relatively small example system, these models are in general complex with a considerable size. Though, in this thesis, we also discuss a possibility for optimising them such that the quantitative results that correspond to the total failures are preserved.

An alternative representation of the VMS failure behaviour can be an abstraction like the SM model of Figure 1—6, and rather than analysing it using standard fault trees (like in Joshi *et al.*, 2007 and Rauzy, 2002) we will show later in this thesis that we can also automatically detect the sequence-dependent failure behaviour, and thereby producing the combinations/sequences of events leading to a complete failure of the system (like those in the right-hand side of the figure). Thereafter, these can be analysed quantitatively, e.g., based on the work in (Merle, 2010) or the quantitative developments[23] of the work in (Walker M., 2009).



Total failure sequences

(1) Both S and C fail

(2) A fails before C

(3) B fails before C

(4) (C fails before A) **and** (C fails before B)

(5) A, B and S all fail

Figure 1—6. Reduced representation for the VMS failure behaviour

This figure[24] shows that the order of failures of any of the two components A or B is only relevant to the system failure behaviour with respect to an occurrence of a failure of the component C. If C fails before any possible failures of the two (A or B) – i.e. transition to state ABS, then all it takes (for a total failure) is either an S failure on its own (and

[23] At the time of writing this thesis, an extension of the work in (Walker M., 2009) with quantitative analysis capabilities is undertaken at the University of Hull.

[24] The transition labels in the figure are abbreviations of component failures, e.g., ‘A’ means ‘A fails’…etc.

irrespective of its order with failure of C), or both A and B failing afterwards (S is not available), and in any order between them[25]. Otherwise, if anyone of A or B fails before C (the system is in the states SBC or ASC, respectively, when C fails), then this is sufficient to cause a total failure, on its own, since S will not be available to take over C. But if C does not fail at all, then failures of all the other components (including S, and irrespective of their orders) become necessary to completely fail this example of VMS.

To summarise, we have outlined through the previous examples two important problems in current safety analysis approaches to systems exhibiting dynamic behaviour. The first problem appears in approaches which convert SMs into static fault trees for the purposes of analysis[26] and which are not always suitable for dynamic systems. Qualitative analysis is based on logic manipulations of Boolean equations and, therefore, the temporal semantics are lost during the assessment, and this can have undesired repercussions which affect the quantitative results afterwards.

The second highlighted problem appears in approaches which though applicable to dynamic systems (e.g. using analysis models such as SEFTs, DFTs or GSPNs) enable mainly quantitative (probabilistic) assessments and not useful qualitative analyses in the absence of probabilistic data about failures.

1.3 Research Hypothesis and Potential Solution

In this thesis, it is speculated that we can analyse non-repairable systems represented as state-automata qualitatively as well as quantitatively, and without any loss of the event-order significance throughout the analysis and that this analysis can in principle be done in a way that can be applied in large systems. Note that this has not been achieved before and therefore represents an original contribution to literature in safety analysis of dynamic systems.

The approach, we propose, allows both qualitative temporal analysis and probabilistic assessments. The analysis is performed on dynamic system models

[25] In this case (after a failure of C), the representation of a B transition before A is redundant and thus irrelevant to the system failure behaviour (i.e. if B fails before a failure of A then, when A fails, there will be two instantaneous transitions to the final state ‘SystemFailure’).
Similarly, the failure-order between S and C is as irrelevant as the failure-order between S, A and B (it is only important to consider the failure-order of C vis-à-vis anyone of A or B).

[26] It rather encourages a prior qualitative analysis and enables following formal assessments throughout the calculation of probabilistic measures (quantitative analysis).

represented as acyclic SMs which are converted to Temporal Fault Trees (TFTs) in a Pandora[27] style (Walker & Papadopoulos, 2006). Pandora is a recent technique of the University of Hull which augments fault trees with temporal information and enables qualitative analysis of that information.

The proposed approach intends to bridge the gap between (1) Boolean-based qualitative analysis (e.g. FTA) and (2) quantitative-only analysis approaches (e.g. Markov solutions for GSPNs, SEFTs or DFTs), by presenting a solution which is primarily twofold.

First, in this thesis we demonstrate the feasibility, and technical applicability, of an automatic generation of Pandora TFTs from SM descriptions of systems which allows the preservation of the significance of the temporal semantics expressed by SMs in the analysis models. For this, we develop an algorithm characterised by its quadratic growth with the number of full paths (from the initial state to the final states) in the SM model. The generated TFTs are processed through a set of Pandora temporal logic rules, to result in the relevant minimal sequences of events which cause a system total failure — i.e., the results of qualitative temporal analysis.

The second element of the approach is a compositional algorithm for constructing large system TFTs from the smaller TFTs of its constituents. Large systems are typically described using smaller models of dynamic behaviour for components. The safety analysis of such systems also demands a compositional approach that can grow and still deliver results as systems grow large in scale and complexity.

1.4 Research Objectives

This first aim of the thesis is to develop an approach that satisfies the hypothesis and it is generic in nature and therefore potentially applicable to any description language that uses a state-transition representation for describing behaviour. Thus, it should be possible to apply this approach to systems described for instance in AltaRica, AADL and EAST-ADL[28] (Freund *et al.*, 2003). A second aim is to develop an approach that can in principle

[27] Pandora means the time, i.e. ORA ([*ώρα*] in Greek), of the Priority-AND (PAND) gate.

[28] The acronym EAST-ADL means Electronic Architecture through Software Technology, Architecture Description Language (we particularly mean EAST-ADL2, i.e. the second version of the language specifications). The language is directed to the automotive industry, and is intended to incorporate state semantics in its domain model.

be automated and in the future incorporated within an automated safety analysis technique like HiP-HOPS[29] (Papadopoulos, 2000). This would mean enabling TFTs, i.e. the automatically generated Pandora failure logic from systems SMs, to be synthesised and analysed within HiP-HOPS. In turn, this would enable taking advantage of further capabilities of this tool. HiP-HOPS is also a dependability versus cost architecture optimisation technique (Parker, 2010). The tool is characterised by its flexibility and its wider scope of application. It has demonstrated its values in analysing (and optimising architectures of) systems described using several languages, e.g. SimulationX models for marine systems (Papadopoulos *et al.*, 2003), Matlab-Simulink models (Papadopoulos *et al.*, 2001) and, most recently, EAST-ADL2 models for automotive systems (Biehl *et al.*, 2010).

The verification of the hypothesis and achievement of the aims above relies on meeting two principal objectives. The first one is an automatic generation of Pandora TFTs from SM models of systems. A key requirement is that the generation algorithm detects the sets of events of which the order is relevant to the system failure behaviour as opposed to those sets in which order is unimportant. A second key requirement is an algorithm which is practically possible in term of complexity and computational expense.

> *In summary, the first objective is an applicable approach to perform a temporal safety analysis which is suitable for complex dynamic systems, and whose behavioural aspects are represented using SMs (or generally, state-transition descriptions).*

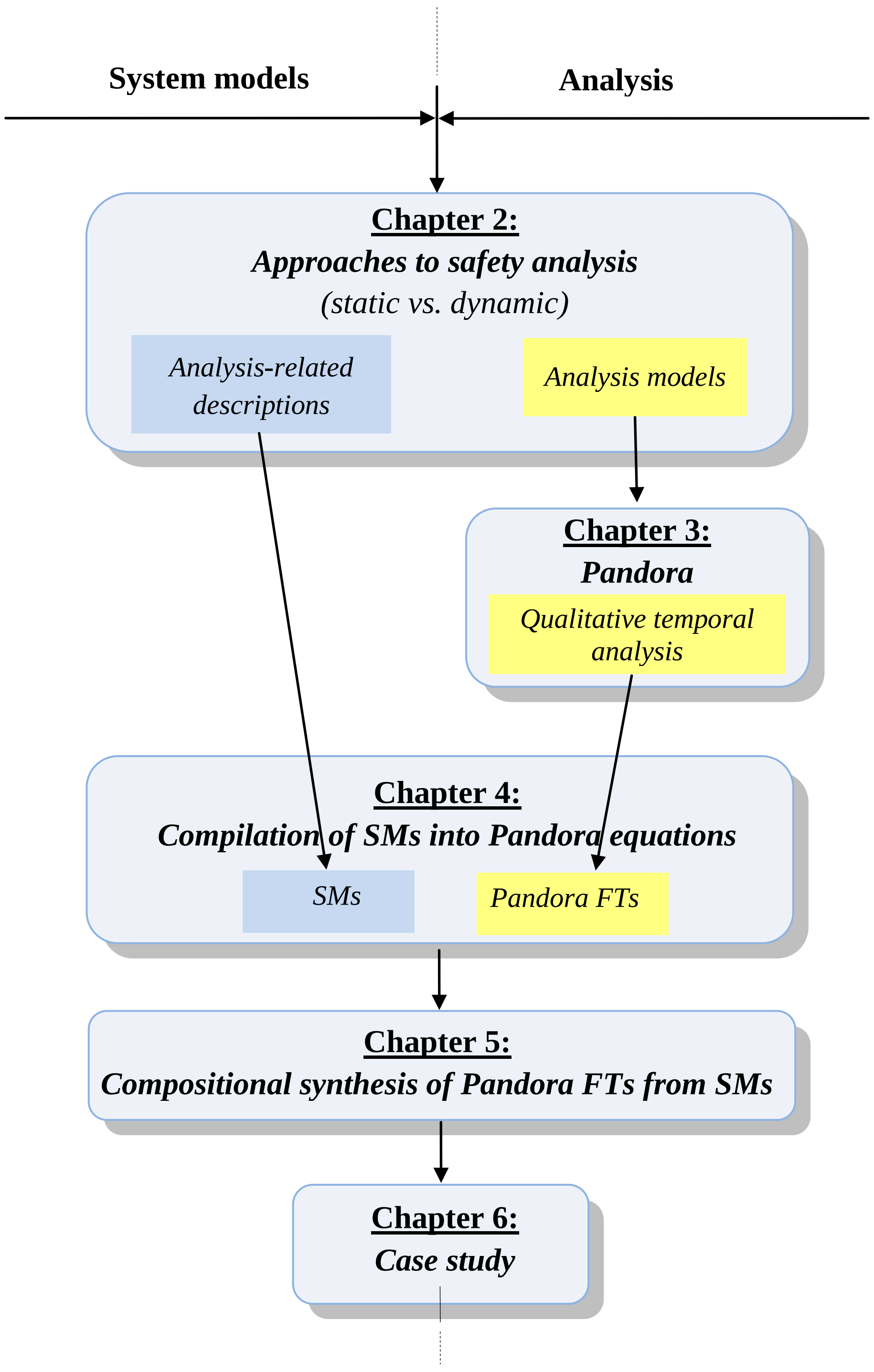
The second objective is to improve the scalability of the approach by enabling the compositional synthesis of large system TFTs from the individual component SMs. The idea here is to apply the proposed algorithm on component SMs, and then having generated local TFTs minimised wherever possible, to proceed and synthesise system TFTs via a process of composition.

> *In summary, the second objective is an approach to compositional synthesis of system TFTs from the SMs of the components of the system.*

[29] The acronym HiP-HOPS means Hierarchically Performed Hazard Origin and Propagation Studies and, in (Papadopoulos, 2000), a state-transition formalism (modes and modes' transitions) is suggested to be used for the description of systems with dynamic behaviour.

1.5 Thesis Structure

The structure of the thesis is illustrated in Figure 1—7.



System models
Analysis
Chapter 2:
Approaches to safety analysis
(static vs. dynamic)
Analysis-related descriptions
Analysis models
Chapter 3:
Pandora
Qualitative temporal analysis
Chapter 4:
Compilation of SMs into Pandora equations
SMs
Pandora FTs
Chapter 5:
Compositional synthesis of Pandora FTs from SMs
Chapter 6:
Case study

Figure 1—7. Organization of the thesis

Chapter 2 consists of a literature review on contemporary safety analysis approaches with an emphasis on automated safety analysis techniques, like AltaRica and HiP-HOPS, as well as some dynamic analysis approaches like those based on SEFTs, DFTs and AADL models. In chapter 3, we continue the literature review with a presentation of the Pandora qualitative temporal analysis technique and in chapter 4, we present our approach for an automatic generation of Pandora TFTs from SM models of systems. Chapter 5 presents our method to improve the scalability of our approach through a compositional synthesis of system TFTs that is applicable to hierarchical state automata. In chapter 6, we apply the technique on the brake-by-wire case study and, finally, we present our conclusion in chapter 7 by discussing the contributions, improvements and future works.

1.6 Publications

The automatic generation of TFTs from SM models of systems, presented in chapter 4, was outlined in (Mahmud, Papadopoulos & Walker, 2010) and published in the proceedings of the 40th annual IEEE/IFIP international conference on Dependable Systems and Networks[30] (DSN 2010).

The algorithm which generates the TFTs has been formalised and presented together with an approach for the compositional synthesis of TFTs from SMs in (Mahmud, Walker, & Papadopoulos, 2011). The latter has been published in the proceedings of the sixth international conference on Availability, Reliability and Security (ARES 2011), and presented at the second workshop and tool session on DYnamic Aspects in DEpendability Models for Fault-Tolerant Systems (DYADEM-FTS).

A journal extension of the latter paper has been published into a special issue of the ACM Performance Evaluation Review (PER) on Modelling Dynamic Behaviours of Complex Distributed Systems (Mahmud, Walker, & Papadopoulos, 2012).

On another note, some safety analysis techniques which are detailed in the literature review are discussed in deliverables of the European project ATESST2 (Advancing Traffic Efficiency and Safety through Software Technology second phase). For example, a review of relevant safety analysis techniques can be found in (Walker, Mahmud, Papadopoulos *et al.*, 2010), and (Törngren, Walker, Papadopoulos, Mahmud *et al.*, 2008).

[30] This was in the framework of the Proactive Failure Avoidance, Recovery and Maintenance (PFARM) workshop.

2 Approaches to safety analysis

According to Laprie (1992) safety is seen as an attribute of dependability, and this is compatible with the concepts' definitions presented in the introduction. Laprie (1992) defines dependability to be: "Trustworthiness of a system such that reliance can justifiably be placed on the service it delivers". The service being the behaviour of the system as it is perceived by its interacting human(s) or other system(s), and a system failure occurs when the delivered service deviates from fulfilling what the system is intended for (i.e. the system function). Safety is, therefore, concerned with the non-occurrence of catastrophic failure consequences on the environment. Examination of systems to understand how they can fail with the determination of frequencies of possible failures lead to *safety analysis*. Such assessments typically provide, in return, a useful resource in the design of failure mitigation mechanisms.

The range of existing safety analysis techniques in research literature and in use in industrial practice is enormous. In this chapter, we explore safety analysis through a comprehensive review of the relevant literature. We study different approaches to safety analysis and identify the corresponding pros and cons. The results from this study provide: (a) a more refined view of the problem to address (and its derivatives) and (b) a classification of the relevant safety analysis techniques including the work presented in this thesis.

This chapter begins with a presentation of relevant *classical* safety analysis techniques, and by using the term 'classical' we mean a number of the foremost analysis techniques, though, still well-established in the safety domain. These include Functional Hazard Analysis (FHA) (ARP4761, 1996), Preliminary System Safety Assessment (PSSA) which is also described in (ARP4761, 1996), Fault Tree Analysis (FTA) (Vesely W. E., 1981) and Failure Modes and Effects Analysis (FMEA) [(ANSI/IEEE-Std.352, 1987) and (Rausand & Høyland, 2004)].

The remaining part of the chapter presents a classification (of some relevant analysis techniques) into two main categories: static and dynamic approaches to safety analysis. The former includes, e.g., Failure Propagation & Transformation Notation (FPTN) (Fenelon & McDermid, 1993), Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) (Papadopoulos Y. , 2000) and Component Fault Trees (CFTs) (Kaiser *et al.*, 2003). The section of dynamic approaches includes fault injection and simulation techniques such as Altarica [(Griffault *et al.*, 1999) and (Bieber *et al.*, 2002)], the Deductive Cause Consequence Analysis technique (DCCA) [(Güdemann *et* *al.*, 2007) and (Ortmeier *et al.*, 2005)], Dynamic Fault Trees (DFTs) (Dugan *et al.*, 2000), Markov analysis (Trivedi, 2001), State Event Fault Trees (SEFTs) (Grunske *et al.*, 2005) and stochastic analysis of models described using the Architecture Analysis and Design Language (AADL) (Rugina A. , 2007).

2.1 Classical safety analysis techniques

Safety analysis typically starts early in the development cycle of a system. This usually means, given a set of desired functions, safety analysis starts taking place when a decision has been made to develop a system that is able to fulfil these functions. To illustrate the integration of safety with the design, Figure 2—1 (adapted from the ARP 4761[31] related to the aerospace industry) represents typical design activities associated with the corresponding safety tasks.

We observe in the Figure 2—1 that a safety technique called Functional Hazard Assessment (FHA) takes place first. Starting from the aircraft level, FHA attempts to identify failure conditions that are associated with the aircraft functions. The technique proceeds during the activities '*Concept Development*' and the '*Allocation of Aircraft Functions to Systems*' of an aircraft design lifecycle. These are preliminary activities preceding more refined design descriptions and that take place when there is primarily a functional focus[32] — i.e., no physical realization has yet been decided.

Before we discuss the relevant safety techniques in more detail, we wish to note that design and safety analysis practitioners in different industries (like aerospace, automotive and railway) may use closely related (and sometimes different terminologies for the same) safety techniques. For example, sometimes in the literature FHA can be substituted with an improved version known as Functional Failure Analysis (FFA) [(Papadopoulos *et al.*, 1999), (Papadopoulos Y. , 2000) and (Johannessen *et al.*, 2001)]. Similarly, FFA is used to identify safety hazards in a conceptual design of a system. It extends FHA to include, e.g., failure classes such as (but not limited to) omission/commission of service provision or early/late delivery of output.

[31] The Society of Automotive Engineers (SAE) standard for the Aerospace Recommendation Practice (ARP 4761).

[32] Emphasis is rather put on the functions of a system including the identification of safety requirements that need to be addressed.

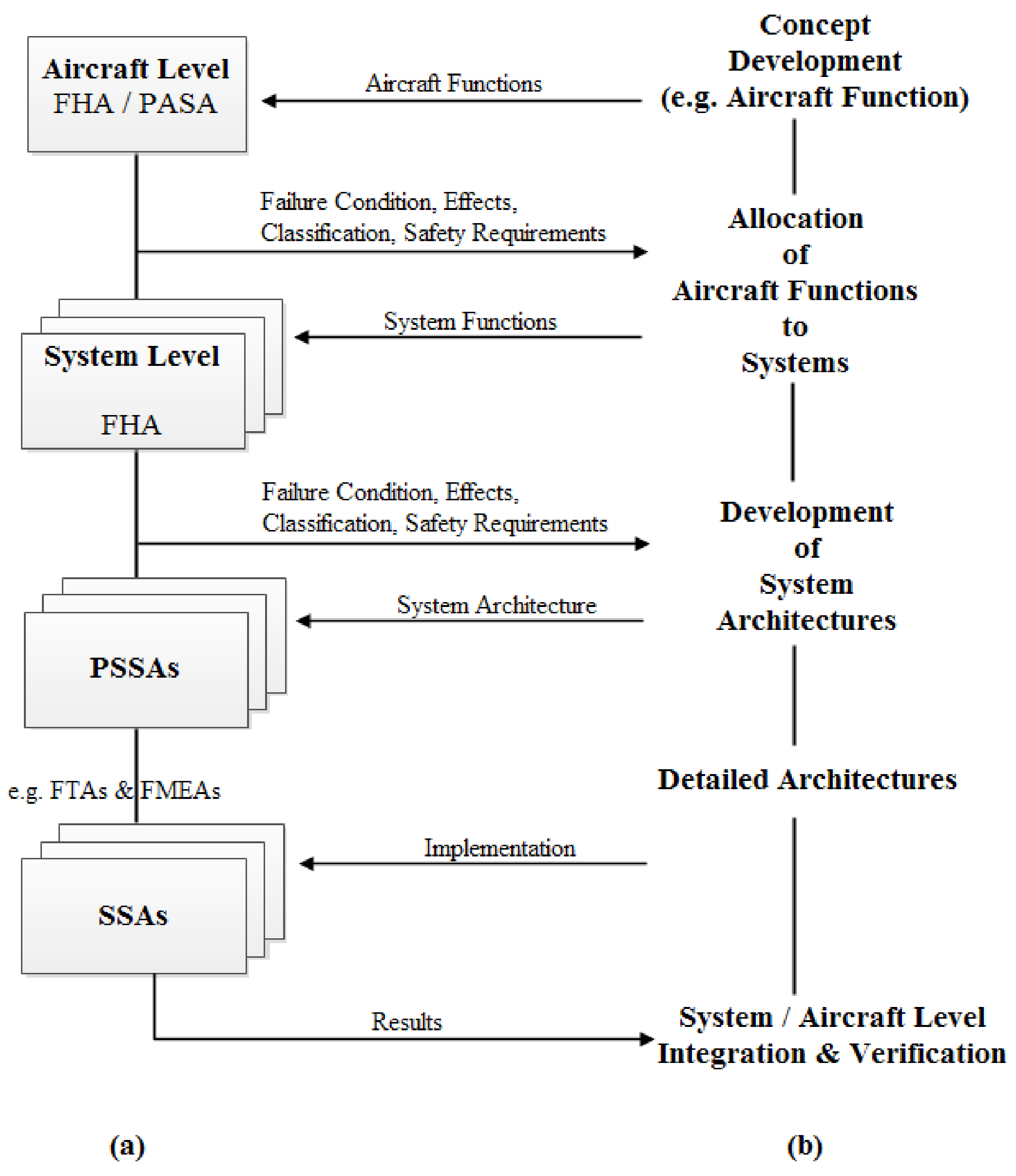

Aircraft Level
FHA / PASA
Aircraft Functions
Concept Development (e.g. Aircraft Function)
Failure Condition, Effects, Classification, Safety Requirements
Allocation of Aircraft Functions to Systems
System Functions
System Level
FHA
Failure Condition, Effects, Classification, Safety Requirements
Development of System Architectures
System Architecture
PSSAs
e.g. FTAs & FMEAs
Detailed Architectures
Implementation
SSAs
Results
System / Aircraft Level Integration & Verification
(a)
(b)

Figure 2—1. (a) Fragment of a Safety Assessment Process alongside (b) a Typical System Development Cycle (Fragments adapted from ARP 4761)

Moreover, Pumfrey (1999) has noted that in the automotive industry FFA can be referred to as FMEA (i.e., establishment of direct relationships from causes to effects of failure). However, it is often omitted in the corresponding literature to mention that a specific kind of FMEA (a predictive[33] FMEA) that is being used to indicate the same technique as

[33] It starts from hypothetical failure modes to assess the corresponding effects. It takes place in the early stages of the design lifecycle.

FFA. The other kind of FMEA starts from known[34] rather hypothetical failure modes, and thus the latter is different from the FFA technique.

Let's now present the big picture of an integration of some relevant safety tasks within a system design lifecycle. Figure 2—1, for instance, depicts the process for an aircraft typical development cycle. But apart from the aircraft specificities, the figure represents a typical association (and irrespective of the type of industry[35]) of a lifecycle activities with the relevant safety tasks. The shown top level displays the design activities and the safety tasks undertaken when the concerns are predominantly functional — during concept development in the figure. The design engineer builds a formal model of the aircraft functions and operations architecture. The safety engineer, on the other hand, extends the architecture with descriptions of failure e.g. a failure propagation model (FPM[36]), i.e., functional failure modes. Nonetheless, the common underlying basis is the functional and safety requirements identified in the aircraft level FHA/PASA. The latter (i.e., PASA), which is used in the aerospace industry, is a preliminary aircraft safety assessment technique. It is used to evaluate a proposed architecture based on the FHA (and severity classification[37] of failure conditions) and derive safety requirements.

Table 2—1 illustrates an example of an aircraft FHA. The function to, e.g., decelerate an aircraft on the ground can fail while the aircraft is either in taxing or in landing mode. The corresponding FHA table shows that a loss of the deceleration capability (even though alerted) has an effect that is classified as more severe[38] if the loss occurs while the aircraft is landing rather than taxiing (even though the failure is non alerted in the latter phase).

[34] i.e., an FMEA that takes place in the latter stages of the design lifecycle.

[35] This is shown in the V lifecycle model in (Pumfrey, 1999) that represents similar activities and safety tasks without any industry-specific connotations.

[36] The use of FPM appears in the insertion of the model-based approach (of which models are primary artefacts) in the industries and in safety standards of the *More Integrated Systems Safety Assessment* (MISSA) European project (://www.missa-fp7.eu/). FPMs can be used as basis for the assessment of the functions (e.g. in MISSA aircraft functions) architectures against the safety requirements.

[37] E.g. catastrophic, critical, marginal and insignificant failures (IEC-61508, 1997).

[38] We wish to note, though, that in the (ARP4761, 1996) hazardous is synonymous to severe-major.

Table 2—1. Example of an aircraft FHA — fragment adapted from ARP4761

Function	Phase	Failure Condition	Failure Effect Condition on Aircraft / Crew	Classification
Decelerate Aircraft on the ground	Taxi	Non Alerted Loss of Deceleration Capability	Crew is unable to stop the aircraft resulting in low speed contact with nearby vehicles, terminal or aircrafts. Crew reaction time can result in potential overrun.	Major
	Landing	Alerted Loss of Deceleration Capability	Crew notifies emergency ground support of a more suitable airport for an immediate preparation for emergency landing. Crew prepares passengers for landing overrun.	Hazardous

After the allocation of functions to systems (during design), another FHA assessment, at system-level though, takes place in order to re-examine the system functions. A system-level FHA is particularly useful to assess the architecture of systems that integrate multiple functions — the FHA re-takes place to take into account condition of failure, not only of a single function, but also of combinations of functions. The outcome helps to start preliminary system safety assessments (i.e., PSSAs) whose objective is to establish system safety requirements and architecture evaluations. PSSA is a technique that determines how failures can cause the functional hazards (identified by the FHA) and thus, when applied to proposed system architectures, it can help to determine whether or not these (i.e., the architectures) can reasonably be considered as acceptably safe. The capacity of PSSA in determining scenarios that lead to the functional hazards makes the technique similar to FTA — i.e., the hazard causes are structured in a fault tree for assessment. Moreover, the basic events of the PSSA fault tree can be used by FMEA analysts for the purpose of, e.g., assuring that (at least those more influential) cause effects have been addressed in the analysis. Although PSSA also defines safety requirements (as in FHA), it is not intended to verify that the more refined (or implemented) architectures meet those safety requirements. Indeed, verification of the safety of a system as a whole is comprehensively complemented by a system safety assessment (SSA) technique.

A safety integrated development cycle like the one represented in Figure 2—1 — or like the '*system and safety lifecycle*' described in (Pumfrey, 1999) — consists of two major processes that are typically performed in an iterative manner: a system design process and a safety analysis process. Yet, the functions that must be accomplished by the system (and their sequential relationships) need to be determined and described at the early stages of the lifecycle. Functional models such as Functional Block Diagrams[39] (FBDs) are often used to represent the functional interrelationships in a system. These diagrams consist of blocks of functions connected by directed lines to represent the function flow and flow direction.

To illustrate the concept we use an example of a vehicle Cruise Control (CC). Figure 2—2 — constituted from (Abele *et al.*, 2010) — represents the corresponding FBD, some subsystems (or components) desired functions being textually described. The input to the CC can be signals for the on, off, set/resume or cancel CC switch modes which require interpretation. Thereafter, the vehicle speed controller has the basic functionality to evaluate[40] the vehicle speed, and thus determines the CC state diagram — a component which decides transitions between states (e.g. on / off) and calculates the speed to be maintained by the CC (set point).

[39] These are also referred to in some literature as Functional Flow Block Diagrams, e.g. in (NASA, 2007).

[40] This is typically performed based on additional inputs from, e.g., the vehicle speed sensor and the wheel speed sensors which are not represented in the figure since they are not within the CC boundary.

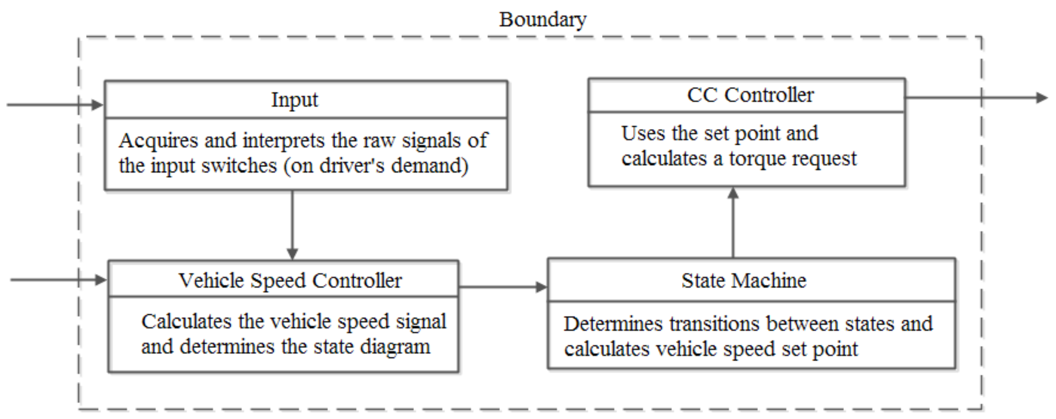

Boundary
Input
Acquires and interprets the raw signals of the input switches (on driver's demand)
CC Controller
Uses the set point and calculates a torque request
Vehicle Speed Controller
Calculates the vehicle speed signal and determines the state diagram
State Machine
Determines transitions between states and calculates vehicle speed set point

Figure 2—2. Functional block diagram of a Cruise Control

The set point is used to calculate a torque request by the controller, and which gets handled afterwards by the Engine Management System (EMS[41]). We wish to note that in situations where more information is available at the early stages, functions can be decomposed hierarchically into a network of sub-functions. We wish to note, though, that standards like the International Electro-technical Commission (IEC-60812, 2006) and the military (MIL-STD-1629A, 1980) recommend the use of FBDs as a basis for both FMEA and FMECA[42] (a FMEA extension for criticality analysis).

FMEA (and FMECA) inductively derive the corresponding effects from hypothetical (or known) causes, whereas FTA[43] works backwards to deduce the causes of some undesired outcome. In Figure 2—1 for example, the SSA uses the PSSA FTA to verify that all significant and previously identified effects can be dealt with as events in the fault tree. At this stage though, these are typically undeveloped events (e.g. because there is no sufficiently available information), but do not need to be further elaborated on for verification of compliance with applicable safety requirements. We wish to emphasize, though, that effects/causes relationships in both techniques, i.e., FTA and FMEA (as well as FMECA) link effects logically to their corresponding networks of causes, and typically with neither concept of time nor information about the temporal sequencing in which the causes occur. Despite this, FTA and FMEA are (until now)

[41] The EMS typically consists of a speed transmission and controls for the torque and idle speed. It is beyond the CC boundary, and thus not represented in Figure 2—2.

[42] Failure Modes, Effects and Criticality Analysis.

[43] It is conducted in the early stages of the lifecycle for, e.g., the PSSA and usually used as a basis (among other techniques) for the SSA (e.g., Figure 2—1).

among the prevalent techniques used in research and in several safety-critical industries — e.g., aerospace, automotive and nuclear.

2.1.1 Fault tree analysis

Fault Tree Analysis (FTA) is a deductive top down technique that allows analysts to perform both quantitative (probabilistic) analysis as well as qualitative (logical) analysis. This is particularly useful when there is no sufficient probabilistic failure information about the constituent parts — e.g. subsystems or components — of the system under analysis (e.g., at the early stages of the conception). FTA starts from a bad (or an undesired) situation called the top event or the root of the fault tree, and then it attempts to deduce the different causes (or possibly combinations of causes) that lead the system to the top event. The root causes can be, e.g., component failures called basic events or combinations of these. Figure 2—3 illustrates the different graphical representations of the events that can appear in a fault tree.

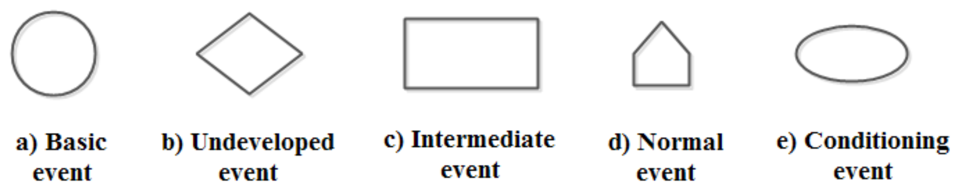


Figure 2—3. Fault tree symbols describing events

A basic event (a) is typically an initiating fault. It is represented as a leaf in the fault tree, which means that it cannot be further expanded, i.e., no other events can lead to it. However, (b) represents an event that for one reason or another has not been further developed, because either this does not impact the analysis or there is no sufficient information. But in the case where an event arises from the combination of other events that are (or ultimately expand to) basic events, then it is labelled as an intermediate event (c). A normal event (d) is not expected to cause any problem on its own and is part of the normal operation of the system. It is typically an external event that may lead to an undesired situation only in combination with other events. The remaining event (e) does not necessarily have to be a fault, but is intended to constrain certain types of logic gates. For example, an INHIBIT gate is true only if its input event is true and its attached conditioning event is also true, the latter can be applied to Priority-AND gates as well. Figure 2—4 illustrates a list of symbols that correspond to the fault tree gates, even though the list is not exhaustive[44].

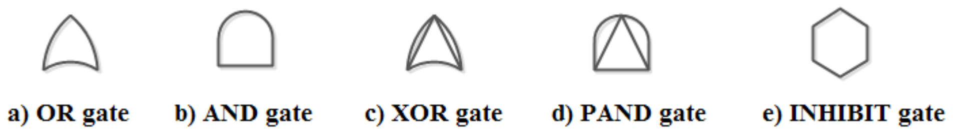


Figure 2—4. Fault tree symbols describing gates

An OR gate (a) is true if any of its inputs is true, whereas an AND gate (b) is true only if all of its inputs are also true. In contrast, an exclusive OR (XOR gate as described in c) is true if exactly one of its inputs is true, while a Priority-AND (PAND gate as in d) is true if all of its inputs occur (i.e., evaluate to true), but according to the sequence specified in the conditioning event that is attached to the gate. Finally, the INHIBIT gate (as in e) described previously acts like an AND gate with two input events: the single input event of the INHIBIT gate and its attached enabling conditioning event. We wish to note, though, that a fault tree is fundamentally Boolean logic, i.e., gates with temporal or event-sequencing information are part of subsequent versions of the fault tree, e.g., the DFT approach in (Vesely *et al.*, 2002).

Fault trees can be used in conjunction with many design activities of the lifecycle of a system; they may also take place in the early stages like in the PSSA activity, though for a predictive role at such stage. For example, qualitative analysis can be performed very early in the lifecycle to speculate as to how a bad situation can happen, i.e., for instance through the minimisation of possible combinations of the deduced failure events that cause the supposed top event of the fault tree. However, quantitative analysis can be used when sufficient[45] probabilistic data becomes available, e.g., by combining figures for actual component failure rates to calculate overall probabilities of system-level faults (i.e., failure rates for top-events of fault trees). We wish to note, though, that quantitative FTA can also be performed in a predictive role (Pumfrey, 1999), e.g., starting with an

[44] For a more detailed list of gate symbols, we refer the reader to (Villemeur, 1992).

[45] i.e., with respect to a certain acceptable level of detail or accuracy.

acceptable failure rate assigned to the top event, and then deciding how to allocate a failure rate solution between the events down the tree.

We use an example that represents an anti-lock braking system[46] (ABS) with some connected components (as in Figure 2—5) to show a system-level fault tree — of an omission of ABS brake reference data (i.e., an output deviation that negatively impacts the brake actuator connected upstream). An ABS is intended to prevent the wheels from locking while braking, which is very useful when a vehicle needs to be urgently stopped on slippery surfaces in particular. Thus, it is not exaggerated to say that a driver, average or professional irrespectively, can be relieved from nerve-wracking situations (with possible catastrophic consequences) if the vehicle is equipped with an ABS. Figure 2—5 shows a simplified ABS version with three inputs — a brake reference, a vehicle speed and a wheel speed data — for an intended delivery of a processed brake reference data to the connected brake actuator. The system is represented only once in the figure, but overall it is actually duplicated on the vehicle wheels.

[46] The example has been made on the basis of some outcome from the projects ATESST2 (http://www.atesst.org) and MAENAD "*Model-based Analysis & Engineering of Novel Architectures for Dependable Electric Vehicles*" (http://www.maenad.eu). We wish to note, though, that it has been simplified for the purposes of illustrating a fault tree and an FMEA (Table 2—2, page 53).

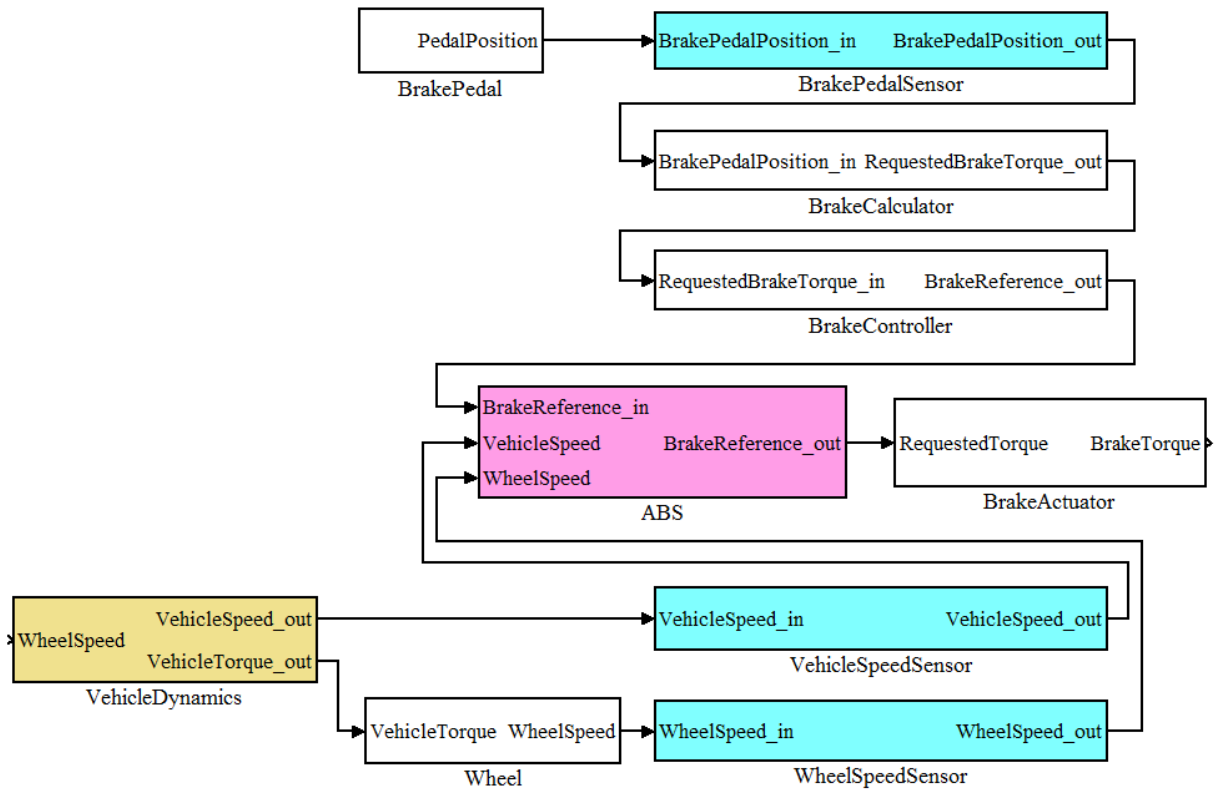


Figure 2—5. Some components influencing the anti-lock braking subsystem

The brake reference, as shown in the figure, is some data that is originally produced by the brake pedal (i.e., a pedal position initially), then propagated and processed by some components in series that are: the brake pedal sensor to capture the pedal position (upon detection of a pedal displacement). It then propagates the pedal position to the next component in the series, i.e., the brake calculator which computes the driver requested torque using the transmitted pedal position. The brake calculator sends thereafter the produced value to the brake controller which decides the required torque on each wheel. The latter is one of the required parameters needed by the ABS function to appropriately decide the braking force (then delegated to the brake actuator) on the corresponding wheel. The other parameters required by the ABS function to fulfil its duty are given by: the sensed vehicle velocity input (captured by the vehicle speed sensor) and the sensed wheel angular velocity input which is detected by the wheel speed sensor. These values originate from a component which is represented as an abstraction of the dynamics of the vehicle — i.e., the effects that the velocities of the wheels can have on the vehicle. The component outcome represents a torque on each wheel, which allows the corresponding speed (of the wheel) to be sensed, and a detectable overall speed for the vehicle.

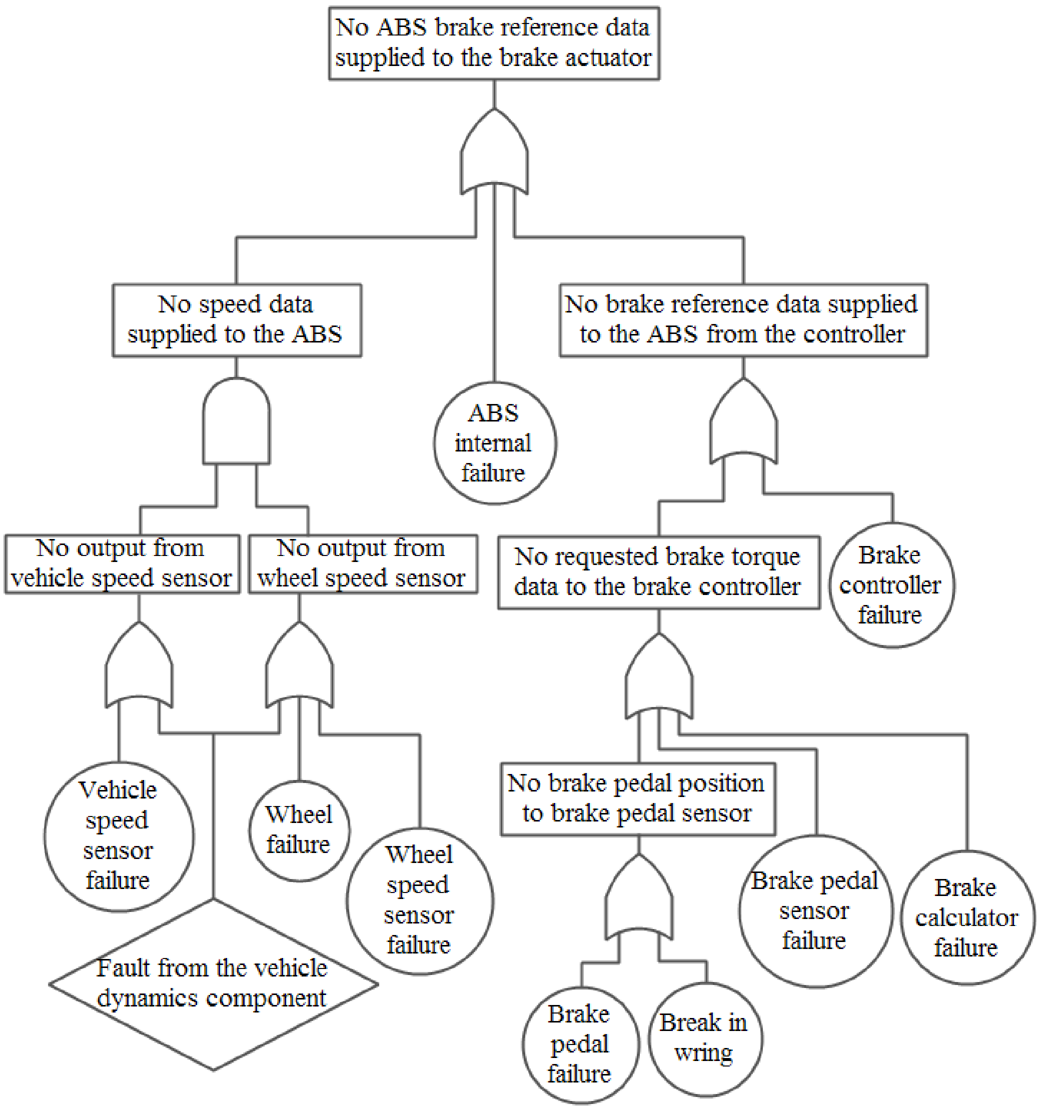

No ABS brake reference data supplied to the brake actuator
No speed data supplied to the ABS
ABS internal failure
No brake reference data supplied to the ABS from the controller
No output from vehicle speed sensor
No output from wheel speed sensor
No requested brake torque data to the brake controller
Brake controller failure
Vehicle speed sensor failure
Wheel failure
Wheel speed sensor failure
Fault from the vehicle dynamics component
No brake pedal position to brake pedal sensor
Brake pedal sensor failure
Brake calculator failure
Brake pedal failure
Break in wring

Figure 2—6. Fault Tree of the "No ABS brake reference data supplied to the brake actuator"

The hierarchal structure of the system to analyse is captured neatly in the fault tree. For example, the fault tree in Figure 2—6, which represents the network of causes leading to the top event "*No ABS brake reference data supplied to the brake actuator*", reflects the structure of the ABS and the connected components in Figure 2—5. It shows how basic and intermediate events (as well as one undeveloped event) connect to the top event; the intermediate events appear as inner nodes linked to more developed events downwards in the fault tree. For instance, a defective brake pedal because of, e.g., a break in its wiring, or an initiating fault in the pedal will cause the brake pedal sensor to fail in its attempt to detect a pedal displacement (omission of input to the brake pedal sensor) — i.e., the fault tree intermediate event "*No brake pedal position to brake pedal sensor*". The latter is sufficient on its own (OR gate) to cause the higher intermediate event "*No requested brake torque data to the brake controller*", and which propagates through the brake controller to the ABS (i.e., the intermediate event "*No brake reference data supplied to the ABS from the controller*") then to the actuator (the top event of the fault tree).

FTA aims at multiple usages (some of which have been mentioned previously). But we put now more emphasis on qualitative FTA, which does not rely on the presence of failure measures (necessary for quantitative FTA). Qualitative analysis works by minimising, i.e., reducing the fault tree such that it displays only the failure events that could, individually or collectively, cause the top event to occur. The outcome is a minimal cut-set form of the fault tree, i.e., a representation limited to the significant failure events — e.g., the fault tree example of Figure 2—7.

In the example, the Boolean logic expression which corresponds to the fault tree before analysis — i.e., left hand side of the Figure 2—7 — is the disjunction of the expressions of the separate branches underneath the top event, i.e., the cut-sets which correspond to the intermediate events E1 and E2. The symbols '**.**' and '**+**' represent the gates 'AND' and 'OR', respectively.

$$E0 = E1 + E2$$

Since E1 and E2 are equal to A + E3 and to B.E4 respectively, then we have:

$$E0 = (A + E3) + (B.E4)$$

Moreover, E3 (resp. E4) is equal to B.C (resp. E5+C). Therefore, we get:

$$E0 = (A + (B.C)) + (B . (E5+C))$$

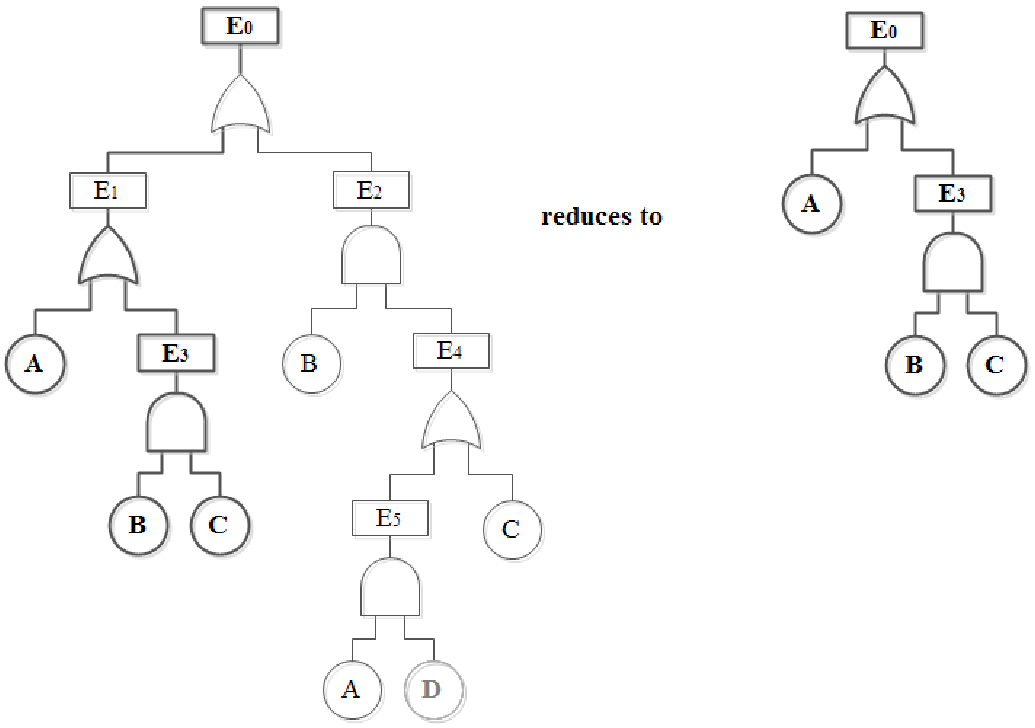


Figure 2—7. An example of minimal cut-sets

Boolean associative and distributive laws[47] in addition to the precedence order of the logic operators (i.e., the fault tree gates) make it possible to remove and re-arrange parentheses in an expression. Therefore, E0 can be re-written as:

E0 = A + B.C + B.E5 + B.C

Since E5 is the conjunction of the basic events A and D, then E0 becomes:

E0 = A + B.C + B.A.D + B.C

We eliminate the redundant B.C using the OR idempotent[48] law, and hence:

E0 = A + B.C + B.A.D

[47] E.g., (X+Y)+Z ⇔ X+(Y+Z) ⇔ X+Y+Z (OR associative law), (X.Y).Z ⇔ X.(Y.Z) ⇔ X.Y.Z (AND associative law), X.(Y+Z) ⇔ X.Y+X.Z (AND distributive law over OR) and X+(Y.Z) ⇔ X+Y.Z (priorities for Boolean evaluation).

[48] X + X ⇔ X (OR idempotent law) and X.X ⇔ X (AND idempotent law).

The conjunction of A, B and D is removed by using an absorption[49] law, and thus the minimal cut-set form of the fault tree (right hand side of Figure 2—7).

$$E0 = A + B.C$$

We must point out that qualitative FTA can alleviate some problems that are encountered in the quantitative[50] aspects during the assessments of complex systems. For example, the fault tree minimal cut-set forms help in allocating probabilistic measures to the relevant failure events, facilitating thereby the consequent quantitative analysis. Thereafter, the quantitative results from FTA (and FMEA as well) feed back into earlier assessments[51] such as system level FTAs and (for the case of an aircraft development) aircraft level FTAs, like in Figure 2—1.

2.1.2 Failure Mode and Effects Analysis

Failure mode and effect analysis (FMEA) is commonly known (and often used) as a bottom-up analysis technique. In that sense, it proceeds by analysing the system components individually, or sometimes collectively, to determine failure effects thereof on the system[52]. The history of FMEA (and its criticality version known as FMECA[53]) goes back to the 1950s (Rausand *et al.*, 2004). At the time, the technique was introduced in response to the growing concerns about the reliabilty of military systems. The standard guidelines in (MIL-STD-1629A, 1980) were thereafter developed and revised during the 1970s.

[49] $X + X.Y \Leftrightarrow X$

[50] In both directions, i.e., probability of occurrence of the top event as well as allocation of probability budgets to lower-level events (i.e., the predictive role of quantitative FTA mentioned earlier).

[51] For the purposes of, e.g., helping in the demonstration of compliance with the quantitative requirements derived from FHAs. This can facilitate the assessments and reviews of the certification authorities (ARP4761, 1996).

[52] The technique aims at addressing the consequences that result from, typically, single failures.

[53] It adds to the FMEA a formal classification of failure effects according to severity and probability of each contributing factor to system level hazards. This helps in implementing corrective measures to mitigate the high risk effects. For more information about FMECA, we refer the reader to (MIL-STD-1629A, 1980).

The technique commonly uses a tabular description to link failure modes[54] (of the system constituent parts) to their induced effects and magnitude identified accordingly. We wish to note, though, that an FMEA table may contain a number of optional columns that correspond to, e.g., means of failure detection, severity of the failure effects (with mitigation strategies if possible) as well as comments and recommendations (Pumfrey, 1999). FMEA can be useful in many ways such as (but not limited to) serving as a basis for probabilistic reliability and availability analysis, providing future references for consideration of design changes (to avoid or minimise the effects of the most critical failures identified), and helping to show how some design alternatives can (or cannot) represent optimal trade-offs[55] — i.e., designs that meet dependability criteria (such as safety, reliability and availability) at a minimum cost.

There are different ways to conduct an FMEA (depending on the application), but the following are generally considered as major steps.

- Definition of the system and its components.
- Identification of the operating states of the components.
- Identification of the component failure modes and the possible effects of each component failure.
- Investigation of other factors e.g. detection and protection.
- Making conclusions and recommendations.

As an example, Table 2—2 represents a part of an FMEA which corresponds to the ABS (depicted in Figure 2—5). For the component “vehicle speed sensor”, two failure modes are considered: no signal and false signal. For example, an omission of output from the vehicle speed sensor (i.e., “No signal”) makes the ABS ineffective, since it becomes impossible to detect the lock up tendencies of the wheels — the ABS interprets that a wheel is being likely to lock up by determining its brake slip according to the signal received from the vehicle speed sensor. As a consequence on the vehicle, the braking

[54] i.e., forms of deviation from correct service (Avizienis *et al.*, 2004).

[55] In (Papadopoulos *et al.*, 2011), the safety analysis tool framework (HiP-HOPS) integrates an automated optimisation of design models using genetic algorithms to evolve designs into new and optimal ones.

system switches to regular (i.e., no ABS function). As for the last column of the table, it shows some comments related to such a situation, like directing the driver to prevent wheel lock up by manually pumping the brake pedal. The comments can also include some warnings to prepare the driver, e.g., a possible yawing of the vehicle depending on the condition of the road, or during a panic stop.

Table 2—2. Fragment of FMEA for the ABS

Component	**Failure Mode**	**Subsystem Effects**	**Vehicle Effects**	**Comments**
Vehicle speed sensor	No signal	Impossible to determine the brake slip for each wheel, and thus the corresponding lock up tendencies cannot be determined.	Vehicle braking in the regular manner (i.e. ABS inactive).	1) The driver has to manually pump the brakes to prevent wheel lock up. 2) The vehicle may spin on wet and slippery roads (or during a panic stop)
	False signal	Incorrect determination of the vehicle speed leading to incorrect calculations of the different wheel brake slips.	1) Possible lock up of wheels during braking (the corresponding lock up tendencies were not detected by the ABS). 2) Possible unnecessary changes to the fluid pressure at each wheel (performed automatically by the ABS).	Effect 1) same as above. Effect 2) possibility of non optimised vehicle braking (depending on the vehicle speed estimation and the brake pedal position).

Despite the purely combinatorial aspect of FTA and FMEA (which makes them classified as static approaches too), there is a contemporary and important research focus — both industrial and academic — on using these techniques in conjunction with influential, state-of-the-art modelling languages like AADL and Altarica. In the circle of the aerospace industry for instance, there is a recent work about generating FMEA and FTs from AADL models. This work is described in (Hecht *et al.*, 2011) and (Hecht *et al.*, 2010) concerning an automated generation of FMEA and in (Joshi *et al.*, 2007) concerning a generation of fault trees. AADL is clearly gaining growing acceptance in the aerospace community and Altarica's modelling and analysis platform — a tool which uses computer-performed generation of fault trees — was qualified as a validation tool in several aerospace projects, including Airbus civil aircraft programs (Bernard R. , 2009). Besides, the company Dassault Aviation[56] has demonstrated the consistency of its Falcon 7X flight controls according to the requirements of the certification authorities with the aid of Altarica models. Dassault Aviation has developed an Altarica tool for the Conception and Analysis of Systems (so-called OCAS) with a generator of fault trees from Altarica[57] models proposed in (Rauzy, 2002). The generated FTs can then be analysed by the Aralia tool (Dutuit & Rauzy, 1997); the performed analysis is both qualitative — in order to extract all the minimal cut-sets — and quantitative to calculate probabilities of the top events.

2.2 Static approaches

We present in this part some relevant static approaches in addition to FTA and FMEA discussed earlier. Static analysis is sufficient for many systems, e.g. where the full failure behaviour of the system is accurately represented through static causal relationships between the individual components — the corresponding failures have their effects on the system individually or in logical combinations with other failures. We wish to note, though, that some static safety techniques have been extended to model and analyse dynamic systems. For example, Papadopoulos (2000) suggests an extension of HiP-HOPS with a modes/ modes' transitions formalism to be used for the description of systems with dynamic behaviour. Besides, there has been a considerable body of work focused on how to enable the tool to handle temporal fault trees [(Walker & Papadopoulos, 2008) and (Walker, 2009)].

[56] For more details about the different participants in Altarica — industry and academic/ research bodies — with their corresponding tools and variations of the language, we refer the reader to (Bernard R. , 2009).

[57] The described compilation into fault trees is based on a variation of Altarica (so-called Altarica data flow). The reason is to drop out features like bidirectional flows which make the compilation difficult.

The techniques discussed in this section are Failure Propagation & Transformation Notation (FPTN), Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) and Component Fault Trees (CFTs).

2.2.1 FPTN

Failure Propagation & Transformation Notation (FPTN) is a simple graphical technique designed to allow a compositional approach to safety analysis and to aid FTA and FMECA in representing the failure behaviour of systems (Fenelon & McDermid, 1993). The technique consists of a hierarchical notation for the description of faults propagation through a compositional, modular architecture of a system. Besides, failures in FPTN are typed according to particular failure domains — e.g., failures of types timing, value or commission / omission (Fenelon & McDermid, 1992). Failures that are lying in the timing domain are those which occur at an incorrect time, whereas value failures can occur, e.g., when an incorrect value is returned by some computation. As for the type commission (resp. omission), it groups failures that are due to a function which is provided when not required (resp. a function which is not provided when required). In FPTN, these types constitute the basis of a simple classification of failures; however, new failure mode types can be added by users of the notation according to their application domains. Besides, transformation of failures from one domain into another (which is not uncommon in many systems) is supported and well-expressed in FPTN — typically, this can happen as failures propagate through a system. For example, a watchdog timer in a real-time system can force a computation exceeding its budgeted execution time to return an approximate value; thus, transforming a failure domain from time to value.

In FPTN, a component or subsystem is typically drawn as a box with a set of inputs and outputs, by means of which architectures with interconnected modules can be graphically described. Besides, a module can also contain other modules, thereby allowing the FPTN descriptions to be hierarchical (Grunske & Neumann, 2002).

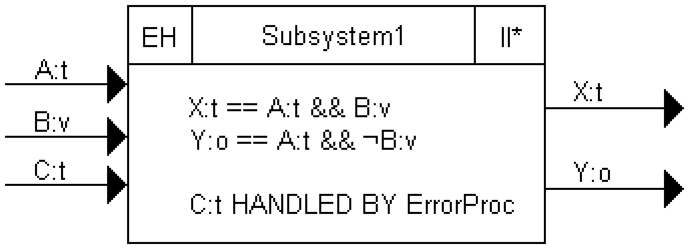


Figure 2—8. An example FPTN graphical description — (used in Walker, Mahmud, Papadopoulos *et al.*, 2010)

Figure 2—8, for example, shows a component named Subsystem1 with an error handler (EH) to the top-left and a criticality class (II*) to the top-right. The component I/O interface consists of a set of inputs (arrows on the left) and a set of outputs (arrows on the right). Each arrow is labelled by its name typed with its domain — e.g., input ‘A’ typed with ‘t’ (timing) and output ‘Y’ typed with ‘o’ (omission). ‘v’ is an abbreviation for value. The failure behaviour of a module is described inside the box in the form of logical equations. The canonical form of these equations is a “sum-of-products” form equivalent to the minimal cut-sets of the fault trees which correspond to the output failure modes. For instance, the fault X (which is lying in the domain ‘t’) is caused by a conjunction of failures A and B lying in the domains ‘t’ and ‘v’ respectively. Besides, the figure shows that the module’s EH handles the failure C:t, and thus this one has no effect on the rest of the system (i.e., the downstream modules). These capabilities enable the technique to be used both deductively and inductively: The former to deduce from an output failure its corresponding root causes (i.e., a fault tree) and the latter by tracing a component failure through the system to determine its effects (an FMECA). We wish to note, though, that the logical equations like those represented in Figure 2—8 are typical; FPTN also enables the description of recovery mechanisms and internal failure modes.

Despite a system architecture being reflected by its corresponding FPTN description (supposedly facilitating the identification of the causes of faults which propagate through the system components), the technique suffers from a lack of means for the analysis of fault propagation (Wallace, 2005). Another issue consists of the difficulty that may arise as a result of, e.g., changes taking place in components. For example, changes which may alter the types of propagated faults (or propagation directions) may, as a result, necessitate reconstruction of modules. This limits the reusability which was hoped through the compositionality of an FPTN failure model.

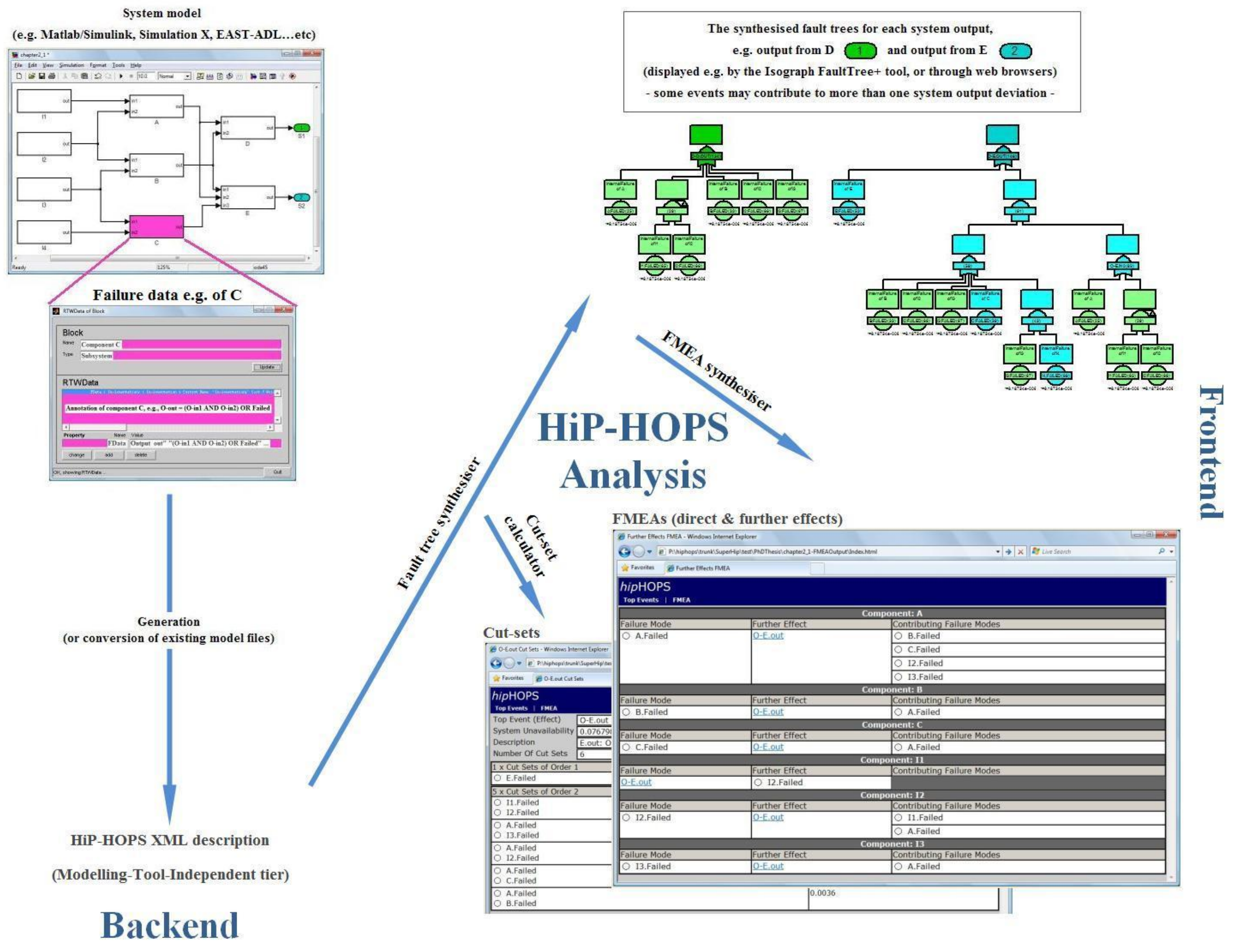
2.2.2 HiP-HOPS

HiP-HOPS (Hierarchically Performed Hazard Origin and Propagation Studies) is an automated safety analysis technique originating from a number of classical techniques such as FFA, FTA and FMEA. The technique starts taking place early in the design lifecycle with exploratory FFA (Papadopoulos, 2000); though, its primary use can be described only after a hierarchical model of the system has been developed — i.e., following the FFA. The failure behaviour of components is analysed using a modification of classical FMEA, called the Interface Focused FMEA (IF-FMEA). The application of this technique generates a model of the local failure behaviour of the component which is represented as a table. The table provides a list of component failure modes observed at the component outputs. For each component output failure, the causes are determined as a logical combination of internal malfunctions or deviations of the component inputs. An IF-FMEA table records component reactions to failures that are generated by other components. Moreover, the table determines the failure modes that the component itself generates and may propagate to other components. Upon determination of local failure behaviour of all components, HiP-HOPS can show how the functional failures (identified in the exploratory FFA) arise from combinations of the low-level component failure modes (identified in the IF-FMEAs). This is done by automatically synthesising fault trees; a fault tree is generated incrementally by parsing the expressions, which are derived from the IF-FMEA, and encountered during a hierarchical traversal of the system model.

Additionally, the HiP-HOPS tool automatically performs minimal cut-set analysis (i.e., the smallest combinations of component failures that are sufficiently needed to cause a system failure) and probabilistic calculations on the minimised fault trees to predict the reliability and availability of the system. Last but not least, the tool performs dependability vs. cost optimisations of system architectures, and has the potential to be extended and configured to any model that provides the topology of the system. In other words, a model specifying the components and the connections between components with failure annotations is suitable for the HiP-HOPS analysis. This has already been demonstrated with, e.g., Simulink system models (Papadopoulos *et al.*, 2004) and marine system designs developed in SimulationX (Papadopoulos *et al.*, 2003).

Figure 2—9 illustrates the HiP-HOPS analysis of a system model. The system components — described using, e.g., Matlab Simulink, SimulationX or EAST-ADL (leftmost hand side of the figure) — are first annotated through a graphical user interface (the GUI underneath the system model). Those annotations consist of failure modes and failure expressions which are required for the fault tree synthesis. After being read and interpreted by the model parser, the annotation formats get converted into another format readable by the fault tree synthesizer for the generation of fault trees. The resultant fault trees (to the top right) — these can also be displayed using the Fault Tree Plus tool[58] — are then analysed (centre of the figure) to be reduced into minimal cut-sets. Finally, the cut-sets are passed to an FMEA synthesizer to generate the FMEA tables (bottom right).

[58] A widely used FTA tool of Isograph.


System model
(e.g. Matlab/Simulink, Simulation X, EAST-ADL…etc)
Failure data e.g. of C
Generation
(or conversion of existing model files)
HiP-HOPS XML description
(Modelling-Tool-Independent tier)
Backend
Fault tree synthesiser
Cut-set calculator
FMEA synthesiser
HiP-HOPS
Analysis
The synthesised fault trees for each system output,
e.g. output from D 1 and output from E 2
(displayed e.g. by the Isograph FaultTree+ tool, or through web browsers)
- some events may contribute to more than one system output deviation -
Frontend
Cut-sets
FMEAs (direct & further effects)

RTWData of Block

Block

Name	Component C
Type	Subsystem

RTWData

Annotation of component C, e.g., O-out = (O-in1 AND O-in2) OR Failed

Property	Name	Value
	FData	Output out" "(O-in1 AND O-in2) OR Failed" ...

OK, showing RTWData ...

O-E.out Cut Sets - Windows Internet Explorer

hipHOPS
Top Events | FMEA

Top Event (Effect)	O-E.out
System Unavailability	0.076790
Description	E.out: O
Number Of Cut Sets	6

1 x Cut Sets of Order 1	
○ E.Failed	
5 x Cut Sets of Order 2	
○ I1.Failed ○ I2.Failed	
○ A.Failed ○ I3.Failed	
○ A.Failed ○ I2.Failed	
○ A.Failed ○ C.Failed	
○ A.Failed ○ B.Failed	0.0036

Further Effects FMEA - Windows Internet Explorer

P:\hiphops\trunk\SuperHip\test\PhDThesis\chapter2_1-FMEAOutput\Index.html

hipHOPS
Top Events | FMEA

Failure Mode	Further Effect	Contributing Failure Modes
Component: A		
○ A.Failed	O-E.out	○ B.Failed ○ C.Failed ○ I2.Failed ○ I3.Failed
Component: B		
○ B.Failed	O-E.out	○ A.Failed
Component: C		
○ C.Failed	O-E.out	○ A.Failed
Component: I1		
O-E.out	○ I2.Failed	
Component: I2		
○ I2.Failed	O-E.out	○ I1.Failed ○ A.Failed
Component: I3		
○ I3.Failed	O-E.out	○ A.Failed

Figure 2—9. HiP-HOPS architecture (the safety analysis tool perspective)

An example which explains how HiP-HOPS mechanises the construction of fault trees, determines the cut-sets and constructs the FMEAs from design models follows.

<u>Example of a HiP-HOPS use</u>

Figure 2—10 shows a Simulink system model which consists of two interconnected subsystems (subsystem1 and subsystem2). Subsystem1.y is the output port of Subsystem1. Similarly, Subsystem2.y is the output port of Subsystem2.

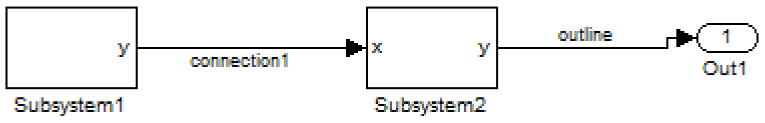


Figure 2—10. Abstracted system model for a HiP-HOPS analysis

Assuming that Subsystem1 has been annotated with the following failure data:

O-connection1 = Failed1

Where 'O-connection1' represents an omission of connection1 (output deviation) and 'Failed1' represents an internal failure (i.e., a basic event) of Subsystem1. Information about cost and weight of a component can also be added as well as probabilities of failures (e.g., a constant failure rate like, for instance, 6.18754E-06 for 'Failed1').

Subsystem2, however, is annotated as follows:

$$\text{O-y}_{\text{(of Subsystem2)}} = \text{O-connection1 OR Failed2}$$

where O-y represents an omission of output y, which is caused either by O-connection1 or an internal failure of Subsystem 2 'Failed2'. Now, we refine Subsystem2 as in Figure 2—11, which shows two subcomponents P and F arranged in parallel, fed by the input x of Subsystem2 and supplying output through subcomponent S then y (of subsystem2).

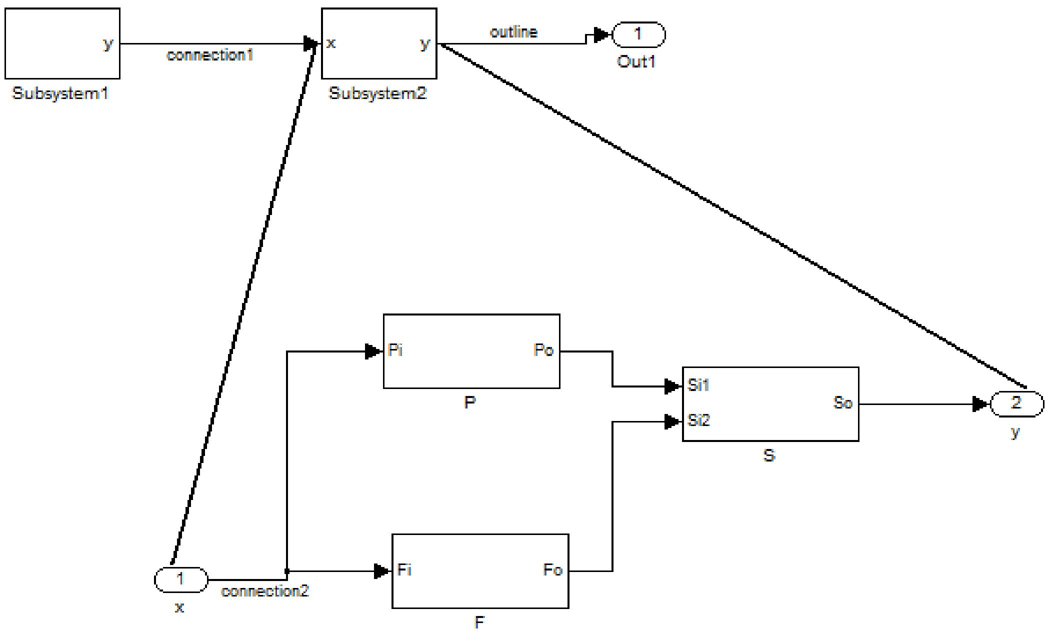


Figure 2—11. Refinement of Subsystem2

An output deviation of subcomponent P (omission in the example) is either caused by an omission on its input (i.e., connection2) or an internal failure of P (i.e., FailedP).

O-Po = O-connection2 OR FailedP

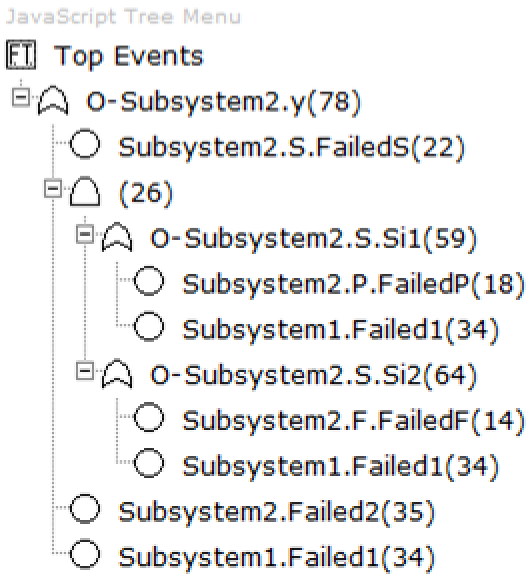
Similarly, an output omission from F is either caused by an input omission on connection2 or an internal failure of F (i.e., FailedF).

O-Fo = O-connection2 OR FailedF

In the example, supply to subcomponent S must be received from at either P or F for its proper functioning. Therefore, an output omission from S is caused by an internal failure (i.e., FailedS) or omission on both inputs (i.e., Si1 and Si2 together).

O-So = (O-Si1 AND O-Si2) OR FailedS

Thereafter, HiP-HOPS parses the system' model and synthesises the component fault trees into system fault trees — one per system failure, i.e. as in Figure 2—12.



```
JavaScript Tree Menu
Top Events
  O-Subsystem2.y(78)
    Subsystem2.S.FailedS(22)
    (26)
      O-Subsystem2.S.Si1(59)
        Subsystem2.P.FailedP(18)
        Subsystem1.Failed1(34)
      O-Subsystem2.S.Si2(64)
        Subsystem2.F.FailedF(14)
        Subsystem1.Failed1(34)
    Subsystem2.Failed2(35)
    Subsystem1.Failed1(34)
```

Figure 2—12. System fault tree synthesised by HiP-HOPS

HiP-HOPS displays the synthesised fault trees using web browsers like Internet Explorer, Mozilla Firefox or Netscape Navigator — plus the editor of Isograph FaultTree+ appendix A —1).

The synthesised fault tree shows a total of four cut-sets[59]. Three of those are of order[60] one (each contains one basic event) and the other cut-set is of order two. The failures Failed1, Failed2 and FailedS, which correspond to Subsystem1, Subsystem2 and its component S respectively, are each a single point of failure. The cut-set of order two, however, consists of both failures FailedP and FailedF — of components P and F[61] respectively — causing together (i.e., the conjunction of both) the top event to occur[62]. The different cut-sets are generated automatically by HiP-HOPS using the tool cut-set calculator; the calculation outcome is shown in appendix A — 2.

[59] A cut set is a branch of the fault tree leading to the top event — the occurrence of the events of the branch causes the top event to occur.

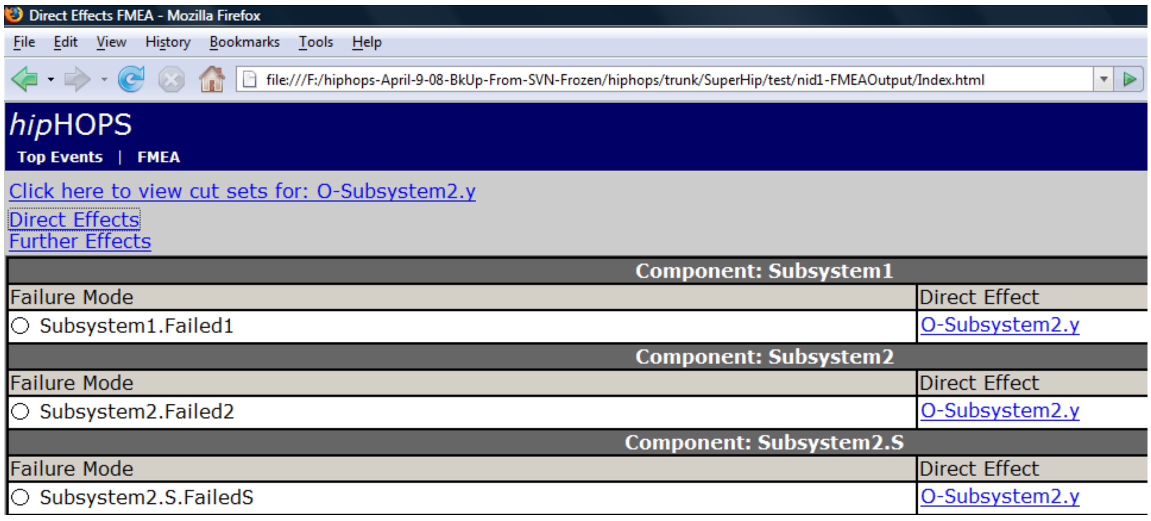
[60] The order of a cut-set is the number of basic events it contains — i.e., a cut set of order n ($n \epsilon \mathbb{N}^+$) contains n basic events.

[61] Both components P and F belong to Subsystem2.

[62] Omission on both inputs of component S (i.e., Si1 and Si2) will ultimately lead to an omission on the output of Subsystem2, thereby causing an occurrence of the top event.

A further detail to note here (about Figure 2—12) is that the numbers appearing to the right of each branch (or sub-branch) are results calculated by HiP-HOPS. These results are based on extra failure data information which we omitted to show within the failure annotations of the components for the sake of clarity — e.g., failure rates as well as cost and weight of subsystems/components implementations.

The HiP-HOPS FMEA synthesiser produces FMEA tables which describe direct and further effects of the failure modes. For example, Figure 2—13 shows that an internal failure of subsystem1 (i.e., Failed1) has a direct effect which is omission on output y of subsystem2. An FMEA table showing the further effects of the failure modes is in appendix A — 3.


Direct Effects FMEA - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///F:/hiphops-April-9-08-BkUp-From-SVN-Frozen/hiphops/trunk/SuperHip/test/nid1-FMEAOutput/Index.html
hipHOPS
Top Events | FMEA

Click here to view cut sets for: O-Subsystem2.y
Direct Effects
Further Effects

Component: Subsystem1	
Failure Mode	Direct Effect
○ Subsystem1.Failed1	O-Subsystem2.y

Component: Subsystem2	
Failure Mode	Direct Effect
○ Subsystem2.Failed2	O-Subsystem2.y

Component: Subsystem2.S	
Failure Mode	Direct Effect
○ Subsystem2.S.FailedS	O-Subsystem2.y

Figure 2—13. Fragment of a HiP-HOPS FMEA table showing the direct effects of failure modes

Overview on optimization of designs with HiP-HOPS

The purpose of optimisation of design models with HiP-HOPS is to have cost-effective and reliable solutions to the problems that have been identified (Papadopoulos, Y. *et al.*, 2005). These problems can be related to safety, availability, reliability, or even too high costs of designs — even though the required dependability criteria like those cited before are met. In this sense, optimisation can be of multiple concerns and may target different objectives, and thereby achieving optimality is a hard problem. The process undertaken can be highlighted through Figure 2—14 which shows (in the left hand side) a system model annotated with failure and cost information, as well as sets of design variants as the functional requirements can be met by many of these (but each variant has different dependability characteristics and cost).

The annotated design model (with function/component elements implementable by variants) is then passed to HiP-HOPS for (dependability vs. cost) evaluation. Thereafter, HiP-HOPS performs automatic optimisation on the design model by selecting the variants to apply in the architecture (right hand side of Figure 2—14); this has been made possible with advances in e.g. genetic algorithms. The returned result is a set of "Pareto" optimal trade-off designs and the process can be iterative to, e.g., review some requirements / design concepts.

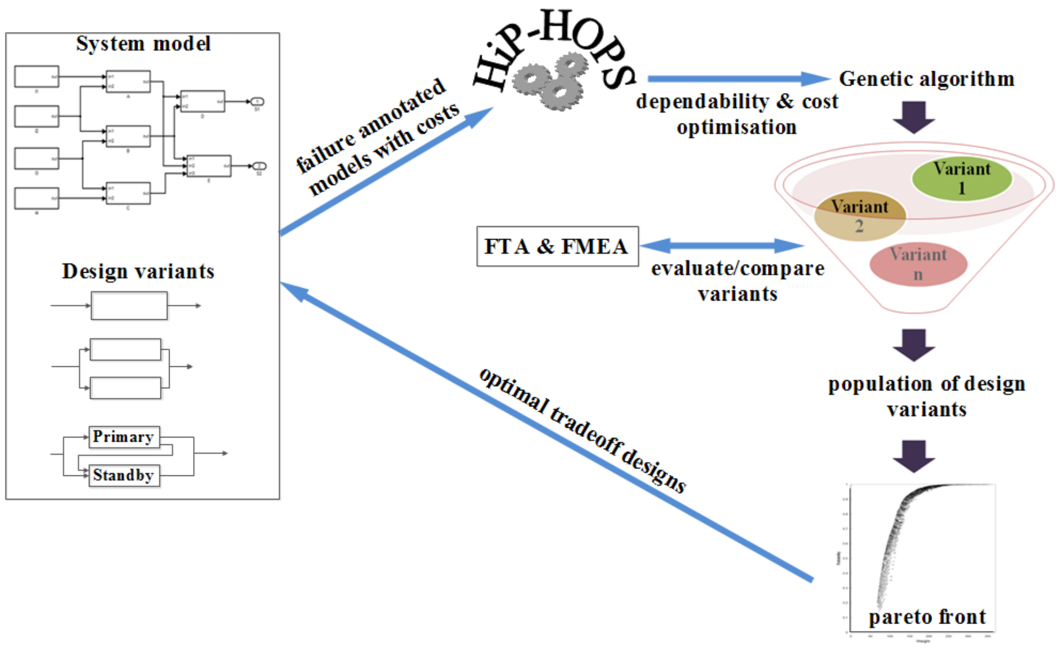


Figure 2—14. HiP-HOPS (the optimisation tool perspective)

2.2.3 Component fault trees

Component Fault Trees (CFTs) enable a compositional method of modelling the failure behaviour of a system through the failure behaviour local to the components. It is a useful way of modelling in a sense that it helps to cope with design changes. Also, it impacts positively the efforts needed for the analysis following these changes (Kaiser, B. *et al.* 2003). Each component of the system can be represented by its own fault tree (i.e., the CFT), which links output failures to input causes through the use of (input / output) ports, logical gates and basic events. Therefore, CFTs are logical structures like standard fault trees and, in that sense, they can still be analysed (qualitatively and quantitatively) using standard fault trees algorithms.

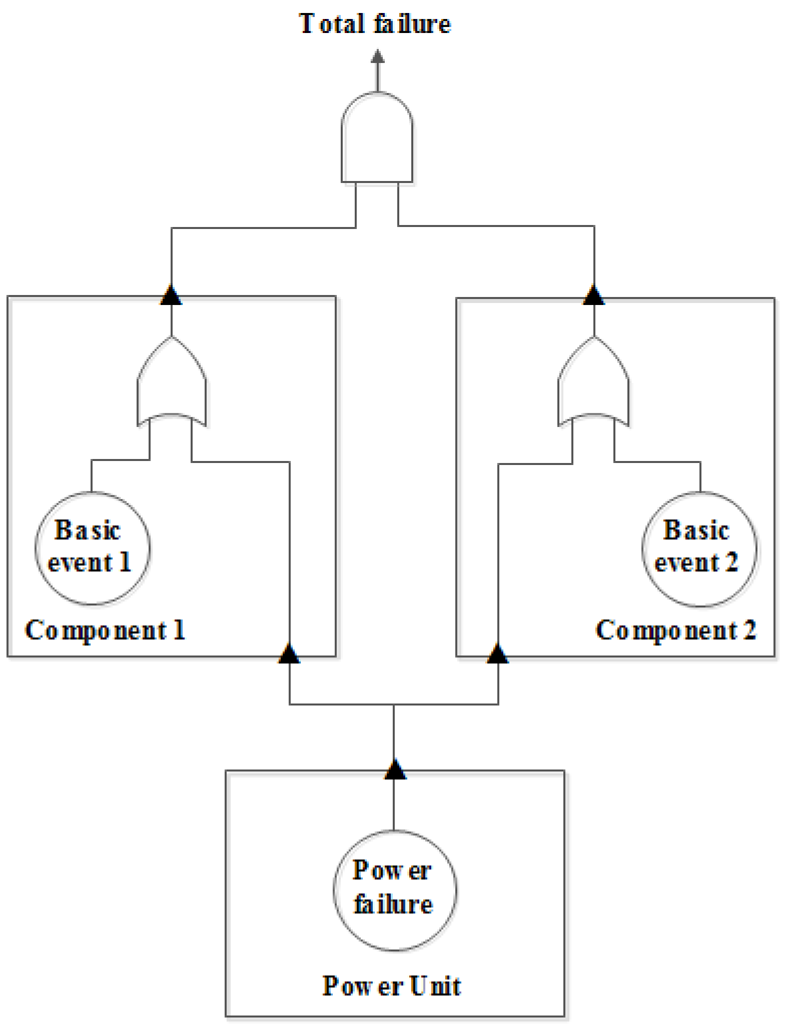


Figure 2—15. Component Fault Tree

For example Figure 2—15 shows two CFTs, each corresponds to a component (e.g. a primary and a secondary) with an internal failure represented by the corresponding basic event. The power unit (bottom) may propagate a power failure to both components via their input ports. The power failure is a common cause failure by which both components fail then propagate their failures through the output ports, and thereby causing a total failure of the system. Though, a system total failure can also happen if both components fail internally.

CFTs, due to their nature, can be developed separately for different parts of the system and the approach is, therefore, favourable for reuse (e.g., making libraries of component types). As for the topology, it appears more as directed acyclic graphs — called Cause Effect Graphs (CEGs) — rather than a real tree structure.

2.3 Dynamic approaches

We discuss in this section some dynamic safety analysis techniques like Altarica and DCCA. As for DFTs, SEFTs and AADL (presented afterwards), these are models which are well-suited to describing the dynamic aspects of complex systems; but the corresponding dynamic analysis requires their conversion to analysis models like GSPNs and Markov chains — though, limited to probabilistic evaluations.

2.3.1 Altarica

The Altarica language provides formal specification capabilities for modelling both functional and dysfunctional[63] behaviours of systems. The technique has the ability to describe complex critical systems using a state-transition representation — called Transition System (TS) in some related literature (like in Bernard R. , 2009) or mode automata (e.g. in Rauzy, 2002). Indeed, the behavioural underlying state/transition formalism was one of the initial choices made for Altarica during its development.

A "TS" representation of a component, called a node, is characterised by variables (whether state variables or data flow variables[64]), events which label the transitions between the states and an assert clause to constraint the data flows according to the states. These concepts are illustrated by the following Altarica modelling of the primary-standby example (i.e., the PS of chapter 1). The node description of the example system shown next has the following state variables (see

Table 2—3): operational, primary, B and S (all declared as Boolean). This is because the PS system can work either in primary mode (i.e., "operational = true" and "primary = true") or in standby mode (i.e., "operational = true" and "primary = false");

[63] By dysfunctional behaviour we mean the behaviour under fault conditions.

[64] Flows render Altarica a data flow language as well.

otherwise, it is completely failed (i.e., "operational = false"). But this is not sufficient, more failure information is needed and particularly while the PS is working in primary mode. For example, the system can operate in primary mode:

(1) with a failed backup component (i.e., "operational = true" and "primary = true" and "B = false");

(2) or with a failed monitoring sensor (i.e., "operational = true" and "primary = true" and "S = false");

(3) or simply under no failure condition anywhere in the system (i.e., all state variables are true).

Such failure information is relevant and impacts the failure behaviour of the PS; for example, a failure of A while the system is in primary mode case (1) would cause a total failure of the PS. However, the effect would be a system degradation instead (but still operational) in the case of primary mode (3) — the graphical TS model in Figure 2—16 clearly shows the relevant failure information, and where the middle of the figure represents a degraded PS system.

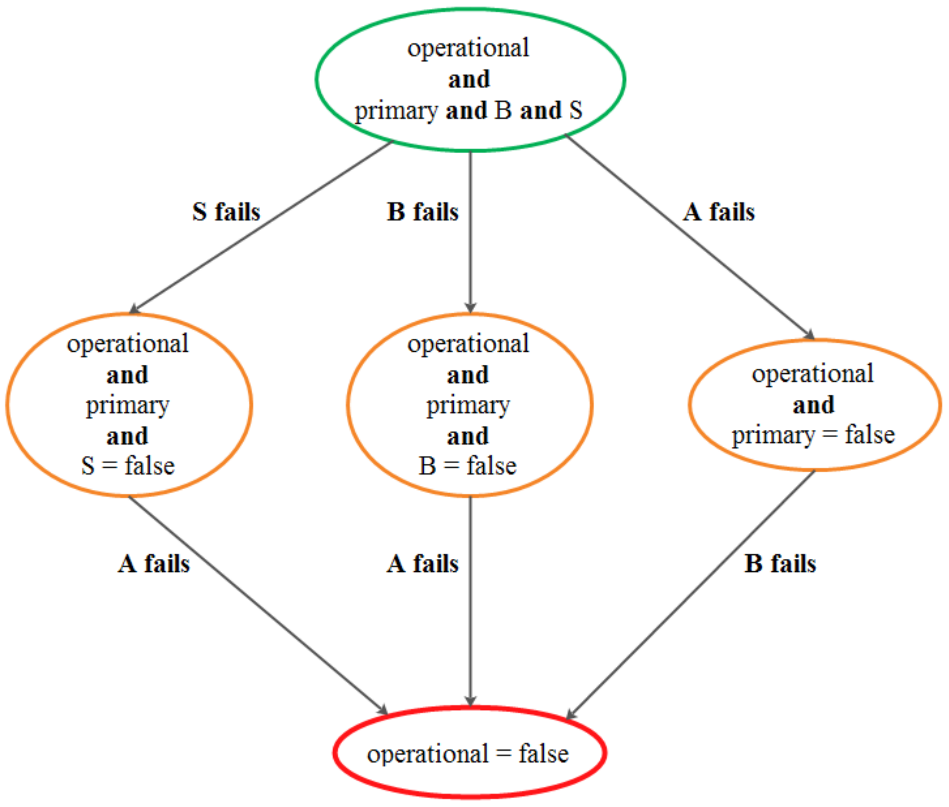

operational and primary and B and S
S fails
B fails
A fails
operational and primary and S = false
operational and primary and B = false
operational and primary = false
A fails
A fails
B fails
operational = false

Figure 2—16. Graphical and labelled transition system associated to the PS Altarica node

Such relevant failure information constitutes an Altarica domain, which is necessary towards the description of a node. For our example, we define the domain as follows (using an abbreviated notation which corresponds to each relevant situation mentioned earlier)

Domain whoisactive = { A0S, AB0, ABS, 0BS };

Where all cases concern the operational status of the PS, but as follows:

A0S means primary mode and B is failed — i.e., case (1)

AB0 means primary mode and S failed — i.e., case (2)

ABS means primary mode and none is failed — i.e., case (3)

0BS means standby mode

Table 2—3. State variables of the PS system.

State	**Type**	**Initial value**
operational	bool	true
primary	bool	true
B	bool	true
S	bool	true

In other words, the system is operating with the first of the two components from the left of the succession “ABS” that is not marked with 0 (i.e., false to indicate that the marked component is failed).

The following represent the textual description of the PS node which conforms to the syntax of Altarica.

```
node      ps
  flow
      input: bool: in;
      output: bool: out;
  state
      operational, primary, B, S : bool;
  event
      Afails, Bfails, Sfails;
  init
      operational := true;
      primary := true;
      B := true;
      S := true;
trans
      operational & primary & B & S |- Afails —> primary := false;
      operational & primary & B & S |- Bfails —> B := false;
      operational & primary & B & S |- Sfails —> S := false;
      operational & (primary = false) |- Bfails —> operational := false;
      operational & primary & (B = false) |- Afails —> operational := false;
      operational & primary & (S = false) |- Afails —> operational := false;
assert
      output = case {operational and input : true,
                           else : false};
edon
```

The keywords “node” and “edon” mark the start and end delimiters of the node respectively. The flow clause represents the input and output flows of the system (and as in Table 2—4) and the states are as emphasised previously. The three failure events correspond to the internal failures of the three components of the PS and initially (as under the “init” clause) all state variables are true.

Table 2—4. Flow variables of the PS system.

Flow	**Type**	**Direction**
input	bool	in
output	bool	out

The transitions clause (i.e., “trans”) shows the values of the state variables before and those that have been impacted after the occurrence of a failure event — these are detailed in the corresponding transition table of the system (Table 2—5). The valuations which are shown for the variables, including flow variables like the input[65] to the system, are called

[65] The input is a single point of failure.

configurations and act as guards for the failure events to cause transitions, thereby leading to new valuations of the variables (i.e., new configurations).

Table 2—5. Transition table of the PS system (i.e., the SM or mode automaton).

Event	Guard	New state			
		operational	**primary**	**B**	**S**
A fails	operational **and** primary **and** B **and** S **and** input	true	false	true	true
B fails	operational **and** primary **and** B **and** S **and** input	true	true	false	true
S fails	operational **and** primary **and** B **and** S **and** input	true	true	true	false
B fails	operational **and** primary = false **and** input	**false**	—	—	—
A fails	operational **and** primary **and** B = false **and** input	**false**	—	—	—
A fails	operational **and** primary **and** S = false **and** input	**false**	—	—	—

Finally, the “assert” clause (which corresponds to Table 2—6) states that there is output as long as the PS is operational with input provided, and there is no output otherwise.

Table 2—6. Assertion table of the PS system.

Assertion	Case	Value
output	operational **and** input	true
output	Anything else	**false**

In a realistic context of modelling, the number of states and transitions can be tremendous. Thus, Altarica was also designed to allow a hierarchical description of a system by declaring instances of nested components, i.e., a tree structure of nodes with a root or main node, intermediate nodes and the leaves of the tree — more details can be found in (Griffault, 2003). We close now this section with a summary such that a failure can be seen as an event which can affect the state of a node, a failure mode can be seen as a transition characterised by a particular failure event and (last but not least) an assert section allows to constrain the flows in function of the states — i.e., when an event occurs (starting from a configuration) the state variables are first assigned with their new values, then the flow variables are calculated in order that the constraints defined by the assertions would be respected. Finally, once a system model has been specified in Altarica, it can then be compiled into lower level formalisms for verification purposes. This can be done using some available compilers which can generate fault trees, state automata or stochastic Petri Nets.

2.3.2 DCCA

The Deductive Cause Consequence Analysis (DCCA) [(Güdemann *et al.*, 2007) and (Ortmeier *et al.*, 2005)] is a formal safety analysis technique. The representation of the system models uses finite automata with temporal logic semantics known as CTL (Computational Tree Logic). The technique uses mathematical methods to determine if a component failure (or combination of component failures) causes a system hazard.

The component failure modes are represented as logical predicates; then a property called ‘criticality’ specifies whether or not a combination of component failure modes can cause a given system failure. This property is defined in CTL and can be extracted automatically from the finite automata in the system model. In the case where the ‘criticality’ property is true, then the corresponding set of failure modes is a critical set (which is similar to a cut-set in standard fault trees). The DCCA technique, thus, attempts to discover all minimal critical sets (i.e., the irreducible sets of failure modes that are necessary to cause a given system failure). Therefore, the technique is also designed to make use of initial results from deductive approaches like FTA; otherwise the efforts which are required to automatically calculate and check each possible set of failure modes would be exponential.

Moreover, results from formalised versions of FTA, such as formal FTA, can also complement DCCA. However, formal FTA annotates each gate with CTL semantics, which can cause problems if the inner nodes of the fault tree cannot be easily formalised. But, DCCA does not require formalisation of the intermediate nodes, and thereby reducing the amount of time required to create the fault trees. Although these various techniques help to reduce the impact of the state-space explosion[66] problem on DCCA, it is still prone to performance and efficiency problems in certain (albeit worst case) scenarios.

A more recent extension to DCCA (Güdemann *et al.*, 2008) enables the technique to take into account the limitation of fault trees in representing the order in which events may occur — particularly the temporal order of events which is significant to the system failure behaviour. This new version of DCCA is so-called Deductive Failure Order Analysis (DFOA). It enables an automatic synthesis of temporal fault trees[67] from system models that are DCCA-annotated. This is achieved by applying DCCA first to obtain the unordered minimal critical sets, then DFOA second to restore a partial ordering to these sets using semantics based on CTL* — i.e., a version which extends CTL with additional operators. With these added capabilities, the DCCA technique can produce results specifying basic combinations of failure modes (for static systems) or sequences of failure modes (for dynamic systems) that must occur to cause a system failure.

2.3.3 Dynamic fault trees

Dugan originated the dynamic fault trees (DFTs) in response to a shortage in modelling sequence-dependent failures by static fault trees (Dugan *et al.*, 1992). This consists mainly of the definition of some special purpose gates for capturing the sequence dependencies in a system, and thereafter solve the dynamic fault tree as a Markov chain. The reason is that it is difficult to develop a correct Markov model for a complex system. However, it is considerably simpler to capture the system dynamics in a DFT model, and thereby converting automatically the fault tree to its equivalent Markov chain. The DFT special gates are summarised as follows:

[66] The state-space explosion, though, is a common problem in all formal model-checking approaches (including Altarica).

[67] These may contain Priority-AND (PAND) and Simultaneous-AND (SAND) gates to represent sequences and simultaneity of events, respectively. These gates are similar to the gates of Pandora (see chapter 3).

a) <u>The functional dependency gate (FDEP)</u>

This gate is typically used to model a dependency between, on the one hand, a set of events that are forced to occur when (on the other hand) an event — i.e., a trigger event — occurs. The trigger event can be for example a failure of the central switch (or computer which acts as a conduit to transmit messages) in a star network topology for instance. The dependent events can therefore be communication failures to the different stations that are connected to the central computer — e.g., each e_i $1 \leq i \leq n$ in Figure 2—17 can represent a communication failure to a station. The FDEP gate has no logical output (it has no impact to any other element in the fault tree) and is, thus, represented with a dashed line.

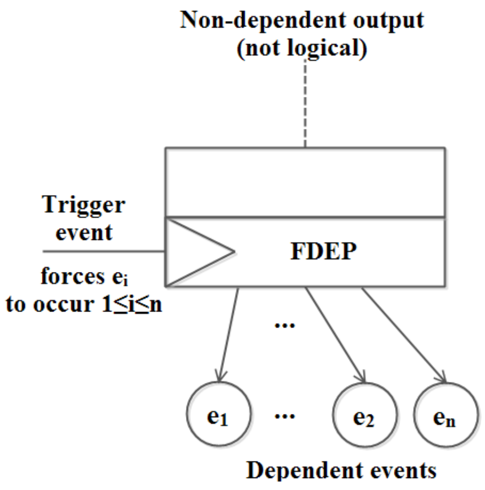


Figure 2—17. Functional-dependency gate

b) <u>Spare gate</u>

This gate is used to model situations where some components (primary units) are substituted by other components (spares). A spare gate has n input in general ($n \geq 2$) and one output: the first input (typically leftmost) relates to the primary unit and becomes true if this unit fails. The other inputs correspond to the different spares envisaged to replace the primary in case of its failure (or failure of any spare already in use since the failure of the primary unit). The substitution of the primary with a spare conforms to a specific order (typically left to right) — i.e., if the primary unit fails, then it should be replaced by the next available spare in sequence. If the primary fails and all spares fail (or are not available[68]), then the output of the spare gate becomes true.

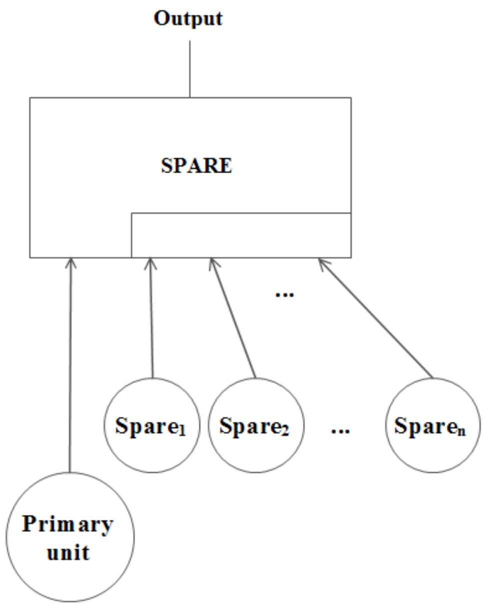


Figure 2—18. Spare gate

The spare units (such as Spare_1, Spare_2…Spare_n in Figure 2—18) have typically reduced failure rates before being switched into active use (i.e., warm spares) or do not fail dormant at all (i.e., cold spares), exception made for hot spares which have the same failure rate irrespective of any state: dormant or active. Systems using cold or warm spares cannot be modeled accurately using combinatorial fault trees. The sequence-dependent failures (i.e., the order in which the primary unit and spares may fail) need to be preserved for a more accurate analysis.

c) Priority-AND gate

This gate captures sequence-dependent failures like in the example of the primary standby system used in (Mahmud *et al.*, 2010) — which is depicted in chapter one through Figure 1—1. In the example, it was shown that the order in which the monitoring sensor S and the primary unit A fail is relevant to the system failure behaviour. It is, thus,

[68] A spare is not available if, for example, it is shared with another gate, and it is still in use in lieu of the primary unit of that other gate.

modelled using a Priority-AND gate — like the one (with two inputs) in the centre of Figure 2—19. The figure shows that a complete failure of the system is caused by an omission of input, the sensor failing before the primary unit or both primary and backup units fail. Assuming that the output ‘Out’ has no failure modes on its own — it must only receive input from at either unit, primary or standby, for the system to function overall.

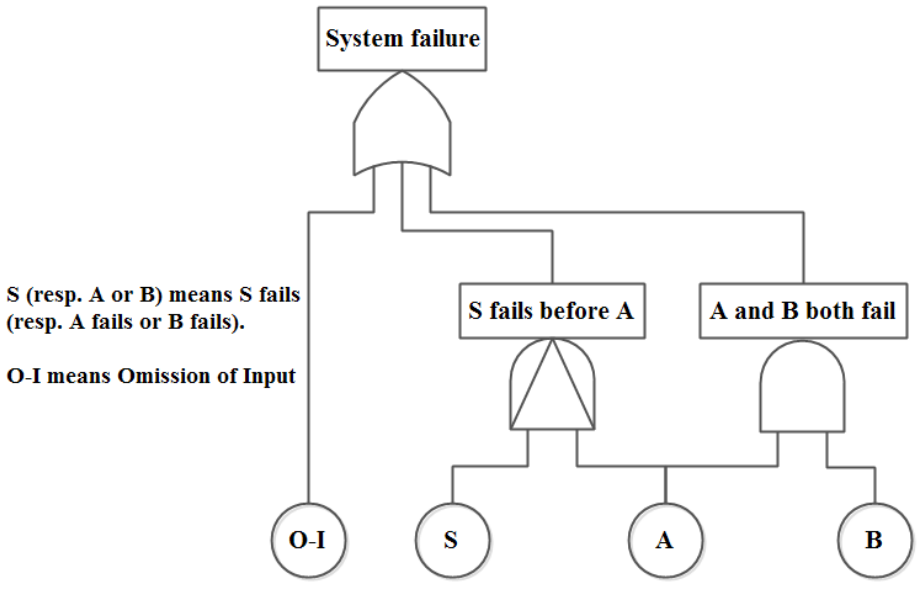

System failure
S (resp. A or B) means S fails (resp. A fails or B fails).
O-I means Omission of Input
S fails before A
A and B both fail
O-I
S
A
B

Figure 2—19. DFT of the PS system

We wish to note that there are other gates that can be used in a DFT, like the Sequence-enforcing[69] gate which forces events to occur in a specific order.

The problem with DFTs is that these are primarily designed for quantitative analysis, typically solved using Markov chains. The following section shows the equivalent Markov model for the DFT of Figure 2—19. It highlights some path redundancies in the model — which can only be eliminated through a qualitative analysis. We wish to note, though, that Markov models are well-known with the state explosion problem. Therefore, any effort towards optimising Markov models will have a positive impact on the analysis overall.

[69] For more details, we refer the reader to (Dugan *et al.*, 1992) and (Vesely *et al.*, 2002).

2.3.4 Markov analysis

Markov Chains (MCs) are very powerful in capturing the dynamic aspects of systems and in the evaluation of safety measures (Trivedi, 2001). They are often used to provide quantitative solutions for dynamic models like DFTs, SEFTs, GSPNs…etc. The underlying formalism of a MC is a state-transition system and the quantitative solution is reached by solving the corresponding differential equations. However, relying on Markov models only for the analysis also means that:

1. Qualitative analysis is not encouraged.
2. The failure rates (which label the transitions in the MCs) are limited to exponential distributions.
3. The Markov models themselves are large and cumbersome, and thus impractical for systems with several components.

A Markov model describes the behaviour of a system such that a state of the model represents a state of the system whether operational or not, and an arc represents a failure event[70]. This structure of the model allows writing the probability of being in a state in the form of an equation according to the source (resp. destination) states of the incoming (resp. outgoing) arrows. For example, let us assume a simple parallel system composed of two components A and B, which can fail independently[71] with the rates λ_A and λ_B respectively (see Figure 2—20).

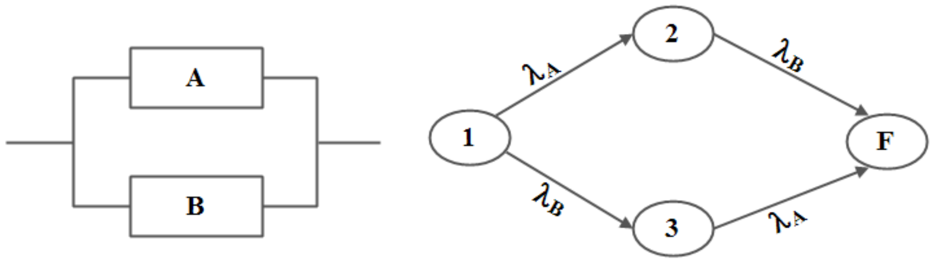


Figure 2—20. A Markov model of a simple non-repairable parallel system

[70] This can be a repair event in the case of repairable systems.

[71] In the example, we suppose that there is no common failure mode by which the two components fail simultaneously.

The state 1 represents a state of the system where both components A and B are OK. However, the state 2 (resp. state 3) represents a state of the system where A (resp. B) has failed, and finally the state F represents a state where both component are failed.

The Markov solution provided by the model of Figure 2—20 is described as follows, and where P_1, P_2, P_3 and P_F are the probabilities of being in states 1, 2, 3 and F respectively.

$$\frac{d}{dt} P_1(t) = -(\lambda_A+\lambda_B)\, P_1(t)$$

$$\frac{d}{dt} P_2(t) = \lambda_A\, P_1(t) - \lambda_B\, P_2(t)$$

$$\frac{d}{dt} P_3(t) = \lambda_B\, P_1(t) - \lambda_A\, P_3(t)$$

$$\frac{d}{dt} P_F(t) = \lambda_B\, P_2(t) + \lambda_A\, P_3(t)$$

The solution of the differential equations is given by:

$$P_1(t) = e^{-(\lambda_A+\lambda_B)t}$$

$$P_2(t) = e^{-\lambda_B t} - e^{-(\lambda_A+\lambda_B)t}$$

$$P_3(t) = e^{-\lambda_A t} - e^{-(\lambda_A+\lambda_B)t}$$

$P_F(t) = e^{-(\lambda_A+\lambda_B)t} - e^{-\lambda_A t} - e^{-\lambda_B t} + 1$ (1 is the constant with respect to the initial condition at time $t_0 = 0$, $P_F(t_0) = 0$)

The probability given by the Markov solution for failure of A and B is the same result given by combinatorial techniques for two components whose failures are independent, and hence:

$$\begin{aligned} P(\text{A failing AND B failing}) &= P(\text{A failing}) \times P(\text{B failing}) \\ &= (1 - e^{-\lambda_A t}) \times (1 - e^{-\lambda_B t}) \\ &= e^{-(\lambda_A+\lambda_B)t} - e^{-\lambda_A t} - e^{-\lambda_B t} + 1 \end{aligned}$$

A similar Markov model of a simple non-repairable parallel system composed of two components failing independently at the same rate λ is represented by Figure 2—21, where state 2 in this model represents a state of the system such that either A or B has failed (not both). This model is optimised but preserves the result which corresponds to the probability of being in state F (i.e., both components have failed).

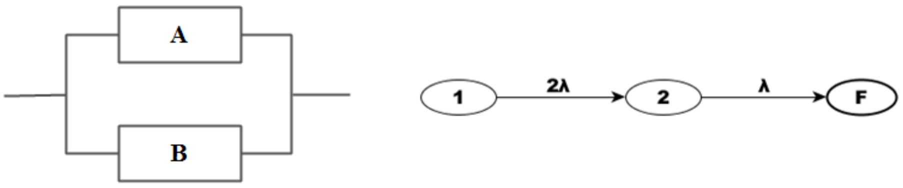


Figure 2—21. An optimised Markov model — same failure rate λ for both components

The Markov solution given by the model of Figure 2—21 is as follows:

$$\frac{d}{dt} P_1(t) = -2\lambda P_1(t)$$

$$\frac{d}{dt} P_2(t) = 2\lambda P_1(t) - \lambda P_2(t)$$

$$\frac{d}{dt} P_F(t) = \lambda P_2(t)$$

The solution of the differential equations is given by:

$P_1(t) = e^{-2\lambda t}$ (at $t_0 = 0$, $P_1(t_0) = 1$)

$P_2(t) = 2(e^{-\lambda t} - e^{-2\lambda t})$ (at $t_0 = 0$, $P_2(t_0) = 0$)

$P_F(t) = e^{-2\lambda t} - 2 e^{-\lambda t} + 1$ (at $t_0 = 0$, $P_F(t_0) = 0$)

$P_F(t)$ is the same as given previously with $\lambda_A = \lambda_B = \lambda$.

Optimisation of Markov models is useful considering their growing size and can be applied to some patterns which are not uncommon in safety-critical systems, like the Markov model of Figure 2—22. The model can be used for the quantitative analysis of the PS example system described in chapter 1 (see Figure 1—1). There are three categories of states: operational, degraded and completely failed. The transitions are labelled with the component failure rates λ_A, λ_B and λ_S for the primary A, the backup B and the monitoring sensor S, respectively. The states are textually described with a succession of symbols representing the system components in the specific order "A, B, S”. This means that the system is functioning with the first one of the two leftmost components that is not substituted with the symbol ‘X’ (which designates a failed component). The symbol ‘!B’ in the figure means that component B is not failed, but cannot take over A’s job. This is due to a premature failure of the sensor which is, thus, unable to activate B upon failure of A.

In Figure 2—23, we optimise the Markov model of Figure 2—22 whose failure states — ‘XX_S’, ‘$X!B_X$’ and ‘XX_X’ — are all merged into one state “Total Failure” and

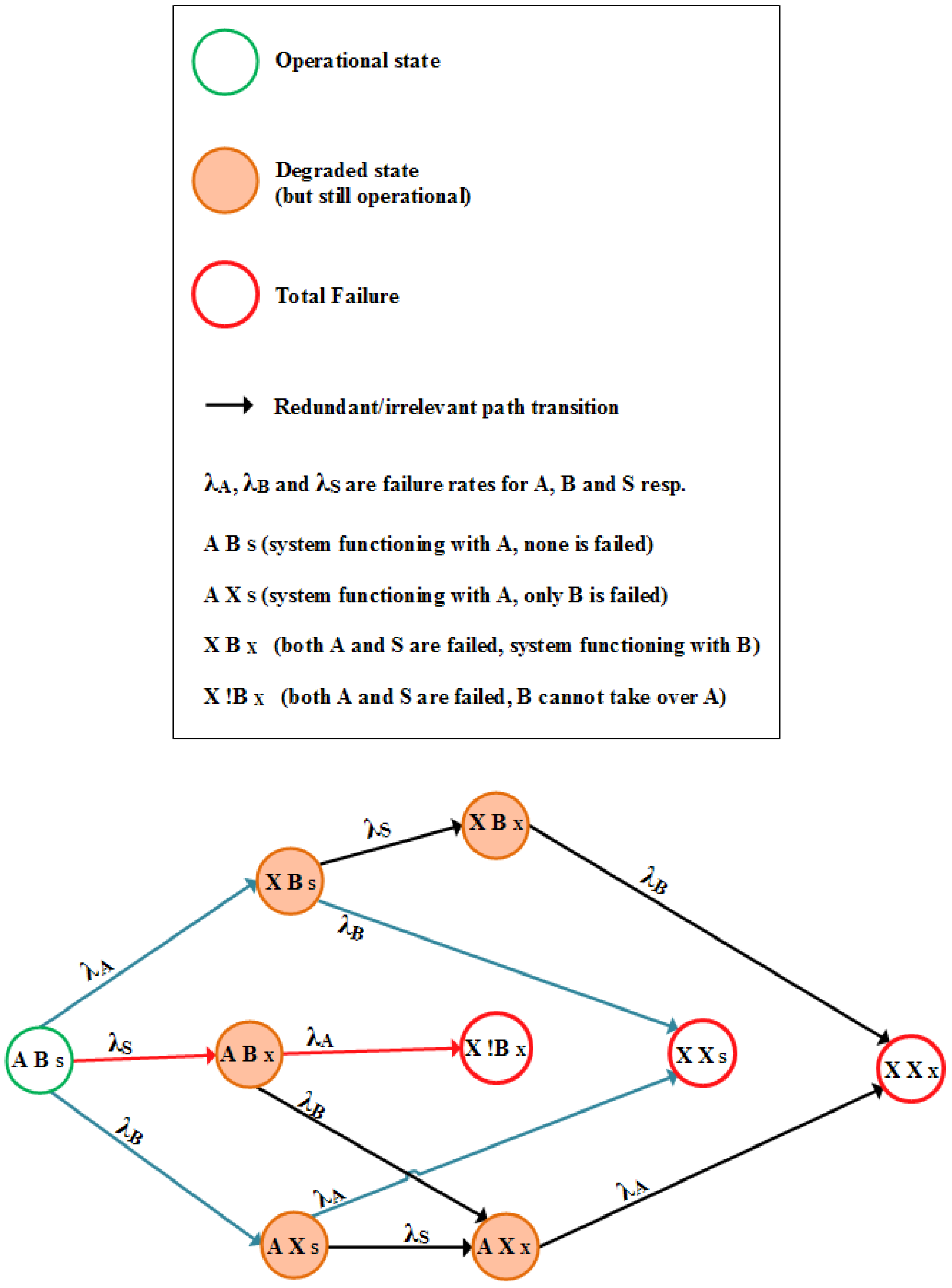

Operational state
Degraded state (but still operational)
Total Failure
Redundant/irrelevant path transition
λA, λB and λS are failure rates for A, B and S resp.
A B s (system functioning with A, none is failed)
A X s (system functioning with A, only B is failed)
X B x (both A and S are failed, system functioning with B)
X !B x (both A and S are failed, B cannot take over A)
A B s
X B s
X B x
A B x
X !B x
X X s
X X x
A X s
A X x
λA
λS
λB

Figure 2—22. Markov model for the primary-standby system.

the redundant transitions are removed. For example once the transition with rate λ_A occurs from the initial state, then only the transition labelled with rate λ_B matters. Therefore, the transitions from "XB$_S$" to "XX$_X$" through "XB$_X$" of Figure 2—22 are all removed. Strictly speaking, the probability of the state "Total Failure" (of Figure 2—23) is the sum of the probabilities of the states 'XX$_S$', 'X!B$_X$' and 'XX$_X$' (of Figure 2—22).

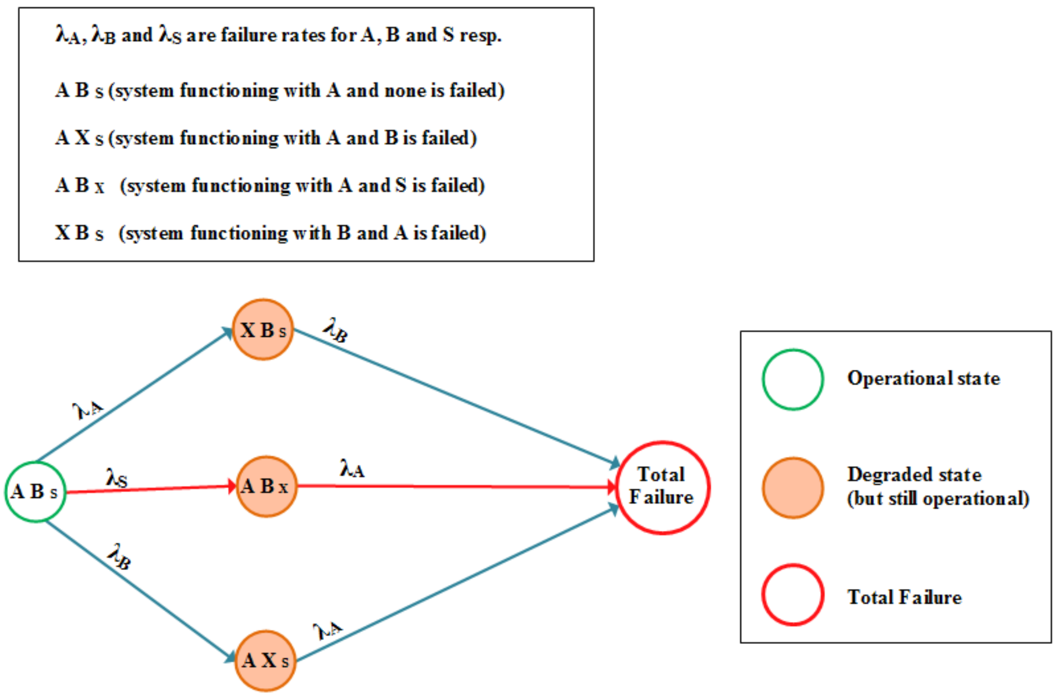

λ_A, λ_B and λ_S are failure rates for A, B and S resp.
A B s (system functioning with A and none is failed)
A X s (system functioning with A and B is failed)
A B x (system functioning with A and S is failed)
X B s (system functioning with B and A is failed)
A B s
X B s
A B x
A X s
Total Failure
λ_A
λ_S
λ_B
λ_B
λ_A
λ_A
Operational state
Degraded state (but still operational)
Total Failure

Figure 2—23. Optimized Markov model for the primary-standby system.

Moreover, if the components A and B have the same failure rate, i.e., $\lambda_A = \lambda_B = \lambda$, then the Markov model in Figure 2—23 can be further optimised as in Figure 2—24. In chapter 4 (section 4.4), we present how we can use the proposed SM conversion algorithm to automatically optimise Markov state diagrams. Briefly in this section, one way of doing it is through their conversion to temporal FTs, which can then be minimised using the Pandora temporal laws. This way, the optimal descriptions of the Markov models can be automatically extracted from their original (often much bigger) diagrams.

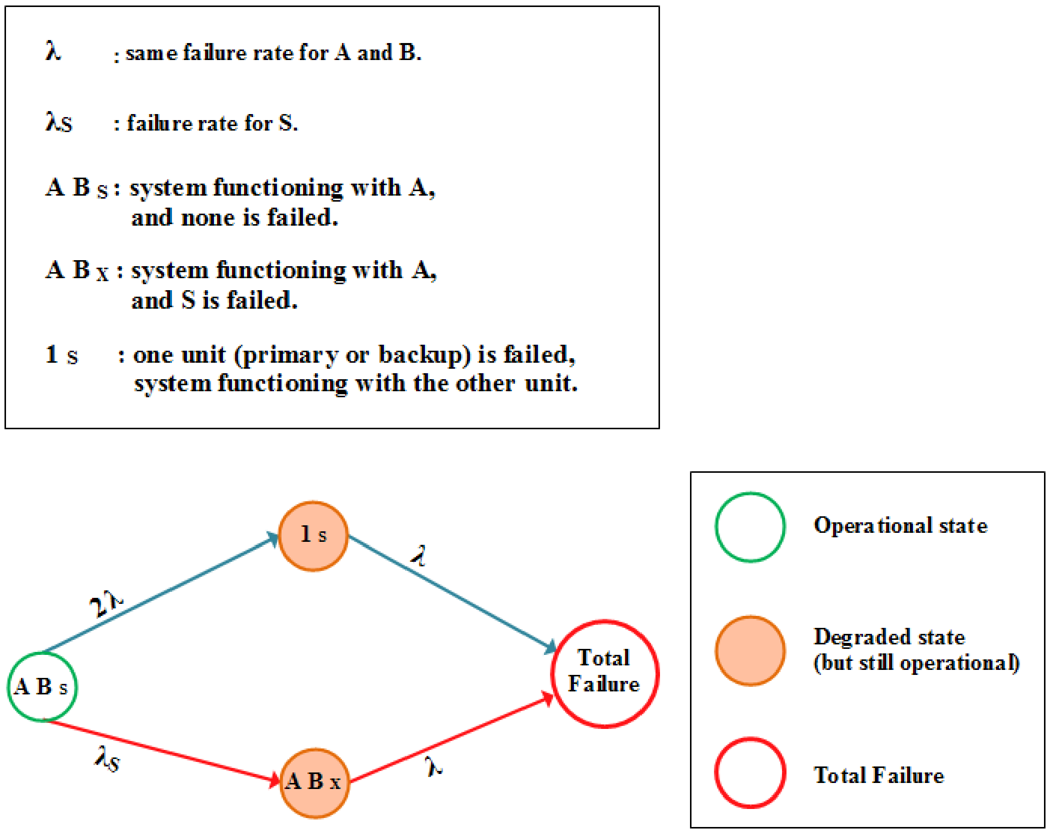

λ : same failure rate for A and B.
λs : failure rate for S.
A B s : system functioning with A, and none is failed.
A B x : system functioning with A, and S is failed.
1 s : one unit (primary or backup) is failed, system functioning with the other unit.
1 s
λ
2λ
A B s
Total Failure
λs
A B x
λ
Operational state
Degraded state (but still operational)
Total Failure

Figure 2—24. A more optimized Markov model for the primary-standby system.

2.3.5 State event fault trees

State Event Fault Trees, or simply SEFTs (Grunske, L. *et al.* 2005), is another failure modelling technique which introduces representation of the temporal order of events. SEFTs enable to represent states and transitions in fault trees and, thus, allow preserving the significance of the temporal semantics expressed in state-based models of systems (and which one wants to analyse). This allows a more accurate modelling and analysis of systems exhibiting dynamic failure behaviour, rather than using standard (combinatorial) fault trees.

In the SEFT approach of modelling, states are conditions which last a period of time and events are instantaneous and may trigger state transitions (i.e. changes of state). These are described by Figure 2—25 together the representation of a component.

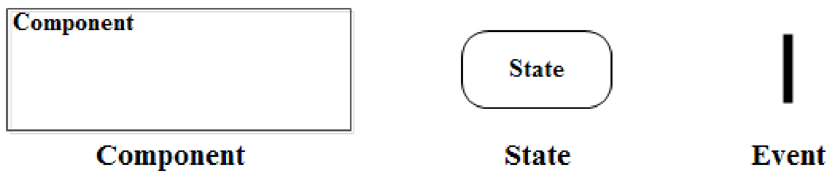


Figure 2—25. Basic SEFT elements

Moreover, the SEFT modelling capabilities allow to clearly differentiate between causal and sequential relations by means of different ports — i.e., event ports (causal) and state ports (sequential), see Figure 2—26.

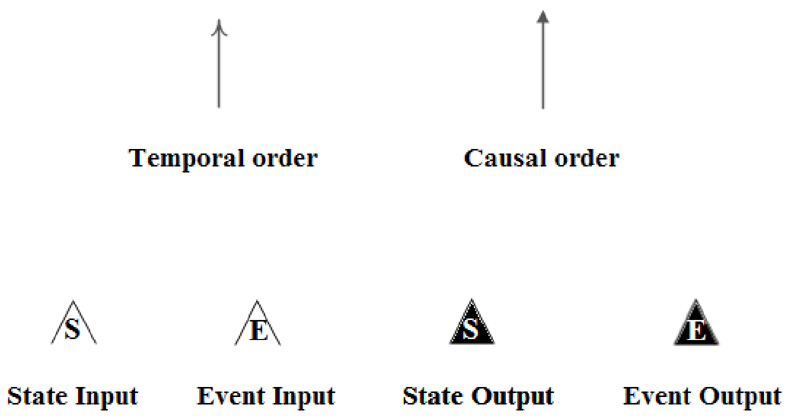


Figure 2—26. SEFT relations and ports

Events can be combined using a conjunction (&) gate and/or a disjunction (≥1) gate to, e.g., trigger another event. Likewise, states may be combined to express that, e.g., two or more states must be true for an event to occur (SEFT gates are depicted by Figure 2—27).

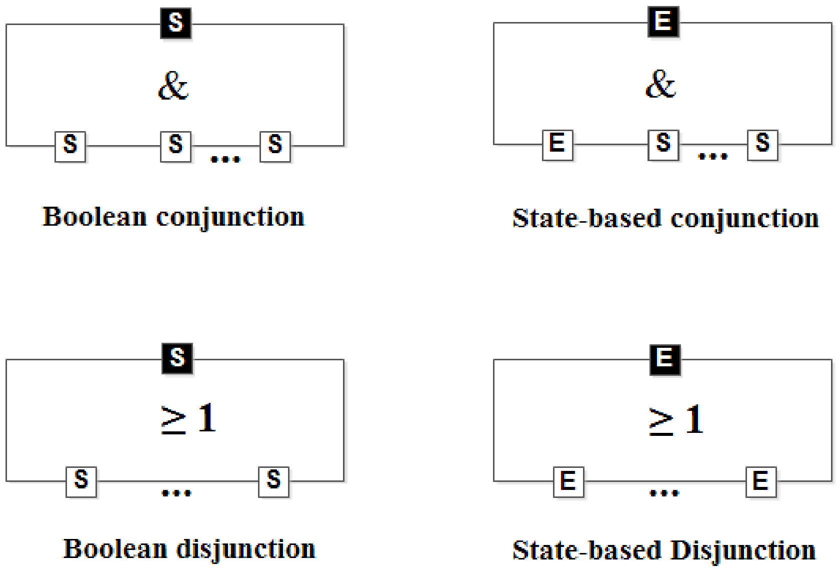


Figure 2—27. Basic SEFT gates

To better illustrate these different modelling elements, we present the following hydraulic distribution system (see Figure 2—28), and which consists of a water tank, two pumps and water distribution. A system failure can be caused by loss of the distribution, or by either a water leakage in the tank (empty tank) or both pumps fail.

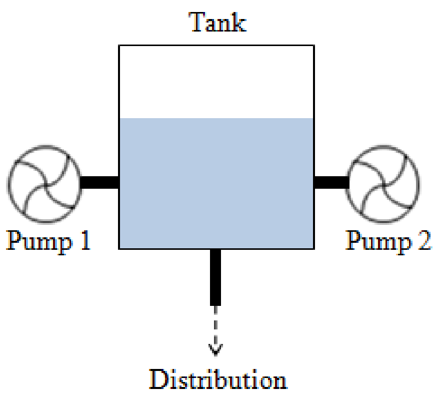


Figure 2—28. Hydraulic Distribution System — adapted from (Bernard R. , 2009).

The corresponding SEFT (see Figure 2—29) shows a conjunction of the two input states pump1 failed and pump2 failed, the corresponding output is then combined in a disjunction with the state “Tank Empty”. Likewise, the output of this second gate is then combined in a disjunction with the event “Loss of distribution”. This is to represent the system failure scenario (top-event of the SEFT).

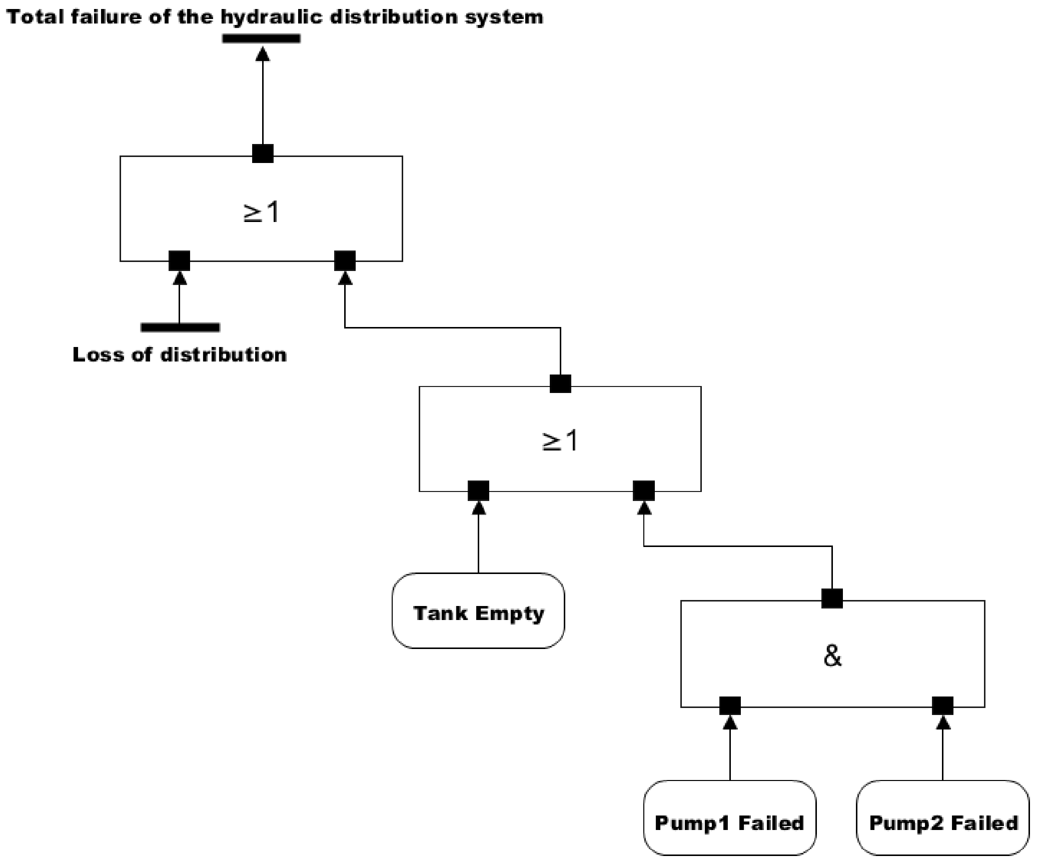


Figure 2—29. SEFT of the Hydraulic Distribution System

Thereafter, the failure behaviour of each component is described by a state machine and a component can be in one state at a time (the active state). A state machine representation in SEFT is shown in Figure 2—30; it describes the failure behaviour of a pump component (both pumps are identical), and such that:

- “Initialises” is triggered by an event through the event input port (causal relation) and as shown in the bottom of the figure.
- “Init” is an internal initialiser event (cannot be externally triggered). It sets the pump “ON” (temporal relation).
- “Fails” is an event which changes the state of the pump from “ON” to “Failed”.
- The “Failed” state is linked to the state output port of the component (causal order).

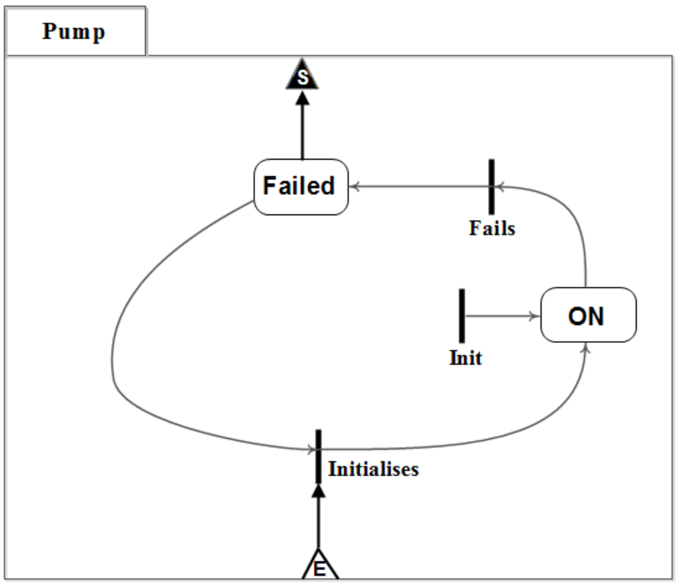


Figure 2—30. Pump component

For dependability assessments, though, SEFTs need to be converted to other models like Deterministic Stochastic Petri Nets (DSPNs) (Ciardo *et al.*, 1993), then automated analysis can be performed by external tools like TimeNet (German *et al.*, 1995). However, this approach is limited to quantitative analysis and is less suitable for qualitative analysis.

2.3.6 AADL

The Architecture Analysis and Design Language (AADL) was developed under the auspices of the International Society of Automotive Engineers (SAE) to describe hardware and software architectures of performance-critical systems in particular. The language has capabilities for the representation of the system as an assembly of software components mapped onto an execution platform. Recently, the language has been extended with an error model annex for dependability modelling.

The dependability model consists of both the architecture model and the error model. The former consists of the description of the components with their connections and interactions (also known as the nominal[72] architecture). The latter addresses the behaviour of components in the presence of, on the one hand, internal faults and repair events and, on the other hand, external propagations from the component environment — only components which are associated with error models (including connections) are part of the AADL dependability model.

An AADL element (e.g. component or connection) can be associated with an error model (effectively a state machine), which consists of a model type and at least one error model implementation. A system error model is a composition of the error models of its components or subsystems and is meant to capture hazards at system level. Error models in AADL can be reusable through the use of an annex library, and there are two kinds of reusable error models: the basic model and the derived model. The basic error model consists of a declaration of error states for a component or a connection, as well as properties to specify changes of states due to error events and propagations. However, the derived error model is the definition of the error state of a component in terms of the error states of its subcomponents.

Figure 2—31 shows a graphical representation of an error model associated with a component. "ON" is the initial state which can change to "OFF" due to a failure event, and then back to "ON" following a repair. It also shows the propagation of an output deviation ("Deviation") — it acts like an event which does not change the local state but is visible outside the component.

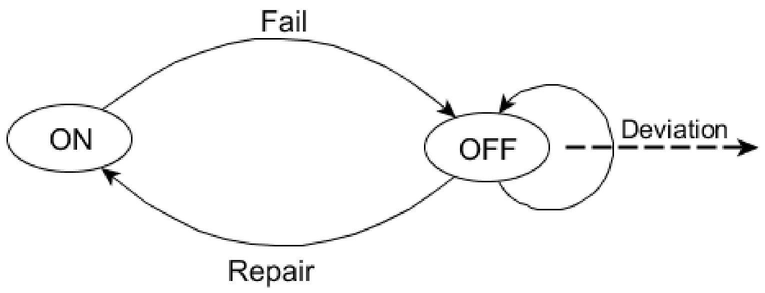


Figure 2—31. AADL SM for a component with an output deviation

[72] It describes both the structural and some aspects of the behavioural designs without any failure condition.

Table 2—7 shows the corresponding textual description of the error model example — the left hand side is for the declaration of states, events and propagations. There is one output deviation in this case with a default value of 0.8 which represents its probability of occurrence. The right hand side of the table represents the implementation part — i.e., the transitions of the state machine (including propagations) as well as failure distributions (poisson) assigned to the events fail and repair (for quantitative assessments).

Table 2—7. Error Model Definition of a component propagating an output deviation — textual description.

Declaration	**Implementation**
error model Example1	**error model implementation** Example1.basic
features	**transitions**
ON: **initial error state;**	ON- [Fail] ->OFF**;**
OFF: **error state;**	OFF- [**out** Deviation] ->OFF**;**
Fail, Repair: **error event;**	OFF- [Repair] ->ON**;**
Deviation: **out error propagation**	**Properties**
{Occurrence => **fixed** 0.8**};**	Occurrence => **poisson** 1.0e-3 **applies to** Fail**;**
end Example1**;**	Occurrence => **poisson** 1.0e-4 **applies to** Repair**;**
	end Example1.basic**;**

A description of a similar example (graphical and textual) with input / output propagation is in Appendix A — 4.

2.4 Discussion

In this chapter, we have presented an overview of relevant safety analysis techniques. We discussed in more details the shortcomings of techniques for dynamic safety analysis and expanded on the motivation of this work which was presented in the introduction. For example, an objective of Altarica (as well as AADL) is to build static fault trees from state models. This is quite different from that of dynamic fault trees which is to represent non-formally a behaviour with dynamic gates to be mainly analysed quantitatively afterwards.

Although state-machine become increasingly a popular notation for description of dynamic behaviour, there is clearly a lack of techniques for the qualitative and quantitative safety analysis of such models. Synthesis and analysis of fault trees has been proposed but classical combinatorial fault trees cannot capture dynamic aspects. In chapter 3 we look in more detail to Pandora an expansion of the fault tree notation with temporal semantics that potentially solves this problem.

3 Pandora

3.1 Introduction

Since FTA is primarily a combinatorial analysis technique, its application on dynamic systems requires that the modelling and analysis aspects of the approach are extended. Indeed, some extension attempts of FTA are not new; the Priority-AND gate (PAND see Figure 3—1) is plausibly one of the first attempts at overcoming the dominant static aspect of FTA. The gate dates back to the mid-seventies (Fussel *et al.*, 1976) and is conceived such that its output is true if all inputs occur, and if they occur in a specific order (typically left to right). This allows the analyst to put a set of events into a sequence[73] (i.e., temporal order) like the events e_1, e_2… e_n in Figure 3—1, and thus helps to express a kind of time-dependent information in the fault tree.

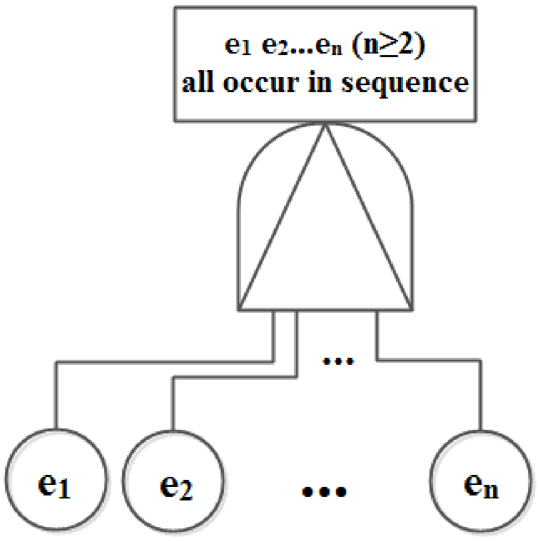


Figure 3—1. Pandora's PAND gate with n inputs (n≥2)

Apart from the modelling perspective of the PAND, there have been efforts made to work around its quantitative analysis (Fussel *et al.*, 1976). But, qualitative analysis of

[73] The explicit definition of the sequence is typically a conditioning event, even though implicitly it is commonly understood that the temporal order of inputs is left to right.

fault trees that contain such gates is neglected in general. Moreover, some issues seemingly remain vague in the following definition of the PAND gate which is provided by the fault tree handbook.

> *"The PRIORITY AND-gate is a special case of the AND-gate in which the output event occurs only if all input events occur in a specified ordered sequence. The sequence is usually shown inside an ellipsis drawn to the right of the gate."* (Vesely *et al.*, 1981, p IV-11)

One issue, for instance, can be about events which can appear more than once in a sequence. Another unclear issue can be about events which may occur at the same time, or events that may contradict[74] each other. Concerning the first issue, we can assume that, under condition of no repair, faults which occur will continue to exist as suggested in the handbook (Vesely W. E., 1981). However, there are still no precise indications regarding simultaneity and contradiction of events. Besides, the software tools — older ones like in (Worrell *et al.*, 1978) or even recent ones e.g. in (Isograph, 2002) — seem to simply treat the PAND as an ordinary AND gate for the logical reductions. But, we have seen how this assumption can lead to erroneous results when it comes to the analysis of dynamic systems — like the primary-standby example system seen in chapter 1 (Figure 1—1). The order in which the primary component and the monitoring sensor fail influences the system failure behaviour overall (see the SM of Figure 1—2 in the same chapter). Thus, it is necessary for a correct analysis to preserve the meaning of the PAND gate, not only during the conversion of the SM to TFTs, but also during the logical reductions of these TFTs.

Pandora (Walker M., 2009) provides two other temporal gates, the Priority-OR (POR) gate and the Simultaneous-AND (SAND) gate. The POR gate (see Figure 3—2) has at least two input events, the first input (typically the leftmost) represents an event which must occur before any other input event of the gate or simply occurs alone; otherwise the POR evaluates to false. All input events except the first one are optional, i.e., they may not occur but the POR output still evaluates to true. However, the SAND gate (see Figure 3—3) evaluates to true only if all its input events occur, but they also must occur at the same time. The symbols which correspond to the temporal gates are represented in Table 3—1.

[74] For example, (e_1 PAND e_2) and (e_2 PAND e_1) where e_1 and e_2 are two events.

Table 3—1. Temporal gate notation

Gate	**Symbol**
PAND	<
POR	|
SAND	&

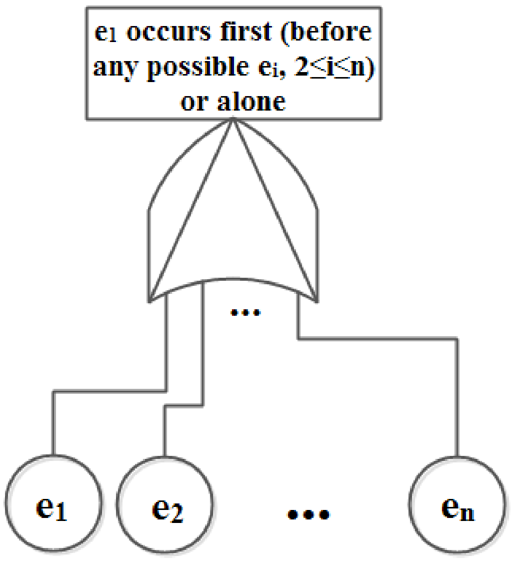


Figure 3—2. Pandora's POR gate with n inputs (n≥2)

In this thesis, we use a variation of Pandora which omits the SAND gate. This is due to the SM representation of the analysis-related information of the system to analyse; though, we assume that events that occur at the same time are due to a common cause failure (which is represented by its corresponding event in the SM). Similarly, the Pandora temporal laws using the SAND operator are removed from this variation.

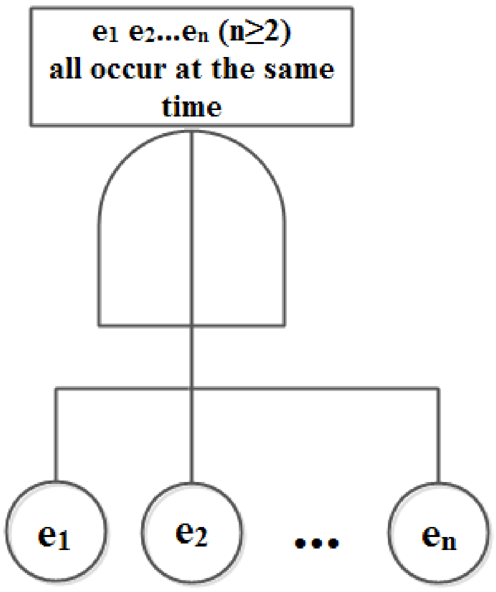


Figure 3—3. Pandora’s SAND gate with n inputs (n≥2)

3.2 Pandora temporal laws adapted to SM-based analysis

Temporal fault trees, in this thesis, are generated from the SMs which describe the failure behaviour of dynamic systems; this will be fully detailed in chapter 4. As we have seen for the example of the primary standby, it is necessary to preserve the significance of the SM temporal semantics during the conversion to TFTs; but this is not sufficient. We also need to preserve the significant fault sequences all along the logical reduction of the generated TFTs.

Pandora (Walker, 2009) provides a set of temporal laws, thereby enabling a true temporal qualitative analysis of dynamic systems. However, many of these laws contain the SAND operator and, therefore, these need to be identified for some alteration — the laws which were conceived to transform expressions containing exclusively SAND operator(s) will be completely removed from our selection.

We present in this section a set of Pandora temporal laws selected to suit our approach for a SM-based analysis. We wish to note, though, that these laws can be verified using temporal truth tables (Walker, 2009) which are similar to Boolean truth tables, but suited for temporal logic. For example, Table 3—2 shows that $A<B \Leftrightarrow (A|B).B$, by using values in the table such that zero indicates that the corresponding event has not occurred (i.e., false). Any other value (strictly greater than zero) indicates that the

corresponding event has occurred (i.e., true). Two events with non-zero values are either simultaneous (holding the same value) or in sequence (the event with the smaller value occurs before the event with the higher value).

Table 3—2. A Pandora temporal truth table

A	B	A<B	A|B	(A|B).B
0	0	0	0	0
0	1	0	0	0
1	0	0	1	0
1	1	0	0	0
1	2	2	1	2
2	1	0	0	0

Table 3—2 shows that A<B evaluates to zero in all rows, except when A and B are both assigned with different non-zero values, one for A and two for B, and A<B evaluates to the higher value (i.e., two). This conforms to the definition of the PAND gate whose input events must all occur, and in sequence so that the output of the gate evaluates to true (i.e., the value assigned to the last input event of the sequence).

Commutative Laws

1. $A<B \neq B<A$
2. $A|B \neq B|A$

Associative Laws

1. $A<(B<C) \neq (A<B)<C$
2. $(A<B)<C \Leftrightarrow A<B<C$

3. $A|(B|C) \neq (A|B)|C$
4. $(A|B)|C \Leftrightarrow A|B|C$
5. $A<(B<C) \Leftrightarrow (A.B)<C \Leftrightarrow (A<C).(B<C)$
6. $(A|B)|C \Leftrightarrow A|B|C \Leftrightarrow A|(B+C) \Leftrightarrow A|B.A|C$

<u>Distributive Laws</u>

1. $A < (B.C) \Leftrightarrow B.(A<C) + C.(A<B)$
2. $A < (B+C) \Leftrightarrow (A|C).(A|B).(B+C)$
3. $A < (B<C) \Leftrightarrow (A<C).(B<C)$
4. $A < (B|C) \Leftrightarrow (A<B).(B|C)$
5. $A < (B|C) \Leftrightarrow (A|B).(B|C)$
6. $A \mid (B+C) \Leftrightarrow (A|B).(A|C)$
7. $A \mid (B.C) \Leftrightarrow A|B + A|C$
8. $A \mid (B<C) \Leftrightarrow (A|C) + (A|B) + A.(C<B)$
9. $A \mid (B|C) \Leftrightarrow (A|B) + (A.C<B)$
10. $(B+C) < A \Leftrightarrow (B<A) + (C<A)$
11. $(B.C) < A \Leftrightarrow (B<A).(C<A)$
12. $(B<C) < A \Leftrightarrow (B<C).(C<A)$
13. $(B|C) < A \Leftrightarrow (B<A).(B|C)$
14. $(B+C) \mid A \Leftrightarrow (B|A) + (C|A)$
15. $(B.C) \mid A \Leftrightarrow (B|A).(C|A)$
16. $(B<C) \mid A \Leftrightarrow (B|C).(C|A)$

17. $(B|C) \mid A \Leftrightarrow (B|C).(B|A)$

18. $(A+B) < (C+D) \Leftrightarrow (A|C).(A|D).(C+D) + (B|C).(B|D).(C+D)$

19. $(A.B) < (C.D) \Leftrightarrow (A<D).(B<D).(C<D) + (A<C).(B<C).(D<C)$

20. $(A|B) < (C|D) \Leftrightarrow (A<C).(A|B).(C|D)$

21. $(A+B) \mid (C+D) \Leftrightarrow (A|C).(A|D) + (B|C).(B|D)$

22. $(A.B) \mid (C.D) \Leftrightarrow (A|C).(B|C) + (A|D).(B|D)$

23. $(A<B) \mid (C<D) \Leftrightarrow (A<B).(B|C) + (A<B).(B|D) + (A<B).(D<C)$

Non-idempotent Laws

1. $A<A \neq A$

2. $A|A \neq A$

Absorption Laws

1. $A . (A < B) \Leftrightarrow A < B$

2. $A . (A \mid B) \Leftrightarrow A \mid B$

3. $A < (A . B) \Leftrightarrow A < B$

4. $A \mid (A . B) \Leftrightarrow A \mid B$

In the following and to the end of the list of laws, ‘0’ means “never occurs” and ‘1’ means “always occurs, i.e. before anything else occurs (first always)”.

5. $A < (A + B) \Leftrightarrow 0$

6. $A \mid (A + B) \Leftrightarrow 0$

7. $(A . B) < A \Leftrightarrow 0$

8. $(A . B) \mid A \Leftrightarrow 0$

9. $(A + B) < A \Leftrightarrow B < A$

10. $(A + B) \mid A \Leftrightarrow B \mid A$

11. $A + (A < B) \Leftrightarrow A$

12. $A + (A \mid B) \Leftrightarrow A$

13. $B + (A < B) \Leftrightarrow B$

14. $B + (A \mid B) \Leftrightarrow A + B$

15. $A<B . A|C \Leftrightarrow A<C<B + A<B<C + A<B . A|C$

16. $A|B \Leftrightarrow A<B + A|B$

17. $A|B|C \Leftrightarrow A<B<C + A<C<B + A<B . A|C + A<C . A|B + A|B|C$

18. $A<B|C \Leftrightarrow A<B<C + A<B . B|C$

<u>Completion Laws</u>

1.	First Completion Law:	$A.B \Leftrightarrow A<B + B<A$
2.	Second Completion Law:	$A+B \Leftrightarrow A|B + B|A$
3.	Third Completion Law:	$A \Leftrightarrow B<A + A|B$

The below are further laws of completion for the purpose of reducing temporal expressions, i.e., RCL (abbreviation of Reduction Completion Law):

4. $A \Leftrightarrow A.B + A|B$

5. $A \Leftrightarrow (A<B + B<A) + A|B$

6. $A \Leftrightarrow B<C<A + C<B<A + A|B|C + C<A|B + B<A|C$

7. $A \Leftrightarrow A.B + A.C + A|B + A|C$

8. $A<B \Leftrightarrow C<A<B + A<C<B + A<B<C + A<B \,.\, B|C$

9. $A \Leftrightarrow B<A + C<A + B<C<A + C<B<A + A<B \,.\, A|C + A<C \,.\, A|B +$
   $A|B|C + B<A<C + C<A<B + A<B<C + A<C<B$

Mutual Exclusion Laws

1. $A<B \,.\, B<A \Leftrightarrow 0$
2. $A|B \,.\, B<A \Leftrightarrow 0$
3. $B|A \,.\, A<B \Leftrightarrow 0$

Simultaneity Laws

1. $A<A \Leftrightarrow 0$
2. $A|A \Leftrightarrow 0$

Extension Laws

1. $A<B \,.\, B<C \Leftrightarrow A<B \,.\, B<C \,.\, A<C \Leftrightarrow A<B<C$
2. $A|B \,.\, B|C \Leftrightarrow A|B \,.\, B|C \,.\, A|C$
3. $A<B \,.\, B<C \,.\, C<A \Leftrightarrow A<C \,.\, B<C \,.\, C<A \,.\, A<C \,.\, B<A \,.\, C<A$

Extended Laws of Extension

1. $C<A \,.\, B<C \Leftrightarrow C<A \,.\, B<C \,.\, B<A$
2. $C|A \,.\, B|C \Leftrightarrow C|A \,.\, B|C \,.\, B|A$
3. $A<B \,.\, B|C \Leftrightarrow A<B \,.\, B|C \,.\, A|C$
4. $A|B \,.\, B<C \Leftrightarrow A|B \,.\, B<C \,.\, A<C$

5. $C<A . B|C \Leftrightarrow C<A . B|C . B<A$

6. $C|A . B<C \Leftrightarrow C|A . B<C . B|A$

POR Transformation Laws

1. $A|B . B \Leftrightarrow A<B$

2. $A|B + B \Leftrightarrow A + B$

3. $A<B . A \Leftrightarrow A<B$

4. $A<B . B \Leftrightarrow A<B$

5. $A<B + A \Leftrightarrow A$

6. $A<B + B \Leftrightarrow B$

7. $A|B + B \Leftrightarrow A + B$

8. $A|B.C + B \Leftrightarrow A.C + B$

9. $A|B|C + B \Leftrightarrow A|C + B$

Priority Laws

1. $A<B + A|B \Leftrightarrow A|B$

2. $A<B . A|B \Leftrightarrow A<B$

3. $A<B + A.B \Leftrightarrow A.B$

4. $A|B + A.B \Leftrightarrow A$[75]

5. $A<B . A|B \Leftrightarrow A<B$[76]

[75] $A|B + A.B \Leftrightarrow A|B + A<B + B<A$ (by Conjunctive Completion Law) $\Leftrightarrow A + A<B$ (by Reductive Completion Law) $\Leftrightarrow A$ (by Absorption Law)

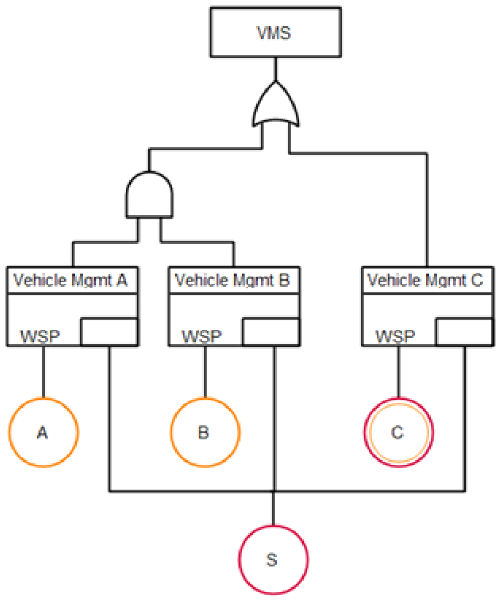
Tautology and Contradiction Laws

1. $A<0 \Leftrightarrow 0$
2. $0<A \Leftrightarrow 0$
3. $A<1 \Leftrightarrow 0$
4. $1<A \Leftrightarrow A$
5. $A|0 \Leftrightarrow A$
6. $0|A \Leftrightarrow 0$
7. $A|1 \Leftrightarrow 0$
8. $1|A \Leftrightarrow 1$

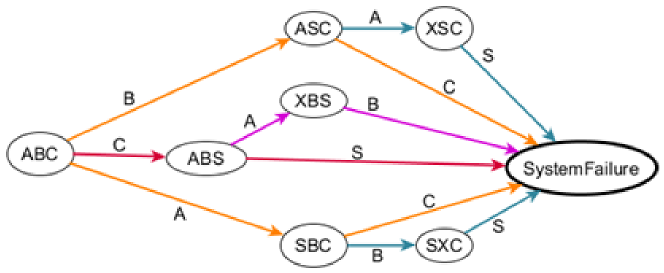
3.3 Case study

In this section, we show how the temporal laws that we have presented are useful by applying them to the vehicle management system (VMS) example of chapter 1. To summarise, the system operates with three vehicle management components (A, B and C) and its failure behaviour expressed by the DFT (Figure 3—4 a) is such that either the spare gate corresponding to the component C is true or the spare gates corresponding to the components A and B are both true to cause the VMS top-event (i.e., the system failure).

[76] $A<B \, . \, A|B \Leftrightarrow A<B \, . \, A<B$ (by POR Transformation) $\Leftrightarrow A<B$ (by Idempotent: $A.A \Leftrightarrow A$)



(a) the VMS DFT



(b) the failure behaviour SM of the VMS

Total failure sequences

(1) **Both S and C fail**

(2) **A fails before C**

(3) **B fails before C**

(4) (C fails before A) **and** (C fails before B)

(5) A, B and S **all** fail

(c) the SystemFailure cut-sequences

Figure 3—4. The VMS DFT, SM and system failure TFTs

The spare S is shared among all components, and a spare gate is true if the corresponding component fails and cannot be replaced by the spare. This can happen because either the spare itself is failed or is unavailable — i.e., it is replacing another failed component.

Figure 3—4 (b) shows the SM which represents the system failure behaviour (as explained in chapter 1). Briefly, the state names use the notation based on the succession "ABC". The initial state ABC" means that all components are working. Thereafter, the position of any failed component in the succession will take either the symbol 'S' or 'X' — 'S' means that the failed component has been replaced by the spare and 'X' means that it cannot be replaced by the spare for the reasons mentioned earlier. As for the transitions, these are labelled with the component symbols to indicate their corresponding failures.

In the next chapter, we will present an algorithm for the automatic generation of TFTs from SMs. That algorithm will generate the following six cut-sequences from the SM of the VMS. To briefly explain how we obtained these cut-sequences; there is a corresponding Pandora formula for each final state, a formula is the disjunction over the paths π (from the initial state to its corresponding final state) of the conjunction of events that label π. Moreover, any one of the events e that label π, which is incident from a state u with an out-degree of two or more (a state at which paths diverge), is the input event, which must occur first or alone, of a POR gate that associates it with the disjunction of other events e' incident from u to the states u' if, and only if, the subpath of π from u to s shares an event with the paths from u through u' to any state. In this section, we put emphasis on the minimisation of these cut-sequences (not the full conversion algorithm) in order to reach their corresponding reduced forms, which are described by Figure 3—4 (c). We will use some of the temporal laws presented in this chapter to achieve these results.

"SystemFailure" =

B.A.C|A|B + (1)

C.A|C|B + (2)

S.C|A|B + (3)

S.A.B|C|A + (4)

C.B|C|A + (5)

S.B.A|C|B (6)

By using POR transformation[77] #1 to all cut-sequences, we obtain the following transformed equivalent results:

"SystemFailure" =

C<A . C<B + (1)

[77] i.e., the first minimisation step which reduces a representation using two gates 'POR' and 'AND' to a representation using one gate only 'PAND', i.e. A|B . B $\Leftrightarrow$ A<B.

A<C . A|B +	(2)
S.C|A|B +	(3)
S . B<A . B|C +	(4)
B<C . B|A +	(5)
S . A<B . A|C	(6)

Then we apply the law of absorption #15 to cut-sequences (2) and (5) to get the following results, where (2.1), (2.2) and (2.3) are introduced by the transformation of (2) and (5.1), (5.2) and (5.3) are introduced by the transformation of (5):

"SystemFailure" =

C<A . C<B +	(1)
A<B<C +	(2.1)
A<C<B +	(2.2)
A<C . A|B +	(2.3)
S.C|A|B +	(3)
S . B<A . B|C +	(4)
B<A<C+	(5.1)
B<C<A	(5.2)
B<C . B|A	(5.3)
S . A<B . A|C	(6)

Next, we apply the completion law #8 on (2.1), (2.2), (2.3) and (5.1) for the purpose to reduce all these to simply A<C and we refer to it as (2_5), i.e., it originates from the previous cut-sequence #2, plus a branch from the cut-sequence #5. Similarly, we apply the same reduction completion law (RCL #8) on (5.1), (5.2), (5.3) and (2.1) to reduce them to B<C and we refer to it as (5_2) for the same reason as the previous one, i.e.,

it originates from the previous cut-sequence #5, plus a branch from the cut-sequence #2. Therefore, by using the Boolean idempotent law ($A + A \Leftrightarrow A$) we have the following:

“SystemFailure” =

$C<A \;.\; C<B\;+$	(1)
$A<B<C\;+$	(2.1)
$A<C<B\;+$	(2.2)
$A<C \;.\; A \vert B\;+$	(2.3)
$B<A<C+$	(5.1)
$S.C \vert A \vert B\;+$	(3)
$S \;.\; B<A \;.\; B \vert C\;+$	(4)
$B<A<C+$	(5.1)
$B<C<A$	(5.2)
$B<C \;.\; B \vert A$	(5.3)
$A<B<C\;+$	(2.1)
$S \;.\; A<B \;.\; A \vert C$	(6)

Then the application of RCL #8 gives:

“SystemFailure” =

$C<A \;.\; C<B\;+$	(1)
$A<C\;+$	(2_5)
$S.C \vert A \vert B\;+$	(3)
$S \;.\; B<A \;.\; B \vert C\;+$	(4)
$B<C\;+$	(5_2)

S . A<B . A|C (6)

By application of the absorption law #15 on both (4) and (6) we will get:

“SystemFailure” =

C<A . C<B + (1)

A<C + (2_5)

S.C|A|B + (3)

S . B<A . B|C + (4.1)

S . B<C<A + (4.2)

S . B<A<C + (4.3)

B<C + (5_2)

S . A<B . A|C + (6.1)

S . A<C<B + (6.2)

S . A<B<C (6.3)

We can also apply the 3-event absorption law (i.e., law #17) on cut-sequence (3), then the Boolean distributive law “A . (B + C) $\Leftrightarrow$ A . B + A . C” on the obtained sub-branches to get the following:

“SystemFailure” =

C<A . C<B + (1)

A<C + (2_5)

S . (C<A<B + (3.1)

C<B<A + (3.2)

C<A . C|B + (3.3)

C<B .C|A + (3.4)

C|A|B) (3.5)

S . B<A . B|C + (4.1)

S . B<C<A + (4.2)

S . B<A<C + (4.3)

B<C + (5_2)

S . A<B . A|C + (6.1)

S . A<C<B + (6.2)

S . A<B<C (6.3)

Next, we associate all of (3.1), (3.2), (3.3), (3.4), (3.5), (4.1), (4.2), (4.3), (6.1), (6.2) and (6.3) by using the Boolean distributive law “A . (B + C) $\Leftrightarrow$ A . B + A . C”. Therefore, we have:

“SystemFailure” =

C<A . C<B + (1)

A<C + (2_5)

S . (C<A<B + (3.1)

C<B<A + (3.2)

C<A . C|B + (3.3)

C<B .C|A + (3.4)

C|A|B + (3.5)

B<A . B|C + (4.1)

B<C<A + (4.2)

B<A<C + (4.3)

A<B . A|C + (6.1)

A<C<B +	(6.2)
A<B<C)	(6.3)
B<C +	(5_2)

We also obtain the following by applying the Boolean absorption law " A + A.B $\Leftrightarrow$ A" on (2_5) and (5_2):

"SystemFailure" =

C<A . C<B +	(1)
S . A<C +	(2_5.1)
A<C +	(2_5.2)
S . (C<A<B +	(3.1)
C<B<A +	(3.2)
C<A . C|B +	(3.3)
C<B .C|A +	(3.4)
C|A|B +	(3.5)
B<A . B|C +	(4.1)
B<C<A +	(4.2)
B<A<C +	(4.3)
A<B . A|C +	(6.1)
A<C<B +	(6.2)
A<B<C) +	(6.3)
S . B<C +	(5_2.1)
B<C	(5_2.2)

(2$_5$.1) and (5$_2$.1) can be associated with (3.1), (3.2), (3.3), (3.4), (3.5), (4.1), (4.2), (4.3), (6.1), (6.2) and (6.3) by using the Boolean distributive law "A . (B + C) $\Leftrightarrow$ A . B + A . C", and hence:

"SystemFailure" =

C<A . C<B +	(1)
A<C +	(2$_5$.2)
B<C +	(5$_2$.2)
S . (	
A<C +	(2$_5$.1)
B<C +	(5$_2$.1)
C<A<B +	(3.1)
C<B<A +	(3.2)
C<A . C|B +	(3.3)
C<B .C|A +	(3.4)
C|A|B +	(3.5)
B<A . B|C +	(4.1)
B<C<A +	(4.2)
B<A<C +	(4.3)
A<B . A|C +	(6.1)
A<C<B +	(6.2)
A<B<C	(6.3)
)	

In the next step, we apply the Boolean Algebra Idempotence law of the logical disjunction on (3,2), (4.2), (4.3), (3,1), (6.2) and (6.3). This is to get them duplicated for the purpose of:

a) Reducing (2_5.1), (5_2.1) (3.1), (3.2), (3.3), (3.4), (3.5), (4.2), (6.2), (4.3) and (6.3) to C by using the 3-event RCL #9.

b) Reducing (3,2), (4.1), (4.2) and (4.3) to B<A by using the RCL #8.

c) Reducing (3,1), (6.1), (6.2) and (6.3) to A<B by using the RCL #8.

The application of the disjunction Idempotence law gives the following:

"SystemFailure" =

C<A . C<B +	(1)
A<C +	(2_5.2)
B<C +	(5_2.2)
S . (	
A<C +	(2_5.1)
B<C +	(5_2.1)
C<A<B +	(3.1)
C<B<A +	(3.2)
C<A . C|B +	(3.3)
C<B .C|A +	(3.4)
C|A|B +	(3.5)
B<C<A +	(4.2)
C<B<A +	(3.2)
B<A . B|C +	(4.1)

B<C<A +	(4.2)
B<A<C +	(4.3)
C<A<B +	(3.1)
A<B . A|C +	(6.1)
A<C<B +	(6.2)
B<A<C +	(4.3)
A<B<C +	(6.3)
A<C<B +	(6.2)
A<B<C	(6.3)

)

Then, by application of the reductions a), b) and c) mentioned earlier, we have:

“SystemFailure” = C<A . C<B + A<C + B<C + S . (**C** + **B<A** + **A<B**)

Finally (and by application of RCL #1, i.e., A<B + B<A $\Leftrightarrow$ A . B) we have the following:

“SystemFailure” = C<A . C<B + A<C + B<C + S . (C + A . B)

These are the cut-sequences described by Figure 3—4 (c).

3.4 Discussion

In this chapter, we have presented an overview of a recent temporal qualitative analysis technique so-called Pandora. The technique not only extends standard fault trees with temporal information modelling capabilities, but also provides a set of temporal laws besides the Boolean logic laws. These are useful to preserve the significance of the temporal semantics all along the logical reduction of the fault trees, and as demonstrated through the case study example in section 3.3.

We have highlighted three temporal gates introduced by Pandora: PAND, POR and SAND. The latter (which is conceived to model the simultaneity of events) has been removed and the original temporal laws were adapted correspondingly. The main reason which has driven the alteration of the original temporal laws is our use of SMs (which describe the failure behaviour of systems) to generate the fault trees. The generated fault trees are, therefore, in all cases (whether static or dynamic) without any SAND gate.

Representation of simultaneity (at the level of the SMs) is useful to be made if we assume common failure modes by which events may occur at the same time. Such an assumption was made in the second case study of this thesis (see chapter 6 about the brake-by-wire system) to model the dynamics of the vehicle (section 6.1).

4 Automatic Generation of Temporal Fault Trees from State Machine Models of Systems

4.1 Introduction:

State machines are an expressive, high-level form of modelling the behaviour of systems. They readily express the different effects of events on a system in different states, making them well-suited to modelling the effect of failure and fault events on a system. We observed in the literature chapter how the use of SMs is becoming prevalent and particularly important for modelling dynamic systems — like for example in (and among other modelling approaches) Altarica [(Rauzy, 2002), (Griffault A. , 2003) and (Bernard R. , 2009)], FSAP/NuSMV-SA [(Bozzano *et al.*, 2003) and (Bozzano *et al.*, 2006)], representation of SEFTs [(Kaiser *et al.*, 2007) and (Grunske *et al.*, 2005)], analysis of DFTs and SEFTs using Markov chains [(Dehlinger *et al.*, 2008), (Dugan *et al.*, 1992), (Dugan *et al.*, 1997), (Dugan *et al.*, 2000), (Kaiser *et al.*, 2007) and (Vesely *et al.*, 2002)] and, most recently, AADL [(Feiler *et al.*, 2007), (Joshi *et al.*, 2007), (Rugina A., 2007) and (Rugina *et al.*, 2008)]. HiP-HOPS also defines an extension with state automata modelling capabilities as described in (Papadopoulos Y., 2000).

We have discussed two different approaches to perform dependability analysis on dynamic systems. The first approach consists of converting the SMs to GSPNs as with AADL error models[78] [(Rugina A. , 2007) and (Rugina *et al.*, 2008)]. Yet this approach is less suitable for qualitative analysis[79]. An alternative approach involves conversion of state machines to combinatorial fault trees, like in (Joshi *et al*, 2007) with AADL and in (Rauzy, 2002) with Altarica. The problem highlighted with this type of conversion consists, in particular, of the loss of the temporal semantics expressed by state machines during their translation to combinatorial (i.e., static) fault trees — these have no concept of event order or sequence, and this can potentially cause serious errors.

[78] These are effectively SMs (showing transitions from normal to degraded and failed states) as observed in (Feiler *et al.*, 2007).

[79] I.e., establishment of direct relationships between causes and effects of failure, as in FMEA. Qualitative analysis is particularly important when probabilistic data are not available, e.g. at early stages of design.

In (Rauzy, 2002), this problem emerging from the conversion of SMs to combinatorial FTs was noted when a conjunction of two mutually exclusive paths through the SM for a k-out-of-n system both became results of the fault tree analysis. This was solved by incorporating NOT gates into the conversion to introduce a partial order, so that different paths through the SM could be distinguished by indicating that some events did not occur. However, although this will work in certain cases (e.g. to prevent the result (A AND B) OR (A AND C) OR (B AND C) when A and B are mutually exclusive), it still cannot distinguish SM paths that differ only in sequence. If, for example, the state machine defines that different sequences of faults, e.g. A before B and B before A, lead the system into two mutually exclusive failure states, the fault trees for these two states will show the combination of A and B as a common cause of both states. This result is logically and probabilistically incorrect, and will thus lead to incorrect conclusions regarding the dependability of the system.

This approach of converting SMs to combinatorial fault trees for dependability analysis is being used in conjunction with influential modelling languages including AADL and Altarica. The former is gaining growing acceptance in the aerospace community and the latter was already used together with the conversion algorithm described in (Rauzy, 2002) in an industrial context. For example, this algoritnm (which compiles mode automata into combinatorial FTs) was used as part of the OCAS tool which has been developped by Dassault Aviation in a certification process of its Falcon 7X flight controls. Thereafter, the Aralia tool — described in (Dutuit *et al.*, 1997) — was used to handle the analysis of the generated FTs from Altarica models (both qualitatively and quantitatively). Besides, the Altarica's modelling and analysis platform — a tool which uses computer-performed generation of fault trees — was qualified as a validation tool in several aerospace projects, including Airbus civil aircraft programs (Bernard R. , 2009).

The potential for erroneous results arising from application of this approach in a subset of SMs must therefore be addressed. To correct the conceptual flaw discussed above and allow true qualitative temporal safety analysis, in this chapter, we outline our technique to enable conversion of SMs to temporal FTs as opposed to static FTs, thereby preserving the significance of the sequencing of faults. The fault trees can then be analysed using Pandora (see chapter 3) to enable qualitative analysis of the generated TFTs. The approach is generally applicable to error models expressed as SMs, including the AADL error model. It can also in principle be combined with a compositional analysis technique like HiP-HOPS (Papadopoulos Y. , 2000), which would enable automatic synthesis and analysis of temporal fault trees from complex models where the error behaviour of the system has been described by SMs.

Our aim in this chapter is, precisely, to propose a new approach for safety analysis of dynamic systems with SM-based failure behavioural models. The core of our approach consists of the conversion of the SMs to dynamic analysis models (i.e., temporal FTs). These will be consequently analysed using a temporal qualitative analysis technique like Pandora. The significance of the temporal semantics expressed by SMs will be, therefore, preserved during the conversion to TFTs, and along the logical reduction of these TFTs by using the temporal laws of Pandora.

We use the primary standby (PS) example system — described in chapter 1 — to demonstrate how the proposed method has helped us analyse the system, and therefore correct the analysis results produced using a conversion of the system's SM to static models (like pure combinatorial FTs), which limits any consequent analysis to static techniques like FTA. The PS example system will also serve in the following chapter to be scaled up from a single to a triple redundancy. The example is, therefore, useful not only to show how we can reuse some of the failure behavioural models in the analysis of larger systems; but it will also serve, mainly and most importantly, to propose a compositional approach for our technique and to demonstrate how it can contribute to an improvement in the overall safety of increasingly complex modern safety-critical systems.

Figure 4—1 captures in a single view the PS example system in (a), the SM which represents its failure behaviour in (b) and its corresponding FT in (c) — where A, B and S represent abbreviations of 'A fails', 'B fails' and 'S fails' respectively. Briefly, the system (as explained in chapter 1) operates either in primary mode (i.e., with component A) or standby mode (i.e., with component B). The sensor S serves to activate B upon detection of an output deviation (e.g. omission of output) from A.

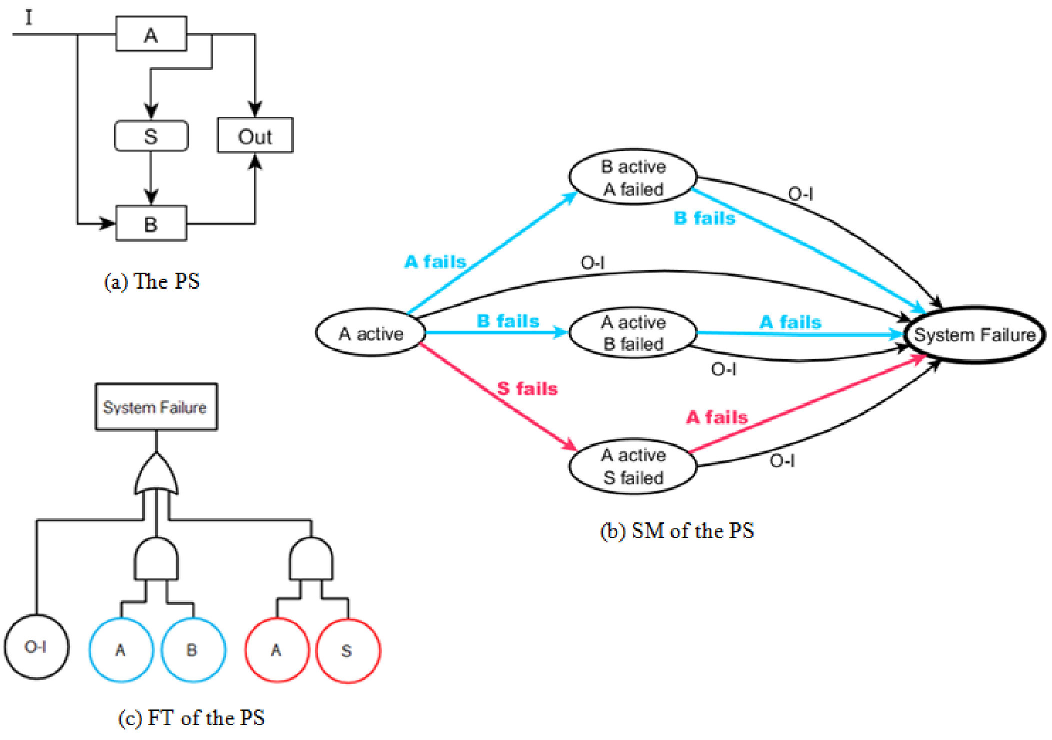


Figure 4—1. (a) The PS example system. (b) SM failure behavioural model of the PS. (c) FT of the PS.

'I' is the input to the system and 'Out' is an abstraction of the output, which we omit to consider in the analysis for the sake of clarity. The SM in (b) shows that an omission of input 'O-I' is a single point of failure of the system, failures of A and B together will lead the system to a total failure (irrespective of the order of the two failures); but about A and S, only a failure of S preceding that one of A is relevant to the system failure behaviour — if this happens in this order, then B cannot be activated upon failure of A; otherwise, the system can still continue operating in standby mode, i.e., A failing first will cause the SM to transit to the top state[80] (B active and A failed, i.e. the system is only degraded), and thus a subsequent failure of S will have no impact on the system (i.e., no transition). This SM clearly shows how the order in which the same two events occur can affect the overall outcome — the system, as a result, will be either degraded or totally failed depending on which one has failed first.

[80] In this case, S has already detected that something went wrong with A and has, therefore, activated B before failing.

Different sequences of the same events causing different effects is not an uncommon pattern of events; many safety-critical systems feature the ability to continue operating with a subset of functionality, e.g. some aeroplanes can continue to fly even after an engine failure. In this case, one event may be a failure of the fire extinguishing system and the other one an engine fire. If the fire extinguishing system fails first, the fire cannot be extinguished and may spread, whereas if an engine fire occurs first, the fire can be extinguished and the aeroplane can continue to fly, albeit in a degraded state.

By transforming a SM to FTs, each of the system failures (i.e., final states of the SM) would become the head (i.e., top event) of their own fault tree, with the root causes beneath. All paths through the SM to a given final state become a disjunction of possible causes of the corresponding top event, and each cause is represented as a conjunction of all failure events in that path which have caused the system to enter the final state. Each conjunction represents one cut set of the fault tree, and if it contains no redundancies, then it is a minimal cut set (MCS). By applying this to the SM of Figure 4—1 (b), we would obtain 'System Failure' as top event of the fault tree as follows and where '**.**' and '+' represent the logical gates 'AND' and 'OR' respectively:

System Failure = O-I**.**S + A**.**S + O-I**.**B + A**.**B + O-I + B**.**A + O-I**.**A

The SM has seven different paths leading to a unique final state (System Failure), and hence the seven cut sets of the unique fault tree expression above. This fault tree expression can be logically reduced according to the absorption law X + X.Y = X, thus we would obtain:

System Failure = A**.**S + A**.**B + O-I + B**.**A

The idempotent law X + X = X would help us to further reduce the expression to become:

System Failure = A**.**S + A**.**B + O-I.

Hence the minimised fault tree of Figure 4—1 (c).

Clearly, by transforming the SM of the PS to a combinatorial fault tree, we would obtain the conjunction of the failures of A and S as a cause of the total failure of the system. This, however, is not accurate and invalidates application of this approach to this and other cases where the temporal ordering of failures is significant — the results are not an accurate representation of the behaviour shown in the SM. Note that any consequent quantitative analysis would also be incorrect. Thus, this kind of transformation can potentially lead to serious errors in the analysis. Therefore, we need firstly to compile

SMs into FTs that have representation capabilities of concept of time or event sequencing, then secondly to be able to preserve the significance of the temporal semantics and event sequencing these new FTs contain along their logical reduction.

4.2 Compilation of SMs into TFTs

Our approach consists of the conversion of SMs (which are dynamic models) to temporal FTs (which are also dynamic) rather pure combinatorial FTs (which are static). We observed in the literature chapter different approaches which extend FTs with temporal capabilities for modelling sequence-dependent failures (a situation frequently exhibited by dynamic systems). Therefore, we start this section by presenting the reasons which have influenced our initial choice concerning the TFT approach targeted by the transformation from the SMs.

4.2.1 The Pandora choice

It is necessary but not sufficient to preserve the significance of the temporal semantics expressed by the SMs during their transformation to the corresponding FTs. The generated FTs (with temporal information if necessary) need to be logically analysed. If these are not already in their minimal form, then a logical reduction is needed. It also necessary to preserve the significance of the temporal semantics (transferred from the SMs to the generated FTs) during all the reduction process.

The approaches to dynamic FTs observed in chapter 2 focus primarily on quantitative analysis, like SEFTs (Kaiser *et al.*, 2007) and DFTs (Vesely *et al.*, 2002) which are solved with Markov chains. The approach in (Merle, 2010), however, attempts to qualitatively analyse dynamic FTs by determining minimal canonical forms of the structure functions corresponding to the FTs. However, a minimal canonical form is not necessarily unique[81] for a structure function of a dynamic FT and the complexity involved was beyond the scope of the work in (Merle, 2010). TFT minimisation itself is also beyond the scope of work of this doctoral thesis.

[81] There are typically as many minimal canonical forms of the structure function as there are minimization criteria.

Pandora (chapter 3) was designed mainly for temporal qualitative analysis. It not only extends FTs with its three temporal gates (PAND, POR and SAND), but also provides sets of temporal laws which complete the Boolean laws — like (among others) temporal commutative, associative and distributive laws, also completion, absorption and reductive laws . Briefly (and as seen in chapter 3), Pandora provides two different approaches to the reduction of its TFTs. One of these relies on temporal logic (i.e., logical reduction through redundancy, contradiction and completion); the other approach is a hierarchical reduction technique which uses dependency trees to model relationships of events.

For the purpose of our study, we use a variation of Pandora where the SAND gate is omitted. We avoid non-determinism in the state machine by assuming a common failure mode by which two (or more) failures occur simultaneously. This assumption was also observed being used in other state-transition based modelling techniques like Markov chains in for instance (ARP4761, 1996), and where a simultaneous failure of two or more components was given one failure rate. Consequently, the temporal laws identified in chapter 3 to be used for the logical reduction of the generated FTs are variations (without SAND operators) of their corresponding Pandora laws. Temporal expressions are generally complex, particularly when compared with Boolean ones. Therefore, the temporal laws are very useful in order to reduce the complexity involved through minimisation of the generated temporal expressions.

4.2.2 Representing SMs using Pandora logic

A state machine will have as many fault trees as there are final states. Each final state represents a system failure (i.e., one fault tree top-event) and each transition between states represents an event in a fault tree. Every full path between the initial state and a given final state becomes a new branch of its corresponding FT — i.e., represented by the conjunction of the events that label that path. In the case where some system failures are sequence-dependent, then the corresponding FTs should be temporal. Thus, not all SMs require the use of temporal gates when transformed to fault trees. In many cases, standard Boolean logic is adequate to represent the behaviour and perform a correct analysis; for example a SM where two events e.g. A then B lead to a ‘Degraded’ state and two other different events e.g. C then D lead to a ‘Failed’ state as in Figure 4—2.

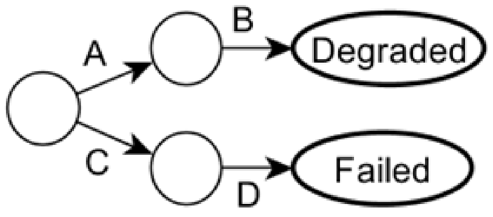


Figure 4—2. SM example of a static system.

In this case, we are concerned with separate combinations of events, i.e., the two branches have no events in common and each event contributes only to a single end state (and ultimately, contributes to only one top event). Here, a change in the sequence of the events (e.g. D before C instead of C before D) will not lead to a different failure. In the case that D happens first, the system simply stays in the initial state and when C occurs it performs two instantaneous transitions to reach the final failed state; thus the failure behaviour is not sequence-dependent as in the case of the SM of the PS in Figure 4—1 (b). Therefore, in this scenario, Boolean logic is sufficient to model the situation unambiguously (Degraded = A.B and Failed = C.D) and thus the simpler transformation of SMs into fault trees described in (Joshi *et al.*, 2007) is sufficient to obtain an accurate analysis.

Imposing temporal constraints during the conversion depends on whether or not the SM has at least one event appearing in more than one path. Typically, if there is at least one event that contributes to the occurrence of more than one system failure, then conversion to temporal fault trees may be needed. It may also be true even if there is an event that is a contributory factor to the occurrence of only one system failure, but as a result of more than one sequence of events. In such cases, an accurate analysis depends upon the correct preservation of the temporal semantics, as different sequences of those shared events – or other events relative to those shared events – may lead to different final states (and thus different system failures). The application of our approach allows complex dynamic analysis to be applied only when necessary, depending on whether each part of the system is static or dynamic.

4.2.3 Pandora description of the PS state machine

Figure 4—1 (b) presents a more detailed view of the failure behaviour of the PS example from Figure 4—1 (a). Note that two paths for a combination A and B have been shown

here for clarity, though as will be seen this does not affect the results. Likewise, the state machine shows that the final state "System Failure" is dependent on different sequences of events. For example, the sequence in which S failing before A can lead to system failure is shown, but the failure of S after A does not lead to system failure (as it has no effect). Other paths that are clearly redundant, such as failures of A then S then B, are omitted.

This type of sequence-dependent failure behaviour is clearly visible from the state machine, but cannot easily be analysed in this form. We could convert it to a combinatorial fault tree, but as explained earlier, the temporal semantics would be lost. Instead, we must convert it into a Pandora temporal fault tree. The first step in the process is to identify shared events, i.e. events that are present in more than one path through the state machine. For example, the event "A fails" contributes to multiple paths through this SM: "A fails → B fails", "B fails → A fails" and "S fails → A fails". These sequences of events are all different causes of the same system failure, and since fault trees do not represent states explicitly, only events, it is necessary to use a temporal fault tree to represent this type of failure behaviour.

To see how the sequence of events affects the conversion process, let us consider three paths, each sufficient on its own to cause the system failure:

Path 1:	A fails → B fails
Path 2:	B fails → A fails
Path 3:	S fails → A fails

The event 'A fails' (which we shall abbreviate to just 'A', and similarly 'B' = 'B fails', 'S' = 'S fails', and 'I' = 'O-I') influences the state of the system in all three paths. However, which path is taken from the initial state depends on whether A, B, or S occurs first, assuming no omission of input occurs before. For example, in the case that A occurs first, we are already in path 1 and S does not influence the state of the system; only a subsequent failure of B (or O-I) leads to system failure. However, if S occurs first (leading us into path 3) then a subsequent failure of A is sufficient to cause system failure.

The only way to accurately model this in a fault tree is to use temporal operators. In particular, we want to ensure that the relative ordering of each event that leaves the initial state (e.g. in this case, A, B, and S each leads to a different path depending on which occurs first) is explicitly represented. The main sequence operator in Pandora is the

PAND gate, and we can use this to unambiguously distinguish between a path in which A occurs first (i.e. path 1) and a path in which S occurs first (i.e. path 3):

Path 1: A < B

Path 3 S < A

However, the situation is more complex than it first appears, as we also want to express priority – the idea that one event must occur first – without also implying that another event must occur. For example, to distinguish between path 2 and path 3, we need to know whether S or B occurs first, but S and B are not shared between these paths – only A is. Thus to distinguish path 3 from path 2, we need to say that S occurred before B, but only if B occurs; B does not have to occur for path 3 to be completed leading to system failure.

To do this, we can take advantage of Pandora's POR gate. The POR gate represents this concept of priority – that one event should take priority over others and must occur first, but without specifying that the other events must also occur. Thus we can represent the three paths as follows:

Path 1: (A < B) . (A | S)

Path 2: (B < A) . (B | S)

Path 3: (S < A) . (S | B)

Thus path 1 specifies two constraints: firstly that both A and B must occur, with A occurring first, and secondly that if S also occurs at all, it must occur after A. The second constraint represents a situation where S has already served its purpose and activated the backup, in which case its subsequent failure has no effect. Thus, in path 1, both A and B must occur for the system to fail as a whole. The PAND gate alone would not work in this type of situation as it would mean that all three events must occur. Only the POR gate correctly represents the temporal semantics and allows us to preserve the sequences of events in the state machine.

Using two operators like this enables us to accurately represent the temporal semantics of the SM, but it does lead to more complex expressions and thus a more complex translation process. By using Pandora's temporal laws, it is possible to simplify the generation of these expressions so that they require only one operator – the POR. The law A<B <=> (A|B).B means that any expression containing a PAND can be converted to use a POR instead, e.g. for Path 1:

(A < B) . (A | S) <=> (A | B) . B . (A | S) <=>

(A | B | S) . B <=> A | B | S . B

A second law was also applied here: A|B . A|C <=> A|B|C. Note that A|B|C is not equivalent to A|(B|C); after the first event, the others can occur in any order, so A|B|C <=> A|C|B. Thus we can represent the three paths as follows:

Path 1:	A | B | S . B
Path 2:	B | A | S . A
Path 3:	S | A | B . A

This form allows us to see both the temporal constraints (represented by the PORs) and the purely combinatorial constraints (represented by ordinary conjunctions).

4.2.4 Automatic generation of Pandora formulae

Informally, normalised graphical notations, e.g. Figure 4—1 (b), describe SMs of systems, subsystems or components. States are represented by ellipses or circles and events are represented by arrows that join states. Each ellipse or circle contains a textual description of its corresponding state. Similarly, arrows are labelled with textual descriptions of the corresponding events.

A state machine has a finite number of states. It may change state when an event occurs, but at each instant it is in only one state. Definition 1 formally describes a state machine.

Definition 1: ***State Machine (SM)***
A state machine is a quadruple (S, Σ, δ, s_0) where:

- *S is a finite set of states.*
- *Σ is a finite set of events, such that $S \cap \Sigma = \emptyset$.*
- *δ is a partial function: $S \times \Sigma \longrightarrow S$ s.t. for $(u, u') \in S^2$ and $e \in \Sigma$, $u' = \delta(u, e)$ iff e is incident from u to u', and we write it as: $u \xrightarrow{e} u'$.*
- *s_0 is the initial state.*

For example, consider again Figure 4—1 (b). The SM that formally describes the PS is defined as follows:

- S = {A active, B active A failed, A active B failed, A active S failed, System Failure}.
- Σ = {A fails, B fails, S fails, O-I}.
- δ is defined as shown in the SM figure.
- s_0 = A active.

For the purpose of this study, we assume a state machine to be acyclic. A cyclic SM may imply that failures are repairable or repeatable, which is incompatible with the semantics of Pandora. There is, therefore, a finite set of possible paths in the state machine. If π is a path from u to u' (u, u') $\in S^2$, we write it as $u \overset{\pi}{\rightsquigarrow} u'$.

Definition 2: ***Paths set***
Let P be the set of all paths in the SM,
$P = \{\pi \mid u \overset{\pi}{\rightsquigarrow} u', (u, u') \in S^2\}$

We write $u \rightsquigarrow u'$ iff $\exists \pi \in P$ s.t. $u \overset{\pi}{\rightsquigarrow} u'$. In such a case, state u' is said to be reachable from state u. However, if there exists a one-event path from u to u', then u' is said to be immediately reachable from u and we write it as $u \rightarrow u'$ (i.e., $u \rightarrow u'$ iff $\exists e \in \Sigma$ s.t. $u \xrightarrow{e} u'$).

We assume that from s_0 we can reach any other state, i.e.,$\forall t_{\neq s0} \in S$ $s_0 \rightsquigarrow t$. Also, $\forall \pi \in P$ $\exists (u, u') \in S^2$ s.t. $u \overset{\pi}{\rightsquigarrow} u'$ and $Seq_\pi = \langle u_0, u_1, \ldots, u_n \rangle$ is the sequence of the states of the path π, where n = length(π) is the number of events that label π and the sequence is ordered for backward traversal, i.e., u_0=u' and u_n=u.

A state machine path can be traversed forwards or backwards. The Pandora formula which corresponds to a final state is generated by performing backward traversals of all paths starting from that final state to the initial state. Forward traversals are performed starting from every 'join' state (i.e., with an out-degree strictly greater than 1) to, at worst, all reachable final states, or until the condition for imposing a temporal constraint is satisfied.

Definition 3: ***Forward and backward incidence sets***
For any state $u \in S$, let $\Sigma_{u\blacktriangleright}$ (resp. $\Sigma_{u\blacktriangleleft}$) be the set of events incident from u (resp. incident to u),
$\Sigma_{u\blacktriangleright} = \{e \in \Sigma \mid \exists u' \in S \text{ s.t. } u \xrightarrow{e} u'\}$
$\Sigma_{u\blacktriangleleft} = \{(e, u') \in \Sigma \times S \mid u' \xrightarrow{e} u\}$

The choice of an event in $\Sigma_{u\blacktriangleright}$ uniquely determines the state that the event is incident to. As for $\Sigma_{u\blacktriangleleft}$, an event incident to u needs to be distinguished from every possible identical event also incident to u, but from a different state. This is done by associating each event with the state from which it is incident, and hence the definition 3 above.

All final states are permanent states — there are no events that lead from a final state to any other state.

Definition 4: ***Set of final states***
Let F be the set of the final states,
$F = \{f \in S \mid \Sigma_{f\blacktriangleright} = \emptyset\}$

There is a corresponding Pandora formula for each final state. Let ϕ_s s∈F be the formula for a final state s. ϕ_s is the disjunction over the paths π (from the initial state s_0 to s) of the conjunction of events that label π. Moreover, any one of the events e that label π, which is incident from a state u with an out-degree of two or more (a state at which paths diverge), is the input event, which must occur first or alone, of a POR gate that associates it with the disjunction of other events e' incident from u to the states u' if, and only if, the subpath of π from u to s shares an event with the paths from u through u' to any state.

Algorithm 1 generates a set Φ of Pandora formulae: $\Phi = \{\phi_s \mid s \in F\}$ — one formula ϕ_s for each final state s. These expressions can then be analysed by Pandora. For each join state s_{i_j} during a path traversal (line 11 of the Algorithm), and for each event

$e' \neq e_{i_j} \in \Sigma_{s_{i_j}\blacktriangleright}$ (line 14), a temporal constraint is imposed (line 21) if, and only if, the two subsets Ω (line 12) and Σ' (initially as in line 15) of Σ share an event ($\Sigma' \cap \Omega \neq \emptyset$). Ω represents the set of the events that label the subpath h of the current full path π_i s.t. $s_{i_j} \overset{h}{\rightsquigarrow} s$ (i.e., h is the part of π_i which has already been traversed) and Σ', which is initialised as {e'} (e' as specified in lines 14 and 16), may possibly be further populated as shown in Algorithm 2 — PopulateAt($t = \delta(s_{i_j}, e')$).

```
Algorithm 1 Φ — ( $SM = (S, \Sigma, \delta, s_0)$ )
1: get P from the state machine
2: get F from S
3: let $\Phi = \emptyset$
4: for each $s \in F$ do
5:    get Ps from P where $Ps = \{\pi \in P | \ s_0 \overset{\pi}{\rightsquigarrow} s\}$
6:    let $n = |Ps|$
7:    let $\pi_i \in Ps \ 1 \le i \le n$ /* $s_0 \overset{\pi_i}{\rightsquigarrow} s$ */
8:    let $len_i = length(\pi_i) \ 1 \le i \le n$ /* the number of events of $\pi_i$ */
9:    let $Seq_{\pi_i} = \langle s_{i0}, \ s_{i1}, \ \ldots, \ s_{i len_i} \rangle \ s_{i_j} \in S \ e_{i_j} \in \Sigma \ 1 \le i \le n \ 1 \le j \le len_i$ s.t.
      $s_{i_j} \xrightarrow{e_{i_j}} s_{i_{(j-1)}}$ where $Seq_{\pi_i}$ is ordered from $s_{i0}=s$ to $s_{i len_i}=s_0$ /* $Seq_{\pi_i}$
      $1 \le i \le n$ is the sequence of states of $\pi_i$ ordered for backward traversal*/
10:   let $\phi_s = \bigvee_{1 \le i \le n}(\bigwedge_{1 \le j \le len_i} e_{i_j})$/*$\phi_s$ is the disjunct over paths $\pi$ of the
      conjunct of events of $\pi$*/
11:   for each $s_{i_j}$ in $Seq_{\pi_i}$ $1 \le i \le n$ $1 \le j \le len_i$ s.t. $|\Sigma_{s_{i_j}\blacktriangleright}| > 1$ /* $\Sigma_{s_{i_j}\blacktriangleright}$ is the
      set of events incident from $s_{i_j}$ */ do
12:      let $\Omega = \{e_{i_j}, e_{i_{(j-1)}}, \ldots, e_{i_1}\}$ /* set of events that label subpath h
         of $\pi_i$ s.t. $s_{i_j} \overset{h}{\rightsquigarrow} s$ */
13:      let $e = e_{i_j}$
14:      for each $e' \neq e_{i_j}$ in $\Sigma_{s_{i_j}\blacktriangleright}$ do
15:         let $\Sigma' = \{e'\}$
16:         let $t \in S$ s.t. $s_{i_j} \xrightarrow{e'} t$
17:         if $((\Sigma' \cap \Omega = \emptyset)$ and $(\Sigma_{t\blacktriangleright} \neq \emptyset))$ then
18:            PopulateAt(t)
19:         end if
20:         if $(\Sigma' \cap \Omega \neq \emptyset)$ then
21:            let $e = e|e'$
22:         end if
23:      end for
24:      replace $e_{i_j}$ with e in $\phi_s$
25:   end for
26:   let $\Phi = \Phi \cup \{\phi_s\}$
27: end for
```

This additional population of Σ’ (performed by algorithm 2) can happen only if e’$\notin\Omega$ (lines 15 and 17 of algorithm 1) and the state that e’ is incident to is not a final state (line 17) — i.e., $t_{\in S} = \delta(s_{i_j}, e')$ is such that $\Sigma_{t\blacktriangleright} \neq \emptyset$ (to explore, perhaps exhaustively, paths that diverge from the join state s_{i_j} through the state t). In such a case, the paths from t are traversed forwards as long as Σ’ and Ω remain disjoint sets (line 5 and line 8 of

Algorithm 2). In other words, a temporal constraint needs to be imposed if, and only if, e'∈ Ω or ∃(y∈Σ, π'∈P, f∈S, v∈S in $Seq_{\pi'}$, w∈S in $Seq_{\pi'}$) s.t. $t \overset{\pi'}{\rightsquigarrow} f$, $v \xrightarrow{y} w$ and y∈ Ω — i.e., either e' must be one of the events that label *h* (the subpath of π_i s.t. $s_{i_j} \overset{h}{\rightsquigarrow} s$) or *h* and π' must share an event.

Algorithm 2 *PopulateAt(S: t)*

```
1: let $e \in \Sigma_{t\blacktriangleright}$
2: let $\Sigma' = \Sigma' \cup \{e\}$
3: let $t'_{\in S} = \delta(t, e)$
4: let $\Sigma_{t\blacktriangleright} = \Sigma_{t\blacktriangleright} \setminus \{e\}$
5: if $((\Sigma' \cap \Omega = \emptyset)$ and $(\Sigma_{t'\blacktriangleright} \neq \emptyset))$ then
6:     PopulateAt(t')
7: end if
8: if $((\Sigma' \cap \Omega = \emptyset)$ and $(\Sigma_{t\blacktriangleright} \neq \emptyset))$ then
9:     PopulateAt(t)
10: end if
```

4.2.5 Conversion of the PS state machine

In general, for every final state of the SM, the algorithm generates a fault tree (possibly a temporal one) with that final state as the top event. A backwards traversal is performed, starting with each final state and ending at the starting state, and thus every path between those states becomes a new branch in the fault tree. Initially, only AND gates are used, so that all events in each path are represented as a conjunction. Next, for each visited state that is common to more than one path (known as a 'join' state), the algorithm tries to detect whether a temporal operator is required to unambiguously distinguish one path from another. For example, the final state of the PS state machine is 'System Failure', and the join states encountered during backwards traversal are all those visited states with more than one output path for each – thus every state in our example SM is a join state except the final one.

At this stage, the algorithm examines the events forming the branches so far and checks to see if any are shared. In the example SM, it would find that A is a shared event for join state 'A active'. Thus to distinguish e.g. path 1 from path 3, the algorithm would determine that the order of A and S is important and would add a temporal constraint to each branch (A|S for path 1, S|A for path 3). Similarly, since path 2 and path 3 both emerge from the same join state, it would examine the events that cause transitions away from the join state (i.e. B and S in this case) and add the appropriate constraints to distinguish them, i.e. 'S|B' for path 3 and 'B|S' for path 2. Since the temporal operators are only added later, SMs with no shared events, like the one in Figure 4—2, will have no temporal constraints and will consist only of logical conjunctions:

Degraded = B.A

Failed = D.C

No further refinements are necessary in that case. But in a situation like the one in Figure 4—3, A and B are shared events and the join state is the initial one. Therefore we have:

Degraded = B.A|B = A<B

Failed = A.B|A = B<A

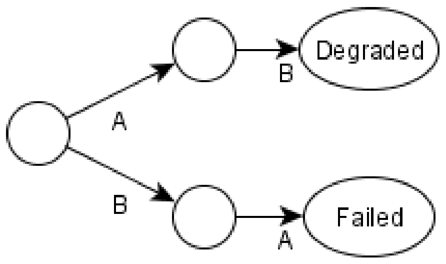


Figure 4—3. Example of a SM with shared events

The result of the algorithm is a set of (possibly temporal) fault trees, one per final error state, each containing one branch for each path through the SM to the given final state. Each sequence or combination of events in the fault trees will be unambiguous, i.e. any shared events will be part of sequences to ensure that the same sequence cannot lead to the top event of more than one fault tree.

Application of the conversion algorithm on the SM of the PS yields the following cut sequences (represented in traversed order), which provide an unambiguous description of the different event sequences that can cause the system to fail:

1) B . A | S | B

2) A . B | S | A

3) A . S | A | B

4) I . A | I | S | B

5) I . B | I | S | A

6) I | S | A | B

7) I . S | I | A | B.

These results show how the sequence of the sensor failure relative to the failure of A, the failure of both primary and standby, and the omission of input alone or relative to other failures are all important in causing system failures.

4.2.6 Minimisation of the generated TFTs

Although the seven results shown previously are sufficient to unambiguously describe the sequences of events that can lead to system failure, these are not minimal. Just as with the initial results of an ordinary fault tree analysis, the sequences can be reduced to a simplified form to give a clearer view of exactly which events are necessary, and in which order should they occur to cause system failure.

We can apply the relevant temporal laws of the variation of Pandora (in which simultaneity is omitted) to perform this minimisation. In the version of Pandora which is adapted to state machines, the laws which include the SAND operator have been modified to remove it. In particular, the three main laws of Pandora, i.e., the Completion Laws (which relate temporal operators to Boolean ones) have their SAND terms are removed in this situation. The three modified Completion Laws are therefore:

X . Y <=> X<Y + Y<X (the Conjunctive Law)

X + Y <=> X|Y + Y|X (the Disjunctive Law)

X <=> Y<X + X|Y (the Reductive Law)

These laws are particularly useful for minimising the results from the conversion algorithm. By applying the Reductive Law, we can both add extra terms to an expression and also reduce multiple cut sequences into one. In this case, cut sequences #4 to #7 can be reduced to just "I" on its own. The other three cut sequences, #1 - #3, can likewise be reduced. Each can be simplified and expanded by the law X<Y . X|Z <=> X<Z<Y + X<Y|Z to obtain the following cut sequences:

A<S<B

A<B|S

B<S<A

B<A|S

S<B<A

S<A|B

These in turn describe constituents of the 3-event Reductive Law, X<Y <=> X<Z<Y + Z<X<Y + X<Y|Z, and can be reduced to even simpler forms. For example, S<B<A + B<S<A + B<A|S reduces to just B<A, and similarly for S<A. Another law, X|Y <=> X<Y + X|Y, means we can obtain S<A<B from S<A|B and thus also reduce S<A|B + A<S<B + A<B|S to just A<B. One further minimisation is then possible here: we can apply the Conjunctive Law to A<B and B<A to obtain A.B. Thus the final minimal cut sequences are as follows:

A.B

S<A

I

These minimised cut sequences give an instant understanding of the failure behaviour of the system: it will fail if both A and B fail, if the sensor S fails before A, or if there is no input. By converting the SM to a temporal fault tree and minimising the expressions produced, we have obtained a concise set of results that provides an more accurate view of the system failure behaviour than the results of the standard FTA expressed by Figure 4—1 (c). In this case, it is clear that preserving the order of A and S is vital in determining whether a system failure will be caused.

4.3 Complexity analysis of the conversion algorithm

The conversion algorithm is biased towards increasingly dynamic systems. The best-case complexity of checking the necessity of a temporal order is *O(n)* and the worst case is *$O(n^2)$*, n being the number of paths from the initial state to the final states in the SM. The best case is a SM where for every divergent path; there exists a sharable event[82] that is incident to the immediately reachable state from the join state at which the path diverges. In such a situation, the price to pay for each join state j is an *$O(m^2)$* operation of temporal order enforcement, and where m is the out-degree of j. The worst case is a SM where for every divergent path, either there are no sharable events or there exists only one sharable event — immediately before the last reachable final state. A SM where there are no sharable events at all (e.g. SM of a static system) is also a worst-case scenario.

To show the quadratic growth of the conversion algorithm with the number of full paths, we define a marking function that marks every state with the number of paths to the final states (these latter states are special cases which are marked with 1). Let *M* be that marking function s.t. $M : S \longrightarrow \mathbb{N}^+$. The number of paths from a state u to the final states is the sum, for all events incident from u to its successors (i.e., the immediately reachable states from u), of the number of paths from these successors to the final states (i.e., their corresponding markings), and hence:

$$M(u) = \begin{cases} 1 & if\ u\ is\ a\ final\ state \\ \\ \displaystyle\sum_{e \in \Sigma_{u\blacktriangleright}} M(\delta(u,e)) & otherwise \end{cases}$$

This algorithm will add the marking of each successor v as many times as there are events that are incident from u to v. Apart from the final states, every state u will be marked such that *M*(u) = $|P^u|$, where $P^u = \{\pi \in P \mid u \overset{\pi}{\rightsquigarrow} f,\ f \in F\}$. For example, marking the states of the SM of the PS in Figure 4—1 (b) gives *M*(A active) = *M*(δ(A active, A fails)) + *M*(δ(A active, O-I)) + *M*(δ(A active, B fails)) + *M*(δ(A active, S fails)), since we have:

[82] An event which causes the condition for imposing a temporal constraint to be satisfied.

M(δ(A active, A fails)) = *M*(B active A failed) = *M*(δ(B active A failed, O-I)) + *M*(δ(B active A failed, B fails)) = 2.

M(δ(A active, O-I)) = *M*(System Failure) = 1.

M(δ(A active, B fails)) = *M*(A active B failed) = *M*(δ(A active B failed, A fails)) + *M*(δ(A active B failed, O-I)) = 2.

M(δ(A active, S fails)) = *M*(A active S failed) = *M*(δ(A active S failed, A fails)) + *M*(δ(A active S failed, O-I)) = 2.

Therefore, the marking of the initial state *M*(A active) is totalling seven full paths in the SM. This marking conforms to the backward traversals that are performed from each of the final states (line 9 of Algorithm 1 in section 4.2.4).

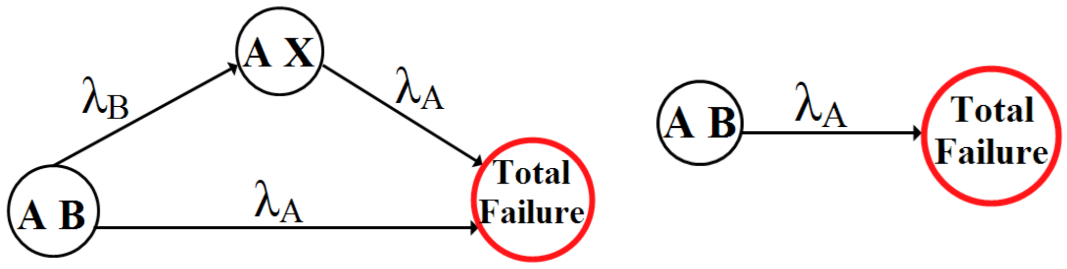
Let N be the total number of paths (from the initial state to the final states). During each backward traversal starting from a final state s, at every visited state u that is a join state (i.e., line 11 of Algorithm 1), at worst, all remaining paths to all possible final states will be exhaustively traversed forwards by Algorithm 2. This results in a total number *M*(u) of traversed paths. This number is preserved along the backward traversal at every state that is not a join state until another join state gets visited, ultimately the initial state s_0 with *M*(s_0) equals to N. This will be repeated Ns times for the final state s, where Ns is the total number of paths from s_0 to s. Thus, for all final states we have a total number of traversed paths which is equal to N × $\sum_{s \in F}$ Ns, i.e., N^2. Note that to determine each Ns s∈F, we can use another version *M'* of the marking function *M*, which marks the initial state with 1. Thereafter, the number of paths reaching a state u is the sum, for all elements in $\Sigma_{u\blacktriangleleft}$, of the number of paths that reach each predecessor u' (i.e., u'→u), and hence:

$$M'(u) = \begin{cases} 1 & \textit{if u is the initial state} \\ \displaystyle\sum_{(e,u') \in \Sigma_{u\blacktriangleleft}} M'(u') & \textit{otherwise} \end{cases}$$

4.4 Supplemental application of the conversion algorithm

In this section, we present an additional contribution by showing that we can apply the conversion algorithm on Markov state diagrams for the purpose to automatically optimise these. Markov models are well-known with the state explosion problem and, therefore, a work which contributes to their optimisation by reducing their size while preserving the quantitative results can have a positive impact overall.

To illuminate this point, let us assume a system with two components A and B with the failure rates λ_A and λ_B respectively. Figure 4—4 (a) represents the corresponding Markov model such that initially both components are working (which is described by the initial state "A B"). Thereafter, if B fails first there will be a transition to the state "A X" (i.e., A is working and B is failed) and then only a failure of A can cause a complete failure of the system. However, if A fails first then the system fails immediately.



(a) before optimisation (b) after optimisation

Figure 4—4. A Markov model optimisation example

For the sake of clarity, in the following we use the symbols A and B to designate the failure events which correspond to the failure rates λ_A and λ_B respectively. The application of the conversion algorithm on the state-transition diagram (a) gives:

"Total failure" = A|B + A.B|A

We can perform some Pandora minimisation on the failure expression generated by the algorithm as follows:

"Total failure" = A|B + B<A (using the temporal law Y.X|Y $\Leftrightarrow$ X<Y)

= A<B + A.¬ B + B<A (using the temporal law X|Y ⇔ X<Y + X.¬Y)

= A. B + A.¬ B (using the temporal law X.Y ⇔ X<Y + Y<X)

= A.(B + ¬B) (using the distributive law)

= A. 1 (tautology)

Therefore, we can simply have: "Total failure" = A and the corresponding optimised Markov model is as represented by Figure 4—4 (b).

Let us now compare the quantitative results provided by the Markov solutions corresponding to both models, i.e., (a) and (b).

1) The left hand side model (a):

$\frac{d}{dt} P_{AB}(t) = -(\lambda_A + \lambda_B) P_{AB}(t)$ where P_{AB} is the probability for the state "AB".

$\frac{d}{dt} P_{AX}(t) = \lambda_B P_{AB}(t) - \lambda_A P_{AX}(t)$ where P_{AX} is the probability for the state "AX".

$\frac{d}{dt} P_F(t) = \lambda_A P_{AB}(t) + \lambda_A P_{AX}(t)$ where P_F is the probability for the final state.

The solution of the differential equations is given by:

$P_{AB}(t) = e^{-(\lambda_A+\lambda_B)t}$

$P_{AX}(t) = e^{-\lambda_A t} - e^{-(\lambda_A+\lambda_B)t}$

Concerning $P_F(t)$ we have:

$\frac{d}{dt} P_F(t) = \lambda_A P_{AB}(t) + \lambda_A P_{AX}(t) = \lambda_A e^{-(\lambda_A+\lambda_B)t} + \lambda_A e^{-\lambda_A t} - \lambda_A e^{-(\lambda_A+\lambda_B)t} = \lambda_A e^{-\lambda_A t}$

$\Rightarrow P_F(t) = -e^{-\lambda_A t} + \text{Cste}$ (the constant is 1 since $P_F(0) = 0 = \text{Cste} - e^{-\lambda_A \times 0} = \text{Cste} - 1$)

$\Rightarrow P_F(t) = 1 - e^{-\lambda_A t}$

2) The right hand side model (b):

$\frac{d}{dt} P_{AB}(t) = -\lambda_A P_{AB}(t)$

$\frac{d}{dt} P_F(t) = \lambda_A P_{AB}(t) = \lambda_A e^{-\lambda_A t}$

The solution of the differential equations is given by:

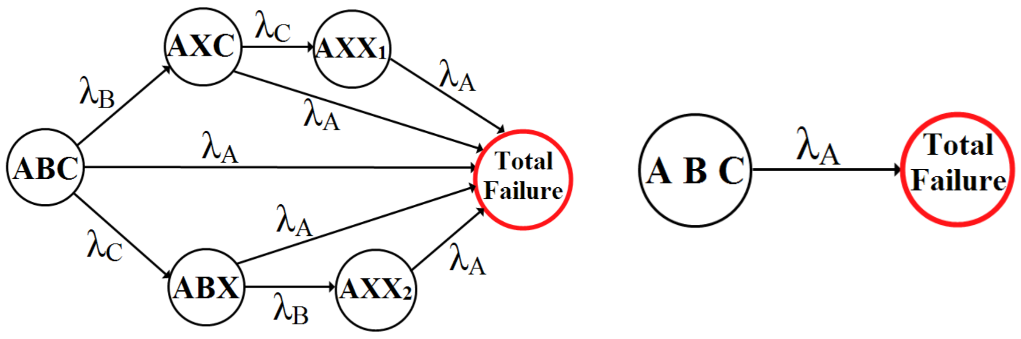
$P_{AB}(t) = e^{-\lambda_A t}$

$P_F(t) = \int \lambda_A e^{-\lambda_A t} dt = - e^{-\lambda_A t}$ + Cste (where Cste is a constant)

Since at time $t_0 = 0$ we have $P_{AB}(t_0) = 1$ and $P_F(t_0) = 0$ then Cste must be 1

This means that $P_F(t) = 1 - e^{-\lambda_A t}$ and this is exactly the same solution as for the previous model.

A similar optimization example consists of the model represented by Figure 4—5 (a). This can be the description of a system with three components A, B and C with failure rates λ_A, λ_B and λ_C respectively. Like in the previous example, the sequence "ABC" in the model means a state where all components are working. A failed component is replaced by the symbol 'X' in that sequence — e.g., "AXC" means a state where all components are working except B is failed and the states "AXX_1" and "AXX_2" mean that both B and C are failed; B has failed before C for the former and after C for the latter.



(a) before optimisation (b) after optimisation

Figure 4—5. A second Markov model optimisation example

The application of the conversion algorithm on the state-transition diagram of Figure 4—5 (a) generates a failure expression which can be minimised (using the Pandora temporal laws) to simply the failure event with rate λ_A (this is fully detailed in appendix

A — 5), and for which the corresponding Markov model is represented by Figure 4—5(b). The Markov solutions give identical final state probabilities in both models (a) and (b).

Similar redundancies can be removed from the Markov model of the generic primary-standby system (depicted in chapter 2 section 2.3.4), and which is represented in Figure 4—6 such that the failure states are all merged into one final state (i.e., the complete failure of the system). The meanings of the transition labels and the state names appearing in the figure are as follows:

λ_A, λ_B and λ_S are failure rates for A, B and S resp.

A B $_S$ (system functioning with A, none is failed)

X B $_S$ (system functioning with B, only A is failed)

A B $_X$ (system functioning with A, only S is failed)

A X $_S$ (system functioning with A, only B is failed)

X B $_X$ (system functioning with B, both A and S are failed)

A X $_X$ (system functioning with A, both B and S are failed)

This representation with one system failure state helps to show some redundant transitions which are similar to those appearing in the Markov model examples of figuresFigure 4—4 andFigure 4—5. For example in the top of the figure from the state "XB_S", either the two transitions labelled with λ_S then λ_B lead the system to a total failure, or simply only the transition with rate λ_B can cause the same effect (from that state). Those together can be optimised to only one transition labelled with λ_B. Similarly, we have two transitions labelled with λ_S then λ_A leading the system (from the state "AX_S", see bottom of Figure 4—6) to the "Total Failure" state, as well as only one transition with the rate λ_A also leading the system to a complete failure (from the same state). Those two scenarios together can, thus, be optimised into a one-transition scenario (labelled with the rate λ_A).

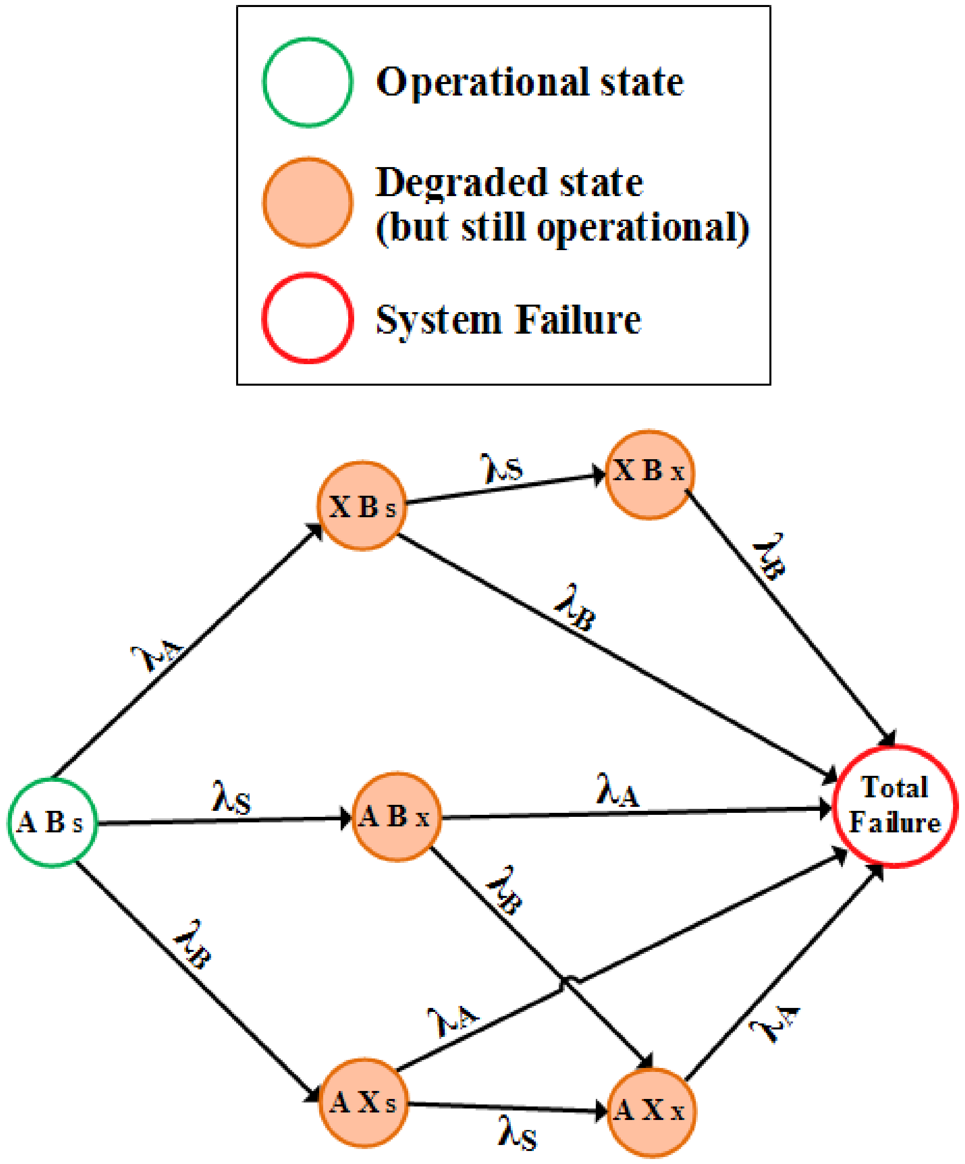


Figure 4—6. Markov model of the PS (all failure states merged into one state)

Like the previous examples, we apply the conversion algorithm on the Markov model to generate the following TFTs, and where (for the sake of clarity) A, B and S replace the rates λ_A, λ_B and λ_S respectively.

"Total Failure" =

B.S|B.A|S|B + (1)

A.B|A.S|B|A + (2)

A.S|A.B|S|A + (3)

A|B.S|B|A + (4)

B|S.A|S|B + (5)

A|S.B|S|A (6)

The application of the POR transformation law (Y.X|Y ⇔ X<Y) gives:

“Total Failure” =

S<B . A<B . A<S + (1)

B<A . S<A . S<B + (2)

S<A . B<A . B<S + (3)

A|B . S<A . S|B + (4)

B|S . A<B . A|S + (5)

A|S . B<A . B|S (6)

Then, the application of the extension law (X<Y . Y<Z ⇔ X<Y . Y<Z . X<Z ⇔ X<Y<Z) on (1), (2) and (3) gives:

“Total Failure” =

A<S<B + (1)

S<B<A + (2)

B<S<A + (3)

A|B . S<A . S|B + (4)

B|S . A<B . A|S + (5)

A|S . B<A . B|S (6)

Next, we apply the extension law (X<Y . Y|Z ⇔ X<Y . Y|Z . X|Z) on (4), (5) and (6); hence:

“Total Failure” =

A<S<B + (1)

S<B<A +   (2)

B<S<A +   (3)

A|B . S<A +   (4)

B|S . A<B +   (5)

A|S . B<A   (6)

By application of the absorption law (X<Y|Z ⇔ X<Y<Z + X<Y . Y|Z) on (4), (5) and (6), we get the following:

"Total Failure" =

A<S<B +   **(1)**

S<B<A +   **(2)**

B<S<A +   **(3)**

S<A<B +   **(4.1)**

S<A . A|B +   (4.2)

A<B<S +   **(5.1)**

A<B . B|S +   **(5.2)**

B<A<S +   **(6.1)**

B<A . A|S   **(6.2)**

(1), (4.1), (5.1) and (5.2) reduce to A<B by using the completion law (X<Y ⇔ Z<X<Y + X<Z<Y + X<Y<Z + X<Y . Y|Z); these show that the occurrence of S is irrelevant. Similarly, (2), (3), (6.1) and (6.2) reduce to B<A by using the same law. However, before reducing we need to preserve (2), (3) and (4.1) for a further reduction afterwards. This can be done by using the Boolean absorption law (X + X ⇔ X), and hence:

"Total Failure" =

A<B +

S<B<A + (2)

B<S<A + (3)

S<A<B + (4.1)

S<A . A|B + (4.2)

B<A

Finally, by using the same previous completion law we can reduce (2), (3), (4.1) and (4.2) to S<A; these show that the occurrence of B is irrelevant. Therefore, we have:

"Total Failure" =

A<B +

S<A +

B<A

These constitute the TFT description of the obtained Markov model of Figure 4—7, and where the state 1 is a state of the PS in which the components A, B and S are all OK, and the states 2, 3 and 4 represent states of the system in which A has failed, S has failed or B has failed, respectively.

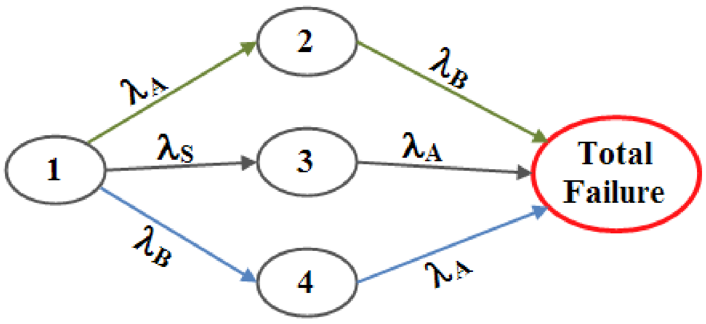


Figure 4—7. The PS Markov model produced without redundancies

In appendix A — 6, we show that the optimised Markov model of the PS preserves the same probabilistic result for the "Total Failure" state.

The use of the optimised Markov model is sufficient if the emphasis is mainly put on computing a probabilistic measure for the complete failure of the system. However, it is always possible to build the full Markov model from the optimised one if some probabilistic results[83] related to partial loss of functionality are required. This can be performed by applying the algorithm below:

<u>Algorithm 3: construction of an elaborated Markov model</u>

Input: optimised Markov model or a reduced SM representation of the failure behaviour of a system.

1. Initiate the construction of the full Model starting from a state where all the system components are operational.
2. Generate a set of target states (with the associated transition rates) by considering the effect of failure of every component.
3. Determine for every new state whether it is an operational or a failed state by checking it against the optimised Markov (or the reduced SM) model.
4. Repeat the same (as steps 2 and 3) for every newly generated state, but only by considering the effects of the remaining failures (which have not already been treated in a same path from the initial state).
5. Stop when no more failure scenarios remain — i.e., all paths have been exhaustively constructed.

For the sake of clarity, to compute the probability of being in a state where, e.g., the PS system is in primary mode but both the backup component and the sensor are failed, we only highlight the two failure scenarios of interest rather than running completely the algorithm to build the full Markov model. We start from the initial state in which all components are functional. Next, if the sensor (resp. backup) fails first, then the generated state would match with state 3 (resp. state 4) of the optimised model in Figure

[83] Though irrelevant to a complete system failure since the corresponding states were removed during optimisation.

4—7, and which shows that the system is still operational. A subsequent failure of the backup (resp. the sensor) leads to the generation of a new state in the full model to build which best matches with the current, i.e. the same state in the optimised model — state 3 (resp. state 4), since only a failure of the primary component causes a change of state in the optimised version. This means that the recently generated state of the full model is still an operational state.

Algorithm 3 is not only useful[84] considering an optimised Markov model as input. But, it is mainly important if the input is an abstract state-transition representation of the failure behaviour of a system. In some circumstances quantitative results may be required directly if it is possible, i.e., without passing through the qualitative analysis process. For example, we have presented in chapter 3 section 3.3 an abstract SM description of the failure behaviour of the vehicle management system VMS (Figure 3—4). That SM is used as input to the algorithm which generates TFTs, and after a minimisation process we can obtain the minimal cut-sequences as demonstrated in that chapter. These obtained cut-sequences can then be quantitatively analysed, e.g., based on the work in (Merle, 2010) which accomodates any failure distribution of basic events. However, if the failure distribution of the basic events is particularly exponential and if a Markov solution is required directly from that reduced SM, we can then apply algorithm 3 to generate the corresponding Markov model — every new state will be tagged operational or failed by checking against the reduced SM along the exceution of the algorithm and every transition label will be substituted with the corresponding failure rate.

4.5 Quantitative analysis of TFTs

The work in (Merle, 2010), i.e., quantitative analysis of dynamic fault trees (DFTs) based on the structure function presents a state-of-the-art method which enables performing probabilistic assessments on dynamic systems irrespective of the failure distribution of the basic failure events. The method uses algebraic expressions for the top-events of DFTs and the probabilistic models of the dynamic gates are given. To facilitate the quantitative analysis, these DFT expressions can be reduced to sum-of-product canonical

[84] To calculate probabilities related to non-complete failure states that have been removed during optimisation.

forms. Minimisation is complex, but this is not uncommon when it comes to qualitatively analyse dynamic or temporal fault trees and there can be many minimisation criteria.

In (Merle, 2010), faults or failure events are considered non-repairable and can be treated as functions of time. For example, let a(t) be a function of time 't' describing a fault 'a'. the possible two values of a(t) are 0 for no fault and 1 for fault. Let d(a) be the date of occurrence of a fault, i.e., single change of the value of a(t) from 0 to 1 at d(a).

Therefore, we have:

$$a(t) = \begin{cases} 0, & if\ 0 \leq t < d(a) \\ 1, & if\ t \geq d(a) \end{cases}$$

In the algebraic models of static gates, this allows functions of time to be manipulated as classical boolean values, though with the help of two identity elements — i.e., one for a fault which always occurs and another one for a fault which never occurs.

To express sequences of events, temporal and dynamic gates were also introduced in Merle's work. But here we focus mainly on the temporal operator '◁' used in the algebraic framework (called non-inclusive BEFORE) and which has exactly the same semantics as the Priority-OR operator '|' of Pandora — i.e., to be true, the second input event of the non-inclusive BEFORE has to occur strictly after the first input event or to not occur at all. Its probabilstic model is given by (assuming that the basic events are statistically independent, and this hypothesis is maintained for all this section):

$$\Pr[A \triangleleft B](t) = \int_0^t f_A(u)\,(1 - F_B(u))du \qquad (1)$$

Where 'a' and 'b' are two events and for an event 'x', $F_x(t)$ is the cumulative distribution function and $f_x(t)$ is the probability density function s.t. $f_x(t) = F_x^{'}(t)$.

Similarly, the probabilistic model given for the Priority-AND is as follows:

$$\Pr[B.(A \triangleleft B)](t) = \int_0^t f_B(u)\,F_A(u))du \qquad (2)$$

Where B.(A◁B) is equivalent to A<B (i.e., A PAND B) in Pandora, the cumulative distribution and the probability density functions are as defined previously (i.e., $F_x(t)$ and $f_x(t)$ for an event x, respectively).

A central issue with these probabilistic models is that they accommodate any failure distribution and are, thus, more realistic since lots of components do not conform to the exponential failure distribution. However, the latter is the more common failure distribution of basic events and to illustrate this aspect on this particular case, let us assume that λ_A and λ_B are the failure rates which are associated with the events A and B, respectively. Therefore, we have:

$$F_A(t) = 1 - e^{-\lambda_A t}$$
$$f_A(t) = \lambda_A \, e^{-\lambda_A t}$$

Similarly,

$$F_B(t) = 1 - e^{-\lambda_B t}$$
$$f_B(t) = \lambda_B \, e^{-\lambda_B t}$$

Therefore, for the particular case of exponential failure distribution for the basic events, we have:

$$\begin{aligned}
\Pr[A \lhd B](t) &= \int_0^t f_A(u)\,(1 - F_B(u))du \\
&= \int_0^t \lambda_A \, e^{-\lambda_A t}\,[1 - (1 - e^{-\lambda_B t})]dt \\
&= \int_0^t \lambda_A \, e^{-\lambda_A t}\,(e^{-\lambda_B t})dt = \int_0^t \lambda_A \, e^{-(\lambda_A+\lambda_B)t}\,dt
\end{aligned}$$

At t = 0, $\Pr[A \lhd B](0) = 0$. Thus we have:

$$\Pr[A \lhd B](t) = -\frac{\lambda_A}{\lambda_A+\lambda_B}\, e^{-(\lambda_A+\lambda_B)t} + \frac{\lambda_A}{\lambda_A+\lambda_B} \qquad (3)$$

Concerning the PAND-equivalent expression in Merle's probabilistic model, for the case of exponential distribution we have:

$$\begin{aligned}
\Pr[B.(A \lhd B)](t) &= \int_0^t f_B(u)\,F_A(u))du \\
&= \int_0^t \lambda_B \, e^{-\lambda_B t}\,(1 - e^{-\lambda_A t})dt
\end{aligned}$$

At t = 0, $\Pr[B.(A \lhd B)](0) = 0$. Therefore, we have:

$$\Pr[B.(A \lhd B)](t) = \frac{\lambda_B}{\lambda_A+\lambda_B}\, e^{-(\lambda_A+\lambda_B)t} - e^{-\lambda_B t} + \frac{\lambda_A}{\lambda_A+\lambda_B} \qquad (4)$$

The application of the probabilistic model of the PAND given in (2) to the particular case of exponential distribution given in (4) shows the same result given in (Fussell *et al.*, 1976) for a PAND gate with two input events.

There have been considerable works in the literature around solving DFTs quantitatively [e.g., in (Amari *et al.*, 2003) & (Fussell *et al.*, 1976)]. However during the period of this thesis, the only[85] novel and mathematically sound approach that includes POR-like[86] gate (i.e., non-inclusive BEFORE) and accommodates any failure distribution for the basic events is in (Merle, 2010), it also deals with qualitative analysis.

4.5.1 Some comparisons with combinatorial techniques

As stated previously in this section, the following hold under the hypothesis of statistical independence, exponential failure distribution and non-simultaneity of occurrence of failure events. Let A and B be two failure events with rates λ_A and λ_B, respectively. Therefore, we have:

$$P(A \text{ AND } B) = P(A) \times P(B) = (1 - e^{-\lambda_A t}) \times (1 - e^{-\lambda_B t}) \qquad (5)$$

$$= e^{-(\lambda_A+\lambda_B)t} - e^{-\lambda_A t} - e^{-\lambda_B t} + 1$$

It was previously shown in chapter 2 section 2.3.4 that (5) gives the same result as the result given by the Markov model of Figure 2—20, which describes the conjunctive failure of two parallel components. Moreover, the Markov model which was optimised for the particular case where the failure rates are identical (i.e., $\lambda_A = \lambda_B = \lambda$) also gives the same result as given by (5) — i.e., P(A AND B) = $e^{-2\lambda t} - 2\,e^{-\lambda t} + 1$.

Let us now compare it with the results given using the probabilistic model in (4) for the PAND gate. The first completion Pandora law given in chapter 3 shows that: A.B ⇔ A<B + B<A. Also, "A<B" and "B<A" are mutually exclusive. Therefore, we have:

$$P(A \text{ AND } B) = P(A \text{ PAND } B) + P(B \text{ PAND } A)$$

$$= \frac{\lambda_B}{\lambda_A+\lambda_B}\, e^{-(\lambda_A+\lambda_B)t} - e^{-\lambda_B t} + \frac{\lambda_A}{\lambda_A+\lambda_B} \qquad \text{for "A PAND B" given by (4)}$$

$$+$$

$$\frac{\lambda_A}{\lambda_A+\lambda_B}\, e^{-(\lambda_A+\lambda_B)t} - e^{-\lambda_A t} + \frac{\lambda_B}{\lambda_A+\lambda_B} \qquad \text{for "B PAND A" given by (4)}$$

[85] i.e., which is found published in the literature.

[86] i.e., semantically like the Pandora POR gate.

$$= e^{-(\lambda_A+\lambda_B)t} - e^{-\lambda_A t} - e^{-\lambda_B t} + 1$$

and this is the same result given in (5).

Let us now compare it using NOT operator ('¬'), i.e., we deduct from one the probabilities of either A or B occurring alone (not both) as well as the probability of neither A nor B occurs, and as follows:

P(A failing AND B failing) = 1 – P(A AND ¬B) – P(¬A AND B) – P(¬A AND ¬B)

$$= 1 - (e^{-\lambda_B t} - e^{-(\lambda_A+\lambda_B)t}) - (e^{-\lambda_A t} - e^{-(\lambda_A+\lambda_B)t}) - (e^{-\lambda_A t} \times e^{-\lambda_B t})$$

$$= 1 - e^{-\lambda_A t} - e^{-\lambda_B t} + e^{-(\lambda_A+\lambda_B)t}$$

and this is the same result given in (5).

4.5.2 Some comparisons with Markov solutions

Let us assume two components whose failure events A and B are associated with their respective failure rates λ_A and λ_B. Figure 4—8 represents a Markov model for the two failure events occurring in sequence. If B fails first, then a transition to state numbered 2 occurs. Otherwise (i.e., A fails first), a transition to state 3 occurs and will be followed by a transition to state 4 upon a subsequent failure of B. Thus, state numbered 4 is the state which we use to determine the Markov probabilistic model for A PAND B.

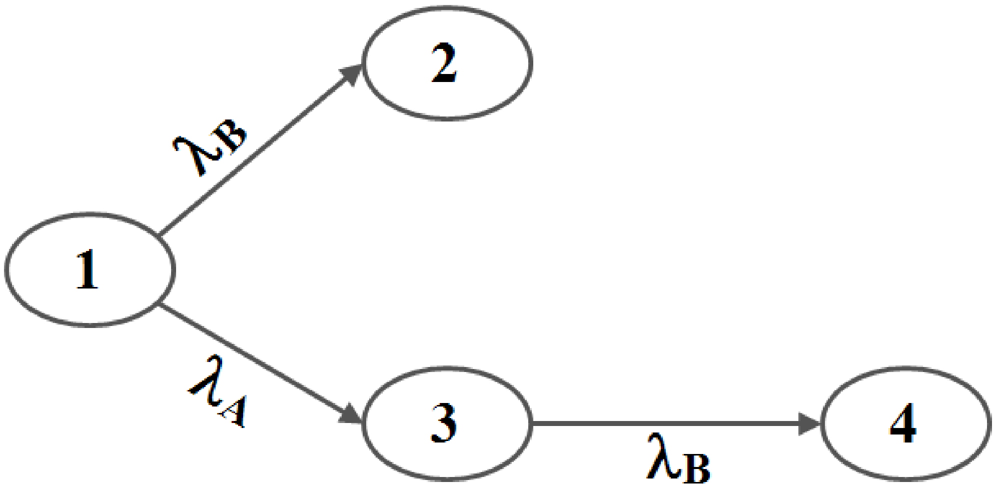


Figure 4—8. Markov model of a PAND gate with two input events (state 4)

If we use this Markov model (which is a state-transition diagram) as input to the conversion algorithm presented in section 4.2.4, we also get A APAND B for state 4, as follows (for the sake of clarity we use A and B instead of λ_A and λ_B, respectively):

Expression(state 2) = B|A

Expression(state 4) = B . A|B $\Leftrightarrow$ A<B (POR transformation law #1, see chapter 3)

The differential equations are as follows:

$$\frac{d}{dt} P_1(t) = -(\lambda_A + \lambda_B)\, P_1(t)$$

$$\frac{d}{dt} P_2(t) = \lambda_B\, P_1(t)$$

$$\frac{d}{dt} P_3(t) = \lambda_A\, P_1(t) - \lambda_B\, P_3(t)$$

$$\frac{d}{dt} P_4(t) = \lambda_B\, P_3(t)$$

and the solution of the differential equations is given by:

$$P_1(t) = e^{-(\lambda_A + \lambda_B)t}$$

$$P_2(t) = -\frac{\lambda_B}{\lambda_A + \lambda_B}\, e^{-(\lambda_A + \lambda_B)t} + \frac{\lambda_B}{\lambda_A + \lambda_B}$$

$$P_3(t) = e^{-\lambda_B t} - e^{-(\lambda_A + \lambda_B)t}$$

$$P_4(t) = \frac{\lambda_B}{\lambda_A + \lambda_B}\, e^{-(\lambda_A + \lambda_B)t} - e^{-\lambda_B t} + \frac{\lambda_A}{\lambda_A + \lambda_B}$$

On the one hand, This Markov solution shows that $P_2(t)$ is the same result given in (3) after swapping A and B, i.e. it is the probability Pr[B$\lhd$A](t), or Pr[B|A](t) by using the vocabulary of Pandora. The corresponding algebraic expression generated by the conversion algorithm is Expression(state 2), which is equal to “B POR A” (i.e., B|A). On the other hand, it shows that $P_4(t)$ is the result of Pr[B.(A$\lhd$B)](t) given in (4) or Pr[A<B](t) by using the vocabulary of Pandora. The corresponding algebraic expression generated by the conversion algorithm is Expression(state 4), which equal to “A PAND B” (i.e., A<B).

Since the probability of being in state 2 is the probability of “B|A”, then the probability of being in the lower path of the Markov model is the probability of “A|B”.

State 3 represents a situation where A occurred and B has not occurred (i.e., “A.¬B”). State 4 represents, as seen earlier, a situation where both A and B occurred, but in sequence (i.e., “A<B”). Since “A|B” means that either A occurs alone (B does not occur at all) or both A and B occur in sequence (A occurs before B occurs), then the probability of “A.¬B + A<B” should be equal to $P_3(t) + P_4(t)$, and which should also be equal to Pr[A◁B](t) — i.e., the probabilistic model given in (3), and as demonstrated next.

To calculate the probability of “A.¬B + A<B” (i.e., probability of “A|B”), we need to recognise whether the events are independent or mutually exclusive. Statistical independence of events is crucial in probability and statistics. Two events are statistically independent if the occurrence of either event has no effect on the occurrence of the other. However, “A.¬B” and “A<B” are mutually exclusive (i.e., we do not deduct the product of their probabilities from their sum), and hence:

On the one hand, we have:

$$\Pr(A|B) = \Pr(A.\neg B) + \Pr(A<B)$$

$$= F(a).(1 - F(b)) + \int_0^t f_b(u)\, F_a(u))du$$

And for the particular case of exponential distribution, we have:

$$\Pr(A|B) = (1 - e^{-\lambda_A t}).\,(e^{-\lambda_B t}) + \int_0^t \lambda_B\, e^{-\lambda_B t}\,(1 - e^{-\lambda_A t})dt$$

$$= -\,\frac{\lambda_A}{\lambda_A + \lambda_B}\, e^{-(\lambda_A+\lambda_B)t} + \frac{\lambda_A}{\lambda_A + \lambda_B}$$ and which is the same as given in (3)

On the other hand, we have:

$$P3(t) + P4(t) = \quad e^{-\lambda_B t} - e^{-(\lambda_A + \lambda_B)t}$$

$$+$$

$$\frac{\lambda_B}{\lambda_A + \lambda_B}\, e^{-(\lambda_A + \lambda_B)t} - e^{-\lambda_B t} + \frac{\lambda_A}{\lambda_A + \lambda_B}$$

$$= -\,\frac{\lambda_A}{\lambda_A + \lambda_B}\, e^{-(\lambda_A+\lambda_B)t} + \frac{\lambda_A}{\lambda_A + \lambda_B} = \Pr[A \lhd B](t)$$

and which is also the same as given in (3).

This means that we can optimise the Markov model of Figure 4—8 to adjust it for the purpose of quantifying a POR gate with two input events, and hence a more accurate model for such a POR gate is described by Figure 4—9.

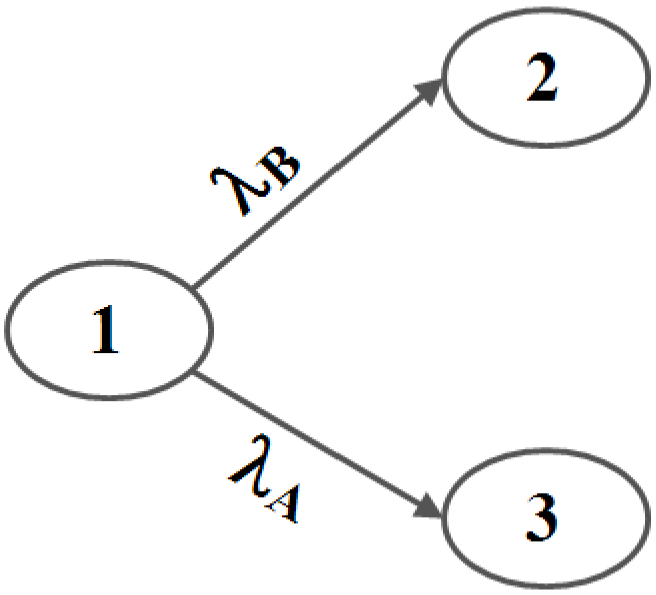


Figure 4—9. Markov model of a POR gate with two input events

The differential equations are as follows:

$$\frac{d}{dt} P_1(t) = -(\lambda_A + \lambda_B)\, P_1(t)$$

$$\frac{d}{dt} P_2(t) = \lambda_B\, P_1(t)$$

$$\frac{d}{dt} P_3(t) = \lambda_A\, P_1(t)$$

and the solution of the differential equations is given by:

$$P_1(t) = e^{-(\lambda_A + \lambda_B)t}$$

$$P_2(t) = -\frac{\lambda_B}{\lambda_A + \lambda_B} e^{-(\lambda_A + \lambda_B)t} + \frac{\lambda_B}{\lambda_A + \lambda_B} \quad = \quad \Pr[B|A](t) = \Pr[B \lhd A](t), \text{ as given in (3)}$$
by swapping A and B

$$P_3(t) = -\frac{\lambda_A}{\lambda_A + \lambda_B} e^{-(\lambda_A + \lambda_B)t} + \frac{\lambda_A}{\lambda_A + \lambda_B} \quad = \quad \Pr[A|B](t) = \Pr[A \lhd B](t), \text{ as in (3)}$$

4.5.3 Probabilistic model of a POR gate with n input events

The model described by Figure 4—9 is more accurate for representing a POR gate with two input events than the preceding model. Indeed, the model of Figure 4—8 was originally intended to describe a PAND gate with two input events, but as we have seen it can also be used to calculate the probabilistic expression of a POR as well. One reason to consider the model of Figure 4—9 as more accurate for this purpose is that it does not only serve to produce exactly the same probabilistic results for a POR as from the preceding model, but it also represents a better candidate model for a POR gate with two input events that we can extend easily to n input events ($n \in N^+$ s.t. $n \geq 2$, see Figure 4—10).

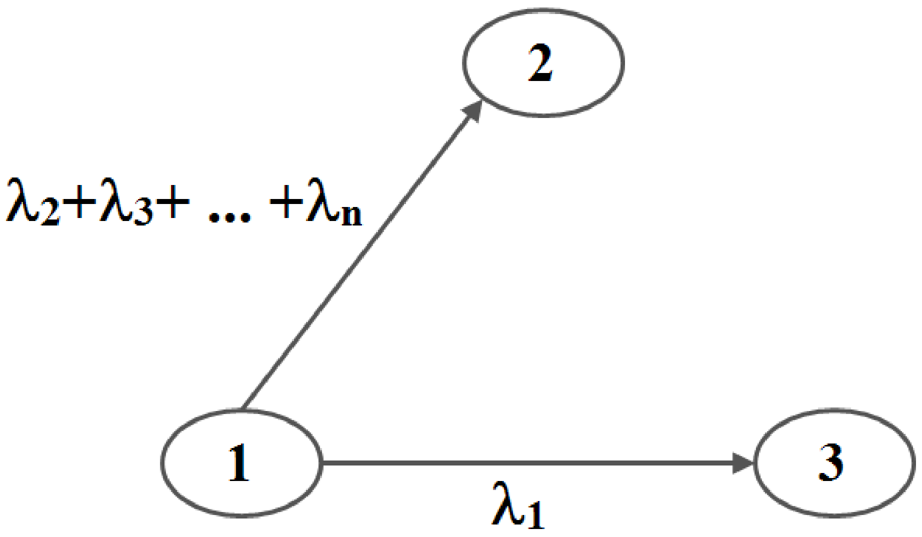


Figure 4—10. Markov model of a POR gate with n input events

In Figure 4—10, each λ_i $1 \leq i \leq n$ represents the failure rate associated with the failure input event F_i $1 \leq i \leq n$ of the POR gate. Therefore, the probability of being in state 3 represents the probability for the expression $F_1|F_2|F_3|$ … $|F_n$ (where $n \geq 2$) which means that either F_1 has occurred alone (none of the F_i $2 \leq i \leq n$ has occurred) or F_1 has occurred first, i.e., before any other F_i $2 \leq i \leq n$ has occurred.

The differential equations are as follows:

$$\frac{d}{dt} P_1(t) = -(\lambda_1 + \lambda_2 + \ldots + \lambda_n) P_1(t)$$

$$\frac{d}{dt} P_2(t) = (\lambda_2+\lambda_3+ \dots +\lambda_n)\, P_1(t)$$

$$\frac{d}{dt} P_3(t) = \lambda_1\, P_1(t)$$

and the solution of the differential equations is given by:

$$P_1(t) = e^{-(\sum_{i=1}^{n} \lambda_i)t}$$

$$P_3(t) = \Pr[F_1|F_2|F_3| \dots |F_n](t) = -\frac{\lambda_1}{\sum_{i=1}^{n} \lambda_i}\, e^{-(\sum_{i=1}^{n} \lambda_i)t} + \frac{\lambda_1}{\sum_{i=1}^{n} \lambda_i}$$

$P_3(t)$ represents the probabilistic model of a POR gate with n input events ($n \geq 2$) for the particular case of exponential failure distribution for the basic events. It gives exactly the same result as given in (3) for the particular case where n=2 (i.e., a POR gate with two input events). We emphasise, once again, that the expression given in (3) is also a particular case for the application of the exponential failure distribution to the expression of Merle (2010) given in (1), and which accommodates any failure distribution for the basic events.

4.6 Discussion

To better model the dynamic failure behaviour, state machines are often used, but these typically require conversion to some other format (like fault trees or Petri nets) before they can be analysed. Existing SM-to-FT methods focus on converting them to standard combinatorial fault trees, but this can result in errors in situations where different sequences of the same events have different outcomes.

In this chapter, we have presented a different approach which converts state machines into Pandora temporal fault trees. This has the benefit of better capturing the dynamic behaviour represented by the original state machines, thus allowing more detailed analysis of different sequences of events. Also, our approach helps to reduce the complexity of performing temporal FTA by generating temporal fault tree expressions only if necessary, resulting in a more manageable logical reduction of the fault trees overall.

We demonstrated this technique on a simple generic redundant system and showed how the corresponding SM can be converted into temporal expressions and subsequently analysed to produce useful qualitative information about the dynamic failure behaviour of the system. We also demonstrated the quadratic growth of the conversion algorithm with the number of full paths (from the initial state to the final states) in the SM. The algorithm has been implemented and we hope that this approach can be developed further and become automated as part of the HiP-HOPS safety analysis tool framework. We also aim to extend it further and potentially make it compatible with other SM-based modelling approaches, such as AADL.

An additional application of the conversion algorithm for the purpose of optimising Markov models has been presented. These models are state-transition diagrams and are well-known with the state explosion problem. Therefore, a solution which enables the automatic reduction of the size of a Markov model (while preserving the quantitative results for the “completely failed” state) can help to facilitate the probabilistic assessments. Moreover, we have thoroughly investigated in this chapter different approaches for the quantitative analysis of TFTs and presented a probabilistic model for the quantification of a POR gate with n input events ($n \geq 2$). The conversion algorithm relies mainly on the POR operator to differentiate sequences of events.

In summary, performing meaningful qualitative safety analysis on dynamic systems means using both dynamic modelling and dynamic analysis approaches; by converting SMs to temporal FTs, it becomes possible to do both and thus can analyse more accurately systems which exhibit dynamic behaviour.

5 Compositional synthesis of Temporal Fault Trees from State Machines

5.1 Introduction

Despite the greater expressiveness and higher-level formalism of SMs (and their extensions like statecharts and mode automata), some open research issues still remain. A central issue is representing large-scale dynamic systems in a way which can serve as a solid basis for a subsequent accurate and dynamic analysis. Indeed, it is difficult to represent in a single coherent and complete picture how a large system which is embedded with several complex interrelated components behaves (with its constituent parts) in conditions of failure. Also, the analysis can be unmanageable due to a considerable number of failure events which increases with the number of components involved.

To remedy this problem, we propose a solution which extends the analysis approach presented in chapter 4 — i.e., conversion of SMs to TFTs. The fault trees are generated from the SMs of the individual components of a system. These ‘sub-models’ (i.e., the SMs) are relatively small and thus easier to understand, thereby making their corresponding TFTs tractable. For example, Figure 5—1 (b) shows the SM of one of the components (the sensor) of the PS system.

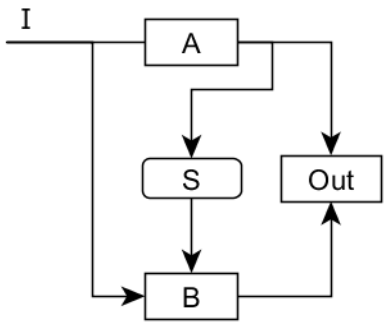
The sensor ‘S’ is initially monitoring the primary component ‘A’. A premature[87] failure of ‘S’ causes the redundant component (‘B’) to be irrevocably disabled; as a consequence, the composed system (the PS) relies exclusively on a single component (the primary ‘A’). In such a situation, the failure state of ‘S’ may only deteriorate to a severe level should an omission of output from ‘A’ (‘O-A’) occur[88]. This wouldn’t happen if the failure of ‘S’ was not premature, as B can take over A’s job (which is still a safe level). Therefore, the effect omission of alert signal (‘O-S’) from the sensor is restricted to a severe failure of ‘S’. ‘O-S’ is a non-atomic[89] event which impacts another component of

[87] During time at which the monitored component is active — i.e., if A fails, S will not be operational to wake B up.

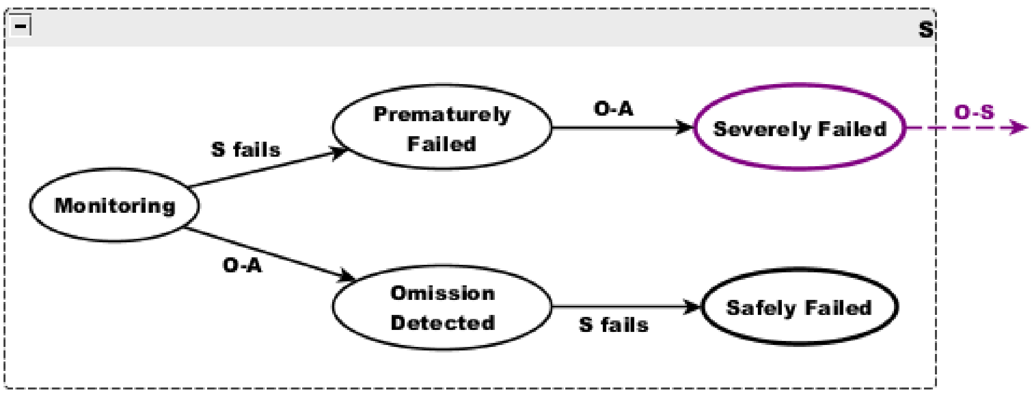
[88] Typically due to an internal failure of ‘A’ (omission of input ‘O-I’ is a single point of failure).

[89] i.e., a fault which is caused by failures in some other components (i.e., it can be expanded into a combination or a sequence of events). It represents one or a group of final state in a component’s SM.

the PS — i.e., similar faults typically cause another component's SM (or the composed system) to change its state.



(a). The PS system.



(b). SM of the sensor S.

Figure 5—1. SM of the sensor of the PS example system.

For example, if the SM of 'S' enters the state "Severely Failed" and thus causing the effect 'O-S', this will impact the SM of 'B' (Figure 5—2) by a transition from "OFF" to "Permanently OFF" (unlike 'A'[90], the initial state of 'B' is "OFF"). Otherwise (i.e., if 'O-A' happens first), then 'S' wakes 'B' up instead (i.e., this alters the state of 'B' to "ON"), and in which case nothing more but an internal failure of 'B' leads the SM to a severe level of failure (the same if 'B' fails dormant then 'O-A' occurs). Omission of input 'O-I' is a single point of failure for the system. It is, therefore, represented at system level — i.e., in the SM of the PS (see Figure 5—3).

[90] Initially, the PS system is functioning in primary mode.

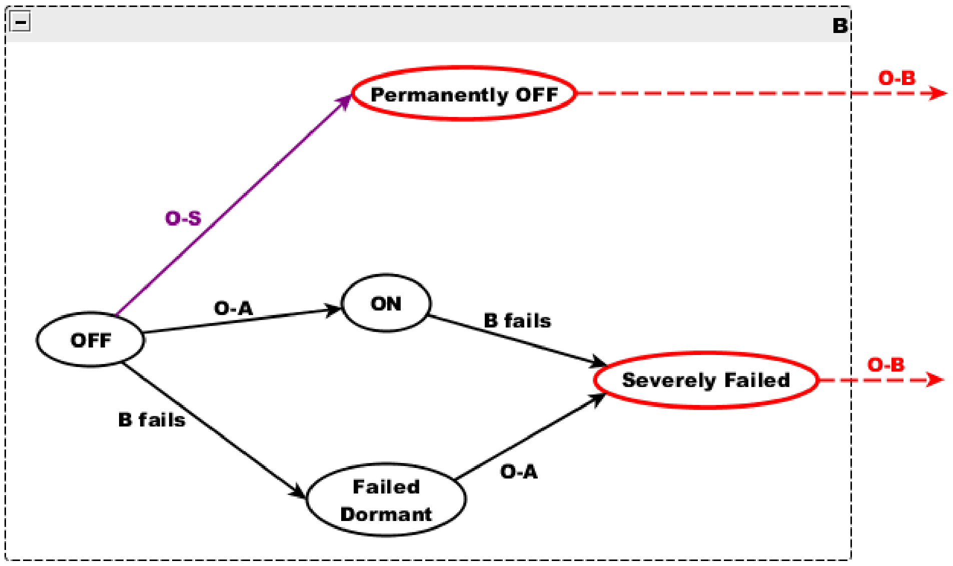


Figure 5—2. SM of the backup B of the PS system.

Unlike the detailed SM of the PS presented in the previous chapters, it is here a highly abstract description of the monolithic behaviour of the system. The system fails as a whole (see Figure 5—3) if there is an omission of input (‘O-I’) or if the backup is unable to take over a failed primary component (and thus omission of output ‘O-B’ from ‘B’) — assuming that the ‘Out’ component of Figure 5—1 (a) does not fail on its own (it only abstracts the output of the system).

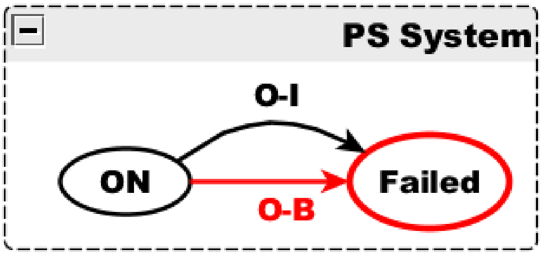


Figure 5—3. Highly abstract SM depicting the monolithic behaviour of the PS.

In this way, the SM of the systems is an abstraction of a hierarchy of the SMs of its components. An event causing a transition in a SM is possibly due to another SM entering one of possible final states (e.g., if the event is not an internal failure). For instance, if the SM of ‘B’ enters either “Permanently OFF” or “Severely Failed”, this will cause ‘O-B’ to occur in the SM of the PS. Similarly, if the SM of ‘S’ enters its “Severely Failed” final state, this will cause ‘O-S’ to occur in the SM of ‘B’.

Modelling the behavioural aspects of the components in this way forms the basis of our compositional approach. The TFTs will be generated from the individual component SMs, and no longer from a single full-scale SM of the system as in the previous chapter. The relatively small component TFTs will be thereafter synthesised by merging then into bigger ones (ultimately extensive system TFTs). For example, Figure 5—4 shows the fault tree with top-event ‘O-S’ (generated from the SM of ‘S’) merged into the fault tree with top-event ‘O-B’ (generated from the SM of ‘B’). These are synthesised to form one expansive cut-sequence of the system fault tree with top-event ‘PS Failed’ (the left branch of Figure 5—4).

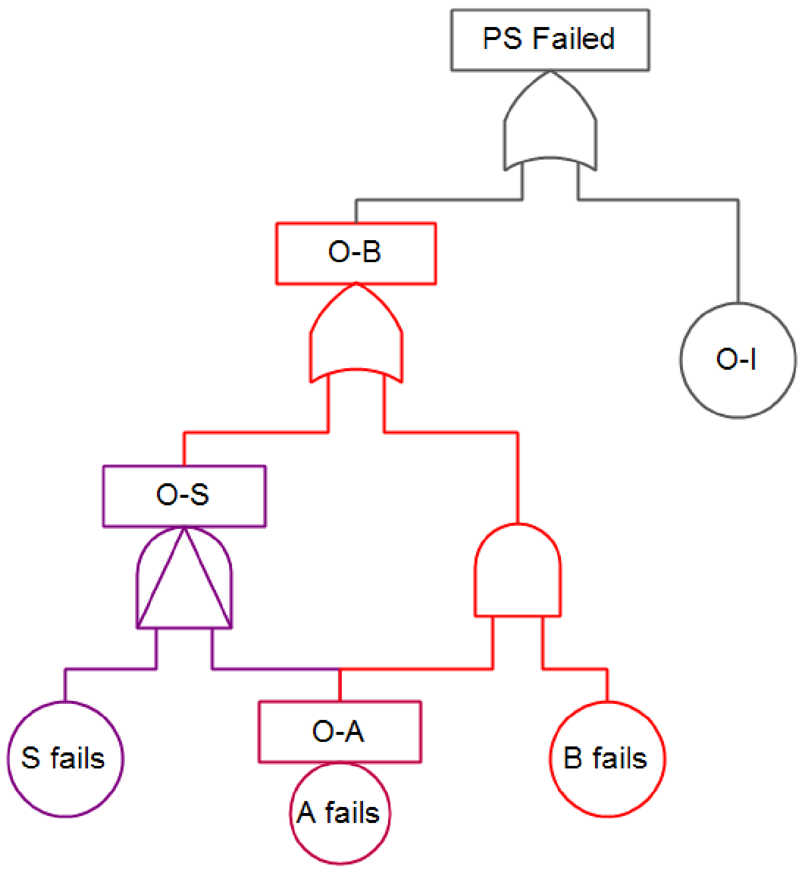


Figure 5—4. Synthesised fault trees.

Using a simple system like the presented PS, we have shown an example for the synthesis of the system TFT (which corresponds to the system failure ‘PS Failed’) from the fault trees which are generated from the SMs of the components. In this chapter, we develop this technique to enable such compositional synthesis of TFTs on a large-scale. Work on TFT minimisation itself is out of the scope of the thesis; nevertheless, minimising the resulted fault trees does not necessarily take place only after the synthesis completes (i.e., once the system TFTs become extensively developed). Indeed, the local TFTs (which are generated from the components' SMs) can be logically reduced wherever possible to remove any new redundancies or contradictions before synthesising them.

5.2 The TFT synthesis approach

Our synthesis technique relies, mainly, on an accurate compositional description of the failure information which complements the architecture of the system to analyse — a state automata model in the form of compositional SMs. The second pillar of the proposed approach is a variation of the conversion algorithm presented in4. We use the algorithm to generate the necessary failure expressions from the individual SMs (we concentrate on the expressions of the final states[91] that represent output deviations). The third pillar is the synthesis algorithm by which the fault trees of these expressions will be merged into (ultimately) comprehensive system fault trees.

The synthesis works backwards by starting with the TFTs of the system failures (i.e., total failures as top events). For instance, in the context of the PS system studied earlier (Figure 5—1.a), we start to generate the first level of the system fault tree of the top event 'PS Failed' from the highly abstract SM of the PS (see Figure 5—3). Thus, the generated preliminary fault tree for the example system is 'PS Failed' = 'O-B' OR 'O-I'. Thereafter, for each event that is derived from the component failures, the synthesis method expands the preliminary system TFTs with the fault trees of the components. Such non-atomic events are typically represented each by the top event of a component's fault tree. For example, 'O-B' (which is an output deviation of the component B) is the top event of the fault tree 'O-S' OR ('O-A' AND 'B fails'). Therefore, expanding 'PS Failed' gives 'O-S' OR ('O-A' AND 'B fails') OR 'O-I'. This will be repeated at each level of every system TFT until no more non-atomic events remain — i.e., the expression of each system failure cannot be expanded any further as all possible substitutions have

[91] We turn our attention to the final states that impact other components or the super-component, i.e., entering such a final state causes a transition in another SM in the hierarchy.

taken place, and the complete fault tree has been synthesised for that particular system failure. For the PS, by substituting ‘O-S’ with ‘S fails’ PAND ‘O-A’ (TFT of component S) then ‘O-A’ with ‘A fails’ (FT of component A), the extensive system-wide TFT of ‘PS Failed’ becomes (‘S fails’ PAND ‘A fails’) OR (‘A fails’ AND ‘B fails’) OR ‘O-I’.

At the end of the synthesis, for every fully developed system (temporal) FT, a final analysis takes place to obtain more reduced cut sequences wherever possible (or minimal cut sets, if the fault tree contains no temporal logic), by using a temporal qualitative analysis approach like Pandora[92]. From this point onwards we will refer to the proposed technique, i.e., the transformation from ***S****tate* ***A****utomata* to synthesised ***F****ault*-trees extended (if necessary) with Pand***ORA*** temporal information as SAF-ORA[93].

5.2.1 Compositional modelling of the analysis-related information

The first central issue mentioned earlier and on which Safora relies is the compositional modelling of the system behaviour. This consists of a highly abstract SM which describes the monolithic failure behaviour of the system at the top-level of the hierarchy and the components (and sub-components) SMs at the lower levels (see Figure 5—5). Typically, each event of the abstract SM of the system is the effect of one (or more) final state(s) being reached in a (or some) component SM(s). If it is the effect of more than one component SM each reaching a final state, then the order in which the final states were entered may affect the overall outcome (and in which case the conversion algorithm will enforce a temporal order as seen in chapter 4).

For example, if two of the SMs of the sub-components level (those which are downstream in the event-causal chain, see Figure 5—5 bottom) each enters a final state, then the remaining SM of the same level (which is upstream) enters its final state too. Thereafter, the conversion algorithm (when applied to the latter SM) will determine if the temporal order in which the first two final states were reached is relevant (and thus needs to be preserved). Moreover, several final states in a SM can each (when entered) cause a

[92] The version which is adapted to SMs, i.e., without simultaneity of events.

[93] Although half of Safora (SAF) is an acronym — State Automata to Fault-trees, the other half (ORA) is not, but preserves the meaning of hour or “time” from its Greek origin (ώρα in Greek). The reason is that Safora is conceived for the premier purpose to produce synthesised temporal FTs from state automata. Nevertheless, Safora has the dual benefit to produce both types of FTs (i.e., static and dynamic) — e.g., if the temporal flavour is not deemed necessary for a fault tree, then it will be static.

same event to occur in any SM upstream in the event-causal chain. In Figure 5—2 for instance, reaching either final state “Permanently OFF” or “Severely Failed” in the SM of the standby component B will cause the event ‘O-B’ (omission of output from B) to occur in the SM of the PS system — i.e., a disjunction of final states being reached.

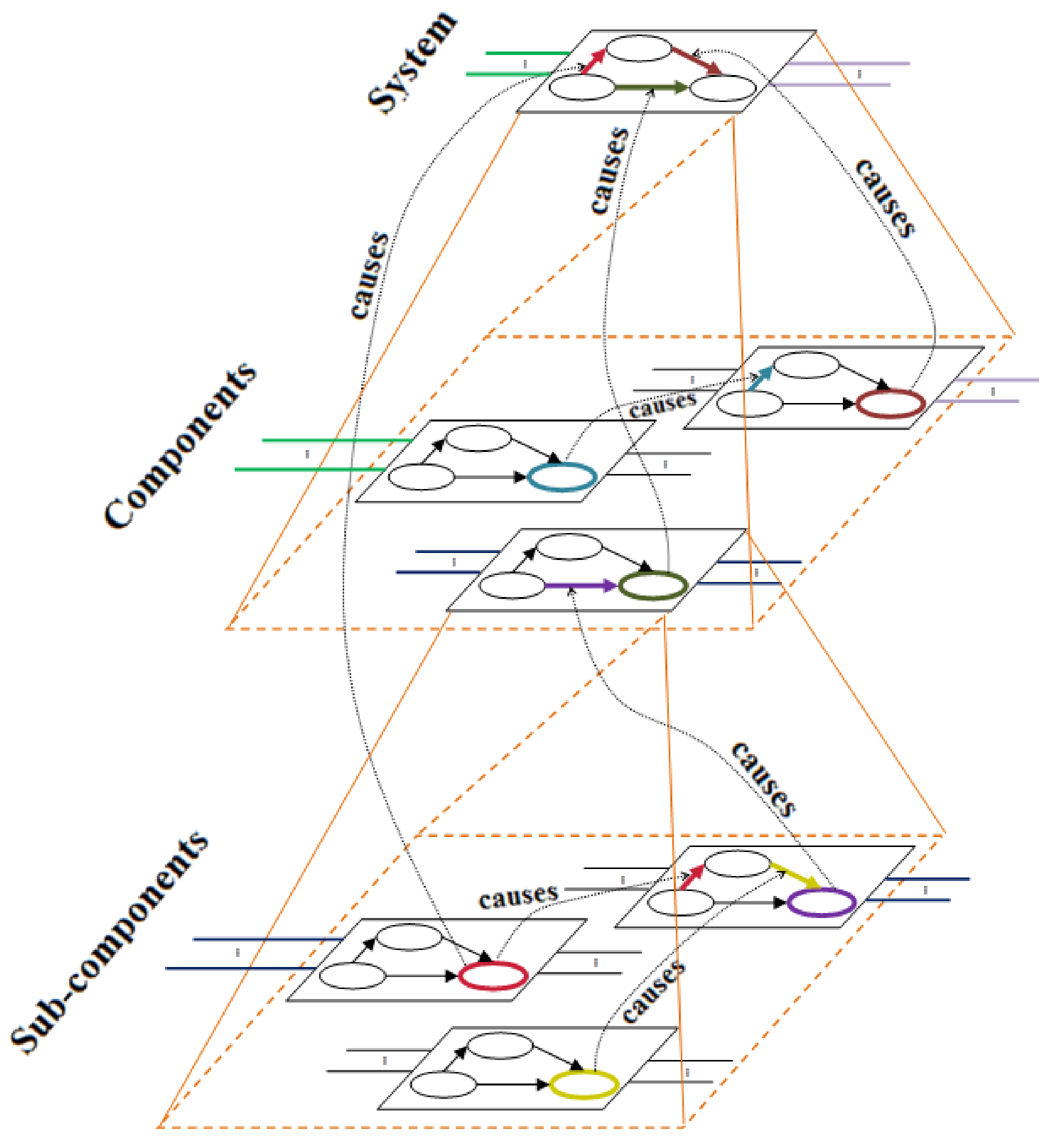


Figure 5—5. Compositional modelling of the system failure behaviour.

Figure 5—5 (top) also shows a situation where a disjunctive combination of entered final states can cause a safety issue upstream (i.e., a system failure in the case of Figure 5—5). In other words, the final state of the abstract SM (top of Figure 5—5) is reached through

different paths; each path has an event which is the effect of reaching a final state in a SM downstream (middle of Figure 5—5).

In this section, we have shown and discussed a way to describe how a system and its constituent parts behave in conditions of failure. The use of compositional SMs for this purpose forms the basis of Safora. In that sense, the generated (temporal) FTs, their synthesis and the produced analysis results all depend on the precision of the state automata representation of the failure behaviour.

5.2.2 The Safora automatic generation and synthesis of TFTs

Assuming a system which is embedded with several complex interrelated components, each modelled by its own SM; the idea of Safora makes it possible to construct the system TFTs for a final analysis in a compositional manner. The first step consists of transforming the hierarchy top level SM — i.e., a highly abstract description of the monolithic behaviour of the system — to produce a set of preliminary TFTs (see Figure 5—6).

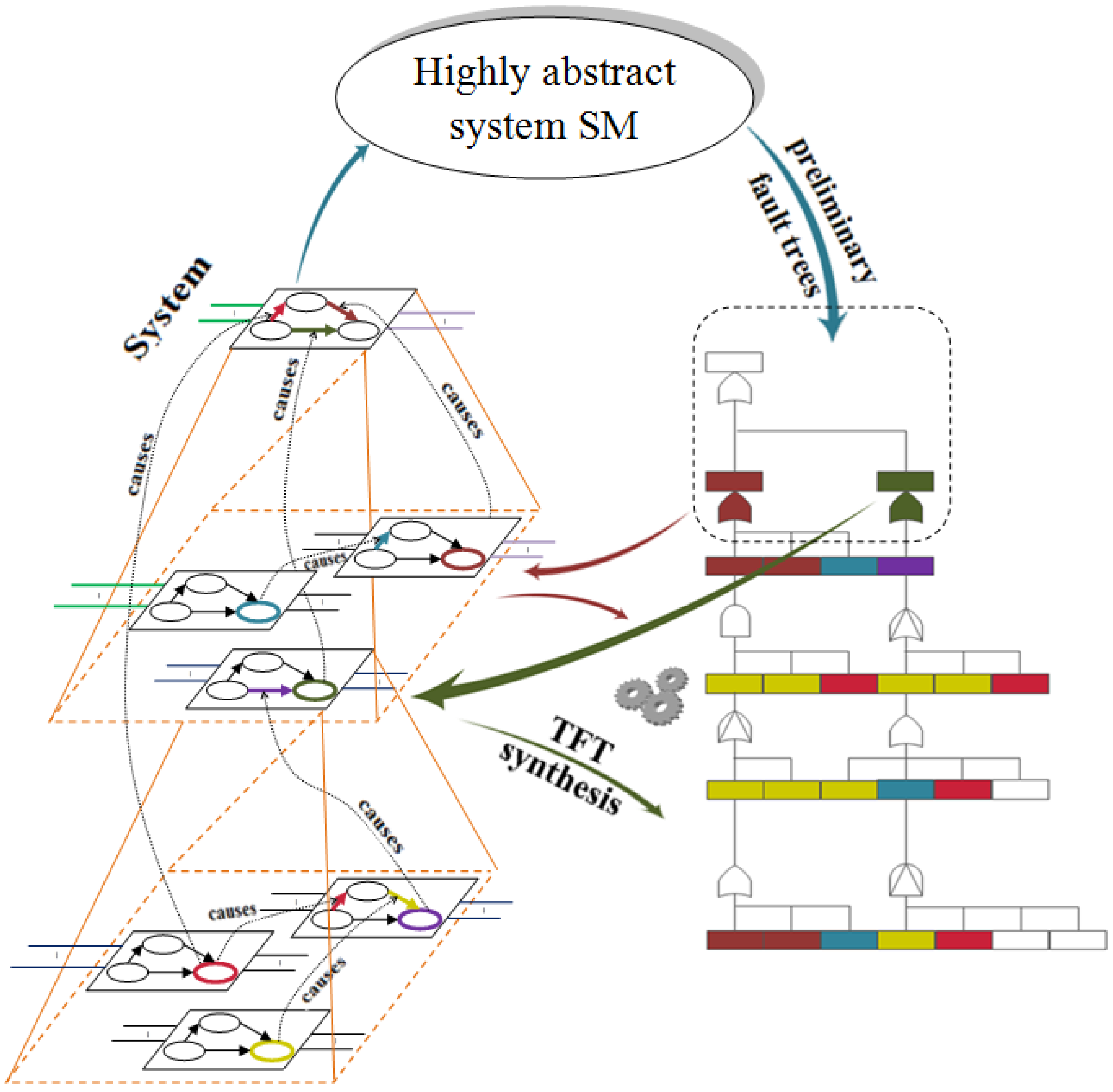


Figure 5—6. Overview of the Safora method.

Algorithm 1 takes the top level SM as parameter. Then, for each final state (line 3), it generates the corresponding failure expression (line 4) of a preliminary system TFT (the definitions used are the same definitions[94] of chapter 4, section 4.2.4). Ultimately, the algorithm produces a set of system failure expressions (line 7); each corresponds to the first level of a system TFT.

[94] Briefly, a SM is a quadruple which consists of: S (a finite set of states), Σ (a finite set of events), δ (a function: $S \times \Sigma \rightarrow S$ s.t. for $(u, u') \in S^2$ and $e \in \Sigma$, $u' = \delta(u, e)$ iff e is incident from u to u', i.e., $u \xrightarrow{e} u'$) and s_0 (the initial state). F is a subset of S containing the final states of the SM.

Algorithm 1 : Φ_{syst} GenerateSystFailureTFTs(SM$_{\mathrm{syst}}$)

Input: A highly abstract SM$_{\mathrm{syst}}$=(S, Σ, δ, s_0) describing the monolithic behaviour of the system.

Output: A set Φ_{syst} of preliminary system failure TFTs.

```
1: let $\Phi_{syst} = \emptyset$
2: get F from S
3: for each s∈F do
4:     let $\phi_s$ = GenerateFailureExpression(SM$_{syst}$, s)
5:     let $\Phi_{syst} = \Phi_{syst} \cup \{\phi_s\}$
6: end for
7: return $\Phi_{syst}$
```

The *'GenrateFailureExpression'* method in line 4 of the algorithm takes the system abstract SM as first parameter and a final state for which it generates a failure expression as second parameter. Then, Algorithm 2 generates that failure expression; it is a variation of the two algorithms of chapter 4 section 4.2.4) merged together. It is distinct in the sense that it produces only the failure expression of the designated final state (the second parameter), rather than a set of expressions i.e. one per final state like in chapter 4.

At this stage, since these failure expressions are generated from a highly abstract SM, each typically contains mainly non-atomic (i.e., expandable) events. For example, Figure 5—6 shows the first level of the system fault tree composed of two non-atomic events, each is a fault tree top-event which corresponds to a SM final state of a lower level component — entering one of these final states causes a transition to the final state in the SM upstream, i.e. a complete failure which corresponds to the top-event of the preliminary system FT. Algorithm 3 is developed to synthesise the lower levels of such FTs by expanding[95] every non-atomic event (lines 2 and 3). We would like to emphasise, though, that a non-atomic event is handled indistinguishably in the Safora approach. In that sense, such event can be the effect of a source component SM as well as a sub-component SM being in a final state — both cases are commonly referred to as downstream SMs in the state automata hierarchy of Figure 5—6.

[95] A non-atomic event which represents a failure of a component without a corresponding SM remains undeveloped.

Algorithm 2 : ϕ_s GenerateFailureExpression(SM, s)

Inputs: - A state machine SM = (S, Σ, δ, s_0).
- A final state s of SM, i.e. $s \in F$ $(F \subset S)$.

Output: The failure expression ϕ_s for s.

```
1:  get P from SM
2:  get Ps from P where Ps = $\{\pi \in P |\ s_0 \overset{\pi}{\rightsquigarrow} s\}$
3:  let n = |Ps|
4:  let $\pi_i \in Ps$ $1 \le i \le n$ /* $s_0 \overset{\pi_i}{\rightsquigarrow} s$ */
5:  let $len_i = length(\pi_i)$ $1 \le i \le n$ /* the number of events of $\pi_i$ */
6:  let $Seq_{\pi_i} = \langle s_{i_0}, s_{i_1}, \ldots, s_{i_{len_i}} \rangle$ $s_{i_j} \in S$ $e_{i_j} \in \Sigma$ $1 \le i \le n$ $1 \le j \le len_i$ s.t. $s_{i_j} \xrightarrow{e_{i_j}} s_{i_{(j-1)}}$ where
    $Seq_{\pi_i}$ is ordered from $s_{i_0} = s$ to $s_{i_{len_i}} = s_0$ /* $Seq_{\pi_i}$ $1 \le i \le n$ is the sequence of states
    of $\pi_i$ ordered for backward traversal*/
7:  let $\phi_s = \bigvee_{1 \le i \le n} (\bigwedge_{1 \le j \le len_i} e_{i_j})$ /*$\phi_s$ is the disjunct over paths $\pi$ of the conjunct
    of events of $\pi$*/
8:  for each $s_{i_j}$ in $Seq_{\pi_i}$ $1 \le i \le n$ $1 \le j \le len_i$ s.t. $|\Sigma_{s_{i_j} \blacktriangleright}| > 1$ do
9:      let e = $e_{i_j}$ /* $\Sigma_{s_{i_j} \blacktriangleright}$ is the set of events incident from $s_{i_j}$ */
10:     for each $e' \neq e_{i_j}$ in $\Sigma_{s_{i_j} \blacktriangleright}$ do
11:         let $t \in S$ s.t. $s_{i_j} \xrightarrow{e'} t$
12:         if $((e' \in \{e_{i_{(j-1)}}, e_{i_{(j-2)}}, \ldots, e_{i_1}\})$ or $(\exists (y \in \Sigma, \pi' \in P, f \in S, v \in S$ in $Seq_{\pi'}$, $w \in S$ in
            $Seq_{\pi'})$ s.t. $t \overset{\pi'}{\rightsquigarrow} f$, $v \xrightarrow{y} w$ and $y \in \{e_{i_j}, e_{i_{(j-1)}}, \ldots, e_{i_1}\}))$ then
13:             let e = e|e' /* e' is one of the events that label subpath h of $\pi_i$ s.t. $s_{i_j} \overset{h}{\rightsquigarrow} s$,
                or h and $\pi'$ share an event */
14:         end if
15:     end for
16:     replace $e_{i_j}$ with e in $\phi_s$
17: end for
18: return $\phi_s$
```

In the case of a source connected component SM that is in an affecting[96] final state; this typically represents an output deviation from the source component which corresponds to a matching input deviation into the impacted component — e.g., the final state *'Severely Failed'* of the sensor SM (see Figure 5—1.b) which affects the SM of the backup component B of the PS (see Figure 5—2). In the second case (i.e., a sub-component's SM being in an affecting final state), this typically represents an output deviation from the sub-component which causes the composed component's SM to change its state — e.g.

[96] i.e., it impacts a component upstream.

the final states of the SM of backup B (see Figure 5—2) each causes the change of state to *'Failed'* in the top-level SM of the PS (see Figure 5—3).

Algorithm 3 : FaultTreeSynthesiser(Φ_{syst})

Input: A set Φ_{syst} of preliminary system failure TFTs.
Output: Each system failure fault tree is synthesised. A fault derived from failures in a component which is not modelled (i.e., incomplete state automata) remains undeveloped.

```
1: for each $\varphi \in \Phi_{syst}$ do
2:     for each derived fault $\eta$ in $\varphi$ do
3:         SynthesiseFaultTree($\varphi$, $\eta$)
4:     end for
5: end for
```

The synthesis is done by working backwards, starting with the expressions of the system failures. The input of algorithm 3 is the set of expressions which are generated from a highly abstract system SM (like the top-level SM of Figure 5—6). The outcome consists of synthesised system fault trees, reached by recursively merging the component fault tree expression which corresponds to a non-atomic event into the higher level expression (line 3) — the *'SynthesiseFaultTree'* method is first invoked to expand the non-atomic events of the preliminary failure expressions (like the events which are immediately below the top event of Figure 5—6), then it recursively synthesises them into, ultimately, full-scale system-wide TFTs (algorithm 4).

Algorithm 4 : SynthesiseFaultTree(φ, η)

```
Inputs:  - A failure expression $\varphi$ corresponding to a fault tree.
         - A fault $\eta$ which is a non-atomic event derived from failures in some
           constituent parts or from input deviations.
Output:  The fault trees that cause $\eta$ are merged into a bigger $\varphi$.

1:  if ($\exists$i 1$\leq$i$\leq$n s.t. $SM_i$ generates the output failure $\eta$) /* otherwise $\eta$ remains un-
    developed — n is the totality of the models which compose the state automata,
    some components may not be modelled */ then
2:    let MergedTree = false
3:    for each final state s of $SM_i$ s.t. $\eta$ is the effect of entering s do
4:      let MergedTree = MergedTree $\bigvee$ GenerateFailureExpres-
        sion($SM_i$, s) /* merge the trees into a disjunction since each one can lead
        to $\eta$ */
5:    end for
6:    replace $\eta$ with MergedTree in $\varphi$
7:    for each derived fault $\gamma$ in MergedTree do
8:      SynthesiseFaultTree($\varphi$, $\gamma$) /* should other expandable events $\gamma$ exist in
        the newly merged tree */
9:    end for
10: end if
```

If distinct final states of a component's SM represent the same failure — i.e., an identical output deviation (e.g. omission of output) from that particular component — then the TFTs which correspond to these final states need to be merged, such that each TFT becomes a new branch of a common bigger tree in a disjunctive form; this is because either branch can lead to that same failure.

Therefore, given a failure expression φ and a non-atomic event η of φ, if the *'SynthesiseFaultTree'* method finds the state machine which produces the output failure η (line 1 of algorithm 4), then it generates a disjunction of local TFT expressions — one expression for each final state that represents η (lines 3 through 5). The newly generated TFTs for each component can then be minimised if appropriate to obtain a simplified intermediate form; this helps to remove any redundancies or complexities as early in the process as possible. Next, these TFTs (combined in a disjunctive form) will be merged into φ by substituting η (line 6). The newly merged disjunctive tree can contain both symbols which represent basic events (e.g. internal failures of the component from which it was generated) as well as non-atomic events. In the case of newly introduced non-atomic events γ, the algorithm (lines 7 through 9) recursively synthesises each corresponding TFT into the fault tree represented by φ (assuming the state automata contains the SM which produces γ).

When no more non-atomic events remain in the expression, the system fault tree has been thoroughly synthesised for that particular system failure as all possible substitutions have taken place. At this point, a final analysis takes place to further reduce the cut sequences (or the cut sets, if it contains no temporal logic) and thus determines the minimal sequences or combinations of events that lead to that system failure.

5.3 Case study

5.3.1 A generic triple-module redundant system

To better illustrate how our proposed compositional approach works, we use a Generic Triple-module Redundant (GTR) system, as depicted by Figure 5—7 (originally used in Walker & Papadopoulos, 2006). Components A, B and C are abstract representations of any kind of input, control or actuating device. These are arranged in a redundant series such that A is the primary component which is backed up, in order, by components B and C. S1 and S2 are monitoring sensors designed to detect omission of output from A and B respectively and to activate, in response, the next backup[97] in the series. D is simply an abstraction of the output of the GTR and 'I' represents the input to the system.

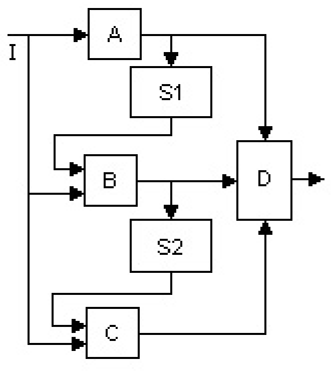


Figure 5—7. GTR system

[97] If S1 detects omission of output from A it should activate B, and if S2 detects omission of output from B it should activate C.

The results of standard FTA performed on the system can be expressed as the following list of MCSs (Walker M. , 2009):

1. Omission of input at I.
2. All of A, B, and C fail.
3. Both A and S1 fail (B will not be activated).
4. All of A, B, and S2 fail (C is not activated).
5. Failure of D.

At first glance, all appear to be correct. If there is no input to the system, then it cannot operate; similarly, if the output component of the system fails, then the system cannot function. If all three main components (A, B, and C) fail, then the system will likewise fail as well. However, MCS #3 and MCS #4 are more complex. These detail situations where the monitoring sensors themselves fail, and are thus unable to detect a failure of the monitored component. In consequence, the next backup cannot be activated and thereby leading to premature system failure.

However, like the system of Figure 5—1.a, the GTR system exhibits dynamic behaviour: its true failure behaviour depends on the chronology of events. The system can function in any of three modes – with A active, B active, or C active – and the transition between those modes is triggered by omission failures detected by the monitors S1 and S2. Different sequences of failure events can lead to the system failing in different ways, and not all of them are correctly represented by the results.

For example, assume that B fails first, and then A fails second; in this case, sensor S1 will not be able to activate B when it detects omission of output from A, and sensor S2 will not activate C because B was never activated — it can only detect an omission of output once its monitored component is activated. Thus the sequence "B fails before A fails" will cause the system to fail regardless of the status of C and S2. This means that MCSs #2 and #4 are unduly optimistic — in certain cases, the system will fail without all of the events in those MCSs having to occur.

Similarly, S1 failing before A means that component B will not be activated upon omission of output from A, since the monitor is not operational and thus unable to detect the omission. Again, this means that the system can fail irrespective of the state of S2 and C. However, if S1 fails *after* A, then it has no effect on the system: it has already served its purpose and activated B, so unless there are further failures elsewhere, the system will continue to operate. Thus MCS #3 can be unnecessarily pessimistic in this case. The same is also true of the relationship between B and S2 — if S2 fails after it has detected omission of B, then it has no effect on the system.

Clearly, standard combinatorial fault trees do not always produce accurate results for even simple dynamic systems like this one. In this example, we have seen that although FTA suggests a failure of all of A, B, and C is necessary to cause system failure, a failure of B before A is sufficient, giving a false sense of security; this information, had it been known, may have resulted in a different system design, e.g. a triple voter or some other different redundancy architecture.

To remedy this problem, we show how we can generate the TFTs from the SMs of the individual components. Thereafter, we demonstrate the Safora approach by building the system TFT (in this case there is only one — i.e., total failure of the GTR system as top event) and then performing the post-synthesis temporal qualitative analysis on it — some temporal logical reductions may be possible to perform during the synthesis of the component TFTs.

5.3.2 State automata of the GTR

The highly abstract description of the monolithic failure behaviour of the GTR system is depicted by Figure 5—8. It corresponds to the top-level SM of the state automata hierarchy (see system level of Figure 5—5. Compositional *modelling of the system failure behaviour.*) — the remaining SMs in this section correspond to the components level of Figure 5—5 (each one of these represents the failure behaviour of a component). The system is initially active (state ‘ON’); but it totally fails if there is omission of output from component D (i.e., ‘O-D’) or omission of input to the system (i.e., ‘O-I’).

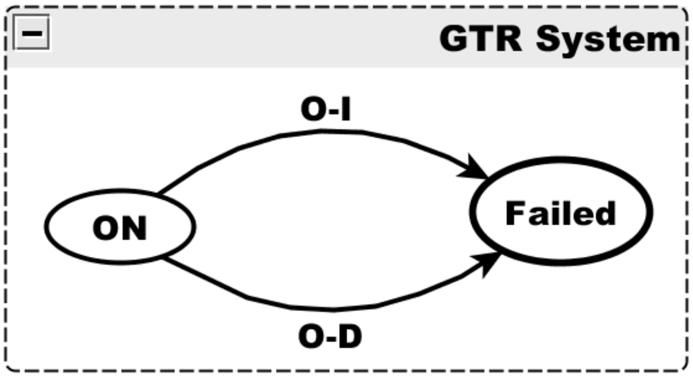


Figure 5—8. Abstract SM of the GTR

Despite the fact that ‘I’ is input to all three components A, B and C (see Figure 5—7 of the GTR), ‘O-I’ is represented at the system level rather than repeated in each component, as can be seen in Figure 5—8. This is because ‘O-I’ affects the whole GTR (it is a single point of failure for the system). Thus, the SM of component A (Figure 5—9) has simply one path from ‘ON’ to ‘Failed’, with the effect of omission of output from A (‘O-A’). The state transition of A is assumed to be triggered only by a basic event of the component (e.g. an internal failure ‘A fails’), since a lack of input is represented by O-I at the system level.

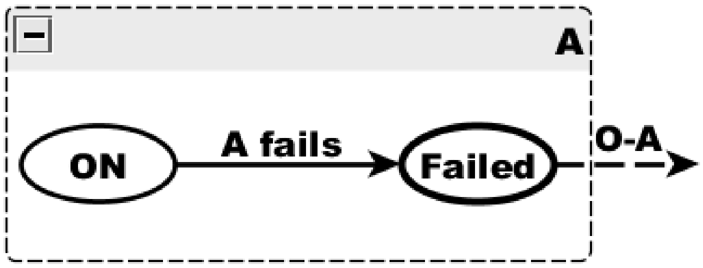


Figure 5—9. SM of A

Sensors S1 and S2 have similar jobs (which is to activate the next backup if omission of output is detected from the monitored component). There is, therefore, potential for reuse of the sensor SM from Figure 5—1 (b), with ‘O-A’ replaced by an input failure from A or B and ‘S fails’ becoming ‘S1 fails’ or ‘S2 fails’ as appropriate. The state machine for S1 is shown in Figure 5—10; it is identical to Figure 5—1 (b) except that ‘S1 fails’ and ‘O-S1’ replace ‘S fails’ and ‘O-S’, respectively.

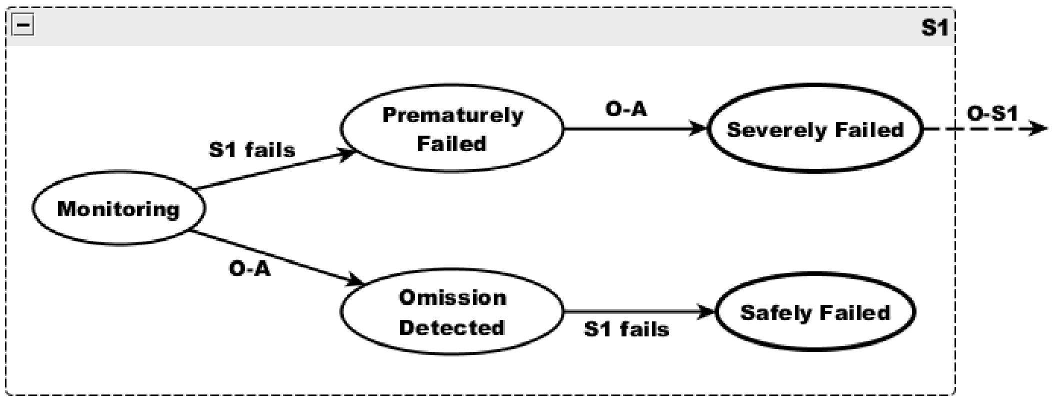


Figure 5—10. SM of S1

However, for the SM of S2 (shown in Figure 5—11), we need to distinguish between two types of omission of output from B. ‘O-B Not Severe’ is a detectable omission of output from B and occurs when B has been activated by S1 upon failure of A and then subsequently fails; it is the effect of entering state ‘Safely Failed’ in the SM of B (see Figure 5—12) and allows S2 to activate C, assuming S2 itself has not yet failed. The other kind of omission is ‘O-B Severe’, an undetectable omission of B caused either by B failing dormant or S1 failing prematurely; either case means that S1 is unable to activate B upon failure of A. This omission is the effect of B entering the ‘Permanently OFF’ or ‘Severely Failed’ states, as seen in Figure 5—12.

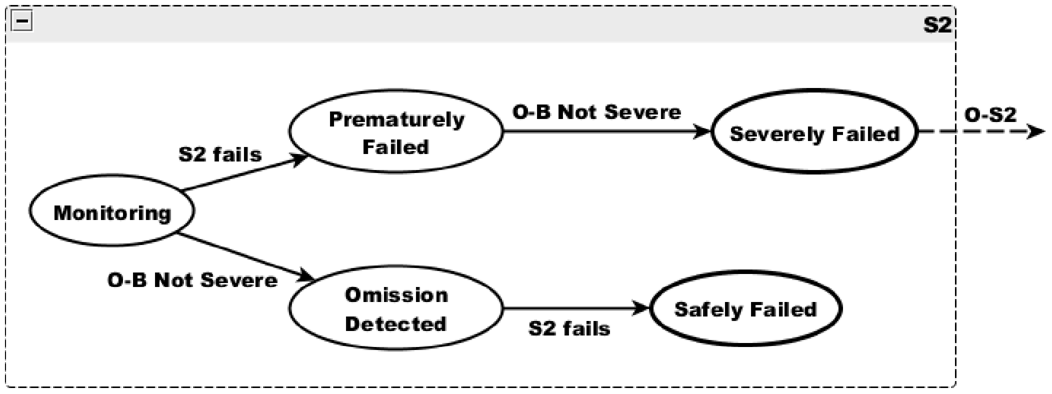


Figure 5—11. SM of S2

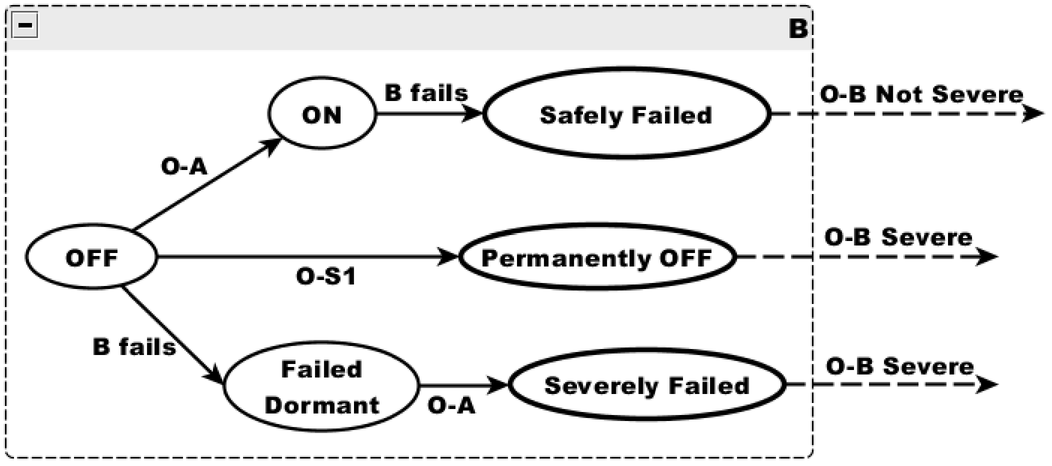


Figure 5—12. SM of B

As can be seen from Figure 5—14, 'O-B Severe' will lead the system to a total failure by causing an omission of output from D ('O-D'). This in turn alters the state of the GTR system to 'Failed' (i.e., complete failure of the system) as shown in Figure 5—8. The SM of D also shows an additional cause for O-D: the basic event 'D fails'. This is the only other single point of failure for the system.

For component C (SM of Figure 5—13), there are two final states, both with omission of output from C (O-C) as common effect. The state 'Permanently OFF' means that C will never be activated and is caused by O-S2 (i.e. S2 has prematurely failed and both A and B have failed in sequence). The other final state is a state where A and B have failed in sequence and where C has also failed (caused by the event 'C fails'). Omission from C is the third cause of failure for output component D, leading to O-D, and thereby causing a total failure of the system.

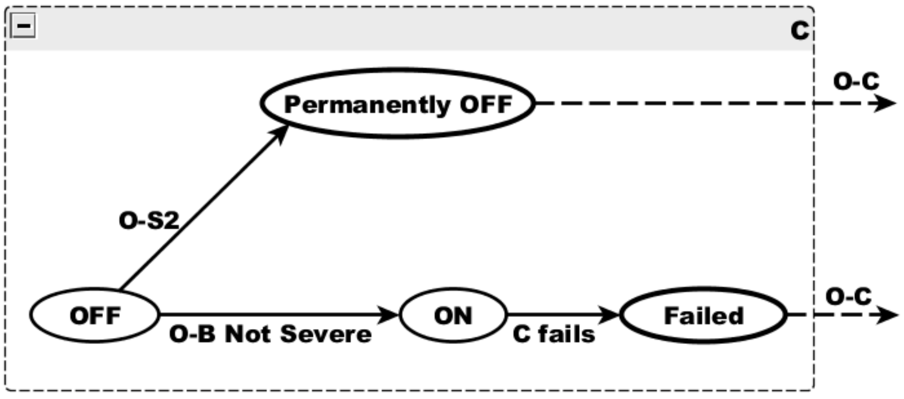


Figure 5—13. SM of C

Note that A and D are each initially in state 'ON' — i.e. the GTR system is initially working with its primary component, and the output is being delivered by the system. Backup components have their initial states set to 'OFF' and sensors are initially set to 'Monitoring'.

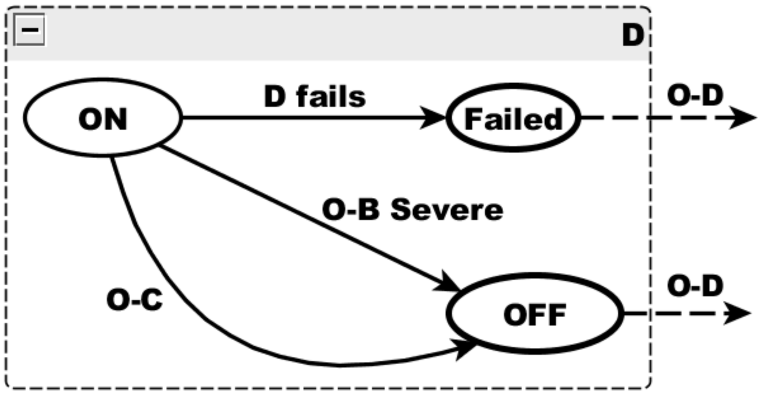


Figure 5—14. SM of D

5.3.3 Synthesis and analysis of TFTs

At this stage, we are ready to start with the highly abstract SM of the GTR, to generate a set of preliminary system fault trees — one TFT per final state in general (there is only one final state in the SM of the GTR in particular). The first level of the GTR system fault tree is represented as follows (by application of algorithms 1 then 2 of section 5.2.2 "The Safora automatic generation and synthesis of TFTs"):

$$\text{Failed}_{\text{(of the GTR system)}} = \text{O-I} + \text{O-D}$$

No logical reduction is possible here at this stage. Thereafter, the application of algorithm 3 (in the same section as above) on the system failure expression will call the method '*SynthesiseFaultTree*' to expand the non-atomic event 'O-D'. This is done by searching in the state automata model for the state machine that produces 'O-D' as effect of entering, at least, one of its final states.

Thereafter, by application of the algorithm of the method '*SynthesiseFaultTree*' (i.e., algorithm 4) on the SM of component D[98], we generate a disjunction of failure expressions; each corresponds to a final state that represents 'O-D' (lines 3 to 5).

[98] We assume that all necessary state machines exist in the Safora model (i.e., we have a complete state automata). If, for example, the SM of D is not part of the model, then 'O-D' will be treated like an atomic event (i.e., it remains as it is and no further work needs to be done, see algorithm 4 line 1) — this is the case of 'O-I' as it represents an input deviation which cannot be expanded any further.

Therefore, we have:

From SM of D:

For the sake of clarity, 'D' is an abbreviation of 'D fails' in the expression.

- Failed$_{\text{(of component D)}}$ = D
- OFF = O-C + O-B Severe

Similarly, later on in this section we use the same notation for all the internal failures of the components — i.e., 'A' is an abbreviation of 'A fails', 'B' is an abbreviation of 'B fails'…etc. The failure expression leading to 'O-D' is the disjunction of the above expressions leading to the final states "Failed" and "OFF". Therefore, we have:

O-D = D + O-C + O-B Severe

Again no logical reduction of the expression is possible at this stage. So, we proceed with merging (line 6) the fault tree with top event 'O-D' into the first level of the system fault tree to obtain the following result in the next level:

Failed = O-I + D + O-C + O-B Severe

The inserted fault tree expression leading to the top event 'O-D' has introduced new non-atomic events into the expanded system expression. These events also represent each a top event of a component's (temporal) fault tree. Therefore, their corresponding expressions need to be merged into the system expression. Algorithm 4 iterates over these non-atomic events (lines 7 through 9), recursively calls the *'SynthesiseFaultTree'* method for each one of these (line 8) in order to determine the corresponding local failure expression (lines 3 through 5), and then synthesise each generated expression into the system failure expression (line 6). Thus, the system fault tree becomes gradually more expanded since, at each iteration, a failure expression local to a component is generated from the component's SM and then synthesised into the system expression.

For example, the newly introduced expandable events into the system failure expression of the second level are 'O-B Severe' and 'O-C', each is represented by two final states of the SM producing it — the states "Permanently OFF" and "Severely Failed" of the SM of B each represents 'O-B Severe' (see Figure 5—12), and similarly the states "Permanently OFF" and "Failed" of the SM of C each represents 'O-C' (see Figure 5—13). Thus, each top event will be replaced with the disjunction of the failure expressions of the final states representing it. Lines 3 through 5 of algorithm 4 generate the below failure expressions through the method *‘GenerateFailureExpression’* (described in algorithm 2) to construct the fault trees leading to the output deviations ‘O-B Severe’ and ‘O-C’.

From SM of B:

- Permanently OFF = O-S1
- Severely Failed = O-A.(B|O-A)
  (equivalent to B<O-A)

Therefore, the TFT which leads to ‘O-B Severe’ is represented by the below disjunction of failure expressions:

O-B Severe = O-S1 + O-A.B|O-A

Some initial minimisation of this failure expression can take place using the temporal law (A|B).B $\Leftrightarrow$ A<B.

Thus, we have now:

O-B Severe = O-S1 + B<O-A

As a result, the following is a more expanded system failure expression, and which corresponds to a temporal fault tree (the use of a temporal gate is necessary):

Failed = O-I + D + O-C + O-S1 + B<O-A

‘O-A’ and ‘O-S1’ are now the most recently introduced non-atomic events and the synthesis algorithm is a depth first traversal and expansion of the fault tree.

Therefore, we have:

From SM of A:

O-A = $\text{Failed}_{(\text{of component A})}$ = A

Therefore, the system failure expression becomes as follows:

Failed = O-I + D + O-C + O-S1 + B<A

<u>From SM of S1:</u>

O-S1 = Severely Failed$_{\text{(of the sensor S1)}}$ = O-A.S1|O-A

Again, we can use the law (A|B).B ⇔ A<B to apply an initial minimisation on the failure expression of the sensor. Thus, we now have:

O-S1 = S1<O-A

Therefore, we have:

Failed = O-I + D + O-C + S1<O-A + B<A

Since the substitution of 'O-S1' with its corresponding expression has introduced 'O-A' again, the algorithm synthesises once more the expression leading to 'O-A' into the system expression. Thus, we have now:

Failed = O-I + D + O-C + S1<A + B<A

At this stage, the remaining non-atomic event is O-C — it was introduced after synthesising the fault tree of component D into the system-wide fault tree (through the synthesis of the corresponding expressions).

Therefore,

<u>From SM of C:</u>

- Permanently OFF = O-S2
- Failed = C.(O-B Not Severe)

The fault tree which leads to O-C is thus the below disjunction of the two expressions:

O-C = O-S2 + C.(O-B Not Severe)

A more expanded system failure expression is, thus, as follows:

Failed = O-I + D + O-S2 + C.(O-B Not Severe) + S1<A + B<A

Obviously, at this level of the system TFT synthesis, new non-atomic events were also introduced through the local failure expression of component C which leads to 'O-C', namely 'O-S2' and 'O-B Not Severe'. Omission from sensor S2 ('O-S2') is caused by a severe failure of the sensor which has a TFT expression similar to that one of sensor S1, but according to the output omission from its monitored component (see section 5.3.2 about the sensor S2).

Therefore,

<u>From SM of S2:</u>

O-S2 = Severely Failed$_{\text{(of the sensor S2)}}$ = (O-B Not Severe).S2|(O-B Not Severe)

Similarly, the expression can be minimised to become as follows:

O-S2 = S2<(O-B Not Severe)

The synthesis of the failure expression of 'O-S2' into the system failure expression gives:

Failed = O-I + D + S2<(O-B Not Severe) + C.(O-B Not Severe) + S1<A + B<A

However, the local failure expression of the sensor has also introduced the non-atomic event 'O-B Not Severe'.

Therefore:

<u>From SM of B:</u>

O-B Not Severe = Safely Failed = B.O-A|B

The expression can also be logically reduced to become as follows (using the same temporal law as in the previous reduction (A|B).B ⇔ A<B):

O-B Not Severe = O-A<B

This gives a more expanded system failure expression by substituting the first occurrence of 'O-B Not Severe' with its minimised failure expression, such as:

Failed = O-I + D + S2<(O-B Not Severe) + C.(O-A<B) + S1<A + B<A

Thereafter, the newly introduced 'O-A' will be substituted with 'A fails' (abbreviated A), and hence:

Failed = O-I + D + S2<(O-B Not Severe) + C.(A<B) + S1<A + B<A

Similarly, substituting the second occurrence of ‘O-B Not Severe’ gives:

Failed = O-I + D + S2<(O-A<B) + C.(A<B) + S1<A + B<A

Then we synthesise the expression of the last occurrence of ‘O-A’ into the system expression, and hence:

Failed = O-I + D + S2<(A<B) + C.(A<B) + S1<A + B<A

Despite the substitution of all non-atomic events, the system failure expression can be expanded further. The cut-sequence ‘S2<(A<B)’ shows that there is no order between ‘S2’ and ‘A’. Indeed, ‘S2<(A<B)’ is true if ‘S2’ occurs before the PAND gate with the two inputs ‘A’ before ‘B’ evaluates to true. Therefore, we can expand it into a disjunctive form of other cut-sequences, each with a possible order in which the events can occur while preserving the same overall outcome, and hence:

S2<(A<B) = S2<A<B + A<S2<B

In the two possibilities of events order (i.e., ‘S2<A<B’ and ‘A<S2<B’), ‘S2’ occurs before ‘B’ and then also before ‘A<B’ evaluates to true. Therefore, the final expansion of the system failure expression gives:

Failed = O-I + D + S2<A<B + A<S2<B + C.(A<B) + S1<A + B<A

Figure 5—15 summarises the synthesis process of the system TFT through a depth first expansion of the non-atomic events. The figure (which is a graphical representation of the system failure expression) clearly shows the below results contrasted against the results obtained from FTA (see Table 5—1).

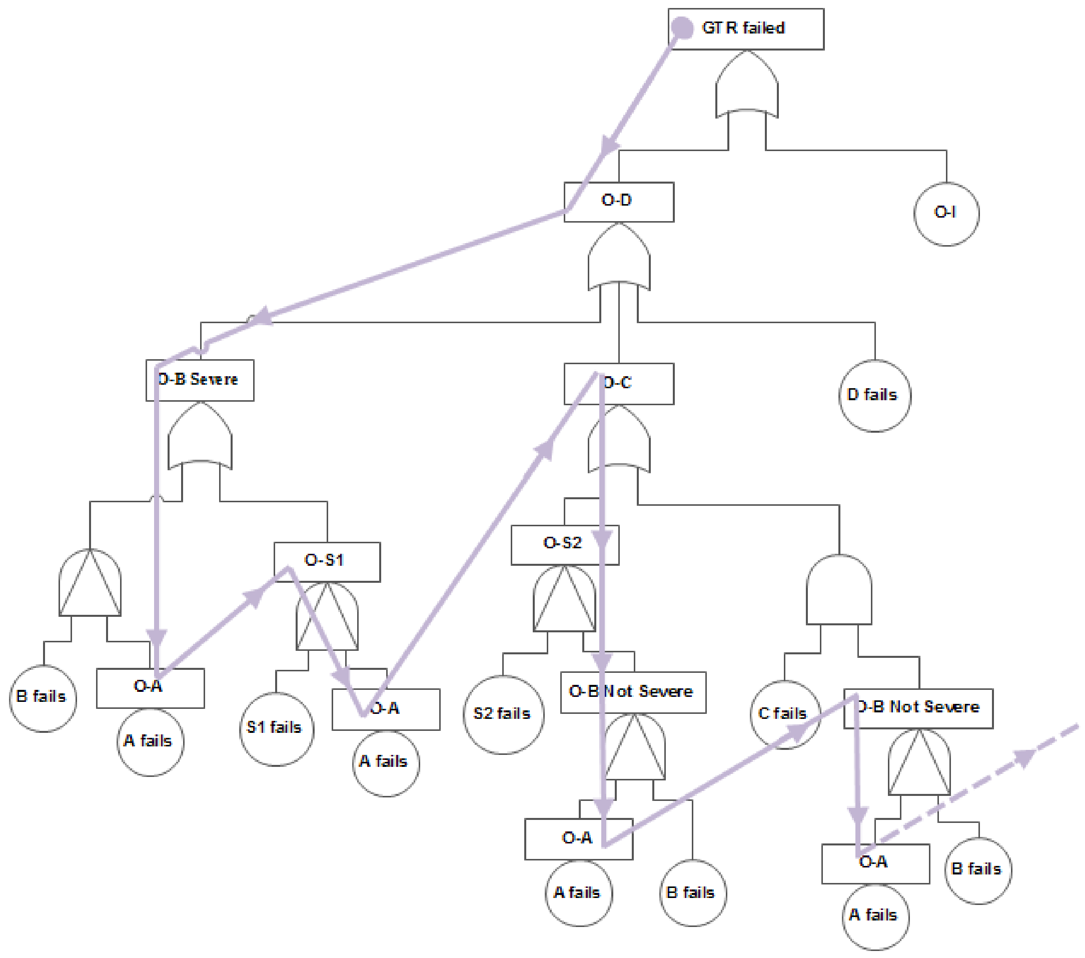


Figure 5—15. Synthesized TFT of the GTR (depth first expansion of non-atomic events)

Table 5—1. Comparison with FTA results

FTA minimal cut sets (Walker M. , 2009)	**Safora generated cut-sequences (minimised using Pandora)**
Omission of input at I	*Idem*
Failure of D	*Idem*
All of A, B, and C fail	B<A
	C.A<B

Both A and S1 fail	S1<A
All of A, B, and S2 fail	S2<A<B
	A<S2<B

While one of the FTA results suggests all three components A, B and C need to fail for a total failure, the expressions which are generated by the Safora technique (and minimised using Pandora) show it as a dangerously optimistic assumption. Indeed, the system can fail for less than that — i.e., a total failure does not require C to fail if B fails before A; C is required to fail otherwise. Another FTA result, which suggests both A and S1 need to fail for a complete failure of the system, seems to be overly pessimistic. Our method shows that the combination of failures of S1 and A is order sensitive — i.e., while S1 failing before A is relevant to the system failure behaviour, the opposite is still a safe situation. Finally, the Safora technique also shows that the combination of failures of A, B and S2 is order sensitive. The combination is relevant to the system failure behaviour irrespective of the order in which A and S2 fail, but each needs to fail before B.

5.3.4 Comparative evaluation

This part is aimed at evaluating the Safora technique by comparing it against the approach where the system is directly modelled with TFTs for a subsequent Pandora qualitative analysis. If the system was originally and entirely modelled as a TFT instead of using component SMs, we may have arrived at a fault tree expression such as the following (Mahmud *et al.*, 2012):

Failed =	D +	(A)
	O-I +	(B)
	S1<(A+O-I) +	(C)
	(B + O-I)<(A+O-I) +	(D)
	S2<((A + O-I)<B + (A + O-I)<O-I) +	(E)

C . ((A + O-I)<B + (A + O-I)<O-I) (F)

In this approach, minimisation of the system failure expression is considerably more complex. (A) and (B) are each a single point of failure, the remaining expressions can be analysed as follows:

(C) <u>S1<(A+O-I):</u>

1. S1<(A+O-I)
2. (S1|A).(S1|O-I).(A+O-I)
3. (S1|A).(S1|O-I).A + (S1|A).(S1|O-I).O-I
4. (S1<A).(S1|O-I) + (S1|A).(S1<O-I)
5. S1<A|O-I + S1<O-I<A + S1<O-I|A + S1<A<O-I
6. S1<A|O-I + S1<O-I|A + S1<A<O-I
7. S1<A|O-I + S1<O-I|A

(D) <u>(B + O-I)<(A+O-I):</u>

Using the temporal law (X + Y)<Z ⇔ X<Z + Y<Z, The cut-sequence (D) can be expressed as a disjunctive form B<(A + O-I) + O-I<(A + O-I), where B<(A + O-I) is similar to (C) and thus can be substituted in the same way with B<A|O-I + B<O-I|A, while O-I<(A + O-I) contains a contradiction and will be, therefore, eliminated:

1. O-I<(A+O-I)
2. (O-I|A).(O-I|O-I).(A+O-I)
3. (O-I|A).0.(A+O-I)
4. 0

(E) S2<((A + O-I)<B + (A + O-I)<O-I):

(E) is more complex, and hence more difficult to reduce:

1. S2<((A + O-I)<B + (A + O-I)<O-I)
2. S2<(A<B + O-I<B + A<O-I + O-I<O-I)
3. S2<(A<B + O-I<B + A<O-I + 0)
4. S2<(A<B + O-I<B + A<O-I)
5. S2|(A<B) . S2|(O-I<B + A<O-I) . (A<B + O-I<B + A<O-I)
6. S2|(A<B) . S2|(O-I<B + A<O-I) . A<B + S2|(A<B). S2|(O-I<B + A<O-I) . (O-I<B + A<O-I)
7. S2|(A<B) . S2|(O-I<B + A<O-I) . A<B + S2|(A<B). S2|(O-I<B) . S2|(A<O-I) . (O-I<B + A<O-I)
8. S2|(A<B) . S2|(O-I<B + A<O-I) . A<B + S2|(A<B) . S2|(O-I<B) . S2|(A<O-I) . O-I<B + S2|(A<B) . S2|(O-I<B) . S2|(A<O-I) . A<O-I
9. S2|(A<B) . S2|(O-I<B) . S2|(A<O-I) . A<B + S2|(A<B) . S2|(O-I<B) . S2|(A<O-I) . O-I<B + S2|(A<B) . S2|(O-I<B) . S2|(A<O-I) . A<O-I
10. S2<(A<B) . S2|(O-I<B) . S2|(A<O-I) + S2|(A<B) . S2<(O-I<B) . S2|(A<O-I) + S2|(A<B) . S2|(O-I<B). S2<(A<O-I)
11. O-I<S2<A<B + S2<O-I<A<B + S2<A<O-I<B + S2<A<B|O-I + O-I<A<S2<B + A<S2<O-I<B + A<S2<B|O-I + A<S2<O-I<B + S2<A<O-I<B + S2<O-I<A<B + S2<O-I<B|A + O-I<A<S2<B + O-I<S2<A<B + O-I<S2<B|A + B<S2<A<O-I + S2<B<A<O-I + S2<A<B<O-I + S2<A<O-I|B + B<A<S2<O-I + A<S2<B<O-I + A<S2<O-I|B

(F) C . ((A + O-I)<B + (A + O-I)<O-I):

This cut-sequence is relatively easier to reduce and as follows:

1. C.((A + O-I)<B + (A + O-I)<O-I)

2. C.(A<B + O-I<B + A<O-I + O-I<O-I)
3. C.(A<B + O-I<B + A<O-I + 0)
4. C.(A<B + O-I<B + A<O-I)
5. C.A<B + C.O-I<B + C.A<O-I

Therefore, the system failure expression is the disjunction of the following cut-sequences:

1. D
2. O-I
3. S1<A|O-I
4. S1<O-I|A
5. B<A|O-I
6. B<O-I|A
7. O-I<S2<A<B
8. S2<O-I<A<B
9. S2<A<O-I<B
10. S2<A<B|O-I
11. O-I<A<S2<B
12. A<S2<O-I<B
13. A<S2<B|O-I
14. A<S2<O-I<B
15. S2<A<O-I<B
16. S2<O-I<A<B
17. S2<O-I<B|A
18. O-I<A<S2<B

19. O-I<S2<A<B

20. O-I<S2<B|A

21. B<S2<A<O-I

22. S2<B<A<O-I

23. S2<A<B<O-I

24. S2<A<O-I|B

25. B<A<S2<O-I

26. A<S2<B<O-I

27. A<S2<O-I|B

28. C.A<B

29. C.O-I<B

30. C.A<O-I

At this stage of the minimisation, the individual cut-sequences have been each logically reduced independently. We can also minimise the cut-sequences against each other. For example, we can apply the rule A|B+B ⇔ A+B to several cut-sequences (e.g. cut-sequences 2, 3 and 5) and the Boolean absorption laws A<B + A ⇔ A and A<B + B ⇔ B to many others (such as 4 and 6). This gives the following final list of seven minimal cut-sequences:

1. D

2. O-I

3. S1< A

4. B< A

5. S2<A<B

6. A<S2<B

7. C.A<B

These results are the same as those obtained using the Safora method, but they are much more complex and they required more steps and thus more time and effort to achieve them.

5.4 Discussion

We have presented a method for the compositional synthesis of TFTs from SMs — each SM describes the failure behaviour of a component, except one which is a highly abstract description of the monolithic failure behaviour of the system. As for the analysis, it can be performed at the component level wherever possible — i.e., logical reduction of the local TFT which is generated from a component's SM before its synthesis into the system TFT. This makes the entire process (i.e., failure modelling, analysis and synthesis) substantially compositional and potentially more scalable; as large-scale system TFTs are synthesised from smaller component TFTs.

The technique works backwards, starting with the first level of each system TFT (i.e., total failures as top events) generated from the abstract system SM, and then recursively generates the TFTs from the SMs of the components which cause the top events. Next, these local TFTs are analysed as much as possible then merged into their corresponding system-wide TFTs. A final analysis takes place once we obtain a fully expanded system TFT (or set of TFTs) in order to determine the root causes of the failures. The TFTs are generated, synthesised and analysed without sacrificing any information about the dynamic behaviour of the system.

A central issue of the approach is the compositional modelling of the failure behaviour. Firstly, this helps to better manage the complexity involved in the transformations from SMs to TFTs — the worst case complexity of the conversion algorithm is $O(n^2)$, n being the number of paths from the initial state to the final states in the SM (see chapter 4, section 4.3). Secondly, the hierarchical modelling of the failure behaviour also simplifies the task of the system designers, allowing them to focus on modelling local failures in each component. Thirdly, the generated TFTs are correspondingly much smaller and then simpler to analyse and to synthesise into system TFTs.

Concerning the analysis outcome, the accuracy of the results relies strongly on the preciseness of the failure behavioural models (i.e., the state automata); though, this not uncommon in dependability modelling and analysis — e.g., the Markov differential solutions depend chiefly on the Markov state diagrams or on whichever dependability models that the Markov chains are checked against (e.g. DFTs).

The technique we have proposed is simpler to understand from both a modelling and analysis perspective and is considerably more scalable than using TFTs directly. We have used a dual-purpose case study based on a generic triple redundant system. First, we have shown how the approach enables a more accurate and full analysis of an increasingly complex dynamic system. Second, we have evaluated the scaling benefits (like clarity and ease of analysis) of using compositional SMs for the initial dynamic modelling by contrast with a direct modelling of the system with a full-scale TFT. Another important issue of the proposed method is the ability to combine it with a compositional analysis technique like HiP-HOPS, which would enable automatic synthesis and analysis of TFTs from complex models of systems in which the error behaviour is described with state automata.

6 Brake-By-Wire case study

This brake-by-wire (BBW) example system is a supplemental case study based mainly on the research prototype used in (Walker, Papadopoulos *et al.*, 2009). The aim is to apply the Safora method in a similar manner as for the GTR example system in order to re-confirm the benefits of the approach. For this purpose, it is sufficient to consider the BBW components which are relevant to the study — i.e., principally the vehicle dynamics, actuators, sensors and the communication bus (see Figure 6—1). The more realistic BBW system is very complex and since the purpose of the study is not to manipulate a huge number of components with their corresponding fault trees; we need therefore to abstract the system to the level shown in Figure 6—1.

The BBW of the figure operates using one actuator and one rotation sensor at each wheel of the vehicle. The actuators are controlled by two electronic control units (ECUs) through a dual communication bus. The latter carries signals from the sensors towards the ECUs and braking commands back from the ECUs through the comparator to the actuators — the comparator forwards the braking commands only in case of identical output from both ECUs. The remaining component is the 'Vehicle Dynamics' which abstracts the output of the system. In that sense, this component describes the different braking effects which are relevant to the BBW system failure behaviour — we put emphasis on braking failures with catastrophic and critical effects as shown in Figure 6—2, and which describes the monolithic failure behaviour of the system.

6.1 Compositional modelling of the BBW failure data

Initially, the BBW system is operating normally — i.e., the corresponding SM is in the "Regular" initial state (see Figure 6—2). Thereafter, we consider only the input deviations that lead the system to either a catastrophic or a critical situation — i.e., the deviations 'Dev-CatastrophicWheelLock' and 'Dev-CriticalWheelLock' respectively. Any input deviation that causes a moderate or a marginal system failure will not be considered at the top level, and thus such a deviation is not part of the highly abstract description of the system monolithic failure behaviour, as represented by Figure 6—2.

‘Dev-CatastrophicWheelLock’ is the effect of entering the states (of the ‘Vehicle Dynamics’ SM, see Figure 6—3) and where: a) both front brakes[99] fail irrespective of the order or at the same time, b) one front and one rear brake both on the same side (either the driver or the passenger side) fail simultaneously, c) the two front brakes and the rear one behind the driver fail at the same time, d) the two rear brakes and the front brake on the driver’s side fail simultaneously and finally e) all four brakes of the vehicle

[99] In Figure 6—3, commission failures of the actuators label the SM transitions — an actuator commission directly causes a failure of the corresponding brake.

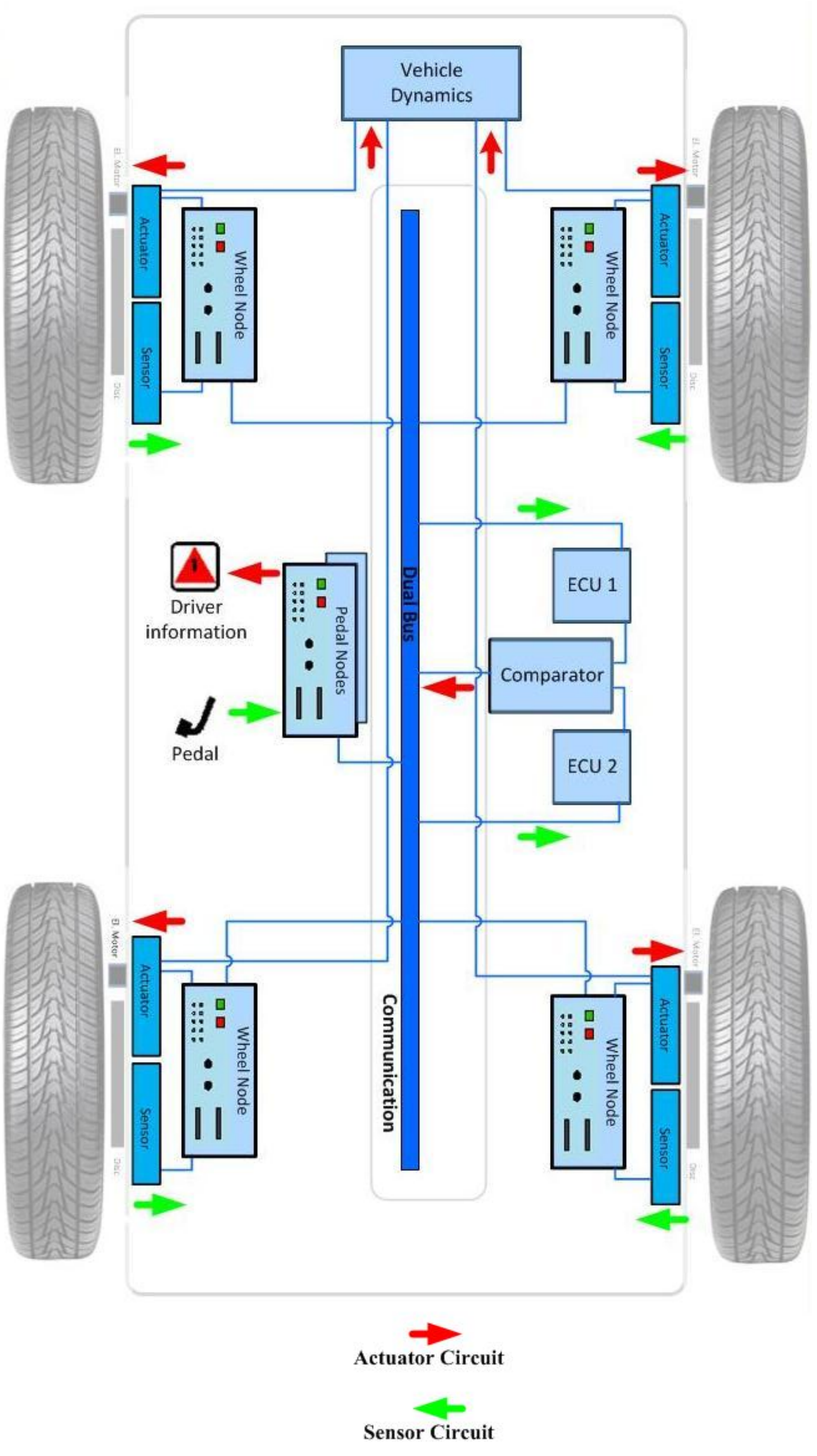


Figure 6—1. Architecture of the brake-by-wire system[100]

[100] Compiled from Maenad (http://www.maenad.eu, 2012 — Torchiaro *et al.*, 2011), Atesst2 (http://www.atesst.org, 2010), (Walker, Papadopoulos *et al.*, 2009) and (Papadopoulos Y. , 2000).

fail at the same time. The other relevant output deviation of the ‘Vehicle Dynamics’ SM is the ‘Dev-CriticalWheelLock’ which is the effect of entering the states where: a) anyone of the front brakes fails followed by a failure of the rear brake of the same side — i.e., either both on the left or on the right side of the vehicle, b) both rear brakes fail irrespective of the order, but not at the same time — which has a moderate effect rather than critical, finally c) a simultaneous failure of the front brake on the driver’s side with the rear brake on the passenger’s side (i.e., opposite to the front one).

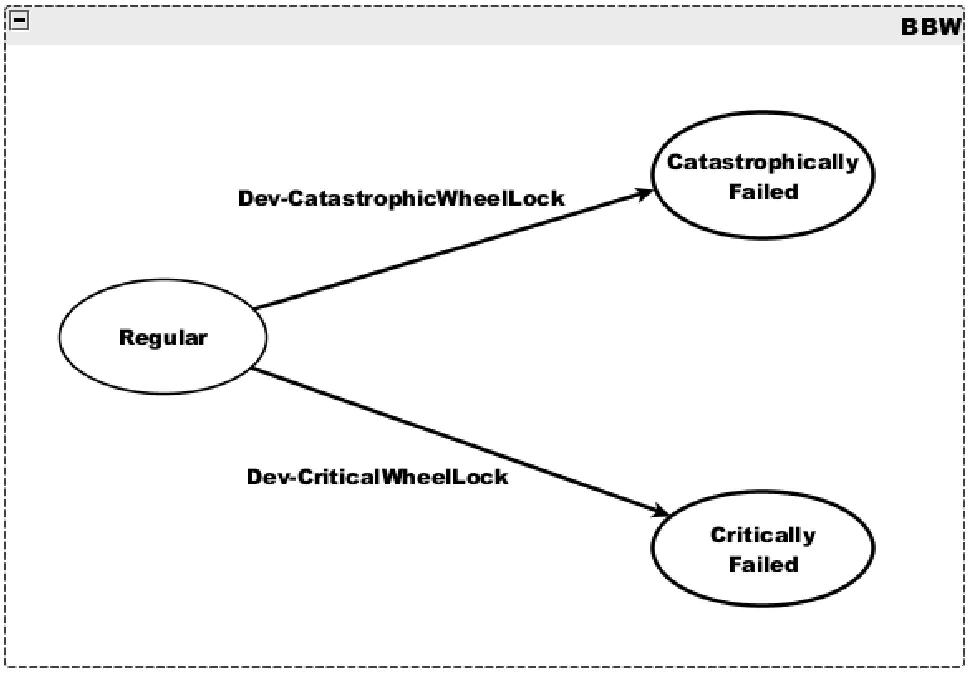


Figure 6—2. A highly abstract BBW SM

The events and state names of Figure 6—3 have suffixes like ‘FD’, ‘FP’, ‘RD’, ‘RP’ which respectively mean “**F**ront on the **D**river’s side”, “**F**ront on the **P**assenger’s side”, “**R**ear on the **D**river’s side (i.e., behind the driver)” and finally “**R**ear on the **P**assenger’s side (i.e., behind the front passenger)”. The reason of this specification (rather than using ‘Right’ and ‘Left’) is to make the case study valid for vehicles manufactured to be used in countries like the UK (where vehicles drive on the right) or for vehicles manufactured to be used in countries like France or the US.

Table 6—1 (which forms the foundation of the 'Vehicle Dynamics' SM) shows the different sequences of braking failures (actuator commissions) with their corresponding effects on the vehicle. We assume, though, that simultaneous failures are due to common failure modes, and which cause separate[101] transitions in the corresponding SM. Therefore, when it comes to simultaneous failures we express these by preceding the suffixes with 'Diag' (two failures at the same time of diagonally opposed actuators) or 'Trig' (three failures at the same time of actuators forming a triangle). For example, the actuator commission failure 'C-ActuatorDiagD' represents a common cause failure by which 'C-ActuatorFD' and 'C-ActuatorRP' occur simultaneously — the prefix 'C-' is to express a commission output deviation and this applies to all the BBW components' SMs. The letter 'D' which follows 'Diag' means that the diagonal line first extremity is the front actuator on the **D**river's side, the other extremity being necessarily (since the line is diagonal) the rear actuator on the passenger's side — the other diagonally opposed actuators are at the front on the **P**assenger's side and at the rear behind the driver, thus 'C-ActuatorDiagP'[102] would represent a common cause failure by which 'C-ActuatorFP' and 'C-ActuatorRD' occur simultaneously.

[101] A similar approach can be found in Markov state diagrams where transitions due to common failure modes are often labelled with separate failure rates like in (ARP4761, 1996).

[102] This failure (i.e., 'C-ActuatorDiagP') is omitted in the study since it causes less critical effects (an acceptable equilibrium of the vehicle can be regained by the locking of the opposite wheel) and does not impact the sequencing of the other failures. However, 'C-ActuatorDiagD' is considered in the study due to the critical effects it may cause since the vehicle veers plausibly from the side into oncoming traffic before equilibrium can be regained by the locking of the opposite wheel.

Table 6—1 also shows commission failures represented like:

‘C-ActuatorD’: causes a simultaneous failure of the front brake on the **D**river’s side and the rear brake behind it on the same side.

‘C-ActuatorP’: causes a simultaneous failure of the front brake on the **P**assenger’s side and the rear brake behind it on the same side.

‘C-ActuatorF’: causes a simultaneous failure of the two **F**ront brakes.

‘C-ActuatorR’: causes a simultaneous failure of the two **R**ear brakes.

The dynamics of the BBW system are highly complex in the automotive industry. We have, therefore, simplified the models and particularly the failure behavioural description of the “Vehicle Dynamics” component to the level presented in Figure 6—3. Complexity analysis of the conversion algorithm is provided in chapter 4 (section 4.3). Thus, it is not intended in this case study to demonstrate the conversions of large scale SMs and to produce huge amounts of complex TFTs. The main purpose is, by contrast, to emphasise and to analyse some relevant sequencing of faults.

In order to limit the complexity of the study, we only focus on sequences of no more than two (strictly ordered) actuator commissions — i.e., sequences of up to two braking failures — and as seen in Figure 6—3. We assume that sequences of three braking failures or more (the four brakes fail in order) are due to common failure modes by which the corresponding wheels actually lock at the same time. Moreover, in the reality it is very rare to observe more than two wheels locking in sequence; but like in Figure 6—3, we treat such situations (even though rare) as simultaneous locking of wheels by representing these as separate events in the vehicle dynamics’ SM — i.e., same time commissions of the corresponding actuators.

Indeed, we turn our attention to the more critical effects of common failure modes (by which some wheels may lock simultaneously). In this context, ‘C-ActuatorTrigFD’ represents a simultaneous failure of the three actuators ‘C-ActuatorFD’, ‘C-ActuatorFP’ and ’C-ActuatorRD’ — which form a triangle (abbrev. Trig) with the **F**ront actuator on the **D**river’s side as the summit (suffix FD). Similarly, ‘C-ActuatorTrigRD’ represents a simultaneous failure of the three actuators ‘C-ActuatorRD’, ‘C-ActuatorRP’ and ‘C-ActuatorFD’. Each of those triangles of commissions of actuators includes those which occur on the driver’s side due to their higher criticality — i.e., ‘C-

ActuatorTrigFP'[103] and 'C-ActuatorTrigRP'[104] are omitted from this study, since each one of these includes the failures of the actuators which occur on the Passenger's side and are thus less critical. Finally, 'C-ActuatorAll' represents a simultaneous failure of all actuators which means that the four brakes fail at the same time, and thus locking all wheels at once.

Figure 6—4 represents the failure behaviour of an actuator. This one executes the ECU command by applying braking pressure on the corresponding wheel (there is one actuator for each wheel see Figure 6—1). Each actuator is initially in a 'Regular' state which means that it is functioning properly. The actuator's SM enters the 'Inaccurate' state due to a commission failure represented by the event 'ActCommission'[105] (i.e., the actuator applies a braking pressure without being commanded to do so) or due to a false braking signal (e.g., an incorrect command to brake) received from the bus, and which is represented by the event 'C-BusCommand'.

---

[103] A common failure mode by which 'C-ActuatorFP', 'C-ActuatorFD' and 'C-ActuatorRP' fail simultaneously.

[104] Same as the previous but for the simultaneous failure of 'C-ActuatorRP', 'C-ActuatorRD' and 'C-ActuatorFP'.

[105] An internal failure like in the case where a jam occurs while the actuator is applying a braking pressure, and which therefore persists when the braking is required to stop.

Table 6—1. Effect classification of the vehicle wheel locking — information gathered from (Walker, 2009) and (Walker *et al.*, 2009).

Failure	**Effect**	**Severity**
either C-ActuatorFD or C-ActuatorFP	Average loss of stability and control with manageable consequences.	***Moderate***
C-ActuatorRD (resp. C-ActuatorRP)	Low loss of stability which can be regained by managing braking through C-ActuatorFP (resp. C-ActuatorFD).	***Marginal***
C-ActuatorFD < C-ActuatorRD	High loss of stability and control with possible severe consequences — narrow margin to manoeuvre.	***Critical***
C-ActuatorFP < C-ActuatorRP	High loss of stability and control — the vehicle veers plausibly towards the driver's side (i.e., possible severe consequences).	***Critical***
C-ActuatorRD < C-ActuatorFD	Average loss of stability with better possibility of manoeuvring.	***Moderate***
C-ActuatorRP < C-ActuatorFP	Average loss of stability with better possibility of manoeuvring.	***Moderate***
C-ActuatorD (i.e., C-ActuatorFD & C-ActuatorRD)	High loss of stability and control with limited possibilities of manoeuvring — the vehicle veers plausibly towards the driver's side.	***Catastrophic***
C-ActuatorP (i.e., C-ActuatorFP & C-ActuatorRP)	High loss of stability and control with very limited possibility of manoeuvring (whether automatically or manually).	***Catastrophic***

C-ActuatorFD < C-ActuatorRP	Average loss of control and the vehicle can regain stability through the locking of opposite wheels — better possibilities of manoeuvring.	***Moderate***
C-ActuatorRP < C-ActuatorFD	Low loss of stability which can be regained through the locking of opposite wheels — good possibilities of manoeuvring.	***Marginal***
C-ActuatorDiagD (i.e., C-ActuatorFD & C-ActuatorRP)	High loss of stability and control — locking of opposite wheels offers, however, some margin to manoeuvre.	***Critical***
C-ActuatorFD < C-ActuatorFP (resp. C-ActuatorFP < C-ActuatorFD)	Locking of the two front wheels causes severe braking and loss of control — the vehicle veers plausibly to the driver's side (resp. passenger's side).	***Catastrophic***
C-ActuatorF (i.e., C-ActuatorFD & C-ActuatorFP)	Locking of the two front wheels causes severe braking and loss of control — the vehicle veers less severely due to simultaneous brake locking.	***Catastrophic***
C-ActuatorRD < C-ActuatorRP (resp. C-ActuatorRP < C-ActuatorRD)	Better margin to manoeuvre with locked rear wheels than front — the vehicle veers plausibly to the driver's side (resp. passenger's side).	***Critical***
C-ActuatorR (i.e., C-ActuatorRD & C-ActuatorRP)	Simultaneous locking of rear wheels is more manageable than front — better potential to regain stability.	***Moderate***
C-ActuatorTrigFD (i.e., C-ActuatorFD & C-ActuatorFP &C-ActuatorRD)	Severe loss of stability — very difficult to manoeuvre.	***Catastrophic***

C-ActuatorTrigRD (i.e., C-ActuatorRD & C-ActuatorRP &C-ActuatorFD)	Severe loss of stability — very difficult to manoeuvre.	***Catastrophic***
C-ActuatorAll (i.e., C-ActuatorFD & C-ActuatorFP & C-ActuatorRD & C-ActuatorRP)	Severe loss of stability — very difficult to manoeuvre.	***Catastrophic***

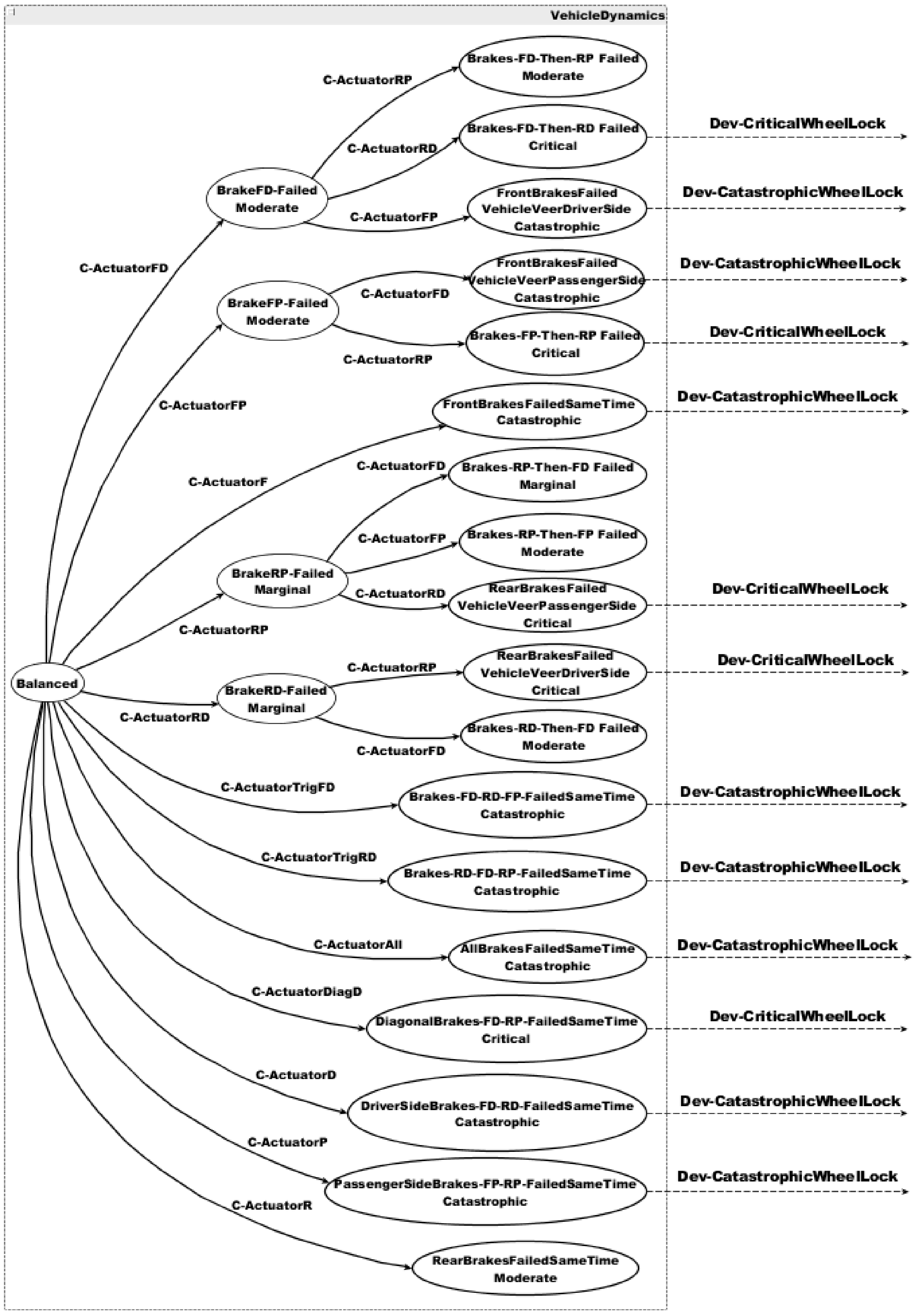

VehicleDynamics
Balanced
C-ActuatorFD
BrakeFD-Failed Moderate
C-ActuatorRP
Brakes-FD-Then-RP Failed Moderate
C-ActuatorRD
Brakes-FD-Then-RD Failed Critical
Dev-CriticalWheelLock
C-ActuatorFP
FrontBrakesFailed VehicleVeerDriverSide Catastrophic
Dev-CatastrophicWheelLock
C-ActuatorFP
BrakeFP-Failed Moderate
C-ActuatorFD
FrontBrakesFailed VehicleVeerPassengerSide Catastrophic
Dev-CatastrophicWheelLock
C-ActuatorRP
Brakes-FP-Then-RP Failed Critical
Dev-CriticalWheelLock
C-ActuatorF
FrontBrakesFailedSameTime Catastrophic
Dev-CatastrophicWheelLock
C-ActuatorRP
BrakeRP-Failed Marginal
C-ActuatorFD
Brakes-RP-Then-FD Failed Marginal
C-ActuatorFP
Brakes-RP-Then-FP Failed Moderate
C-ActuatorRD
RearBrakesFailed VehicleVeerPassengerSide Critical
Dev-CriticalWheelLock
C-ActuatorRD
BrakeRD-Failed Marginal
C-ActuatorRP
RearBrakesFailed VehicleVeerDriverSide Critical
Dev-CriticalWheelLock
C-ActuatorFD
Brakes-RD-Then-FD Failed Moderate
C-ActuatorTrigFD
Brakes-FD-RD-FP-FailedSameTime Catastrophic
Dev-CatastrophicWheelLock
C-ActuatorTrigRD
Brakes-RD-FD-RP-FailedSameTime Catastrophic
Dev-CatastrophicWheelLock
C-ActuatorAll
AllBrakesFailedSameTime Catastrophic
Dev-CatastrophicWheelLock
C-ActuatorDiagD
DiagonalBrakes-FD-RP-FailedSameTime Critical
Dev-CriticalWheelLock
C-ActuatorD
DriverSideBrakes-FD-RD-FailedSameTime Catastrophic
Dev-CatastrophicWheelLock
C-ActuatorP
PassengerSideBrakes-FP-RP-FailedSameTime Catastrophic
Dev-CatastrophicWheelLock
C-ActuatorR
RearBrakesFailedSameTime Moderate

Figure 6—3. Vehicle Dynamics SM

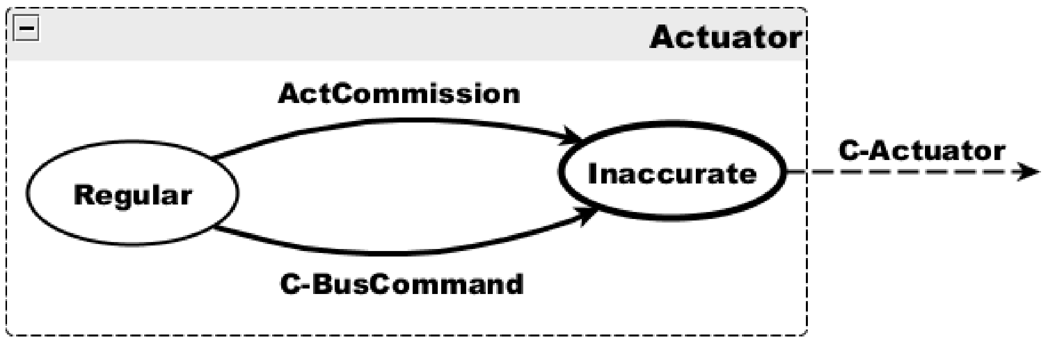


Figure 6—4. Actuator SM

Similarly, the SM of the bus (see Figure 6—5) is initially in the 'Regular' state and can transit to either the "Command Incorrect" or the "Data Incorrect" state. The former transition is due to an incorrect command to brake (i.e., 'C-BrakeCommand') which is propagated from the ECUs through the bus to the actuators. The latter is due to an internal failure of the bus (i.e., 'BusCommission') — e.g., electromagnetic interference or memory bits that are stuck in the bus. Corrupted data is plausibly also due to incorrect sensor data (i.e., 'C-SensorData') since the bus is a dual communication means — i.e., communicating data from the sensors to the ECUs and then brake commands from the ECUs to the actuators.

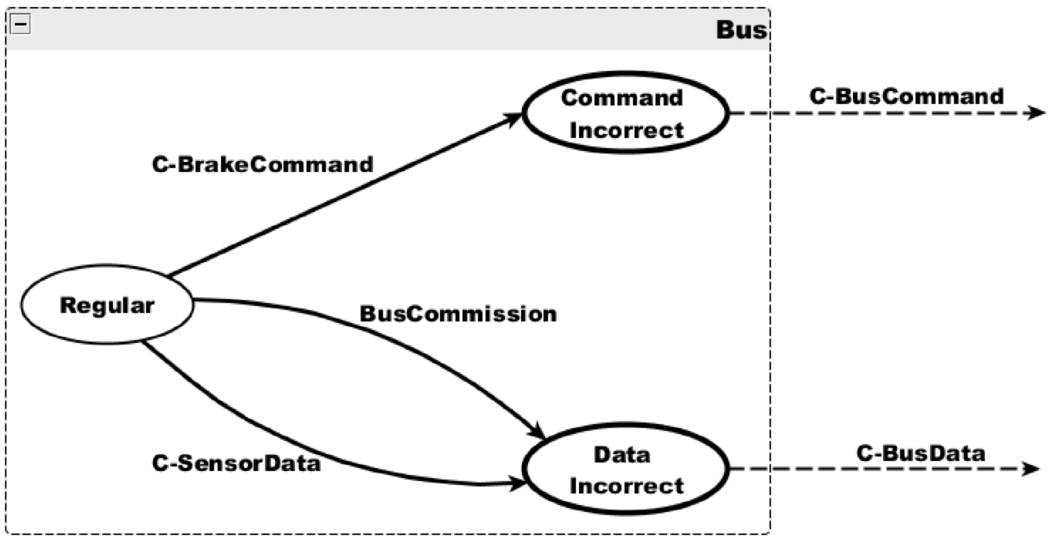


Figure 6—5. Bus SM

The comparator whose failure behaviour is described in Figure 6—6 forwards the brake commands from the two ECUs to the actuators only if the commands are identical. This is to prevent accidental braking signals generated from one ECU to cause undesired wheel locking. Therefore, an incorrect braking command must be duplicated for its propagation to the actuators (as seen in Figure 6—6 where both twin C-ECUs are required to enter the state 'Propagating').

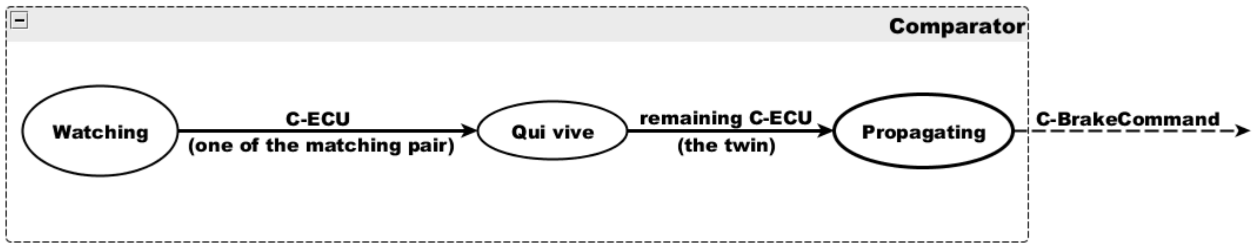


Figure 6—6. Comparator SM

Concerning the ECU, the corresponding SM transits from the state 'Sound' to the state 'Unsound' only due to corrupted data received through the communication bus (i.e., 'C-BusData'). We assume that both ECUs do not fail internally since these often recover quickly from their own hardware or software malfunctions.

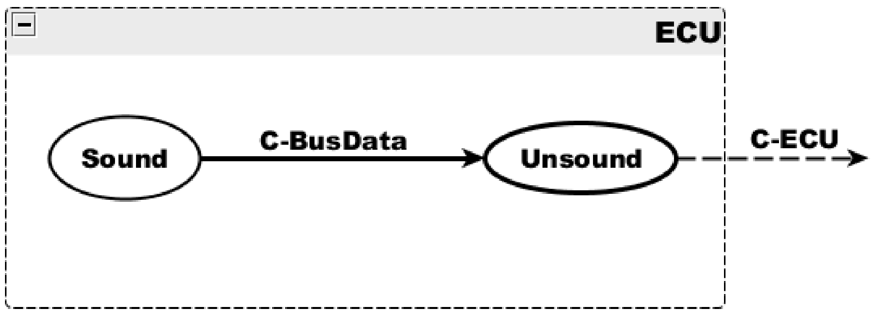


Figure 6—7. ECU SM

Figure 6—8 describes the SM of a sensor. Each sensor measures the rotation speed of the corresponding wheel, it then supplies the ECUs with the related data based on which braking commands may be produced. The ECUs command to brake if the sensor data indicates that the corresponding wheel is turning too fast or may jeopardise the control of the vehicle. The sensors may fail completely; but this is a situation which

can be detected by the ECUs since these would not receive any sensor data after a pre-determined period of time. For the purpose of this case study, however, we only consider sensor high-bias internal failures (i.e., ‘SenCommission’) that cause high value errors. These sensor errors lead to an interpretation that the rotation speed of the corresponding wheel is too high, and hence causing braking commands to be made by the ECUs.

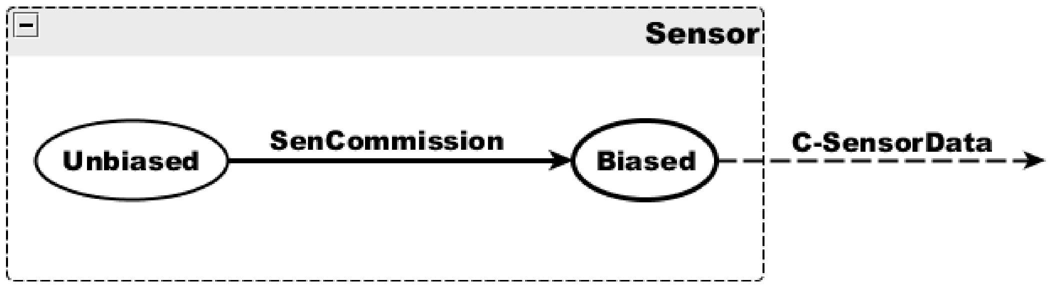


Figure 6—8. Sensor SM

The failure behaviour of each sensor (resp. actuator) needs to be described only once since the sensors (resp. the actuators) are all identical. Similarly, representing the SM of one ECU is sufficient since both ECUs are the same.

6.2 Compositional synthesis of the BBW system fault trees

The application of the Safora method (see chapter 5 section 5.2.2) to the BBW compositional SMs starts with the conversion of the abstract system’s SM to generate a set of preliminary system FTs like:

“Catastrophically Failed” = Dev-CatastrophicWheelLock

“Critically Failed” = Dev- CriticalWheelLock

The relevant system failure states of the study, i.e., “Catastrophically Failed” and “Critically Failed” are respectively reached through the input deviations ‘Dev-CatastrophicWheelLock’ and ‘Dev- CriticalWheelLock’. The corresponding matching output deviations can be expanded by applying the conversion algorithm on the ‘Vehicle Dynamics’ SM (see Figure 6—3).

The failure expression of the first output deviation (i.e., 'Dev-CatastrophicWheelLock') is thus as follows:

Dev-CatastrophicWheelLock =

C-ActuatorFP.C-ActuatorFD|C-ActuatorRD|C-ActuatorFP|C-ActuatorRP +

C-ActuatorFD.C-ActuatorFP|C-ActuatorRD|C-ActuatorRP|C-ActuatorFD +

C-ActuatorF +

C-ActuatorTrigFD +

C-ActuatorTrigRD +

C-ActuatorAll +

C-ActuatorD +

C-ActuatorP

Some minimisation is possible at this first level using the law of POR transformation like:

$$B . A|B \Leftrightarrow A<B$$

Therefore, the system failure expression of 'Dev-CatastrophicWheelLock' becomes as follows:

Dev-CatastrophicWheelLock =

C-ActuatorFD<C-ActuatorFP.(C-ActuatorFD|C-ActuatorRD|C-ActuatorRP) + **(1)**

C-ActuatorFP <C-ActuatorFD.(C-ActuatorFP|C-ActuatorRD|C-ActuatorRP|) + **(2)**

C-ActuatorF +

C-ActuatorTrigFD +

C-ActuatorTrigRD +

C-ActuatorAll +

C-ActuatorD +

C-ActuatorP

The failure expression of the second output deviation (i.e., ‘Dev- CriticalWheelLock’) is as follows:

Dev-CriticalWheelLock =

C-ActuatorRD.C-ActuatorFD|C-ActuatorRD|C-ActuatorFP|C-ActuatorRP +

C-ActuatorRP.C-ActuatorFP|C-ActuatorRD|C-ActuatorRP|C-ActuatorFD +

C-ActuatorRD.C-ActuatorRP|C-ActuatorRD|C-ActuatorFP|C-ActuatorFD +

C-ActuatorRP.C-ActuatorRD|C-ActuatorFP|C-ActuatorRP|C-ActuatorFD +

C-ActuatorDiagD

Like the first minimisation which has been performed on the ‘Dev-CatastrophicWheelLock’ branch, we can also reduce ‘Dev-CriticalWheelLock’ at this level using the POR transformation mentioned earlier. This gives:

Dev-CriticalWheelLock =

C-ActuatorFD<C-ActuatorRD.(C-ActuatorFD|C-ActuatorFP|C-ActuatorRP) + **(3)**

C-ActuatorFP<C-ActuatorRP.(C-ActuatorFP|C-ActuatorRD|C-ActuatorFD) + **(4)**

C-ActuatorRP<C-ActuatorRD.(C-ActuatorRP|C-ActuatorFP|C-ActuatorFD) + **(5)**

C-ActuatorRD<C-ActuatorRP.(C-ActuatorRD|C-ActuatorFP|C-ActuatorFD) + **(6)**

C-ActuatorDiagD

The two expressions of ‘Dev-CatastrophicWheelLock’ and ‘Dev-CriticalWheelLock’ can be further reduced using some information produced by the conversion of the SM representing the “Vehicle Dynamics” to TFTs. The conversion also produces sequences of failures which have moderate or marginal effects; we can merge these results like in:

Dev-Moderate-OR-MarginalWheelLock =

C-ActuatorRP.C-ActuatorFD|C-ActuatorRD|C-ActuatorFP|C-ActuatorRP +

C-ActuatorFD.C-ActuatorRP|C-ActuatorRD|C-ActuatorFP|C-ActuatorFD +

C-ActuatorFP.C-ActuatorRP|C-ActuatorRD|C-ActuatorFP|C-ActuatorFD +

C-ActuatorFD.C-ActuatorRD|C-ActuatorFP|C-ActuatorRP|C-ActuatorFD +

C-ActuatorR

This branch, even though irrelevant as suggested in the study, can complete the picture to make a global view of the effects of the sequencing of the actuator commissions or, as consequences, unwanted locking of wheels. It can be minimised using the POR transformation as well like in the following:

Dev-Moderate-OR-MarginalWheelLock =

C-ActuatorFD<C-ActuatorRP.(C-ActuatorFD|C-ActuatorRD|C-ActuatorFP) + **(7)**

C-ActuatorRP<C-ActuatorFD.(C-ActuatorRP|C-ActuatorRD|C-ActuatorFP) + **(8)**

C-ActuatorRP<C-ActuatorFP.(C-ActuatorRP|C-ActuatorRD|C-ActuatorFD) + **(9)**

C-ActuatorRD<C-ActuatorFD.(C-ActuatorRD|C-ActuatorFP|C-ActuatorRP) + **(10)**

C-ActuatorR

By applying a temporal extension law to (1), we have the different order possibilities for the occurrence of the events ‘C-ActuatorRD’ and ‘C-ActuatorRP’ as follows:

C-ActuatorFD< C-ActuatorRD<C-ActuatorRP<C-ActuatorFP + **(1.1)**

C-ActuatorFD< C-ActuatorRP<C-ActuatorRD<C-ActuatorFP + **(1.2)**

C-ActuatorFD<C-ActuatorFP<C-ActuatorRD<C-ActuatorRP + **(1.3)**

C-ActuatorFD<C-ActuatorFP<C-ActuatorRP<C-ActuatorRD + **(1.4)**

C-ActuatorFD<C-ActuatorRD<C-ActuatorFP<C-ActuatorRP + **(1.5)**

C-ActuatorFD<C-ActuatorRP<C-ActuatorFP<C-ActuatorRD + **(1.6)**

C-ActuatorFD< C-ActuatorRD<C-ActuatorFP.(C-ActuatorFD|C-ActuatorRP) + **(1.7)**

C-ActuatorFD<C-ActuatorFP<C-ActuatorRD.(C-ActuatorFD|C-ActuatorRP) + **(1.8)**

C-ActuatorFD< C-ActuatorRP<C-ActuatorFP.(C-ActuatorFD|C-ActuatorRD) + **(1.9)**

C-ActuatorFD<C-ActuatorFP<C-ActuatorRP.(C-ActuatorFD|C-ActuatorRD) + **(1.10)**

C-ActuatorFD<C-ActuatorFP.(C-ActuatorFD|C-ActuatorRD|C-ActuatorRP) **(1.11)**

However, by conformance to the hypothesis made earlier, i.e., sequences of up to two failures — or simultaneous failures otherwise; the extension of (1) can be expressed as follows:

C-ActuatorFD< C-ActuatorRD + **(1.1)**

C-ActuatorFD< C-ActuatorRP + **(1.2)**

C-ActuatorFD<C-ActuatorFP + **(1.3)**

C-ActuatorFD<C-ActuatorFP.(C-ActuatorFD|C-ActuatorRD|C-ActuatorRP) **(1.4)**

'C-ActuatorRD', as one of the POR optional input events, should not occur before 'C-ActuatorFD'; otherwise the effect would be moderate and not catastrophic (see Figure 6—3 and branch 10 of 'Dev-Moderate-OR-MarginalWheelLock'). Similarly, 'C-ActuatorRP' (as another POR optional input event) should not occur before 'C-ActuatorFD'; otherwise the effect would be marginal (see Figure 6—3 and branch 8). Therefore, we can express all the failure sequence possibilities (with respect to the hypothesis) as follows:

C-ActuatorFD< C-ActuatorRD + **(1.1)**

C-ActuatorFD< C-ActuatorRP + **(1.2)**

C-ActuatorFD<C-ActuatorFP **(1.3)**

At this stage of the minimisation of (1), the branch (1.1) produces a critical effect and not catastrophic. It would, thus, be absorbed through the expansion of branch (3) of 'Dev-CriticalWheelLock'. Moreover, (1.2) produces a moderate and not a catastrophic effect (see Figure 6—3). It would, thus, be absorbed through the expansion of (7) of 'Dev-Moderate-OR-MarginalWheelLock'. This means that (1) can be reduced to simply (1.3), i.e., C-ActuatorFD<C-ActuatorFP. In a similar way, we can reduce (2) to C-ActuatorFP<C-ActuatorFD and thus, together with (1.3), would minimise to C-ActuatorFD . C-ActuatorFP (their simultaneous occurrence is treated as a separate event, i.e., 'C-ActuatorF'). Therefore, the order in which the front brakes fail is not relevant to

the catastrophic effect. It is, however, relevant in the case where we like to speculate about the veer side of the vehicle, but such thing is more conceptual and then requires us to preserve the corresponding states separate (as in Figure 6—3) rather than unifying their effects under the 'Dev-CatastrophicWheelLock' output deviation.

By proceeding in a similar manner, we can minimise the expressions of 'Dev-CriticalWheelLock' to get the following results:

Dev-CriticalWheelLock =

C-ActuatorFD<C-ActuatorRD + **(3)**

C-ActuatorFP<C-ActuatorRP + **(4)**

C-ActuatorRP<C-ActuatorRD + **(5)**

C-ActuatorRD<C-ActuatorRP + **(6)**

C-ActuatorDiagD

Thereafter, we can merge (5) and (6) using the logical AND gate — this should not be interpreted as the common failure mode by which the rear brakes fail simultaneously, and which has a moderate effect instead. This gives therefore:

Dev-CriticalWheelLock =

C-ActuatorFD<C-ActuatorRD + **(3)**

C-ActuatorFP<C-ActuatorRP + **(4)**

C-ActuatorRD . C-ActuatorRP + **(5')**

C-ActuatorDiagD

After minimising as much as possible, both expressions of 'Dev-CatastrophicWheelLock' and 'Dev-CriticalWheelLock' will be synthesised respectively into the "Catastrophically Failed" and "Critically Failed" system fault trees at the next level. At this stage of the synthesis, we have already seen how the Safora method preserves the significance of the temporal order of events through its conversion approach of the SMs to TFTs, and also along the logical reduction of the produced results by using the Pandora temporal laws. Though as stated earlier, we do not intend to produce vast amount of complex TFTs to minimise and synthesise, so we rather select one branch like (4) to compare our results with those in (Walker M. *et al.*, 2009). At the level of the branch (4), we need to expand the 'C-ActuatorFP' and 'C-ActuatorRP' actuator commission input deviations, where each needs to be substituted with the failure expression of the matching output deviation from the corresponding actuator. Since the actuators are all identical and with the same failure behaviour (described in Figure 6—4), we need to apply the conversion algorithm only once to the actuator's SM and then, at a later stage, we designate to which wheel the expression is by using the suffix representations mentioned earlier (idem concerning the sensors). Therefore, we get from the SM of one actuator the following expression:

C-Actuator = ActCommission + C-BusCommand

Thereafter, we get from the SM of the bus the failure expression of the matching output deviation of 'C-BusCommand', and hence:

C-BusCommand = C-BrakeCommand

Its synthesis into the 'C-Actuator' failure expression gives:

C-Actuator = ActCommission + C-BrakeCommand

Similarly the 'C-BrakeCommand' input deviation can be expanded with its corresponding failure expression (of the matching output deviation) by converting the SM of the comparator as follows:

C-BrakeCommand = C-ECU1 . C-ECU2("the twin")

However, in order to propagate 'C-BrakeCommand' the comparator ensures that 'C-ECU1' and 'C-ECU2' are identical. Therefore, some minimisation is possible at this level of the synthesis such that:

C-BrakeCommand = C-ECU1 . C-ECU2

= C-ECU . C-ECU (where 'C-ECU' = 'C-ECU1' = 'C-ECU2')

= C-ECU (using the Boolean Idempotent law, i.e., A .A $\Leftrightarrow$ A)

Thus, the synthesis of the failure expression of 'C-BrakeCommand' into the 'C-Actuator' failure expression gives:

C-Actuator = ActCommission + C-ECU

To extend ‘C-Actuator’ with the failure expression of ‘C-ECU’, we need to apply the conversion algorithm on the SM of the ECU. Therefore, we have:

C-ECU = C-BusData

After synthesis into the C-Actuator failure expression we have:

C-Actuator = ActCommission + C-BusData

At this level, there is a newly introduced input deviation (i.e., ‘C-BusData’) which is derived from failures and/or errors propagated through the bus. The expansion of ‘C-BusData’ by applying the conversion algorithm on the SM of the bus gives:

C-BusData = C-SensorData + BusCommission

After the substitution of ‘C-BusData’ with “C-SensorData + BusCommission” into the failure expression of ‘C-Actuator’ we get:

C-Actuator = ActCommission + (C-SensorData + BusCommission)

The full synthesis of the ‘C-Actuator’ failure expression requires its extension with the failure expression of the ‘sub’-fault-tree whose top-event is ‘C-SensorData’. The conversion of the SM of the sensor gives:

C-SensorData = SenCommission

The last step of the synthesis of the ‘C-Actuator’ failure expression, therefore, gives:

C-Actuator = ActCommission + (SenCommission + BusCommission)

Then, by the Boolean associative law we have:

C-Actuator = ActCommission + SenCommission + BusCommission

These were common Safora steps of SM conversions, minimisation wherever possible and synthesis of produced results. Therefore, we have the following for both actuator commissions of (4):

C-ActuatorFP = ActCommissionFP + SenCommissionFP + BusCommission

And

C-ActuatorRP = ActCommissionRP + SenCommissionRP + BusCommission

The failure of the dual communication bus 'BusCommission' is a common cause failure for all actuators leading to the locking of all wheels.

The cut-sequence (4) — i.e., C-ActuatorFP<C-ActuatorRP — leading to a critical output deviation can , therefore, have its events substituted as follows:

(ActCommissionFP + SenCommissionFP + BusCommission) < (ActCommissionRP + SenCommissionRP + BusCommission)

By using the distributive temporal law, the cut-sequence becomes:

ActCommissionFP < (ActCommissionRP + SenCommissionRP + BusCommission) + **(4.1)**

SenCommissionFP < (ActCommissionRP + SenCommissionRP + BusCommission) + **(4.2)**

BusCommission < (ActCommissionRP + SenCommissionRP + BusCommission) **(4.3)**

(4.3) can be substituted with the following by using the temporal law A<(B+C) ⇔ A|B . A|C . (B+C), and hence:

(4.3) ⇔ (BusCommission|ActCommissionRP) . (BusCommission|SenCommissionRP) . (BusCommission|BusCommission) . (ActCommissionRP + SenCommissionRP + BusCommission)

Therefore, (4.3) can be completely eliminated using first the simultaneity law, i.e. A|A ⇔ 0 and thus the conjunct (BusCommission|BusCommission) ⇔ 0, then second the Boolean law A.0 ⇔ 0, and hence:

(4.3) ⇔ (BusCommission|ActCommissionRP) . (BusCommission|SenCommissionRP) . 0 . (ActCommissionRP + SenCommissionRP + BusCommission)

⇔ 0

Thereafter, using the Boolean law A+0 ⇔ 0, (4) becomes only as follows:

ActCommissionFP < (ActCommissionRP + SenCommissionRP + BusCommission) + **(4.1)**

SenCommissionFP < (ActCommissionRP + SenCommissionRP + BusCommission) + **(4.2)**

At this stage, we can also expand both (4.1) and (4.2) using the same temporal law as before, i.e. A<(B+C) ⇔ A|B . A|C . (B+C). therefore, (4) is equivalent to:

(ActCommissionFP|ActCommissionRP).(ActCommissionFP|SenCommissionRP).
(ActCommissionFP|BusCommission). (ActCommissionRP + SenCommissionRP + BusCommission) + **(4.1)**

(SenCommissionFP|ActCommissionRP).(SenCommissionFP|SenCommissionRP).
(SenCommissionFP|BusCommission). (ActCommissionRP + SenCommissionRP + BusCommission) **(4.2)**

With the Boolean distributive law, i.e. A.(B+C) ⇔ A.B + A.C, each one of (4.1) and (4.2) can be expanded to three disjunctions totalling six for both as follows:

(ActCommissionFP|ActCommissionRP).(ActCommissionFP|SenCommissionRP).
(ActCommissionFP|BusCommission) . ActCommissionRP + **(4.1.1)**

(ActCommissionFP|ActCommissionRP).(ActCommissionFP|SenCommissionRP).
(ActCommissionFP|BusCommission) . SenCommissionRP + **(4.1.2)**

(ActCommissionFP|ActCommissionRP).(ActCommissionFP|SenCommissionRP).
(ActCommissionFP|BusCommission) . BusCommission + **(4.1.3)**

(SenCommissionFP|ActCommissionRP).(SenCommissionFP|SenCommissionRP).
(SenCommissionFP|BusCommission) . ActCommissionRP + **(4.2.1)**

(SenCommissionFP|ActCommissionRP).(SenCommissionFP|SenCommissionRP).
(SenCommissionFP|BusCommission) . SenCommissionRP + **(4.2.2)**

(SenCommissionFP|ActCommissionRP).(SenCommissionFP|SenCommissionRP).
(SenCommissionFP|BusCommission) . BusCommission **(4.2.3)**

Now we can use the temporal law B.A|B ⇔ A<B to minimise as follows:

(ActCommissionFP<ActCommissionRP).(ActCommissionFP|SenCommissionRP).
(ActCommissionFP|BusCommission) + **(4.1.1)**

(ActCommissionFP|ActCommissionRP).**(ActCommissionFP<SenCommissionRP)**.
(ActCommissionFP|BusCommission) + **(4.1.2)**

(ActCommissionFP|ActCommissionRP).(ActCommissionFP|SenCommissionRP).
(ActCommissionFP<BusCommission) + **(4.1.3)**

(SenCommissionFP<ActCommissionRP).(SenCommissionFP|SenCommissionRP).
(SenCommissionFP|BusCommission) + **(4.2.1)**

(SenCommissionFP|ActCommissionRP).**(SenCommissionFP<SenCommissionRP)**.
(SenCommissionFP|BusCommission) + **(4.2.2)**

(SenCommissionFP|ActCommissionRP).(SenCommissionFP|SenCommissionRP).
(SenCommissionFP<BusCommission) **(4.2.3)**

Those results clearly show that for the sequence where the brake at the front on the passenger's side (FP) fail before the brake at the rear same side (RP) — i.e., the output deviations from their corresponding actuators occur in the same order C-ActuatorFP<C-ActuatorRP — one of the following is sufficient to happen (see Figure 6—9):

(a) Internal failures of the corresponding actuators occur in the same order.

(b) Internal failures of the corresponding sensors occur in the same order.

(c) An internal failure of the actuator FP occurs before an internal failure of the sensor RP.

(d) An internal failure of the sensor FP occurs before an internal failure of the actuator RP.

(e) An internal failure of the actuator FP occurs before the bus fails.

(f) An internal failure of the sensor FP occurs before the bus fails.

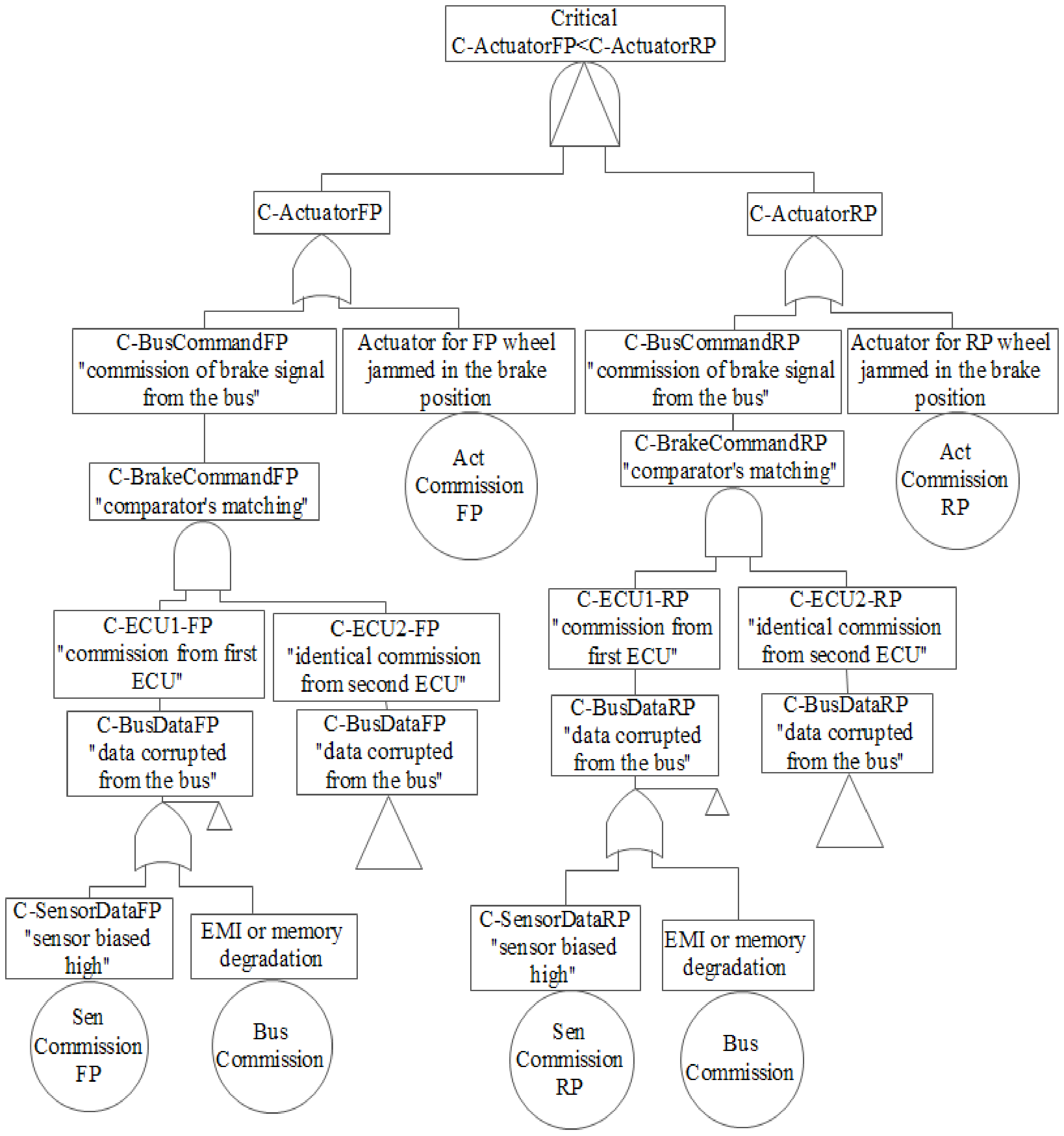

Critical C-ActuatorFP<C-ActuatorRP
C-ActuatorFP
C-ActuatorRP
C-BusCommandFP "commission of brake signal from the bus"
Actuator for FP wheel jammed in the brake position
C-BusCommandRP "commission of brake signal from the bus"
Actuator for RP wheel jammed in the brake position
C-BrakeCommandFP "comparator's matching"
Act Commission FP
C-BrakeCommandRP "comparator's matching"
Act Commission RP
C-ECU1-FP "commission from first ECU"
C-ECU2-FP "identical commission from second ECU"
C-ECU1-RP "commission from first ECU"
C-ECU2-RP "identical commission from second ECU"
C-BusDataFP "data corrupted from the bus"
C-BusDataFP "data corrupted from the bus"
C-BusDataRP "data corrupted from the bus"
C-BusDataRP "data corrupted from the bus"
C-SensorDataFP "sensor biased high"
EMI or memory degradation
C-SensorDataRP "sensor biased high"
EMI or memory degradation
Sen Commission FP
Bus Commission
Sen Commission RP
Bus Commission

Figure 6—9. [C-ActuatorFP < C-ActuatorRP] cut-sequence (4) of the "Critically Failed" hazard

6.3 Discussion

A simplified version of a BBW system has been used for this study. A compositional modelling of the analysis related information has been provided, and as required by the Safora method in order to generate and synthesise the system TFTs from those which are local to the components. Once a hierarchical description has been provided, the method started with the conversion of the highly abstract SM (which describes the monolithic failure behaviour of the BBW system) to the first system FTs — though, emphasis at this level was put on the input deviations which lead the system to catastrophic or critical situations. Marginal and moderate situations were considered at the level of a downstream SM (which describes the vehicle dynamics) — the related states can influence the fault sequences and, therefore, impose a temporal order on faults leading to the more severe states. For example, if a commission failure occurs from the actuator FP before a commission occurs from the actuator RP (which was thoroughly studied in section 6.2) this will lead to a critical situation. However, a reverse order in which those commission failures occur will lead to a less critical situation (i.e., moderate see Figure 6—3).

More emphasis was put on one cut-sequence causing a critical effect — it was not intended in the case study to go through all the generated cut-sequences (with catastrophic and critical effects together), as this would have produced vast amounts of complex results which exceed the space allowed for the thesis. However, we have demonstrated the benefits of Safora since the method (given a hierarchical description of complex failure behaviour of a dynamic system) automatically detects when to impose temporal constraints to distinguish the severe sequences of faults from those which are less severe. The process is repeated for all the components until the system (temporal) fault trees become fully synthesised. Minimisation was performed wherever possible during the synthesis, but this one often needs to be completed afterwards to logically reduce the produced large-scale TFTs.

The ability to reduce the complexity of the produced results was another highlighted benefit of Safora. This was particularly demonstrated in the GTR case study of chapter 5 due to the nature of the example system — i.e., several system constituents exhibit dynamic behaviour with more complex component interdependencies. This has helped us to show how we can minimise, wherever possible, the TFTs which are local to the system components before synthesising them into larger TFTs, and thus significantly reduce the complexity which is inherent in analysing TFTs. In this regard, a full comparison with a direct TFT modelling of the system has been provided (see section 5.3.4). The BBW example system, however, exhibits the dynamic behaviour at the level of one of its components (i.e., the “Vehicle Dynamics” see Figure 6—3). This has also provided a good case to demonstrate the benefits of our approach concerning the generation of both static and dynamic fault trees, the necessity of the temporal order of failure events being automatically detected during the conversion of the component SMs

— e.g., the SM of Figure 6—3 (which is highly dynamic and thus produces mainly TFTs) vs. the remaining component SMs (which produce static fault trees).

7 Conclusions

In this thesis we set out to study the existing safety analysis techniques which are performed on systems exhibiting dynamic behaviour. We started to explore the modelling approaches that are used to describe how such systems can fail. In the literature survey, we observed the widespread use of state transition models for this purpose (whether higher level descriptions e.g. SMs or lower level formalisms like Petri nets and Markov models). We pointed out SMs as a prevalent and expressive paradigm being used in several approaches, e.g., the state extension of HiP-HOPS in (Papadopoulos, 2000) and also with some influential modelling languages which are highly accepted in the aerospace community, like Altarica in (Rauzy, 2002) and AADL in (Joshi *et al.*, 2007).

This dissertation has presented our approach to the analysis of dynamic systems whose behavioural aspects are represented using SMs. The contributions of this thesis can be summarised as follows:

- We have outlined an important problem in the existing approaches to the SM-based analysis of dynamic systems. Briefly, one approach is based on the conversion of SMs to static fault trees [like in (Rauzy, 2002) and (Joshi, Vestal, & Binns, 2007)]. In this approach, the significance of the SM temporal semantics is lost during the conversion. The other identified approach consists of the conversion of SMs to GSPNs and Markov models. This is applied to AADL descriptions like in (Rugina A. , 2007); however, this approach is less suitable for qualitative analysis.

- A novel algorithm which generates (temporal) fault trees from SMs has been presented. One of the advantages of the algorithm is that the temporal constraints are imposed only when necessary during the conversion. This will positively impact the efforts needed for the minimisation of the generated fault trees — minimisation of TFTs is known for being a complex process.

- Minimisation is done using the Pandora sets of temporal laws; these include the SAND operator (for simultaneity). Therefore, we have identified then adapted a selection of temporal laws to the TFTs which we generate from the SMs — the generated TFTs are without SAND gates.

- We have suggested a novel, compositional (and hence potentially more scalable) method which is based on our approach to generate TFTs from SMs.

The method relies on a hierarchical description of the failure information (which is based on the failure behavioural descriptions local to the components of the system), then synthesises large system TFTs from smaller component TFTs (generated from the individual component SMs).

- We have presented a contribution regarding the optimisation of Markov models. These are state-transition based models and known with the state explosion problem. We have demonstrated how we can apply the conversion algorithm on these models, and hence we generate TFTs which we minimise using Pandora. This allows an automatic optimisation by identifying the redundancies to remove from the Markov models.

- The algorithm which converts SMs to TFTs mainly uses POR gates to impose temporal constraints during the conversion. Therefore and from a quantitative perspective, the probabilistic models given in (Merle, 2010), and which accommodate any failure distribution for the basic events, were thoroughly investigated. Some detailed comparisons with combinatorial techniques as well as with Markov solutions were made and a probabilistic model for a POR gate with n input events ($n \in N^+$ s.t. $n \geq 2$) was proposed in this thesis.

Considerable work remains to be done such as:

- Further development of this analysis approach to make it automated as part of the HiP-HOPS safety analysis tool framework.

- Potentially make the approach compatible with other SM-based modelling approaches, like interfacing with OSATE[106] (an Open Source AADL Tool Environment). This would allow Pandora TFTs to be generated from AADL state machines.

- The algorithm worst-case complexity of checking the necessity of a temporal order during the conversion is $O(n^2)$, n being the number of paths from the initial state to the final states in the SM. An improvement can be done, e.g., to ensure that paths diverging at join states are not traversed more than once each.

[106] http://www.aadl.info/aadl/currentsite/tool/osate.html

- We assumed that events are non-repairable which is compatible with the semantics of Pandora. This also conforms to (Vesely *et al.*, 1981) and, in consequence, we have also made the assumption that the SMs are acyclic. However, repairable events would be worth being considererd as many users aim at modeling repairable systems. This would imply the existence of cyclic behaviours and hence revision of the algorithm of this dissertation. The current implementation version simply breaks the cycles found during path traversals and acknowledge users where these have been detected in the SM.

- The TFTs which are generated by the algorithm of this thesis are logically reduced using the sets of Pandora temporal laws. Work on minimisation such as ensuring that the reduced cut-sequences are minimal and related automation issues is not part of this thesis, but can be found in (Walker M., 2009). However, we would like also to study further the work presented in (Merle, 2010) concerning the determination of the structure function of dynamic fault trees. Also, we want to investigate the complexity involved in the simplification of a structure function to a minimal canonical form and the calculation of this canonical form.

Closing this thesis, we wish to say that our approach to dynamic safety analysis contributes to solving some of the problems encountered in the application of similar techniques. This potentially could have a positive impact whether in the industry, where the generation of fault trees from SMs has been applied to Altarica models (in some Airbus projects for instance), or in the research domain like with AADL models. Moreover, the SM conversion algorithm that we developed, in theory, could enable its application in large and complex systems. In this context, we have presented in chapter 5 a method for the compositional synthesis of TFTs from SMs, which could improve the scalability of the application of this algorithm; also we have carried out two case studies[107] of significant complexity and demonstrated in principle both the feasibility and value of the approach. However, a conclusive evaluation of the scalability of our approach could only be achieved in a much wider and more realistic context of application.

[107] See chapter 5 (section 5.3) and chapter 6.

References:

AADL Website: http://www.aadl.info/aadl/currentsite/.

AADL-Committee-As2cAnnexE. (2006). *SAE Architecture Analysis and Design Language (AADL) Annex Volume 1, Annex A: Graphical AADL Notation, Annex C: AADL Meta-Model and Interchange Formats, Annex D: Language Compliance and Application Program Interface Annex E: Error Model Annex, AS5506/1.* from: http://standards.sae.org/as5506/1/.

AADL-Subcommittee. (2007). *AADL Behavior Language Compliance and Application Program Interface, Apr 2007: http://aadl.sei.cmu.edu/aadl/documents/Behaviour_Annex1.6.pdf.*

Abele, A., Hagl, F., Sjöstedt, C.-J., Lönn, H., & Sandberg, A. (2010). *Advancing Traffic Efficiency and Safety through Software Technology phase 2 (ATESST2), Deliverable D6.1.2, Case Study.* Stappert, F. (Ed.). ATESST2 - Grant Agreement 224442. http://www.atesst.org/home/liblocal/docs/ATESST2_Deliverable_D6.1.2_V1.0.pdf.

Amari, S., Dill, G., & Howals, E. (2003). A new approach to solve dynamic fault-trees. *In Proceedings IEEE Annual Reliability and Maintainability Symposium (RAMS 2003), pages 374–379, Tampa, FL, USA, 2003. 16, 52.*

ANSI/IEEE-Std.352. (1987). *IEEE Guide for General Principles of Reliability Analysis of Nuclear Power Generating Station Safety Systems, ANSI/IEEE Std 352-1987 (Revision of ANSI/IEEE Std 352-1975).* doi: 10.1109/IEEESTD.1987.101069. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=159367&isnumber=4154.

Arnold, A., Point, G., Griffault, A., & Rauzy, A. (2005). *AltaRica Manuel méthodologique.* LaBRI, Université Bordeaux I et CNRS (UMR 5800).

ARP4761. (1996). Aerospace Recommended Practice: Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment. Society of Automotive Engineering. Warrendale, PA: SAE.

Avižienis, A., Laprie, J.-C., & Randell, B. (2001). Fundamental Concepts of Computer System Dependability. *In: IARP/IEEE-RAS Workshop on Robot Dependability: Technological Challenge of Dependable Robots in Human Environments – Seoul, Korea, May 21-22, 2001* .

Avižienis, A., Laprie, J.-C., Randell, B., & Landwehr, C. (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. *In: IEEE Trans. Dependable Secur. Comput. , 1*, 11-33.

Béounes, C., Aguéra, M., Arlat, J., Bachmann, S., Bourdeau, C., Doucet, J.-E., et al. (1993). Surf-2: a program for dependability evaluation of complex hardware and software systems. *in 23rd IEEE Int. Symposium on Fault Tolerant Computing, (Toulouse, France), pp.668-673, 1993.*

Bernard, R. (2009). *Multi-system safety analyses. PhD Thesis. N° d'ordre: 3885.* Université Bordeaux I, École Doctorale de Mathématiques et Informatique, France.

Bernard, R., & Pouzolz, F. (2010). Presentation: Failure propagation modeling: multi-system safety analysis. *EADB Safety Reliability Directives & Innovation, Airbus, France. "http://cisec.enseeiht.fr/index.php?option=com_content&view=article&id=53:news&catid=37:past-events&Itemid=61".*

Bieber, P., Castel, C., & Seguin, C. (2002). Combination of fault tree analysis and model checking for safety assessment of complex system. *In Proceedings of the 4th European Depting Conference on Dependable Computing (EDCC), ser. LNCS, vol. 2485. Springer, pp. 19–31.*

Biehl, M., DeJiu, C., & Törngren, M. (2010). Integrating Safety Analysis into the Model-based Development Toolchain of Automotive Embedded Systems. *In Proceedings of the LCTES 2010, ACM Press.*

Bondavalli, A., & Simoncini, L. (1990). Failure classification with respect to detection. *PDCS: Predictably Dependable Computing Systems, Esprit Project Nr 3092.*

Bozzano, M., & Villafiorita, A. (2003). Improving System Reliability via Model Checking: the FSAP /NuSMV-SA Safety Analysis Platform. *In: Proceedings of SAFECOMP 2003, Edinburgh, Scotland, United Kingdom, 23-26 September, 2003* , 49-62.

Bozzano, M., Jochim, C., & Tapparo, F. (2006). The FSAP/NuSMV-SA Safety Analysis Platform. *Presented at ECAI 2006. Riva del Garda, Italy, August 28 - September 1 2006.*

Cavada, R., Cimatti, A., Jochim, C. A., Keighren, G., Olivetti, E., Pistore, M., et al. (1998). *NuSMV 2.4 User Manual.* Italy: ITC-irst - Via Sommarive 18, 38055 Povo (Trento). Italy.

Ciardo, G., & Lindemann, C. (1993). Analysis of deterministic and stochastic Petri nets. *Proceedings of the fifth international workshop on Petri Nets and performance models (PNPM'93), Toulouse, France, October 1993.*

Codetta-Raiteri, D. (2005). *Extended Fault Trees Analysis supported by Stochastic Petri Nets. PhD Thesis.* Dipartimento di Informatica, Università degli Studi di Torino, Italy.

Dehlinger, J., & Dugan, J. (2008). *Analyzing Dynamic Fault Trees Derived from Model-based System Architectures.* Retrieved from Nuclear Engineering and Technology, VOL.40 NO.5 AUGUST 2008: http://article.nuclear.or.kr/jknsfile/v40/JK0400365.pdf

Dugan, J. B., Bavuso, S. J., & Boyd, M. A. (1992). Dynamic fault-tree models for fault-tolerant computer systems. *Reliability, IEEE Transactions on , vol.41, no.3, pp.363-377, Sep 1992. doi: 10.1109/24.159800. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=159800&isnumber=4184* .

Dugan, J. B., Venkataraman, B., & Gulati, R. (1997). DIFtree: a software package for the analysis of dynamic fault tree models. *Reliability and Maintainability Symposium. 1997 Proceedings, Annual, pp.64-70, 13-16 Jan 1997, doi: 10.1109/RAMS.1997.571666, ISSN: 0149-144X. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=571666&isnumber=12344.*

Dugan, J., Sullivan, K., & Coppit, D. (2000). Developing a Low-Cost, High-Quality Software Tool for Fault Tree Analysis. *IEEE Transactions on Reliability, Vol 49 Issue 1, Mar 2000, ISSN 0018-9529. pp 49-59* .

Dutuit, Y., & Rauzy, A. (1997). Exact and truncated computations of prime implicants of coherent and non-coherent fault trees within aralia. *Reliability Engineering and System Safety, 58 :127–144, 1997* .

Ericson, C. (1999). Fault Tree Analysis – A History. *In: Proceedings of the 17th International System Safety Conference.*

Feiler, P. H., & Rugina, A. E. (2007). *Dependability Modeling with the Architecture Analysis and Design Language (AADL).* Carnegie Mellon Software Engineering Institute, N°CMU/SEI-2007-TN-043, 2007.

Feiler, P. H., Gluch, D. P., & Hudak, J. J. (2006). *The Architecture Analysis & Design Language (AADL): An Introduction.* Carnegie Mellon Software Engineering Institute (SEI), N°CMU/SEI-2006-TN-011 February 2006.

Fenelon, P., & McDermid, J. A. (1993). An integrated toolset for software safety analysis. *Journal of Systems and Software 21(3), pp 279-290.*

Fenelon, P., & McDermid, J. A. (1992). *New directions in software safety: Causal modelling as an aid to integration.* Technical report, High Integrity Systems Engineering Group, Dept of Computer Science, University of York, 1992.

Freund, U., Gurrieri, O., Küster, J., Lönn, H., Migge, J., Reiser, M.-O., et al. (2003). *An Architecture Description Language for developing Automotive ECU-Software.* Internal Report N° A03-R-337 || freund03a (2003) - http://hal.inria.fr/inria-00099774/fr/.

Fussell, J., Aber, E., & Rahl, R. (1976). On the Quantitative Analysis of Priority-AND Failure Logic. *IEEE Transactions on Reliability, vol. R-25, no. 5, pp 324–326, 1976. 2, 12, 52* .

Gaufillet, P., Bodeveix, J.-P., Filali, M., & Vernadat, F. (2006). *AADL behavior Annex presentation (from SAE AS2C, Detroit, Apr 2006): http://www.aadl.info/aadl/documents/MamounBehaviorAnnexApril2006.pdf.*

German, R., & Mitzlaff, J. (1995). Transient analysis of deterministic and stochastic Petri nets with TimeNET. *Proceedings of the 8th International Conference on Computer Performance Evaluation, Modelling Techniques, and Tools and MMB (Lecture Notes in Computer Science, vol 977, o209-223). Heidelberg, Germany, 1995.*

Griffault, A. (2003). Conception et validation d'un protocole avec le modèle AltaRica. *In: Jean-Marc Jézéquel, ed., AFADL : Approches Formelles dans l'Assistance au Développement de Logiciels, LABRI UMR CNRS 5800, université Bordeaux I ,* 293-307.

Griffault, A., Arnold, A., Point, G., & Rauzy, A. (1999). The Altarica Formalism for Describing Concurrent Systems. *Fundamenta Informaticae, 34, 1999* .

Grunske, L., & Neumann, R. (2002). Quality improvement by integrating non-functional properties in software architecture specification. *In proceedings of the 2nd Workshop on Evaluating and Architecting System dependabilitY (EASY 02) at ASPLOS-X, pp 23-32, San Jose, California, Oct 2002.*

Grunske, L., Kaiser, B., & Papadopoulos, Y. (2005). Model-driven Safety Evaluation with State-Event-Based Component Failure Annotations. *Component-Based Software Engineering, 8th International Symposium, CBSE 2005, Proceedings, pp 33-48* .

Güdemann, M., Ortmeier, F., & Reif, W. (2008). Computing Ordered Minimal Critical Sets. In G. Tarnai, & E. Schnieder (Ed.), *Proceedings of Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 2008).*

Güdemann, M., Ortmeier, F., & Reif, W. (2007). Using deductive cause-consequence analysis (DCCA) with SCADE. In F. Saglietti, & N. Oster (Ed.), *Computer Safety, Reliability, and Security, 26th International Conference, SAFECOMP 2007, ser. LNCS, vol. 4680. Springer, pp. 465–478.*

Hadjidj, R., & Boucheneb, H. (2011). Efficient Reachability Analysis for Time Petri Nets. *IEEE Transactions on Computers, vol.60, no.8, pp.1085-1099, Aug. 2011. doi: 10.1109/TC.2010.195. URL:*

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5601689&isnumber=5929503. Preprint published online 23 Sept. 2010.

Hecht, M., Lam, A., & Vogl, C. (2011). A Tool Set for Integrated Software and Hardware Dependability Analysis Using the Architecture Analysis and Design Language (AADL) and Error Model Annex. *16th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), pp.361-366, 27-29 April 2011, doi: 10.1109/ICECCS.2011.44. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5773411&isnumber=5773374.*

Hecht, M., Lam, A., Howes, R., & Vogl, C. (2010). Automated Generation of Failure Modes and Effects Analyses from AADL Architectural and Error Models. *Proc. 2010 Systems and Software Development Conference, Salt Lake City, UT, May, 2010, available online at http://sstc-online.org/2010/.*

IEC-60812. (2006). *Analysis Techniques for System Reliability – Procedure for Failure Mode and Effects Analysis (FMEA).* 3 rue de Varembé, CH-1211 Geneva 20, Switzerland: International Electrotechnical Commission IEC-60812.

IEC-61508. (1997). *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems.* 3 rue de Varembé CH 1211 Geneva Switzerland: International Electrotechnical Commission 65A/179-185, IEC-61508.

Isograph. (2002). *Fault Tree+ v10.1. Reliability Analysis Software*. Retrieved March 12, 2012, from http://www.isograph-software.com/index.htm

Johannessen, P., Grante, C., Alminger, A., Eklund, U., & Torin, J. (2001). Hazard Analysis in Object Oriented Design of Dependable Systems. Proceedings of the International Conference on Dependable Systems and Networks (DSN 2001) (formerly: FTCS), 1-4 July 2001, Göteborg, Sweden.

Joshi, A., Vestal, S., & Binns, P. (2007). Automatic Generation of Static Fault Trees from AADL Models. *In: DSN Workshop on Architecting Dependable Systems, Edinburgh, Scotland - UK.*

Kaiser, B., Gramlich, C., & Förster, M. (2007). State/event fault trees – A safety analysis method for software controlled systems. *Reliability Engineering & System Safety, vol 92, pp 1521-1537* .

Kaiser, B., Liggesmeyer, P., & Mäckel, O. (2003). A new component concept for fault trees. (P. Lindsay, & T. Cant, Eds.) *In Proceedings of the 8th Australian workshop on Safety critical systems and software - Vol. 33 (SCS '03). Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 37-46* .

KLETZ, T. A. (1997). HAZOP – Past and Future. *Reliability Engineering and System Safety, 55(3), pp 263-266.*

Laprie, J.-C. (1998). Dependability of computer systems: from concepts to limits. *In: Proc. of IFIP Intern. Workshop on Dependable Computing and Its Applications (DCIA98)* .

Laprie, J.-C. (Ed.). (1992). Dependability: Basic concepts and terminology in English, French, German, Italian and Japanese. Vienna, Austria. *Dependable Computing and Fault Tolerance. Springer-Verlag, 265 p.*

Laprie, J.-C. (1985). Dependable Computing and Fault Tolerance: Concepts and Terminology. *In: Proceedings of the 15th International Symposium on Fault-Tolerant Computing (FTCS-15),* (pp. 2–11). Ann Arbor.

Mahmud, N., Papadopoulos, Y., & Walker, M. (2010). A Translation of State Machines to Temporal Fault Trees. *In: Proceedings of Dependable Systems and Networks Workshops, 40th IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W '2010) — PFARM workshop, pp. 45-51, ISBN: 978-1-4244-7729-6.*

Mahmud, N., Walker, M., & Papadopoulos, Y. (2011). Compositional Synthesis of Temporal Fault Trees from State Machines. *International Conference on Availability, Reliability and Security (ARES 2011), Second Workshop and Tool Session on DYnamic Aspects in DEpendability Models for Fault-Tolerant Systems (DYADEM-FTS), Aug. 2011, Vienna, pp. 429-435, ISBN: 978-0-7695-4485-4.*

Mahmud, N., Walker, M., & Papadopoulos, Y. (2012). Compositional synthesis of Temporal Fault Trees from State Machines. (S. Distefano, A. Puliafito, & K. S. Trivedi, Eds.) *ACM SIGMETRICS, Performance Evaluation Review, Special Issue on Modeling Dynamic Behaviors of Complex Distributed Systems, vol. 39, no. 4, April 2012. pp. 79-88. ISSN: 0163-5999.*

DOI=10.1145/2185395.2185444 http://doi.acm.org/10.1145/2185395.2185444 .

Merle, G. (2010). Algebraic modelling of Dynamic Fault Trees, contribution to qualitative and quantitative analysis. *Ph.D. Thesis N° ENSC-2010/233, École Normale Supérieure de Cachan - ENSC, Laboratoire Universitaire de Recherche en Production Automatisée (LURPA ENS CACHAN / EA 1385), France.*

MIL-STD-1629A. (1980). *Procedures for performing a failure mode, effects and criticality analysis.* http://www.sre.org/pubs/Mil-Std-1629A.pdf.

NASA. (2007). *Systems Engineering Handbook.* National Aeronautics and Space Administration. NASA/SP-2007-6105 Rev1. NASA Headquarters .Washington, D.C. 20546. December 2007.

Ortmeier, F., Reif, W., & Schellhorn, G. (2005). Deductive Cause-Consequence Analysis (DCCA). *Proceedings of the 16th IFAC World Congress Elsevier Jun-2006 ISBN: 978-0-08-045108-4 and 0-08-045108-X.*

Papadopoulos, Y. (2000). *Safety-Directed System Monitoring Using Safety Cases.* PhD Thesis. The University of York, Department of Computer Science.

Papadopoulos, Y., & Maruhn, M. (2001). Model-Based Automated Synthesis of Fault Trees from Matlab-Simulink Models. *Int. Conf. On Dependable Systems and Networks Pages 77-82.*

Papadopoulos, Y., & McDermid, J. (1999). Hierarchically Performed Hazard Origin and Propagation Studies. *Proceedings of SAFECOMP'99, 18th international Conference on Computer Safety, Reliability and Security, Toulouse, 1999.*

Papadopoulos, Y., & Petersen, U. (2003). Combining ship machinery system design and first principle safety analysis. *IMDC'03, 8th Int'l Marine Design Conference, Athens , pages 1:415-426.*

Papadopoulos, Y., Walker, M., Parker, D., Rüde, E., Hamann, R., Uhlig, A., et al. (2011). Engineering Failure Analysis and Design Optimisation with HiP-HOPS. *Journal of Engineering Failure Analysis, DOI: 10.1016/j.engfailanal.2010.09.025, Elsevier Science, ISSN: 1350 6307* .

Parker, D. (2010). *Multi-Objective Optimisation of Safety-Critical Hierarchical Systems.* PhD Thesis. University of Hull.

Pouzolz, F. (2010). Presentation: Overview of Model Based Safety Assessment based on Altarica language in Airbus. *ROSAS project RObustnessand SAfety in the System design, EADB Safety Reliability Directives & Innovation, Airbus, France. "http://cisec.enseeiht.fr/images/docs/100209MBSA/04-fpz_rosas_missa_summary_20100209.pdf.pdf".*

Pumfrey, J. (1999). *The Principled Design of Computer System Safety Analyses.* Ph.D. Thesis, The University of York.

Rausand, M., & Høyland, A. (2004). *System Reliability Theory: Models and Statistical Methods - Second Edition.* New Jersey, USA, ISBN 0-471-47133-X: Wiley.

Rauzy, A. (2002). Mode automata and their compilation into fault trees. *Reliability Engineering and System Safety, 78(1)* , 1-21.

Redmill, F., Chudleigh, M., & Catmur, J. (1999). *System safety: HAZOP and software HAZOP.* ISBN 0471982806: Wiley.

Rugina, A. (2007). *Dependability Modeling and Evaluation – From AADL to Stochastic Petri Nets.* Ph.D. Thesis, Institut National Polytechnique de Toulouse - INPT, France.

Rugina, A. E., Kanoun, K., & Kaaniche, M. (2008). The ADAPT Tool: From AADL Architectural Models to Stochastic Petri Nets through Model Transformation. *7th European Dependable Computing Conference (EDCC), Kaunas, Lituanie, 2008.*

Sullivan, K. J., Dugan, J. B., & Coppit, D. (1999). The Galileo fault tree analysis tool. *Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on, pp.232-235, 1999, doi: 10.1109/FTCS.1999.781056. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=781056&isnumber=16917.*

Torchiaro, S., Rösel, B., Chen, D., Naseer, T., Hagl, F., & Lönn, H. (2011). *(S. Cerchio, Eds.) Model-based Analysis & Engineering of Novel Architectures for*

Dependable Electric Vehicles (MAENAD). Deliverable D6.1.1. Preliminary case study definition and evaluation metrics. Grant Agreement 260057. http://www.maenad.eu.

Törngren, M., Walker, M., Papadopoulos, Y., Mahmud, N., Espinoza, H., Tagliabò, F., et al. (2008). *Advancing Traffic Efficiency and Safety through Software Technology phase 2 (ATESST2), Deliverable D2.1, State of practice and State of the art.* ATESST2 - Grant Agreement 224442. http://www.atesst.org.

Trivedi, K. S. (2001). *Probability and Statistics with Reliability, Queuing, and Computer Science Applications.* John Wiley and Sons, New York, 2001, Second Edition, ISBN 0-471-33341-7.

Vesely, W. E. (1981). *Fault Tree Handbook, US Nuclear Regulatory Committee Report NUREG-0492.* US NRC Washington DC United States.

Vesely, W. E., Goldberg, F. F., Roberts, N. H., & Haasl, D. F. (1981). *Fault Tree Handbook.* Washington D.C., USA. US Nuclear Regulatory Commission.

Vesely, W., Stamatelatos, M., Dugan, J., Fragola, J., Minarick III, J., & Railsback, J. (2002). *Fault Tree Handbook with Aerospace Applications.* NASA Office of Safety and Mission Assurance, NASA Headquarters, Washington DC 20546, from: http://www.hq.nasa.gov/office/codeq/doctree/fthb.pdf.

Villemeur, A. (1992). *Reliability, Availability, Maintainability and Safety Assessment: Volume 1.* Chichester, UK. John Wiley & Sons.

Walker, M. (2009). *Pandora – A Logic for the Qualitative Analysis of Temporal Fault Trees. PhD Thesis.* University of Hull, UK.

Walker, M., & Papadopoulos, Y. (2006). Pandora: The Time of Priority-AND gates. *12th IFAC Symposium on Information Control Problems in Manufacturing (INCOM'06), St Etienne, France. pp 237- 242. (Best Paper in Track Award).*

Walker, M., & Papadopoulos, Y. (2008). Synthesis and analysis of temporal fault trees with PANDORA: The time of Priority AND gates. *Nonlinear Analysis: Hybrid Systems, Volume 2, Issue 2, June 2008, Pages 368-382, ISSN 1751-570X, 10.1016/j.nahs.2006.05.003. http://www.sciencedirect.com/science/article/pii/S1751570X06000574* .

Walker, M., Bottaci, L., & Papadopoulos, Y. (2007). Compositional Temporal Fault Tree Analysis. (F. Saglietti, & N. Oster, Eds.) *Computer Safety, Reliability, and Security — SAFECOMP'07, Lecture Notes in Computer Science 4680, pp. 105-119. Springer ISBN 978-3-540-7* .

Walker, M., Mahmud, N., Papadopoulos, Y., Tagliabò, F., Torchiaro, S., Schierano, W., et al. (2010). *Advancing Traffic Efficiency and Safety through Software Technology phase 2 (ATESST2), D2.1 Appendix A3.2, Review of relevant Safety Analysis Techniques.* ATESST2 - Grant Agreement 224442. http://www.atesst.org.

Walker, M., Papadopoulos, Y., Parker, D., Lönn, H., Törngren, M., Chen, D., et al. (2009). Semi-Automatic FMEA Supporting Complex Systems with Combinations and Sequences of Failures. *SAE Int. J. Passeng. Cars - Mech. Syst. 2(1):791-802, 2009, doi:10.4271/2009-01-0738* .

Wallace, M. (2005). Modular architectural representation and analysis of fault propagation and transformation. *Electronic Notes in Theoretical Computer Science. 141(3):53-71* .

Worrell, R. B., & Stack, D. W. (1978). *A SETS User Manual for the Fault Tree Analyst.* U.S. Nuclear Regulatory Commission. Office of Nuclear Regulatory Research. NUREG CR-04651.

Appendices

A — 1. HiP-HOPS Fault trees displayed using Isograph FaultTree+

Figure A — 1 represents the fault trees synthesised by HiP-HOPS (as those of Figure 2—12), but displayed using the editor of the tool Isograph FaultTree+.

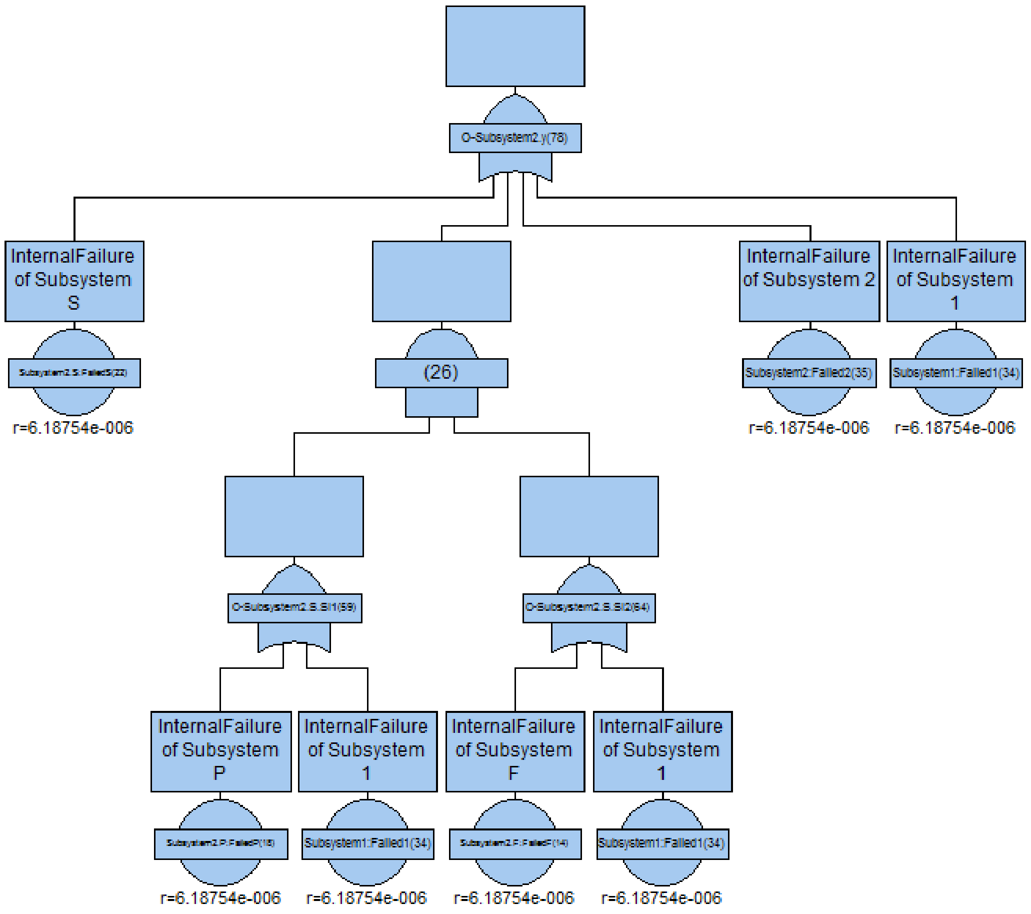
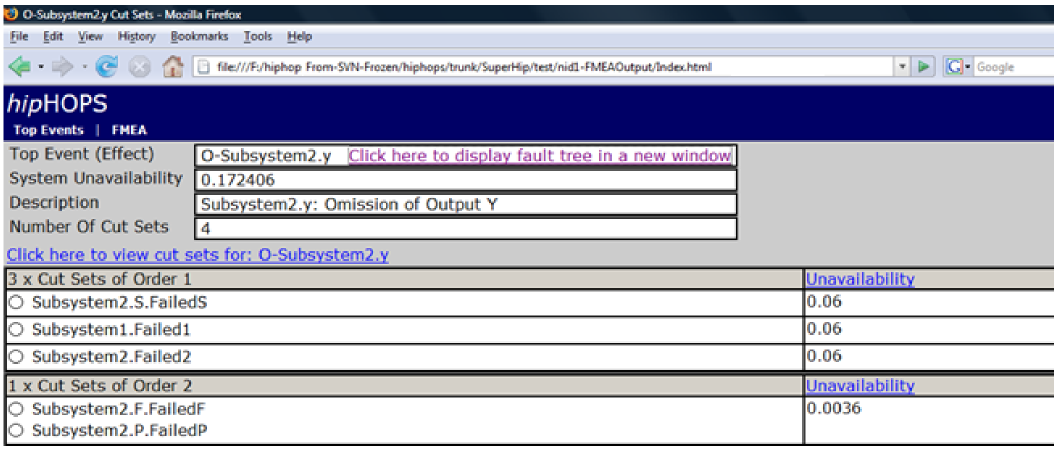


Figure A — 1. HiP-HOPS synthesised fault tree displayed using FaultTree+

A — 2. Outcome of HiP-HOPS Cut-set calculator

Figure A — 2 describes three cut-sets of order one and one cut-set of order two — i.e., a result of the HiP-HOPS analysis of the failure annotations of the model described in Figure 2—11.


O-Subsystem2.y Cut Sets - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///F:/hiphop From-SVN-Frozen/hiphops/trunk/SuperHip/test/nid1-FMEAOutput/Index.html
Google
hipHOPS
Top Events | FMEA

Top Event (Effect)	O-Subsystem2.y Click here to display fault tree in a new window
System Unavailability	0.172406
Description	Subsystem2.y: Omission of Output Y
Number Of Cut Sets	4

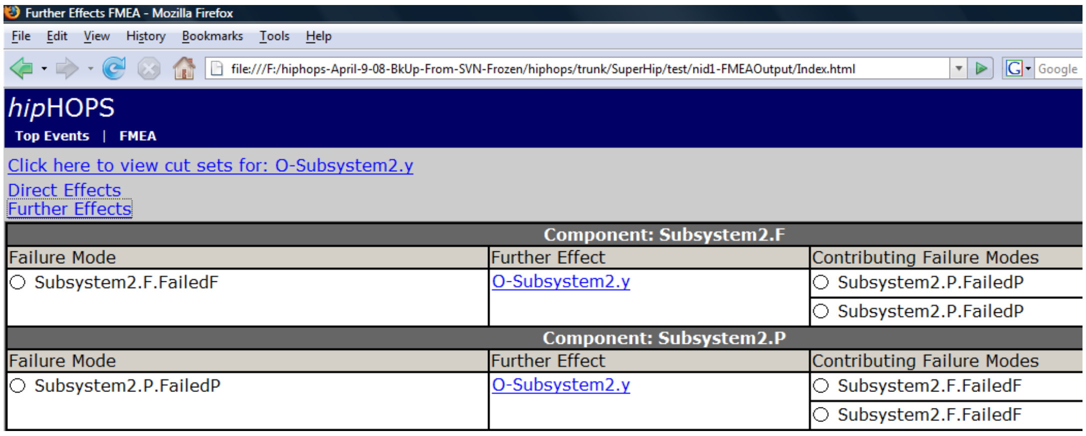
Click here to view cut sets for: O-Subsystem2.y

3 x Cut Sets of Order 1	Unavailability
○ Subsystem2.S.FailedS	0.06
○ Subsystem1.Failed1	0.06
○ Subsystem2.Failed2	0.06
1 x Cut Sets of Order 2	**Unavailability**
○ Subsystem2.F.FailedF ○ Subsystem2.P.FailedP	0.0036

Figure A — 2. Cut-sets determined by HiP-HOPS for the omission on output y of Subsystem2

A — 3. A HiP-HOPS FMEA table showing the further effects of the failure modes

Figure A — 3 shows the further effect of a failure of, e.g., component F of Subsystem2 — i.e., an omission on the output of Subsystem2, but with a contributing failure of component P (i.e., FailedP).


Further Effects FMEA - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///F:/hiphops-April-9-08-BkUp-From-SVN-Frozen/hiphops/trunk/SuperHip/test/nid1-FMEAOutput/Index.html
Google
hipHOPS
Top Events | FMEA

Click here to view cut sets for: O-Subsystem2.y
Direct Effects
Further Effects

Component: Subsystem2.F		
Failure Mode	Further Effect	Contributing Failure Modes
○ Subsystem2.F.FailedF	O-Subsystem2.y	○ Subsystem2.P.FailedP ○ Subsystem2.P.FailedP
Component: Subsystem2.P		
Failure Mode	Further Effect	Contributing Failure Modes
○ Subsystem2.P.FailedP	O-Subsystem2.y	○ Subsystem2.F.FailedF ○ Subsystem2.F.FailedF

Figure A — 3. Fragment of an FMEA table synthesised by HiP-HOPS (a further effects view)

A — 4. AADL error model with input / output propagation

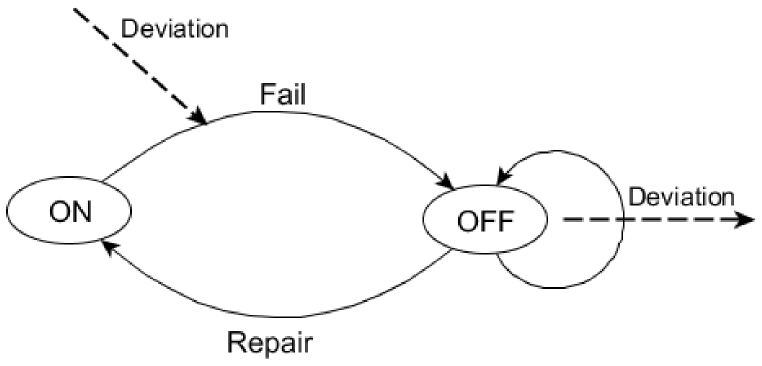


Figure A — 4. AADL SM for a component with an input-output deviation

Table A — 1. AADL textual description of a component SM propagating an input / output deviation

Declaration	Implementation

error model Example2
features
ON: **initial error state;**
OFF: **error state;**
Fail, Repair: **error event;**
Deviation: **in out error propagation**
{Occurrence => **fixed** 0.8**};**
end Example2**;**
	error model implementation Example2.basic
	transitions
	ON- [Fail] ->OFF**;**
	OFF- [**out** Deviation] ->OFF**;**
	OFF- [Repair] ->ON**;**
	ON- [**in** Deviation] ->OFF**;**
	Properties
	Occurrence => **poisson** 1.0e-3 **applies to** Fail**;**
	Occurrence => **poisson** 1.0e-4 **applies to** Repair**;**
	end Example2.basic**;**

A — 5. Optimisation example of a Markov model

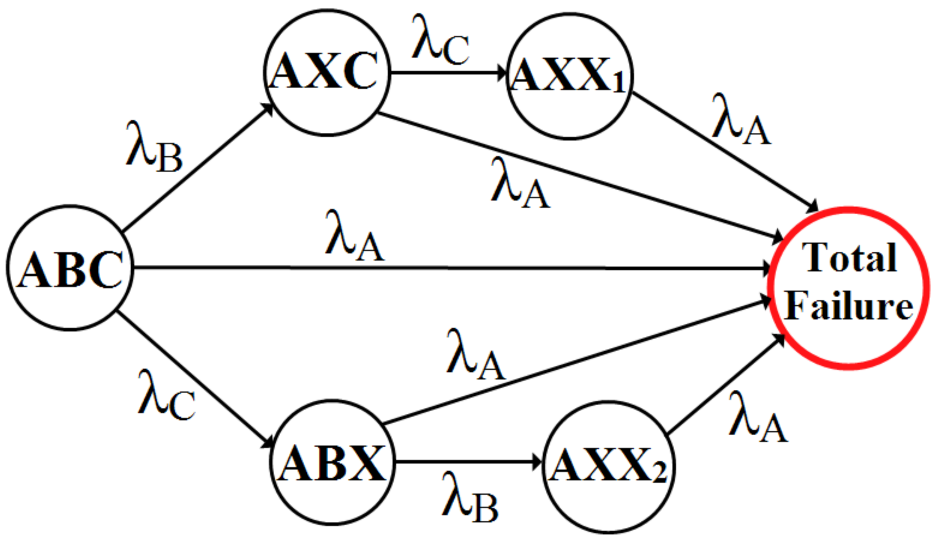


Figure A — 5. A reducible Markov model

A, B and C are the three components of the system with the failure rates λ_A, λ_B and λ_C respectively (see Figure A — 5). The application of the conversion algorithm (see chapter 4) on the Markov model of the figure gives the following failure expression:

“Total Failure” = A|C|B + (1)

A.B|A.C|A|B + (2)

A|B.C|A|B + (3)

A.C|A.B|C|A + (4)

A|C.B|C|A (5)

where (for the sake of clarity) A, B and C represent the failure events assigned with the rates λ_A, λ_B and λ_C respectively.

Now, by using the temporal law Y.X|Y ⇔ X<Y then by redundancy elimination we have:

Cut-sequence (2) ⇔ B<A.C<A.C<B ⇔ C<B<A

Cut-sequence (3) $\Leftrightarrow A|B.C<A.C|B \Leftrightarrow A|B.C<A$

Cut-sequence (4) $\Leftrightarrow C<A.B<C.B<A \Leftrightarrow B<C<A$

Cut-sequence (5) $\Leftrightarrow A|C.B<A.B|C \Leftrightarrow A|C.B<A$

Then, by expansion we have:

Cut-sequence (1) $\Leftrightarrow A<B<C + A<C<B + A<B.A|C + A<C.A|B + A|C|B$

Cut-sequence (3) $\Leftrightarrow C<A<B + A|B.C<A$

Thus, we have:

$$\begin{aligned}
\text{“Total Failure”} = {} & A<B<C + && (1.1)\\
& A<C<B + && (1.2)\\
& A<B.A|C + && (1.3)\\
& A<C.A|B + && (1.4)\\
& A|C|B + && (1.5)\\
& C<B<A + && (2)\\
& C<A<B + && (3.1)\\
& A|B.C<A + && (3.2)\\
& B<C<A + && (4)\\
& A|C.B<A && (5)
\end{aligned}$$

Similarly, we can apply the expansion law to (5) s.t.:
(5) $\Leftrightarrow B<A<C + A|C.B<A$

Thus, we now have:

$$\begin{aligned}
\text{“Total Failure”} = {} & A<B<C + && (1.1)\\
& A<C<B + && (1.2)
\end{aligned}$$

$$A<B.A|C + \qquad (1.3)$$

$$A<C.A|B + \qquad (1.4)$$

$$A|C|B + \qquad (1.5)$$

$$C<B<A + \qquad (2)$$

$$C<A<B + \qquad (3.1)$$

$$A|B.C<A + \qquad (3.2)$$

$$B<C<A + \qquad (4)$$

$$B<A<C + \qquad (5.1)$$

$$A|C.B<A \qquad (5.2)$$

The expanded cut-sequences show that the occurrences of B and C, both after A or before A and in any order between them, or either B or C occurs before A and the other one after A, or either B or C occurs alone and irrespective of its order with A, or simply the two (i.e., B and C) do not occur; all these compose the “Total Failure” expression. Therefore, the occurrences of B and C are completely irrelevant and the “Total Failure” expression can simply be minimised to A such that:

“Total Failure” = A.

A — 6. Markov solutions for the PS (before and after optimisation)

We present in this appendix the two Markov models for the PS (before and after optimisation) and show that the reduced Model preserves the quantitative result which corresponds to the "Total Failure" state. For the sake of clarity, the states of the Markov model of the PS (presented in chapter 4 section 4.4) are numbered in Figure A — 6. This also helps to better link the states to their corresponding probabilities. The meanings of the transition labels and the state numbers appearing in the figure are as follows:

λ_A, λ_B and λ_S are failure rates for A, B and S resp.

State 1 corresponds to the state A B $_S$ (system functioning with A, none is failed)

State 2 corresponds to the state X B $_S$ (system functioning with B, only A is failed)

State 3 corresponds to the state A B $_X$ (system functioning with A, only S is failed)

State 4 corresponds to the state A X $_S$ (system functioning with A, only B is failed)

State 5 corresponds to the state X B $_X$ (system functioning with B, both A and S are failed)

State 6 corresponds to the state A X $_X$ (system functioning with A, both B and S are failed)

State 7 corresponds to the total failure state.

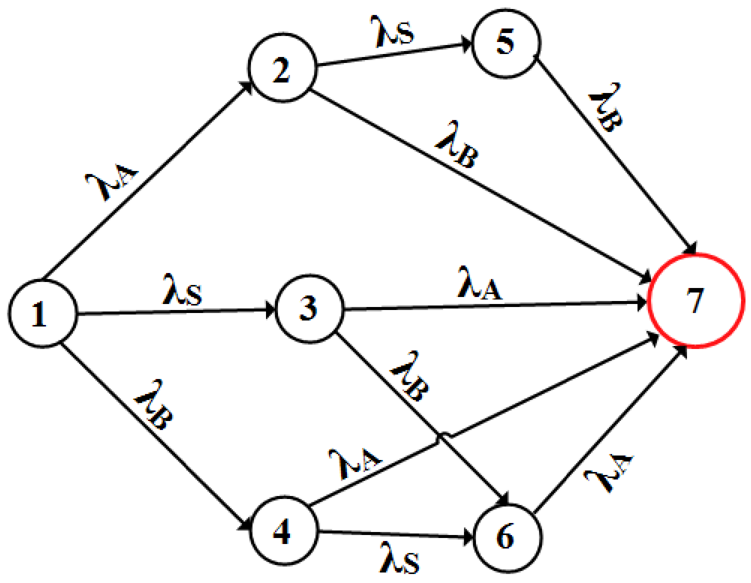


Figure A — 6. Markov model of the PS (before optimisation)

The quantitative results provided by the Markov solution of the model before optimisation are given by solving the following differential equations:

$$\frac{d}{dt} P_1(t) = -(\lambda_A+\lambda_B+\lambda_S) P_1(t)$$

$$\frac{d}{dt} P_2(t) = \lambda_A P_1(t) - (\lambda_S+\lambda_B) P_2(t)$$

$$\frac{d}{dt} P_3(t) = \lambda_S P_1(t) - (\lambda_A+\lambda_B) P_3(t)$$

$$\frac{d}{dt} P_4(t) = \lambda_B P_1(t) - (\lambda_A+\lambda_S) P_4(t)$$

$$\frac{d}{dt} P_5(t) = \lambda_S P_2(t) - \lambda_B P_5(t)$$

$$\frac{d}{dt} P_6(t) = \lambda_B P_3(t) + \lambda_S P_4(t) - \lambda_A P_6(t)$$

$$\frac{d}{dt} P_7(t) = \lambda_B (P_5(t) + P_2(t)) + \lambda_A (P_3(t) + P_4(t) + P_6(t))$$

The solution of the differential equations is given by:

$$P_1(t) = e^{-(\lambda_S+\lambda_A+\lambda_B)t}$$

$$P_2(t) = e^{-(\lambda_S+\lambda_B)t} - e^{-(\lambda_S+\lambda_A+\lambda_B)t}$$

$$P_3(t) = e^{-(\lambda_A+\lambda_B)t} - e^{-(\lambda_S+\lambda_A+\lambda_B)t}$$

$$P_4(t) = e^{-(\lambda_A+\lambda_S)t} - e^{-(\lambda_S+\lambda_A+\lambda_B)t}$$

$$P_5(t) = \frac{\lambda_A}{\lambda_S+\lambda_A} e^{-\lambda_B t} + \frac{\lambda_S}{\lambda_S+\lambda_A} e^{-(\lambda_S+\lambda_A+\lambda_B)t} - e^{-(\lambda_S+\lambda_B)t}$$

$$P_6(t) = e^{-(\lambda_S+\lambda_A+\lambda_B)t} - e^{-(\lambda_A+\lambda_B)t} - e^{-(\lambda_S+\lambda_A)t} + e^{-\lambda_A t}$$

$$P_7(t) = \frac{\lambda_A}{\lambda_S+\lambda_A} e^{-(\lambda_S+\lambda_A+\lambda_B)t} - \frac{\lambda_A}{\lambda_S+\lambda_A} e^{-\lambda_B t} - e^{-\lambda_A t} + 1 \quad \text{(total failure probability)}$$

Figure A — 7 represents an optimised Markov model for the PS. State 1 is a state in which the components A, B and S are all OK, and the states 2, 3 and 4 represent states of the system in which A has failed, S has failed or B has failed, respectively. State 5 is the total failure state of the system.

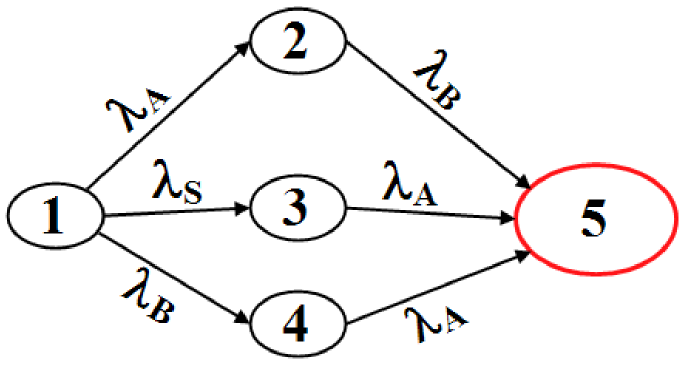


Figure A — 7. Markov model of the PS (after optimisation)

The quantitative results provided by the corresponding Markov solution are given by solving the differential equations shown next.

$$\frac{d}{dt} P_1(t) = -(\lambda_A+\lambda_B+\lambda_S)\, P_1(t)$$

$$\frac{d}{dt} P_2(t) = \lambda_A\, P_1(t) - \lambda_B\, P_2(t)$$

$$\frac{d}{dt} P_3(t) = \lambda_S\, P_1(t) - \lambda_A\, P_3(t)$$

$$\frac{d}{dt} P_4(t) = \lambda_B\, P_1(t) - \lambda_A\, P_4(t)$$

$$\frac{d}{dt} P_5(t) = \lambda_B\, P_2(t) + \lambda_A\, P_3(t) + \lambda_A\, P_4(t)$$

The solution of the differential equations is given by:

$$P_1(t) = e^{-(\lambda_S+\lambda_A+\lambda_B)t}$$

$$P_2(t) = \frac{\lambda_A}{\lambda_S+\lambda_A} e^{-\lambda_B t} - \frac{\lambda_A}{\lambda_S+\lambda_A} e^{-(\lambda_S+\lambda_A+\lambda_B)t}$$

$$P_3(t) = \frac{\lambda_S}{\lambda_S+\lambda_B} e^{-\lambda_A t} - \frac{\lambda_S}{\lambda_S+\lambda_B} e^{-(\lambda_S+\lambda_A+\lambda_B)t}$$

$$P_4(t) = \frac{\lambda_B}{\lambda_S+\lambda_B} e^{-\lambda_A t} - \frac{\lambda_B}{\lambda_S+\lambda_B} e^{-(\lambda_S+\lambda_A+\lambda_B)t}$$

$$P_5(t) = \frac{\lambda_A}{\lambda_S+\lambda_A} e^{-(\lambda_S+\lambda_A+\lambda_B)t} - \frac{\lambda_A}{\lambda_S+\lambda_A} e^{-\lambda_B t} - e^{-\lambda_A t} + 1 \quad \text{(total failure probability)}$$

$P_5(t)$ is the same as the probabilistic result $P_7(t)$ of the total failure (state 7) in the model before optimisation.