Pós-Graduação em Ciência da Computação

André Luís Ribeiro Didier

An Algebra of Temporal Faults

Ph.D. Thesis

Federal University of Pernambuco
posgraduacao@cin.ufpe.br
<www.cin.ufpe.br/~posgraduacao>

Recife, PE
March 2016

André Luís Ribeiro Didier

An Algebra of Temporal Faults

A Ph.D. Thesis presented to the Center for Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Philosophy Doctor in Computer Science.

Federal University of Pernambuco

Center of Informatics

Graduate in Computer Science

Supervisor: Alexandre Cabral Mota

Co-supervisor: Alexander Romanovsky

Recife, PE

March 2016

André Luís Ribeiro Didier
An Algebra of Temporal Faults/ André Luís Ribeiro Didier– Recife, PE, March 2016-
89 p. : il.(alguma color.); 30 cm.

Supervisor: Alexandre Cabral Mota

Co-supervisor: Alexander Romanovsky

Ph.D. Thesis – Federal University of Pernambuco
Center of Informatics
Graduate in Computer Science, March 2016.

1. Fault Trees. 2. Dependability. 3. Fault Tolerance. 4. Fault Removal. I. Alexandre Cabral Mota II. Alexander Romanovsky III. Universidade Federal de Pernambuco. IV. Centro de Informática. V. Título

CDU 02:141:005.7

André Luís Ribeiro Didier

An Algebra of Temporal Faults

A Ph.D. Thesis presented to the Center for Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Philosophy Doctor in Computer Science.

Prof. Augusto Cesar Alves Sampaio
Centro de Informática/UFPE

Prof. Paulo Romero Martins Maciel
Centro de Informática/UFPE

Prof. Enrique Andrés López Droguett
Departamento de Engenharia de Produção/UFPE

Recife, PE
March 2016

I dedicate this thesis to Juliana, Luciana (pipoquinha), and Bianca (snowflake).

Acknowledgements

If I were afraid of the path, I wouldn't have gotten here.

Two men helped me to build this path far before I started my scholar journey: Roberto and Júnior. My two grandparents couldn't see how far I got. My heart was with them all the time, but I was physically far away from them in their very last breath. May God have them in his arms.

It is now ten years since I graduated. I met professors Alexandre and Augusto still during the Computing Science undergrad course. They have been present in my academic life ever since. Their comments, instructions, talks, (even jokes), are what molded my path to here. I have no words to express how much I thank them.

CNPq and FACEPE were keen to guarantee my existential needs. The former with the trip to Newcastle upon Tyne, and the latter during the time I stayed in Recife, before and after the trip.

I thank to Sascha Romanovsky for accepting me as his advisee while I was a Research Assistant of the COMPASS project. His comments, instructions, and knowledge were of great importance for this work.

My stay in Newcastle upon Tyne could be as good as it was without the hospitality, useful discussions, and support of my colleagues at Newcastle University. A big THANK YOU to John Fitzgerald, Zoe Andrews, Richard Payne, Claire Smith, Dee Carr, Claire Ingram, my shared office colleague Anirban Bhattacharyya, and all other staff members.

Still in Newcastle upon Tyne, I thank all friends my family and I made outside University. Thanks to Kelechi Dibie and her family to welcome us for the Christmas' and new year's dinners. They were our family abroad.

I thank all my family for their patience to have me away in several family reunions, due the time required to do this work. In special, my two girls and my wife.

"Mathematical reasoning may be regarded rather schematically as the exercise of a combination of two facilities, which we may call intuition and ingenuity.
(Alan Turing)

Resumo

A modelagem de defeitos é essencial na antecipação de falhas em sistemas críticos. Tradicionalmente, Árvores de Defeitos Estáticas são empregadas para este fim, mas Árvores de Defeitos Temporais e Dinâmicas têm ganhado evidência devido ao seu rico poder para modelar e detectar ~~complexa propagação~~ de defeitos que levam a uma falha.

Em um trabalho anterior, mostramos uma estratégia baseada na álgebra de processos CSP e modelos Simulink para obter rastros de defeitos que levam a uma falha. Apesar de o trabalho usar Árvores de Defeitos Estáticas, poderíamos ~~usar~~ Árvores de Defeitos Temporais ou Dinâmicas. ~~Neste~~ trabalho, apresentamos duas álgebras: (i) uma para modelar defeitos a partir de modelos ~~arquiteturais formalmente~~, racionalizando sobre combinações de defeitos, e (ii) uma álgebra temporal de defeitos (com noção de propagação de defeitos) e provar que ela é de fato uma álgebra Booleana. Isso permite herdar as propriedades de álgebras Booleanas, leis e técnicas de redução existentes, as quais são muito benéficas para a modelagem e análise de defeitos. Nós ilustramos nosso trabalho ~~com simples, mas reais estudos de caso, alguns~~ fornecidos pelo nosso parceiro industrial, a EMBRAER.

Palavras-chave: Simulink, CSP, FDR, Fault Tree Analysis, Temporal Fault Trees, Dynamic Fault Trees, Pandora, Fault Injection

Abstract

Faults modelling is essential to anticipate failures in critical systems. Traditionally, Static Fault Trees are employed to this end, but Temporal and Dynamic Fault Trees are gaining evidence due to their enriched power to model and detect intricate propagation of faults that lead to a failure.

In previous work, we showed a strategy based on the process algebra CSP and Simulink models to obtain fault traces that lead to a failure. Although that work used Static Fault Trees, it could be used Temporal or Dynamic Fault Trees. In this work we present two algebras: (i) one to model faults from architectural models formally, reasoning about faults combinations, and (ii) an algebra of temporal faults (with a notion of fault propagation) and prove that it is indeed a Boolean algebra. This allows us to inherit Boolean algebra's properties, laws and existing reduction techniques, which are very beneficial for faults modelling and analysis. We illustrate our work on simple but real case studies, some supplied by our industrial partner EMBRAER.

Keywords: Simulink, CSP, FDR, Fault Tree Analysis, Temporal Fault Trees, Dynamic Fault Trees, Pandora, Fault Injection

List of Figures

List of Tables

List of abbreviations and acronyms

ActA	Activation Algebra p. 25
AFP	archive of formal proofs p. 64
ATF	Algebra of Temporal Faults pp. 25, 35, 67–70, 73–75, 77, 78
BDD	Binary Decision Diagram pp. 15, 23, 35, 40, 45, 48–51, 54, 77
BN	Bayesian network p. 45
CML	COMPASS Modelling Language p. 32
CPN	coloured Petri-net p. 45
CSp	cold spare pp. 24, 36, 43–46, 48, 51
CSP	Communicating Sequential Processes p. 32
CSP_M	Communicating Sequential Processes pp. 24, 25, 35, 59, 61, 67
CTMC	continuous-time Markov chain p. 45
DD	Dependence Diagram pp. 32, 33
DFT	Dynamic Fault Tree pp. 15, 17, 21, 23, 24, 29, 33, 35, 36, 40, 42–48, 51, 67, 71
DNF	disjunctive normal form pp. 36, 42, 45, 50, 51
DRBD	Dynamic Reliability Block Diagram p. 33
DTMC	discrete-time Markov chain pp. 32, 43, 48
FBA	Free Boolean Algebra pp. 35, 47, 52, 53, 64, 67–69, 74, 77
FDEP	functional dependency pp. 24, 36, 43, 44, 46
FDR	Failures and Divergences Refinement pp. 24, 59, 60
FMEA	Failure Modes and Effects Analysis p. 33
FSM	Finite State Machine p. 48
FT	fault tree pp. 15, 21, 23–25, 29, 30, 35–38, 40, 42, 47, 53–57, 61, 67, 77
FTA	Fault Tree Analysis pp. 21, 23–25, 35, 37, 39, 41, 43, 45, 54
HCAS	cardiac assist system p. 45
HiP-HOPS	Hierarchically Performed Hazard Origin and Propagation Studies pp. 24, 30, 37, 61
HLPN	high-level Petri-net p. 48
HOL	higher-order logic p. 63
Isar	Intelligible semi-automated reasoning pp. 35, 63
ITL	Interval Temporal Logic p. 47
LTL	linear temporal logic p. 40
MCS	minimal cut set pp. 23, 36, 39, 42, 44
MCSeq	minimal cut sequence pp. 23, 25, 41, 42, 44, 45, 50

PN	Petri-net p. 31
ROBDD	Reduced Ordered Binary Decision Diagram pp. 48–50
SBDD	Sequential Binary Decision Diagram pp. 45, 48, 51
SEQ	sequence enforcing pp. 24, 36, 43–45
SFT	Static Fault Tree pp. 15, 21, 23, 24, 32, 33, 35–40, 42, 44, 46–48, 50, 52, 53, 61, 67, 71
SoS	System of Systems p. 30
SWN	stochastic well-formed net p. 45
SysML	Systems Modelling Language p. 32
TFT	Temporal Fault Tree pp. 15, 21, 23, 24, 29, 35, 40–44, 46–48, 67, 71
TTT	Temporal Truth Table pp. 17, 41, 42, 50
UML	Unified Modelling Language p. 32
UTP	Unifying Theories of Programming p. 25
WSp	warm spare pp. 24, 51
Z	Z Notation pp. 48, 64

Contents

Introduction

The development process of critical control systems is based essentially on the rigorous execution of guides and regulations (ANAC, 2011; FAA, 1993; FAA, 2007; SAE, 1996). Specialized agencies (like FAA, EASA and ANAC in the aviation field) use these guides and regulations to certify such systems.

Safety plays a crucial concern on critical systems and it is the responsibility of the safety assessment process. ARP-4761 (SAE, 1996) defines several techniques to perform safety assessment. One of them is FTA. It is a deductive process that uses trees to model faults and their dependencies and propagation. In such trees, the premises are the leaves (basic events) and the conclusions are the roots (top events). Intermediary events use gates to combine basic events and each kind of gate has its own combination semantics definition. For example, the most traditional gates are OR ($\vee$) and AND ($\wedge$). FTs that use only these gates are called *coherent fault trees* (ANDREWS, 2001; ANDREWS; BEESON, 2003; OLIVA, 2006; CONTINI; COJAZZI; RENDA, 2008; VAURIO, 2016). They combine the events as *at least one shall occur* and *all shall occur*, respectively. To analyse fault trees, their structures are abstracted as Boolean expressions called *structure expressions*. The analysis of coherent FTs uses a well-defined algorithm based on the Shannon's method—which originated the Binary Decision Diagram (BDD) (AKERS, 1978; BOUTE, 1976)—to obtain minimal cut set[1]s (MCSs) from the structure expressions and a general formula to calculate the probability of top events.

Besides the traditional OR and AND gates, the Fault Tree Handbook (VESELY et al., 1981) defines other gates. For example the Priority-AND (PAND, $<$) gate, which considers the order of occurrence of events. Although the work defines these new gates, there is no algorithm to perform the analysis of trees that contain such new gates. This motivated the introduction of two new kinds of fault trees: DFTs and TFTs. These variant trees can capture sequence dependencies of fault events in a system. The difference from TFT to DFT is that TFTs use temporal gates directly, while DFT does not—DFTs gates are an abstraction of temporal gates. To differentiate traditional fault trees from the other two, we will call traditional fault trees as SFTs.

The work reported in (WALKER; PAPADOPOULOS, 2009) aims at performing the full implementation of the Fault Tree Handbook, adding temporal gates to its Pandora[2] methodology. This implementation created a new kind of tree: TFTs. In such trees, events ordering is well-defined and they developed an algebraic framework to reduce structure expressions to obtain minimal cut sequences (MCSeqs) and perform probabilistic analysis.

[1] Also known as *prime implicants* and *minterms*—two names that came from Boolean logic.
[2] Pandora stands for: P-AND-ORA, which translates to Priority AND, Time.

Reducing expressions is also desirable to check for tautologies, for example.

DFTs introduce very different gates to capture dynamic configurations of systems: cold spare (CSp), functional dependency (FDEP), and sequence enforcing (SEQ). The semantics of the first is to add "backup" events, so the gate is active if the primary event and all spares are active. The second adds basic events dependency from a trigger event. The third forces the occurrence of events in a particular order. There is also a warm spare (WSp) gate that is slightly different from the CSp gate. They differ on the nature of sparing, whether fast (warm, always-on) or slow (cold, stand-by). The readiness of the backup system in a WSp gate is higher than in a CSp gate. The work reported in (MERLE; ROUSSEL; LESAGE, 2011a) shows an algebraic framework to compositionally reduce DFT gates to order-based gates and perform probabilistic analysis of structure expressions. Thus, despite some limitations for spare gates (MERLE; ROUSSEL; LESAGE, 2014), the structure expressions used in TFTs and DFTs can be formulated in terms of a generic order-based operator.

The NOT operator is absent in the algebras reported in (WALKER; PAPADOPOULOS, 2009; WALKER, 2009; MERLE, 2010; MERLE; ROUSSEL; LESAGE, 2011b). There is no consensus about the relevance of its use: (i) it can be misleading, generating non-coherent analysis (OLIVA, 2006), or (ii) it can be essential in practical use (ANDREWS, 2001). Our concern is that the decision of the relevance of its use should not be due to the choice of events-occurrence representation. The algebra created in this work defines the NOT operator and allows its use, as we show in Chapter 3.

In previous work (DIDIER, 2012; DIDIER; MOTA, 2012), we proposed a systematic hardware-based faults identification strategy to obtain failure expressions as defined in HiP-HOPS for SFTs. We considered faults in components or subsystems, but if we obtain failure expressions of a whole system, they are in fact structure expressions of an FT. We focused on hardware faults because we assume that software does not fail as a function of time (wear, corrosion, etc). We inherited this view from our industrial partner (EMBRAER), which assumes that functional behaviour is completely analysed by functional verification (SNOOKE; PRICE, 2011). We followed industry common practices using Simulink diagrams (NISE, 1992) as a starting point. The work (DIDIER; MOTA, 2012) was based on Communicating Sequential Processes[3] (CSP_M) to allow an automatic analysis using the model checker FDR. Thus, our strategy required the translation from Simulink to CSP_M (JESUS et al., 2011). It then runs FDR to obtain several counter-examples (which are fault traces) ending in failures. For two case studies provided by our industrial partner, we showed that our automatically created failure logic matches with the engineer's provided one or is better (a weaker proposition).

In this work, we present a strategy to perform Fault Tree Analyses on systems

[3] This variant "M" is the machine-readable version of CSP.

by: (i) injecting faults on its nominal model, and (ii) complement the injected faults by modelling faults in a formal model. Figure 1 show the strategy overview. "Faults injection" block is obtained from part of the work reported in (DIDIER, 2012). It starts with Simulink modelling, converts the model to CSP_M and then obtains fault events sequences. The fault events sequences are then mapped to the Algebra of Temporal Faults (ATF). As fault names are obtained directly from components and subsystems in a Simulink model, the Activation Algebra (ActA) (in the "Faults Modelling" group) allows them to be modified or complemented. ActA also allows reasoning about faults that are not modelled in Simulink as, for example, common cause or environmental faults. ActA can be used to model faults formally, reasoning about basic fault events and top-event failures. Each predicate in ActA generates an expression in ATF, which allows expressions reduction to obtain a canonical form that enables finding MCSeqs and calculating top-events probability.

FTA has associated system requirements which are in fact acceptance criteria. Once FT's acceptance criteria is modelled as expressions in ATF, we formally check whether they are accepted by system models' expressions.

The main contributions of this work are:

1. Define a denotational and algebraic model to express fault events order with ATF, and formally verify FT's acceptance;
2. Reusing Simulink models, obtaining fault event sequences and mapping to ATF;
3. Reasoning about faults modelling in ActA, obtaining formal expressions of top-event failures;
4. Illustrating the application of the laws on real case studies, ~~two~~ provided by our industrial partner.

Other contributions of this work are:

1. Defining a new operator to express order explicitly and proving that the resulting algebra—(ATF) using this operator and Boolean operators—is a conservative extension of the Boolean algebra (also published in (DIDIER; MOTA, 2016));
2. Generalising laws in ATF in terms of abstract properties, similar to healthiness conditions in the Unifying Theories of Programming (UTP) (HOARE; HE, 1998).

We used Isabelle/HOL 2015[4], theories in Isabelle's library, and a theory in the AFP library (JASKELIOFF; MERZ, 2005) to prove all theorems presented in this work.

[4] The 2002 tutorial is reported in (NIPKOW; PAULSON; WENZEL, 2002), but there is a newer version published with the tool itself. The tool and the tutorial are available on their website at http://isabelle.in.tum.de .

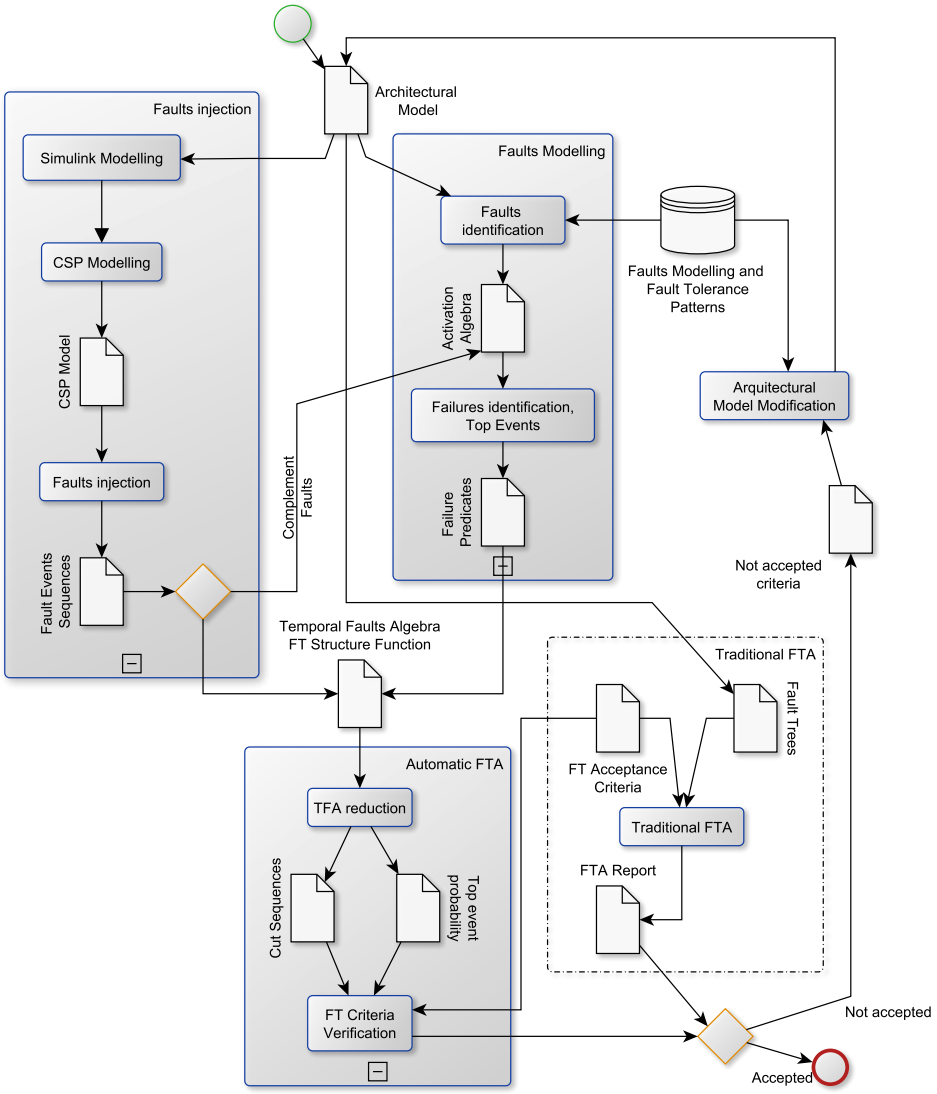


Figure 1 – Strategy overview

This thesis is organized as follows: in Part I we show the concepts and tools used as basis for this work. Part II describes the results: Chapter 3 presents our strategy, Chapter 4 the case study and the application of the proposed strategy, and we present our conclusions and future work in Chapter 5.

Part I

Background

1 Basic concepts

Means to dependability are obtained by modelling and analysing a system. It is strongly related to faults modelling, which depends on which analyses we want to perform. FTs are present in several stages of systems' modelling. We introduce dependability and faults modelling in Section 1.1.

A fault tree is a snapshot[1] of a faults topology of a system, subsystem or component. The time relation of fault events in TFTs and DFTs allows the analysis of different configurations (or snapshots) of a system, subsystem or component. We discuss these time relations in Section 1.2.

1.1 Systems, dependability and fault modelling

Computing systems are characterized by five properties: functionality, performance, cost, dependability, security. The work reported in (SOMMERVILLE, 2011, p. 289–302) explain these properties—including dependability—with a focus in software. Hardware and software are connected, as software faults may cause software-controlled hardware to fail, and hardware faults may send incorrect data, causing software to fail.

The work reported in (AVIZIENIS et al., 2004) summarizes all concepts of and related to dependability for computing systems that contains software and hardware. In the following, we show these concepts and highlight those used in this work.

1.1.1 Systems

Before introducing systems' dependability, we first describe what a system is and its characteristics. It is an entity that interacts with other systems (software and hardware as well), users (humans), and the physical world. These other entities are the *environment* of the given system, and its *boundary* is the frontier between the system and its environment.

The *function* of a system is what the system is intended to do, and its *behaviour* is what the system does to implement its function. The *total state* of a system are the means to implement its function and defined as the set of the following states: computation, communication, stored information, interconnection, and physical condition. The *service* delivered by a system is its behaviour as it is perceived by its boundary. A system can both provide and consume services.

[1] Whether a top event indeed causes a catastrophic or major failure is out of the scope of this thesis; we consider that, if it is possible that such failure occur, then it will.

The *structure* of a system is how it is composed: a system is composed of components, and each component is another system, etc. This concept of hierarchical compositionality in systems, is what originated the concept of System of Systems (SoS) (MAIER, 1998) and is object of analysis in HiP-HOPS. Such recursion stops when a component—or a constituent system—is considered to be atomic. A system is the total state of its atomic components.

1.1.2 Dependability

The concepts of dependability are: (i) threats to, (ii) attributes of, and (iii) means to attain.

Threats to dependability are the so-called *fault-error-failure* chain. A failure is a service deviation perceived on systems' boundary. An error is the part of the total state of a system that leads to subsequent service failure. Depending on how a system tolerate internal errors, many errors may not reach system's boundary. Finally, a fault is what causes an error. In this case, we say that the fault is active, the fault *occurred*. Otherwise, the fault is dormant, and has not occurred (yet). A *degraded* mode of a system is when there are active faults, so some functions of the system are inoperative, but the system still delivers its service.

There are two acceptable definitions of dependability reported in (AVIZIENIS et al., 2004). One is more general, difficult to measure: "the ability to deliver service that can justifiably be *trusted*". A more precise definition that uses the definition of service failure is: "the ability to avoid service failures that are more frequent and more severe than is acceptable". This definition has two implications about system's requirements: there should be how it can fail, and what are the acceptable severity and frequency of its failures.

The following systems' dependability attributes enlightens such requirements:

Availability: the readiness for correct service;

Reliability: continuity of correct service;

Safety: absence of catastrophic consequences on the environment (other systems, users, and the physical world). Safety can be verified using FTs, which is part of the objective of this work;

Integrity: absence of improper systems alterations;

Maintainability: ability to be modified and repaired.

A system description should mention all or most of these attributes, at least the first three.

The implementation of these attributes requires a the deep analysis of system's models. The *means to attain dependability* are summarized as follows:

Prevention is about avoiding incorporate faults during development.

Tolerance deals with usage of mechanisms to still deliver a—possibly degraded—service even in the presence of faults.

Removal is about detecting and removing (or reducing severity of) failures from a system, as in the development stage, as in production stage.

Forecasting is about predicting likely faults so they can removed, or tackling their effects.

The intersection of this work with dependability is in fault removal during development and fault tolerance (analysis). There are some techniques for fault removal, summarized as follows:

- Static verification
  - System

    Static analysis: Range from inspection or walk-through, data flow analysis, complexity analysis, abstract interpretation, compiler checks, vulnerability search, etc.

    Theorem proving: Check properties of system models.
  - Behaviour model

    Model checking: Usually the model is a state-transition model (Petri-nets (PNs), finite or infinite state automata). Model-checking verifies all possible states on a given system's model.
- Dynamic verification
  - Symbolic inputs

    Symbolic Execution: It's the execution with respect to variables (symbols) as inputs.
  - Actual inputs

    Testing: Selected input values are set on system's inputs and their outputs are compared to expected values. Outputs in this case are observed faults, in case of hardware testing or software's mutation testing, and criteria-based, in case of software testing.

Verification methods are often used in combination. For example, symbolic execution may be used to obtain testing patterns, test inputs can be obtained by model-checking as in (CARVALHO et al., 2015), faults can be used as symbolic inputs, and system behaviour can be observed using model-checking as in (DIDIER; MOTA, 2012; DIDIER, 2012) (this technique is called fault injection; see also (AVRESKY et al., 1996)).

The techniques to attain fault tolerance are summarized as follows:

Error detection: is used to identify the presence of an error. It can be a concurrent or a preemptive detection. Concurrent detection takes place during normal service, while preemptive detection takes place while normal service is suspended.

Recovery: transforms a system state that contains errors into a state without them. The behaviour of the system upon recovery is equivalent to the normal behaviour. Techniques range from rollback to a previously saved state without errors, error masking, isolation of faulty components, to reconfiguration using spare components.

In this work, we use a combination of: (i) fault-injection, (ii) theorem proving, and (iii) symbolic execution. We use these methods to obtain an erroneous behaviour of the system which is compared to system's dependability attributes (safety). We explain how these methods combine in Chapter 3.

1.1.3 Fault Modelling

Fault modelling plays an important role in reasoning about the fault-error-failure chain. They are the initial steps to perform the verification of a system.

Systems Modelling Language (SysML) (Object Management Group (OMG), 2012) is a profile for Unified Modelling Language (UML) that provides features to model structure and behaviour of systems. The works reported in (ANDREWS et al., 2013b; ANDREWS et al., 2013a) define several structural and behavioural views in SysML to model the fault-error-failure chain and fault tolerance. Fault, error, failures, and fault propagation have structural views, which are related to behavioural views to describe fault activation and recovery. These works map SysML to two formal languages—COMPASS Modelling Language (CML) (BRYANS; CANHAM; WOODCOCK, 2014) and Communicating Sequential Processes (CSP) (ROSCOE, 1997), respectively—to verify fault tolerance.

In (SAE, 1996) the safety assessment process for civil airborne systems and equipment describes development cycles and methods to "clearly identify each failure condition". The methods that involves failure identification are: (i) SFT, (ii) Dependence Diagram[2] (DD) (MODARRES; KAMINSKIY; KRIVTSOV, 2009, p. 198), (iii) Markov chain, and

[2] Also known as Reliability Block Diagram (RBD).

(iv) Failure Modes and Effects Analysis (FMEA) (MIKULAK; MCDERMOTT; BEAUREGARD, 2008). The first three are top-down methods, that starts with undesired failure conditions and moves to lower levels to obtain more detailed conditions that causes the top-level event. DDs are an alternate method of representing the data in SFT. FMEA is a bottom-up method that identifies failure modes of a component and determines the effects on the next higher level. We detail SFT in Section 2.1.1.

DFTs are an extension of SFTs and models dynamic behaviour of system's faults. Similarly to the relation of SFTs and DDs, the work reported in (DISTEFANO; PULIAFITO, 2009) demonstrates the relation of DFTs to Dynamic Reliability Block Diagrams (DRBDs) (DISTEFANO; PULIAFITO, 2009). As the models (DFT and DRBD) are equivalent, this work sticks to DFT due to the amount of work already published. We detail DFTs in Section 2.1.3.

1.2 Time relation of fault events

The most general case for time relations is to consider that each fault event has a continuous time duration. They are the basis on how fault events discretisation are defined. In Figure 2 we show all possibilities of events relations in a continuous time line (from A to B; the converse relation is similar):

a. A starts and ends before B starts;

b. A starts before and ends after B has started, but before B has ended;

c. A starts before B and ends after B has ended (A contains B);

d. A and B start at the same time, but A ends before B;

e. B starts after A, but they end at the same time;

f. A and B start and end at the same time;

g. A starts before B and ends when B starts.

Although the occurrence of fault events has at least seven possibilities, what really matters when analysing systems is when a fault is *detected*. Considering that fault detection corresponds to the start of a fault event, from Figure 2 we clearly identify which event comes first: A comes before than B, except in the cases (d) and (f), where they start exactly at the same time. If fault events are independent (they are not susceptible to have a common cause) then the probability of they are starting at the same time is very low. In Chapter 3 we abstract events relation in continuous time as an *exclusive before*

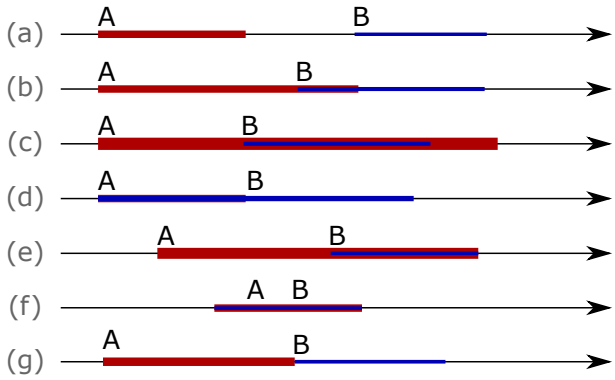


Figure 2 – Relation of two events with duration

relation, based on fault *detection* (it is similar—at least implicitly—to what is reported in (WALKER; PAPADOPOULOS, 2009; MERLE; ROUSSEL; LESAGE, 2011a)).

2 Analysis and tools

Structure expressions are used to analyse fault trees. In general, a structure expression comes from gates semantics and basic events. Basic events become variables and gates become operators (a gate may become one or more operators). In Section 2.1 we explain SFTs, TFTs, DFTs, and their respective structure expressions.

Free Boolean Algebras (FBAs) (GIVANT; HALMOS, 2009, pp. 256-266) and BDDs are a basis to analyse structure expressions. Also, we were inspired by FBA concepts to create our Algebra of Temporal Faults (Chapter 3). We explain BDDs and derived techniques in Section 2.2, and FBAs in Section 2.3.

The use of the Boolean operator *NOT*: (i) can be misleading, generating non-coherent fault trees, or (ii) can be essential in practical use. We discuss such cases in Section 2.4.

To reuse a nominal model to analyse faults we need fault injection. In Section 2.5 we explain how we used Simulink and CSP_M to inject faults and obtain failure logic from a nominal model.

Finally, in Section 2.6 we present basic usage of Isabelle/HOL and Intelligible semi-automated reasoning (Isar), which were essential to carry out the proofs presented in this thesis.

2.1 Fault Tree Analysis and structure expressions

FTA was introduced in the Fault Tree Handbook with Static Fault Trees. FTA is a deductive method that investigates what are the possible causes of a unwanted event. The method starts with the top-level event as the unwanted event and the combination of lower-level events that can cause it. Events are combined using gates, and each gate has a well defined semantics. It continues until basic (atomic) events are reached. An SFT represents, in a single view—very often considering faults outside of the boundaries of a system—different states in which a particular failure is active in a system. The most traditional gates are AND and OR, which are equivalent to Boolean operators. These gates are also called coherent gates because they construct coherent trees (see Section 2.4 about the use of NOT (¬) gates). SFT's gates and analysis are detailed in Section 2.1.1.

TFTs were created aiming at fully implement the Fault Tree Handbook (VESELY et al., 1981). The PAND gate was first defined for SFTs, but its analysis was left open in the handbook. The analysis of TFTs uses a denotational semantics based on *sequence values* to express ordering of events, thus tackling PAND's order. We explain TFTs and

the sequence values in Section 2.1.2.

With component and system design evolution, DFTs were created to tackle dynamic behaviour: fault-tolerance-related components (CSp), functional dependency (FDEP), and analysis of particular order of occurrence of faults (SEQ). SFT's gates are still present as DFT's gates. We explain them and DFT's analysis in Section 2.1.3.

The structure of an FT (or the structure of a MCS, explained further) is represented with a formula. The variables represent occurrence of basic events. Unary and binary relation symbols capture the semantics of gates. A formula with these characteristics is called *structure expression* or *structure function* (as the expression depends on the variables). The semantics of a structure expression is that the top-level event occurs if some combination of basic events occur.

The results obtained from the evaluation of FTs are shown in the Fault Tree Handbook. We summarize them as:

- Qualitative

  MCSs: Smallest combinations of components failures causing system failure. They are obtained from the reduction to a normal form. For example, in SFTs, structure expressions are reduced to disjunctive normal form (DNF). Each term in a reduced DNF is a MCS.

  Importances: Qualitative rankings on contributions to system failure. A single fault causing a catastrophic failure is usually unacceptable. Ranking MCSs is the same as ordering them in ascending order of their size (smaller first).

- Quantitative

  Numerical probabilities: Probabilities of system and MCS' failures. A system failure probability is obtained by assigning probabilities to basic events and then calculating it accordingly to gates' semantics. MCS' failure probability is the calculation of the probability of the occurrence of *all* basic events of a specific MCS.

  Importances: Quantitative rankings on contributions to system failure. Ranking MCSs is the same as ordering them in descending order of some unreliability formula (higher first). These formulas used to calculate importance vary. The most commons are: (i) system unavailability, and (ii) system failure occurrence rate.

  Sensitivity evaluation: Modifying characteristics of components and evaluate their impact. For a particular event in a tree, a higher and a lower failure probability value are assigned. If system's unavailability is not changed, then such event is not important—the system is not sensitive to such an event.

As stated in (STAMATELATOS et al., 2002), there are other uses of FTA. One of great importance is using it to minimize and optimize resources, which has been object of study in HiP-HOPS (ADACHI et al., 2011). Through importance measures, FTA not only identifies what is important but also what is unimportant, thus removing components without impacting the overall failure probability, which is related to the quantitative importance and sensitivity evaluation.

In important stages of critical systems, FTA plays essential role. At least three dependability means can be achieved using FTs:

Removal. An FTA indicates if the probability of failure of a subsystem is high, so such subsystem should be removed or left to be incorporated in combination of a more reliable component.

Tolerance. An FTA indicate whether a single fault—or fewer combinations than expected—could lead to a catastrophic failure. In this case, a system should have replication, or stages of fault detection and error handling. Also, the probability of failure of the chosen fault tolerance method can be evaluated.

In Sections 2.1.1 to 2.1.3 we show briefly FTs's symbology and means to analyse FTs. We will detail it structure expressions extraction because they are a common means to perform both qualitative and quantitative analysis.

2.1.1 Static Fault Trees

SFT's gates and structure expressions are explained in this section.

The Fault Tree Handbook shows traditional symbols for gates and events. Basic events are usually drawn as rectangle (for the text) and a circle below it. Top-level and intermediary events are drawn as a rectangle (for the text) and a gate below it. When an FT becomes too large, transfer in and out symbols can be used. They are usually drawn as triangles with a letter or a number. Figure 3 depicts traditional gates as specified in the Fault Tree Handbook, and Figure 4 shows an FT using the Fault Tree Analyser[1]—a free commercial tool. In this work, to keep a visual identity with other FTs, and to avoid symbols confusion, we use gates symbols as shown in Figure 5.

Structure expressions in FTA are defined in terms of set theory, using symbols for fault events occurrence. If a fault event symbol is in a set, then it means that fault has occurred. A set is a combination of fault events that causes the top-level event of a tree. A structure expression of a tree is denoted by a set of sets of fault event combinations. The OR gate becomes the union operator between sets and the AND gate, the intersection. For

[1] <http://www.fault-tree-analysis-software.com>, accessed 2/feb/2016

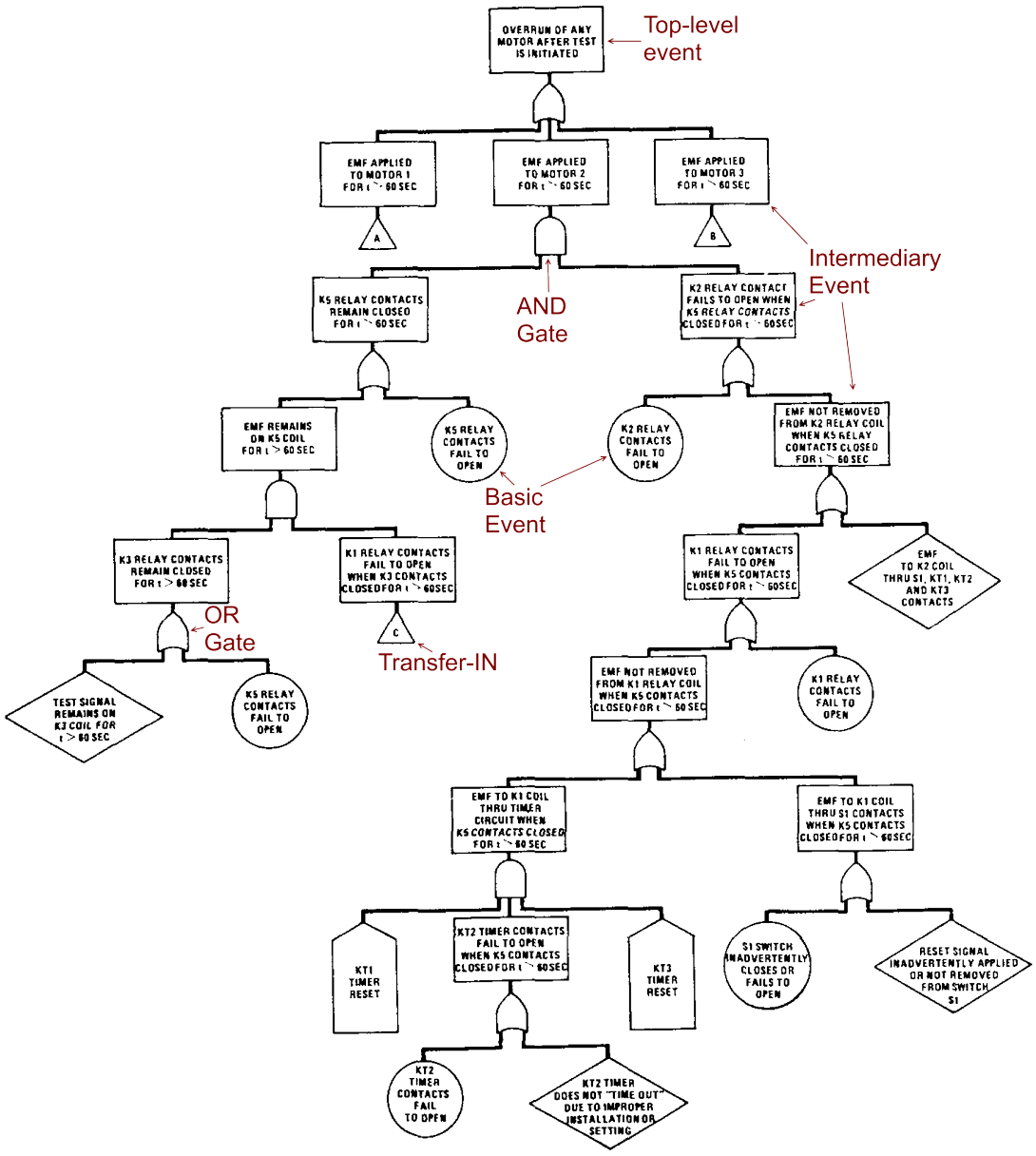


Figure 3 – SFT symbols as in the Fault Tree Handbook

example, if a system contains fault events a, b, and c, fault trees for this system contain at most all these three events. The occurrence of the fault event a is denoted by a set of sets A, which contains the following sets:

1. $\{a\}$: only a occurs;
2. $\{a, b\}$: a and b occur;
3. $\{a, c\}$: a and c occur;
4. $\{a, b, c\}$: all three events occur.

The fault tree in Figure 6 contains only two events and the resulting structure expression for this FT is the expression $A \cap B$ (TOP), where A and B are the sets of sets

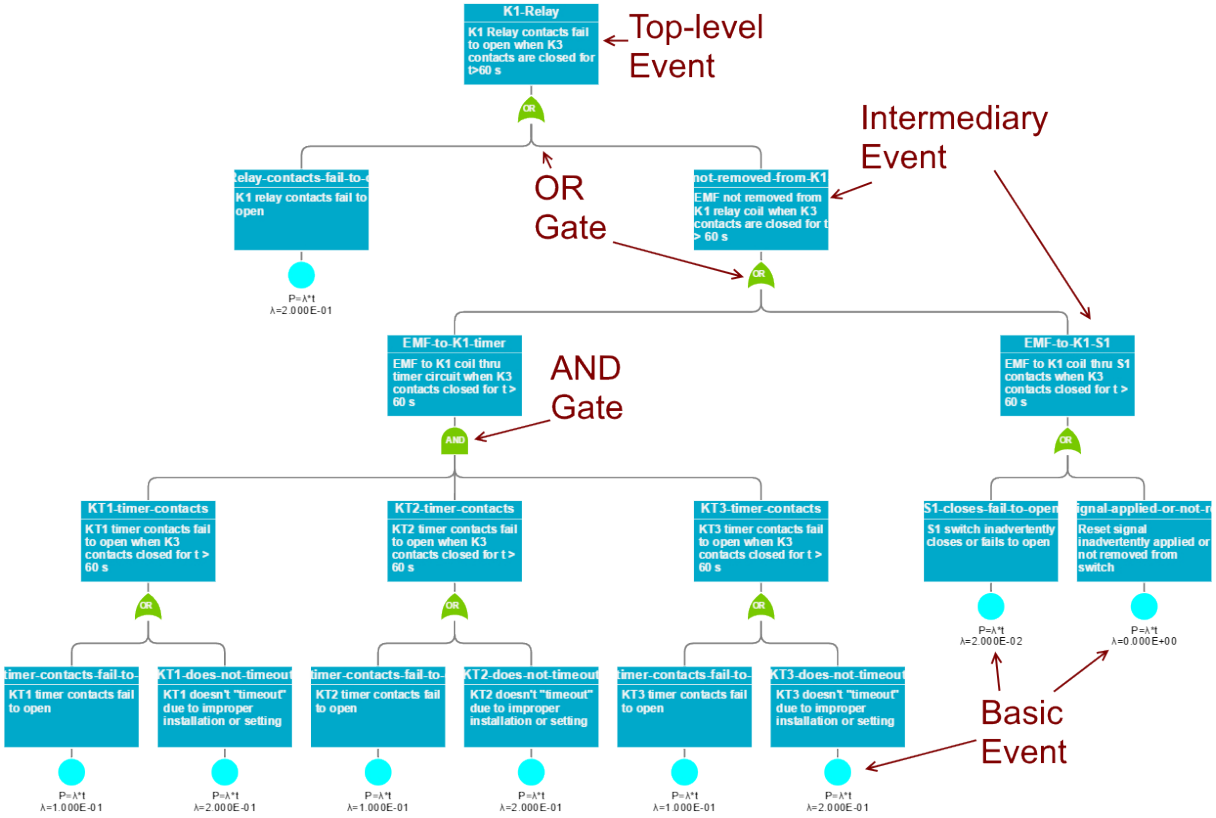
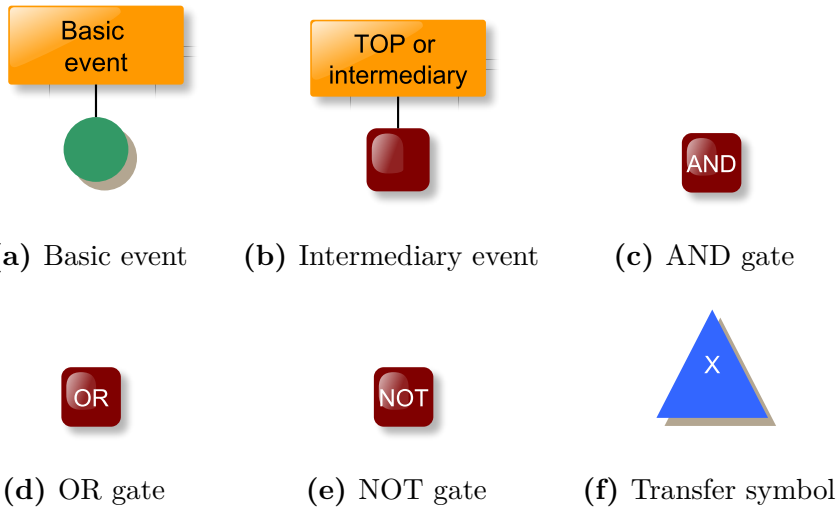


Figure 4 – SFT symbols using a free commercial tool



(a) Basic event **(b)** Intermediary event **(c)** AND gate

(d) OR gate **(e)** NOT gate **(f)** Transfer symbol

Figure 5 – SFT gates

that contain a and b, respectively. The resulting combinations for TOP are $\{a, b\}$ and $\{a, b, c\}$ (fault events a and b occur in all possibilities).

After obtaining structure expressions, the next step is to reduce the expressions to a canonical form to obtain the *MCSs*, which are the sets that contain the minimum and sufficient events to activate the top-level failure. Probabilistic analysis is then performed on these events to obtain the overall probability of occurrence of the top-level event.

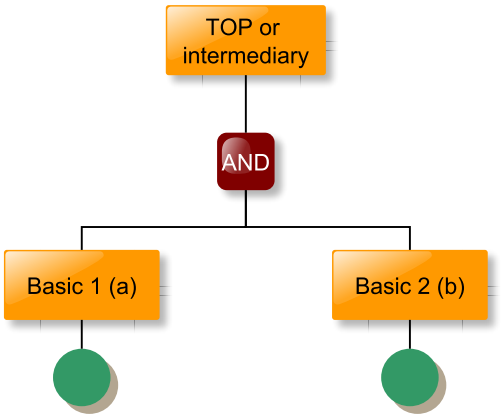


Figure 6 – Very simple example of a fault tree

The Fault Tree Handbook shows an algorithm based on Shannon's method to reduce structure expressions to obtain minimal cut sets. The Boolean expression of the tree shown in Figure 6 is $TOP = A \wedge B$. A technique called BDD—which derives from Shannon's method—is explained in Section 2.2.2.

2.1.2 Temporal Fault Trees

There are at least two versions of TFTs. One is described in (PALSHIKAR, 2002) and uses more traditional style of temporal logic (a variation of linear temporal logic (LTL)). The other version is ~~also~~ called Pandora, and is the one we refer to in the following.

TFTs were conceived taking a slightly different direction to DFT. They are focused on directly expressing ordering relationships rather than different states of a system. Basically they extend SFT's PAND gates, allowing analysis of FT with such gates. It is simpler to express than DFT, but lacks the fault-tolerance-related gate of DFTs (which we show in Section 2.1.3).

Structure expressions are also present in TFTs (WALKER; PAPADOPOULOS, 2009; WALKER, 2009; WALKER; PAPADOPOULOS, 2010), through the Pandora methodology. These expressions use the SFT operators OR and AND, and three new operators related to events ordering: PAND, Priority-OR (POR, |), and Simultaneous-AND (SAND, &). The semantics of the PAND in TFTs is similar to the semantics of the PAND described in the Fault Tree Handbook. To avoid ambiguous expressions, the semantics in TFTs is stated in terms of natural numbers, using a *sequence value* function. For every possible combination of events ordering, it assigns a sequence value to each fault event. For example, if event A occurs before event B, then the sequence value of A is lower than the sequence value of B, and one formula to express this is $A < B$.

An invariant on sequence values is that there are no gaps for assigned values. For

Table 2 – TTT of TFT's operators and sequence value numbers

A	B	AND	OR	PAND	POR	SAND
0	0	0	0	0	0	0
0	1	0	1	0	0	0
1	0	0	1	0	1	0
1	1	1	1	0	0	1
1	2	2	1	2	1	0
2	1	2	1	0	0	0

(a) PAND gate **(b)** POR gate **(c)** SAND gate

Figure 7 – TFT-specific gates

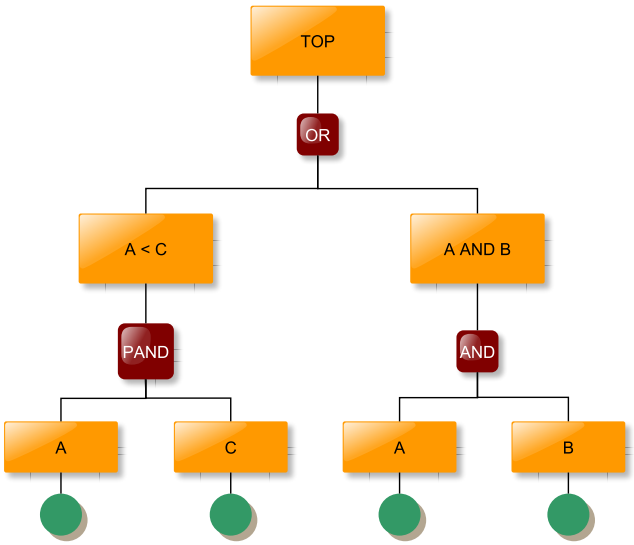


Figure 8 – TFT small example

example, if faults A and B occur at the same time and there are only these two events, then they should both be assigned value 1. On the other hand, if A occurs before B, then the assigned values are 1 and 2, respectively. Value zero means that the event is not active on the combination. Similar to Boolean's truth tables, the Pandora methodology defines Temporal Truth Tables (TTTs). They represent formula values for every combination of events. Table 2 shows the TTT of all TFT operators, and Figure 7 shows TFT-specific symbols used in this work. To illustrate TFTs, for the formula $(A < C) \vee (A \wedge B)$, we show: (i) the TFT in Figure 8, and (ii) its corresponding TTT in Table 3 (the column '#' indicates the MCSeq number).

Table 3 – TTT of a simple example

#	A	B	C	$A<C$	$A \wedge B$	$(A<C) \vee (A \wedge B)$
01	0	0	0	0	0	**0**
02	0	0	1	0	0	**0**
03	0	1	0	0	0	**0**
04	0	1	1	0	0	**0**
05	0	1	2	0	0	**0**
06	0	2	1	0	0	**0**
07	1	0	0	0	0	**0**
08	1	0	1	0	0	**0**
09	1	0	2	2	0	**2**
10	1	1	0	0	1	**1**
11	1	1	1	0	1	**1**
12	1	1	2	2	1	**1**
13	1	2	1	0	2	**2**
14	1	2	2	2	2	**2**
15	1	2	3	3	2	**2**
16	1	3	2	2	3	**2**
17	2	0	1	0	0	**0**
18	2	1	0	0	2	**2**
19	2	1	1	0	2	**2**
20	2	1	2	0	2	**2**
21	2	1	3	3	2	**2**
22	2	2	1	0	2	**2**
23	2	3	1	0	3	**3**
24	3	1	2	0	3	**3**
25	3	2	1	0	3	**3**

From structure expressions in order-sensitive FTs (TFT and DFT), MCSeqs are obtained. Several approaches represent MCSeq's ordering differently. For the best of our knowledge they are introduced in (TANG; DUGAN, 2004) similarly as MCS, allowing set elements with arrows ("$\rightarrow$") to represent order.

For TFTs, in (WALKER, 2009) MCSeqs are represented as a DNF using AND and the temporal operators (PAND, POR, and SAND) as doublets (a single temporal relation)—which are the minimal terms—or prime implicants—in the DNF. In a doublet, the expression is a product (of ANDs) of temporal operators.

The normal form for TFT is similar to SFT: it is a DNF with temporal operators (PAND, POR, SAND) in the minterms. The reduction of TFT structure expressions is achieved using dependency trees. In a dependency tree, if all children of a tree node are true, then the node is also true. Conversely, if a node is true, then all its children are also true. An issue with dependency trees is that they grow exponentially. Accordingly to the work reported in (WALKER; PAPADOPOULOS, 2010), it is already infeasible to deal with seven fault events in TFTs. The solution is based on a mixed application of dependency trees, modularisation of independent subtrees, and algebraic laws (WALKER; PAPADOPOULOS, 2009). We show dependency trees in Section 2.2.3. Some of these

algebraic laws are:

$$(X < Y) \vee (X \,\&\, Y) \vee (Y < X) = X \wedge Y \qquad \text{Conjunctive Completion Law} \qquad (2.1a)$$

$$(X \mid Y) \vee (X \,\&\, Y) \vee (Y \mid X) = X \vee Y \qquad \text{Disjunctive Completion Law} \qquad (2.1b)$$

$$(X \mid Y) \vee (X \,\&\, Y) \vee (Y < X) = X \qquad \text{Reductive Completion Law 1st} \qquad (2.1c)$$

$$(X \wedge Y) \vee (X \mid Y) = X \qquad \text{Reductive Completion Law 2nd} \qquad (2.1d)$$

2.1.3 Dynamic Fault Trees

Dynamic Fault Trees were designed with the goal of analysing complex systems with dynamic redundancy management and complex fault and recovery mechanisms (DUGAN; BAVUSO; BOYD, 1992). The idea was to create easy-to-use and less error-prone modelling tools than using DTMCs—or simply *Markov chains*—directly. So, since the very beginning, DFTs were intended to be evaluated using Markov chains. Figure 9 depicts the original gate symbols as shown in (DUGAN; BAVUSO; BOYD, 1992; BOYD, 1992). In this work, we use gates symbols as depicted in Figure 10. The informal semantics of them are:

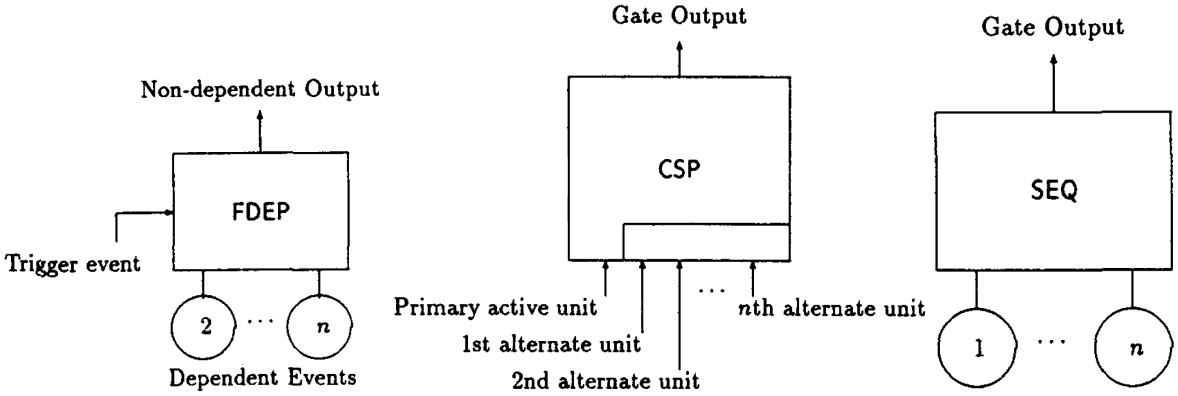
FDEP: When the trigger event occurs, the dependent events are forced to occur. Timing in this gate between trigger event and dependent events occurrences can be at the same time (like in TFT's SAND gate), or in a small amount of time, thus implying an order of occurrence, depending on the kind of dependency.

CSp: It is a specific gate to handle spare components. It is important to note that connected inputs are not components—they are fault events of connected components. If the i*th* input is already active (fault has occurred), then it is expected that the input $i+1$ is not, following the specified order. The output becomes true after all connected inputs become true. A spare event can be connected to more than one CSp gate, representing the spare unit connection to one or more components.

PAND: The same as in TFT: when the connected input events occur in the specified order, it outputs true.

SEQ: The connected events *shall* occur in the specified order. It is different from the PAND gate, because the latter *detects* the specified order. The usage of this gate is usually associated with FDEP.

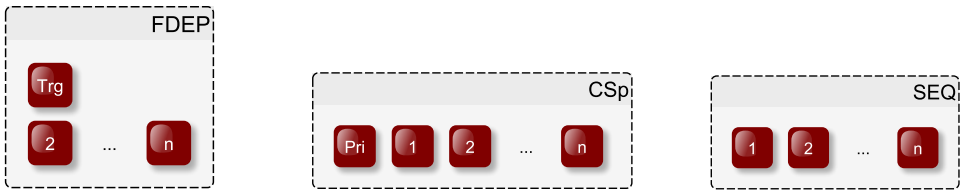
There are several means to analyse qualitative and quantitative ~~analysis~~. The works reported in (MERLE, 2010; MERLE et al., 2010; MERLE; ROUSSEL; LESAGE, 2011a; MERLE; ROUSSEL; LESAGE, 2014) use structure expressions to perform both qualitative and quantitative analysis, and the work reported in (MERLE; ROUSSEL; LESAGE, 2014) summarizes other approaches. We increment their summary (Table 4) and categorize them as:



(a) FDEP gate **(b)** CSp gate **(c)** SEQ gate

Figure 9 – DFTs's original gates symbols

Figure 10 – DFTs's gates symbols



(a) FDEP gate **(b)** CSp gate **(c)** SEQ gate

Source: (DUGAN; BAVUSO; BOYD, 1992; BOYD, 1992)

- Finding MCSeqs (qualitative analysis) is obtained by replacing DFT gates with SFT gates, using the text as its logical constraints. MCSs in the SFT are expanded using timing constraints from the texts into MCSeq. In this case, the behaviour of spare events cannot be correctly taken into account;

- Quantitative analysis is based in converting a DFT to a well-defined formalism to calculate the probability of its top-level event. Table 4 shows the conversion, the calculation, and where the method is explained.

In (MERLE, 2010; MERLE et al., 2010; MERLE; ROUSSEL; LESAGE, 2011a) fault events occur in a specific time and are instantaneous (similar to detected faults), stated through a "date-of-occurrence" function. As the "date-of-occurrence" function is stated in continuous time, the probability of two events occurring at the same time is negligible. In fact, useful information is obtained from the possibilities of relation in time of the occurrence of the events. DFT gates' algebraic model is summarized in Table 5. Structure expressions are written with an algebra that has operators OR and AND, and three new operators to express events ordering: (i) non-inclusive-before (NIBefore, $\lhd$), (ii) simultaneous ($\triangle$), and (iii) inclusive-before (IBefore, $\trianglelefteq$). The NI-Before and the simultaneous operators are similar to TFT's POR and SAND operators, respectively. The

Table 4 – DFT conversion to calculate probability of top-level event

Conversion	Calculation	Explained in
Automaton-like structure	continuous-time Markov chain (CTMC) (IANNELLI; PUGLIESE, 2014; ANDERSON, 2012)	(COPPIT; SULLIVAN; DUGAN, 2000)
Bayesian network (BN) (PEARL, 1985)	Inference algorithm (model-specific)	(BOBBIO et al., 2005)
Stochastic well-formed net (SWN) (CHIOLA et al., 1993) (a kind of coloured Petri-net (CPN) (JENSEN, 1987))	CTMC	(BOBBIO; RAITERI, 2004)
Sequential Binary Decision Diagram (SBDD) (TANNOUS; XING; DUGAN, 2011; XING; TANNOUS; DUGAN, 2012) (a modified version of BDD)	model-specific	(TANNOUS; XING; DUGAN, 2011; XING; TANNOUS; DUGAN, 2012)

IBefore is a composition of NI-Before and simultaneous operators. Table 6 summarizes the date-of-occurrence function for all operators. An infinite value means the event never occurs.

MCSeqs are extracted from canonical form of structure expressions written in a DNF. Minimal terms are products of variables and NI-Before operators (the other operators can be written as combinations of NI-Before). Structure expressions' reduction uses algebraic laws as, for example:

$$(a \lhd b) \vee (a \triangle b) \vee (b \lhd a) = a \vee b \tag{2.2a}$$

$$(a \wedge (b \lhd a)) \vee (a \triangle b) \vee (b \wedge (a \lhd b)) = a \wedge b \tag{2.2b}$$

$$(a \trianglelefteq b) \wedge (b \trianglelefteq a) = a \triangle b \tag{2.2c}$$

Figure 12 shows an example of DFT extracted from (MERLE; ROUSSEL; LESAGE, 2014). It is a cardiac assist system (HCAS) which is divided in four modules: trigger, CPU unit, motor section, and pumps. The trigger is divided in two components, CS and SS. The failure of any CS or SS, triggers a CPU unit failure. The primary CPU (P) has a warm[2] spare (B). The motor module fails if both M and MC fails. In order for the pumps unit to fail, all three pumps need to fail, and the left-hand side spare gate needs to fail before (or at the same time as) the right-hand side spare gate (PAND gate[3]). The top-level

[2] Warm spare gates only differ from CSp on the activation time.

[3] Although the original example uses a PAND gate, accordingly to the informal description, a SEQ gate would fit better.

Table 5 – Algebraic model of DFT gates with inputs A and B

Gate	Algebraic model of gate's output	Note
FDEP	$A_T = T \vee A$ and $B_T = T \vee B$	A_T and B_T replaces A and B on the resulting expression
CSp	$(B_a \wedge (A \lhd B_a)) \vee (A \wedge (B_d \lhd A))$	A is the active input, and B is the spare. Subscripts a and d represents component's state—*active* and *dormant*, respectively, which are used on the failure distribution formulas
PAND	$B \wedge (A \trianglelefteq B)$	

Table 6 – Date-of-occurrence function for operators defined in (MERLE, 2010)

Operator	Expression	Value if $\mathbf{d(a) < d(b)}$	Value if $\mathbf{d(a) = d(b)}$	Value if $\mathbf{d(a) > d(b)}$
OR	$d(a \vee b)$	$d(a)$	$d(a)$	$d(b)$
AND	$d(a \wedge b)$	$d(b)$	$d(a)$	$d(a)$
NI-Before	$d(a \lhd b)$	$d(a)$	$+\infty$	$+\infty$
simultaneous	$d(a \triangle b)$	$+\infty$	$d(a)$	$+\infty$
IBefore	$d(a \trianglelefteq b)$	$d(a)$	$d(a)$	$+\infty$

event structure expression is:

$$\begin{aligned} SYSTEM = & CS \vee SS \vee (M \wedge MC) \vee \\ & (P \wedge (B_d \lhd P)) \vee (B_a \wedge (P \lhd B_a)) \vee \\ & (BP_a \wedge (P2 \lhd P1) \wedge (P1 \lhd BP_a)) \vee (P2 \wedge (P1 \lhd BP_a) \wedge (BP_a \lhd P2)) \end{aligned} \tag{2.3}$$

2.2 Structure expressions analysis

In this section we detail the non-state-based methods to analyse structure expressions. Another common approach to analyse fault tree is to perform structure expression analysis based on algebraic laws. Boolean laws are well-known and are used for SFTs, temporal laws (WALKER, 2009; WALKER; PAPADOPOULOS, 2010) for TFTs, and the works reported in (MERLE, 2010; MERLE; ROUSSEL; LESAGE, 2011a) show laws for DFTs. An issue with algebraic laws is that in some cases, the expression needs to be expanded before it gets reduced, so reduction automation is not trivial without a theorem prover. For example, the following TFT's structure expression needs to be expanded

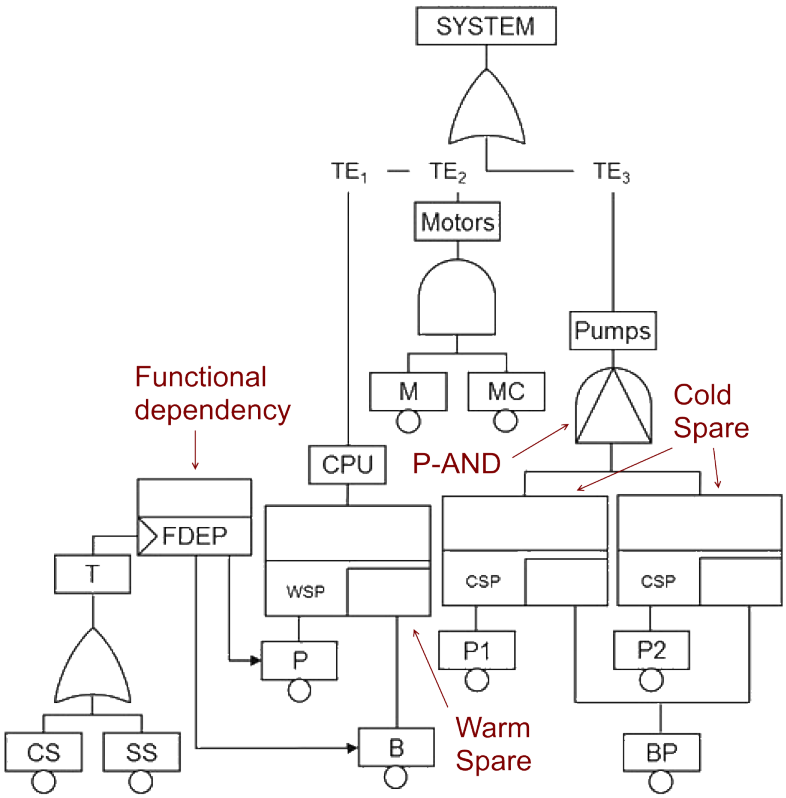


Figure 12 – DFT example

(WALKER; PAPADOPOULOS, 2010) before it gets reduced:

$$
\begin{aligned}
& (X \wedge Y) \vee ((X < Y) \wedge Z) \\
(X < Y) \vee (X \,\&\, Y) \vee (Y < X) \vee ((X < Y) \wedge Z) \quad & \text{by Eq. (2.1a)} \\
(X < Y) \vee (X \,\&\, Y) \vee (Y < X) & \\
X \wedge Y \quad & \text{by Eq. (2.1a)}
\end{aligned}
$$

A denotational semantics to Boolean expressions—and consequently to SFT—is given by FBAs (Section 2.3).

There are several works with state-based analysis for FTs (SFT, TFT, and DFT). We show some of them in Section 2.2.1.

2.2.1 State-based and temporal logic analysis

The work reported in (SCHELLHORN; THUMS; REIF, 2002) shows a formal approach to analyse SFT using Interval Temporal Logic (ITL) (MOSZKOWSKI, 1982). Instead of tackling basic events ordering (as in PAND), it considers causal relation over a gate, as for example, a relation of a basic event and a higher-level intermediary event.

For TFTs, the works reported in (MAHMUD; PAPADOPOULOS; WALKER, 2010; MAHMUD; WALKER; PAPADOPOULOS, 2012) shows an inverse solution. They map

Finite State Machines (FSMs) to Pandora logic, then verify system properties. They show that such a mapping simplifies expressions reduction, thus improving performance on the analysis.

Although there is formal modelling for DFTs, most of the works are state-based. The work reported in (COPPIT; SULLIVAN; DUGAN, 2000) shows a formal model for DFT analysis. It models Markov chains in Z Notation (Z) (SPIVEY, 1998), ~~as well as~~ each DFT element (basic events and gates). The analysis uses a quantifier on states of the resulting Markov chain automaton. The work reported in (GULATI; DUGAN, 1997) shows a methodology to perform a modular analysis of DFTs based on BDD and Markov chain. As DFT extends SFT, it identifies subtrees that are purely SFT and uses BDD, otherwise, it performs Markov chain analysis. Still on the state-based approaches, the work reported in (SIMEU-ABAZI; LEFEBVRE; DERAIN, 2011) maps DFTs to high-level Petri-net (HLPN) (JENSEN, 1983) to analyse false alarms.

In the following we show specific methods that are designed to reduce structure expressions. In essence, the methods are very similar. Structure expressions for SFTs can be reduced using BDDs (Section 2.2.2), TFTs can be reduced using dependency trees (Section 2.2.3), and the works reported in (TANNOUS; XING; DUGAN, 2011; XING; TANNOUS; DUGAN, 2012) show the analysis of standby systems (CSp gates) using SBDDs (Section 2.2.4).

2.2.2 Binary Decision Diagrams

BDDs are directed acyclic graphs that represents a Boolean expression. They are still referred to as BDD, but the more spread version is the Reduced Ordered Binary Decision Diagram (ROBDD) (BRACE; RUDELL; BRYANT, 1990), which is an optimisation. There are two ways to generate a BDD for an expression: (i) derive a diagram from the truth-table, or (ii) expand the paths based on Shannon's method (described in Fault Tree Handbook).

To demonstrate the expressiveness of a BDD, Figure 13 shows a diagram for a truth table with three variables (Table 7). In a node, when a path is chosen, the variable of the node assumes the edge value. For example, the top-level node variable of Figure 13 is A. Following the right-hand side of the node, all leaf nodes corresponds to the lines of the truth-table that A has "0" values (the first four lines). The symbol nodes are replaced by the values assumed by a specific formula.

Following Shannon's method, we choose a variable, and build the lower level BDD assuming the edge value for the chosen variable. In the remainder of the path, the variable's value is unchanged. For example, the expression $A \vee (\neg B \wedge C)$ has value "0" on lines a and c, and value "1" on the other lines. Choosing variable A first, then B and C, the resulting BDD with the binary values nodes (called sink nodes "false" and "true") for this formula

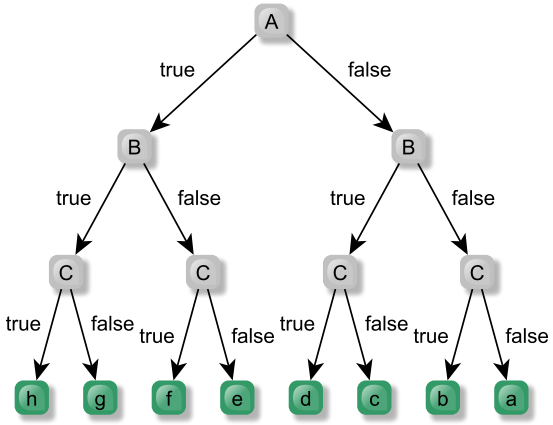


Figure 13 – A diagram for a truth table

Table 7 – Truth table for a formula outputs with three variables (A, B, and C)

A	B	C	Formula
0	0	0	a
0	0	1	b
0	1	0	c
0	1	1	d
1	0	0	e
1	0	1	f
1	1	0	g
1	1	1	h

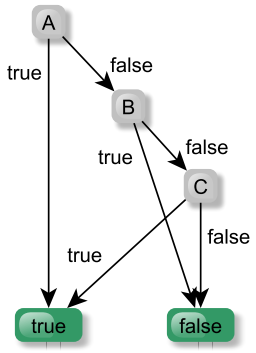


Figure 14 – A BDD for the expression $A \vee (\neg B \wedge C)$

is depicted in Figure 14. Starting from the top-level node A, the formula expressed by the BDD is true if A assumes value true. If A is false, and B is false, the expression is only true if C is true.

Figure 14 is a ROBDD. To be considered a ROBDD, the BDD must meet the following constraints (BRACE; RUDELL; BRYANT, 1990):

- the variables are assigned a constant ordering;
- every path to sink nodes visit the input variables in ascending order;

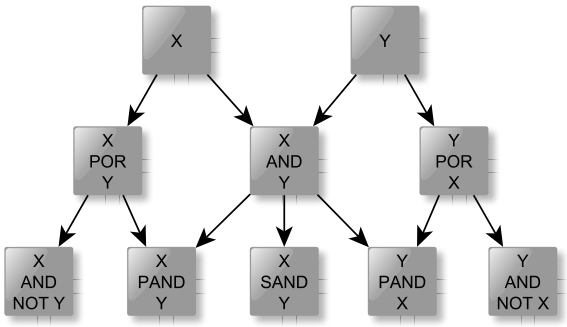


Figure 15 – Dependency tree for variables X and Y

- each node represents a distinct logic function.

The size of a ROBDD for a given expression depends on the chosen variable ordering. The work reported in (RUDELL, 1993) shows initial findings on variable order, and the work reported in (KISSMANN; HOFFMANN, 2014) shows heuristics to improve the performance for optimal order search.

For SFTs the evaluation of a BDD is the calculation of the probability of the paths ending in *true*. For ~~the~~ example of Figure 14, the probability of the expression is obtained from the formula: $P(A \vee (\neg A \wedge \neg B \wedge C))$. Note that the formula in the probability calculation is different from the formula that originated the diagram.

2.2.3 Dependency trees

A dependency tree is a hierarchical acyclic graph of expressions that shows all possible cut sequences for any given set of events. It is a graphical view of a TTT. At the top of a dependency tree are the variables, that is, the single events that occur in an expression. On the lower levels are the increasingly complex expressions. Each node represents a MCSeq. Figure 15 shows a dependency tree with all nodes for variables X and Y.

The reduction of a structure expression is given by activation (true values) of nodes. If a node is active (true), then all child nodes are also active; the converse is also true: if all node's children are active, then it is also active. The reduced expression is given by the DNF created with the expressions of active higher level nodes. To reduce the formula given on the beginning of this section, ~~the expression~~ $(X \wedge Y) \vee ((X < Y) \wedge Z)$, we create the dependency tree depicted in Figure 16. Nodes marked as "1" are those MCSeqs given directly by the formula. Nodes marked as "2" are child nodes of the "1"'s nodes, and so forth. The node of the expression $((X < Y) \wedge Z)$ is a grandchild of $X \wedge Y$, thus it is not necessary. The final expression if obtained by the active higher level node, which is $X \wedge Y$.

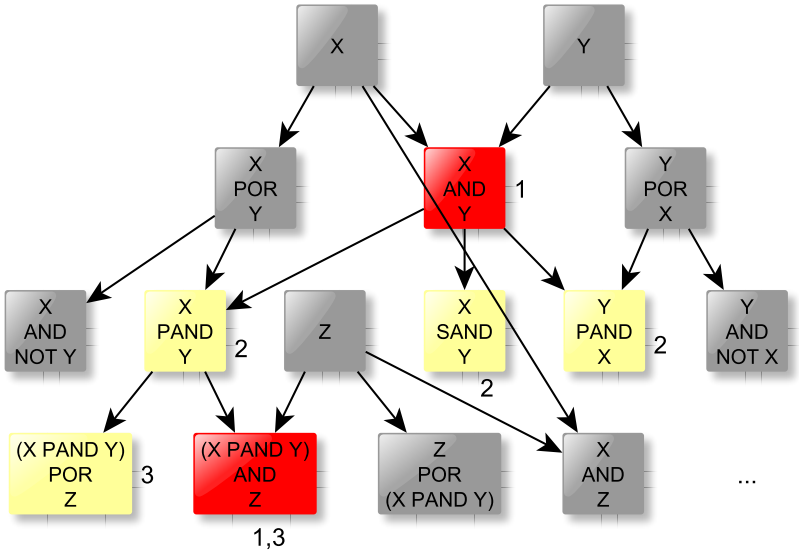


Figure 16 – Dependency tree for the formula $(X \wedge Y) \vee ((X < Y) \wedge Z)$

2.2.4 Sequential Binary Decision Diagrams

SBDD is an extension of BDD to tackle ordering of events in DFTs for CSp and WSp gates. Ordering of events in CSp gates (XING; TANNOUS; DUGAN, 2012) is slightly different compared to WSp (TANNOUS; XING; DUGAN, 2011). A backup system in CSp gets activated slower than in WSp, which implies that there are less failure possibilities in CSp, but its the readiness ~~is~~ lower than in WSp. SBDD adds a new node kind that contains a binary operation of fault events, which allows to express the ordering of events. One kind of operation expresses the slowness of the relation of the fault events of CSp, and another one expresses the readiness of the WSp. The latter semantics is similar to the semantics of PAND and IBefore (combined with AND) gates.

SBDD creation has two steps: (i) CSp or WSp DFT conversion, and (ii) SBDD model generation. In (i), it is a DFT-to-DFT conversion. CSp and WSp gates are converted to a new, but equivalent DFT without CSp and WSp gates, where the operations appear as basic events and are combined using other gates. In (ii), the SBDD model is created. The model may contain nodes that are contradictory as, for example, nodes that assumes that an event A is false and a binary operation that contains A is true. This step ends when all contradictions are removed. The evaluation is similar to BDD's: each path ending in true is a minimal term in the DNF that may contain one of the binary operations and individual events.

2.3 Free Boolean Algebras

Another means to analyse SFTs is to use an FBA to perform set-theoretical operations (intersection, difference, etc.) to reduce expressions. In this section we briefly present FBA theory and elements.

Instead of using an axiomatic definition of Boolean algebra, we follow its set-theoretical definition, as shown in (STOLL, 1979, pp. 254–258) and (GIVANT; HALMOS, 2009, pp. 8–11). This definition is more elegant because it represents a Boolean algebra as an algebra of sets and does not rely on axioms (which can be misleading, case there is a unfounded axiom).

Definition 2.1 (Boolean Algebra)**.** *A Boolean algebra is defined as a triple* $\langle B, \cap, - \rangle$*, where* B *is a set with at least two elements,* $\cap$ *is the intersection (also called meet or* infimum*) and* $-$ *is the complement (also called negation).*

The other Boolean elements (union, $\bot$, and $\top$) are derived from the other two operators:

$\cup$ is the union (also called join or *supremum*): $A \cup B = -(-A \cap -B)$

$\bot$ is the bottom (also called zero): $\bot = A \cap -A$

$\top$ is the top (also called unit): $\top = -\bot$

A Free Boolean Algebra is defined from a set E of generators. A generator can be represented as a proposition in statement calculus (STOLL, 1979, p. 274). For example, "valve A is stuck closed" and "motor X is malfunctioning" are valid statements. A Free Boolean Algebra is constructed from $\mathbb{P}(E)$, where $\mathbb{P}$ is the power set. Note that if E has n symbols, $\mathbb{P}(E)$ has 2^n elements, called *atoms* of a finite Boolean algebra. For the two statements above, the atoms are:

- "Valve A is stuck closed" and "motor X is malfunctioning"
- "Valve A is stuck closed" and "motor X is *not* malfunctioning"
- "Valve A is *not* stuck closed" and "motor X is malfunctioning"
- "Valve A is *not* stuck closed" and "motor X is *not* malfunctioning"

Such Boolean algebra has 2^{2^n} formulas (GIVANT; HALMOS, 2009, p. 261). For example, if $E = \{a, b\}$, then $\mathbb{P}(E) = \{\{\}, \{a\}, \{b\}, \{a, b\}\}$, hence the Boolean algebra generated by E contains sixteen (2^{2^2}) formulas: $\{\}$, $\{\{\}\}$, $\{\{\}, \{a\}\}$, $\{\{\}, \{b\}\}, \ldots$, $\{\{a\}, \{a, b\}\}, \ldots$, $\{\{b\}, \{a, b\}\}, \ldots$, $\{\{\}, \{a\}, \{b\}, \{a, b\}\}$.

The Boolean algebra B can be inductively defined using some constructs.

Definition 2.2 (Inductive Free Boolean Algebra)**.** *Let s be a statement, then:*

$$\mathbf{var}\, s = \{X | s \in X\} \implies \mathbf{var}\, s \in B \qquad \textit{(variable)} \tag{2.4a}$$

$$X \in B \implies -X \in B \qquad \textit{(complement)} \tag{2.4b}$$

$$X \in B \wedge Y \in B \implies X \cap Y \in B \qquad \textit{(intersection)} \tag{2.4c}$$

The characterisation of a "free" Boolean algebra comes from that, for some valuation function a, some of the formulas evaluates to "1". Given a function $p : B \times \{0, 1\} \to B$, such that:

$$p(i, j) = \begin{cases} i & j = 1 \\ -i & j = 0 \end{cases} \tag{2.5}$$

Lemma 2.1 (Free generators (valuation))**.** *Let F be a finite set, such that $F \subseteq E$, and $a : F \to \{0, 1\}$, a necessary and sufficient condition for a set E of generators of a Boolean algebra B to be free is then:*

$$\bigwedge_{i \in F} p(i, a(i)) \neq 0 \tag{2.6}$$

Proof. See (GIVANT; HALMOS, 2009, p. 258). □

Essentially, Lemma 2.1 states that there is no relation between generators, such as $a = -b$.

Lemma 2.2 (Free generators (algebraic))**.** *Let i and j be statements, such that $i, j \in E$, hence from Definition 2.2 and Lemma 2.1 it is necessary and sufficient that:*

$$\mathbf{var}\, i = \mathbf{var}\, j \iff i = j \tag{2.7a}$$

$$\mathbf{var}\, i \neq -\mathbf{var}\, j \tag{2.7b}$$

$$-\mathbf{var}\, i \neq \mathbf{var}\, j \tag{2.7c}$$

Proof. See (HUFFMAN, 2010, p. 4) □

2.4 Using NOT operator in fault trees

Although the Fault Tree Handbook introduces several gates, the vast majority of SFT analysis would fit in FTs with only AND and OR gates (coherent FTs). Qualitative analysis requires the reduction of the structure expression of FTs and, when NOT gates are present (non-coherent FTs). Such a reduction can cause the interpretation of failure logic to be misled (ANDREWS, 2001; OLIVA, 2006; ANDREWS; BEESON, 2003; CONTINI; COJAZZI; RENDA, 2008; VAURIO, 2016). The work reported in (OLIVA, 2006) shows three funny examples of this kind of problem, and the works reported in (ANDREWS, 2001; OLIVA, 2006; CONTINI; COJAZZI; RENDA, 2008) shows how to solve it using

BDDs. In the following we show: (i) the second example presented in (OLIVA, 2006), which highlights the problem when using NOT gates (Section 2.4.1), and (ii) the second example presented in (ANDREWS, 2001), which defends the usefulness of NOT gates in a multitasking system (Section 2.4.2).

Negated events in non-coherent analysis are in fact the working state of a component. The probability contribution of a negated basic event is close to 1. The problem with non-coherent FTs is that its analysis can cause impossible situations. The general formula to identify coherency is given in (ANDREWS, 2001; CONTINI; COJAZZI; RENDA, 2008) in terms of a structure function.

Definition 2.3 (FT Coherency). *Let* $\Phi(x) : B^n \to B^1$ *be a binary function of a vector of binary variables, such that* $x = [x_1, x_2, \ldots, x_n]$*, representing the states of* n *system's components.*

A binary structure function $\Phi(x)$ *is coherent if all the following hold:*

1. $\Phi(x)$ *is monotonic (non-decreasing) in each variable;*
2. *Each* x_i *is relevant, which means that* $\Phi(x)[x_i/1] \neq \Phi(x)[x_i/0]$ *for some vector* x*.*

where $B^1 = \{0, 1\}$, $B^n = B^{n-1} \times B^1$, $x_i = 1$ implies that component i is failed, and $\Phi(x) = 1$ implies the system is failed. For $y = [y_1, y_2, \ldots, y_n]$, monotonicity of Φ means that for *all* i, $x_i \geq y_i$ ($y_i = 1 \implies x_i = 1$), and for *some* i, $x_i > y_i$ ($x_i = 1$ and $y_i = 0$). Variable replacement ($[a/b]$) is as usual: $\Phi(x)[x_i/a] = [x_1, \ldots, x_{i-1}, a, x_{i+1}, \ldots, x_n]$

2.4.1 Non-coherent FT misleads

In this section we illustrate—with the second example detailed in (OLIVA, 2006)—how non-coherent FT misleads.

A college student who wants to visit her mother in another city has two options: wake up early (x_3) and take ride with a friend (x_1), or wake up late ($\neg x_3$) and take the metro (x_2). The top-event failure is "fail to visit mother" with expression $S = (x_1 \wedge x_3) \vee (x_2 \wedge \neg x_3)$. Its fault tree is depicted in Figure 17. It is clear that the structure function is non-coherent in x_3 accordingly to Definition 2.3: $\Phi(1, 1, x_3)[x_3/1] = \Phi(1, 1, x_3)[x_3/0]$.

The problem with this tree is the interpretation of the qualitative results. One of the possibilities in this scenario is that the college student would take a ride AND take the metro ($x_1 \wedge x_2$). Quantitatively, the analysis of the probabilities shows that this result is not negligible, but its interpretation is impossible.

2.4.2 Usefulness of NOT gates in FTA

In this section we show the second example detailed in (ANDREWS, 2001).

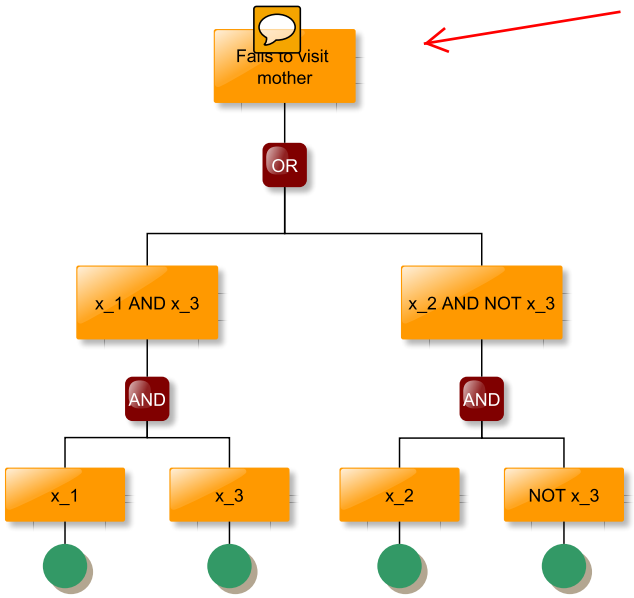


Figure 17 – Non-coherent FT college student's example

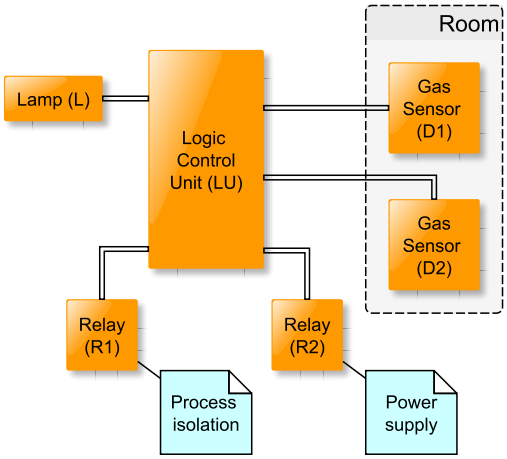


Figure 18 – Gas detection system

The gas detection system depicted in Figure 18 has two sensors D_1 and D_2 which are used to detect a leakage in a confined space. When a leakage is detected, these sensors send a signal to the logic control unit LU, which performs three tasks:

1. shuts-down the main system (process isolation) by de-energizing relay R_1;
2. inform the operator of the leakage by lamp and siren L;
3. deactivates all possible ignition sources, which is the interruption of power supply by de-energizing relay R_2.

The system is in fail state if does not perform one of these three tasks. The fault tree that represents this generic failure is depicted in Figure 19. G_1, G_2, and G_3 are subtrees

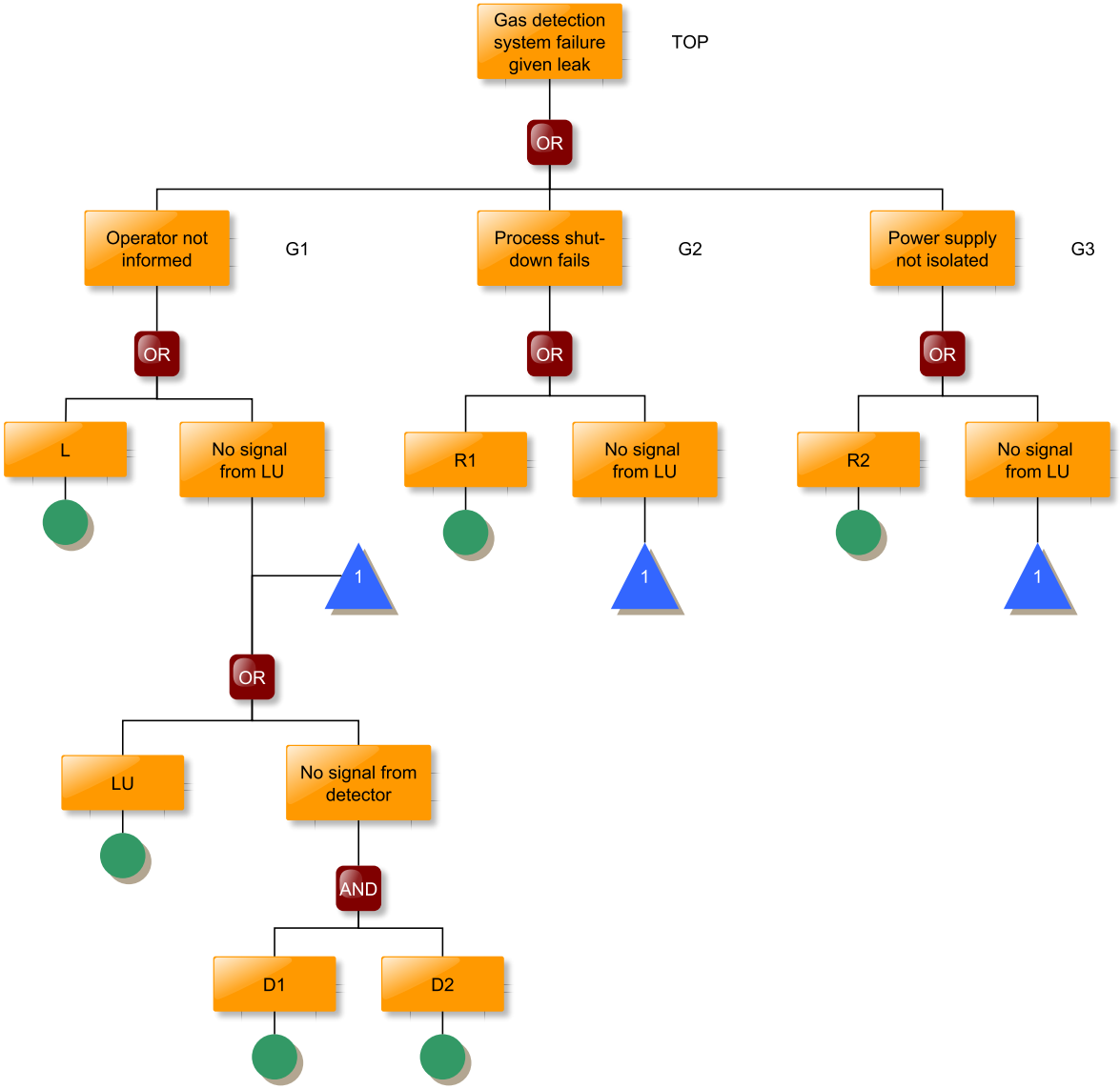


Figure 19 – FT for a generic failure in the gas detection system

that represents the three tasks "Operator not informed", "Process shut-down fails", and "Power supply not isolated", respectively. All three tasks will fail if their respective main component fails (L, R_1, and R_2) or there is no signal from LU (LU fails or both D_1 and D_2 fail). The structure expressions for the subtrees are:

$$G_1 = L \vee LU \vee (D_1 \wedge D_2)$$
$$G_2 = R_1 \vee LU \vee (D_1 \wedge D_2)$$
$$G_3 = R_2 \vee LU \vee (D_1 \wedge D_2)$$

Analysing in more detail, there are different degrees of system failure. There are eight outcomes (given the three tasks) and the most critical one is when both process shut-down (G_2) and power supply isolation (G_3) fail keeping energized upon a leakage, and the operator is not informed (G_1), but the operator information system is working (lamp

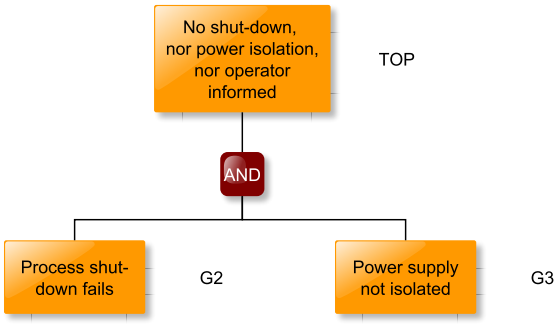


Figure 20 – *Coherent* FT for the most critical outcome of the gas detection system

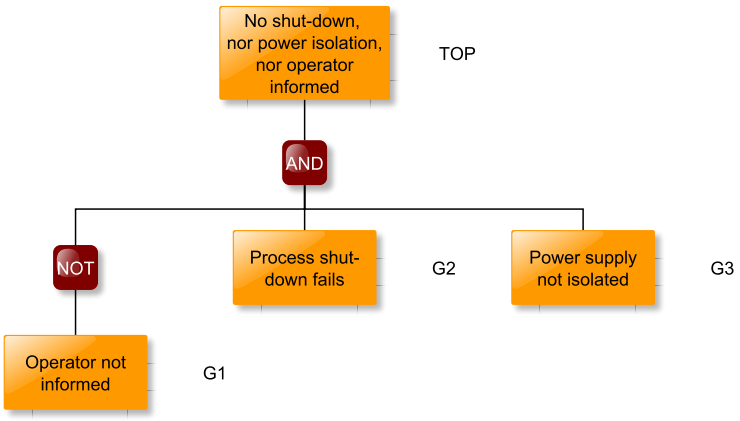


Figure 21 – *Non-coherent* FT for the most critical outcome of the gas detection system

and siren are off, but they are operational). The coherent FT of this outcome is depicted in Figure 20. The minimal cut sets obtained from this will be: $\{R_1, R_2\}$, $\{D_1, D_2\}$, and $\{LU\}$.

Quantification of the coherent FT will overestimate the probability of the critical outcome unless the part of the system that is working (lamp and siren L, LU, and sensors D_1 and D_2) is taken into account. The non-coherent FT with the working part is shown in Figure 21.

If the operator *can* be informed, then cut sets $\{D_1, D_2\}$ and $\{LU\}$ could not have occurred (see Figure 19), thus the correct qualitative analysis should consider only cut set $\{R_1, R_2\}$. Reducing the expressions of the non-coherent FT (Figure 21), we obtain the structure expression: $\neg L \wedge \neg LU \wedge R_1 \wedge R_2 \wedge (\neg D_1 \vee \neg D_2)$. The approximation for this expression, removing the negated events, gives the cut set $\{R_1, R_2\}$, which gives a correct quantitative analysis.

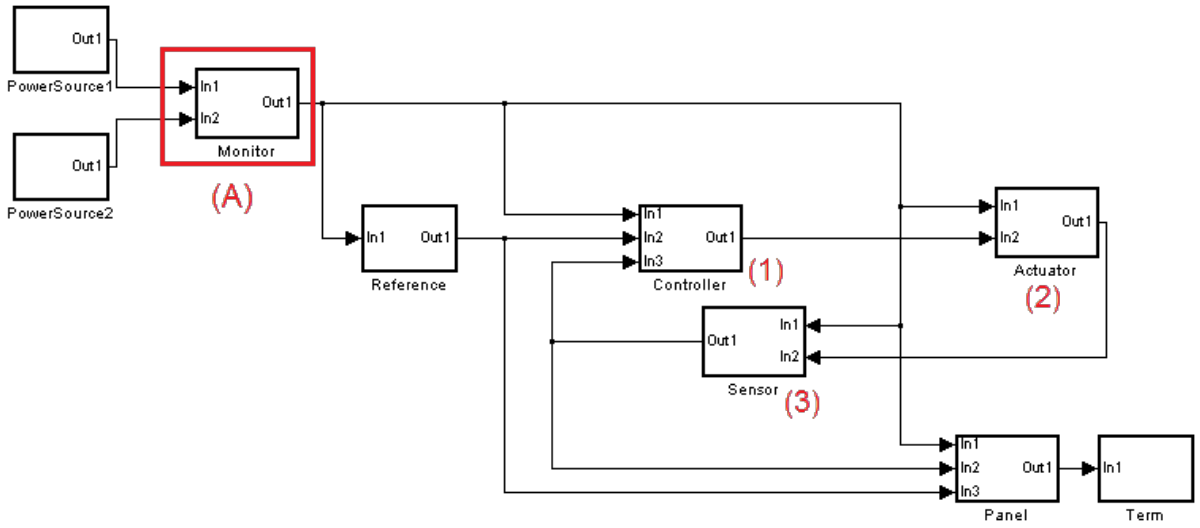


Figure 22 – Block diagram of the ACS provided by EMBRAER (nominal model)

2.5 Systems' nominal model and faults injection

Control system modelling using Simulink block diagrams (MATHWORKS, 2010b) is recommended in (NISE, 1992) and have been used by our industrial partner. It is a complementary tool of Matlab (MATHWORKS, 2010a). In fact, it works as a graphical interface to Matlab. A Simulink model has blocks and connections between these blocks, named signals. Each block has inputs and outputs and an internal behaviour expressed by its mathematical formula, which defines a function of the inputs for each output. There are many predefined blocks in the tool. It is also possible to create new blocks or use subsystems that encapsulate other blocks. A simulation adds extra parameters to a block diagram, like elapsed time and time between states. The elapsed time of a simulation is an abstraction for the quantity of possible simulation states and the time between states is related to the lowest common denominator of the sample time. Some components define different sample times, depending on their mode of operation. Usually, the value for this property is set to *auto*, allowing Simulink to choose a proper value automatically.

Nowadays, control systems are usually composed of an electromechanical part and a processor. Figure 22 shows the components of a feedback system (ASTROM; MURRAY, 2008) which was provided by EMBRAER. In this system, the feedback behaviour is given by the *Controller* (1), *Actuator* (2) and *Sensor* (3). A command is received by the *Controller*, which sends a signal to the *Actuator* to start its movement. The *Sensor* detects the actual position of the *Actuator* and sends it back to the *Controller*, which adjusts the given command to achieve the desired position. This loop (feedback) continues until the desired position given by the original command is reached.

Figure 23 shows the internal elements of the monitor component (Figure 22 (A)),

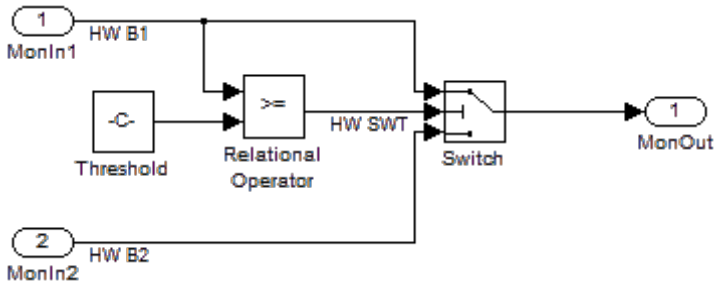


Figure 23 – Internal diagram of the monitor component (Figure 22 (A)).

which is used as case study in Chapter 4 to illustrate our strategy. The outputs of the hardware elements are annotated with HW, which are the two power sources and an internal component of the monitor (switch command).

To perform a formal verification in a Simulink system design model, the work reported in (JESUS et al., 2011) translates a Simulink model to the CSP$_M$ language. The resulting CSP$_M$ code is then used to check if it meets functional requirements also encoded in CSP$_M$.

In our previous work, reported in (DIDIER; MOTA, 2012), we modified such a translation to perform fault injection using hardware annotations allowing a subsystem or part to "break" randomly. We designed a CSP$_M$ process to act as an observer, watching outputs of the nominal version and comparing to the outputs of the "breakable" version (with injected faults) of the system. When the CSP$_M$ process of the model and the observer are loaded into the FDR model-checker, counter-examples are generated for each output that differs from the nominal model, thus obtaining a *sequence* of injected faults combinations that leads to the unexpected output, which are indeed *fault traces*.

In what follows, injected faults and the top-level failure have generic names based on the names of the Simulink model blocks. It is out of the scope of (DIDIER; MOTA, 2012) to define event names.

For the Simulink model shown in Figure 23, some representative fault traces are:

```
TRACE 1:
failure.Hardware.N04_RelationalOperator.1.EXP.B.true
failure.Hardware.N04_RelationalOperator.1.ACT.B.false
failure.Hardware.N04_MonIn2.1.EXP.I.5
failure.Hardware.N04_MonIn2.1.ACT.OMISSION
out.1.OMISSION

TRACE 2:
failure.Hardware.N04_MonIn2.1.EXP.I.5
failure.Hardware.N04_MonIn2.1.ACT.OMISSION
failure.Hardware.N04_RelationalOperator.1.EXP.B.true
failure.Hardware.N04_RelationalOperator.1.ACT.B.false
out.1.OMISSION
```

```
TRACE 3:
failure.Hardware.N04_MonIn1.1.EXP.I.5
failure.Hardware.N04_MonIn1.1.ACT.OMISSION
failure.Hardware.N04_MonIn2.1.EXP.I.5
failure.Hardware.N04_MonIn2.1.ACT.OMISSION
out.1.OMISSION

TRACE 4:
failure.Hardware.N04_MonIn2.1.EXP.I.5
failure.Hardware.N04_MonIn2.1.ACT.OMISSION
failure.Hardware.N04_MonIn1.1.EXP.I.5
failure.Hardware.N04_MonIn1.1.ACT.OMISSION
out.1.OMISSION

TRACE 5:
failure.Hardware.N04_MonIn1.1.EXP.I.5
failure.Hardware.N04_MonIn1.1.ACT.OMISSION
failure.Hardware.N04_RelationalOperator.1.EXP.B.false
failure.Hardware.N04_RelationalOperator.1.ACT.B.true
out.1.OMISSION

TRACE 6:
failure.Hardware.N04_MonIn1.1.EXP.I.5
failure.Hardware.N04_MonIn1.1.ACT.OMISSION
failure.Hardware.N04_RelationalOperator.1.EXP.B.false
failure.Hardware.N04_RelationalOperator.1.ACT.B.true
failure.Hardware.N04_MonIn2.1.EXP.I.5
failure.Hardware.N04_MonIn2.1.ACT.OMISSION
out.1.OMISSION

TRACE 7:
failure.Hardware.N04_MonIn1.1.EXP.I.5
failure.Hardware.N04_MonIn1.1.ACT.OMISSION
failure.Hardware.N04_MonIn2.1.EXP.I.5
failure.Hardware.N04_MonIn2.1.ACT.OMISSION
failure.Hardware.N04_RelationalOperator.1.EXP.B.false
failure.Hardware.N04_RelationalOperator.1.ACT.B.true

TRACE 8:
failure.Hardware.N04_MonIn2.1.EXP.I.5
failure.Hardware.N04_MonIn2.1.ACT.OMISSION
failure.Hardware.N04_MonIn1.1.EXP.I.5
failure.Hardware.N04_MonIn1.1.ACT.OMISSION
failure.Hardware.N04_RelationalOperator.1.EXP.B.false
failure.Hardware.N04_RelationalOperator.1.ACT.B.true
```

where `N04` is the subsystem name of the monitor in the Simulink diagram, `MonIn1` (first input of the monitor), `MonIn2` (second input of the monitor), and `RelationalOperator` (switcher controller) are the names of the hardware components in the Simulink diagram.

We only show eight counter-examples, but FDR generates a total of 64 counter-examples for this system. The other counter-examples are similar to the traces shown with different internal events.

To reuse HiP-HOPS, which is based on SFTs, each fault trace is abstracted as a conjunction (AND combination of the inner events), and the several conjunction-based fault events are combined using OR s (disjunctions). The result of the combination is a Boolean expression that represents the conditions that cause an undesirable output, the failure logic of the model.

If the failure logic is obtained for a whole system, it is indeed the structure expression of a fault tree for a general failure as the top-level event. Although it is possible to obtain the failure logic for a larger system, it may be impractical due to state-space explosion in CSP_M model analysis. Thus it should be used for components and subsystems or small systems following HiP-HOPS compositional structure. Using failure logic as subsystem annotations in (PAPADOPOULOS et al., 2001), it is possible to obtain structure expressions for a larger system. It is worth noting that the goal of the work reported in (DIDIER; MOTA, 2012) was to connect with HiP-HOPS, which is based on static fault trees. But we already knew that we had a richer fault modelling information than that presented in (DIDIER; MOTA, 2012) because we abstracted traces (which already capture fault events ordering) to create propositions (any fault events order combination).

To show how these traces become failure logic, let us abbreviate fault names as:

```
A = failure.Hardware.N04_MonIn1.1
B = failure.Hardware.N04_MonIn2.1
S = failure.Hardware.N04_RelationalOperator
```

So, for each trace, we obtain an expression:

$$\begin{aligned}
\texttt{TRACE 1} &= S \wedge B \\
\texttt{TRACE 2} &= B \wedge S \\
\texttt{TRACE 3} &= A \wedge B \\
\texttt{TRACE 4} &= B \wedge A \\
\texttt{TRACE 5} &= A \wedge S \\
\texttt{TRACE 6} &= A \wedge S \wedge B \\
\texttt{TRACE 7} &= A \wedge B \wedge S \\
\texttt{TRACE 8} &= B \wedge A \wedge S
\end{aligned}$$

And we combine them as a single Boolean expression: $\texttt{TRACE 1} \vee \texttt{TRACE 2} \vee \texttt{TRACE 3} \vee \texttt{TRACE 4} \vee \texttt{TRACE 5} \vee \texttt{TRACE 6} \vee \texttt{TRACE 7} \vee \texttt{TRACE 8}$, which by a traditional Boolean reduction strategy results in:

$$(A \wedge B) \vee (S \wedge (A \vee B))$$

Table 8 – Annotations table of the ACS provided by EMBRAER

Component	Deviation	Port	Annotation
PowerSource	LowPower	Out1	PowerSourceFailure
Monitor	LowPower	Out1	(SwitchFailure AND (LowPower-In1 OR LowPower-In2)) OR (LowPower-In1 AND LowPower-In2)
Reference	OmissionSignal	Out1	ReferenceDeviceFailure OR LowPower-In1

The above expression is exactly the same failure logic expression provided by EMBRAER if we use the following association (Table 8):

$$A = \text{LowPower-In1}$$
$$B = \text{LowPower-In2}$$
$$S = \text{SwitchFailure}$$

Note that when we combine each fault with AND s, we lose the information about order[4]: $S \wedge B$ and $B \wedge S$ are equal, due to the commutative law of Boolean expressions.

Our strategy finds fault combinations S and B (in the sense of S occurring before B) as well as B and S (in the sense of B occurring before S) but abstracts this ordering information obtaining B and S, which is equivalent to S and B in Boolean Algebra. If A fails before S, the system fails because it should switch to B, but the switcher is in a faulty state. On the other hand, if S fails before A, the switcher fails because it inadvertently switched to B when A was still operational. When A fails, nothing changes and the output of the system is obtained from B.

We also employed the strategy proposed in the work (DIDIER; MOTA, 2012) in another case study and obtained a weaker failure expression (that is, our expression considers more cases). The failure expression provided by the engineers of our industrial partner was stronger because they considered that one component has a very low probability of failure and removed it from the failure analysis. Although acceptable, it may cause incorrect analysis. Our strategy avoids this kind of issue by being completely systematic.

2.6 Isabelle/HOL

From the site[5] of the creators:

[4] In our previous work we designed the observer to ignore order as well, by making similar traces—with different ordering—the same size. Here we modified the observer specification to make similar traces with different sizes.

[5] Accessed 27/jan/2016: <https://isabelle.in.tum.de/overview.html>

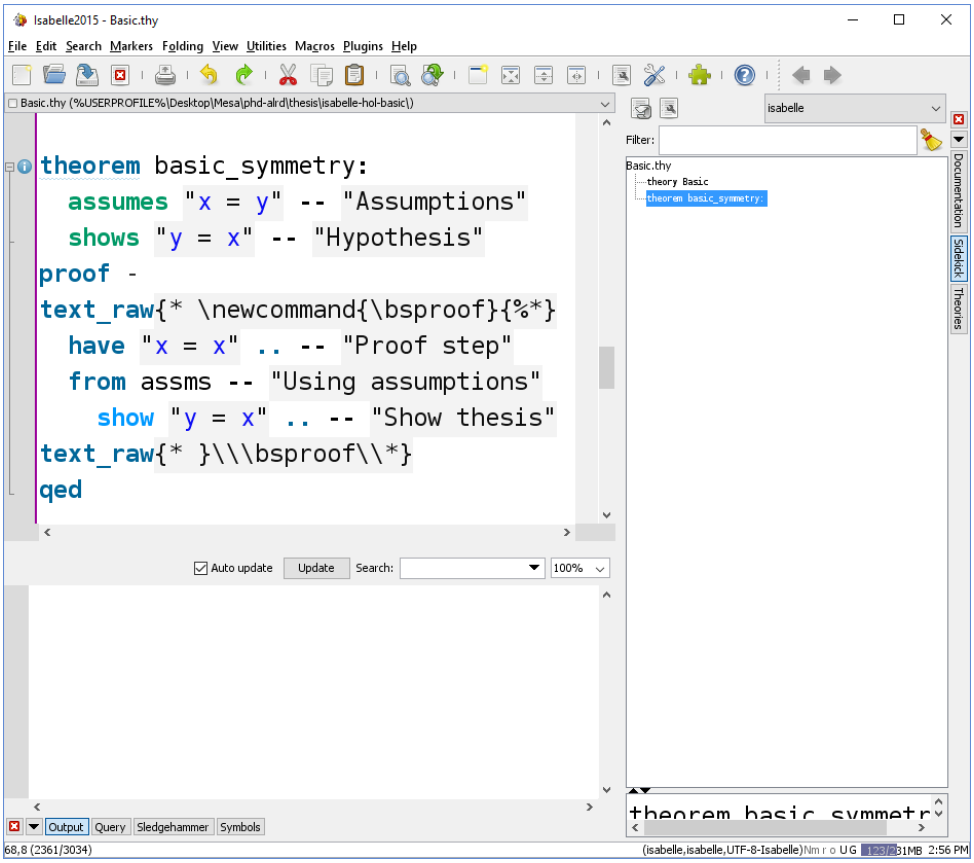


Figure 24 – Isabelle/HOL window, showing the basic symmetry theorem

> Isabelle is a generic proof assistant. It allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas in a logical calculus. The main application is the formalization of mathematical proofs and in particular formal verification, which includes proving the correctness of computer hardware or software and proving properties of computer languages and protocols.

Isabelle/HOL is the most widespread instance of Isabelle. HOL stands for higher-order logic. Isabelle/HOL provides a HOL proving environment ready to use, which include (co)datatypes, inductive definitions, recursive functions, locales, custom syntax definition, etc. Proofs can be written in both human[6] and machine-readable language based in Isar. The tool also includes the *sledgehammer*, a port to call external first-order provers to find proofs automatically. The user interface is based in jEdit[7], which provides a text editor, syntax parser, shortcuts, etc (see Figure 24).

Theories in Isabelle/HOL are based in a few axioms. Isabelle/HOL Library's theories—that comes with the installer—and user's theories are based on these axioms.

[6] By human we mean that anyone with mathematics and logic basic knowledge—it means that deep programming knowledge is not essential.

[7] Accessed 27/jan/2016: <http://www.jedit.org/>

This design decision avoids inconsistencies and paradoxes (similar as it is in Z).

Besides the provided theories, its active community provides a comprehensive archive of formal proofs[8] (AFP). Each entry in this archive can be cited and usually contains an *abstract*, a document, and a theory file. For example, a Free Boolean Algebra theory is available in (HUFFMAN, 2010). To use it, it is enough to download and put on the same directory of your own theory files.

Bellow we show an example and explain the overall syntax of the human and machine-readable language.

```
theorem basic_symmetry:
  assumes "x = y" — Assumptions
  shows "y = x" — Hypothesis
proof -
have "x = x" .. — Proof step
  from assms — Using assumptions
    show "y = x" .. — Show thesis
qed
```

Finally, Isabelle/HOL provides LaTeX syntax sugar and allow easy document preparation: this entire section was written in a theory file mixing Isabelle's and LaTeX's syntax). The above theorem can be written using Isabelle's quotation and anti-quotations. For example, we can write it using usual LaTeX theorem environment:

Theorem 2.1 (Basic symmetry)**.** *Assuming* `x = y`*, thus:*

$$y = x$$

Proof. **have** "x = x" .. — Proof step
from assms — Using assumptions
show "y = x" .. — Show thesis □

Otherwise specified, in the next sections we will omit proofs because they are all verified using Isabelle/HOL. The complete listing is in Appendix A.

[8] Accessed 27/jan/2016: <http://afp.sourceforge.net/>

Part II

Results

3 A free algebra to express structure expressions of ordered events

There are some hints to use sets of sequences mixed with sets of variables (as in SFTs) as shown in (TANG; DUGAN, 2004). Differently, we use a uniform representation of sets of sequences. Considering performance, instead of using the sets of sequences directly, we use a symbolic representation using propositions for systems and FTs.

Recall from Sections 1.2 and 2.1 that fault events are independent on one another if the events are not susceptible to a common cause. The set-theoretical abstraction of structure expressions for SFTs (VESELY et al., 1981, pp. VI-11) is very close to an FBA, where each generator in FBAs corresponds to a fault event symbol in fault trees. In FBAs, as generators are "free", they are independent on one another and Boolean formulas are written as a set of sets of possibilities, which are similar to the structure expressions of SFTs.

The set of sets for FBAs are the denotational semantics for Boolean algebras. We use the concept of generators to define the denotational semantics of ATF using a set of lists without repetition (distinct lists). The choice of lists is because this structure inherently associates a generator to an index, making implicit the representation of order. These lists are composed by non-repeated elements (distinct lists) because the events in fault trees are non-repairable, thus they do not occur more than once.

This list representation is different from the Sequence Number function used in (WALKER; PAPADOPOULOS, 2009; WALKER, 2009), but is related to the concept that there should be no gaps between consecutive events occurrence. It is different because order 0 (zero) in (WALKER; PAPADOPOULOS, 2009; WALKER, 2009) means non-occurrence. It may cause a discontinuity because 0 to 1 is different of 1 to 2. In FBAs the non-occurrence of an event is just the absence of the event, thus we use the same representation of non-occurrence in ATF to avoid this discontinuity.

We shown in Section 2.1 that there is an omnipresence of order-based operators to analyse TFTs and DFTs. And that each approach describes a new algebra based on different representations of events ordering with similar theorems to reduce expressions to a canonical form.

From the need to tackle events ordering and from the ordering information we had from fault injection that we developed in (DIDIER; MOTA, 2012), we defined a lists-based algebra, called ATF, to express and analyse systems considering events ordering. We also provide a mapping from fault traces (DIDIER; MOTA, 2012) (from CSP_M models) to this

algebra. The order-specific operations are expressed with a new operator ($\rightarrow$) that we call XBefore (or exclusive before).

In the following we show the definitions and laws of our proposed ATF. To avoid repetition, let S, T and U be sets of distinct lists. A list xs is distinct if it has no repeated element. So, if x is in xs, then it has a unique associated index i and we denote it as $x = xs_i$. Furthermore, as we follow an FBA characterisation, we also need to show that the generators are independent.

The ATF form a free algebra, similarly to FBAs. *Infimum* and *Supremum* are defined as set intersection ($\cap$) and union ($\cup$) respectively. The order within the algebra is defined with set inclusion ($\subseteq$).

To distinguish the permutations that are not defined in FBA, we need a new operator. We give the definition of XBefore ($\rightarrow$) in terms of list concatenation, similar to the work reported in (DIDIER; MOTA, 2015):

$$S \rightarrow T = \{zs | \exists xs, ys \bullet (\mathbf{set}\, xs) \cap (\mathbf{set}\, ys) = \{\} \wedge xs \in S \wedge ys \in T \wedge zs = xs \mathbin{@} ys\} \quad (3.1)$$

where the **set** function returns the set of the elements of a list, and @ concatenates two lists.

In some cases it is more intuitive to use the XBefore definition in terms of lists slicing because it uses indexes explicitly. Lists slicing is the operation of taking or dropping elements, obtaining a sublist. In slicing, the starting index is inclusive, and the ending is exclusive. Thus the first index is 0 and the last index is the list length. For example, the list $xs_{[i..|xs|]}$ is equal to the xs list, where $|xs|$ is the list length. We use the following notation for list slicing:

$$xs_{[i..j]} = \text{starts at } i \text{ and ends at } j - 1 \quad (3.2a)$$

$$xs_{[..j]} = xs_{[0..j]} \quad (3.2b)$$

$$xs_{[i..]} = xs_{[i..|xs|]} \quad (3.2c)$$

List slicing and concatenation are complementary: concatenating two consecutive slices results in the original list:

$$\forall i \bullet xs_{[..i]} \mathbin{@} xs_{[i..]} = xs \quad (3.3)$$

There is an equivalent definition of XBefore with concatenation using lists slicing:

$$S \rightarrow T = \left\{zs | \exists i \bullet zs_{[..i]} \in S \wedge zs_{[i..]} \in T\right\} \quad (3.4)$$

A variable in ATF is defined by one generator, and denotes its occurrence:

$$\mathbf{var}\, x = \{zs | x \in zs\} \quad (3.5)$$

The following expressions are sufficient to define the ATF in terms of an inductively defined set (**atf**):

$$\mathbf{var}\, x \in \mathrm{ATF}set \qquad \text{Variable} \tag{3.6a}$$

$$S \in \mathbf{atf} \implies -S \in \mathbf{atf} \qquad \text{Complement, Negation} \tag{3.6b}$$

$$S \in \mathbf{atf} \wedge T \in \mathbf{atf} \implies S \cap T \in \mathbf{atf} \qquad \text{Intersection, } \textit{Infimum} \tag{3.6c}$$

$$S \in \mathbf{atf} \wedge T \in \mathbf{atf} \implies S \to T \in \mathbf{atf} \qquad \text{XBefore} \tag{3.6d}$$

Following the definitions, the expressions below are also valid for **atf**:

$$\mathrm{UNIV} \in \mathbf{atf} \qquad \text{Universal set, True} \tag{3.6e}$$

$$\{\} \in \mathbf{atf} \qquad \text{Empty set, False} \tag{3.6f}$$

$$S \in \mathbf{atf} \wedge T \in \mathbf{atf} \implies S \cup T \in \mathbf{atf} \qquad \text{Union, } \textit{Supremum} \tag{3.6g}$$

The following expressions are valid for generators a and b and are sufficient to show that the generators are independent:

$$\mathbf{var}\, a \subseteq \mathbf{var}\, b \iff a = b \tag{3.7a}$$

$$\mathbf{var}\, a = \mathbf{var}\, b \iff a = b \tag{3.7b}$$

$$\mathbf{var}\, a \not\subseteq -\,\mathbf{var}\, b \tag{3.7c}$$

$$\mathbf{var}\, a \neq -\,\mathbf{var}\, b \tag{3.7d}$$

$$-\,\mathbf{var}\, a \not\subseteq \mathbf{var}\, b \tag{3.7e}$$

$$-\,\mathbf{var}\, a \neq \mathbf{var}\, b \tag{3.7f}$$

Expressions (3.6a) to (3.6g) and (3.7a) to (3.7f) implies that the ATF without the XBefore operator (3.1) forms a Boolean algebra based on sets of lists. And this is also equivalent to an FBA with the same generators.

In our previous work (DIDIER; MOTA, 2015) we stated a relation of XBefore and *supremum*, provided the operands are variables (3.5). Now we generalise this relation in terms of abstract properties of the operands of the XBefore. We name these properties as *temporal properties*.

3.1 Temporal properties (*tempo*)

Temporal properties give a more abstract and less restrictive shape on the XBefore laws. These properties avoid the requirement that every operand of XBefore should be a variable (3.5).

The first temporal property is about disjoint split. If the first part of a list is in a given set, then every remainder part is not. So, if a generator is in the beginning of a list,

it must not be at the ending (and vice-versa).

$$\mathbf{tempo}_1\, S = \forall i, j, zs \bullet i \leq j \implies \neg \left(zs_{[..i]} \in S \wedge zs_{[j..]} \in S \right) \tag{3.8a}$$

$$\mathbf{tempo}_2\, S = \forall i, zs \bullet zs \in S \iff zs_{[..i]} \in S \vee zs_{[i..]} \in S \tag{3.8b}$$

$$\mathbf{tempo}_3\, S = \forall i, j, zs \bullet j < i \implies \left(zs_{[j..i]} \in S \iff zs_{[..i]} \in S \wedge zs_{[j..]} \in S \right) \tag{3.8c}$$

$$\mathbf{tempo}_4\, S = \forall zs \bullet zs \in S \iff (\exists i \bullet zs_{[i..(i+1)]} \in S) \tag{3.8d}$$

The second temporal property is about belonging to one sublist in the beginning or in the end. If a generator is in a list, then it must be at the beginning or at the ending.

The third temporal property is about belonging to one sublist in the middle. If a generator belongs to a sublist between i and j, then it belongs to the sublist that starts at first position and ends in j and to the sublist that starts at i and ends at the last position (both sublists contain the sublist in the middle).

Finally, if a generator belongs to a list, then there is a sublist of size one that contains the generator.

Variables have all four temporal properties. For a generator x, the following is valid:

$$\mathbf{tempo}_1\, (\mathbf{var}\, x) \wedge \mathbf{tempo}_2\, (\mathbf{var}\, x) \wedge \mathbf{tempo}_3\, (\mathbf{var}\, x) \wedge \mathbf{tempo}_4\, (\mathbf{var}\, x)$$

In our previous work (DIDIER; MOTA, 2015) we used set difference to specify the XBefore operator. Provided $\mathbf{tempo}_1\, S$ and $\mathbf{tempo}_1\, T$, XBefore in (DIDIER; MOTA, 2015) is equivalent to (3.1):

$$S \to T = \{zs | \exists xs, ys \bullet xs \in S - T \wedge ys \in T - S \wedge \text{distinct}\, zs \wedge zs = xs \mathbin{@} ys\} \tag{3.9}$$

Other expressions also meet one or more temporal properties:

$$\mathbf{tempo}_1\, S \wedge \mathbf{tempo}_1\, T \implies \mathbf{tempo}_1\, (S \cap T) \tag{3.10a}$$

$$\mathbf{tempo}_3\, S \wedge \mathbf{tempo}_3\, T \implies \mathbf{tempo}_3\, (S \cap T) \tag{3.10b}$$

$$\mathbf{tempo}_2\, S \wedge \mathbf{tempo}_2\, T \implies \mathbf{tempo}_2\, (S \cup T) \tag{3.10c}$$

$$\mathbf{tempo}_4\, S \wedge \mathbf{tempo}_4\, T \implies \mathbf{tempo}_4\, (S \cup T) \tag{3.10d}$$

3.2 XBefore laws

We now show some laws to be used in the algebraic reduction of ATF formulas. The laws follow from the definition of XBefore, from events independence, and from the temporal properties.

We use a normal form similar to the disjunctive normal form (DNF) of Boolean algebra. In DNF each sub-expression is a minimal cut set for SFT. In our normal form, also called DNF, we allow AND s, NOT s, and exclusive-before (XBefore, $\rightarrow$) s to be in the sub-expressions. Each sub-expression is a set of minimal cut sequences for TFT and DFT. The following formulas are in DNF:

$$(A \cap -B) \cup ((A \rightarrow B) \cap C)$$
$$A \cup B$$
$$A \rightarrow B$$
$$A \cap B$$
$$A \rightarrow B \rightarrow C$$

The following formulas are *not* in DNF:

$$-(A \cup B)$$
$$A \cap (B \cup C)$$
$$A \rightarrow (B \cup C)$$
$$A \rightarrow (B \cap C)$$

But to transform the last two formulas into DNF, one can use Laws (3.14a), (3.14b), (3.14c) and (3.14d), for instance.

We define events independence ($\diamond\!\!\!\!\!\!\diamond$) as the property that one operand does not imply the other. For example, we need to avoid that the operands of XBefore are $\mathbf{var}\, a$ and $\mathbf{var}\, a \cup \mathbf{var}\, b$ (it results in $\{\}$, see (3.12e)).

$$S \diamond\!\!\!\!\!\!\diamond T = \forall i, zs \bullet \neg \left(zs_{[i..(i+1)]} \in S \wedge zs_{[i..(i+1)]} \in T \right) \tag{3.11}$$

The absence of occurrences ($\{\}$, the empty set of **atf**) is a "0" for the XBefore operator.

$$\{\} \rightarrow S = \{\} \qquad \text{left-false-absorb} \qquad (3.12a)$$
$$S \rightarrow \{\} = \{\} \qquad \text{right-false-absorb} \qquad (3.12b)$$
$$(S \rightarrow T) \cup S = S \qquad \text{left-union-absorb} \qquad (3.12c)$$
$$(T \rightarrow S) \cup S = S \qquad \text{right-union-absorb} \qquad (3.12d)$$
$$\mathbf{tempo}_1\, S \implies S \rightarrow S = \{\} \qquad \text{non-idempotent} \qquad (3.12e)$$
$$\mathbf{tempo}_1\, S \wedge \mathbf{tempo}_1\, T \wedge \mathbf{tempo}_1\, U \implies$$
$$S \rightarrow (T \rightarrow U) = (S \rightarrow T) \rightarrow U \qquad \text{associativity} \qquad (3.12f)$$

The XBefore is absorbed by one of the operands: if one of the operands may happen alone, thus the order with any other operand is irrelevant. However, an event cannot come before itself, thus XBefore is not idempotent. The XBefore but is associative.

To allow formula reduction we need the relation of XBefore to the other Boolean operators. First we use the XBefore as operands of union and intersection.

$$\begin{aligned}
&\mathbf{tempo}_1\, S \wedge \mathbf{tempo}_1\, T \implies \\
&\qquad (S \to T) \cap (T \to S) = \{\} && \text{inter-equiv-false} && (3.13a)\\
&\mathbf{tempo}_{1-4}\, S \wedge \mathbf{tempo}_{1-4}\, T \wedge S \diamond T \implies \\
&\qquad (S \to T) \cup (T \to S) = S \cap T && \text{union-equiv-inter} && (3.13b)
\end{aligned}$$

As the XBefore is not symmetric, the intersection of symmetrical sets is empty. The union of the symmetric is a partition of the intersection of the operands.

In our previous work (DIDIER; MOTA, 2015), we stated that S and T had to be variables. For example, of the form $\mathbf{var}\, s$ and $\mathbf{var}\, t$. Now, each law requires that the operands satisfy some of the temporal properties, avoiding using variables explicitly.

Boolean operators are used as operands of the XBefore in the following laws.

$$\begin{aligned}
&(S \cup T) \to U = (S \to U) \cup (T \to U) && \text{left-union-dist} && (3.14a)\\
&S \to (T \cup U) = (S \to T) \cup (S \to U) && \text{right-union-dist} && (3.14b)\\
&\mathbf{tempo}_{1-4}\, S \wedge \mathbf{tempo}_{1-4}\, T \wedge S \diamond T \implies \\
&\qquad (S \cap T) \to U = (S \to T \to U) \cup \\
&\qquad\qquad (T \to S \to U) && \text{left-inter-dist} && (3.14c)\\
&\mathbf{tempo}_{1-4}\, T \wedge \mathbf{tempo}_{1-4}\, U \wedge T \diamond U \implies \\
&\qquad S \to (T \cap U) = (S \to T \to U) \cup \\
&\qquad\qquad (S \to U \to T) && \text{right-inter-dist} && (3.14d)\\
&\mathbf{tempo}_2\, S \implies S \cap (T \to U) = ((S \cap T) \to U) \cup \\
&\qquad\qquad (T \to (S \cap U)) && \text{unordered} && (3.14e)
\end{aligned}$$

XBefore is distributive over union. On the other hand, the intersection is related to order. Thus it is not distributive with XBefore. Finally, the intersection of an event with an XBefore states that such an event can occur in any order within the events in the XBefore.

The law name, unordered, of (3.14e) is clearer if we expand (3.14e) with (3.14c) and (3.14d):

$$\begin{aligned}
&\mathbf{tempo}_{1-4}\, S \wedge \mathbf{tempo}_{1-4}\, T \wedge \\
&\mathbf{tempo}_{1-4}\, U \wedge S \diamond T \wedge S \diamond U \implies \\
&\qquad S \cap (T \to U) = (S \to T \to U) \cup \\
&\qquad\qquad (T \to S \to U) \cup \\
&\qquad\qquad (T \to U \to S) && \text{expanded-unordered} && (3.15)
\end{aligned}$$

4 Case study

EMBRAER provided us with the Simulink model of an Actuator Control System (depicted in Figure 22). The failure logic of this system (that is, for each of its constituent components) was also provided by EMBRAER (we show some of them in Table 8). In what follows we illustrate our strategy using the Monitor component.

A monitor component is a system commonly used for fault tolerance (O'CONNOR; NEWTON; BROMLEY, 2002; KOREN; KRISHNA, 2007). Initially, the monitor connects the main input (power source on input port 1) with its output. It observes the value of this input port and compares it to a threshold. If the value is below the threshold, the monitor disconnects the output from the main input and connects to the secondary input. We present the Simulink model for this monitor in Figure 23.

Now we show two contributions: (i) using only Boolean operators, thus ignoring ordering, we can obtain the same results obtained in (DIDIER; MOTA, 2012), and (ii) we represent each of the fault traces reported in (DIDIER; MOTA, 2012) as a term in our proposed algebra of temporal faults. Similarly to the association of fault events of Table 8 in Section 2.5, we associate the fault events as:

$$a = \text{LowPower-In1} \qquad A = \mathbf{var}\, a$$
$$b = \text{LowPower-In2} \qquad B = \mathbf{var}\, b$$
$$s = \text{SwitchFailure} \qquad S = \mathbf{var}\, s$$

4.1 Structure expressions with Boolean operators

In this section we show that the same result reported in (DIDIER; MOTA, 2012) in terms of static failure logic (or Boolean propositions) can be obtained with our Boolean operator without using XBefore. For each trace shown in Section 2.5, a mapping function[1]

[1] In this work we do not show the mapping function from traces to ATF (and the mapping function with XBefore in Section 4.2). The mapping rules follow the traces: XBefore is obtained by the order of occurrence and the absence of an event is the complement ($-$).

($\overset{\rightsquigarrow}{\text{B}}$) generates the following sets of lists:

`TRACE 1:`	$[s,b] \overset{\rightsquigarrow}{\text{B}} S \cap B \cap -A$	$\{[s,b],[b,s]\}$
`TRACE 2:`	$[b,s] \overset{\rightsquigarrow}{\text{B}} B \cap S \cap -A$	$\{[s,b],[b,s]\}$
`TRACE 3:`	$[a,b] \overset{\rightsquigarrow}{\text{B}} A \cap B \cap -S$	$\{[a,b],[b,a]\}$
`TRACE 4:`	$[b,a] \overset{\rightsquigarrow}{\text{B}} B \cap A \cap -S$	$\{[a,b],[b,a]\}$
`TRACE 5:`	$[a,s] \overset{\rightsquigarrow}{\text{B}} A \cap S \cap -B$	$\{[a,s],[s,a]\}$
`TRACE 6:`	$[a,s,b] \overset{\rightsquigarrow}{\text{B}} A \cap S \cap B$	$\{[a,b,s],[a,s,b],\ldots,[s,b,a]\}$
`TRACE 7:`	$[a,b,s] \overset{\rightsquigarrow}{\text{B}} A \cap B \cap S$	$\{[a,b,s],[a,s,b],\ldots,[s,b,a]\}$
`TRACE 8:`	$[b,a,s] \overset{\rightsquigarrow}{\text{B}} B \cap A \cap S$	$\{[a,b,s],[a,s,b],\ldots,[s,b,a]\}$

They represent the same faults shown in Section 2.5. Note that the negation in the formula is very simple to represent in ATF (and FBA) because it is just the absence of the generator.

Combining the above sets with unions (ORs), we obtain the following formula set:

$$\{[s,b],[b,s],[a,b],[b,a],[a,s],[s,a],[a,b,s],[a,s,b],\ldots,[s,b,a]\}$$

If we use Boolean expression reduction instead, it results in the following expression in ATF (and in FBA):

$$(A \cap B) \cup (S \cap (A \cup B))$$

which is equivalent to the set of sets above and is equivalent to EMBRAER failure logic expression shown in Table 8 (with AND s as $\cap$ and OR s as $\cup$). This shows that ATF can represent (static) failure logic as in our previous work (DIDIER; MOTA, 2012).

4.2 Structure expressions with XBefore

Now, by using TFA with the XBefore operator and a mapping function ($\overset{\rightsquigarrow}{\text{XB}}$), we can capture each possible individual sequences as generated by the work (DIDIER; MOTA,

2012):

$$\begin{aligned}
&\texttt{TRACE 1:}\ [s,b] \overset{\rightsquigarrow}{\text{XB}} (S \to B) \cap -A && \{[s,b]\}\\
&\texttt{TRACE 2:}\ [b,s] \overset{\rightsquigarrow}{\text{XB}} (B \to S) \cap -A && \{[b,s]\}\\
&\texttt{TRACE 3:}\ [a,b] \overset{\rightsquigarrow}{\text{XB}} (A \to B) \cap -S && \{[a,b]\}\\
&\texttt{TRACE 4:}\ [b,a] \overset{\rightsquigarrow}{\text{XB}} (B \to A) \cap -S && \{[b,a]\}\\
&\texttt{TRACE 5:}\ [a,s] \overset{\rightsquigarrow}{\text{XB}} (A \to S) \cap -B && \{[a,s]\}\\
&\texttt{TRACE 6:}\ [a,s,b] \overset{\rightsquigarrow}{\text{XB}} A \to S \to B && \{[a,s,b]\}\\
&\texttt{TRACE 7:}\ [a,b,s] \overset{\rightsquigarrow}{\text{XB}} A \to B \to S && \{[a,b,s]\}\\
&\texttt{TRACE 8:}\ [b,a,s] \overset{\rightsquigarrow}{\text{XB}} B \to A \to S && \{[b,a,s]\}
\end{aligned}$$

Using ATF and combining each trace with ORs (unions), we obtain the following set:

$$M_L = \{[a,b],[b,a],[b,s],[s,b],[a,s],[a,b,s],[a,s,b],[s,a,b]\}$$

From the above traces, we also build an ATF expression by mapping each trace to an XBefore expression, composing all resulting XBefore expressions with ORs and reducing them using the XBefore laws (Section 3.2), resulting in an expression (M_A) that is equivalent to the above set of lists ($M_L \equiv M_A$). The failure expression of the monitor[2] is:

$$\begin{aligned}
M_A = {} & ((S \to B) \cap -A) \cup ((B \to S) \cap -A) \cup \\
& ((A \to B) \cap -S) \cup ((B \to A) \cap -S) \cup \\
& ((A \to S) \cap -B) \cup \\
& (A \to S \to B) \cup (A \to B \to S) \cup (B \to A \to S) \\
= {} & (B \cap S \cap -A) \cup && \text{by (3.13b)} \\
& (B \cap A \cap -S) \cup && \text{by (3.13b)} \\
& ((A \to S) \cap -B) \cup \\
& (A \to S \to B) \cup (A \to B \to S) \cup (B \to A \to S) \\
= {} & (B \cap S \cap -A) \cup \\
& (B \cap A \cap -S) \cup \\
& ((A \to S) \cap -B) \cup \\
& ((A \to S) \cap B) && \text{by (3.15)} \\
= {} & (B \cap S \cap -A) \cup (B \cap A \cap -S) \cup (A \to S) && \text{by absorption}
\end{aligned}$$

[2] In the final formula, $(B \cap S \cap -A) \cup (A \cap B \cap -S)$ is equivalent to $(B \cap (S \oplus A))$. There is a typo in our previous work (DIDIER; MOTA, 2015). The expression was written with an OR ($\vee$) but it should an XOR ($\oplus$).

The semantics of the above expression is: (i) fault b (**var** b) occurs and fault a (**var** a) or fault s (**var** s) occurs (but not both a and s), or (ii) fault a occurs before fault s, which is more precise than the expression found without considering order of events.

5 Conclusion

In this work we presented a foundational theory to support a more precise representation of fault events as compared to our previous strategy for injecting faults (DIDIER; MOTA, 2012). The failure logic is essential for system safety assessment because it is used as basic input for building fault trees (PAPADOPOULOS et al., 2001; JESUS et al., 2011; GOMES et al., 2010). Furthermore, we still connect the strategy presented in (MOTA et al., 2010) with the works reported in (JESUS et al., 2011) (functional analysis) and in (GOMES et al., 2010; PAPADOPOULOS et al., 2001) (safety assessment) because our new algebra is at least a Boolean algebra.

The work reported in (WALKER, 2009; WALKER; PAPADOPOULOS, 2009; WALKER; PAPADOPOULOS, 2010) tackles simultaneity with "nearly simultaneous" events (EDIFOR; WALKER; GORDON, 2013). But we consider instantaneous events, like the work reported in (MERLE; ROUSSEL; LESAGE, 2014), because we assume that simultaneity is probabilistically impossible.

The distinct lists representation in our algebra does not allow obtaining minimal cut sequences directly from the formula, similar to FBAs. The sets in an FBA formula are already the minimal cut sets. In our work, ATF allows us to find minimal cut sequences (with XBefore) from the formulas in DNF algebraically: each sub-expression is a minimal cut sequence.

Boolean formulas reduction can be achieved by: (i) application of Boolean laws, (ii) BDD, or (iii) FBAs. We used Boolean and XBefore laws to reduce ATF formulas. The work reported in (TANNOUS; XING; DUGAN, 2011; XING; TANNOUS; DUGAN, 2012) uses Sequential BDDs to reduce formulas with order-based operators. We plan to use similar concepts in a future work.

Although we do not use negation (NOT operator) with XBefore in our case study it is part of ATF, so it could be used. As future work we will demonstrate the relations of NOT and XBefore, as we did for AND and OR. We will also define laws to avoid the conditions that cause non-coherent analysis (OLIVA, 2006). The issue with negated events comes up when both an event and its negation appear on the same tree. One very restrictive solution to this issue is applying the *generators independence* laws (3.7d, 3.7f) on basic events of a tree, by actually considering the negation of an event a different event (for instance, $\mathbf{var}\, a = \mathbf{var}\, e$ and $\mathbf{var}\, b = -\mathbf{var}\, e$). We look forward to obtain a less restrictive law.

We showed the DNF for ATF, but did not demonstrate that every formula can be converted into DNF. The laws shown in this work should be sufficient to this demonstration.

Also, we did not show the mapping rules from traces to ATF (with Boolean operators only and with XBefore). The mapping rules follow the traces: XBefore is obtained by the order of occurrence and the absence of an event is the complement ($-$). We leave these demonstrations as future work.

Bibliography

ADACHI, M. et al. An approach to optimization of fault tolerant architectures using HiP-HOPS. *Software: Practice and Experience*, John Wiley & Sons, Ltd., v. 41, n. 11, p. 1303–1327, 2011. ISSN 1097-024X.

AKERS. Binary Decision Diagrams. *IEEE Transactions on Computers*, Institute of Electrical & Electronics Engineers (IEEE), C-27, n. 6, p. 509–516, jun 1978.

ANAC. *Aeronautical Product Certification (in portuguese)*. 2011. DOU Nº 230, Seção 1, p. 28, 01/12/2011. Disponível em: <http://www2.anac.gov.br/biblioteca/resolucao/2011/RBAC21EMD01.pdf>.

ANDERSON, W. J. *Continuous-Time Markov Chains*. Springer New York, 2012. Disponível em: <http://www.ebook.de/de/product/25435927/william_j_anderson_continuous_time_markov_chains.html>.

ANDREWS, J.; BEESON, S. Birnbaum's measure of component importance for noncoherent systems. *IEEE Transactions on Reliability*, Institute of Electrical & Electronics Engineers (IEEE), v. 52, n. 2, p. 213–219, jun 2003. Disponível em: <http://dx.doi.org/10.1109/TR.2003.809656>.

ANDREWS, J. D. The use of not logic in fault tree analysis. *Quality and Reliability Engineering International*, John Wiley & Sons, Ltd., v. 17, n. 3, p. 143–150, 2001. ISSN 1099-1638. Disponível em: <http://dx.doi.org/10.1002/qre.405>.

ANDREWS, Z. et al. *Report on Timed Fault Tree Analysis — Fault modelling*. [S.l.], 2013. Disponível em: <http://www.compass-research.eu/Project/Deliverables/D242.pdf>.

ANDREWS, Z. et al. Model-based development of fault tolerant systems of systems. In: *Systems Conference (SysCon), 2013 IEEE International*. [S.l.: s.n.], 2013. p. 356–363.

ASTROM, K. J.; MURRAY, R. M. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ, USA: Princeton University Press, 2008. ISBN 0691135762, 9780691135762.

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, v. 1, n. 1, p. 11–33, 2004. ISSN 1545-5971.

AVRESKY, D. et al. Fault injection for formal testing of fault tolerance. *IEEE Transactions on Reliability*, Institute of Electrical & Electronics Engineers (IEEE), v. 45, n. 3, p. 443–455, 1996. Disponível em: <http://dx.doi.org/10.1109/24.537015>.

BOBBIO, A.; RAITERI, D. Parametric fault trees with dynamic gates and repair boxes. In: *Reliability and Maintainability, 2004 Annual Symposium - RAMS*. [S.l.: s.n.], 2004. p. 459–465.

BOBBIO, A. et al. *DBNet, a tool to convert Dynamic Fault Trees to Dynamic Bayesian Networks*. [S.l.], 2005.

BOUTE, R. The binary decision machine as programmable controller. *Euromicro Newsletter*, Elsevier BV, v. 2, n. 1, p. 16–22, jan 1976.

BOYD, M. A. *Dynamic Fault Tree Models: Techniques for Analysis of Advanced Fault Tolerant Computer Systems*. Tese (Doutorado) — Duke University, Durham, NC, USA, 1992. UMI Order No. GAX92-02503.

BRACE, K. S.; RUDELL, R. L.; BRYANT, R. E. Efficient implementation of a BDD package. In: *Conference proceedings on 27th ACM/IEEE design automation conference - DAC '90*. Association for Computing Machinery (ACM), 1990. Disponível em: <http://dx.doi.org/10.1145/123186.123222>.

BRYANS, J.; CANHAM, S.; WOODCOCK, J. *CML Definition 4*. [S.l.], 2014. Disponível em: <http://www.compass-research.eu/Project/Deliverables/D23.5-final-version.pdf>.

CARVALHO, G. et al. NAT2TEST Tool: From Natural Language Requirements to Test Cases Based on CSP. In: *Software Engineering and Formal Methods*. Springer Science + Business Media, 2015. p. 283–290. Disponível em: <http://dx.doi.org/10.1007/978-3-319-22969-0_20>.

CHIOLA, G. et al. Stochastic well-formed colored nets and symmetric modeling applications. *IEEE Transactions on Computers*, Institute of Electrical & Electronics Engineers (IEEE), v. 42, n. 11, p. 1343–1360, 1993. Disponível em: <http://dx.doi.org/10.1109/12.247838>.

CONTINI, S.; COJAZZI, G.; RENDA, G. On the use of non-coherent fault trees in safety and security studies. *Reliability Engineering & System Safety*, v. 93, n. 12, p. 1886 – 1895, 2008. ISSN 0951-8320. 17th European Safety and Reliability Conference. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0951832008001117>.

COPPIT, D.; SULLIVAN, K. J.; DUGAN, J. B. Formal semantics of models for computational engineering: a case study on dynamic fault trees. In: *Software Reliability Engineering, 2000. ISSRE 2000. Proceedings. 11th International Symposium on*. [S.l.: s.n.], 2000. p. 270 –282. ISSN 1071-9458.

DIDIER, A. *Estratégia sistemática para identificar falhas em componentes de hardware usando comportamento nominal*. Dissertação (Mestrado) — Universidade Federal de Pernambuco, 2 2012.

DIDIER, A.; MOTA, A. Identifying Hardware Failures Systematically. In: GHEYI, R.; NAUMANN, D. (Ed.). *Formal Methods: Foundations and Applications*. [S.l.]: Springer Berlin / Heidelberg, 2012, (Lecture Notes in Computer Science, v. 7498). p. 115–130. ISBN 978-3-642-33295-1.

DIDIER, A.; MOTA, A. An Algebra of Temporal Faults. *Information Systems Frontiers*, jan 2016. ISSN 1572-9419. Submitted to Information Systems Frontiers in jan/2016 as a special issue.

DIDIER, A. L. R.; MOTA, A. A Lattice-Based Representation of Temporal Failures. In: *Information Reuse and Integration (IRI), 2015 IEEE International Conference on*. [S.l.: s.n.], 2015. p. 295–302.

DISTEFANO, S.; PULIAFITO, A. Dependability Evaluation with Dynamic Reliability Block Diagrams and Dynamic Fault Trees. *IEEE Transactions on Dependable and Secure Computing*, Institute of Electrical & Electronics Engineers (IEEE), v. 6, n. 1, p. 4–17, jan 2009. Disponível em: <http://dx.doi.org/10.1109/TDSC.2007.70242>.

DUGAN, J. B.; BAVUSO, S. J.; BOYD, M. A. Dynamic fault-tree models for fault-tolerant computer systems. *Reliability, IEEE Transactions on*, v. 41, n. 3, p. 363 –377, sep 1992. ISSN 0018-9529.

EDIFOR, E.; WALKER, M.; GORDON, N. Quantification of Simultaneous-AND Gates in Temporal Fault Trees. In: ZAMOJSKI, W. et al. (Ed.). *New Results in Dependability and Computer Systems*. [S.l.]: Springer International Publishing, 2013, (Advances in Intelligent Systems and Computing, v. 224). p. 141–151. ISBN 978-3-319-00944-5.

FAA. Book, Online. *RTCA, Inc., Document RTCA/DO-178B*. [S.l.]: U.S. Dept. of Transportation, Federal Aviation Administration, [Washington, D.C.] :, 1993. [1] p. : p.

FAA. *Part 25 - Airworthiness Standards: Transport Category Airplanes*. [S.l.], 2007.

GIVANT, S.; HALMOS, P. *Introduction to Boolean Algebras*. [s.n.], 2009. XIV. (Undergraduate Texts in Mathematics, XIV). ISBN 978-0-387-68436-9. Disponível em: <http://www.springer.com/mathematics/book/978-0-387-40293-2>.

GOMES, A. et al. Systematic Model-Based Safety Assessment Via Probabilistic Model Checking. In: MARGARIA, T.; STEFFEN, B. (Ed.). *ISoLA (1)*. [S.l.]: Springer, 2010. (Lecture Notes in Computer Science, v. 6415), p. 625–639. ISBN 978-3-642-16557-3.

GULATI, R.; DUGAN, J. A modular approach for analyzing static and dynamic fault trees. In: *Reliability and Maintainability Symposium. 1997 Proceedings, Annual*. [S.l.: s.n.], 1997. p. 57 –63.

HOARE, C. A. R.; HE, J. *Unifying Theories of Programming*. Prentice Hall Englewood Cliffs, 1998. v. 14. Disponível em: <http://www.unifyingtheories.org/>.

HUFFMAN, B. Free Boolean Algebra. *Archive of Formal Proofs*, v. 2010, mar. 2010. ISSN 2150-914x. Disponível em: <http://afp.sourceforge.net/entries/Free-Boolean-Algebra.shtml>.

IANNELLI, M.; PUGLIESE, A. An Introduction to Mathematical Population Dynamics: Along the trail of Volterra and Lotka. In: _____. Cham: Springer International Publishing, 2014. cap. Continuous-time Markov chains, p. 329–334. ISBN 978-3-319-03026-5. Disponível em: <http://dx.doi.org/10.1007/978-3-319-03026-5_13>.

JASKELIOFF, M.; MERZ, S. Proving the Correctness of Disk Paxos. *Archive of Formal Proofs*, jun. 2005. ISSN 2150-914x. <http://afp.sf.net/entries/DiskPaxos.shtml>, Formal proof development.

JENSEN, K. High-Level Petri Nets. In: *Applications and Theory of Petri Nets*. Springer Science + Business Media, 1983. p. 166–180. Disponível em: <http://dx.doi.org/10.1007/978-3-642-69028-0_12>.

JENSEN, K. Coloured Petri Nets. In: *Petri Nets: Central Models and Their Properties*. Springer Science + Business Media, 1987. p. 248–299. Disponível em: <http://dx.doi.org/10.1007/978-3-540-47919-2_10>.

JESUS, J. et al. Architectural Verification of Control Systems Using CSP. In: QIN, S.; QIU, Z. (Ed.). *ICFEM*. [S.l.]: Springer, 2011. (Lecture Notes in Computer Science, v. 6991), p. 323–339. ISBN 978-3-642-24558-9.

KISSMANN, P.; HOFFMANN, J. BDD Ordering Heuristics for Classical Planning. *Journal of Artificial Intelligence Research*, v. 51, 2014. Disponível em: <http://doi.org/10.1613/jair.4586>.

KOREN, I.; KRISHNA, C. M. *Fault Tolerant Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007. ISBN 0120885255.

MAHMUD, N.; PAPADOPOULOS, Y.; WALKER, M. A translation of State Machines to temporal fault trees. *2010 International Conference on Dependable Systems and Networks Workshops (DSN-W)*, IEEE, Jun 2010. Disponível em: <http://dx.doi.org/10.1109/DSNW.2010.5542620>.

MAHMUD, N.; WALKER, M.; PAPADOPOULOS, Y. Compositional Synthesis of Temporal Fault Trees from State Machines. *SIGMETRICS Perform. Eval. Rev.*, ACM, New York, NY, USA, v. 39, n. 4, p. 79–88, abr. 2012. ISSN 0163-5999.

MAIER, M. W. Architecting principles for systems-of-systems. *Systems Engineering*, John Wiley & Sons, Inc., v. 1, n. 4, p. 267–284, 1998. ISSN 1520-6858.

MATHWORKS. *Matlab®*. 2010. Disponível em: <http://www.mathworks.com/products/matlab>.

MATHWORKS. *Simulink®*. 2010. Disponível em: <http://www.mathworks.com/products/simulink>.

MERLE, G. *Algebraic modelling of Dynamic Fault Trees, contribution to qualitative and quantitative analysis*. Tese (Theses) — École normale supérieure de Cachan - ENS Cachan, jul. 2010. Disponível em: <https://tel.archives-ouvertes.fr/tel-00502012>.

MERLE, G.; ROUSSEL, J.-M.; LESAGE, J.-J. Algebraic determination of the structure function of Dynamic Fault Trees. *Reliability Engineering & System Safety*, Elsevier BV, v. 96, n. 2, p. 267–277, Feb 2011. ISSN 0951-8320.

MERLE, G.; ROUSSEL, J.-M.; LESAGE, J.-J. Dynamic fault tree analysis based on the structure function. *2011 Proceedings - Annual Reliability and Maintainability Symposium*, IEEE, Jan 2011. Disponível em: <http://dx.doi.org/10.1109/RAMS.2011.5754452>.

MERLE, G.; ROUSSEL, J.-M.; LESAGE, J.-J. Quantitative Analysis of Dynamic Fault Trees Based on the Structure Function. *Quality and Reliability Engineering International*, Wiley-Blackwell, v. 30, n. 1, p. 143–156, Feb 2014. ISSN 0748-8017.

MERLE, G. et al. Probabilistic Algebraic Analysis of Fault Trees With Priority Dynamic Gates and Repeated Events. *IEEE Trans. Rel.*, Institute of Electrical & Electronics Engineers (IEEE), v. 59, n. 1, p. 250–261, Mar 2010. ISSN 1558-1721.

MIKULAK, R.; MCDERMOTT, R.; BEAUREGARD, M. *The Basics of FMEA, 2nd Edition*. CRC Press, 2008. ISBN 9781439809617. Disponível em: <https://books.google.com.br/books?id=rM5Vi_0K9bUC>.

MODARRES, M.; KAMINSKIY, M. P.; KRIVTSOV, V. *Reliability engineering and risk analysis: a practical guide*. [S.l.]: CRC press, 2009. ISBN 1420047051, 9781420047059.

MOSZKOWSKI, B. *A Temporal Logic for Multi-Level Reasoning About Hardware,*. [S.l.], 1982. Disponível em: <http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA324174>.

MOTA, A. et al. Evolving a Safe System Design Iteratively. In: SCHOITSCH, E. (Ed.). *SAFECOMP*. [S.l.]: Springer, 2010. (Lecture Notes in Computer Science, v. 6351), p. 361–374. ISBN 978-3-642-15650-2.

NIPKOW, T.; PAULSON, L. C.; WENZEL, M. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Springer, 2002. v. 2283. (LNCS, v. 2283). Disponível em: <https://isabelle.in.tum.de/>.

NISE, N. S. *Control systems engineering*. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1992. ISBN 0-8053-5420-4.

Object Management Group (OMG). *Systems Modelling Language (SysML) 1.3*. 2012. Website. Disponível em: <http://www.omg.org/spec/SysML/1.3>.

O'CONNOR, P.; NEWTON, D.; BROMLEY, R. *Practical reliability engineering*. [S.l.]: Wiley, 2002. ISBN 9780470844632.

OLIVA, S. Non-Coherent Fault Trees Can Be Misleading. *e-Journal of System Safety*, v. 42, n. 3, May-June 2006. Accessed in 13/jan/2016. Disponível em: <http://www.system-safety.org/ejss/past/mayjune2006ejss/spotlight2_p1.php>.

PALSHIKAR, G. K. Temporal fault trees. *Information and Software Technology*, v. 44, n. 3, p. 137 – 150, 2002. ISSN 0950-5849.

PAPADOPOULOS, Y. et al. Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Reliability Engineering & System Safety*, v. 71, n. 3, p. 229–247, 2001. ISSN 0951-8320.

PEARL, J. *Bayesian Networks: a model of self-activated memory for evidential reasoning*. [S.l.], 1985. Disponível em: <ftp://ftp.cs.ucla.edu/pub/stat_ser/r43-1985.pdf>.

ROSCOE, A. W. *The Theory and Practice of Concurrency*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1997. Paperback. ISBN 0136744095.

RUDELL, R. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In: *Proceedings of the 1993 IEEE/ACM International Conference on Computer-aided Design*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1993. (ICCAD '93), p. 42–47. ISBN 0-8186-4490-7. Disponível em: <http://dl.acm.org/citation.cfm?id=259794.259802>.

SAE. Miscellaneous, *SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*. [S.l.]: Society of Automotive Engineers (SAE), 1996.

SCHELLHORN, G.; THUMS, A.; REIF, W. *Formal Fault Tree Semantics*. 2002.

SIMEU-ABAZI, Z.; LEFEBVRE, A.; DERAIN, J.-P. A methodology of alarm filtering using dynamic fault tree. *Reliability Engineering & System Safety*, Elsevier BV, v. 96, n. 2, p. 257–266, Feb 2011. ISSN 0951-8320.

SNOOKE, N.; PRICE, C. Model-driven automated software FMEA. In: *Reliability and Maintainability Symposium*. [S.l.: s.n.], 2011. p. 1–6. ISSN 0149-144X.

SOMMERVILLE, I. *Software Engineering*. Pearson, 2011. (International Computer Science Series). ISBN 9780137053469. Disponível em: <http://books.google.com.br/books?id=l0egcQAACAAJ>.

SPIVEY, J. M. *The Z Notation: A Reference Manual*. Second edition. Prentice Hall International (UK) Ltd, 1998. Disponível em: <http://spivey.oriel.ox.ac.uk/~mike/zrm/>.

STAMATELATOS, M. et al. *Fault Tree Handbook with Aerospace Applications*. Washington, DC 20546, 2002. Disponível em: <http://www.hq.nasa.gov/office/codeq/doctree/fthb.pdf>.

STOLL, R. R. *Set Theory and Logic*. Dover Publications, 1979. (Dover books on advanced mathematics). ISBN 9780486638294. Disponível em: <https://books.google.com.br/books?id=3-nrPB7BQKMC>.

TANG, Z.; DUGAN, J. Minimal cut set/sequence generation for dynamic fault trees. In: *Reliability and Maintainability, 2004 Annual Symposium - RAMS*. [S.l.: s.n.], 2004. p. 207–213.

TANNOUS, O.; XING, L.; DUGAN, J. B. Reliability analysis of warm standby systems using sequential BDD. *2011 Proceedings - Annual Reliability and Maintainability Symposium*, IEEE, Jan 2011.

VAURIO, J. K. Importances of components and events in non-coherent systems and risk models. *Reliability Engineering & System Safety*, v. 147, p. 117 – 122, 2016. ISSN 0951-8320. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0951832015003348>.

VESELY, W. et al. *Fault Tree Handbook*. US Independent Agencies and Commissions, 1981. ISBN 9780160055829. Disponível em: <http://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0492/>.

WALKER, M.; PAPADOPOULOS, Y. Synthesis and analysis of temporal fault trees with PANDORA: The time of Priority AND gates. *Nonlinear Analysis: Hybrid Systems*, v. 2, n. 2, p. 368 – 382, 2008. ISSN 1751-570X. Proceedings of the International Conference on Hybrid Systems and Applications, Lafayette, LA, USA, May 2006: Part II.

WALKER, M.; PAPADOPOULOS, Y. Qualitative temporal analysis: Towards a full implementation of the Fault Tree Handbook. *Control Engineering Practice*, v. 17, n. 10, p. 1115 – 1125, 2009. ISSN 0967-0661.

WALKER, M.; PAPADOPOULOS, Y. A hierarchical method for the reduction of temporal expressions in Pandora. In: *Proceedings of the First Workshop on DYnamic Aspects in DEpendability Models for Fault-Tolerant Systems*. New York, NY, USA: ACM, 2010. (DYADEM-FTS '10), p. 7–12. ISBN 978-1-60558-916-9.

WALKER, M. D. *Pandora: a logic for the qualitative analysis of temporal fault trees*. Tese (Doutorado) — University of Hull, May 2009. Disponível em: <https://hydra.hull.ac.uk/resources/hull:2526>.

XING, L.; TANNOUS, O.; DUGAN, J. B. Reliability Analysis of Nonrepairable Cold-Standby Systems Using Sequential Binary Decision Diagrams. *IEEE Trans. Syst., Man, Cybern. A*, Institute of Electrical & Electronics Engineers (IEEE), v. 42, n. 3, p. 715–726, May 2012. ISSN 1558-2426.

Appendix

APPENDIX A – Formal proofs in Isabelle/HOL