ISBN 978-82-326-4602-9 (printed ver.)
ISBN 978-82-326-4603-6 (electronic ver.)
ISSN 1503-8181

Liu Yang

Doctoral thesis

Doctoral theses at NTNU, 2020:127

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of
Philosophiae Doctor
Faculty of Engineering
Department of Mechanical and Industrial
Engineering

NTNU

Doctoral theses at NTNU, 2020:127

Liu Yang

Finite Degradation Structures

A Unified Framework of Combinatorial Models in Probabilistic Risk/Safety Assessment

Liu Yang

Finite Degradation Structures

A Unified Framework of Combinatorial Models in Probabilistic Risk/Safety Assessment

Thesis for the Degree of Philosophiae Doctor

Trondheim, June 2020

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Norwegian University of Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Engineering
Department of Mechanical and Industrial Engineering

© Liu Yang

ISBN 978-82-326-4602-9 (printed ver.)
ISBN 978-82-326-4603-6 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2020:127

Printed by NTNU Grafisk senter

Preface

On 26th November 2016, the very beginning of my PhD life, my supervisor Professor Antoine Rauzy sent me an email. In this email, he wrote down some tips of developing software. Although we have exchanged hundreds of emails in these three years, I still remember that one. Now, I would like to quote some sentences from it and share my understandings after these three years.

1. *"You may need to develop your own software, which is probably a few thousands of lines of Python code with a few dozens of classes. It is not very big, but is sufficiently big to require the application of software engineering principles."*

2. *"Architecting a software is extremely difficult. Don't expect to get the right architecture at the first attempt. Don't hesitate to spend a lot of time in refactoring your code. Refactoring is the key."*

3. *"Don't waste your time in commenting your code."*

4. *"Small functions are easier to test than large ones. Specialized functions are easier to test than (too) generic ones. The KISS principle: Keep It Simple Stupid."*

5. *"Make it work, then make it fast if and only if it is necessary."*

These sentences are talking about software development, but they are also my guidance during the whole PhD period. I rewrite them as follows:

1. Make your work complete even it is small; 2. refactoring is the key of progressing your work, never stop or hesitate; 3. always focus on the main objectives, don't waste time in useless details; 4. simplicity brings efficiency; 5. make it work first and then optimize.

The PhD period is only a small part of my life. It is not very big, but is sufficiently big to require the quality of being a good researcher. I hope that the modeling framework proposed this thesis, which is not very big, but is sufficiently big to become a unified framework of combinatorial models in probabilistic risk/safety assessment.

Liu Yang

Trondheim, 24th November 2019

Acknowledgement

First and foremost, I would like to thank my main supervisor, Professor Antoine Rauzy, for his tireless guidance and professional supervision during the last three years. I would say that without his guidance, I could never be able to accomplish such a PhD thesis. I would also thank him for opening many new worlds for me, from abstract maths to practical use cases, thank him for leading me crossing the border of my knowledge and ability, and thank him for teaching me how to become a real researcher.

I would like to thank Professor Cecilia Haskins and Professor Mary Ann Lundteigen for sharing their thoughts on topics of this work. Also thanks to my colleagues, Xue, Lei, Juntao, Yun, Shenae, Renny, Himanshu, Xiaopeng, Xingheng and many others, for the joys and laughs we had together.

I would like to thank for administrative staff of department of Mechanical and Industrial Engineering for the friendly work environment.

Last but not least, I would like to thank my family and friends, who support me from the beginning to the end. Especially, thanks to my father Professor Weiming Yang, my mother Professor Ning Chen, and my grandfather Professor Zitian Yang, for being my spiritual leaders who encouraged me to become a researcher.

Abstract

This PhD thesis presents a new modeling framework, called finite degradation structures (FDSs), which can be used as a unified framework of combinatorial models in probabilistic risk/safety assessment.

The so-called combinatorial models refer to those models where the behavior of the system is described as the combination of behaviors of its components. The Boolean combinatorial models have been well mastered by practitioners, such as fault trees, reliability block diagrams and their alternatives. However, when the state of component/system becomes multi-valued, such Boolean models become less applicable. Although more powerful modeling formalisms exist, e.g. Markov chains, Petri nets and guarded transition systems, their computational complexity increases dramatically when leaving the combinatorial realm.

A good compromise is to stay in the combinatorial realm while allow the state of component/system to be multi-valued. This provides the original motivation of this PhD thesis.

Technically, the modeling framework proposed this thesis extends formally all the concepts defined in fault tree analysis from Boolean systems into multistate systems. The most highlighted part of this work is the use of partially ordered set as state space of multistate component/system. Thanks to this partial order, we are able to define minimal cut/path sets for multistate systems. In our framework, the notion of minimal cutsets is covered by the notion of minimal (degraded) scenarios, which characterizes the minimal paths that the system degrades from an operation state into an undesired state, and the notion of minimal path sets is covered by the notion of maximal (degraded) scenarios, which characterizes the maximal ability that the system remains in a good state. The probabilistic indicators are also included in the proposed modeling framework, e.g. state probabilities, conditional probabilities, sensitivity factors, etc.

The models built on the framework of FDSs are called finite degradation models (FDMs). FDMs generalize both the syntax and the semantics of fault trees to multistate cases. The decision diagrams are used to implement the required calculation of scenarios and probabilistic indicators of FDMs. We adjusted the data structure and the algorithms that have been applied on binary decision diagrams to fit FDMs. Moreover, we also developed a software to realize the computerized modeling and assessment of FDMs. As experimental results, we show the full analysis of a safety instrumented system made of sensors, logic solvers and valves and a simple train control system made of hot-standby subsystems.

Contents

List of Tables **xvi**

List of Figures **xx**

Abbreviations **xxi**

List of Symbols **xxiii**

1 Introduction **1**

1.1 Background 1

1.2 Objectives 2

1.3 Research approaches and work process 3

1.4 Structure of thesis 4

1.5 Academic publications 6

I State of the Art **9**

2 Review of Combinatorial Models in System Reliability Theory **11**

2.1 Taxonomy of reliability models 11

2.2 Combinatorial models 13
2.2.1 Overview 13
2.2.2 Fault tree methodology and limitations 13
2.2.3 Existing solutions of extending fault tree analysis to multistate systems 15
2.2.4 Need for new modeling framework 18
2.3 Illustrative case study: a safety instrumented system (SIS) 20
2.3.1 System description 20
2.3.2 Failure modes 21
2.3.3 Assumptions and parameters 21
2.3.4 Failure mechanism and modeling questions 22

II Theoretical Development **25**

3 Algebraic Framework of Finite Degradation Structures **27**
3.1 Partially ordered sets 28
3.2 Finite degradation structures (FDSs) 31
3.2.1 Definitions 31
3.2.2 Typical examples 32
3.2.3 The FDS for SIS 34
3.3 Operations on FDSs 35
3.3.1 Monoidal product 36
3.3.2 Abstractions 38
3.3.3 Operations 40
3.3.4 The operations for SIS 43

4 Finite degradation models: a unified formalism of combinatorial reliability models **47**

4.1 Syntax: the structure of a model 47
4.1.1 Well-formed formulas 47
4.1.2 Finite degradation models (FDMs) 48
4.1.3 Graphical representation: expression tree 49
4.2 Semantics: the meaning of a model 51
4.2.1 Interpretation of formulas 51
4.2.2 Interpretation of FDMs 51
4.3 The model of SIS 54
4.3.1 Fault tree like model 54
4.3.2 Reliability block diagram like model 56

5 Assessment of Finite Degradation Models 59

5.1 Scenarios analysis 60
5.1.1 Scenarios 60
5.1.2 Conditional scenarios 61
5.1.3 Critical scenarios 63
5.2 Probabilistic calculations 68
5.2.1 State probabilities 68
5.2.2 Conditional probability 69
5.2.3 Sensitivity analysis 71
5.2.4 Approximation of probability by critical scenarios 73
5.3 Decision diagram based assessment 73
5.3.1 Decision diagrams 73
5.3.2 Construction of decision diagrams 75
5.3.3 Calculation of scenarios 79
5.3.4 Calculation of probabilistic indicators 82

III Application **85**

6 Implementation and Experiments **87**

6.1 LatticeX: object-oriented implementation of FDMs 87

6.1.1 Software architecture 87

6.1.2 Modeling language FDS-ML 88

6.1.3 Construction of expression trees 92

6.2 Modeling library 93

6.2.1 Boolean logic operators 93

6.2.2 Meet and join 93

6.2.3 k-out-of-n 95

6.2.4 Dependent components 95

6.3 Experiments 99

6.3.1 Safety instrumented system 99

6.3.2 A simplified train control system 107

7 Modeling Epistemic Space of Degradation Processes **111**

7.1 Problem statement 111

7.1.1 Incomplete knowledge on states 111

7.1.2 Epistemic space 113

7.2 FDSs in epistemic space 114

7.2.1 Degradation orders among epistemic states 114

7.2.2 Basic belief assignment of epistemic states 118

7.2.3 Transformation of FDSs 119

7.3 FDMs in epistemic space 121

7.3.1 Transformation of operations 122

7.3.2 Reinterpretation of FDMs: a case study 126

7.3.3 Uncertainty analysis 129

8 Summary of main results and future work 135

8.1 Summary of main results 135

8.1.1 Modeling framework of finite degradation structures . . . 135

8.1.2 Calculation algorithms 138

8.1.3 Software and modeling language 140

8.2 Recommendations for future work 140

8.3 Closing remarks 143

References 153

A Appendices 155

A.1 Attributes of the classes in LatticeX 155

A.2 Grammar of FDS-ML defined by EBNF 156

A.3 Textual model of the safety instrumented system 159

A.4 Sensitivity results of the safety instrumented system 159

B Article on Model Synthesis 167

List of Tables

2.1 Comparison of existing multistate combinatorial models. 19

2.2 TA4 Reliability parameters 22

3.1 Typical FDSs applied in reliability and safety analyses. 32

3.2 The valuation of $\phi : \mathbf{WF}^2 \twoheadrightarrow \mathbf{WDF}$. 42

3.3 Valuation of $\ogreaterthan$ and $\|$. 43

4.1 Interpretation of the expression tree in Figure 4.1. 50

6.1 Number of scenarios and critical scenarios for safety channel 1. . . 100

6.2 Number of scenarios and critical scenarios for safety channel 2. . . 100

6.3 Average probabilities for p_{SC1} and p_{SC2} within 8760 hours. . . . 102

6.4 Number of scenarios and critical scenarios for $System$. 102

6.5 Number of scenarios and critical scenarios for $System$ satisfying Eq.(6.4). 103

6.6 Average probabilities of Pr$\{System = y|\mathbf{C_S}\}$ and Pr$\{System = y|\mathbf{C_S}'\}$ within 8760 hours. 104

6.7 Definitions of acronyms 107

6.8 Failure and repair rates of each hot-standby subsystem. 108

6.9 Valuation of $\vee_{TCC} : \mathbf{WF}^2 \twoheadrightarrow \mathbf{WDF}$ 109
6.10 Number of scenarios and critical scenarios for TCC. 109

7.1 Number of uncertain states and the required new valuations for $\diamondsuit$. 113
7.2 Allocation example of $p(s)$ in $m(X)$ where $\Theta = \{W, D, F\}$. . . . 121
7.3 Comparison of FDSs in state space and in epistemic space. 122
7.4 Definitions of acronyms 127
7.5 The valuation of $\sqcup_{XOR} : \mathbf{WF}^2 \twoheadrightarrow \mathbf{WF}$. 128
7.6 Numbers of scenarios and critical scenariosfor $\mathcal{M}_1$ and $\mathcal{M}_2$. . . . 129
7.7 Probability measure in the domain $\mathbf{WF}$ of state variables without uncertainty. 131
7.8 Mass assignment in the domain $(\mathbf{WF})^u$ of state variables with uncertainty. 131
7.9 Results of $\mathbf{Bel}$, $\mathbf{Pl}$, $\mathbf{Best}$ and $\mathbf{Worst}$ in the domain of $Attack\_OR$ and $Attack\_XOR$ for $\mathcal{M}_1$ and $\mathcal{M}_2$. 133

A.1 The constructs in EBNF and their meanings. 158

List of Figures

1.1 Outline of the structure of thesis. 5

2.1 Reliability models, assessment techniques and indicators. 12

2.2 Safety instrumented system in TA4 of ISO/TR 12489 20

3.1 Hasse diagram example. 28

3.2 Poset without greatest and least element. 29

3.3 Example of (a) join-semi-lattices and (b) meet-semi-lattice. 30

3.4 Classification of posets. 30

3.5 Hasse diagram of $\mathbf{WF}$, $\mathbf{WDF}$, $\mathbf{SWF}$, $\mathbf{WFdFu}$ and $\mathbf{WFdFs}$. . 32

3.6 The structure of $\mathbf{n}$ and $\mathbf{WnF}$. 33

3.7 The FDS $\mathbf{SIS}$. 34

3.8 The probability measures for sensors, logic solvers and valves. . . 36

3.9 $\mathbf{WF}^2$, $\mathbf{WF} \otimes \mathbf{WDF}$, $\mathbf{WDF} \otimes \mathbf{WF}$ and $\mathbf{WDF}^2$. 37

3.10 Examples of abstraction. 38

3.11 Examples of disordered abstraction. 39

3.12 A coherent but not strictly order-preserving abstraction. 39

3.13 Strongly-coherent abstractions φ_1, φ_2 and weakly-coherent abstraction φ_3. 40

3.14 Illustration of $\phi : \mathbf{WF}^2 \twoheadrightarrow \mathbf{WDF}$. 42

3.15 The block-diagram-like representations of $\ominus$ and $\|$. 43

3.16 The illustration of $\ominus$ and $\|$. 46

4.1 The expression tree of the model in Eq.(4.2). 50

4.2 The two channels of the SIS in Figure 2.2. 55

4.3 The expression tree representation of the model in Eq.(4.10). 55

5.1 The three sets of scenarios $\mathbf{Sce}(w = W)$, $\mathbf{Sce}(w = D)$ and $\mathbf{Sce}(w = F)$ for w defined by the meet operation $\wedge : \mathbf{WDF}^2 \twoheadrightarrow \mathbf{WDF}$. 61

5.2 A simple fault tree where $w = a.b + c$. 64

5.3 Valuation of $[\![\mathcal{M}]\!] : \mathbf{WF}^3 \twoheadrightarrow \mathbf{WF}$ and relevant sets of scenarios. . 65

5.4 Order-similarity when $\mathrm{dom}(w) = \mathbf{WF}$. 66

5.5 Order-similarity when $\mathrm{dom}(w) = \mathbf{WDF}$ and $\mathrm{dom}(w) = \mathbf{W2F}$. . 67

5.6 The decision diagram $\mathrm{DD}(v)$ with $\mathrm{dom}(v) = \{s_1, s_2, ..., s_m\}$. . . 74

5.7 $\mathrm{DD}(v)$ where $\mathrm{dom}(v)$ is respectively $\mathbf{WF}$, $\mathbf{WDF}$, $\mathbf{SWF}$, $\mathbf{WFdFu}$ and $\mathbf{WFdFs}$. 74

5.8 $\mathrm{DD}(\diamondsuit(u, v))$. 75

5.9 Algorithm of `BuildDD`. 76

5.10 Algorithm of `Combine`. 77

5.11 Algorithm of `Hook`. 77

5.12 Expression tree example. 78

5.13 The results of `BuildDD(n3)`, `BuildDD(n4)` and `BuildDD(n5)`. 78

5.14 The results of `BuildDD(n1)` and `BuildDD(n2)`. 79

5.15 The result of `BuildDD(n0)`. 79

5.16 Algorithm of `Scenario`. 80

5.17 The three sets of scenarios $\mathbf{Sce}(w = W)$, $\mathbf{Sce}(w = D)$ and $\mathbf{Sce}(w = F)$ determined by DD(f_w). 80

5.18 Algorithm of `ConditionalScenario`. 81

5.19 Algorithms of `MinimalScenarios` and `MaximalScenarios`. 81

5.20 Algorithm of `Probability_DD`. 82

5.21 Algorithm of `Probability_Sce`. 82

6.1 Functional architecture of **LatticeX**. 88

6.2 UML class diagram of the classes in **LatticeX**. 89

6.3 The fault tree in Figure 5.2 written in FDS-ML. 89

6.4 The declaration of **WF**, **WDF**, **SWF**, **WFdFu** and **WFdFs** in FDS-ML. 90

6.5 Generic form of textual FDM. 91

6.6 Illustration of the logic operations $\wedge$, $\vee$ and $\neg$. 93

6.7 Illustration of the valuation of $\wedge$ and $\vee$ applied on the ternary FDS **WDF**. 94

6.8 Illustration of $\mathcal{O}_3^1$, $\mathcal{O}_3^2$ and $\mathcal{O}_3^3$. 96

6.9 Idea of taking into account stochastically dependent components in FDMs. 97

6.10 A hot-standby system consisting of a master unit A, a standby unit B and a perfect switch S. 97

6.11 The procedure of abstracting a hot-standby system in FDMs. . . . 98

6.12 State probabilities for $SC1$ and $SC2$. 101

6.13 State probabilities for $System$ satisfying the conditions in Eq.(6.3) and Eq.(6.4). 105

6.14 A train control system. 107

6.15 State probabilities for TCC within 8760 hours. 110

7.1 The epistemic space built over **WF**, **WDF** and **W2F**. 115

7.2 The left- and right-transformation of the Boolean join and meet operators. 123

7.3 The inner-transformation of the Boolean join and meet operators. . 124

7.4 The coherent abstraction $\alpha_{\mathbf{WF}} : (\mathbf{WF} \otimes \mathbf{WF})^u \twoheadrightarrow (\mathbf{WF})^u \otimes (\mathbf{WF})^u$. 125

7.5 Case study in (Misuri et al. 2018), where the premises of the storage farm are outlined in white; the two intrusion paths considered, 'Via Ground' and 'Via Water' are reported as white arrows. 126

7.6 The mass assignment m in the domain of $Attack\_OR$ and $Attack\_XOR$ for $\mathcal{M}_1$ and $\mathcal{M}_2$. 132

8.1 Comparison of the algebraic framework between fault trees and FDSs. 136

8.2 Comparison between fault trees and FDMs. 137

8.3 The binary decision diagrams and the extended decision diagrams for FDMs. 139

A.1 The attributes of the classes related to `FDS`. 156

A.2 The attributes of the classes related to `Formula`. 157

A.3 Definition of identifier in FDS-ML. 158

A.4 Definition of domain in FDS-ML 158

A.5 Definition of operator in FDS-ML 159

A.6 Definition of block in FDS-ML 160

A.7 The textual model of system in Figure 2.2 written in FDS-ML. . . 161

A.8 Sensitivity $\mathbf{Sen}(System = W, v = c)$, $\forall v \in$ **S**, $\forall c \in$ **SIS**. 162

A.9 Sensitivity $\mathbf{Sen}(System = F_s, v = c)$, $\forall v \in$ **S**, $\forall c \in$ **SIS**. 163

A.10 Sensitivity $\mathbf{Sen}(System = F_{dd}, v = c)$, $\forall v \in$ **S**, $\forall c \in$ **SIS**. . . . 164

A.11 Sensitivity $\mathbf{Sen}(System = F_{du}, v = c)$, $\forall v \in$ **S**, $\forall c \in$ **SIS**. . . . 165

Abbreviations

BDD	Binary decision diagram
DFT	Dynamic fault tree
EUC	Equipment under control
FDS	Finite degradation structure
FDS-ML	Finite Degradation Structure Modeling Language
FDM	Finite degradation model
FT	Fault tree
MBSA	Model Based Safety Analysis
MBSE	Model Based Systems Engineering
MDD	Multi-valued decision diagram
MMDD	Multistate multi-valued decision diagram
MSS	Multi-state systems
MVL	Multi-valued logic
UGF	Universal Generating Function
OOP	Object-oriented programming
OBDD	Ordered binary decision diagram
POP	Procedure-oriented programming
PRA/PSA	Probabilistic Risk/Safety Analyses
RBD	Reliability block diagram
SIS	Safety instrumented system
UML	Unified Modeling Language
ZBDD	Zero-suppressed binary decision diagram

List of Symbols

$\mathbf{FDS}$	The set of all finite degradation structures.
$\otimes$	Monoidal product of finite degradation structures.
Φ	The set of abstractions between finite degradation structures.
$\langle \mathbf{FDS}, \otimes, \Phi \rangle$	The algebraic framework of finite degradation structures.
$\mathbb{B}$	The Boolean set of truth values with $\mathbb{B} = \{0, 1\}$.
$\vee$	The conjunction or join operator.
$\wedge$	The disjunction or meet operator.
$\neg$	The negation operator.
$\sqsubseteq$	Partial order interpreted as degradation order.
$\sim$	Incomparable relation with respect to the order $\sqsubseteq$.
$\cong$	Equal up to isomorphism.
$\perp$	Least element of a poset.
$\top$	Greatest element of a poset.
p	Probability measure.
$p_{\otimes}$	Probability measure in the result domain of a product $\otimes$.
$\langle D, \sqsubseteq \rangle$	Partially ordered set.
$\langle D, \sqsubseteq, \perp \rangle$	Meet-semi-lattice.
$\langle D, \sqsubseteq, \top \rangle$	Join-semi-lattice.
$\langle D, \sqsubseteq, \perp, p \rangle$	Finite degradation structure.
W	Working state
S	Standby state
D	Degraded state
F	Failed state
F_d	Failed-detected state
F_u	Failed-undetected state
F_{dang}	Failed-dangerously state
F_{safe}	Failed-safely state
$\mathbf{WF}$	Finite degradation structure with states W and F.
$\mathbf{WDF}$	Finite degradation structure with states W, D and F.

$\mathbf{SWF}$	Finite degradation structure with states S, W and F.
$\mathbf{WFdFu}$	Finite degradation structure with states W, F_d and F_u.
$\mathbf{WFdFs}$	Finite degradation structure with states W, F_{dang} and F_{safe}.
$\mathbf{SIS}$	The FDS for the component in safety instrumented system.
F_s	Failed safely state in $\mathbf{SIS}$.
F_{dd}	Failed dangerous detected state in $\mathbf{SIS}$.
F_{du}	Failed dangerous undetected state in $\mathbf{SIS}$.
$\mathbf{n}$	Linearly ordered FDS with $n \in \mathbb{N}^+$ states.
$\mathbf{WnF}$	FDS with one least element and n maximal elements.
$\mathcal{L}^n$	The product of n FDS $\mathcal{L}$.
$\mathbf{1}$	The identity FDS of the product $\otimes$.
$\varphi : \mathcal{S} \twoheadrightarrow \mathcal{T}$	Abstraction φ from $\mathcal{S}$ to $\mathcal{T}$.
$\text{dom}(\varphi)$	The domain of φ.
$\text{codom}(\varphi)$	The codomain of φ.
$\oslash$	The series composition in the safety instrumented system.
$\parallel$	The parallel composition in the safety instrumented system.
$\mathbf{U}$	A set of FDSs.
$\mathbf{O}$	The set of operators (or operations).
$\mathbb{N}$	The set of natural numbers.
α	The arity of operator.
$\mathbf{V}$	The set of variables.
$\mathbf{S}$	The set of state variables.
$\mathbf{F}$	The set of flow variables.
$f, f_1, ..., f_n$	Well-formed and well-typed formulas.
$\diamondsuit$	Example of operator.
$w := f$	Equation that defines w by f.
$\mathcal{E}$	The set of equations.
$\mathcal{M}$	Finite degradation model.
$\text{dom}(v)$	The valuation domain of a variable v.
$\text{var}(f)$	The set of variables appearing in the formula f.
$\langle \diamondsuit, n_l, n_r \rangle$	Internal node of a formula tree, labeled with $\diamondsuit$ and pointing to the left-child node n_l and right-child node n_r.
$\langle v, /, / \rangle$	Terminal node of a formula tree labeled with variable v.
$/$	`nil`.
f_w	The syntactic solution of w.
$[\![\cdot]\!]$	The interpretation of a symbol.
σ	The partial valuation of state variables.
$\vec{\sigma}$	The total valuation of variables.
$[\![w]\!](\sigma)$	The valuation of w under σ.
$[\![\mathcal{M}]\!]_w$	The semantic solution of w in $\mathcal{M}$.

$w = y$	The observer indicating that the valuation of w is y.
$w \neq y$	The observer indicating that the valuation of w is not y.
$w \in Y$	The observer indicating that the valuation of w is in Y.
$w \notin Y$	The observer indicating that the valuation of w is not in Y.
$\overline{\mathbf{v}}$	A state vector or a scenario in $\bigotimes_{v\in\mathbf{S}} \mathrm{dom}(v)$.
$\mathbf{Sce}(o)$	The set of scenarios satisfying the observer o.
$\mathrm{Min}\mathbf{Sce}(o)$	The set of minimal scenarios satisfying the observer o.
$\mathrm{Max}\mathbf{Sce}(o)$	The set of maximal scenarios satisfying the observer o.
$\mathbf{Sce}(w = y\|v = c)$	The set of conditional scenarios satisfying the observer $w = y$ and the valuation $\sigma(v) = c$.
$\mathbf{C_S}$	The conditions made on state variables.
$\mathbf{C_F}$	The conditions made on flow variables.
$I_\mathbf{S}$	The index set of variables appearing in $\mathbf{C_S}$.
$I_\mathbf{F}$	The index set of variables appearing in $\mathbf{C_F}$.
$\mathbf{Sce}(\mathbf{C_F}\|\mathbf{C_S})$	The set of conditional scenarios satisfying the conditions in $\mathbf{C_S}$ and $\mathbf{C_F}$.
p_i	Probability measure in $\mathrm{dom}(v_i)$.
$\Pr\{\overline{\mathbf{v}}\}$	The scenario probability of $\overline{\mathbf{v}}$.
$p_w(y)$	The state probability of y in $\mathrm{dom}(w)$.
$\Pr\{w = y\|v_j = c\}$	The probability that $[\![\mathcal{M}]\!]_w(\sigma) = y$ under the condition that $\sigma(v_j) = c$.
$\Pr\{\mathbf{C_F}\|\mathbf{C_S}\}$	The conditional probability satisfying the conditions in $\mathbf{C_S}$ and $\mathbf{C_F}$.
$\mathbf{Sen}(w = y, v_j = c)$	The sensitivity of $p_w(y)$ with respect to $p_j(c)$.
$\alpha_j(x, c)$	The sensitivity coefficient of x by c for p_j.
$\mathbf{A}_j$	The sensitivity matrix for p_j.
$\mathrm{DD}(f)$	The decision diagram encoding the valuation of f.
(s, v, n_1, n_2)	Internal node of a decision diagram, labeled by state constant s, variable constant v, and pointing to the then-child n_1 and the else-child n_2.
$(s, /, /, /)$	Terminal node of a decision diagram labeled with state constant s.
$\prec$	The variable ordering on state variables.
$\mathcal{O}^k_n$	The k-out-of-n operator for Boolean systems.
$\wedge_{\mathbf{HS}}$	The operation between the main unit and the standby unit in a hot-standby system.
Θ	A finite set of states.
Ω	The epistemic space built over the set Θ.
$\Omega/_\equiv$	The quotient set of Ω by $\equiv$.

m	Mass assignment.
$\mathbf{Bel}(X)$	The belief of X.
$\mathbf{Pl}(X)$	The plausibility of X.
$\mathbf{Com}(X)$	The commonality of X.
$(\mathcal{L})^u$	The transformation of the FDS $\mathcal{L}$ into the epistemic space.
$\mathbf{Best}(s)$	The belief that the degradation level is in the best case to be s.
$\mathbf{Worst}(s)$	The belief that the degradation level is in the worst case to be s.
ϕ_L	The left-transformation of an operation ϕ.
ϕ_R	The right-transformation of an operation ϕ.
ϕ_u	The inner-transformation of an operation ϕ.
ϕ^u	The outer-transformation of an operation ϕ.

Chapter 1

Introduction

1.1 Background

Probabilistic risk/safety assessment (PRA/PSA) is aimed at evaluating the risk of a system using a probabilistic method: a comprehensive structured approach for identifying failure scenarios, constituting a conceptual and a mathematical tool for deriving numerical estimates of risk (Verma et al. 2010). The PRA/PSA is currently being widely applied in many fields, *viz.*, chemical and process plants, nuclear facilities, aerospace, and even financial management.

The general procedure of PRA/PSA consists of six steps: 1) a thorough understanding of the system; 2) hazard identification (of initiate events); 3) accident sequence modeling; 4) system modeling (i.e. quantification of failure probabilities and accident sequence frequencies); 5) consequence analysis; 6) risk management, including risk estimation, risk evaluation and decision-making. Regarding these six steps, this PhD thesis aims at contributing to step 4), i.e. to propose more concise and convenient models for PRA/PSA.

Since models are built to simulate reality, simplifications and idealizations are inevitable in the modeling process, which may influence the accuracy of PRA/PSA in decision-making. It is thus of high importance to choose appropriate models or modeling approaches to balance the trade-off between the accuracy of results and the complexity of calculation.

Since 1960s, a variety of modeling methodologies has been put forward, such as fault trees, reliability block diagrams, etc. These models belong to the category of combinatorial models, i.e. where the system's state is modeled by the combination of the states of its components. Fault trees and reliability block diagrams are Boolean (combinatorial) models, where the state of component/system is assumed to be either working or failed. In spite of their great success in the last century, these traditional modeling approaches are now challenging by the rapidly expanding complexity of systems. Although a lot of software has been developed to facilitate the modeling process, these methods, which are highly relied on their graphical representations, are still limited by the size of either computer screens or drawing papers. Moreover, models written in those formalism encode coarse approximations of the system's behavior under study. They make actually strong assumptions — stochastic independence and binary assumption — and for this reason are unable to represent faithfully cold redundancies, time dependencies, resource sharing, reconfiguration, and mutistate systems.

As of today, more powerful modeling formalisms exist, e.g. Markov chains and stochastic Petri nets. These models belong to the category of state/transition models (technically finite state automata), which are different in essence from combinatorial models. When leaving the combinatorial realm, the computational complexity increases dramatically because those models mainly use stochastic simulations to calculate the required indicators.

A good compromise is to stay in the combinatorial realm while allow the state of component/system to be multi-valued. This provides with the motivation of this PhD thesis to contribute in developing more suitable and convenient modeling framework for PRA/PSA.

1.2 Objectives

The main objective of this PhD thesis is to develop a unified modeling frameworks of combinatorial reliability models. To realize this objective, the following tasks are identified:

1. Have a broad literature review of (industrial and academic) modeling formalisms of combinatorial reliability models.

2. Propose possible complementary analysis tools for existing combinatorial reliability models.

3. Make a summary of existing combinatorial reliability models and draw the blueprint of the new modeling framework.

4. Establish both theoretical and algorithmic foundations of the objective modeling framework.

5. Make experiments and evaluations of the objective modeling framework and suggest its possible future industrial applications.

1.3 Research approaches and work process

The PhD work follows the way of defining the problem, developing mathematical foundations, designing computer algorithms and implementing experiments. The work process consists of 6 stages with their respective research approaches.

Stage 1: Literature review This PhD project starts by fundamental PhD courses and state-of-the-art literature review. For literature review, we first classify existing reliability models into several categories. Then, we summarize their (a) commonalities/differences and (b) advantages/deficiencies. Based on the results of (a), we identify the minimal set of concepts that should be included in the objective modeling framework and based on the results of (b), we decide the most appropriate theories and technologies that can be used to build the objective modeling framework. The result of literature review is now updated and given in Part I of this thesis.

Stage 2: Development of fault tree synthesis tool Before start to work on the new modeling framework, we developed a small complementary tool for fault tree analysis and published a paper on this subject, see Yang and Rauzy (2019b). This paper is also attached in Appendices B of this thesis. The main objective at this stage is to prepare the knowledge on Boolean expression diagrams and the algorithms on binary decision diagrams. These two techniques are the technical foundations in the implementation of the new modeling framework of this thesis.

Stage 3: Establishment of theoretical foundations The theoretical foundations of our new framework are established based on the literature review results of stage 1. The main task is to theoretically extend all the concepts used in fault tree analysis from Boolean systems into multistate systems. The objective at this stage is to make the proposed modeling framework logically self-consistent and mathematically correct. At this stage, we published several conference papers to present the modeling methodology of our new framework, see Yang et al. (2018), Yang and Rauzy (2018), Yang et al. (2019).

Stage 4: Implementation in Python In order to benefit from the efficiency of automatic calculation by computers, we developed a software called **LatticeX** in Python as a full computer-based implementation of the proposed modeling framework. At this stage, we focus on the design of data structures and algorithms.

During the development of **LatticeX**, we perform both unit and functional tests to verify its correctness. The latest version of algorithms can be found in our paper Rauzy and Yang (2019). To facilitate the modeling process, we designed a specific modeling language called FDS-ML to allow writing models in text files as input of **LatticeX**. The introduction of FDS-ML can be found in our paper Yang and Rauzy (2019a).

Stage 5: Practical case studies In order to illustrate the practical interest of the proposed modeling framework, we look for suitable case studies in existing literature and apply our modeling methodology to these case studies to make comparison with their original modeling approaches. We started from the simplest cases, i.e. that the systems are modeled by fault trees, reliability block diagrams, etc. Some results are presented in our paper Yang and Rauzy (2018). Then, after our software has been developed, we are able to deal with more complex systems, which may have more states, more components and more types of interrelations. Some results are presented in Section 6.3. Moreover, inspired by these case studies, we recognized that it is necessary to integrate a modeling library in our framework (and finally in our software), so that some commonly used models and operators can be stored and reused. The current progress is presented Section 6.2.

Stage 6: Looking for potential applications In addition to traditional reliability and safety analyses, we also look for other fields of applications of our new modeling framework. Up to now, we have made some progresses on two applications. The first is the modeling of epistemic uncertainties in combinatorial reliability and safety models. The current results are presented in Chapter 7. We also submitted a paper related to this subject entitled “epistemic space of degradation processes”. The second application is to use the proposed modeling framework as a formal interface between the Model-Based System Engineering (MBSE) and the Model-Based Safety Assessment (MBSA) to support their structural and behavioral synchronizations. This idea is presented in our paper Yang et al. (2018).

1.4 Structure of thesis

The structure of this thesis is pictured Figure 1.1. In this figure, we also indicate where the results of the 6 stages mentioned above can be found.

The thesis is divided into three parts.

- Part I
  This part synthesizes the results of the literature review. In Chapter 2, we

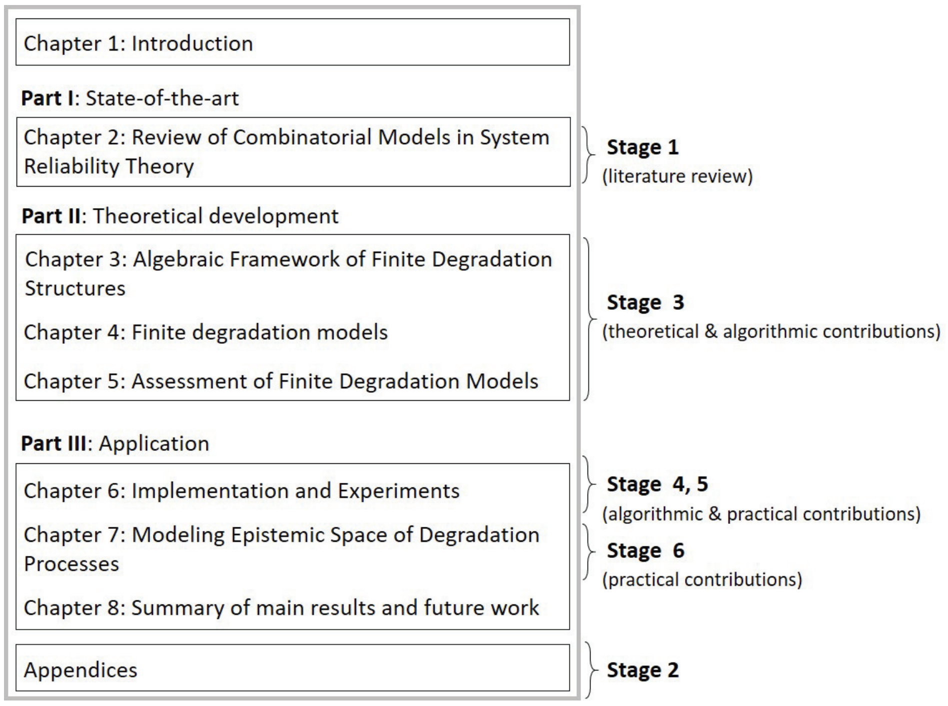

Chapter 1: Introduction
Part I: State-of-the-art
Chapter 2: Review of Combinatorial Models in System Reliability Theory
Stage 1
(literature review)
Part II: Theoretical development
Chapter 3: Algebraic Framework of Finite Degradation Structures
Chapter 4: Finite degradation models
Chapter 5: Assessment of Finite Degradation Models
Stage 3
(theoretical & algorithmic contributions)
Part III: Application
Chapter 6: Implementation and Experiments
Chapter 7: Modeling Epistemic Space of Degradation Processes
Chapter 8: Summary of main results and future work
Stage 4, 5
(algorithmic & practical contributions)
Stage 6
(practical contributions)
Appendices
Stage 2

Figure 1.1: Outline of the structure of thesis.

first present the taxonomy of reliability and safety models and then summarize the existing combinatorial models. Based on the literature review, we pinpoint the need of new modeling framework. At the end of this chapter, we present the case study of a safety instrumented system, which will be used throughout the entire thesis to illustrate our modeling methodology.

- Part II
  This part focuses on the theoretical development of the proposed modeling framework. Chapter 3 presents the notion of finite degradation structures and the operations that can be applied on finite degradation structures. Chapter 4 defines both of the syntax and the semantics of the models built on finite degradation structures. Chapter 5 introduces the two types of assessments of finite degradation models, i.e. the scenarios analysis and the probabilistic calculation. The decision diagram based algorithms that are used to perform the required assessments are also included in this chapter.

- Part III
  This part introduces the implementation and the applications of finite degradation models. Chapter 6 presents the software we have developed to

support the computer-based implementation of finite degradation models. The functional architecture of the software, the modeling language FDS-ML and the experimental results are given this chapter. Chapter 7 introduces the methodology of modeling the epistemic uncertainty using finite degradation models. Finally, Chapter 8 summarizes this PhD thesis and discusses the future work.

1.5 Academic publications

The academic publications during the PhD period are listed as follows:

1. (Conference paper)
   Reliability modeling using finite degradation structures
   Liu Yang and Antoine Rauzy
   3rd International Conference on System Reliability and Safety (ICSRS 2018), Barcelona, November 2018

   Note: In this paper, we present the first version of the theoretical framework of finite degradation structures. We also show the modeling and the assessment of several simple multistate systems.

2. (Conference paper)
   Finite degradation structures: a formal framework to support the interface between MBSE and MBSA
   Liu Yang, Antoine Rauzy and Cecilia Haskins
   2018 IEEE International Systems Engineering Symposium (ISSE), Rome, October 2018

   Note: In this paper, we discuss the role of finite degradation structures of being the interface between the Model-Based System Engineering and the Model-Based Safety Assessment to support their structural and behavioral synchronizations.

3. (Conference paper)
   Reliability assessment of phased-mission systems with AltaRica 3.0
   Michel Batteux, Tatiana Prosvirnova, Antoine Rauzy and Liu Yang
   3rd International Conference on System Reliability and Safety (ICSRS 2018), Barcelona, November 2018

   Note: This paper presents a modeling pattern of phased-mission systems using the high-level modeling language AltaRica 3.0.

4. (Conference paper)
   Finite degradation analysis of multiple safety instrumented systems

Liu Yang, Antoine Rauzy and Mary Ann Lundteigen
29th European Safety and Reliability Conference (ESREL 2019), Hanover, September 2019

Note: In this paper, we present the modeling and the assessment of multiple safety instrumented systems using finite degradation structures. This work is now improved. The results are updated in Section 6.3.1.

5. (Conference paper)
   FDS-ML: a new modeling formalism for probabilistic risk and safety analyses
   Liu Yang and Antoine Rauzy
   6th International Symposium on Model-Based Safety and Assessment (IMBSA 2019), Thessaloniki, October 2019

   Note: In this paper, we present the modeling language FDS-ML, which is designed to describe finite degradation models in text files.

6. (Journal paper)
   Model synthesis using Boolean expression diagrams
   Liu Yang and Antoine Rauzy
   Reliability Engineering & System Safety, 2019.

   Note: In this paper, we present a fault tree synthesis tool called model synthesis, which is used to reshape large automatically generated fault trees to make them more informative for reliability and safety analyses.

7. (Journal paper)
   Finite degradation structures
   Antoine Rauzy and Liu Yang
   Accepted and under publication by Journal of Applied Logic, November 2019.

   Note: In this paper, we complete the theoretical framework of finite degradation structures, where the formal definitions and the mathematical demonstrations can be found.

8. (Journal paper)
   Decision diagram algorithms to extract minimal cutsets of finite degradation models
   Antoine Rauzy and Liu Yang
   Information, November 2019.

   Note: In this paper, we report our newest algorithms of extracting minimal cutsets of finite degradation models.

9. (Journal paper, under review)
   Epistemic space of degradation processes
   Liu Yang and Antoine Rauzy
   Under review by Journal of Applied Non-Classical Logics, submitted in July 2019.

   Note: In this paper, we present the modeling of epistemic uncertainty using finite degradation models. The results of this work are now updated in Chapter 7.

Part I

State of the Art

Chapter 2

Review of Combinatorial Models in System Reliability Theory

Probabilistic risk/safety analyses (PRA/PSA) are used in virtually all industries to assess whether the risk of operating complex technical systems (aircraft, nuclear power plants, offshore platforms...) is low enough to be socially acceptable. The WASH1400 report (Rasmussen 1975), which followed the Three Mile Island nuclear accident, is usually considered as the historical starting point of their worldwide, cross-industry adoption.

PRA/PSA models describe how systems may degrade and eventually fail under the occurrence of random events such as failures of mechanical components, sudden changes of environmental conditions or human errors.

In this chapter, we will give a short review of the combinatorial models applied in PRA/PSA and explain the motivation of proposing the modeling framework of finite degradation structures. At the end, a case study extracted from the standard ISO/TR 12489 is presented. This case study will be used throughout the thesis to illustrate the proposed modeling methodology.

2.1 Taxonomy of reliability models

To start, consider the process of analyzing system reliability pictured Figure 2.1. The process is divided into 4 steps: system identification, construction of reliability models, assessment of models and analysis of resultant indicators.

In this process, we separate PRA/PSA models (step 2) with their assessment tech-

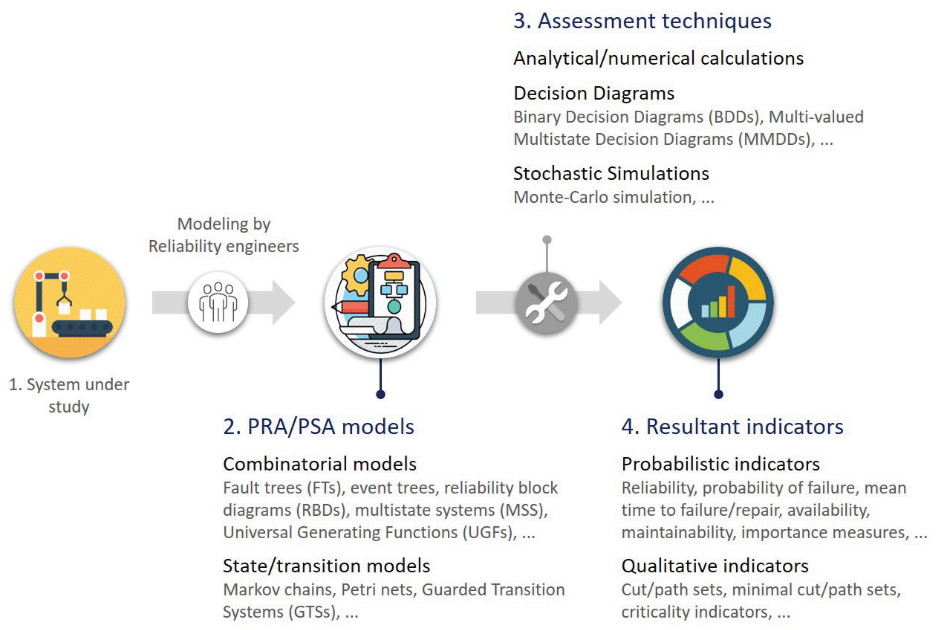

3. Assessment techniques
Analytical/numerical calculations
Decision Diagrams
Binary Decision Diagrams (BDDs), Multi-valued Multistate Decision Diagrams (MMDDs), ...
Stochastic Simulations
Monte-Carlo simulation, ...
Modeling by Reliability engineers
1. System under study
2. PRA/PSA models
Combinatorial models
Fault trees (FTs), event trees, reliability block diagrams (RBDs), multistate systems (MSS), Universal Generating Functions (UGFs), ...
State/transition models
Markov chains, Petri nets, Guarded Transition Systems (GTSs), ...
4. Resultant indicators
Probabilistic indicators
Reliability, probability of failure, mean time to failure/repair, availability, maintainability, importance measures, ...
Qualitative indicators
Cut/path sets, minimal cut/path sets, criticality indicators, ...

Figure 2.1: Reliability models, assessment techniques and indicators.

niques (step 3). For the former, we concern on their expressive power of describing reliability and safety behaviors, while for the latter, we concern on their computational accuracy and efficiency. It is worth noting that there isn't a strict one-to-one correspondence between the model and its assessment technique. For instance, the Monte-Carlo simulation technique can be applied to many models, such as Markov chains, Petri nets, guarded transition systems and fuzzy fault trees.

The PRA/PSA models can be splitted into 3 nested categories (Rauzy 2018), which are (probabilized) Boolean formulas, (stochastic) finite state automata and (stochastic) process algebras.

In Figure 2.1, we only list the first two categories that are related to this thesis:

– **Combinatorial** models, where the system's behavior is modeled by the combination of the components' behavior according to certain rules. This category includes the models like fault trees, event trees, reliability block diagrams and the so-called multistate systems.

– **State/transition** models (i.e. finite state automata), where the system's behavior is modeled by transitions (triggered by events) between states. This category includes the models like Markov chains, Petri nets (Marsan et al.

1995), guarded transition systems (Rauzy 2008) and dynamic fault trees.

In this thesis, we are focusing on the category of combinatorial models. But sometimes it is also useful to embed state/transition models in combinatorial models. This will be briefly presented Section 6.2.4.

In the following section, we will give a short review of the existing combinatorial reliability models. Their advantages and deficiencies are discussed in details. Finally, we will pinpoint the motivation of proposing the new modeling framework.

2.2 Combinatorial models

2.2.1 Overview

The word "combinatorial" refers to the property that the model of a system can be constructed by successively composing submodels of its constituent subsystems according to certain combination rules.

The advantages of using combinatorial models are twofold. First, thanks to this compositionality, we can decompose large complex systems into independent parts, model and analyze them separately, and finally use combination rules to compose them to analyze the behavior of the whole system. It avoids modeling of the entire system at once and also facilitates the validation and verification of the model.

The second advantage is that such decomposition/composition process exhibits implicitly a hierarchical structure. Through this structure, we can either top-down trace the causes of failure or bottom-up propagate failure probabilities to calculate probabilistic indicators of higher abstraction level objects.

2.2.2 Fault tree methodology and limitations

In reliability and safety analysis, the widest applied combinatorial models are fault trees. Fault tree analysis was first conceived by Watson et al. (1961) to study the Minuteman Launch Control System. Now, it is one of the most prominent techniques used by a wide range of industries (Kumamoto 1996, Andrews 2002). Textbooks of fault tree analysis can be seen in e.g. Rausand (2004), Ruijters and Stoelinga (2015).

Fault trees have been well mastered by practitioners. Safety standards such as IEC 61508 (IEC61508 2010) (safety systems), ISO 26262 (Standard 2011) (automotive industry), or ARP 4761 (International 1996) (avionic industry) recommend their use.

Event trees and reliability block diagrams are fault tree alternatives, which are also widely used in PRA/PSA. Many other modeling formalisms are also related to fault trees, e.g. Go-Flows (Matsuoka and Kobayashi 1988) and HiP-HOPS (Papadopoulos and McDermid 1999, Adachi et al. 2011).

Strictly speaking, fault trees, event trees and reliability block diagrams are graphical representations of (probabilized) Boolean formulas. As graphical representations, they visualize the logical relationships between events and causes that lead to the failure of the system. Such logic diagrams facilitate the process of identifying failures and possible interactions. They also create a visual aid for system analysis and management.

However, the graphical representations are not the main concern of this thesis. In the following sections, when we discuss limitations, advantages and deficiencies of a modeling approach, we refer to its mathematical or theoretical framework not the graphical representation.

According to Fussell (1975), the limitations of fault tree methodology fall into two categories: limitations occurring during implementation and limitations in the theory. After the literature review, we summarize these limitations as follows:

- **Limitation 1** (practical): the computational complexity of extracting (minimal) cutsets out of large models, due to the exponential blow-up of the number of state combinations.
- **Limitation 2** (theoretical): the assumption that components are stochastically independent. Accordingly, dynamic behaviors such as cold standby, repairs and resource sharing cannot be captured.
- **Limitation 3** (theoretical): the binary assumption, i.e. components can only be either working or failed. Accordingly, degraded performances cannot be considered.

This thesis aims at solving the limitation 3, i.e. to fully extend the fault tree analysis into multistate realm. For this reason, a more detailed literature review of existing solutions for limitation 3 is given next section. As for limitation 1 and 2, we address some typical solutions below.

Solutions for limitation 1 This limitation comes from the computational complexity of combinatorial problems. It exists in both Boolean and multistate combinatorial models. For Boolean models, the widely applied algorithms of extracting (minimal) cutsets are MOCUS (Fussell 1972) and the algorithms on Binary Decision

Diagrams (BDDs) (Akers 1978, Lee 1959). Now, the latter become more prevailing than the former for being more efficient. Many researches have been dedicated to improve the computation efficiency of BDD algorithms, e.g. OBDDs (Bryant 1986), ZBDDs (Minato 1993) and advanced algorithms, see e.g. Brace et al. (1991), Minato (1993), Rauzy (2001), Jung et al. (2004), Rauzy (2012). Moreover, since the calculation efficiency of OBDD strongly depends on its variable ordering, a lot of ordering heuristics have been invented to improve the efficiency, see e.g. Bouissou et al. (1997), Ibañez-Llano and Rauzy (2008), Mo et al. (2013).

Solutions for limitation 2 The main direction of solving this limitation is to apply state/transition models to model the dynamic dependencies. The dynamic fault trees (DFTs) (Dugan et al. 1990; 1992) are extension of fault trees where the temporal sequencing information are modeled by additional dynamic gates. Many researches have contributed to DFTs, see e.g. Tang and Dugan (2004), Walker and Papadopoulos (2009), Merle et al. (2010), Zhang et al. (2011), Rauzy (2011), Chaux et al. (2013). DFTs can also link to other state/transition models, e.g. Petri nets (Codetta-Raiteri 2005) and Markov chains (Boudali et al. 2007b;a). These models exhibit the trend of combining fault trees and state/transition models, such as Boolean logic driven Markov processes (Bouissou and Bon 2003, Piriou et al. 2017), reliability block diagrams driven Petri nets (Signoret et al. 2013) and state/event fault trees (Kaiser et al. 2007). In these hybrid models, fault trees are used at higher abstraction level to structure the state/transition models. More powerful formalisms also exist to structure state/transition models, e.g. high-level modeling language AltaRica 3.0 (Prosvirnova 2014, Batteux et al. 2013) and SAML (Gudemann and Ortmeier 2010).

2.2.3 Existing solutions of extending fault tree analysis to multistate systems

Under the binary assumption, systems and components are assumed to be either working or failed. However, in practice, the performance level of systems and their components may vary from perfect operation to complete failure. To be able to model the intermediate states between working and failed, the notion of multistate systems was introduced in the context of cannibalization (Hirsch et al. 1968, Hochberg 1973).

In this section, we classify existing combinatorial models related to this notion of multistate systems into five categories and discuss whether they fully extend the fault tree analysis to multistate systems or not. It is worth reminding that the results of fault tree analysis are of two types: qualitative results (e.g. (minimal) cut/path sets) and quantitative results (e.g. probabilistic indicators like probability of failure, mean time between failures and importance measures). These results are used as criteria to judge the advantages and deficiencies of the following modeling approaches.

(1) Models with totally ordered state space

In these models, the state space of multistate component/system is assumed to be totally ordered, i.e. the valuation set of a multistate component/system that has m states is $\{0, 1, ..., m\}$ with the total ordering $0 < 1 < ... < m$. Denote the valuation set of the system by M and the valuation set of its ith component by M_i. Then, the structure function of a system consisting of n components is treated as a discrete function from $\prod_{i=1}^{n} M_i$ to M (Murchland 1975, Janan 1985, Wood 1985). Most of the researches related to this subject target on coherent systems, see e.g. Griffith (1980), El-Neweihi et al. (1978), Barlow and Wu (1978). Thanks to this total ordering among states, the notion of minimal cut/path sets can be defined as the minimal upper vectors or the maximal lower vectors with respect to different degradation levels, see e.g. Huang (1984), Janan (1985). Recent researches can be found in e.g. Ohi (2010), Zaitseva and Levashenko (2013), Liu et al. (2015) and Kvassay et al. (2016).

Advantage This approach is suitable for modeling linear degradation or linear performance levels between the ideal operating state and the complete failed state. As results, both qualitative and quantitative results can be obtained.

Deficiency It cannot be applied if there are pairs of incomparable states.

(2) Models with partially ordered state space

Yu et al. (1994) indicated that *"the degradation process of components may fall into separate directions, so that the states cannot always be totally ordered by a reasonable degradation relation"*. Therefore, a good solution is to use partially ordered set to model the state space of multistate component/system. Ohi (2013) stated that *"a basic problem of reliability theory is to explain the algebraic and stochastic relationship between a product partially ordered set and a partially ordered set through an increasing mapping from the former to the latter"*. Except for the work of Yu et al. (1994) and Ohi (2013), there are however not many researches related to this subject. In Yu et al. (1994) and Ohi (2013), the definition and the calculation of minimal cut/path sets and probabilistic indicators are well established. The stochastic analyses of such systems can be found in e.g. Langseth and Lindqvist (1998), Lindqvist (2003) and Ohi (2016).

Advantage Comparing to the models in category (1), the incomparable states are taken into account. As results, both qualitative and quantitative results can be obtained.

Deficiency In all existing literature related to this subject that the author has reviewed, the operations used to model the system are limited to the two universal operations — supremum and infimum — for partially ordered sets. However, to model reliability and safety behaviors, these two operations seem to be lack of versatility.

(3) Extended fault trees

Buchacker (Buchacker et al. 1999, Buchacker 2000) introduced the notion of extended fault trees to deal with multistate components. In such fault trees, each non-working state (i.e. degradation or failed state) is modeled by an individual basic event. Their interrelations are modeled by specifically designed multivalued logic gates. Related researches can be found in e.g. Portinale and Codetta-Raiteri (2011).

Advantage This approach is easy to understand and use. As results, probabilistic indicators of the top event can be calculated.

Deficiency To the best of the author's knowledge, the minimal cut/path sets haven't been formally defined in such models yet.

(4) Multi-valued logic or decision diagram based models

As extension of BDD, the notion of multiple-valued decision diagram (MDD) was introduced by Miller (1993) to graphically represent multi-valued logic functions. Then, Xing (2007), Xing and Dai (2009) proposed the notion of multistate multivalued decision diagrams (MMDD) based on the work of Zang et al. (2003). Applications of MMDD or MDD can be found in e.g. Shrestha et al. (2010), Zaitseva and Levashenko (2013), Li et al. (2014; 2017), Mo et al. (2018), Zhao et al. (2019).

Advantage This approach is a direct extension of BDD in multistate cases. As results, probabilistic indicators can be calculated.

Deficiency In most of the cases, the structure function of combinatorial models is a certain type of logic functions. Therefore, it is very appropriate to use decision diagrams to represent its valuation. But in other words, decision diagrams are just graphical representations of logic functions, while the determination of the structure function represented by the decision diagram should be done by other approaches. For example, in the work of Zang et al. (2003), Xing (2007), Xing and Dai (2009), they use multistate fault trees to define the required structure function. For this reason, in Figure 2.1 we prefer to regard those approaches by which we can define structure functions as PRA/PSA models, while regard the decision diagrams as assessment technique of PRA/PSA models. Accordingly, whether the minimal cut/path sets can be calculated or not doesn't depend on the decision diagram, but depends on the approach by which its structure function is defined.

(5) Universal Generating Function

The Universal Generating Function (UGF) techniques have been applied to model the performance distribution of multistate systems using algebraic procedures (Levitin 2005). Mathematically, the UGF of a random variable X is defined as $u_X(z) = \sum_{j=1}^{k} p_j z^{x_j}$. The UGF of a random variable Y such that $Y = f(X_1, X_2, ..., X_n)$ is calculated as follows:

$$u_Y(z) = \sum_{j_1=1}^{k_1} \sum_{j_2=1}^{k_2} \cdots \sum_{j_n=1}^{k_n} (p_{1j_1} p_{2j_2} \cdots p_{nj_n}) z^{f(x_{1j_1}, x_{2j_2}, \dots, x_{nj_n})}$$

In the above formula, $X_i = (x_{i1}, x_{i2}, ..., x_{ik_i})$ is the state vector of the ith component, p_{ij_i} is the corresponding probability of state x_{ij_i} and f is the structure function that models the relation between $X_1, ..., X_n$ and Y. This method was introduced by Ushakov (1986; 1988) and then developed by a burst of research papers, e.g. Levitin (2004), Lisnianski (2003; 2007), Lisnianski et al. (2010).

Advantage The calculation of probabilistic indicators of Y from X_i is easy to be implemented. The computation algorithms are comparatively easier than those of decision diagrams.

Deficiency To the best of the author's knowledge, the structure function f in such model should be a real function, which means that the states of $X_1, ..., X_n$ and Y are modeled by real numbers. If f is a discrete logic function, the complexity of this approach is equivalent (in theory) to those decision diagram based methods in (4). Moreover, the (minimal) cut/path sets — in the view of fault tree analysis not the reachability problem of networks — haven't been included in the UGF methodology yet.

2.2.4 Need for new modeling framework

We summarize the aforementioned five categories of models in Table 2.1, comparing their valuation set of multistate component/system, operations used to model the interrelations and accessible assessment results.

Table 2.1: Comparison of existing multistate combinatorial models.

	Valuation set			Operations	Results	
	Finite	Ordered	Values	(op.)	PI*	MC(P)S*
(1) TOSP*	Yes	Yes	$\mathbb{N}$	min & max, MVL op.	Yes	Yes
(2) POSP*	Yes	Yes	Symbols	inf & sup**	Yes	Yes
(3) EFTs*	Yes	No	Symbols	MVL op.	Yes	No
(4) MVL/DD*models	Yes	No	Symbols	MVL op.	Yes	No
(5) UGF*	Yes	No	$\mathbb{R}$	Arithmetic op.	Yes	No
(new) FDSs*	Yes	Yes	Symbols	MVL op.	Yes	Yes

* TOSP: totally ordered state space; POSP: partially ordered state space; EFTs: extended fault trees; MVL: multi-valued logic; DD: decision diagram; UGF: universal generating function; PI: probabilistic indicators; MC(P)S: minimal cut/path sets; FDSs: finite degradation structures.
** infimum and supremum.

From this table, we can conclude that:

- The valuation set of multistate component/system is always finite.
- The elements in the valuation set can be ordered or not. The difference is that if they are ordered, the notion of minimal cut/path sets can be defined with respect to this order.
- The elements in the valuation set can be either numerical or symbolic. They should be consistent with the operations that are used to define the structure function of the system, i.e. arithmetic operations for numerical values and logic operations for symbolical values.
- All the five types of models support the calculation of probabilistic indicators, but only those models whose valuation set is an ordered set can support the calculation of minimal cut/path sets.

Th new modeling framework proposed this thesis is called finite degradation structures (FDSs). According to the conclusions above, we can draw the blueprint of FDSs:

- The valuation set should be **finite** and **partially ordered**. The elements in this set represent the states of multistate component/system. To be more informative in reliability and safety analyses, these elements should be **symbolic**, e.g. Working, Degraded, Failed, Safe, Dangerous, Normal, etc.
- The structure function of the system is constructed by **multi-valued logic operations**. Based on these operations, we can express the failure logic of the system in terms of different combinations of its components' states.

- Under the framework of FDSs, we should be able to calculate probabilistic indicators and minimal cut/path sets (as those in fault tree analysis) for multistate systems.

The algebraic framework of FDSs satisfying the above requirements is presented Chapter 3. To complete the 4 steps illustrated Figure 2.1, we should then define the models built on FDSs (step 2), choose appropriate assessment technique (step 3) and determine the accessible results (step 4). These will be presented respectively Chapter 4 and Chapter 5.

2.3 Illustrative case study: a safety instrumented system (SIS)

2.3.1 System description

Safety instrumented systems (SISs) are designed to keep an equipment under control in a safe state when some abnormal conditions occur. As illustrative use case, the TA4 system of ISO/TR 12489 (ISO12489 2013) is pictured Figure 2.2.

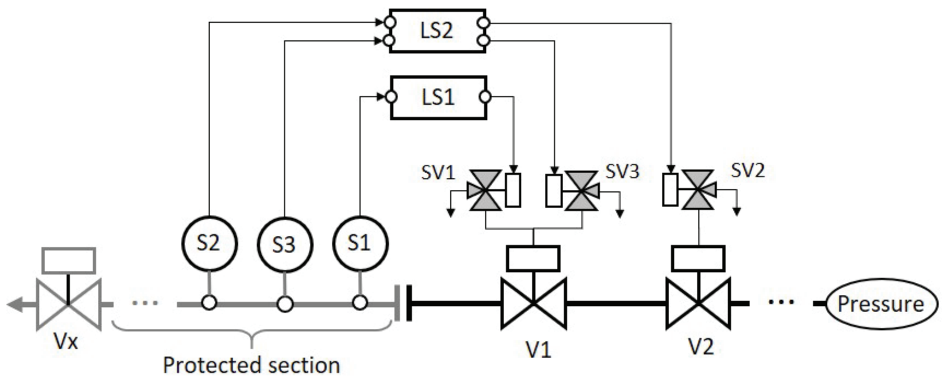


Figure 2.2: Safety instrumented system in TA4 of ISO/TR 12489

This system is designed to protect a pipe section from the overpressure that may damage the equipment located downstream. It works on demand, i.e. the system is activated when an overpressure occurs in the pipe (due to the irregular flow of oil, gas and water extracted from wells).

The system is made of three types of elements: sensors (S1, S2 and S3) in charge of detecting overpressure, logic solvers (LS1 and LS2) in charge of making the decision and the two isolation valves V1 and V2 which can be closed to release the pressure. The logic solver LS2 works according to a 1-out-of-2 logic, i.e. that it sends the command to close the valves if at least one out of two sensors S2 and S3 detects an overpressure.

2.3.2 Failure modes

According to the standard IEC 61508 (IEC61508 2010), the failure modes of the components of a safety instrumented system can be classified along two directions: *safe* versus *dangerous* failure modes and *detected* versus *undetected* failure modes. These two directions can also be combined. According to what precedes, we shall consider three failure modes: safe-failure (F_s), dangerous-detected-failure (F_{dd}) and dangerous-undetected-failure (F_{du}).

The definitions of failure modes can be found in IEC 61508 Part 4. Particularly, we quote the definitions of safe-failure and dangerous-failure below.

– **Safe-failure**: a failure of an element and/or subsystem and/or system that plays a part in implementing the safety function that a) results in the spurious operation of the safety function to put the EUC (or part thereof) into a safe state or maintain a safe state; or b) increases the probability of the spurious operation of the safety function to put the EUC (or part thereof) into a safe state or maintain a safe state.

– **Dangerous-failure**: a failure of an element and/or subsystem and/or system that plays a part in implementing the safety function that a) prevents a safety function from operating when required (demand mode) or causes a safety function to fail (continuous mode) such that the EUC is put into a hazardous or potentially hazardous state; or b) decreases the probability that the safety function operates correctly when required.

In this case study, we should clarify that the safe failures are those who contribute to closing the isolation valves, even though there is no overpressure (i.e. spurious triggers). Reversely, the dangerous failures are those who contribute to keeping the isolation valves open, even though there is an overpressure.

2.3.3 Assumptions and parameters

ISO/TR 12489 makes the additional following assumptions.

– The three solenoid valves (SV1, SV2 and SV3) are perfectly reliable so that they are not considered in the analysis. All other components may fail (independently). Their probabilities of failure follow negative exponential distributions. The parameters of these distributions are given Table. 2.2.

- F_s is always detected.
- Logic solvers embed autotest facilities so that their dangerous failures are immediately detected. On the contrary, dangerous failures of valves remain undetected between two maintenance interventions. Dangerous failures of sensors may be detected or not.
- The system is maintained once a year (once in 8760 hours). The production is stopped during the maintenance. Components are as good as new after the maintenance.

Table 2.2: TA4 Reliability parameters

Failure rates	Sensor	Logic solver	Isolation valve
λ_{F_s}	$3.0 \times 10^{-5}\, h^{-1}$	$3.0 \times 10^{-5}\, h^{-1}$	2.9×10^{-4}
$\lambda_{F_{dd}}$	$3.0 \times 10^{-5}\, h^{-1}$	$6.0 \times 10^{-7}\, h^{-1}$	NA
$\lambda_{F_{du}}$	$3.0 \times 10^{-7}\, h^{-1}$	NA	$2.9 \times 10^{-6}\, h^{-1}$

2.3.4 Failure mechanism and modeling questions

In ISO/TR 12489, this system is modeled by fault trees. Since fault trees are Boolean models, safe failures and dangerous failures can only be analyzed separately, i.e. one fault tree for F_s and another fault tree for F_{dd} and F_{du}.

To the best of the author's knowledge, the combinations of F_s with F_{dd} and F_{du} have not been formally analyzed neither in the standard nor in other previous work. However, such combinations may happen in reality, e.g. a sensor rises a false alarm (i.e. F_s) while the relevant valve is failed to close (i.e. F_{du}). In such case, the effect of the false alarm seems to be neutralized by the failure of the valve. But such scenario is often ignored because its probability — the probability of having two failures simultaneously — is much lower than the probability of having one failure.

Here, our first question is that:
"Does the importance of a scenario depend only on its occurrence probability?"

The answer is obvious no. The importance of a scenario should depend not only on its occurrence probability but also on its criticality.

Take the fault tree analysis as example. The minimal cutsets of a fault tree are the critical scenarios, representing the minimal combinations of components' failure that can lead to the system's failure. In probabilistic aspect, minimal cutsets are also the scenarios whose occurrence probabilities are significant if the components in the system are reliable (i.e. the probability of working is much greater than its probability of failure.). However, the criticality of minimal cutsets is not determined by their occurrence probabilities. Instead, it is defined by the logic function represented by the fault tree. In other words, we can still calculate minimal cutsets and use them to analyze the qualitative behavior of the system even though the probabilities of basic events are not provided.

To be more generic, the combinatorial models in PRA/PSA always have two perspectives: the logical perspective and the probabilistic perspective. Regarding these two perspectives, we shall make a distinction between "critical scenarios" and "significant scenarios". The former refer to those scenarios that are critical to characterize the change of behavior of the system. The latter refer to those scenarios whose occurrence probabilities are significant comparing to the others.

However, there isn't a direct relation between critical scenarios and significant scenarios, i.e. a scenario is critical doesn't mean that it has a high occurrence probability, and vice versa. Ignoring non-significant scenarios before modeling may leave out critical scenarios that are important in determining the system's safety/risk performance.

As stated in the beginning of this subsection, the simultaneous occurrence of F_s with F_{dd} and F_{du} is ignored in the fault tree models of this safety instrumented system. It means that we can never obtain the cutsets containing those combinations of failures, while these cutsets may be helpful in real-time analysis especially when some of the failures have already happened.

Now, the second question is that:
"What are the difficulties in modeling such combination of failures using existing modeling approaches?"

In Section 2.2.3, we have discussed the advantages and deficiencies of the five categories of models. In the following part, we will show concretely their modeling difficulties when applied to this safety instrumented system.

First, if we want to calculate the minimal cut/path sets containing those combinations of failures, only the models in category (1) and category (2) can be used according to Table 2.1. However, the models in category (1) seem to be inappropriate because we cannot linearly order the four states W, F_s, F_{dd} and F_{du} in the state space of components in this safety instrumented system. The models in category (2) seem also to be inappropriate because we cannot use only the infimum and the supremum to describe the failure logic of this system.

If we compromise that the calculation of minimal cut/path sets is not necessary, then the models in categories (3), (4) and (5) are theoretically applicable. However, they still have practical difficulties. For the extended fault trees in category (3), if each non-working state is modeled by an individual basic event, then in this case, there are totally $3^3 \times 2^4 = 432$ basic events (i.e. 3 states for each sensor and 2 states for each logic solver and valve). The construction of a fault tree with 432 basic events is a time-consuming and error-prone work. The MMDD techniques in category (4) will meet the same problem if they also use multistate fault trees to define the structure function of the system. As for category (5), the UGF technique requires to use real numbers to represent the states of multistate component/system, while states of components in this system are symbolic, i.e. W, F_s, F_{dd} and F_{du}.

These modeling difficulties motivate us to propose the framework of finite degradation structures, which is supposed to become a unified framework of combinatorial models in PRA/PSA. In the following chapters, we will provide a detail explanation of the modeling and the analysis of this safety instrumented system under the framework of finite degradation structures.

Part II

Theoretical Development

Chapter 3

Algebraic Framework of Finite Degradation Structures

This chapter defines formally the algebraic framework of finite degradation structures (FDSs). Theoretically, an **algebra** is a pair $\langle A, \mathcal{O} \rangle$ comprised by a set A with a collection $\mathcal{O}$ of finitary operations on A. The elements in A are indivisible objects (or symbols) and the operations in $\mathcal{O}$ define the relations of the elements in A. Given an algebra $\langle A, \mathcal{O} \rangle$, *models* can be built on it, i.e. as interpretations satisfying the rules in $\langle A, \mathcal{O} \rangle$.

The algebraic framework of Boolean combinatorial models, e.g. fault trees and reliability block diagrams, is the Boolean algebra, which can be denoted by the quadruple $\langle \mathbb{B}, \vee, \wedge, \neg \rangle$, where $\mathbb{B} = \{0, 1\}$ and $\vee, \wedge, \neg$ are the logic operations—conjunction, disjunction and negation—defined on $\mathbb{B}$. The algebraic framework of FDSs is denoted by the triple $\langle \mathbf{FDS}, \otimes, \Phi \rangle$, where $\mathbf{FDS}$ stands for the set of FDSs and $\otimes, \Phi$ are the two main operations — the monoidal product and the abstractions — defined on $\mathbf{FDS}$. Mathematically, the set $\mathbf{FDS}$ is closed under these two operations $\otimes$ and Φ. In this chapter, we will define all the mathematical concepts in the framework $\langle \mathbf{FDS}, \otimes, \Phi \rangle$. The models built over $\langle \mathbf{FDS}, \otimes, \Phi \rangle$ will be presented Chapter 4.

The reminder of this chapter is as follows. Section 3.1 reviews the concepts related to partially ordered sets. Section 3.2 defines FDSs and provides a set of examples. Section 3.3 defines the operations on FDSs.

3.1 Partially ordered sets

Definition 3.1.1 (Poset)**.** Let D be a set and $\sqsubseteq$ be an order over D. The pair $\langle D, \sqsubseteq\rangle$ is a *partially ordered set* (poset) if the following axioms hold, i.e. $\forall a, b, c \in D$:

– $a \sqsubseteq a$ (*Reflexivity*);

– if $a \sqsubseteq b$ and $b \sqsubseteq c$, then $a \sqsubseteq c$ (*Transitivity*);

– if $a \sqsubseteq b$ and $b \sqsubseteq a$, then $a = b$ (*Antisymmetry*).

The order $\sqsubseteq$ satisfying the above axioms is called a *partial order*.

For any two elements a, b in D, if either $a \sqsubseteq b$ or $b \sqsubseteq a$ holds, a and b are *comparable*. Otherwise, a and b are *incomparable*, denoted by $a \sim b$. Particularly, if every pair of elements a, b in D is comparable, the poset $\langle D, \sqsubseteq\rangle$ is a *totally ordered set* and $\sqsubseteq$ is a *total* order or *linear* order.

A poset can be visualized through its **Hasse diagram**. In such diagram, the elements of the poset are represented as vertices and each order relation $x \sqsubset y$ is drawn as a line segment that goes *upward* from x to y. An example is given Figure 3.1. In this example, the partial orders are defined by the inclusion of sets, i.e. $\forall X, Y \subseteq \{x, y, z\}, X \sqsubseteq Y \Leftrightarrow X \subseteq Y$.

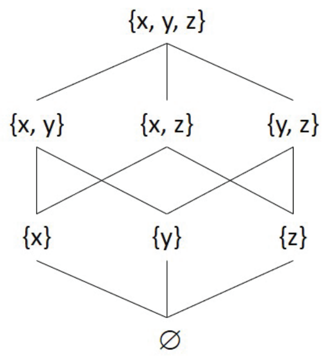


Figure 3.1: Hasse diagram example.

Definition 3.1.2 (Extrema)**.** An element $\top$ in $\langle D, \sqsubseteq\rangle$ is a *greatest* element if $\forall a \in D, a \sqsubseteq \top$. An element $\perp$ in $\langle D, \sqsubseteq\rangle$ is a *least* element if $\forall a \in D, \perp \sqsubseteq a$.

For instance, the greatest element of the poset in Figure 3.1 is $\{x, y, z\}$ and its least element is $\emptyset$.

It is easy to verify that a poset with finite elements can have at most one greatest or least element. Assume for a contradiction that D has two least elements $\bot_1$ and $\bot_2$, then we have both $\bot_1 \sqsubseteq \bot_2$ and $\bot_2 \sqsubseteq \bot_1$, which by antisymmetry means that $\bot_1 = \bot_2$. The proof of the uniqueness of the greatest elements can be done in a same way.

Definition 3.1.3 (Maximal and minimal elements)**.** The set of maximal elements and the set of minimal elements of a poset $\mathcal{L}$, denoted respectively by $\max(\mathcal{L})$ and $\min(\mathcal{L})$, are defined as follows:

$$\begin{aligned} \max(\mathcal{L}) &\overset{def}{=} \{g \in \mathcal{L} | \nexists x \in \mathcal{L}, g \sqsubset x\} \\ \min(\mathcal{L}) &\overset{def}{=} \{l \in \mathcal{L} | \nexists x \in \mathcal{L}, x \sqsubset l\} \end{aligned} \tag{3.1}$$

Take the poset in Figure 3.2 as an example. In this case, the greatest element and the least element are removed. According to the above definition, the maximal elements are $\{x, y\}, \{x, z\}, \{y, z\}$ and the minimal elements are $\{x\}, \{y\}, \{z\}$. These maximal and minimal elements characterize not the extreme of a poset but the upper and lower bounds of a poset.

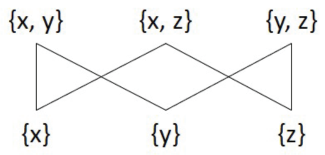


Figure 3.2: Poset without greatest and least element.

It is worth mentioning here that the notion of maximal and minimal elements is the key to generalize the notion of minimal cut/path sets from fault trees to multistate models.

Definition 3.1.4 (Semi-lattice)**.** A *meet-semi-lattice*, denoted by $\langle D, \sqsubseteq, \bot\rangle$, is a poset $\langle D, \sqsubseteq\rangle$ that has a least element $\bot \in D$. A *join-semi-lattice*, denoted by $\langle D, \sqsubseteq, \top\rangle$, is a poset $\langle D, \sqsubseteq\rangle$ that has a greatest element $\top \in D$.

Typically, if a poset has both the greatest element and the least element, it is called a *lattice*. For instance, the poset in Figure 3.1 is a lattice. If we remove one of its extreme elements, we obtain the semi-lattices, e.g. Figure 3.3 (a) is a join-semi-lattice and (b) is a meet-semi-lattice.

In abstract algebra, the Boolean algebra is a complemented distributive lattice, which captures in essence properties of both set operations and logic operations.

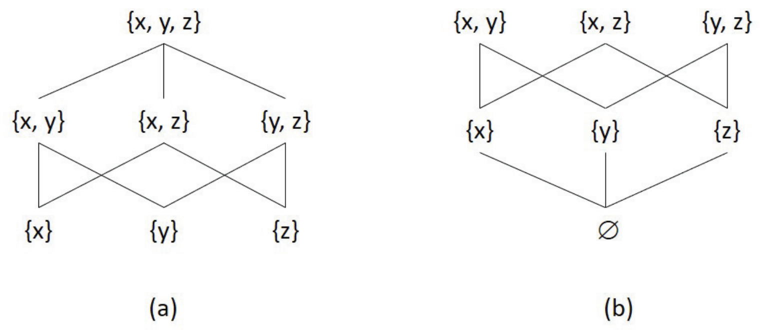


Figure 3.3: Example of (a) join-semi-lattices and (b) meet-semi-lattice.

As pictured Figure 3.4, the finite degradation structures (FDSs) proposed this thesis are meet-semi-lattices, which have a larger coverage than the set of lattices that lays the foundation of Boolean algebra. For this reason, FDSs can be seen as a more generalized algebraic framework compared to the Boolean algebra and the binary assumption is thus eliminated in FDSs. Consequently, the logic operations that are used to model the reliability and safety behavior of Boolean systems can be natually extended in multistate cases using FDSs.

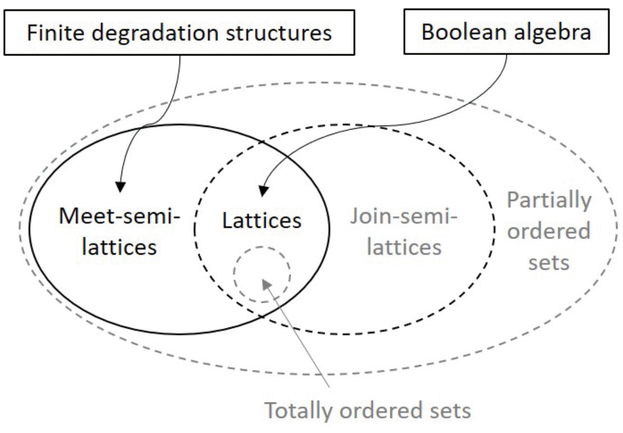


Figure 3.4: Classification of posets.

3.2 Finite degradation structures (FDSs)

3.2.1 Definitions

Definition 3.2.1 (FDS)**.** A *finite degradation structure* (FDS) is defined as a quadruple $\langle D, \sqsubseteq, \perp, p \rangle$, where D is a finite set, $\langle D, \sqsubseteq, \perp \rangle$ forms a meet-semi-lattice and $p : D \to [0, 1]$ is a probability measure on D satisfying that $\sum_{s \in D} p(s) = 1$.

Practically, a FDS is an atomic model of an individual object of the system under study. This individual object can be either a component, a subsystem (i.e. a group of components) or the system itself as a whole. Let $\mathcal{L} : \langle D, \sqsubseteq, \perp, p \rangle$ be the model of an object $\mathcal{A}$, then the interpretation of $\mathcal{L}$ is explained as follows:

- The elements in D represent the **states** of $\mathcal{A}$, which discretize the performance that we want to model from $\mathcal{A}$. These states are named symbolically, e.g. Working, Degraded, Failed, 0.5, 1, 4.7, x, y, etc. To define the probability measure p on D, these states should be stochastically independent (i.e. mutually exclusive) and complete (i.e. collectively exhaustive).
- The partial order $\sqsubseteq$ is interpreted as the **degradation order**, which scales the degradation level between each pair of states. The relation $x \sqsubset y$ is interpreted as "x is less degraded than y". The degradation order is a partial order for allowing having incomparable states. In reliability and safety models, states are not always comparable because they may correspond to different types of degradation, e.g. safe failure and dangerous failure, and it is not necessary or not even possible to order their degradation levels.
- The least element $\perp$ is the **least degraded state** of $\mathcal{A}$. In other words, it represents the (intact) ideal operating state in which the $\mathcal{A}$ is as good as new. The intuition behind this definition is that the state of a component cannot be less degraded than when it is new. This uniqueness of the least degraded state $\perp$ makes the poset D be a meet-semi-lattice.
- The probability measure p is embedded in $\mathcal{L}$ to perform the **probabilistic calculations**. These calculations will be presented Chapter 5. Moreover, p can also be time-variant, i.e. $p : D \times \mathbb{R}^+ \to [0, 1]$, where $\forall s \in D$, $p(s, t)$ represents the probability of being in state s at time t.

In the name of FDS, "finite" indicates that D is a finite set; "degradation" means that the partial order $\sqsubseteq$ is interpreted as the degradation order; and "structure" emphasizes that D is not a simple set but a set that embeds an order structure.

3.2.2 Typical examples

Five typical examples of FDSs are defined Table 3.1. Their Hasse diagrams are given Figure 3.5.

Table 3.1: Typical FDSs applied in reliability and safety analyses.

Name	States	Orders	Bottom ($\perp$)
WF	$\{W, F\}$	$W \sqsubset F$	W
WDF	$\{W, D, F\}$	$W \sqsubset D \sqsubset F$	W
SWF	$\{S, W, F\}$	$S \sqsubset W \sqsubset F$	S
WFdFu	$\{W, F_d, F_u\}$	$W \sqsubset F_d, W \sqsubset F_u$	W
WFdFs	$\{W, F_{dang}, F_{safe}\}$	$W \sqsubset F_{dang}, W \sqsubset F_{safe}$	W

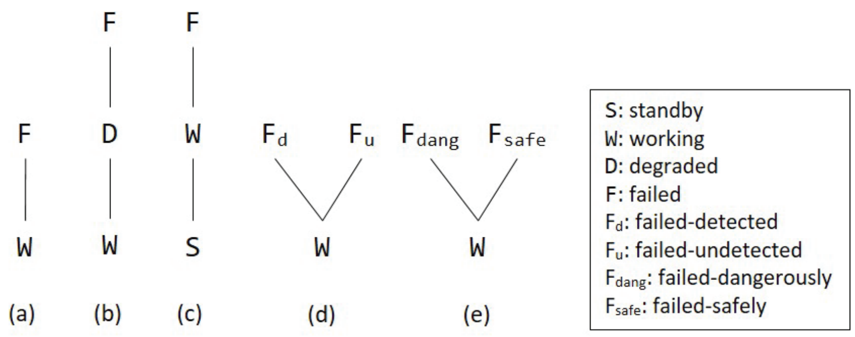


Figure 3.5: Hasse diagram of **WF**, **WDF**, **SWF**, **WFdFu** and **WFdFs**.

- **WF** is a binary FDS with the two classical states: working and failed. The degradation order in this FDS is obviously $W \sqsubset F$. This ordering is consistent with the ordering $0 < 1$ in Boolean algebra. Therefore, the systems that can be modeled by fault trees (or reliability block diagrams) can be modeled in the framework of FDSs using **WF**.

- **WDF** is a ternary FDS with an intermediate degraded state D between W and F. In this case, the degradation order is the linear order $W \sqsubset D \sqsubset F$.

- **SWF** is also a ternary FDS with an additional standby state S for standby components. In this FDS, the standby state S is the least degraded state because we may consider that the component cannot fail in standby state but can fail in working state.

- **WFdFu** and **WFdFs** are the ternary FDSs where there are two failure modes, i.e. the detected/undetected failures and the safe/dangerous failures.

In algebraic point of view, we can see that the degradation orders in $\mathbf{WDF}$ and $\mathbf{SWF}$ and the degradation orders in $\mathbf{WFdFu}$ and $\mathbf{WFdFs}$ are similar. In mathematics, this similarity is called *order isomorphic*.

Definition 3.2.2 (Equal up to isomorphism)**.** Two posets $\langle D_1, \sqsubseteq\rangle$ and $\langle D_2, \sqsubseteq\rangle$ are *equal up to isomorphism* (or *isomorphic*), denoted by $\langle D_1, \sqsubseteq\rangle \cong \langle D_2, \sqsubseteq\rangle$, if there exist an isomorphism $\alpha : D_1 \to D_2$ such that $\forall x, y \in D_1, x \sqsubseteq y \Leftrightarrow \alpha(x) \sqsubseteq \alpha(y)$.

Whenever two posets are order isomorphic, they can be considered to be "essentially the same" in the sense that one of the orders can be obtained from the other just by renaming the elements.

In this sense, we generalize the above five FDSs into two types of FDSs: $\mathbf{n}$ and $\mathbf{WnF}$.

Definition 3.2.3 ($\mathbf{n}$)**.** We denote the linearly ordered FDSs by $\mathbf{n}$, where $n \in \mathbb{N}^+$ stands for the number of states inside.

Definition 3.2.4 ($\mathbf{WnF}$)**.** We denote the FDS that has one least state W and $n(n \geq 1)$ maximal states $F_1, ..., F_n$ with the degradation orders $W \sqsubset F_i, \forall i = 1, ..., n$ and $F_i \sim F_j, i \neq j$ by $\mathbf{WnF}$.

The Hasse diagrams of $\mathbf{n}$ and $\mathbf{WnF}$ are pictured Figure 3.6.

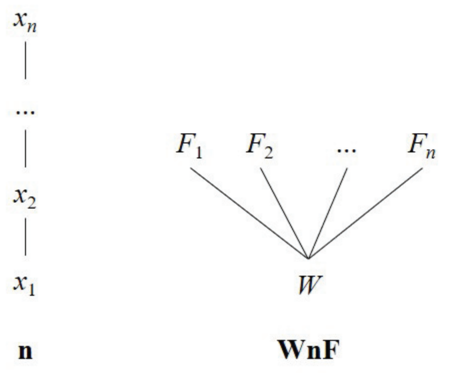


Figure 3.6: The structure of $\mathbf{n}$ and $\mathbf{WnF}$.

Practically, $\mathbf{n}$ is used to model the component that experiences a linear degradation and $\mathbf{WnF}$ is used to model the component that has one working state and n incomparable exclusive failure modes.

Accordingly, we have the following order isomorphisms:

$$\mathbf{WF} \cong \mathbf{2}$$

$$\mathbf{WDF} \cong \mathbf{SWF} \cong \mathbf{3}$$

$$\mathbf{WFdFu} \cong \mathbf{WFdFs} \cong \mathbf{W2F}$$

The order isomorphism is also used to define the commutativity, associativity and distributivity of the operations on FDSs. This will be presented Section 3.3.

Instead of the two types of FDSs $\mathbf{n}$ and $\mathbf{WnF}$, analysts can also design other appropriate FDSs for specific systems. In the following section, we will show a specific FDS that is used to model the safety instrumented system presented Section 2.3.

3.2.3 The FDS for SIS

For the safety instrumented system presented Section 2.3, we specifically design a FDS, named as **SIS**, to model the state space of the components (i.e. sensors, logic solvers and valves) in this system.

The Hasse diagram of **SIS** is pictured Figure 3.7.

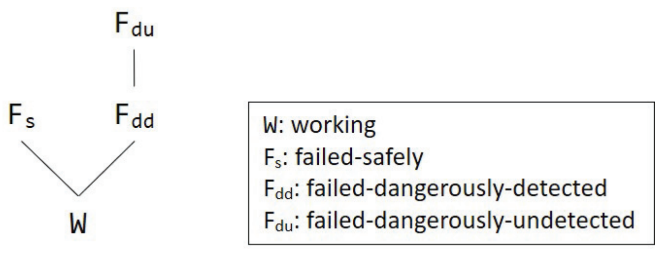

F_du
F_s
F_dd
W
W: working
F_s: failed-safely
F_dd: failed-dangerously-detected
F_du: failed-dangerously-undetected

Figure 3.7: The FDS **SIS**.

There are four states in **SIS**: a working state W, a failed-safely state F_s, a failed-dangerously-detected state F_{dd} and a failed-dangerously-undetected state F_{du}. The degradation orders in **SIS** are explained as follows:

- $W \sqsubset F_s$ and $W \sqsubset F_{dd}$, since the working state is always less degraded than a failed state.
- $F_{dd} \sqsubset F_{du}$, since F_{du} is not revealed when it occurs so that the system has a high potential to be in a more hazardous situation, while F_{dd} can be noticed

so that mitigating reactions can be carried out before resulting in hazardous consequences.

– $F_s \sim F_{dd}$ and $F_s \sim F_{du}$, since the risk represented by safe failures and dangerous failures, both in terms of frequency of occurrence and severity of consequences, are very different. On the one hand, spurious triggers of SIS have a strong economic impact, but indeed no impact on safety. On the other hand, dangerous failures have an impact on safety. If they remain undetected, they may lead to a catastrophic accident. Therefore, safe failures and dangerous failures are considered in this case to be incomparable.

According to the assumptions made in ISO/TR 12489, the probability of failure of each component follows negative exponential distribution. Denote the failure rate of F_s, F_{dd} and F_{du} by λ_{F_s}, $\lambda_{F_{dd}}$ and $\lambda_{F_{du}}$. Then, the probability measure in the FDS $\mathbf{SIS}$ of each component can be defined analytically as follows:

$$\begin{cases} p(W,t) &= e^{-\lambda_W t} \\ p(F_{du},t) &= \frac{\lambda_{F_{du}}}{\lambda_W}(1-e^{-\lambda_W t}) \\ p(F_{dd},t) &= \frac{\lambda_{F_{dd}}}{\lambda_W}(1-e^{-\lambda_W t}) \\ p(F_s,t) &= \frac{\lambda_{F_s}}{\lambda_W}(1-e^{-\lambda_W t}) \end{cases} \tag{3.2}$$

where $\lambda_W = \lambda_{F_{du}} + \lambda_{F_{dd}} + \lambda_{F_s}$.

According to the parameters in Table 2.2, we calculate the probability measure for each type of components. The results are pictured Figure 3.8. Particularly, since the dangerous failure of logic solvers (LS) are always detected and the dangerous failure of valves (V) are always undetected, then $p_{LS}(F_{du},t) = 0$ and $p_V(F_{dd},t) = 0, \forall t \geq 0$.

The probability measures of components (i.e. at bottom-level of the model) are treated as inputs of the model. Similar to fault tree analysis, these probabilities can propagate to higher-level objects through the well-defined logic operations.

3.3 Operations on FDSs

Denote the set of FDSs by $\mathbf{FDS}$. Then, $\mathbf{FDS}$ is closed under two (collections of) operations: the monoidal product $\otimes$ and the abstractions in Φ. These two operations together form the algebraic framework $\langle \mathbf{FDS}, \otimes, \Phi \rangle$.

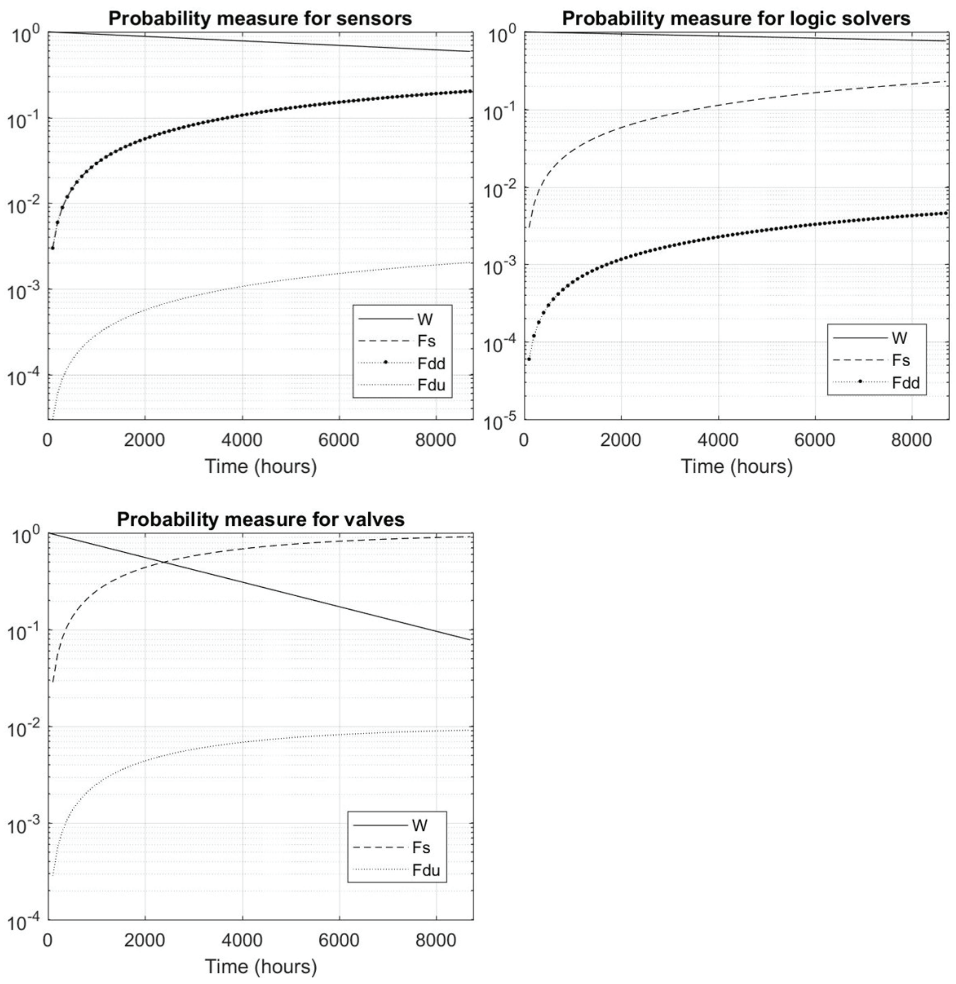


Figure 3.8: The probability measures for sensors, logic solvers and valves.

3.3.1 Monoidal product

Definition 3.3.1 (Product)**.** The *monoidal product* (or product for short) on FDSs is defined as the bifunctor $\otimes : \mathbf{FDS} \times \mathbf{FDS} \to \mathbf{FDS}$ such that for any two FDSs $\mathcal{L}_1 : \langle D_1, \sqsubseteq, \perp_1, p_1 \rangle, \mathcal{L}_2 : \langle D_2, \sqsubseteq, \perp_2, p_2 \rangle$,

$$\mathcal{L}_1 \otimes \mathcal{L}_2 \overset{def}{=} \langle D_\otimes, \sqsubseteq, \perp_\otimes, p_\otimes \rangle$$

where:

– $D_\otimes = D_1 \times D_2$.

- $\forall (x_1, x_2), (y_1, y_2) \in D_\otimes, (x_1, x_2) \sqsubseteq (y_1, y_2) \Leftrightarrow x_1 \sqsubseteq y_1 \wedge x_2 \sqsubseteq y_2$.
- $\bot_\otimes = (\bot_1, \bot_2)$.
- $\forall (x, y) \in D_\otimes, p_\otimes(x, y) = p_1(x) \cdot p_2(y)$, if p_1, p_2 are independent.

It is easy to prove that $\mathcal{L}_1 \otimes \mathcal{L}_2$ is also a FDS. First, $D_\otimes$ is a finite set since D_1 and D_2 are finite. Then, $\sqsubseteq$ is a partial order on $D_\otimes$ since it satisfies the axioms in Definition 3.1.1. $\bot_\otimes$ is the least element in $D_\otimes$ since $\forall x_1 \in D_1, \forall x_2 \in D_2, \bot_1 \sqsubseteq x_2, \bot_2 \sqsubseteq x_2 \Leftrightarrow (\bot_1, \bot_2) = \bot_\otimes \sqsubseteq (x_1, x_2) \in D_\otimes$. Finally, $p_\otimes$ is a probability measure on $D_\otimes$ since $\sum_{(x,y) \in D_\otimes} p_\otimes(x, y) = (\sum_{x \in D_1} p_1(x)) \cdot (\sum_{y \in D_2} p_2(y)) = 1$.

Therefore, $\mathbf{FDS}$ is closed under the monoidal product $\otimes$.

Figure 3.9 (a) - (d) illustrate respectively the Hasse diagram of the product $\mathbf{WF}^2$, $\mathbf{WF} \otimes \mathbf{WDF}$, $\mathbf{WDF} \otimes \mathbf{WF}$ and $\mathbf{WDF}^2$, where $\mathcal{L}^n$ stands for the product of n $\mathcal{L}$.

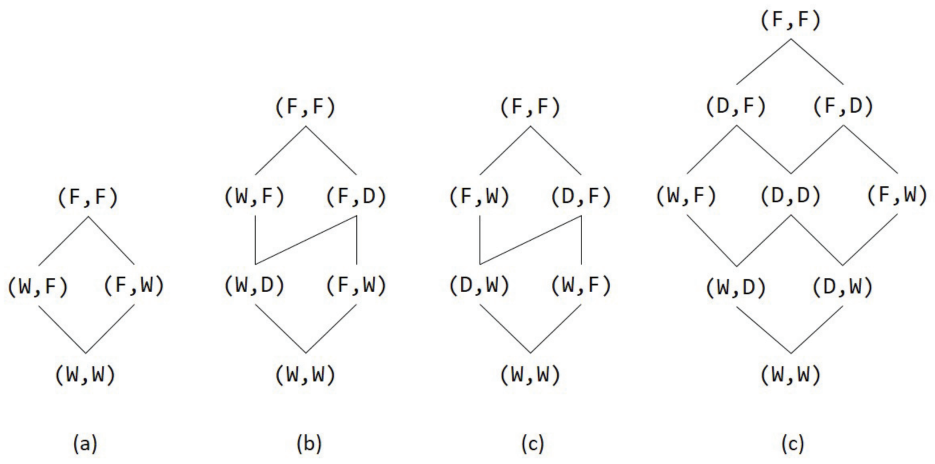


Figure 3.9: $\mathbf{WF}^2$, $\mathbf{WF} \otimes \mathbf{WDF}$, $\mathbf{WDF} \otimes \mathbf{WF}$ and $\mathbf{WDF}^2$.

Proposition 3.3.1 (Properties)**.** $\forall \mathcal{A}, \mathcal{B}, \mathcal{C} \in \mathbf{FDS}$, the following equalities hold:

- $\mathcal{A} \otimes \mathcal{B} \cong \mathcal{B} \otimes \mathcal{A}$ (Commutativity)
- $\mathcal{A} \otimes (\mathcal{B} \otimes \mathcal{C}) \cong (\mathcal{A} \otimes \mathcal{B}) \otimes \mathcal{C}$ (Associativity)
- $\mathbf{1} \otimes \mathcal{A} \cong \mathcal{A} \otimes \mathbf{1} \cong \mathcal{A}$ (Identity), where $\mathbf{1} : \langle \{\bot\}, \sqsubseteq, \bot, p \rangle$ is the unary FDS that contains only one element $\bot$ with the order $\bot \sqsubseteq \bot$ and $p(\bot) = 1$.

The monoidal product $\otimes$ on FDSs can be seen as the analogue of the Cartesian product on finite sets. The existence of such monoidal product $\otimes$ makes it possible to define — both algebraically and stochastically — the combination of states of components in combinatorial models.

3.3.2 Abstractions

Definition 3.3.2 (Abstraction)**.** Let $\mathcal{S} : \langle D_S, \sqsubseteq, \bot_S, p_S \rangle, \mathcal{T} : \langle D_T, \sqsubseteq, \bot_T, p_T \rangle$ be two FDSs. An *abstraction* from $\mathcal{S}$ to $\mathcal{T}$ is defined as a surjective mapping $\varphi : \mathcal{S} \twoheadrightarrow \mathcal{T}$ such that:

- $\forall y \in \mathcal{T}, \exists x \in \mathcal{S}, \varphi(x) = y$.
- $\varphi(\bot_S) = \bot_T$.
- $\forall y \in \mathcal{T}, p_T(y) = \sum_{x \in \varphi^{-1}[y]} p_S(x)$.

$\mathcal{S}$ is called the *codomain* of φ, denoted by codom(φ), and $\mathcal{T}$ is called the *domain* of φ, denoted by dom(φ). The symbol $\twoheadrightarrow$ is used for abstractions.

Figure 3.10 gives two examples of the abstractions. The abstraction in (a) abstracts the two degraded states (type 1 and 2) into one degraded state. The abstraction in (b) abstracts the Working state and the Standby state into the Normal state and abstracts the two failed states (type 1 and 2) into the Abnormal state.

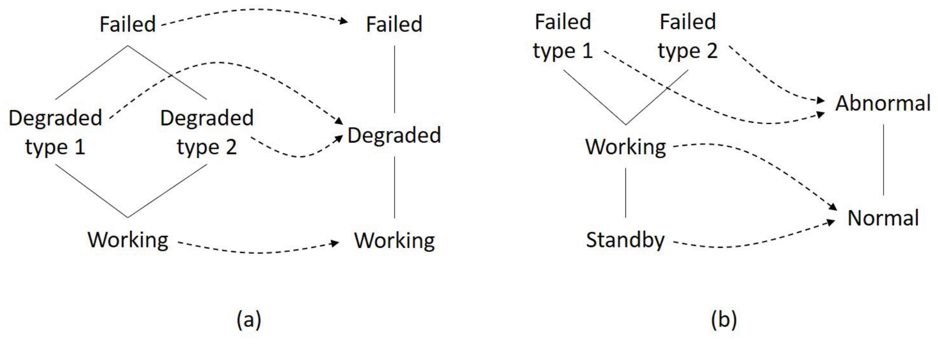


Figure 3.10: Examples of abstraction.

From reliability and safety point of view, these two abstractions are reasonable because they abstract the state space from a more complicated structure into a simpler one while keeping the degradation orders coherent.

Definition 3.3.3 (Weak-coherency)**.** An abstraction $\varphi : \mathcal{S} \twoheadrightarrow \mathcal{T}$ is *weakly-coherent* (or coherent for short) if $\forall x, y \in \mathcal{S}, x \sqsubseteq y \Rightarrow \varphi(x) \sqsubseteq \varphi(y)$.

We can verify that the two abstractions in Figure 3.10 are both weakly-coherent.

If we modify the two abstractions in Figure 3.10 into the ones in Figure 3.11, then according to Definition 3.3.3, the abstraction in Figure 3.11 (a) is still coherent, but the abstraction in Figure 3.11 (b) is not. For the latter, some degradation orders in the codomain of the abstraction are reversed.

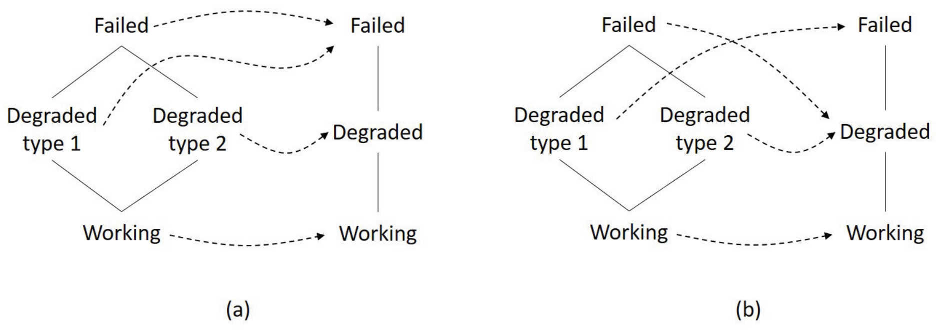


Figure 3.11: Examples of disordered abstraction.

Now, consider the abstraction $\varphi : \mathbf{W2F} \twoheadrightarrow \mathbf{3}$ pictured Figure 3.12. According to Definition 3.3.3, φ is coherent, but the degradation orders are not strictly preserved, i.e. the states y and z are incomparable (i.e. $y \sim z$) in dom(φ) while their images are comparable (i.e. $\varphi(z) \sqsubset \varphi(y)$) in codom$(\varphi)$.

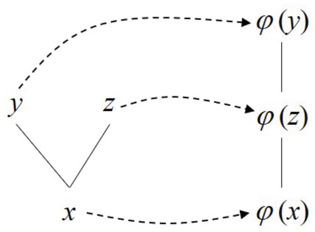


Figure 3.12: A coherent but not strictly order-preserving abstraction.

Therefore, to judge the order-preserving property of incomparable states, we introduce the notion of strong-coherency, which is mathematically defined below.

Definition 3.3.4 (Strong-coherency)**.** An abstraction $\varphi : \mathcal{S} \twoheadrightarrow \mathcal{T}$ is *strongly-coherent* if it is coherent and $\forall x, y \in \mathcal{S}$ such that $x \sim y$, one of the following conditions holds:

- $\varphi(x) \sim \varphi(y)$;
- $\varphi(x) = \varphi(y)$;
- $\varphi(x) \sqsubseteq \varphi(y)$ if $\exists y' \in \mathcal{S}, x \sqsubseteq y' \wedge \varphi(y) = \varphi(y')$ or $\exists x' \in \mathcal{S}, x' \sqsubseteq y \wedge \varphi(x) = \varphi(x')$.

As comparison, three coherent abstractions are pictured Figure 3.13, where φ_1 and φ_2 are strongly-coherent and φ_3 is weakly-coherent.

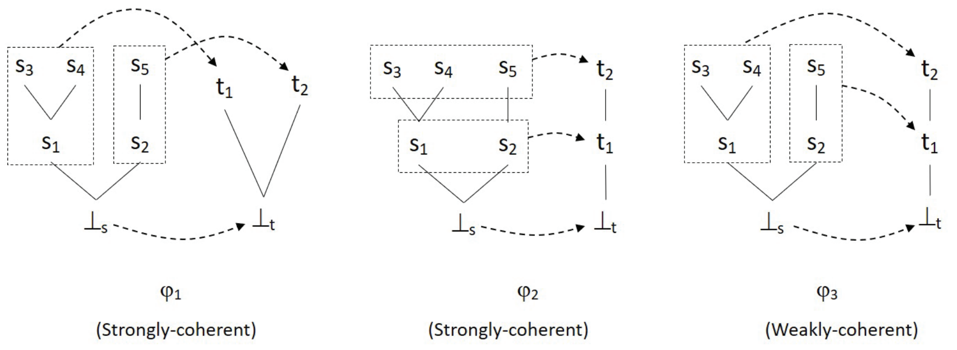


Figure 3.13: Strongly-coherent abstractions φ_1, φ_2 and weakly-coherent abstraction φ_3.

The set of all abstractions between FDSs is denoted by $\Phi : \mathbf{FDS} \twoheadrightarrow \mathbf{FDS}$. The set $\mathbf{FDS}$ is closed under the abstractions in Φ.

3.3.3 Operations

Definition 3.3.5 (Operation)**.** An *operation* under the framework $\langle \mathbf{FDS}, \otimes, \Phi \rangle$ is an abstraction $\phi \in \Phi$ in the following form:

$$\phi : \bigotimes_{i=1}^{n} \mathcal{S}_i \twoheadrightarrow \mathcal{T} \tag{3.3}$$

where $n (n \geq 1)$ is the number of inputs called the *arity* of the operation, $\mathcal{T}$ is the codomain of the operation, i.e. $\mathcal{T} = \text{codom}(\phi)$, and $\mathcal{S}_i \in \mathbf{FDS}, i = 1, ..., n$ are domains of the n input arguements of the operation.

The operation ϕ defines the mapping between the state combinations in $\bigotimes_{i=1}^{n} \mathcal{S}_i$ and the states in $\mathcal{T}$.

Denote the probability measure in $\mathcal{S}_i$ by p_i, and assume that p_i's are independent. Then, $\forall (s_1, ..., s_n) \in \bigotimes_{i=1}^{n} \mathcal{S}_i$, the product measure $p_{\otimes}$ is calculated according to

Definition 3.3.1 as follows:

$$p_{\otimes}(s_1, ..., s_n) = \prod_{i=1}^{n} p_i(s_i) \tag{3.4}$$

Then, $\forall y \in \mathcal{T}$, the state probability $p_{\mathcal{T}}(y)$ in $\mathcal{T}$ is calculated according to Definition 3.3.2 as follows:

$$p_{\mathcal{T}}(y) = \sum_{(s_1,...,s_n) \in \phi^{-1}[y]} p_{\otimes}(s_1, ..., s_n) \tag{3.5}$$

where $\phi^{-1}[y]$ stands for the set of preimages of y under ϕ.

Finally, we have:

$$p_{\mathcal{T}}(y) = \sum_{(s_1,...,s_n) \in \phi^{-1}[y]} \left(\prod_{i=1}^{n} p_i(s_i) \right) \tag{3.6}$$

Eq.(3.6) quantifies the propagation of probabilities from $\mathcal{S}_i$ to $\mathcal{T}$ through the operation ϕ.

The operations on FDSs are multi-valued logic functions. They are used to model the interrelationship between components and compose the behavior of components to map the behavior of the system. In $\langle \mathbf{FDS}, \otimes, \Phi \rangle$, the operations are not limited to the Boolean logic conjunction, disjunction and negation. Instead, they can be fully customized according to needs.

Example 3.3.1. Let's take a concrete example. A system S is made of two components A and B. The states of these components are either working or failed. The failure mechanism of S is described as follows:

– If both components are working, then S is working;

– If both components are failed, then S is failed;

– If only one of the two components is failed, then S is degraded.

To model the failure mechanism of S, we can use an operation $\phi : \mathbf{WF}^2 \twoheadrightarrow \mathbf{WDF}$ with the valuation rules defined Table 3.2. The graphical representation of this operation is pictured Figure 3.14.

From Figure 3.14, we can easily verify that ϕ is strong-coherent. Let u, v, w be the variables represent respectively the state of A, B and S. Then, the definition of w is written as $w := \phi(u, v)$.

Table 3.2: The valuation of $\phi : \mathbf{WF}^2 \twoheadrightarrow \mathbf{WDF}$.

$\phi(u,v)$		v	
		W	F
u	W	W	D
	F	D	F

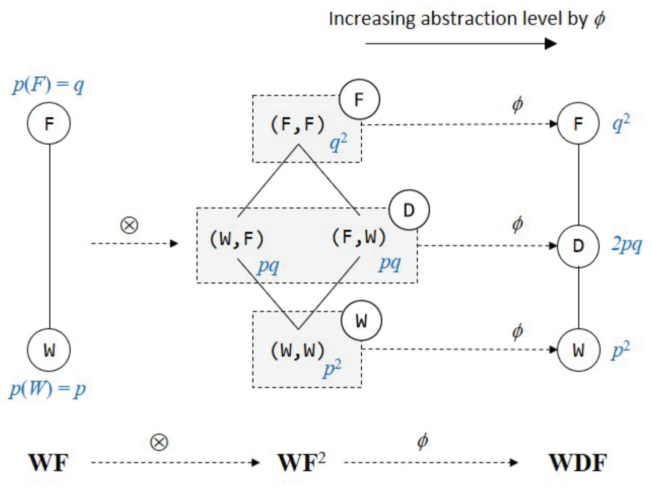


Figure 3.14: Illustration of $\phi : \mathbf{WF}^2 \twoheadrightarrow \mathbf{WDF}$.

For probabilities, assume that for both A and B, the probability that the component is working is p and the probability that the component is failed is q. Then, if A and B are stochastically independent, the product measure $p_{\otimes}$ in $\mathbf{WF}^2$ are calculated as follows:

$$\begin{cases} p_{\otimes}(W,W) &= p_A(W) \cdot p_B(W) = p^2 \\ p_{\otimes}(W,F) &= p_A(W) \cdot p_B(F) = pq \\ p_{\otimes}(F,W) &= p_A(F) \cdot p_B(W) = pq \\ p_{\otimes}(F,F) &= p_A(F) \cdot p_B(F) = q^2 \end{cases} \tag{3.7}$$

As results, the probability measure p_S in $\mathbf{WDF}$ through ϕ is calculated as follows:

$$\begin{cases} p_S(W) &= p_{\otimes}(W,W) = p^2 \\ p_S(D) &= p_{\otimes}(W,F) + p_{\otimes}(F,W) = 2pq \\ p_S(F) &= p_{\otimes}(F,F) = p^2 \end{cases} \tag{3.8}$$

which means that the probability that S is working is p^2, the probability that S is degraded is $2pq$ and the probability that S is failed is q^2.

Proposition 3.3.2 (Composition of operations)**.** Let $\phi : \bigotimes_{i=1}^{n} \mathcal{S}_i \twoheadrightarrow \mathcal{T}$ be an operation. Then, for all $j = 1, ..., n$, if there exists an operation $\varphi_j : \mathcal{A} \twoheadrightarrow \mathcal{S}_i$, then

ϕ and φ_j can be composed to one operation $\phi' : \mathcal{S}_1 \otimes \cdots \otimes \mathcal{S}_{j-1} \otimes \mathcal{A} \otimes \mathcal{S}_{j+1} \otimes \cdots \otimes \mathcal{S}_n \twoheadrightarrow \mathcal{T}$ such that $\forall s_i \in \mathcal{S}_i, i \neq j$ and $\forall x \in \mathcal{A}$, $\phi'(s_1, ..., s_{j-1}, x, s_{j+1}, ..., s_n) = \phi(s_1, ..., s_{j-1}, \varphi_j(x), s_{j+1}, ..., s_n)$.

When system gets large and complex, it is not possible to use only one single operation to model its behavior. As stated Section 2.2.1, the key reason of using combinatorial models is that they allow to decompose large system into independent parts, create submodels of those parts and then compose the submodels following certain rules. Here, these rules refer to the operations on FDSs. The compositionality of operation is the mathematical basis of the compositionality of models built on FDSs.

3.3.4 The operations for SIS

For the safety instrumented system presented Section 2.3, two operations are considered, i.e. the series composition and the parallel composition of components.

The series composition of two components is denoted by $\ogreaterthan : \mathbf{SIS}^2 \twoheadrightarrow \mathbf{SIS}$ and parallel composition of two components is denoted by $\|: \mathbf{SIS}^2 \twoheadrightarrow \mathbf{SIS}$. The block-diagram-like representation of $\ogreaterthan$ and $\|$ is pictured Figure 3.15.

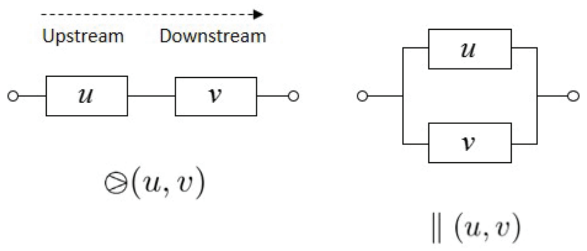


Figure 3.15: The block-diagram-like representations of $\ogreaterthan$ and $\|$.

The valuation of $\ogreaterthan$ and $\|$ is given Table 3.3. We shall explain them one by one as follows.

Table 3.3: Valuation of $\ogreaterthan$ and $\|$.

$\ogreaterthan(u, v)$		v			
		W	F_s	F_{dd}	F_{du}
u	W	W	F_s	F_{dd}	F_{du}
	F_s	F_s	F_s	F_{dd}	F_{dd}
	F_{dd}	F_{dd}	F_s	F_{dd}	F_{dd}
	F_{du}	F_{du}	F_s	F_{dd}	F_{du}

$\| \, (u, v)$		v			
		W	F_s	F_{dd}	F_{du}
u	W	W	F_s	W	W
	F_s	F_s	F_s	F_s	F_s
	F_{dd}	W	F_s	F_{dd}	F_{du}
	F_{du}	W	F_s	F_{du}	F_{du}

For the series composition $\ogreaterthan(u, v)$:

– This operation is not symmetric. u represents the state of the upstream component while v represents the state of the downstream component. The direction is defined as: sensors $\rightarrow$ logic solvers $\rightarrow$ valves, which is the direction of information flow or reaction flow of the safety instrumented system.

– The only case that the output of the operation is working is that both components are working, i.e. $\ogreaterthan(W, W) = W$.

– If the downstream component is working, the output of the operation only depends on the state of the upstream component, i.e. $\forall s \in \mathbf{SIS}$, $\ogreaterthan(s, W) = s$.

– $\forall s \in \mathbf{SIS}$, $\ogreaterthan(s, F_s) = F_s$, because the safe failure (or spurious trip) of the downstream component cannot be corrected or mitigated by the upstream component. Similarly, $\forall s \in \mathbf{SIS}$, $\ogreaterthan(s, F_{dd}) = F_{dd}$, because the dangerous detected failure of the downstream component cannot be corrected or mitigated by the upstream component.

– The reason of $\ogreaterthan(F_s, F_{du}) = F_{dd}$ is twofold. First, since F_s is assumed to be detected, the result of $\ogreaterthan(F_s, F_{du})$ should also be detected. Second, since the dangerous undetected failure of the downstream component cannot be corrected or mitigated by the upstream component, the result of $\ogreaterthan(F_s, F_{du})$ should be a dangerous failure. Therefore, the result of $\ogreaterthan(F_s, F_{du})$ is the dangerous-detected failure F_{dd}. The reason of $\ogreaterthan(F_{dd}, F_{du}) = F_{dd}$ is similar.

For the parallel composition $\parallel (u, v)$:

– $\forall s \in \mathbf{SIS}$, $\ogreaterthan(s, F_s) = F_s$ and $\ogreaterthan(F_s, s) = F_s$, because the safe failure (or spurious trip) of any of the two components in parallel composition cannot be corrected or mitigated by the other component.

– $\forall s \in \mathbf{SIS}$, $s \neq F_s$, $\ogreaterthan(s, W) = W$ and $\ogreaterthan(W, s) = W$, because the parallel composition is considered as a 1-out-of-2 logic, i.e. the result is working if at least one component is working.

– $\ogreaterthan(F_{du}, F_{du}) = \ogreaterthan(F_{du}, F_{dd}) = \ogreaterthan(F_{dd}, F_{du}) = F_{du}$ and $\ogreaterthan(F_{dd}, F_{dd}) = F_{dd}$ because for dangerous failures, only when the failures of both components are detected, the result is detected.

It is easy to verify that $\ogreaterthan$ is not commutative but associative and $\|$ is commutative and associative. The following equalities hold for all $x, y, z \in \mathbf{SIS}$:

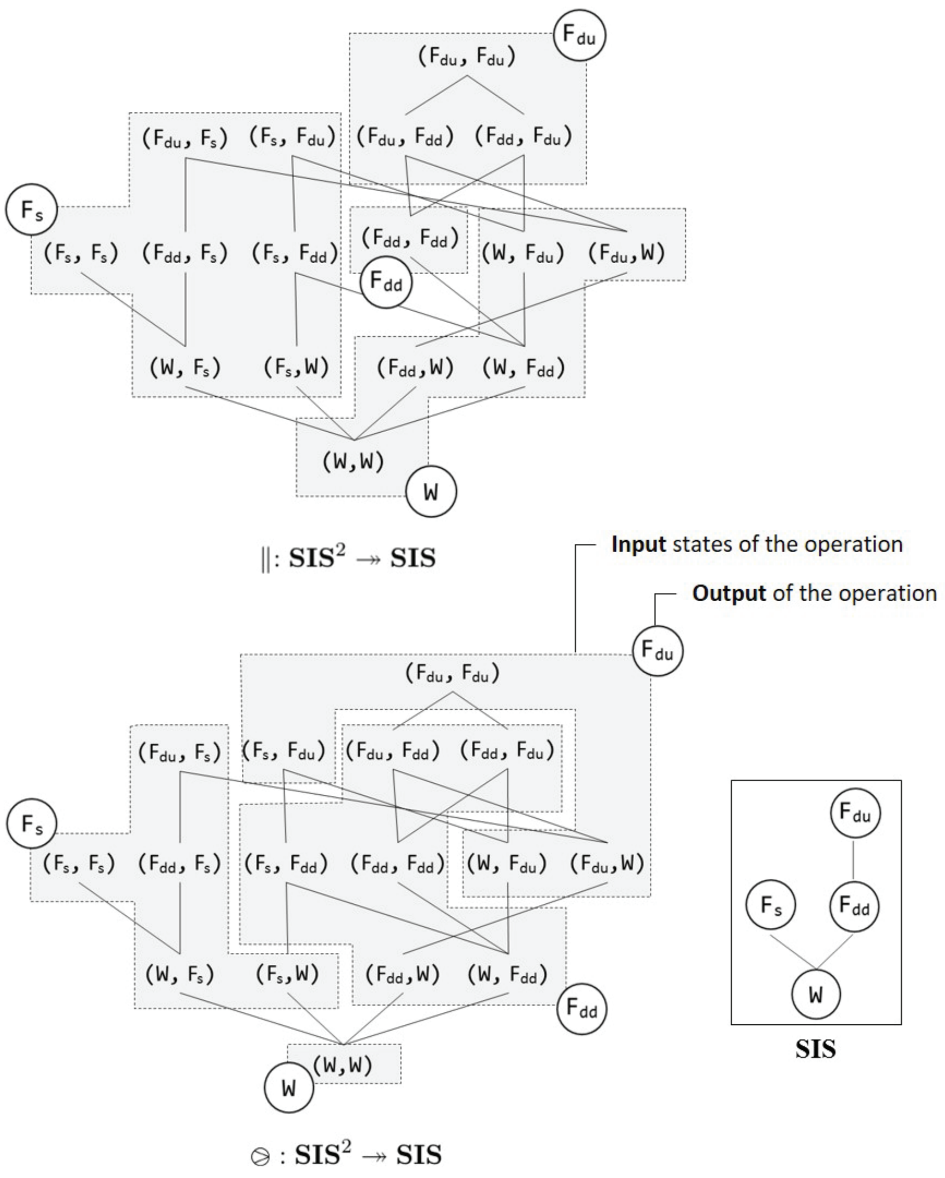
$$\begin{array}{rcl} \ogreaterthan(\ogreaterthan(x,y),z) & = & \ogreaterthan(x,\ogreaterthan(y,z)) \\ \|\,(\|\,(x,y),z) & = & \|\,(x,\|\,(y,z)) \\ \|\,(x,y) & = & \|\,(y,x) \end{array} \tag{3.9}$$

Moreover, $\ogreaterthan$ and $\|$ are also distributive, i.e. the following equalities hold for all $x, y, z \in \mathbf{SIS}$:

$$\begin{array}{rcl} \ogreaterthan(x,\|\,(y,z)) & = & \|\,(\ogreaterthan(x,y),\ogreaterthan(x,z)) \\ \ogreaterthan(\|\,(x,y),z) & = & \|\,(\ogreaterthan(x,z),\ogreaterthan(y,z)) \\ \|\,(\ogreaterthan(x,y),z) & = & \ogreaterthan(\|\,(x,z),\|\,(y,z)) \end{array} \tag{3.10}$$

Figure 3.16 illustrates the valuation of $\ogreaterthan$ and $\|$ in Hasse diagrams. From this figure, we can easily verify that $\|$ is strong-coherent while $\ogreaterthan$ is not coherent.

The strong-coherency of $\|$ indicates that there is an order-similarity between $\mathbf{SIS}^2$ and $\mathbf{SIS}$ under $\|$. The order-similarity will be further discussed in Section 5.1.3.



(F_du, F_du)
(F_du, F_s) (F_s, F_du) (F_du, F_dd) (F_dd, F_du)
(F_s, F_s) (F_dd, F_s) (F_s, F_dd) (F_dd, F_dd) (W, F_du) (F_du, W)
(W, F_s) (F_s, W) (F_dd, W) (W, F_dd)
(W, W)
F_du
F_s
F_dd
W
‖: SIS² ↠ SIS
Input states of the operation
Output of the operation
⊘ : SIS² ↠ SIS
SIS

Figure 3.16: The illustration of $\oslash$ and $\|$.

Chapter 4

Finite degradation models: a unified formalism of combinatorial reliability models

The models built on the framework $\langle \mathbf{FDS}, \otimes, \Phi \rangle$ are named as finite degradation models (FDMs). A *model* (of a theory) is an interpretation satisfying the sentences in a formal language (of that theory). When it comes to a (formal) language, there is always a set of symbols (called the language's alphabet), rules governing the structure of sentences (i.e. the *syntax* of the language) and meanings assigned to syntactically valid sentences in a language (i.e. the *semantics* of the language). For FDM, its syntax is defined by well-formed formulas and its semantics is determined by operations on FDSs.

The remainder of this chapter is as follows. Section 4.1 defines the syntax of FDMs. Section 4.2 gives the interpretation of FDMs. Finally, Section 4.3 shows the modeling of the safety instrumented system presented Section 2.3.

4.1 Syntax: the structure of a model

4.1.1 Well-formed formulas

Let $\mathbf{U}$ be a finite set of FDSs, $\mathbf{V}$ be a finite set of *variables*, $\mathbf{O}$ be a finite set of *operators* on $\mathbf{U}$ and $\alpha : \mathbf{O} \to \mathbb{N}$ be the arity of operators.

Each variable v of $\mathbf{V}$ is assumed to take its value in the support set of one of the FDSs of $\mathbf{U}$. This FDS is called the domain of v and is denoted by $\mathrm{dom}(v)$.

Definition 4.1.1 (Formulas)**.** The set of formulas built over $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{O}$ is the smallest set such that:

– Constants, i.e. elements of FDSs in $\mathbf{U}$, are formulas.

– Variables of $\mathbf{V}$ are formulas.

– If $\diamondsuit$ is an operator of $\mathbf{O}$ with $\alpha(\diamondsuit) = n$ and $f_1, ..., f_n$ are formulas, then $\diamondsuit(f_1, ..., f_n)$ is a formula.

A formula is *well-formed* if it is syntactically correct according to Definition 4.1.1. A formula is *well-typed*, if each operator in it has the correct number of inputs and each input is of the correct type. In the sequel, we shall say simply formula instead of well-formed well-typed formula.

4.1.2 Finite degradation models (FDMs)

The syntactic structure of a finite degradation model is obtained by lifting up fault tree constructions to $\langle \mathbf{FDS}, \otimes, \Phi \rangle$.

Definition 4.1.2 (FDM)**.** A *finite degradation model* (FDM) $\mathcal{M}$ is a pair $\langle \mathbf{V} = \mathbf{S} \uplus \mathbf{F}, \mathcal{E} \rangle$ written in the following form:

$$\mathcal{M} : \left\{ \begin{array}{ccc} w_1 & := & f_1 \\ w_2 & := & f_2 \\ ... & & ... \\ w_m & := & f_m \end{array} \right\} \tag{4.1}$$

where:

– $\mathbf{S} = \{v_1, ..., v_m\}, m \geq 1$, is a finite set of *state variables*;

– $\mathbf{F} = \{w_1, ..., w_n\}, n \geq 1$, is a finite set of *flow variables*;

– $\mathcal{E} = \{w_1 := f_1, ..., w_n := f_n\}$ is a set of equations such that for any $w_i \in \mathbf{F}$, there is exactly one equation $w_i := f_i \in \mathcal{E}$ whose left-hand side member is w_i and f_i is a formula built over the given sets of constants, variables and operators. We say that this equation defines w_i and that f_i is the definition of w_i.

The flow variables in $\mathbf{F}$ are variables that appear in the left-hand side member of the equations in $\mathcal{E}$, while the state variables in $\mathbf{S}$ are variables that only appear in the right-hand side member of the equations in $\mathcal{E}$. Accordingly, $\mathbf{V}$ is divided into two disjoint sets, i.e. $\mathbf{V} = \mathbf{S} \uplus \mathbf{F}$. The terms "state" and "flow" comes from guarded transition systems (Rauzy 2008).

Denote the set of variables appearing in the formula f by var(f).

Definition 4.1.3. Given that $w := f$, we say that w *depends* on a variable u if either $u \in$ var(f) or there exits a variable $u' \in$ var(f) that depends on u.

The FDM $\mathcal{M}$ is *looped* if there exists a flow variable that depends on itself. It is *loop-free* or *data-flow* otherwise. From now, we shall consider only data-flow models.

A *root variable* of $\mathcal{M}$ is a flow variable that occurs in none of the right members of equations. $\mathcal{M}$ is *uniquely rooted* if it has only one root variable. This unique root of $\mathcal{M}$ represents in general the state of the system as a whole.

It is easy to see that FDMs generalize fault trees: state and flow variables play respectively the role of basic and internal events; the equations play the role of intermediate gates and the root variable plays the role of top event.

4.1.3 Graphical representation: expression tree

If $\mathcal{M}$ is loop-free, the syntactic structure of $\mathcal{M}$ can be represented graphically by an expression tree.

Assume that the operators in $\mathbf{O}$ are binary. Then, the expression tree representing $\mathcal{M}$ is a binary expression tree where:

- Each internal node is an *operator* node, denoted by the triple $\langle \Diamond, n_l, n_r \rangle$, which is labeled with an operator $\Diamond$ and pointing to the two child-nodes n_l (left-child) and n_r (right-child).
- Each terminal node is a *variable* node, denoted by $\langle v, /, / \rangle$, which is only labeled with a state variable $v \in \mathbf{S}$ and has no child-node. The symbol / stands for `nil`.

Each node of this expression tree encodes a formula. This formula can be obtained recursively as follows:

- The formula encoded by the terminal node $\langle v, /, / \rangle$ is v.
- The formula encoded by the internal node $\langle \Diamond, n_l, n_r \rangle$ is $\Diamond(f_l, f_r)$, where f_l, f_r are the two formulas respectively encoded by n_l and n_r.

Example 4.1.1. Take the following FDM as an example:

$$\mathcal{M} : \left\{ \begin{array}{lcl} w_1 & := & \diamondsuit_1(v_1, v_2) \\ w_2 & := & \diamondsuit_2(w_1, v_3) \end{array} \right\} \tag{4.2}$$

The expression tree representing this model is pictured Figure 4.1.

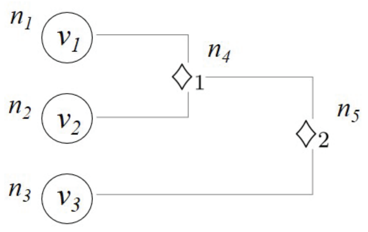


Figure 4.1: The expression tree of the model in Eq.(4.2).

The node definitions of the expression tree in Figure 4.1 are given Table 4.1. Flow variables do not appear explicitly in the expression tree. Instead, each flow variable is associated to the operator node encoding its definition formula.

Table 4.1: Interpretation of the expression tree in Figure 4.1.

Node	Definition	Formula	Assocated variables
n_1	$\langle v_1, /, / \rangle$	v_1	v_1
n_2	$\langle v_2, /, / \rangle$	v_2	v_2
n_3	$\langle v_3, /, / \rangle$	v_3	v_3
n_4	$\langle \diamondsuit_1, n_1, n_2 \rangle$	$\diamondsuit_1(v_1, v_2)$	w_1
n_5	$\langle \diamondsuit_2, n_4, n_5 \rangle$	$\diamondsuit_2(\diamondsuit_1(v_1, v_2), v_3)$	w_2

Definition 4.1.4 (Syntactic solution)**.** Given $\mathcal{M}$, the *syntactic solution* of a flow variable w in $\mathcal{M}$, denoted by f_w, is defined as the formula encoded by the operator node in the expression tree of $\mathcal{M}$ that is associated to w.

Take the FDM in Eq.(4.2) as example. The syntactic solutions of w_1 and w_2 are respectively the formulas encoded by the two nodes n_4 and n_5, i.e.

$$\left\{ \begin{array}{lcl} f_{w_1} & = & \diamondsuit_1(v_1, v_2) \\ f_{w_2} & = & \diamondsuit_2(f_{w_1}, v_3) = \diamondsuit_2(\diamondsuit_1(v_1, v_2), v_3) \end{array} \right. \tag{4.3}$$

It is worth mentioning that in the assessment of FDMs, we don't write the syntactic solutions like in Eq.(4.3), because when $\mathcal{M}$ gets large, the syntactic solution may be too long to write. Instead, the assessment algorithm is directed implemented from the expression tree, see Section 5.3. This expression tree is the data structure used in our software to encode FDMs.

The notion of syntactic solution is only defined as a concept to compare with the semantic solution defined next section. Moreover, let w be a flow variable defined by $w \coloneqq f$. The variables in its syntactic solution f_w are all state variables (i.e. $\text{var}(f_w) \subseteq \mathbf{S}$), while the variables in f can be state variables or flow variables.

4.2 Semantics: the meaning of a model

Defining the semantics of a model means to assign meaning to each symbol written in the model. The meaning of a symbol is called its *interpretation*, denoted by $[\![.]\!]$.

4.2.1 Interpretation of formulas

Let f be a formula written over $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{O}$. Then, a variable valuation of f is a mapping $\sigma : \text{var}(f) \rightarrow \prod_{v \in \text{var}(f)} \text{dom}(v)$, which associates with each variable a value from its domain.

Denote the interpretation of f under the variable valuation σ by $[\![f]\!](\sigma)$. Then, $[\![f]\!](\sigma)$ can be defined recursively as follows:

- If f is reduced to a constant c, then $[\![f]\!](\sigma) = c$.
- If f is reduced to a variable v, then $[\![f]\!](\sigma) = [\![v]\!](\sigma) = \sigma(v)$, which is a valuation of v in its supporting domain, i.e. $\sigma(v) \in \text{dom}(v) \in \mathbf{U}$.
- If f is in the form of $\diamondsuit(f_1, ..., f_n)$, then $[\![f]\!](\sigma) = [\![\diamondsuit]\!]([\![f_1]\!](\sigma), ..., [\![f_n]\!](\sigma))$, where $[\![\diamondsuit]\!]$ is interpreted as the following operation:

$$[\![\diamondsuit]\!] : \bigotimes_{i=1}^{n} \mathcal{S}_i \twoheadrightarrow \mathcal{T} \tag{4.4}$$

  such that,

  - $\mathcal{S}_1, ..., \mathcal{S}_n \in \mathbf{U}$ are the domains of the n input arguments.
  - $\mathcal{T}$ is the domain of f, i.e. $\text{dom}(f) = \text{codom}([\![\diamondsuit]\!]) = \mathcal{T}$.

4.2.2 Interpretation of FDMs

A FDM $\mathcal{M}$ is *well-typed* if $\text{dom}(f) = \text{dom}(w)$ for each equation $w \coloneqq f$ in $\mathcal{M}$. From now, we shall only consider well-typed models.

Let σ be a partial variable valuation that only assigns values to state variables in $\mathbf{S}$:

$$\sigma : \mathbf{S} \to \prod_{v \in \mathbf{S}} \mathrm{dom}(v) \tag{4.5}$$

A variable valuation σ is *admissible* in $\mathcal{M}$ if $[\![w]\!](\sigma) = [\![f]\!](\sigma)$ for each equation $w := f$ in $\mathcal{M}$.

Proposition 4.2.1 (Unicity of admissible variable valuations)**.** Let $\mathcal{M}$ be a FDM and σ be a partial variable valuation. There is a unique way to extend σ into an admissible total valuation $\vec{\sigma}$ of variables of $\mathcal{M}$:

$$\vec{\sigma} : \mathbf{V} \to \prod_{v \in \mathbf{V}} \mathrm{dom}(v) \tag{4.6}$$

Since there is indeed a one-to-one-correspondence between variable valuations and elements of $\prod_{v \in \mathbf{V}} \mathrm{dom}(v)$, we shall thus make no distinction between them in the sequel.

Definition 4.2.1 (Interpretation of FDMs)**.** Given a FDM $\mathcal{M} : \langle \mathbf{V} = \mathbf{S} \uplus \mathbf{F}, \mathcal{E} \rangle$, the interpretation of $\mathcal{M}$ is the following abstraction:

$$[\![\mathcal{M}]\!] : \bigotimes_{v \in \mathbf{S}} \mathrm{dom}(v) \twoheadrightarrow \bigotimes_{w \in \mathbf{F}} \mathrm{dom}(w) \tag{4.7}$$

which assigns values to flow variables according to the valuation of state variables.

Definition 4.2.2 (Semantic solution)**.** Given $\mathcal{M}$, the *semantic solution* of a flow variable w is the following operation:

$$[\![\mathcal{M}]\!]_w : \bigotimes_{v \in \mathbf{S}} \mathrm{dom}(v) \twoheadrightarrow \mathrm{dom}(w) \tag{4.8}$$

such that $[\![w]\!](\vec{\sigma}) = [\![\mathcal{M}]\!]_w(\sigma)$, where $\vec{\sigma}$ is the unique extension of σ in Eq.(4.6).

Now, let's make a comparison between the syntactic solution and the semantic solution of a flow variable w.

As defined Definition 4.1.4, the syntactic solution of w is a formula f_w written over the state variables in $\mathrm{var}(f_w)$. Therefore, it can be interpreted as the following operation:

$$[\![f_w]\!] : \bigotimes_{v \in \mathrm{var}(f_w)} \mathrm{dom}(v) \twoheadrightarrow \mathrm{dom}(w) \tag{4.9}$$

Proposition 4.2.2. According to Eq.(4.8) and Eq.(4.9), we can deduce that:

- If $\text{var}(f_w) = \mathbf{S}$, then $[\![\mathcal{M}]\!]_w = [\![f_w]\!]$.
- If $\text{var}(f_w) \subset \mathbf{S}$, then $\forall(x_1, ..., x_k) \in \bigotimes_{v\in\text{var}(f_w)} \text{dom}(v)$ and $\forall(y_1, ..., y_l) \in \bigotimes_{v\in\mathbf{S}\setminus\text{var}(f_w)} \text{dom}(v)$, $[\![\mathcal{M}]\!]_w(x_1, ..., x_k, y_1, ..., y_l) = [\![f_w]\!](x_1, ..., x_k)$.

In other words, the domain of $[\![f_w]\!]$ only contains the state variables that w depends on, while the domain of $[\![\mathcal{M}]\!]_w$ is the complete state space of state variables in $\mathcal{M}$.

Example 4.2.1. Take again the model $\mathcal{M}$ in Eq.(4.2) as example. In this case, the partial valuation of state variables is:

$$\sigma : \mathbf{S} \to \bigotimes_{i=1}^{3} \text{dom}(v_i)$$

The syntactic solutions f_{w_1} and f_{w_1} are interpreted as follows:

$$[\![f_{w_1}]\!] : \text{dom}(v_1) \otimes \text{dom}(v_2) \twoheadrightarrow \text{dom}(w_1)$$

$$[\![f_{w_2}]\!] : \bigotimes_{i=1}^{3} \text{dom}(v_i) \twoheadrightarrow \text{dom}(w_2)$$

such that $\forall x \in \text{dom}(v_1), \forall y \in \text{dom}(v_2), \forall z \in \text{dom}(v_3)$,

$$[\![f_{w_1}]\!](x, y) = [\![\diamondsuit_1]\!](x, y)$$

$$[\![f_{w_2}]\!](x, y, z) = [\![\diamondsuit_2]\!]([\![\diamondsuit_1]\!](x, y), z)$$

Therefore, the semantic solutions $[\![\mathcal{M}]\!]_{w_1}$ and $[\![\mathcal{M}]\!]_{w_1}$ are the following operations:

$$[\![\mathcal{M}]\!]_{w_1} : \bigotimes_{i=1}^{3} \text{dom}(v_i) \twoheadrightarrow \text{dom}(w_1)$$

$$[\![\mathcal{M}]\!]_{w_2} : \bigotimes_{i=1}^{3} \text{dom}(v_i) \twoheadrightarrow \text{dom}(w_2)$$

such that $\forall \sigma(v_1, v_2, v_3) = (x, y, z) \in \bigotimes_{i=1}^{3} \text{dom}(v_i)$,

$$[\![\mathcal{M}]\!]_{w_1}(\sigma) = [\![f_{w_1}]\!](x, y)$$

$$[\![\mathcal{M}]\!]_{w_2}(\sigma) = [\![f_{w_2}]\!](x, y, z)$$

Finally, $[\![\mathcal{M}]\!]$ is the following abstraction:

$$[\![\mathcal{M}]\!] : \bigotimes_{i=1}^{3} \mathrm{dom}(v_i) \twoheadrightarrow \bigotimes_{j=1}^{2} \mathrm{dom}(w_j)$$

such that $\forall \sigma(v_1, v_2, v_3) = (x, y, z) \in \bigotimes_{i=1}^{3} \mathrm{dom}(v_i)$,

$$\begin{array}{rcl} [\![\mathcal{M}]\!](\sigma) & \overset{def}{=} & ([\![\mathcal{M}]\!]_{w_1}(\sigma), [\![\mathcal{M}]\!]_{w_2}(\sigma)) \\ & = & ([\![f_{w_1}]\!](x, y), [\![f_{w_2}]\!](x, y, z)) \\ & = & ([\![\Diamond_1]\!](x, y), [\![\Diamond_2]\!]([\![\Diamond_1]\!](x, y), z)) \end{array}$$

4.3 The model of SIS

In this section, we shall use the two operations — $\circledgtr$ and $\|$ (see Section 3.3.4) — to construct the FDM of the safety instrumented system presented Section 2.3. To be comparative, two equivalent ways of constructing FDMs are presented.

4.3.1 Fault tree like model

The first way is similar to the construction of fault trees, i.e. either by top-down decomposing system or by bottom-up grouping components according the given operations.

The functional decomposition of the safety instrumented system is pictured Figure 4.2. The system is first decomposed into two safety channels $SC1$ and $SC2$. $SC1$ is comprised by one sensor $S1$, one logic solver $LS1$ and one valve $V1$. $SC2$ is comprised by two sensors $S2$ and $S3$, one logic solver $LS2$ and two valves $V1$ and $V2$. The safety channel $SC2$ is further decomposed to a group of sensors GS, a group of valves GV with the single logic solver $LS2$.

According to this decomposition, the FDM of such system can be written as follows:

$$\mathcal{M}_1 : \left\{ \begin{array}{rcl} System & := & \| \, (SC1, SC2) \\ SC1 & := & \circledgtr(\circledgtr(S1, LS1), V1) \\ SC2 & := & \circledgtr(\circledgtr(GS, LS2), GV) \\ GS & := & \| \, (S2, S3) \\ GV & := & \| \, (V1, V2) \end{array} \right\} \tag{4.10}$$

where,

- $\mathbf{O} = \{\circledgtr, \|\}$;

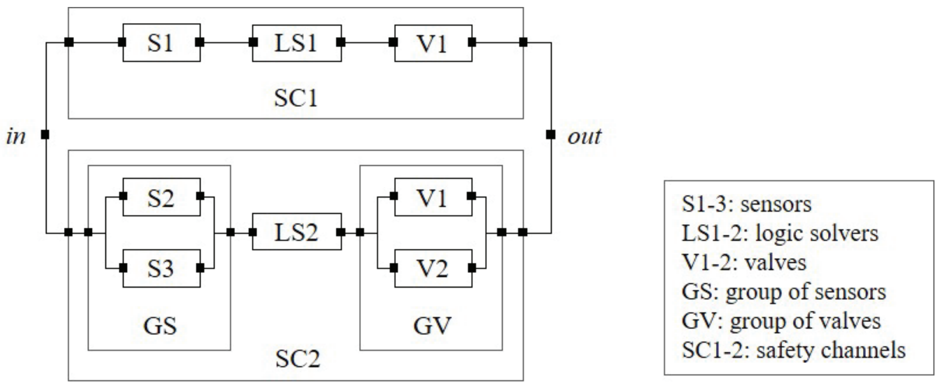


Figure 4.2: The two channels of the SIS in Figure 2.2.

- $\mathbf{S} = \{S1, S2, S3, LS1, LS2, V1, V2\}$;
- $\mathbf{F} = \{SC1, SC2, GS, GV, System\}$.

The expression tree of $\mathcal{M}_1$ is pictured Figure 4.3. In this figure, we use dashed arrows to show the association of flow variables to their corresponding operator nodes.

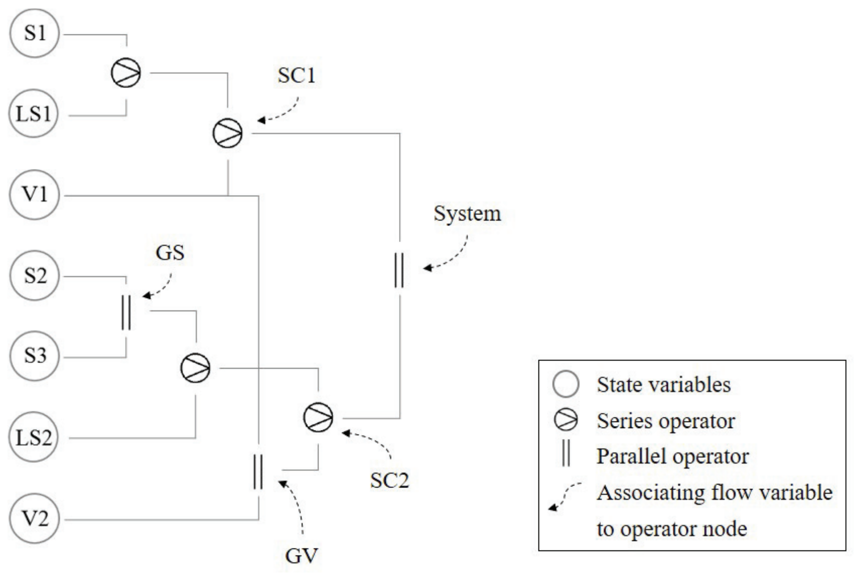


Figure 4.3: The expression tree representation of the model in Eq.(4.10).

From this expression tree, we can thus deduce the syntactic solutions of the 5 flow variables. The results are given below:

$$\begin{cases} f_{GS} & = & \parallel (S2, S3) \\ f_{GV} & = & \parallel (V1, V2) \\ f_{SC1} & = & \ominus(\ominus(S1, LS1), V1) \\ f_{SC2} & = & \ominus(\ominus(\parallel (S2, S3), LS2), \parallel (V1, V2)) \\ f_{System} & = & \parallel (\ominus(\ominus(S1, LS1), V1), \ominus(\ominus(\parallel (S2, S3), LS2), \parallel (V1, V2))) \end{cases} \tag{4.11}$$

Therefore, the model $\mathcal{M}_1$ in Eq.(4.10) is interpreted as:

$$[\![\mathcal{M}_1]\!] : \mathbf{SIS}^7 \twoheadrightarrow \mathbf{SIS}^5 \tag{4.12}$$

such that $\forall \sigma(\mathbf{S}) = (s_1, ls_1, v_1, s_2, s_3, ls_2, v_2) \in \mathbf{SIS}^7$:

$$\begin{cases} [\![\mathcal{M}_1]\!]_{GS}(\sigma) & = & [\![f_{GS}]\!]_\sigma = \parallel (s_2, s_3) \\ [\![\mathcal{M}_1]\!]_{GV}(\sigma) & = & [\![f_{GV}]\!]_\sigma = \parallel (v_1, v_2) \\ [\![\mathcal{M}_1]\!]_{SC1}(\sigma) & = & [\![f_{SC1}]\!]_\sigma = \ominus(\ominus(s_1, ls_1), v_1) \\ [\![\mathcal{M}_1]\!]_{SC2}(\sigma) & = & [\![f_{SC2}]\!]_\sigma = \ominus(\ominus([\![f_{GS}]\!]_\sigma, ls_2), [\![f_{GV}]\!]_\sigma) \\ & = & \ominus(\ominus(\parallel (s_2, s_3), ls_2), \parallel (v_1, v_2)) \\ [\![\mathcal{M}_1]\!]_{System}(\sigma) & = & [\![f_{System}]\!]_\sigma = \parallel ([\![f_{SC1}]\!]_\sigma, [\![f_{SC2}]\!]_\sigma) \\ & = & \parallel (\ominus(\ominus(s_1, ls_1), v_1), \ominus(\ominus(\parallel (s_2, s_3), ls_2), \parallel (v_1, v_2))) \end{cases}$$

Remark here again that in the assessment of FDMs (see Chapter 5), we never calculate or write the syntactic solutions and the semantic solutions like above. Those formulas are just given here to show what means mathematically the concept of syntactic/semantic solutions. In the assessment, the former are encoded by expression trees and the latter are encoded by decision diagrams.

4.3.2 Reliability block diagram like model

The second way of constructing FDMs is similar to the construction of reliability block diagrams, i.e. components are successively "connected" by concatenating their inputs and outputs.

The input and the output of each unit are marked by small black rectangles in Figure 4.2. The input and the output of the whole system is denoted by in and out. The input and the output of a component X is denoted by $X.in$ and $X.out$.

The FDM constructed following such idea of concatenating inputs and outputs can be written as follows:

$$\mathcal{M}_2 : \left\{ \begin{array}{rcl} S1.in & := & in \\ S1.out & := & \circleddash(S1.in, S1) \\ S2.in & := & in \\ S2.out & := & \circleddash(S2.in, S2) \\ S3.in & := & in \\ S3.out & := & \circleddash(S3.in, S3) \\ LS1.in & := & S1.out \\ LS1.out & := & \circleddash(LS1.in, LS1) \\ LS2.in & := & \parallel(S2.out, S3.out) \\ LS2.out & := & \circleddash(LS2.in, LS2) \\ V1.in & := & LS1.out \\ V1.out & := & \circleddash(V1.in, V1) \\ V1'.in & := & LS2.out \\ V1'.out & := & \circleddash(V1'.in, V1) \\ V2.in & := & LS2.out \\ V2.out & := & \circleddash(V2.in, V2) \\ SC2.out & := & \parallel(V1'.out, V2.out) \\ out & := & \parallel(V1.out, SC2.out) \end{array} \right\} \tag{4.13}$$

Since the component $V1$ is shared in different the safety channels, we use $V1.in$ and $V1.out$ for its input and output in safety channel 1, while use $V1'.in$ and $V1'.out$ for its input and output in safety channel 2.

From Eq.(4.10) and Eq.(4.13), we can see that the syntactic structure of $\mathcal{M}_1$ and $\mathcal{M}_2$ are different. But we can prove that $[\![\mathcal{M}_1]\!]_{System} = [\![\mathcal{M}_2]\!]_{out}$ when $in = W$ is satisfied in $\mathcal{M}_2$.

First, according to the distributivity of $\circleddash$ and $\parallel$ presented Section 3.2.3, we can deduce that the definition formula of $SC2.out$ in $\mathcal{M}_2$ can be reformed as follows:

$$\begin{array}{rcl} \parallel(V1'.out, V2.out) & = & \parallel(\circleddash(LS2.out, V1), \circleddash(LS2.out, V2)) \\ & = & \circleddash(LS2.out, \parallel(V1, V2)) \end{array}$$

Then, if $in = W$, we can verify that:

$$\begin{array}{rcl} f_{out} & = & \parallel(\circleddash(LS2.out, V1), \parallel(V1'.out, V2.out)) \\ & = & \parallel(V1.out, \circleddash(LS2.out, \parallel(V1, V2))) \\ & = & \parallel(\circleddash(\circleddash(S1, LS1), V1), \circleddash(\circleddash(\parallel(S2, S3), LS2), \parallel(V1, V2))) \\ & = & f_{System} \end{array}$$

Therefore, $\forall \sigma \in \mathbf{SIS}^7$, $[\![\mathcal{M}_1]\!]_{System}(\sigma) = [\![\mathcal{M}_2]\!]_{out}(\sigma)$ when $in = W$ in $\mathcal{M}_2$.

To summarize, the ways of constructing FDM for a given system is generally not unique. This section presents two ways, i.e. the fault tree like and the reliability block diagram like constructions. The former is somehow more compact and more abstract than the latter. Although the syntactic structure of the two models are different, the valuation of their root variables is equivalent under appropriate conditions.

We can also discover that the role of flow variables in a FDM is twofold. On the one hand, flow variables can be used to structure the model and sometimes the addition of intermediate flow variables may facilitate the modeling process. On the other hand, flow variables also declare the "observable" points of the model, i.e. on which we can observe how the valuation of components' states influences the system's state. For this reason, flow variables are used to define the observers in the assessment of FDMs, see next chapter.

Chapter 5

Assessment of Finite Degradation Models

Similar to fault tree analysis, the assessment of FDMs includes also a qualitative part and a quantitative part.

- The qualitative assessment of FDMs is the scenarios analysis, which can be seen as the formal extension of the cutsets analysis from fault trees to FDMs. As results, we define and calculate the set of scenarios, critical (maximal/minimal) scenarios and conditional scenarios.

- The quantitative assessment of FDMs is the probabilistic calculation of probabilistic reliability and safety indicators, such as state probabilities, conditional probabilities and sensitivity factors.

The assessment technique used for FDMs is also the decision diagram technique. We modify the structure of classical binary decision diagram to fit FDMs and provide new algorithms to calculate the required scenarios and probabilistic indicators.

The remainder of this chapter is as follows. Section 5.1 introduces the scenarios analysis for FDMs. Section 5.2 introduces the probabilistic calculations for FDMs. Finally, Section 5.3 presents the decision diagrams and the algorithms to support the required calculations.

5.1 Scenarios analysis

5.1.1 Scenarios

Definition 5.1.1 (Observer)**.** An *observer* of a FDM $\mathcal{M} : \langle \mathbf{V} = \mathbf{S} \uplus \mathbf{F}, \mathcal{E}\rangle$ is a predicate on the valuation of a flow variable in $\mathbf{F}$.

Let w be a flow variable of $\mathcal{M}$, y be a state in $\text{dom}(w)$ and Y be a subset of $\text{dom}(w)$. Then, an observer of w can be: $w = y$, $w \neq y$, $w \in Y$ or $w \notin Y$.

The observers are used to indicate the objective behavior that should be analyzed in the current assessment. In fault tree analysis, the observer is the occurrence of the TOP event. In FDMs, such notion is generalized to fit multistate systems, i.e. any flow variable and any state (or subset of states) of its valuation domain can be selected as the target of the assessment.

Definition 5.1.2 (Set of scenarios)**.** Given an observer $w = y$, we define the set of *scenarios* satisfying $w = y$, denoted by $\mathbf{Sce}(w = y)$, as follows:

$$\mathbf{Sce}(w = y) \overset{def}{=} \{\overline{\mathbf{v}} | \overline{\mathbf{v}} \in \bigotimes_{v \in \mathbf{S}} \text{dom}(v), [\![\mathcal{M}]\!]_w(\overline{\mathbf{v}}) = y\} \tag{5.1}$$

$\overline{\mathbf{v}}$ is called a state vector or a scenario of $\mathcal{M}$.

$[\![\mathcal{M}]\!]_w$ is the semantic solution of w (see Definition 4.2.2). The set of scenarios $\mathbf{Sce}(w = y)$ is thus the set of preimages of y under $[\![\mathcal{M}]\!]_w$, i.e. $\mathbf{Sce}(w = y) = ([\![\mathcal{M}]\!]_w)^{-1}[y]$.

Moreover, we can deduce that $\forall y, z \in \text{dom}(w)$,

$$y \neq z \Rightarrow \mathbf{Sce}(w = y) \cap \mathbf{Sce}(w = z) = \emptyset$$

Therefore, the other sets of scenarios $\mathbf{Sce}(w \neq y)$, $\mathbf{Sce}(w \in Y)$ and $\mathbf{Sce}(w \notin Y)$ can be calculated from $\mathbf{Sce}(w = y)$ by the following formulas:

$$\begin{array}{r}\mathbf{Sce}(w \in Y) = \biguplus_{y \in Y} \mathbf{Sce}(w = y)\\ \mathbf{Sce}(w \neq y) = \biguplus_{z \in \text{dom}(w), z \neq y} \mathbf{Sce}(w = z)\\ \mathbf{Sce}(w \notin Y) = \biguplus_{z \in \text{dom}(w), z \notin Y} \mathbf{Sce}(w = z)\end{array} \tag{5.2}$$

where $\uplus$ stands for the disjoint union of sets.

Example 5.1.1. Take the following model as an example, where $\wedge$ is the meet operator (see Section 6.2.2) and $\text{dom}(u) = \text{dom}(v) = \mathbf{WDF}$.

$$\mathcal{M} : \{\ w \ := \ \wedge(u, v)\ \} \tag{5.3}$$

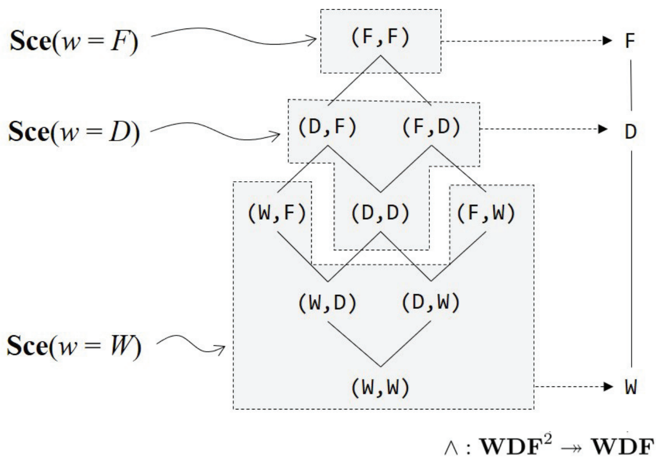


Figure 5.1: The three sets of scenarios $\mathbf{Sce}(w = W)$, $\mathbf{Sce}(w = D)$ and $\mathbf{Sce}(w = F)$ for w defined by the meet operation $\wedge : \mathbf{WDF}^2 \twoheadrightarrow \mathbf{WDF}$.

The valuation of $\wedge$ is pictured Figure 5.1.

In this figure, we also illustrate the three sets of scenarios $\mathbf{Sce}(w = W)$, $\mathbf{Sce}(w = D)$ and $\mathbf{Sce}(w = F)$:

$$\begin{cases} \mathbf{Sce}(w = W) & = & \{(W, W), (W, D), (D, W), (W, F), (F, W)\} \\ \mathbf{Sce}(w = D) & = & \{(D, D), (F, D), (D, F)\} \\ \mathbf{Sce}(w = F) & = & \{(F, F)\} \end{cases} \tag{5.4}$$

We can see that the domain $\mathbf{WDF}^2$ is partitioned into three zones with respect to the three observers $w = W$, $w = D$ and $s = F$.

5.1.2 Conditional scenarios

In the previous section, the set of scenarios are selected by a given valuation of flow variable. However, it might also be useful to select scenarios by the valuation of state variables.

Let v be a state variable of $\mathcal{M}$ and σ be a partial variable valuation of states variables.

Definition 5.1.3. Given $\mathbf{Sce}(w = y)$, the set of conditional scenarios $\mathbf{Sce}(w = y|v = c)$ is defined as follows:

$$\mathbf{Sce}(w = y|v = c) \overset{def}{=} \{\overline{\mathbf{v}}|\overline{\mathbf{v}} \in \mathbf{Sce}(w = y), \sigma(v) = c\} \tag{5.5}$$

$\mathbf{Sce}(w = y|v = c)$ is a subset of $\mathbf{Sce}(w = y)$, in which the valuation of the state variable v is limited to be c.

Example 5.1.2. Take the model in Eq.(5.4) as an example. Then, $\forall y \in \mathbf{WDF}, \forall c \in \mathbf{WDF}$, the conditional scenarios are listed below:

$$\left\{\begin{array}{lcl} \mathbf{Sce}(w = W|u = W) &=& \{(W,W),(W,D),(W,F)\} \\ \mathbf{Sce}(w = W|u = D) &=& \{(D,W)\} \\ \mathbf{Sce}(w = W|u = F) &=& \{(F,W)\} \\ && \\ \mathbf{Sce}(w = D|u = W) &=& \emptyset \\ \mathbf{Sce}(w = D|u = D) &=& \{(D,D),(D,F)\} \\ \mathbf{Sce}(w = D|u = F) &=& \{(F,D)\} \\ && \\ \mathbf{Sce}(w = F|u = W) &=& \emptyset \\ \mathbf{Sce}(w = F|u = D) &=& \emptyset \\ \mathbf{Sce}(w = F|u = F) &=& \{(F,F)\} \end{array}\right. \tag{5.6}$$

We can also deduce that $\forall c, d \in \text{dom}(v)$,

$$c \neq d \Rightarrow \mathbf{Sce}(w = y|v = c) \cap \mathbf{Sce}(w = y|v = d) = \emptyset \tag{5.7}$$

Proposition 5.1.1. The set of scenarios $\mathbf{Sce}(w = y)$ can be decomposed into the sum of disjoint sets of conditional scenarios $\mathbf{Sce}(w = y|v = c)$:

$$\mathbf{Sce}(w = y) = \biguplus_{c \in \text{dom}(v)} \mathbf{Sce}(w = y|v = c) \tag{5.8}$$

To be more generic, denote respectively the set of conditions made on the valuation of state variables and made on the valuation offlow variables by $\mathbf{C_S}$ and $\mathbf{C_F}$, such that:

$$\left\{\begin{array}{lcll} \mathbf{C_S} &=& \bigcup_{k \in I_\mathbf{S}} \{v_k = c_k\}, & v_k \in \mathbf{S}, c_k \in \text{dom}(v_k) \\ \mathbf{C_F} &=& \bigcup_{l \in I_\mathbf{F}} \{w_l = y_l\}, & w_l \in \mathbf{F}, y_l \in \text{dom}(w_l) \end{array}\right. \tag{5.9}$$

where $I_\mathbf{S}, I_\mathbf{F}$ are respectively the index sets of variables appearing in $\mathbf{C_S}$ and $\mathbf{C_F}$.

Proposition 5.1.2. The set of conditional scenarios $\mathbf{Sce}(\mathbf{C_F}|\mathbf{C_S})$ satisfying the conditions in both $\mathbf{C_S}$ and $\mathbf{C_F}$ can be calculated as follows:

$$\mathbf{Sce}(\mathbf{C_F}|\mathbf{C_S}) = \bigcap_{l \in I_\mathbf{F}} \bigcap_{k \in I_\mathbf{S}} \mathbf{Sce}(w_l = y_l|v_k = c_k) \tag{5.10}$$

$\mathbf{Sce}(\mathbf{C_F}|\mathbf{C_S})$ is the extension of the notion of cutsets from fault trees to FDMs. In fault trees, a cutset is defined as the combination of components' failures that causes the system (or top event) failure. In FDMs, $\mathbf{Sce}(\mathbf{C_F}|\mathbf{C_S})$ is the set of scenarios satisfying the conditions in both $\mathbf{C_F}$ and $\mathbf{C_S}$.

The reason of separating the conditions made for flow variables and state variables is twofold. First, regarding the interpretation of the model $[\![\mathcal{M}]\!]$ (see Definition 4.2.1), the target zones of the conditions in $\mathbf{C_F}$ and $\mathbf{C_S}$ are different in mathematical sense, i.e. the former is in codom($[\![\mathcal{M}]\!]$) while the latter is in dom($[\![\mathcal{M}]\!]$). Second, in practical sense, the valuation of flow variables often indicates the resultant behaviors, while the valuation of state variables often indicates the causes leading to those results. The conditions made on the former reflect the scope of the target results that we want to analyze, while the conditions made on the latter reflect the scope of the given information that we have on its causes.

5.1.3 Critical scenarios

Let $\mathbf{Sce}(o)$ be the set of scenarios with a given observer o. To be generic, such observer can also be $\mathbf{C_S}$ and $\mathbf{C_F}$.

Definition 5.1.4. Given a set of scenarios $\mathbf{Sce}(o)$, we define its sets of *minimal* scenarios and *maximal* scenarios, denoted respectively by Min$\mathbf{Sce}(o)$ and Max$\mathbf{Sce}(o)$, as follows:

$$\begin{array}{rcl} \text{Min}\mathbf{Sce}(o) & \overset{def}{=} & \min(\mathbf{Sce}(o)) \\ & = & \{\overline{\mathbf{v}} \in \mathbf{Sce}(o) | \nexists \overline{\mathbf{u}} \in \mathbf{Sce}(o), \overline{\mathbf{u}} \sqsubset \overline{\mathbf{v}}\} \\ \text{Max}\mathbf{Sce}(o) & \overset{def}{=} & \max(\mathbf{Sce}(o)) \\ & = & \{\overline{\mathbf{v}} \in \mathbf{Sce}(o) | \nexists \overline{\mathbf{u}} \in \mathbf{Sce}(o), \overline{\mathbf{v}} \sqsubset \overline{\mathbf{u}}\} \end{array} \tag{5.11}$$

Example 5.1.3. Take the model in Figure 5.1 as an example. Following the above definition, the minimal and the maximal scenarios of $\mathbf{Sce}(w = W)$, $\mathbf{Sce}(w = D)$ and $\mathbf{Sce}(w = F)$ are listed below:

$$\begin{cases} \text{Min}\mathbf{Sce}(w = W) & = & \{(W, W)\} \\ \text{Min}\mathbf{Sce}(w = D) & = & \{(D, D)\} \\ \text{Min}\mathbf{Sce}(w = F) & = & \{(F, F)\} \\ & & \\ \text{Max}\mathbf{Sce}(w = W) & = & \{(W, F), (F, W)\} \\ \text{Max}\mathbf{Sce}(w = D) & = & \{(D, F), (F, D)\} \\ \text{Max}\mathbf{Sce}(w = F) & = & \{(F, F)\} \end{cases} \tag{5.12}$$

From Figure 5.1, we can see that the above minimal and maximal scenarios locate exactly in the boundary of $\mathbf{Sce}(w = W)$, $\mathbf{Sce}(w = D)$ and $\mathbf{Sce}(w = F)$.

These scenarios are also called *critical* scenarios, since they characterize the critical situations in which the value of the observer is about to change.

Critical scenarios versus minimal cut/path sets

In fault tree analysis, a *cut set* is defined a set of basic events whose (simultaneous) occurrence ensures that the TOP event occurs. A cutset is said to be *minimal* if the set cannot be reduced without loosing its status as a cutset. A *path set* is defined as a set of basic events whose nonoccurrence (simultaneously) ensures that the TOP event does not occur. A path set is said to be *minimal* if the set cannot be reduced without loosing its status as a path set (Rausand 2004):

Take the fault tree in Figure 5.2 as an concrete example. The TOP event w is modeled by the Boolean formula $w = a.b + c$.

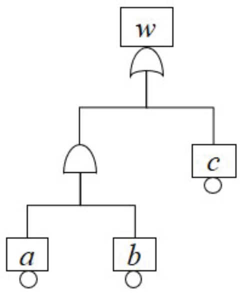


Figure 5.2: A simple fault tree where $w = a.b + c$.

According to the definitions in Rausand (2004), the set of cutsets of this fault tree is $\{abc, ab, ac, bc, c\}$, where the minimal cutsets are $\{ab, c\}$. The set of path sets of this fault tree is $\{\bar{a}\bar{c}, \bar{b}\bar{c}, \bar{a}\bar{b}\bar{c}\}$, where the minimal path sets are $\{\bar{a}\bar{c}, \bar{b}\bar{c}\}$.

This fault tree can be easily transformed in FDMs using the operators $\vee$ and $\wedge$ defined in Section 6.2.1. Its equivalent FDM is given below:

$$\mathcal{M} : \left\{ \; w \quad := \quad \vee(\wedge(a, b), c) \; \right\} \tag{5.13}$$

In this case, the domain of the state variables a, b, c is the binary FDS $\mathbf{WF}$. The valuation of $[\![\mathcal{M}]\!] : \mathbf{WF}^3 \twoheadrightarrow \mathbf{WF}$ is pictured Figure 5.3.

In this figure, we also partition the two sets of scenarios $\mathbf{Sce}(w = W)$ and $\mathbf{Sce}(w = F)$. Comparing the scenarios in $\mathbf{Sce}(w = W)$ and $\mathbf{Sce}(w = F)$ with the cut sets and path sets, we can see that:

– There is a one-to-one-correspondence between the scenarios in $\mathbf{Sce}(w = W)$ with the path sets and a one-to-one-correspondence between the max-

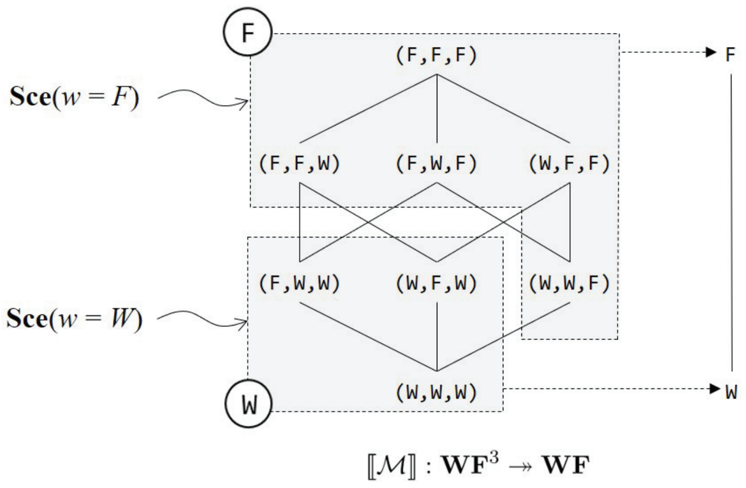


Figure 5.3: Valuation of $[\![\mathcal{M}]\!] : \mathbf{WF}^3 \twoheadrightarrow \mathbf{WF}$ and relevant sets of scenarios.

imal scenarios in Max$\mathbf{Sce}(w = W)$ with the minimal path sets, i.e.

$$\mathbf{Sce}(w = W) = \left\{ \begin{array}{ccc} (W, W, W) & \rightleftharpoons & \bar{a}\bar{b}\bar{c} \\ (F, W, W) & \rightleftharpoons & \bar{b}\bar{c} \\ (W, F, W) & \rightleftharpoons & \bar{a}\bar{c} \end{array} \right\} = \text{path sets}$$

$$\text{Max}\mathbf{Sce}(w = W) = \left\{ \begin{array}{ccc} (F, W, W) & \rightleftharpoons & \bar{b}\bar{c} \\ (W, F, W) & \rightleftharpoons & \bar{a}\bar{c} \end{array} \right\} = \text{minimal path sets}$$

– There is also a one-to-one-correspondence between the scenarios in $\mathbf{Sce}(w = F)$ with the cutsets and a one-to-one-correspondence between the miniimal scenarios in Min$\mathbf{Sce}(w = F)$ with the minimal cutsets, i.e.

$$\mathbf{Sce}(w = F) = \left\{ \begin{array}{ccc} (W, W, F) & \rightleftharpoons & c \\ (F, F, W) & \rightleftharpoons & ab \\ (F, W, F) & \rightleftharpoons & ac \\ (W, F, F) & \rightleftharpoons & bc \\ (F, F, F) & \rightleftharpoons & abc \end{array} \right\} = \text{cutsets}$$

$$\text{Min}\mathbf{Sce}(w = F) = \left\{ \begin{array}{ccc} (W, W, F) & \rightleftharpoons & c \\ (F, F, W) & \rightleftharpoons & ab \end{array} \right\} = \text{minimal cutsets}$$

These correspondences are validated not only in the fault tree of Figure 5.2. In stead, for any fault tree, the results of cut/path sets and minimal cut/path sets are one-to-one correspondent to the scenarios and minimal/maximal scenarios of the observers $TOP = W$ and $TOP = F$ in its equivalent FDMs.

Scenarios analysis for multistate coherent systems

An operator is *coherent* if its interpreted operation is a coherent abstraction (see Definition 3.3.3).

A (loop-free uniquely-rooted) model $\mathcal{M}$ is *coherent* if $[\![\mathcal{M}]\!]$ is a coherent abstraction. Moreover, if the operators in $\mathcal{M}$ are all coherent, then $\mathcal{M}$ is coherent.

If $\mathcal{M}$ is coherent, then for any flow variable w in $\mathcal{M}$, $[\![\mathcal{M}]\!]_w$ is coherent. The coherency of $[\![\mathcal{M}]\!]_w$ indicates that there exists an **order-similarity** between $\text{dom}([\![\mathcal{M}]\!]_w)$ and $\text{dom}(w)$.

Mathematically, if $[\![\mathcal{M}]\!]_w$ is coherent, then $\forall y, z \in \text{dom}(w)$, $\forall \bar{\mathbf{u}} \in \mathbf{Sce}(w = y), \forall \bar{\mathbf{v}} \in \mathbf{Sce}(w = z)$:

$$\begin{aligned} y \sqsubseteq z &\Rightarrow \bar{\mathbf{u}} \sqsubseteq \bar{\mathbf{v}} \\ y \sim z &\Rightarrow \bar{\mathbf{u}} \sim \bar{\mathbf{v}} \end{aligned} \tag{5.14}$$

In the sequel, we shall use three typical FDSs: $\mathbf{WF}$, $\mathbf{WDF}$ and $\mathbf{W2F}$, to illustrate such order-similarity.

First, let $\text{dom}(w) = \mathbf{WF}$. The order-similarity in this case is pictured Figure 5.4. In this figure, we use a inverted choanoid to represent the domain $\bigotimes_{v \in \mathbf{S}} \text{dom}(v)$, since is an arbitrary FDS with the least element $\perp$.

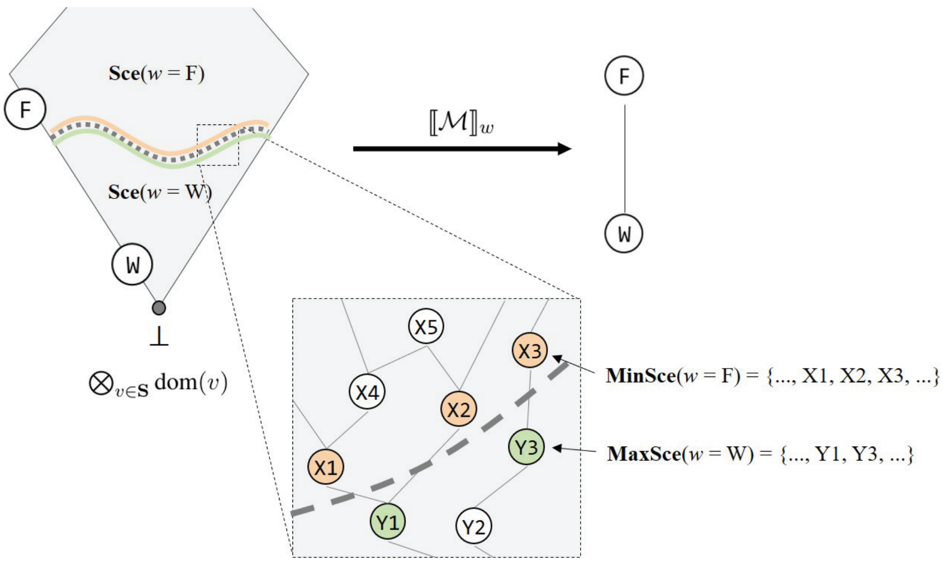

Sce(w = F)
F
Sce(w = W)
W
⊥
⊗v∈S dom(v)
⟦M⟧w
F
W
X5
X3
X4
X2
X1
Y3
Y1
Y2
MinSce(w = F) = {..., X1, X2, X3, ...}
MaxSce(w = W) = {..., Y1, Y3, ...}

Figure 5.4: Order-similarity when $\text{dom}(w) = \mathbf{WF}$.

From this figure, we can see that the domain $\bigotimes_{v \in \mathbf{S}} \text{dom}(v)$ is partitioned into two zones: $\mathbf{Sce}(w = W)$ and $\mathbf{Sce}(w = F)$. If $[\![\mathcal{M}]\!]_w$ is coherent, then according to the order-similarity listed Eq.(5.14), we can deduce that the entire zone of $\mathbf{Sce}(w = F)$ should locate more upward than the zone of $\mathbf{Sce}(w = W)$.

In the rectangle area of Figure 5.4, we zoom in the boundary between $\mathbf{Sce}(w = W)$ and $\mathbf{Sce}(w = F)$. This boundary is theoretically defined by the fmaximal scenarios of $\text{Max}\mathbf{Sce}(w = W)$ and the minimal scenarios of $\text{Min}\mathbf{Sce}(w = F)$. From this rectangle area, we can see that:

- The two scenarios $Y1, Y3$ in $\text{Max}\mathbf{Sce}(w = W)$ are critical for $w = W$ because any **degradation** (i.e. moving upwards in the diagram) will directly degrade the valuation of w from W into F. However, $Y2$ is not critical since if it degrades to $Y3$, the valuation of w remains to be W.
- Symmetrically, the three scenarios $X1, X2, X3$ in $\text{Min}\mathbf{Sce}(w = F)$ are critical for $w = F$ because any **improvement** (i.e. moving downwards in the diagram) will directly improve the valuation of w from F into W. However, $X4, X5$ are not critical since none of their improvements improves the valuation of w.

When $\text{dom}(w) = \mathbf{WDF}$ and $\text{dom}(w) = \mathbf{W2F}$, the order-similarity between the domain $\bigotimes_{v\in\mathbf{S}} \text{dom}(v)$ and $\text{dom}(w)$ is illustrated in the same way in Figure 5.5.

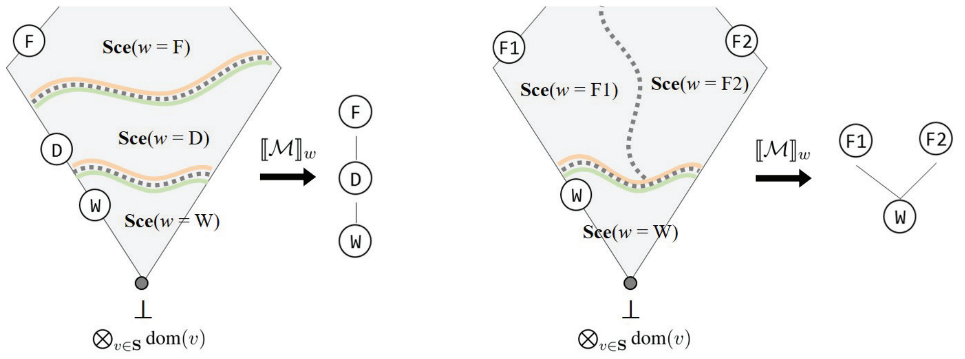


Figure 5.5: Order-similarity when $\text{dom}(w) = \mathbf{WDF}$ and $\text{dom}(w) = \mathbf{W2F}$.

To summarize, if $[\![\mathcal{M}]\!]_w$ is coherent, then:

- $\forall \overline{\mathbf{v}} \in \text{Min}\mathbf{Sce}(w = y)$, if $\exists \overline{\mathbf{u}} \in \bigotimes_{v\in\mathbf{S}} \text{dom}(v)$ such that $\overline{\mathbf{u}} \sqsubset \overline{\mathbf{v}}$, then $\overline{\mathbf{u}} \sqsubset \mathbf{Sce}(w = y)$. It means that the valuation of w will be immediately improved if $\overline{\mathbf{v}}$ is improved to $\overline{\mathbf{u}}$.

– $\forall \overline{\mathbf{v}} \in \text{Max}\mathbf{Sce}(w = y)$, if $\exists \overline{\mathbf{u}} \in \bigotimes_{v \in \mathbf{S}} \text{dom}(v)$ such that $\overline{\mathbf{v}} \sqsubset \overline{\mathbf{u}}$, then $\mathbf{Sce}(w = y) \sqsubset \overline{\mathbf{u}}$. It means that the valuation of w will be immediately degraded if $\overline{\mathbf{v}}$ is degraded to $\overline{\mathbf{u}}$.

The scenarios analysis of coherent systems can be reduced to the analysis of critical scenarios. These scenarios characterize the extreme conditions that the state of the observer is about to change. The advantage of this reduction is that it can make the analysis more efficient since the critical scenarios only account for a small proportion of the total scenarios.

Moreover, by comparing the current situation with the critical scenarios, we can know qualitatively that "how many steps the system may enter into an undesired state from the current state". This qualitative information can be used therefore to forewarn potential risks in real-time analysis.

5.2 Probabilistic calculations

5.2.1 State probabilities

Assume that there are n state variables in $\mathbf{S}$, i.e. $\mathbf{S} = \{v_1, v_2, ..., v_n\}$. Denote the probability measure in $\text{dom}(v_i)$ by p_i and assume that p_i's are independent.

Let $\sigma : \mathbf{S} \to \bigotimes_{i=1}^{n} \text{dom}(v_i)$ be the partial valuation of state variables. Then, for each scenario $\overline{\mathbf{v}} = \sigma(v_1, v_2, ..., v_n) = (\sigma(v_1), \sigma(v_2), ..., \sigma(v_n)) \in \bigotimes_{i=1}^{n} \text{dom}(v_i)$, the scenario probability $p_{\otimes}(\overline{\mathbf{v}})$ is calculated (see Eq.(3.4)) as follows:

$$p_{\otimes}(\overline{\mathbf{v}}) = \prod_{i=1}^{n} p_i(\sigma(v_i)) \tag{5.15}$$

Given an observer $w = y$, the state probability $p_w(y)$ is the probability that the valuation of w is y, i.e. $[\![\mathcal{M}]\!]_w = y$. It can be calculated by definition (see Definition 3.3.2) as follows:

$$p_w(y) = \sum_{\overline{\mathbf{v}} \in [\![\mathcal{M}]\!]_w^{-1}[y]} p_{\otimes}(\overline{\mathbf{v}}) \tag{5.16}$$

According to the definition of $\mathbf{Sce}(w = y)$, we can deduce that:

$$p_w(y) = \sum_{\overline{\mathbf{v}} \in \mathbf{Sce}(w=y)} p_{\otimes}(\overline{\mathbf{v}}) \tag{5.17}$$

Example 5.2.1. Take again the model $\mathcal{M}$ in Eq.(5.3) as example. The state probabilities $p_w(W)$, $p_w(D)$ and $p_w(F)$ can thus be calculated according to the sets of scenarios in Eq.(5.4) as follows:

$$\left\{\begin{array}{lcl} p_w(W) & = & \sum_{(x,y)\in\mathbf{Sce}(w=W)} p_u(x)p_v(y) \\ & = & p_u(W)p_v(W)+p_u(W)p_v(D)+p_u(D)p_v(W)+p_u(W)p_v(F) \\ & & +p_u(F)p_v(W) \\ p_w(D) & = & \sum_{(x,y)\in\mathbf{Sce}(w=D)} p_u(x)p_v(y) \\ & = & p_u(D)p_v(D)+p_u(D)p_v(F)+p_u(F)p_v(D) \\ p_w(F) & = & \sum_{(x,y)\in\mathbf{Sce}(w=F)} p_u(x)p_v(y) \\ & = & p_u(F)p_v(F) \end{array}\right. \tag{5.18}$$

5.2.2 Conditional probability

In probability theory, the conditional probability of an event A given an event B is defined as the quotient of the probability of the joint of A and B and the probability of B (Kolmogorov 1950), i.e.

$$\text{Pr}\{A|B\} = \frac{\text{Pr}\{A\cap B\}}{\text{Pr}\{B\}} \tag{5.19}$$

Let $\overline{\mathbf{v}} = \sigma(v_1, v_2, ..., v_n) = (\sigma(v_1), \sigma(v_2), ..., \sigma(v_n))$ be a scenario.

For any state variable $v_j \in \mathbf{S}$ and for any value $c \in \text{dom}(v_j)$, denote the probability that $\overline{\mathbf{v}}$ occurs given that $\sigma(v_j) = c$ by $\text{Pr}\{\overline{\mathbf{v}}|v_j = c\}$. Then, $\text{Pr}\{\overline{\mathbf{v}}|v_j = c\}$ can be calculated according to Eq.(5.19) as follows:

$$\text{Pr}\{\overline{\mathbf{v}}|v_j = c\} = \left\{\begin{array}{ll} 0, & \text{if } \sigma(v_j) \neq c. \\ p_\otimes(\overline{\mathbf{v}})/p_j(c), & \text{if } \sigma(v_j) = c. \end{array}\right. \tag{5.20}$$

Denote the probability that $w = y$ under the condition that $\sigma(v_j) = c$ by $\text{Pr}\{w = y|v_j = c\}$. Then, $\text{Pr}\{w = y|v_j = c\}$ can be calculated according to Eq.(5.17) as follows:

$$\text{Pr}\{w = y|v_j = c\} = \sum_{\overline{\mathbf{v}}\in\mathbf{Sce}(w=y)} \text{Pr}\{\overline{\mathbf{v}}|v_j = c\} \tag{5.21}$$

Combining the above two formulas, we can deduce that:

$$\text{Pr}\{w = y|v_j = c\} = \left\{\begin{array}{ll} 0, & \text{if } \sigma(v_j) \neq c. \\ \sum_{\overline{\mathbf{v}}\in\mathbf{Sce}(w=y)} p_\otimes(\overline{\mathbf{v}})/p_j(c), & \text{if } \sigma(v_j) = c. \end{array}\right. \tag{5.22}$$

According to the definition of $\mathbf{Sce}(w = y|v_j = c)$, if $\sigma(v_j) \neq c$, then $\mathbf{Sce}(w = y|v_j = c) = \emptyset$ and accordingly $\text{Pr}\{w = y|v_j = c\} = 0$.

If $\mathbf{Sce}(w = y|v_j = c) \neq \emptyset$, then:

$$\Pr\{w = y|v_j = c\} = \sum_{\overline{\mathbf{v}} \in \mathbf{Sce}(w=y|v_j=c)} \left(\prod_{i=1, i\neq j}^{n} p_i(\sigma(v_i)) \right) \tag{5.23}$$

Example 5.2.2. Take again the model $\mathcal{M}$ in Eq.(5.3) as example. The conditional probabilities $\Pr\{w = y|u = c\}$ can be calculated according to the conditional scenarios in Eq.(5.6) as follows:

$$\begin{cases} \Pr\{w = W|u = W\} & = & p_v(W) + p_v(D) + p_v(F) \\ \Pr\{w = W|u = D\} & = & p_v(W) \\ \Pr\{w = W|u = F\} & = & p_v(W) \\ \\ \Pr\{w = D|u = W\} & = & 0 \\ \Pr\{w = D|u = D\} & = & p_v(D) + p_v(F) \\ \Pr\{w = D|u = F\} & = & p_v(D) \\ \\ \Pr\{w = F|u = W\} & = & 0 \\ \Pr\{w = F|u = D\} & = & 0 \\ \Pr\{w = F|u = F\} & = & p_v(F) \end{cases} \tag{5.24}$$

To be more generic, denote the conditional probability satisfying the conditions in $\mathbf{C_S}$ and $\mathbf{C_F}$ (see Eq.(5.9)) by $\Pr\{\mathbf{C_F}|\mathbf{C_S}\}$. Similar to Eq.(5.23), if $\mathbf{Sce}(\mathbf{C_F}|\mathbf{C_S}) = \emptyset$, then $\Pr\{\mathbf{C_F}|\mathbf{C_S}\} = 0$. If $\mathbf{Sce}(\mathbf{C_F}|\mathbf{C_S}) \neq \emptyset$, $\Pr\{\mathbf{C_F}|\mathbf{C_S}\}$ is thus calculated as follows:

$$\Pr\{\mathbf{C_F}|\mathbf{C_S}\} = \sum_{\overline{\mathbf{v}} \in \mathbf{Sce}(\mathbf{C_F}|\mathbf{C_S})} \left(\prod_{i=1, i\notin I_\mathbf{S}}^{n} p_i(\sigma(v_i)) \right) \tag{5.25}$$

where $I_\mathbf{S}$ is the index set of state variables appearing in $\mathbf{C_S}$.

$\Pr\{\mathbf{C_F}|\mathbf{C_S}\}$ is the extension of the notion of TOP event probability from fault trees to FDMs. For multistate systems, it is more convenient to use $\Pr\{\mathbf{C_F}|\mathbf{C_S}\}$ as probabilistic indicator since it can represent not only the probability of failed states but also the probability of other non-failed states.

Proposition 5.2.1. For any state variable $v_j \in \mathbf{S}$, the state probability $p_w(y)$ can be decomposed according to the valuations of v_j as follows:

$$p_w(y) = \sum_{c \in \text{dom}(v_j)} p_j(c) \cdot \Pr\{w = y|v_j = c\} \tag{5.26}$$

where p_j is the probability measure in $\text{dom}(v_j)$.

Example 5.2.3. Take again the model $\mathcal{M}$ in Eq.(5.3) as example. We can decompose the state probabilities $p_w(W)$, $p_w(D)$ and $p_w(F)$ according to the valuations of u as follows:

$$\begin{cases} p_w(W) & = & p_u(W) \cdot \text{Pr}\{w = W | u = W\} \\ & + & p_u(D) \cdot \text{Pr}\{w = W | u = D\} \\ & + & p_u(F) \cdot \text{Pr}\{w = W | u = F\} \\ \\ p_w(D) & = & p_u(W) \cdot \text{Pr}\{w = D | u = W\} \\ & + & p_u(D) \cdot \text{Pr}\{w = D | u = D\} \\ & + & p_u(D) \cdot \text{Pr}\{w = D | u = F\} \\ \\ p_w(F) & = & p_u(W) \cdot \text{Pr}\{w = F | u = W\} \\ & + & p_u(D) \cdot \text{Pr}\{w = F | u = D\} \\ & + & p_u(F) \cdot \text{Pr}\{w = F | u = F\} \end{cases} \tag{5.27}$$

5.2.3 Sensitivity analysis

Definition 5.2.1. Denote the *sensitivity* of $p_w(y)$ with respect to $p_j(c)$ ($p_j(c) \neq 0$) by $\mathbf{Sen}(w = y, v_j = c)$. It is defined as the following partial derivative:

$$\mathbf{Sen}(w = y, v_j = c) \overset{def}{=} \frac{\partial p_w(y)}{\partial p_j(c)} \tag{5.28}$$

Mathematically, $\mathbf{Sen}(w = y, v_j = c)$ quantifies the robustness of $p_w(y)$ under the variation of $p_j(c)$. If $\mathbf{Sen}(w = y, v_j = c)$ is large, it means that a small change of $p_j(c)$ will lead to a comparatively large change of $p_w(y)$.

According to the decomposition of $p_w(y)$ in Eq.(5.26), we can deduce that:

$$\frac{\partial p_w(y)}{\partial p_j(c)} = \sum_{x \in \text{dom}(v_j)} \left(\frac{\partial p_j(x)}{\partial p_j(c)} \cdot \text{Pr}\{w = y | v_j = x\} + p_j(x) \cdot \frac{\partial \text{Pr}\{w = y | v_j = x\}}{\partial p_j(c)} \right)$$

From Eq.(5.22), we can see that $\text{Pr}\{w = y | v_j = x\}$ is independent to p_j, i.e. $\forall c \in \text{dom}(v_j)$:

$$\frac{\partial \text{Pr}\{w = y | v_j = x\}}{\partial p_j(c)} = 0$$

Then,

$$\mathbf{Sen}(w = y, v_j = c) = \sum_{x \in \text{dom}(v_j)} \frac{\partial p_j(x)}{\partial p_j(c)} \cdot \text{Pr}\{w = y | v_j = x\} \tag{5.29}$$

For the sake of simplicity, denote the quotient $\frac{\partial p_j(x)}{\partial p_j(c)}$ by $\alpha_j(x, c)$, which we call the sensitivity coefficient of x by c for p_j. $\alpha_j(x, c)$ quantifies the variation of $p_j(x)$ with respect to the variation of $p_j(c)$, where c, x are the two states in dom(v_j).

The following equalities should hold for all α_j:

$$\left\{\begin{array}{lcl} \alpha_j(c,x) \cdot \alpha_j(x,c) & = & 1 \\ \alpha_j(c,c) & = & 1 \\ \sum_{c \in \text{dom}(v_j)} \alpha_j(c,x) & = & 0 \end{array}\right. \tag{5.30}$$

For the sake of simplicity, we use a matrix $\mathbf{A}_j$ (similar to the Jacobian matrix) to denote the sensitivity coefficients:

$$\mathbf{A}_{j,kl} = \alpha_j(c_k, c_l) = \frac{\partial p_j(c_k)}{\partial p_j(c_l)} \tag{5.31}$$

where $k, l = 1, 2, ..., |\text{dom}(v_j)|$ and $c_k, c_l \in \text{dom}(v_j)$.

Finally,

$$\mathbf{Sen}(w = y, v_j = c_l) = \sum_k \mathbf{A}_{j,kl} \cdot \text{Pr}\{w = y | v_j = c_k\} \tag{5.32}$$

Example 5.2.4. Take the Boolean domain as an example, i.e. dom$(v_j) = \mathbf{WF}$. Since $p_j(W) = 1 - p_j(F)$, the sensitivity coefficient matrix $\mathbf{A}_j^{\mathbf{WF}}$ for v_j is as follows:

$$\mathbf{A}_j^{\mathbf{WF}} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{5.33}$$

We also found that the sensitivity factor can be related to the importance measures, for instance, the Birnbaum importance measure.

Denote the Birnbaum importance measure of the jth component by $I^B(j)$. The classical meaning of $I^B(j)$ is the probability that the system is in a state where the jth component is critical (which means that if the jth component fails, the system will fail). Mathematically,

$$I^B(j) \overset{def}{=} \text{Pr}\{w = W | v_j = W\} - \text{Pr}\{w = W | v_j = F\}$$

where v_j is the state variable of the jth component.

According to Eq.(5.32), we can verify that

$$I^B(j) = \mathbf{A}_j^{\mathbf{WF}} \cdot \begin{bmatrix} \text{Pr}\{w = W | v_j = W\} \\ \text{Pr}\{w = W | v_j = F\} \end{bmatrix} = \mathbf{Sen}(w = W, v_j = W)$$

The other importance measures will be included in our future work for FDMs.

5.2.4 Approximation of probability by critical scenarios

Let v be a state variable. The component modeled by v is *reliable* means that the magnitude of the probability measure p_v in $\mathrm{dom}(v)$ satisfies the following condition:

$$\forall x, y \in \mathrm{dom}(v), x \sqsubseteq y \Rightarrow p_v(x) \gg p_v(y) \tag{5.34}$$

If all components are reliable, i.e. the above condition holds for all state variables, then the scenario probability $p_\otimes(\overline{\mathbf{v}})$ is concentrated in the minimal scenarios. Then, $p_w(y)$ can be approximated as follows:

$$p_w(y) \approx \sum_{\overline{\mathbf{v}} \in \mathrm{Min}\mathbf{Sce}(w=y)} p_\otimes(\overline{\mathbf{v}}) \tag{5.35}$$

Example 5.2.5. Take again the model $\mathcal{M}$ in Eq.(5.3) as example. If the probability measures p_u and p_v satisfy that $p_u(W) \gg p_u(D) \gg p_u(F)$ and $p_v(W) \gg p_v(D) \gg p_v(F)$, then the state probabilities $p_w(W)$, $p_w(D)$ and $p_w(F)$ can be approximated according to the sets of minimal scenarios in Eq.(5.12) as follows:

$$\begin{cases} p_w(W) & \approx & \sum_{(x,y) \in \mathrm{Min}\mathbf{Sce}(w=W)} p_u(x) p_v(y) = p_u(W) p_v(W) \\ p_w(D) & \approx & \sum_{(x,y) \in \mathrm{Min}\mathbf{Sce}(w=D)} p_u(x) p_v(y) = p_u(D) p_v(D) \\ p_w(F) & \approx & \sum_{(x,y) \in \mathrm{Min}\mathbf{Sce}(w=F)} p_u(x) p_v(y) = p_u(F) p_v(F) \end{cases} \tag{5.36}$$

5.3 Decision diagram based assessment

5.3.1 Decision diagrams

Decision diagrams are graphical representations of logic functions. For fault tree analysis, Binary Decision Diagrams (BDDs) are used to encode the valuation of Boolean functions. For FDMs assessment, decision diagrams are used to encode the valuation of the syntactic solution of flow variable.

The decision diagram used in this thesis can be seen as an extension of BDDs in multistate cases. The two types of nodes in the decision diagram are defined as follows:

– Each internal node is denoted by the quadruple (s, v, n_1, n_2), which is labeled by a state constant s and a variable constant v and has two out-edges pointing respectively to the child-nodes n_1 and n_2. n_1 and n_2 are called respectively the *then-child* and the *else-child*.

– Each terminal node is denoted by $(s, /, /, /)$, which is only labeled with a state constant s and has no child node.

Denote the decision diagram that encodes the valuation of a formula f by $\mathrm{DD}(f)$. According to the definition of $[\![f]\!]$ presented Section 4.2.1, the decision diagram $\mathrm{DD}(f)$ is interpreted as follows.

First, if f is reduced to a constant c, then $\mathrm{DD}(f)$ is made of only one terminal node $(c, /, /, /)$, meaning that $[\![f]\!](\sigma) = c, \forall\sigma$.

Second, if f is reduced to a variable v, then $\mathrm{DD}(f)$ is in the form of Figure 5.6. It encodes the valuation $[\![f]\!](\sigma) = \sigma(v) \in \mathrm{dom}(v)$. It is worth mentioning that in $\mathrm{DD}(v)$, we introduce a secondary ordering (different from the main variable ordering), which is the ordering of the state labels $s_1, s_2, ..., s_m$ in the chain of the variable nodes in $\mathrm{DD}(v)$. This secondary ordering can be used to improve the computational efficiency. But in the algorithms presented this thesis, this secondary ordering is arbitrary. More advanced algorithms can be found in our paper Rauzy and Yang (2019).

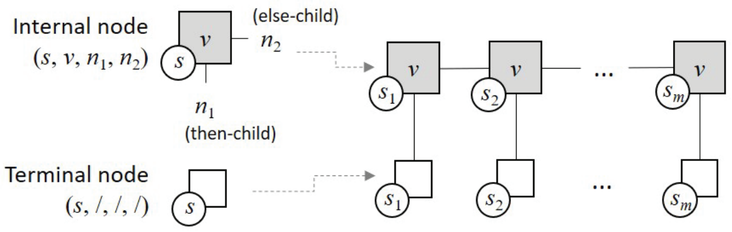


Figure 5.6: The decision diagram $\mathrm{DD}(v)$ with $\mathrm{dom}(v) = \{s_1, s_2, ..., s_m\}$.

Figure 5.7 shows the $\mathrm{DD}(v)$, where $\mathrm{dom}(v)$ is respectively the FDS $\mathbf{WF}$, $\mathbf{WDF}$, $\mathbf{SWF}$, $\mathbf{WFdFu}$ and $\mathbf{WFdFs}$.

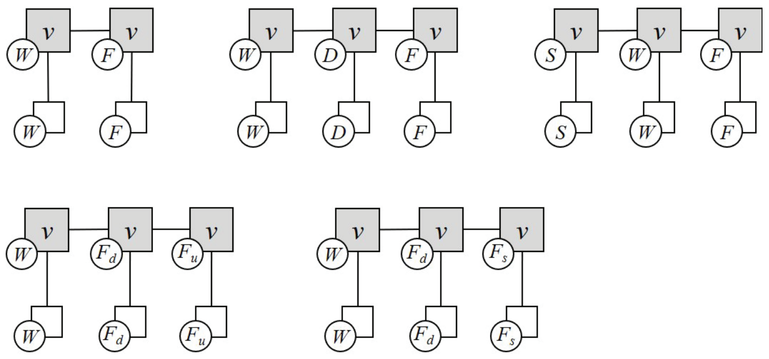


Figure 5.7: $\mathrm{DD}(v)$ where $\mathrm{dom}(v)$ is respectively $\mathbf{WF}$, $\mathbf{WDF}$, $\mathbf{SWF}$, $\mathbf{WFdFu}$ and $\mathbf{WFdFs}$.

Finally, if f is in the form of $\diamondsuit(f_1, ..., f_n)$, then DD$(f)$ encodes the valuation $[\![f]\!](\sigma) = [\![\diamondsuit]\!]([\![f_1]\!](\sigma), ..., [\![f_n]\!](\sigma))$. Consequently, DD$(f)$ is constructed recursively based on the sub-diagrams DD(f_1),...,DD(f_n) and the operation $[\![\diamondsuit]\!]$. The construction algorithm will be given Section 5.3.2.

Figure 5.8 shows the decision diagram of the formula $f = \diamondsuit(u, v)$. Assume that dom$(u) = \{t_1, t_2, ..., t_n\}$ and dom$(v) = \{s_1, s_2, ..., s_m\}$.

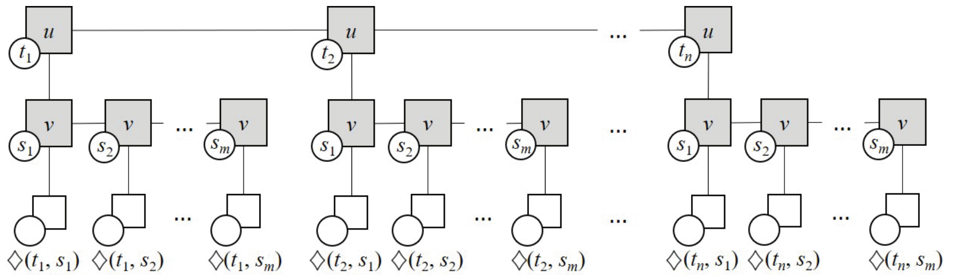


Figure 5.8: DD$(\diamondsuit(u, v))$.

In this decision diagram, the edges that go down the diagram (i.e. through the then-child of each internal node) form the valuation **paths**. A valuation path represents a possible valuation $\sigma(u, v) = (t_i, s_j), 1 \leq i \leq n, 1 \leq j \leq m$. The terminal node $(\diamondsuit(t_i, s_j), /, /, /)$ at the end of this path represents the valuation result of the formula $f = \diamondsuit(u, v)$, which is $\diamondsuit(t_i, s_j)$.

The ordering of variables in each valuation path is called the *variable ordering* of the decision diagram. In the sequel, we shall use the symbol $\prec$ for this variable ordering. For example, the variable ordering of the decision diagram in Figure 5.8 is $u \prec v$.

In this thesis, we suggest to use the depth-first-left-most (DFLM) traversal on the expression tree of f to automatically obtain the variable ordering of DD(f). In DFLM traversal of an expression tree, $v_1 \prec v_2$ means that the variable node labeled with v_1 locates in the down-side and left-side branch of the variable node labeled with v_2. For instance, the variable ordering obtained from the expression tree in Figure 4.1 is $v_1 \prec v_2 \prec v_3$.

5.3.2 Construction of decision diagrams

The algorithm of constructing decision diagram from an expression tree node `f` is given Figure 5.9. The function `BuildDD(f)` returns the root node of the required decision diagram.

```
function BuildDD(f)
    if IsVariableNode(f):
        return BuildDDForVariable(f)
    else: //f is operator node
        m = BuildDD(f.leftChild)
        n = BuildDD(f.rightChild)
        return Combine(f.operator,m,n)
```

Figure 5.9: Algorithm of `BuildDD`.

If `f` is a variable (terminal) node, i.e. $\texttt{f} = \langle v, /, / \rangle$, then `BuildDD(f)` returns the decision diagram $\mathrm{DD}(v)$ in the form of Figure 5.6.

If `f` is an operator node, i.e.

$$\texttt{f} = \langle \texttt{f.operator}, \texttt{f.leftChild}, \texttt{f.rightChild} \rangle,$$

then we first construct the decision diagrams — `m` and `n` — for the two child-nodes `f.leftChild` and `f.rightChild`, and then combine `m` and `n` according to the valuation of `f.operator` through the function `Combine`.

The algorithm of the function `Combine` is given Figure 5.10. This algorithm should be self-explanatory.

The algorithm of `Hook` is given Figure 5.11. The function `Hook` is applied to organize the internal node layers, i.e. to decide which internal node should locate more upward than another one according to the variable ordering $\prec$.

Example 5.3.1. Take the expression tree in Figure 5.12 as a concrete example. The variable ordering obtained by the DFLM traversal of this expression tree is $u \prec v \prec w$. The decision diagram of this expression tree is constructed by implementing the function `BuildDD(n0)`.

According to the algorithm given Figure 5.9, `BuildDD(n0)` is proceeded as follows:

1. First, since $n_0 = \langle \diamondsuit_1, n_1, n_2 \rangle$ is an internal node, the result of `BuildDD(n0)` is obtained by:

   $$\texttt{BuildDD(n0)} \leftarrow \texttt{Combine(}\diamondsuit_1\texttt{,BuildDD(n1),BuildDD(n2))}$$

2. Since $n_1 = \langle \diamondsuit_2, n_3, n_4 \rangle$ and $n_2 = \langle \diamondsuit_3, n_3, n_5 \rangle$ are also internal nodes, then:

   $$\texttt{BuildDD(n1)} \leftarrow \texttt{Combine(}\diamondsuit_2\texttt{,BuildDD(n3),BuildDD(n4))}$$

```
function Combine(operator,m,n)
    if m==nil or n==nil:
        return nil
    if IsInternalNode(m):
        if IsInternalNode(n):
            return Hook(operator,m,n)
        else: //n is a terminal node
            s = m.state
            v = m.variable
            n1 = Combine(operator,m.thenChild,n)
            n2 = Combine(operator,m.elseChild,n)
            return NewInternalNode(s,v,n1,n2)

    else: //m is a terminal node
        if IsInternalNode(n):
            s = n.state
            v = n.variable
            n1 = Combine(operator,m,n.thenChild)
            n2 = Combine(operator,m,n.elseChild)
            return NewInternalNode(s,v,n1,n2)
        else: //n is a terminal node
            return AssignValue(operator,m,n)
```

Figure 5.10: Algorithm of `Combine`.

```
function Hook(operator,m,n) // m, n are both internal nodes
    if m.variable.order <= n.variable.order:
        s = m.state
        v = m.variable
        if m.variable.order == n.variable.order:
            n1 = Combine(operator,m.thenChild,n.thenChild)
        else:
            n1 = Combine(operator,m.thenChild,n)
        n2 = Combine(operator,m.elseChild,n)
        return NewInternalNode(s,v,n1,n2)
    else:
        return Hook(operator,n,m)
```

Figure 5.11: Algorithm of `Hook`.

$$\texttt{BuildDD(n2)} \leftarrow \texttt{Combine(}\Diamond_3\texttt{,BuildDD(n3),BuildDD(n5))}$$

3. Since $n_3 = \langle u, /, / \rangle$, $n_4 = \langle v, /, / \rangle$ and $n_5 = \langle w, /, / \rangle$ are terminal nodes,

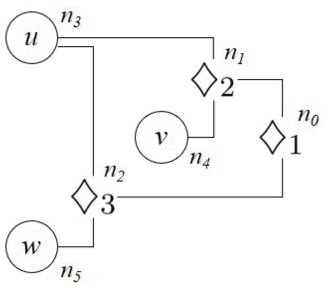


Figure 5.12: Expression tree example.

the results of `BuildDD(n3)`, `BuildDD(n4)` and `BuildDD(n5)` are the decision diagrams DD(u), DD(v) and DD(w) (see Figure 5.13) constructed by the function `BuildDDForVariable`. In this case, we assume that dom$(u) = \{x_1, ..., x_m\}$, dom$(v) = \{y_1, ..., y_n\}$ and dom$(w) = \{z_1, ..., z_l\}$.

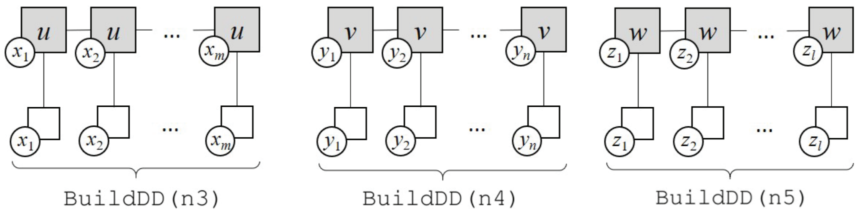


Figure 5.13: The results of `BuildDD(n3)`, `BuildDD(n4)` and `BuildDD(n5)`.

4. According to the results of `BuildDD(n3)`, `BuildDD(n4)` and `BuildDD(n5)`, `BuildDD(n1)` and `BuildDD(n2)` (in step 2) can be constructed through the function `Combine`. The results are given Figure 5.14. The states labeled in the terminal nodes are:

$$a_{ij} = \diamondsuit_2(x_i, y_j)$$

$$b_{ik} = \diamondsuit_3(x_i, z_k)$$

where $i = 1, ..., m$, $j = 1, ..., n$ and $k = 1, ..., l$.

5. Finally, `BuildDD(n0)` is constructed through the function `Combine`. The result is given Figure 5.15. The state labeled in the terminal node of the valuation path $\sigma(u, v, w) = (x_i, y_j, z_k)$ is:

$$c_{ijk} = \diamondsuit_1(a_{ij}, b_{ik})$$

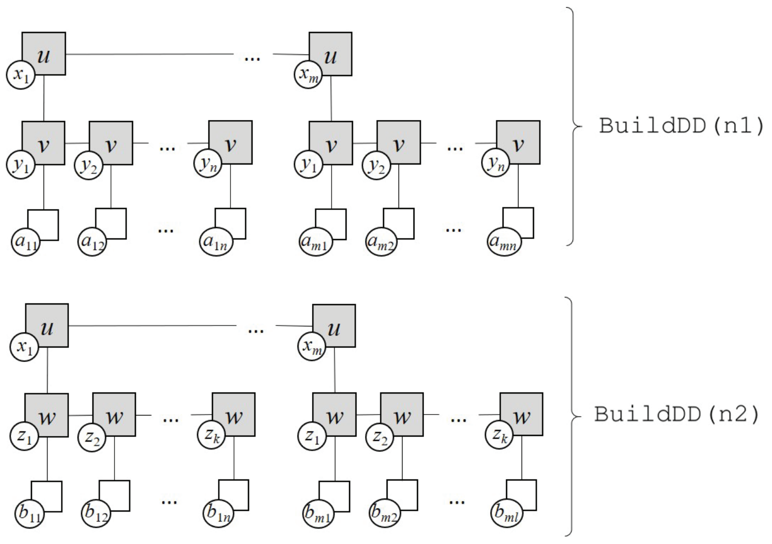


Figure 5.14: The results of `BuildDD(n1)` and `BuildDD(n2)`.

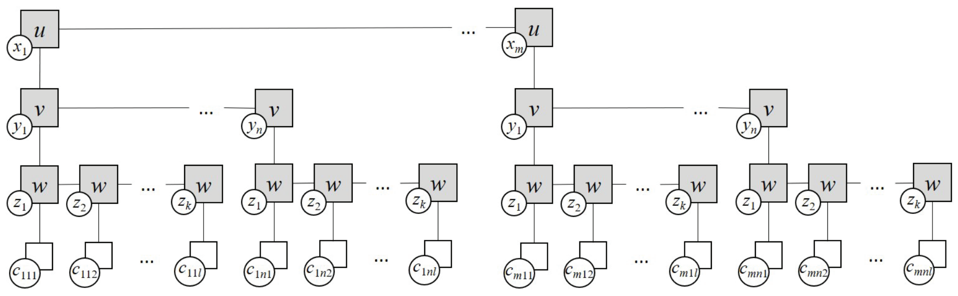


Figure 5.15: The result of `BuildDD(n0)`.

Once the decision diagram is constructed, it can be used to calculate the set of scenarios and probabilistic indicators.

5.3.3 Calculation of scenarios

Given an observer $w = y$, the set of scenarios $\mathbf{Sce}(w = y)$ can be obtained by the decision diagram DD(f_w), where f_w is the syntactic solution of w. It is worth noticing that the scenarios obtained by DD(f_w) contains only the valuation of state variables that w depends on.

The algorithm of calculating $\mathbf{Sce}(w = y)$ is given Figure 5.16. `n` is a node in DD(f_w), `y` is the target state in the observer $w = y$, `path` is the valuation path formed by the valuation of the state variables that have been already passed and

`Sce` is the result set of scenarios. The calculation of $\mathbf{Sce}(w = y)$ should start with `Scenarios(r,y,(),{})`, where `r` is the root node of $\text{DD}(f_w)$, `()` is the empty path and `{}` is the empty set.

```
function Scenarios(n,y,path,Sce)
    if IsTerminalNode(n):
        if y==n.state:
            add path to Sce
        return Sce
    else: # n is internal node
        add n.state to path
        Sce = Scenarios(n.downChild,y,path,Sce)
        if n.rightChild!=None:
            Sce = Scenarios(n.rightChild,y,Copy(path),Sce)
        return Sce
```

Figure 5.16: Algorithm of `Scenario`.

Example 5.3.2. Take the model in Eq.(5.3) as an example. The decision diagram $\text{DD}(f_w)$ is pictured Figure 5.17. The valuation paths that end in different terminal nodes are marked in different colors. These paths finally form the scenarios in $\mathbf{Sce}(w = W)$, $\mathbf{Sce}(w = D)$ and $\mathbf{Sce}(w = F)$.

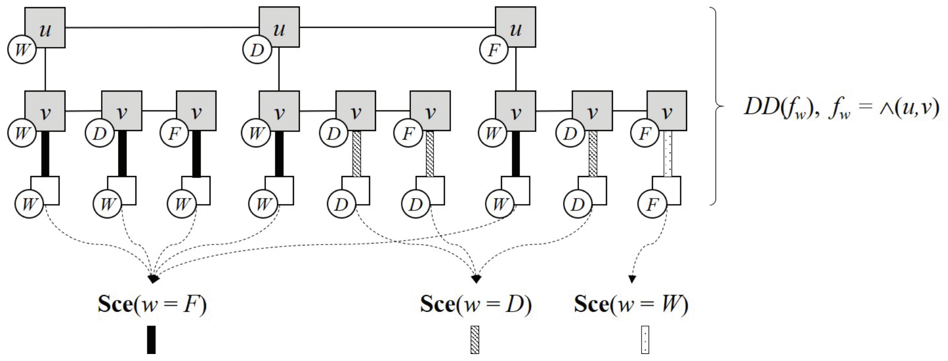


Figure 5.17: The three sets of scenarios $\mathbf{Sce}(w = W)$, $\mathbf{Sce}(w = D)$ and $\mathbf{Sce}(w = F)$ determined by $\text{DD}(f_w)$.

The conditional scenarios $\mathbf{Sce}(w = y|v_j = c)$ can be easily calculated from $\mathbf{Sce}(w = y)$ according to its definition (see Definition 5.1.2). The algorithm is given Figure 5.18.

In `ConditionalScenario(Sce,j,c)`, `Sce` stands for the original set of scenarios $\mathbf{Sce}(w = y)$, j is the index/order of v_j and `c` is the target state in $v_j = c$.

```
function ConditionalScenario(Sce,j,c)
    ConditionSce = {}
    for v in Sce:
        if v[j]==c:
            add v to ConditionSce
    return ConditionSce
```

Figure 5.18: Algorithm of `ConditionalScenario`.

Given a set of scenarios `Sce`, its minimal and maximal scenarios can also be easily calculated according to their definition (see Definition 5.1.4). The algorithms are given Figure 5.19.

```
function MinimalScenarios(Sce)
    MinSce = {}
    for u in Sce:
        IsMinimal = true
        for v in Sce:
            if IsLessDegraded(v,u):
                IsMinimal = false
                break
        if IsMinimal:
            add u to MinSce
    return MinSce

function MaximalScenarios(Sce)
    MaxSce = {}
    for u in Sce:
        IsMaximal = true
        for v in Sce:
            if IsLessDegraded(u,v):
                IsMaximal = false
                break
        if IsMaximal:
            add u to MaxSce
    return MaxSce
```

Figure 5.19: Algorithms of `MinimalScenarios` and `MaximalScenarios`.

The function `IsLessDegraded(u,v)` is used to compare the degradation order between the two scenarios `u` and `v`.

5.3.4 Calculation of probabilistic indicators

There are two ways of calculating the state probability $p_w(y)$. First, it can be directly calculated by DD(f_w). The algorithm is given Figure 5.20.

```
function Probability_DD(n,y)
    if IsTerminalNode(n):
        if n.state == y:
            return 1
        else:
            return 0
    else:
        p1 = Probability_DD(n.thenChild,y)
        p2 = Probability_DD(n.elseChild,y)
        p = StateProbability(n)
        return p*p1 + p2
```

Figure 5.20: Algorithm of `Probability_DD`.

In `Probability_DD(n,y)`, `n` is a node of the decision diagram and `y` is the state in $p_w(y)$. The calculation should start with `Probability_DD(r,y)` where `r` is the root node of DD(f_w). If $n = (s, v, n_1, n_2)$, then the function `StateProbability(n)` returns the probability $p_v(s)$, i.e. the probability of s in dom(v).

The second way of calculating $p_w(y)$ is to use the result of the set of scenarios **Sce**$(w = y)$. The algorithm is given Figure 5.21.

```
function Probability_Sce(Sce,model)
    p_Sce = 0
    for v in Sce:
        p_Sce = p_Sce + Probability(v,model)
    return p_Sce

function Probability(v,model)
    p_v = 1
    for i = 1:n :
        s = v[i]
        p_s = LookForProbability(i,s,model)
        p_v = p_v * p_s
    return p_v
```

Figure 5.21: Algorithm of `Probability_Sce`.

The function `Probability_Sce(Sce,model)` returns the sum of probabilities of the scenarios in `Sce`. In other words, if `Sce` is the set of conditional scenarios, `Probability_Sce(Sce,model)` will thus return the corresponding conditional probability.

To be concrete, the result of `Probability_Sce(Sce,model)` is:

– $p_w(y)$, if `Sce` $= \mathbf{Sce}(w = y)$.

– $p_j(c) \cdot \Pr\{w = y|v_j = c\}$, if `Sce` $= \mathbf{Sce}(w = y|v_j = c)$ and p_j is the probability measure defined on $\mathrm{dom}(v_j)$.

– $p_{\mathbf{S}} \cdot \Pr\{\mathbf{C_F}|\mathbf{C_S}\}$, if `Sce` $= \mathbf{Sce}(\mathbf{C_F}|\mathbf{C_S})$ and $p_{\mathbf{S}} = \prod_{i \in I_{\mathbf{S}}} p_i(c_i)$, c_i is the valuation of v_i in $\mathbf{C_S}$ and $I_{\mathbf{S}}$ is the index set of state variables in $\mathbf{C_S}$.

The function `Probability(v,model)` returns the scenario probability $p_{\otimes}(\overline{\mathbf{v}})$ (see Eq.(5.15). The object `model` is the instance of the class `Model` in Figure 6.2 that stores all the defined objects in the FDM, including the input probability measure in the domain of each state variable.

Part III

Application

Chapter 6

Implementation and Experiments

6.1 LatticeX: object-oriented implementation of FDMs

6.1.1 Software architecture

LatticeX is a small software that we developed to implement the modeling and the assessment of FDMs.

The functional architecture of **LatticeX** is given Figure 6.1. First, three main functions are classified: the compilation of input models, the modeling of FDMs and the assessment of FDMs.

The input of **LatticeX** is the FDM written in text file using the modeling language FDS-ML. This modeling language will be presented Section 6.1.2. To translate the textual model in **LatticeX**, a compiler is thus included.

Then, the software construct the FDM according to the input file. In this step, all relevant objects — operators, variables, equations and FDSs — are created and stored in the software. The FDM is encoded by means of expression trees.

According to the observer defined in the input file, the FDM is interpreted by means of decision diagrams. Finally, the software implements the calculation of required indicators following the algorithms presented Section 5.3 based on these decision diagrams.

The first version of **LatticeX** is developed in Python. The technique of object-oriented programming (OOP) is used. OOP uses classes and objects to create models. A *class* is an extensible program-code-template for creating objects (i.e. instances of class), providing initial values for member variables and implementa-

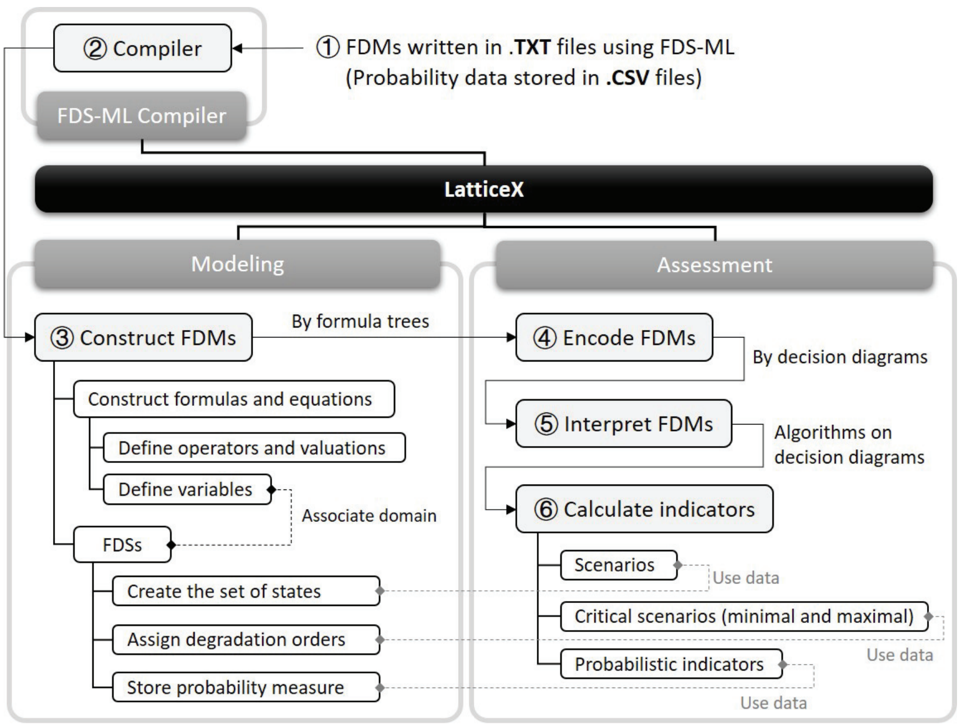


Figure 6.1: Functional architecture of **LatticeX**.

tions of member functions or methods (Bruce 2002). Comparing to the procedure-oriented programming (POP), OOP encapsulates functions (i.e. attributes and methods) into objects so that it is easy for developers to maintain and modify existing code.

The main classes in **LatticeX** are pictured Figure 6.2 using UML class diagram. There are in total 12 classes, classified into four categories: FDS, Formula, Decision Diagram and Model. The attributes of these classes can be found Appendix A.1.

6.1.2 Modeling language FDS-ML

FDS-ML is the abbreviation of Finite Degradation Structure - Modeling Language, which is a formal language designed for writing FDMs in text files. In this section, we will main illustrate the use of FDS-ML, while its detailed grammar can be found Appendix A.2.

Generally, a textual FDM is made of three parts: the declaration of *domains*, the

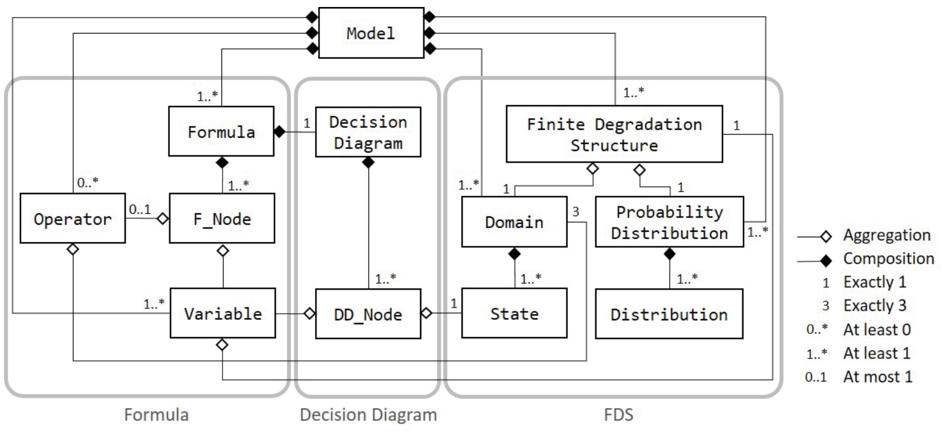


Figure 6.2: UML class diagram of the classes in **LatticeX**.

declaration of *operators* and the *model* of the system.

First, take the fault tree in Figure 5.2 as an example. The textual FDM of this fault tree is given Figure 6.3.

```
domain WF {W,F} (W<F)

operator OR(WF, WF) return WF
    W, W -> W
    *, F -> F
    F, * -> F
end

operator AND(WF, WF) return WF
    W, * -> W
    *, W -> W
    F, F -> F
end

block System
    WF a (W = 0.9,F = 0.1)
    WF b,c (W = 0.8,F = 0.2)
    assertion
        w := OR(AND(a,b),c)
    observer w = W
end
```

Figure 6.3: The fault tree in Figure 5.2 written in FDS-ML.

Domain declaration The domains that should be declared individually are the meet-semi-lattice structures of the valuation domains of state variables. The declaration begins with the keyword **`domain`**. There follow the name of the domain (e.g. `WF`), the set of states in the domain enclosed by braces (e.g. `{W,F}`) and the degradation orders enclosed by parentheses (e.g. `W<F`).

Figure 6.4 shows the declaration of the domains $\mathbf{WF}$, $\mathbf{WDF}$, $\mathbf{SWF}$, $\mathbf{WFdFu}$ and $\mathbf{WFdFs}$.

```
domain WF {W,F} (W<F)
domain WDF {W,D,F} (W<D,D<F)
domain SWF {S,W,F} (S<W,W<F)
domain WFdFu {W,Fd,Fu} (W<Fd,W<Fu)
domain WFdFs {W,F_dang,F_safe} (W<F_dang,W<F_safe)
```

Figure 6.4: The declaration of $\mathbf{WF}$, $\mathbf{WDF}$, $\mathbf{SWF}$, $\mathbf{WFdFu}$ and $\mathbf{WFdFs}$ in FDS-ML.

Operator declaration The operator declaration is exemplified in Figure 6.3 line 3 – 13. The declaration begins with the keyword **`operator`**. There follow the name of the operator (e.g. `OR`), the names of the two input domains enclosed by parentheses (e.g. `(WF,WF)`) and the name of the output domain right after the keyword **`return`**. The body of the operator declaration is a list of statements indicating the valuation mappings of this operator. For instance, a valuation $\Diamond(x, y) = z$ of the operator $\Diamond$ is written as `x,y -> z`. The symbol $\star$ matches any value in the corresponding domain.

Model The model is exemplified in Figure 6.3 line 15 – 21. It begins with the keyword **`block`** and there follows the name of the model (e.g. `System`). The lines right after are the declaration of state variables. A state variable v whose valuation domain is named as `DomainName` is declared in the following form:

```
DomainName v (...)
```

The content in the parentheses is the probability measure in dom(v). The probability should be assigned state by state. In the current version of FDS-ML, we provide two ways of assigning state probabilities. First, if the state probability is a constant value, it can be directly written using real numbers, e.g.

```
WF v_1 (W = 0.99,F = 0.01)
```

If the state probability is time-dependent, it can be stored in CSV file and uploaded to the software, e.g.

```
WF v_2 (W = 'ProbFile_W.csv',F = 'ProbFile_F.csv')
```

Once the state variables are declared, the equations can be written in the part of **`assertion`**. Notice that only state variables should be declared before used in the equations. Each equation $w \coloneqq f$ is written in the same form `w := f`. The names of operators in `f` should be consistent to those declared before. Finally, the observer is declared after the keyword **`observer`**.

Figure 6.5 shows a more generic form of the textual FDM.

```
domain A (a_1,a_2,...) (a_1<a_2,...)
domain B (b_1,b_2,...) (b_1<b_2,...)
...

operator Op_1 (A,A) return A
    a_1,a_1 -> a_1
    a_1,a_2 -> a_2
    ...
end

operator Op_2 (A,B) return A
    a_1,b_1 -> a_1
    a_1,b_2 -> a_2
    ...
end
...

block ModelOfSystem
    A v_1 (a_1 = ..., a_2 = ..., ...)
    A v_2 (a_1 = ..., a_2 = ..., ...)
    B v_3 (b_1 = ..., b_2 = ..., ...)
    ...
    assertion
        w_1 := Op_1(v_1,v_2)
        w_2 := Op_2(w_1,v_3)
        ...
    observer w = d
end
```

Figure 6.5: Generic form of textual FDM.

6.1.3 Construction of expression trees

In this section, we present the method of parsing the equations written in FDS-ML to construct the expression tree of the model.

In FDS-ML, equations are written in (quasi) Polish notation. Therefore, the construction of expression tree from a given set of equations is similar to parsing expressions in Polish notation using stacks. The parsing consists of two steps:

1. Split objects (i.e. tokens) by parentheses and comma, and put them into the stack.
2. Pop out the tokens three-by-three each time when meet a right parenthesis ")" to form the operator node.

Example 6.1.1. Take the equation `w := OR(AND(a,b),c)` in Figure 6.3 as an example. The tokens in the stack before the first right parenthesis are given below:

```
 b
 a
AND
OR
```

When meet the first ")", the three topside tokens `b`, `a` and `AND` are popped out, forming the operator node $n_3 = \langle \texttt{AND}, n_1, n_2\rangle$, where $n_1 = \langle a, /, /\rangle$ and $n_2 = \langle b, /, /\rangle$. Once we form an operator node, its label (e.g. `n_3`) should be put into the stack.

Then, we continue parsing the rest expression. The tokens in the stack before the second right parenthesis are given below:

```
 c
n_3
OR
```

When meet the second ")", these three tokens are popped out, forming the operator node $n_5 = \langle \texttt{OR}, n_3, n_4\rangle$, where $n_4 = \langle c, /, /\rangle$.

Finally, when the parsing is finished, n_5 is returned as the root node of this local expression tree and associated to the flow variable `w`.

6.2 Modeling library

In this section, we provide a small library of typical operators that have been applied in fault trees and reliability block diagrams.

6.2.1 Boolean logic operators

In Boolean logic, we have three classical operations: conjunction, disjunction and negation.

The *conjunction* of a set of operands is true if and only if all of its operands are true. This operation is interpreted as the AND-gate in fault trees and interpreted as the parallel composition in reliability block diagrams. In FDMs, it is interpreted as the abstraction $\wedge : \mathbf{WF}^2 \twoheadrightarrow \mathbf{WF}$.

The *disjunction* of a set of operands is true if and only if one or more of its operands is true. It is interpreted as the OR-gate in fault trees and interpreted as the series composition in reliability block diagrams. In FDMs, it is interpreted as the abstraction $\vee : \mathbf{WF}^2 \twoheadrightarrow \mathbf{WF}$.

The *negation* produces a value of true when its operand is false and a value of false when its operand is true. It is interpreted as the NOT-gate in fault trees. In FDMs, it is interpreted as the abstraction $\neg : \mathbf{WF} \twoheadrightarrow \mathbf{WF}$.

Figure 6.6 pictures the valuation of these operations under the framework $\langle \mathbf{FDS}, \otimes, \Phi \rangle$. It is easy to verify that $\wedge$ and $\vee$ are strongly-coherent, while $\neg$ is not coherent.

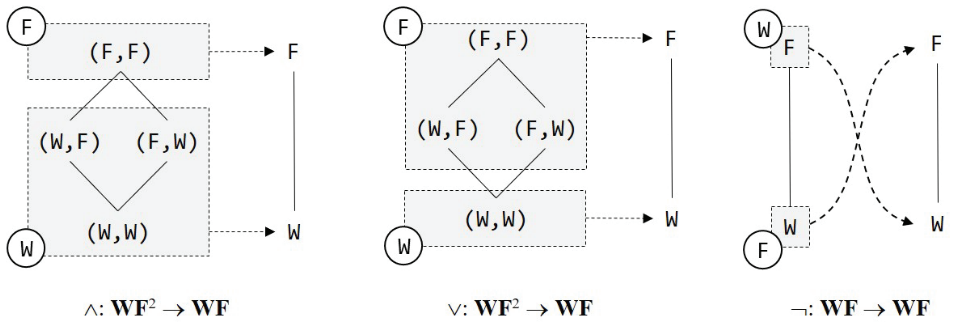


Figure 6.6: Illustration of the logic operations $\wedge$, $\vee$ and $\neg$.

6.2.2 Meet and join

In fact, the above Boolean conjunction and disjunction operators can be generalized to the meet and join operators, which are originally defined to obtain the infimum and supremum of two elements in an ordered set.

Definition 6.2.1 (Meet and join)**.** Let $\mathbf{n}$ be a linearly-ordered FDS (see Definition 3.2.3), then:

- the *meet* operation over $\mathbf{n}$ is defined as $\wedge : \mathbf{n} \otimes \mathbf{n} \twoheadrightarrow \mathbf{n}$ such that $\forall x, y \in \mathbf{n}$, $\wedge(x, y) = x$ if $x \sqsubseteq y$ and $\wedge(x, y) = y$ if $y \sqsubseteq x$.

- the *join* operation over $\mathbf{n}$ is defined as $\vee : \mathbf{n} \otimes \mathbf{n} \twoheadrightarrow \mathbf{n}$ such that $\forall x, y \in \mathbf{n}$, $\vee(x, y) = y$ if $x \sqsubseteq y$ and $\vee(x, y) = x$ if $y \sqsubseteq x$.

It is easy to verify that $\wedge$ and $\vee$ are strongly-coherent operators. The FDMs written over $\wedge$ and $\vee$ are always coherent.

In reliability and safety models, the two operations — $\wedge(x, y)$ and $\vee(x, y)$ — stand respectively for the "optimistic" and the "pessimistic" viewpoint of how we consider the state combination (x, y). For instance, consider the case that there are two components in the system: one is failed and another is working. If we consider that the system is still working, then we can use the meet operator, i.e. $\wedge(W, F) = W$. If we consider that the system is already failed, then we can use the join operator, i.e. $\vee(W, F) = F$. Such optimistic and pessimistic viewpoints extend the notion of "AND/OR" and "parallel/series connection" in multistate cases.

Figure 6.7 pictures the valuation of $\wedge$ and $\vee$ applied on the ternary FDS $\mathbf{WDF}$.

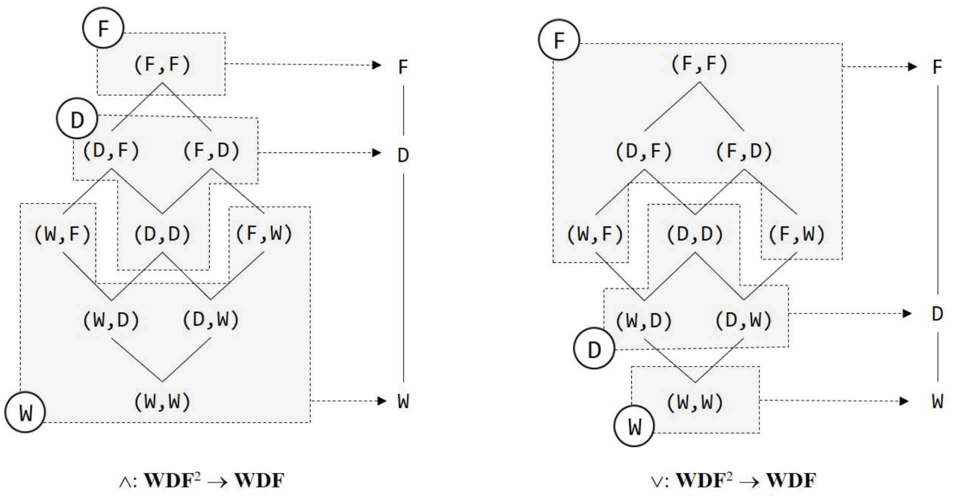


Figure 6.7: Illustration of the valuation of $\wedge$ and $\vee$ applied on the ternary FDS $\mathbf{WDF}$.

Moreover, we can deduce that $\wedge$ and $\vee$ are commutative, associative and distributive, i.e. the following equalities hold $\forall x, y, z \in \mathbf{n}$:

$$\begin{aligned} \wedge(x,y) &= \wedge(y,x) \\ \vee(x,y) &= \vee(y,x) \\ \wedge(\wedge(x,y),z) &= \wedge(x,\wedge(y,z)) \\ \vee(\vee(x,y),z) &= \vee(z,\vee(y,z)) \\ \wedge(\vee(x,y),z) &= \vee(\wedge(x,z),\wedge(y,z)) \\ \vee(\wedge(x,y),z) &= \wedge(\vee(x,z),\vee(y,z)) \end{aligned} \tag{6.1}$$

6.2.3 k-out-of-n

A k-out-of-n configuration means that there are n Boolean components in the system and the system is working if at least k components are working; otherwise, the system is failed.

The k-out-of-n ($k$oo$n$) operator for Boolean systems can be defined recursively by the meet $\wedge$ and the join $\vee$ applied on the binary FDS $\mathbf{WF}$.

Definition 6.2.2 ($k$oo$n$)**.** Denote the $k$oo$n$ operation by $\mathcal{O}_n^k : \mathbf{WF}^n \twoheadrightarrow \mathbf{WF}$, $1 \leq k \leq n$. It can be defined recursively as follows:

$$\mathcal{O}_n^k(v_1, ..., v_n) = \begin{cases} \mathcal{O}_{n-1}^{k-1}(v_2, ..., v_n), & k > 1, v_1 = W \\ \vee(v_1, \mathcal{O}_{n-1}^{k}(v_2, ..., v_n)), & k > 1, v_1 = F \\ \wedge(...(\wedge(v_1, v_2), ...), v_n), & k > 1, k = n \\ \vee(...(\vee(v_1, v_2), ...), v_n), & k = 1 \end{cases} \tag{6.2}$$

The $k$oo$n$ operator is a strongly-coherent operator since $\vee$ and $\wedge$ are strongly-coherent operators.

Figure 6.8 illustrates the $k$oo$n$ operation in the case that $n = 3$. We can see in this figure that the value of k determines where locates the boundary between the working and the failed state of the system.

6.2.4 Dependent components

In combinatorial models, components at bottom level are assumed to be stochastically independent. For this reason, it is unable to represent faithfully cold redundancies, time dependencies, resource sharing, reconfiguration, etc.

However, inspired by the Boolean logic driven Markov processes and reliability block diagrams driven Petri nets (see Section 2.2.2), we provide a solution of taking into account stochastically dependent components in FDMs. The idea is pictured Figure 6.9.

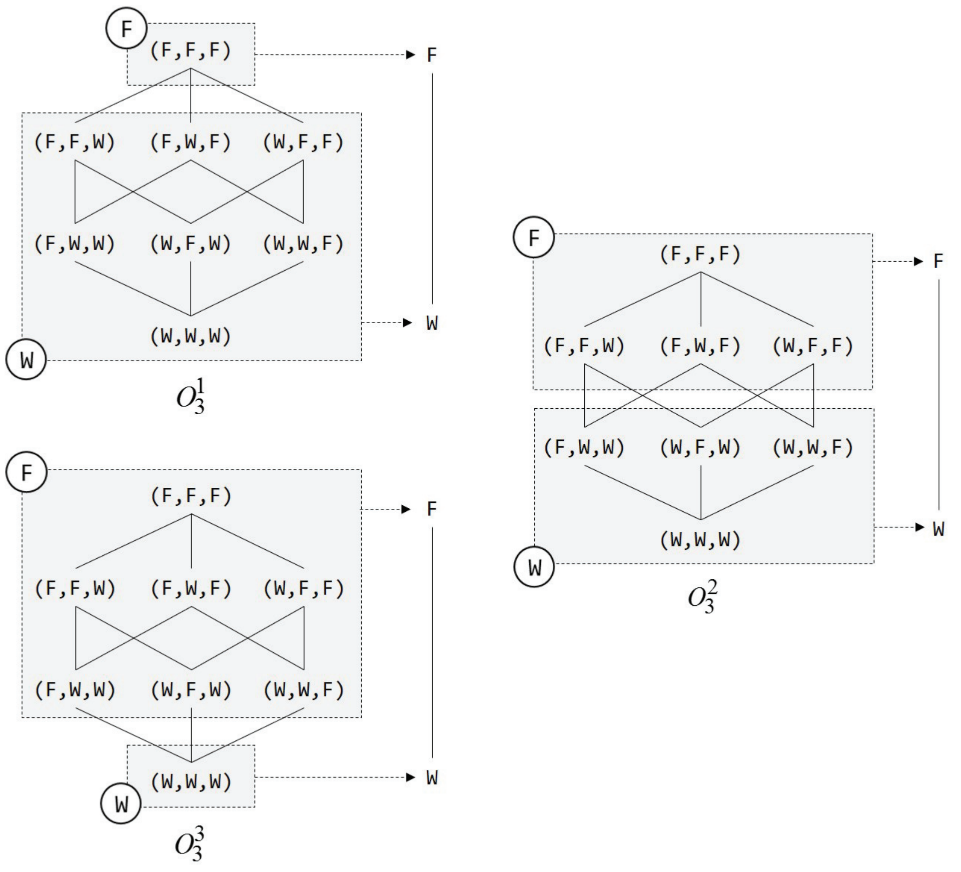


Figure 6.8: Illustration of $\mathcal{O}_3^1$, $\mathcal{O}_3^2$ and $\mathcal{O}_3^3$.

In Figure 6.9, the hierarchical structure inside the triangle is the expression tree of a FDM. The nodes at bottom level of the expression tree are the state variables representing the states of components. If these state variables are (stochastically) independent, probability measures can be directly assigned to their valuation domains and propagate to high abstraction level objects through the operations. However, if there are state variables that are stochastically dependent, the probability measure cannot be assigned to their valuation domains separately. Instead, it can be assigned to the product of their domains, see Figure 6.9 red arrow. Such probability can also propagate to high abstraction level objects to support the calculation of probabilistic indicators.

We will use a hot-standby system to illustrate the modeling idea pictured Figure 6.9.

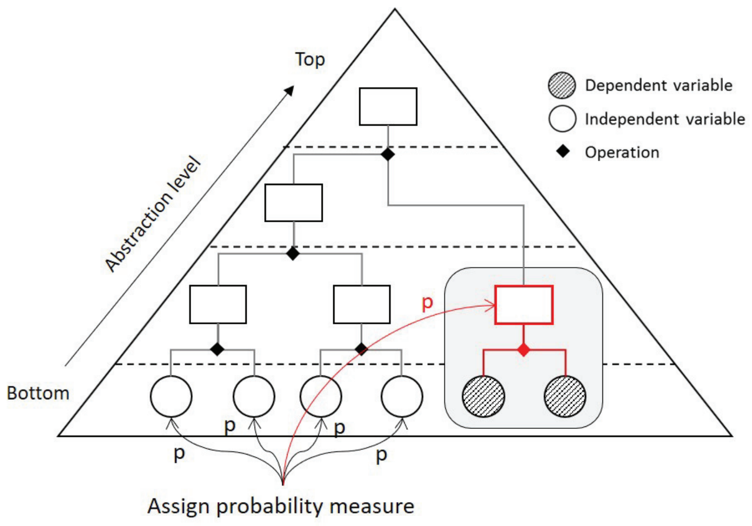


Figure 6.9: Idea of taking into account stochastically dependent components in FDMs.

Standby or redundancy is a technique widely used to enhance system reliability and availability. In general, there are three types in standby, i.e. cold, hot and warm standby. Cold standby implies that the spare units cannot fail while they are waiting. Hot standby implies that a spare unit undergoes the same operational environment as the master unit. Warm standby corresponds to an intermediate case and a spare unit undergoes milder operational environment than the master unit. In different cases, the life time of the spare unit may differ from the master unit, i.e. they are time-dependent.

A hot-standby system is pictured Figure 6.10. It consists of a master unit A, a standby unit B and a perfect switch S. The master unit A can be in two states: working or failed. The standby unit B can be in three states: standby, working or failed. The system is failed only if both units are failed; otherwise, the system is working.

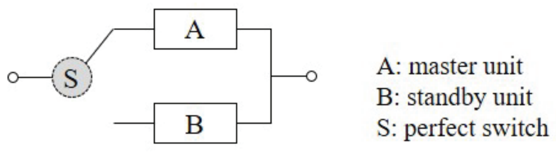


Figure 6.10: A hot-standby system consisting of a master unit A, a standby unit B and a perfect switch S.

We shall make the following assumptions:

– The switch is perfect and failure-free. It means that the failure of A can be detected immediately and then B is activated with probability 1.

– The time to failure of both A and B is exponentially distributed with the failure rates λ_A and λ_B.

– The repair is scheduled only if both A and B are failed. It repairs both A and B as good as new. The time to repair is also exponentially distributed with the repair rate μ.

The modeling procedure of such system is pictured Figure 6.11.

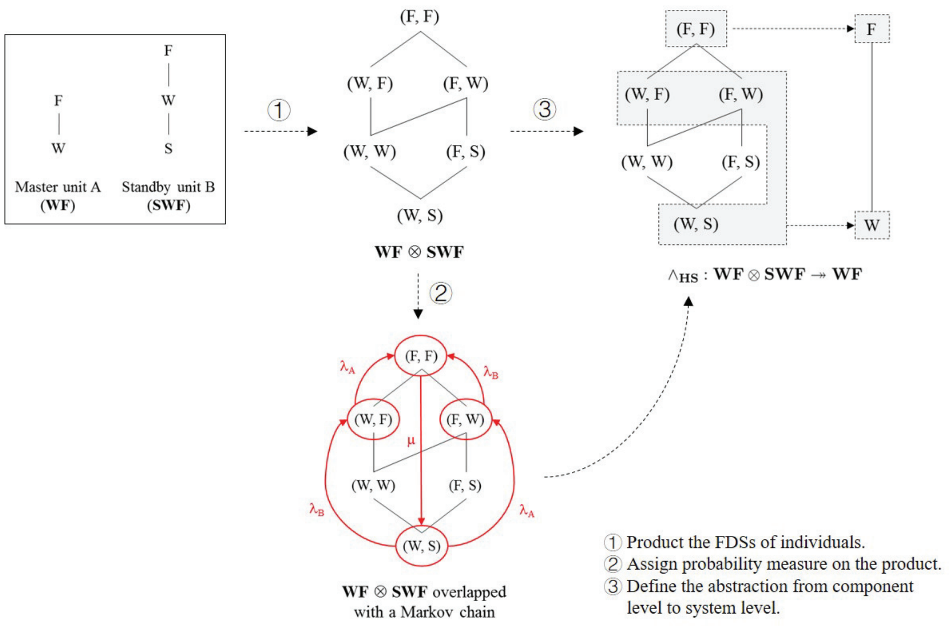

F
W
Master unit A (WF)
F
W
S
Standby unit B (SWF)
①
(F, F)
(W, F)
(F, W)
(W, W)
(F, S)
(W, S)
WF ⊗ SWF
②
③
F
W
∧HS : WF ⊗ SWF ↠ WF
λA
λB
μ
WF ⊗ SWF overlapped with a Markov chain
① Product the FDSs of individuals.
② Assign probability measure on the product.
③ Define the abstraction from component level to system level.

Figure 6.11: The procedure of abstracting a hot-standby system in FDMs.

First, the states of unit A and B are modeled respectively by the FDSs $\mathbf{WF}$ and $\mathbf{SWF}$. Since A and B are stochastically dependent, the probability measure should be assigned to the product $\mathbf{WF} \otimes \mathbf{SWF}$ not assigned to $\mathbf{WF}$ and $\mathbf{SWF}$ separately.

To calculate the probability measure on $\mathbf{WF} \otimes \mathbf{SWF}$, a Markov chain model is used, which is marked in red in Figure 6.11 step 2. Moreover, since the two states (W, W) and (F, S) are never reachable (under the assumption of perfect switch), their probability is zero for all time.

Finally, according to the functionality of this system, the operation between A and B is defined as $\wedge_{\mathbf{HS}} : \mathbf{WF} \otimes \mathbf{SWF} \twoheadrightarrow \mathbf{WF}$, whose valuation is pictured Figure 6.11.

Therefore, the hot-standby system of A and B can be modeled using the equation $HS := \wedge_{\mathbf{HS}}(v_A, v_B)$, where $\mathrm{dom}(v_A) = \mathbf{WF}$ and $\mathrm{dom}(v_B) = \mathbf{SWF}$. A case study made of such hot-standby subsystems can be found Section 6.3.2.

Through the operation $\wedge_{\mathbf{HS}}$, the state probabilities $p_{HS}(y)$, $y \in \mathrm{dom}(HS)$, are calculated as follows:

$$\begin{cases} p_{HS}(W) & = & p(W,S) + p(W,F) + p(F,W) \\ p_{HS}(F) & = & p(F,F) \end{cases}$$

where p is obtained by the Markov chain pictured Figure 6.11.

6.3 Experiments

6.3.1 Safety instrumented system

In this section, we provide the full assessment of the safety instrumented system presented Section 2.3.

According to the assumptions made in Section 2.3, the valuation of logic solvers and valves should be restricted by the following conditions, i.e. logic solves cannot be in F_{du} state and valves cannot be in F_{dd} state:

$$\mathbf{C_S} : \left\{ \begin{array}{c} LS1 \neq F_{du} \\ LS2 \neq F_{du} \\ V1 \neq F_{dd} \\ V2 \neq F_{dd} \end{array} \right\} \tag{6.3}$$

The model used in the assessment is the one in Eq.(4.10). Its textual model can be found Appendix A.3.

Safety channel 1 and 2

First, we implement the assessment of the two safety channels in the safety instrumented system.

The safety channel 1 (SC1) and the safety channel 2 (SC2) are pictured Figure 4.2. SC1 is made of a sensor $S1$, a logic solver $LS1$ and a valve $V1$, and SC2 is made of two sensors $S2, S3$, a logic solver $LS2$ and two valves $V1, V2$.

The analysis of safety channel 1 is targeted by the flow variable $SC1$. The observers are therefore: $SC1 = W$, $SC1 = F_s$, $SC1 = F_{dd}$ and $SC1 = F_{du}$.

Table 6.1: Number of scenarios and critical scenarios for safety channel 1.

y	W	F_s	F_{dd}	F_{du}
$\lvert \mathbf{Sce}(SC1 = y \vert \mathbf{C_S}) \rvert$	1	17	15	3
$\lvert \text{Min}\mathbf{Sce}(SC1 = y \vert \mathbf{C_S}) \rvert$	1	3	2	2
$\lvert \text{Max}\mathbf{Sce}(SC1 = y \vert \mathbf{C_S}) \rvert$	1	4	4	1

The numbers of scenarios and critical scenarios for $SC1$ are given Table 6.1.

The critical scenarios are listed below:

$$\begin{array}{lcl}
\text{Min}\mathbf{Sce}(SC1 = W|\mathbf{C_S}) & = & \{(W, W, W)\} \\
\text{Min}\mathbf{Sce}(SC1 = F_s|\mathbf{C_S}) & = & \{(F_s, W, W), (W, F_s, W), (W, W, F_s)\} \\
\text{Min}\mathbf{Sce}(SC1 = F_{dd}|\mathbf{C_S}) & = & \{(F_{dd}, W, W), (W, F_{dd}, W)\} \\
\text{Min}\mathbf{Sce}(SC1 = F_{du}|\mathbf{C_S}) & = & \{(W, W, F_{du}), (F_{du}, W, W)\} \\
 & & \\
\text{Max}\mathbf{Sce}(SC1 = W|\mathbf{C_S}) & = & \{(W, W, W)\} \\
\text{Max}\mathbf{Sce}(SC1 = F_s|\mathbf{C_S}) & = & \{(F_{du}, F_{dd}, F_s), (F_s, F_{dd}, F_s), (F_s, F_s, F_s), (F_{du}, F_s, F_s)\} \\
\text{Max}\mathbf{Sce}(SC1 = F_{dd}|\mathbf{C_S}) & = & \{(F_{du}, F_{dd}, F_{du}), (F_s, F_{dd}, F_{du}), (F_s, F_s, F_{du}), (F_{du}, F_s, F_{du})\} \\
\text{Max}\mathbf{Sce}(SC1 = F_{du}|\mathbf{C_S}) & = & \{(F_{du}, W, F_{du})\}
\end{array}$$

For each scenario (x, y, z), the valuation order is the same with the variable ordering in DD(f_{SC1}), i.e. $S1 \prec LS1 \prec V1$, such that $x = \sigma(S1)$, $y = \sigma(LS1)$ and $z = \sigma(V1)$.

The analysis of safety channel 2 is targeted by the flow variable $SC2$. The observers are therefore: $SC2 = W$, $SC2 = F_s$, $SC2 = F_{dd}$ and $SC2 = F_{du}$.

The numbers of scenarios and critical scenarios for $SC2$ are given Table 6.2.

Table 6.2: Number of scenarios and critical scenarios for safety channel 2.

y	W	F_s	F_{dd}	F_{du}
$\lvert \mathbf{Sce}(SC2 = y \vert \mathbf{C_S}) \rvert$	15	309	91	17
$\lvert \text{Min}\mathbf{Sce}(SC2 = y \vert \mathbf{C_S}) \rvert$	1	5	4	3
$\lvert \text{Max}\mathbf{Sce}(SC2 = y \vert \mathbf{C_S}) \rvert$	4	24	8	1

The critical scenarios are listed below (for those sets whose number of scenarios is larger than four, we only list their first four scenarios):

$$
\begin{aligned}
\textbf{MinSce}(SC2 = W|\mathbf{C_S}) &= \{(W, W, W, W, W)\}\\
\textbf{MinSce}(SC2 = F_s|\mathbf{C_S}) &= \{(W, W, W, W, F_s), (W, W, W, F_s, W),\\
&\quad (W, W, F_s, W, W), (W, F_s, W, W, W),\\
&\quad (F_s, W, W, W, W)\}\\
\textbf{MinSce}(SC2 = F_{dd}|\mathbf{C_S}) &= \{(F_{dd}, F_{dd}, W, W, W), (W, W, F_{dd}, W, W),\\
&\quad (W, F_s, W, F_{du}, F_{du}), (F_s, W, W, F_{du}, F_{du})\}\\
\textbf{MinSce}(SC2 = F_{du}|\mathbf{C_S}) &= \{(W, W, W, F_{du}, F_{du}), (F_{dd}, F_{du}, W, W, W),\\
&\quad (F_{du}, F_{dd}, W, W, W)\}
\end{aligned}
$$

$$
\begin{aligned}
\textbf{MaxSce}(SC2 = W|\mathbf{C_S}) &= \{(F_{du}, W, W, W, F_{du}), (F_{du}, W, W, F_{du}, W),\\
&\quad (W, F_{du}, W, W, F_{du}), (W, F_{du}, W, F_{du}, W)\}\\
\textbf{MaxSce}(SC2 = F_s|\mathbf{C_S}) &= \{(F_{du}, F_{du}, F_{dd}, F_{du}, F_s), (F_{du}, F_{du}, F_{dd}, F_s, F_{du}),\\
&\quad (F_{du}, F_{du}, F_{dd}, F_s, F_s), (F_{du}, F_{du}, F_s, F_{du}, F_s), \dots\}\\
\textbf{MaxSce}(SC2 = F_{dd}|\mathbf{C_S}) &= \{(F_{du}, F_{du}, F_{dd}, F_{du}, F_{du}), (F_{du}, F_{du}, F_s, F_{du}, F_{du}),\\
&\quad (F_{du}, F_s, F_{dd}, F_{du}, F_{du}), (F_{du}, F_s, F_s, F_{du}, F_{du}), \dots\}\\
\textbf{MaxSce}(SC2 = F_{du}|\mathbf{C_S}) &= \{(F_{du}, F_{du}, W, F_{du}, F_{du})\}
\end{aligned}
$$

Similarly, the valuation order of each scenario (a, b, c, d, e) is the same with the variable ordering in DD(f_{SC2}), i.e. $S2 \prec S3 \prec LS2 \prec V1 \prec V2$, such that $a = \sigma(S2)$, $b = \sigma(S3)$, $c = \sigma(LS2)$, $d = \sigma(V1)$ and $e = \sigma(V2)$.

As quantitative results, the state probabilities for $SC1$ and $SC2$ are pictured Figure 6.12. Their average probabilities with 8760 hours are given Table 6.3.

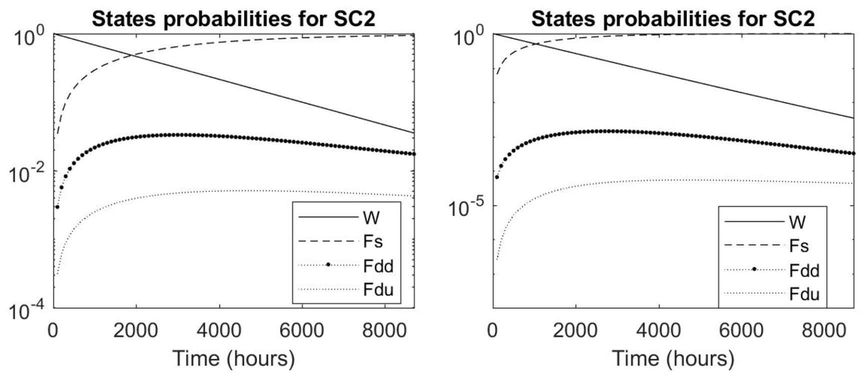


Figure 6.12: State probabilities for $SC1$ and $SC2$.

System-level analysis

The system-level analysis is targeted by the flow variable $System$ (see Eq.(4.10). The observers are therefore: $System = W$, $System = F_s$, $System = F_{dd}$ and $System = F_{du}$.

Table 6.3: Average probabilities for p_{SC1} and p_{SC2} within 8760 hours.

y	W	F_s	F_{dd}	F_{du}
$p_{SC1,avg}(y)$	0.2915	0.6792	0.0251	0.0042
$p_{SC2,avg}(y)$	0.1756	0.8235	8.6355×10^{-4}	4.0485×10^{-5}

Case 1 In this case, we use the conditions in Eq.(6.3) to restrict the valuation of state variables.

The numbers of scenarios and critical scenarios for $System$ are given Table 6.4.

Table 6.4: Number of scenarios and critical scenarios for $System$.

y	W	F_s	F_{dd}	F_{du}
$\lvert \mathbf{Sce}(System = y \lvert \mathbf{C_S}) \rvert$	170	3958	740	316
$\lvert \text{Min}\mathbf{Sce}(System = y \lvert \mathbf{C_S}) \rvert$	1	7	12	9
$\lvert \text{Max}\mathbf{Sce}(System = y \lvert \mathbf{C_S}) \rvert$	16	96	32	12

Since it is not possible to list all the critical scenarios here, we choose the first five scenarios in $\text{Max}\mathbf{Sce}(System = W|\mathbf{C_S})$ and $\text{Min}\mathbf{Sce}(System = F_{dd}|\mathbf{C_S})$ as example and list them below:

$$\begin{aligned}
\text{Max}\mathbf{Sce}(System = W|\mathbf{C_S}) \quad = \quad & \{(W, W, W, F_{du}, F_s, F_{dd}, F_{dd}), \\
& (W, W, W, F_s, F_s, F_{dd}, F_{du}), \\
& (W, W, W, F_s, F_{du}, F_{dd}, F_{du}), \\
& (W, W, W, F_{du}, F_{du}, F_{dd}, F_{du}), \\
& (F_{du}, F_{dd}, F_{du}, W, F_{du}, W, W), ...\}
\end{aligned}$$

$$\begin{aligned}
\text{Min}\mathbf{Sce}(System = F_{dd}|\mathbf{C_S}) \quad = \quad & \{(F_{dd}, W, W, W, W, F_{dd}, W), \\
& (F_s, W, F_{du}, F_s, W, W, F_{du}), \\
& (F_{dd}, W, W, F_{dd}, F_{dd}, W, W), \\
& (F_s, W, F_{du}, W, F_s, W, F_{du}), \\
& (F_{dd}, W, F_{du}, F_s, W, W, F_{du}), ...\}
\end{aligned}$$

Similar to the previous cases, the valuation order in each scenario above is the same with the variable ordering in DD($System$), i.e. $S1 \prec LS1 \prec V1 \prec S2 \prec S3 \prec LS2 \prec V2$.

First, we can observe that the combinations of different failure modes appear in these critical scenarios, e.g. $(W, W, W, F_{du}, F_s, F_{dd}, F_{dd})$ in $\text{Max}\mathbf{Sce}(System = W|\mathbf{C_S})$ and $(F_s, W, F_{du}, F_s, W, W, F_{du})$ in $\text{Min}\mathbf{Sce}(System = F_{dd}|\mathbf{C_S})$. As mentioned Section 2.3.4, these scenarios have not been considered in the original model of this safety instrumented system in the standard ISO/TR 12489.

Second, as stated Section 2.3.4, we should always have the awareness that critical scenarios are different from significant scenarios (i.e. those who have relative high probability of occurrence). In FDMs, critical scenarios doesn't need to be significant and vice versa. Sometimes, it is practical to truncate scenarios by their occurrence probabilities. But sometimes, it is also useful to have all the critical scenarios at hand to foresee the coming risks in real time analysis.

Take the scenarios in $\text{Min}\mathbf{Sce}(System = F_{dd}|\mathbf{C_S})$ as example. Denote the scenario $(F_{dd}, W, W, W, W, F_{dd}, W)$ by σ_1 and the scenario $(F_s, W, F_{du}, F_s, W, W, F_{du})$ by σ_2. Generally, the probability of σ_2 will be much lower than the probability of σ_1 for having more failed states. But, if the system is currently in the state $(F_s, W, F_{du}, F_s, W, W, W)$, σ_2 now becomes important since it indicates that if there is one more F_{du} failure of $V2$, the system will change its state from W to F_{dd}. In this case, the significant scenario σ_1 is useless because the system will never enter into such scenario from its current situation.

The state probabilities for $System$ in this case are pictured Figure 6.13. Their average probabilities are given Table 6.6.

Case 2 In this case, we show a conditional analysis of the safety instrumented system. We simulate a scene where it is known for some reason (e.g. a specialized inspection) that $S2$ is working, $LS1$ is working and $V2$ is working. According to this scene, the valuation of $S2$, $LS1$ and $V2$ is restricted using the new set of conditions $\mathbf{C_S}'$ below:

$$\mathbf{C_S}' : \left\{ \begin{array}{c} S2 = W \\ LS1 = W \\ LS2 \neq F_{du} \\ V1 \neq F_{dd} \\ V2 = W \end{array} \right\} \tag{6.4}$$

The results of the conditional scenarios analysis are given Table 6.5. Compare to Table 6.4, we can see that the number of scenarios decreases since the valuation range of state variables is narrowed in $\mathbf{C_S}'$.

Table 6.5: Number of scenarios and critical scenarios for $System$ satisfying Eq.(6.4).

y	W	F_s	F_{dd}	F_{du}
$\lvert\mathbf{Sce}(System = y\lvert\mathbf{C_S}')\rvert$	25	95	12	12
$\lvert\text{Min}\mathbf{Sce}(System = y\lvert\mathbf{C_S}')\rvert$	1	4	1	2
$\lvert\text{Max}\mathbf{Sce}(System = y\lvert\mathbf{C_S}')\rvert$	4	12	4	2

To be concrete, we list (the first four) scenarios of each set of critical scenarios below:

$$\begin{aligned}
\text{Min}\mathbf{Sce}(System = W|\mathbf{C_S}') &= (W, W, W, W, W, W, W)\\
\text{Min}\mathbf{Sce}(System = F_s|\mathbf{C_S}') &= \{(F_s, W, W, W, W, W, W),\\
&\quad (W, W, F_s, W, W, W, W),\\
&\quad (W, W, W, W, F_s, W, W),\\
&\quad (W, W, W, W, W, F_s, W)\}\\
\text{Min}\mathbf{Sce}(System = F_{dd}|\mathbf{C_S}') &= (F_{dd}, W, W, W, W, F_{dd}, W)\\
\text{Min}\mathbf{Sce}(System = F_{du}|\mathbf{C_S}') &= \{(W, W, F_{du}, W, W, F_{dd}, W),\\
&\quad (F_{du}, W, W, W, W, F_{dd}, W)\}\\
\\
\text{Max}\mathbf{Sce}(System = W|\mathbf{C_S}') &= \{(F_s, W, F_{du}, W, F_{du}, W, W),\\
&\quad (W, W, W, W, F_s, F_{dd}, W),\\
&\quad (W, W, W, W, F_{du}, F_{dd}, W),\\
&\quad (F_{du}, W, F_{du}, W, F_{du}, W, W)\}\\
\text{Max}\mathbf{Sce}(System = F_s|\mathbf{C_S}') &= \{(F_s, W, F_s, W, F_s, F_s, W),\\
&\quad (F_s, W, F_s, W, F_s, F_{dd}, W),\\
&\quad (F_s, W, F_s, W, F_{du}, W, W),\\
&\quad (F_s, W, F_s, W, F_{du}, F_{dd}, W), ...\}\\
\text{Max}\mathbf{Sce}(System = F_{dd}|\mathbf{C_S}') &= \{(F_s, W, F_{du}, W, F_s, F_{dd}, W),\\
&\quad (F_s, W, F_{du}, W, F_{du}, F_{dd}, W),\\
&\quad (F_{dd}, W, F_{du}, W, F_s, F_{dd}, W),\\
&\quad (F_{dd}, W, F_{du}, W, F_{du}, F_{dd}, W)\}\\
\text{Max}\mathbf{Sce}(System = F_{du}|\mathbf{C_S}') &= \{(F_{du}, W, F_{du}, W, F_s, F_{dd}, W),\\
&\quad (F_{du}, W, F_{du}, W, F_{du}, F_{dd}, W)\}
\end{aligned}$$

We can see that the critical scenarios are also changed due to the change of valuation conditions from $\mathbf{C_S}$ to $\mathbf{C_S}'$.

The state probabilities for $System$ are pictured Figure 6.13. Their average probabilities are given Table 6.6. We can see that comparing to case 1, the probability of being in W increases while the probability of being in a failed state decreases.

Table 6.6: Average probabilities of Pr$\{System = y|\mathbf{C_S}\}$ and Pr$\{System = y|\mathbf{C_S}'\}$ within 8760 hours.

y	W	F_s	F_{dd}	F_{du}
Pr$\{System = y\|\mathbf{C_S}\}$	0.1628	0.8370	8.9081×10^{-5}	5.3380×10^{-5}
Pr$\{System = y\|\mathbf{C_S}'\}$	0.2992	0.7007	6.5952×10^{-5}	1.3351×10^{-5}

Sensitivity analysis

The observers used in the sensitivity analysis are $System = W$, $System = F_s$, $System = F_{dd}$ and $System = F_{du}$. By default, the conditions of $\mathbf{C_S}$ in Eq.(6.3) are used.

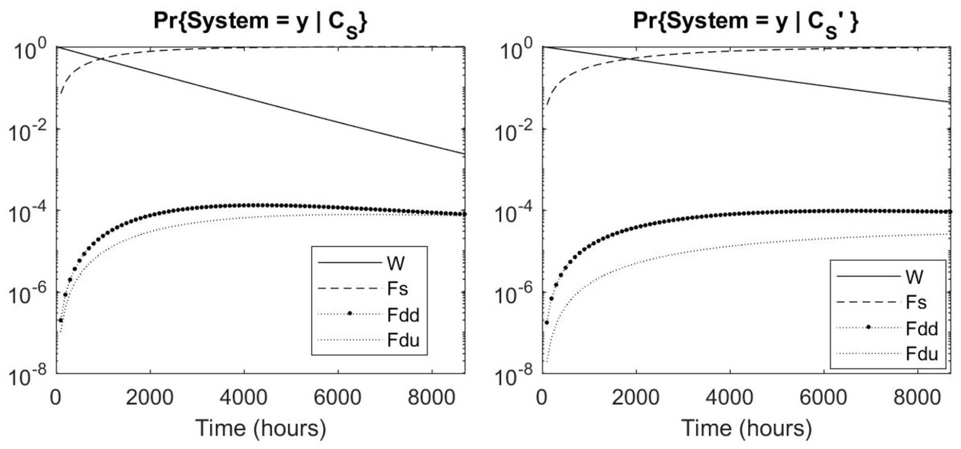


Figure 6.13: State probabilities for $System$ satisfying the conditions in Eq.(6.3) and Eq.(6.4).

The formula of calculating the sensitivity factor $\mathbf{Sen}(System = y, v = c), \forall v \in \mathbf{S}, \forall c \in \text{dom}(v) = \mathbf{SIS}$ is given Eq.(5.32).

To calculate $\mathbf{Sen}(System = y, v = c)$, we should first obtain the sensitivity coefficient matrices. According to Eq.(5.31), the coefficient matrix $\mathbf{A}$ of a state variable v is defined by the partial derivative of the probability measure defined on $\text{dom}(v)$. In this system, the domain of each state variable is the FDS $\mathbf{SIS}$. Therefore, we should first obtain the probability measure p defined on $\mathbf{SIS}$.

The analytical solution of p is given Eq.(3.2). Accordingly, we can deduce that:

$$\left\{\begin{array}{rcl} p(F_s) & = & \frac{\lambda_{F_s}}{\lambda_W}(1-p(W)) \\ p(F_{dd}) & = & \frac{\lambda_{F_{dd}}}{\lambda_W}(1-p(W)) \\ p(F_{du}) & = & \frac{\lambda_{F_{du}}}{\lambda_W}(1-p(W)) \end{array}\right. \tag{6.5}$$

where $\lambda_W = \lambda_{F_{du}} + \lambda_{F_{dd}} + \lambda_{F_s}$.

Then, the sensitivity coefficient matrix $\mathbf{A}$ for the state variables whose domain is

SIS is defined as follows:

$$\mathbf{A} = \begin{bmatrix} \frac{\partial p(W)}{\partial p(W)} & \frac{\partial p(F_s)}{\partial p(W)} & \frac{\partial p(F_{dd})}{\partial p(W)} & \frac{\partial p(F_{du})}{\partial p(W)} \\ \frac{\partial p(W)}{\partial p(F_s)} & \frac{\partial p(F_s)}{\partial p(F_s)} & \frac{\partial p(F_{dd})}{\partial p(F_s)} & \frac{\partial p(F_{du})}{\partial p(F_s)} \\ \frac{\partial p(W)}{\partial p(F_{dd})} & \frac{\partial p(F_s)}{\partial p(F_{dd})} & \frac{\partial p(F_{dd})}{\partial p(F_{dd})} & \frac{\partial p(F_{du})}{\partial p(F_{dd})} \\ \frac{\partial p(W)}{\partial p(F_{du})} & \frac{\partial p(F_s)}{\partial p(F_{du}))} & \frac{\partial p(F_{dd})}{\partial p(F_{du}))} & \frac{\partial p(F_{du})}{\partial p(F_{du}))} \end{bmatrix} \tag{6.6}$$

$$= \begin{bmatrix} 1 & -\frac{\lambda_{F_s}}{\lambda_W} & -\frac{\lambda_{F_{dd}}}{\lambda_W} & -\frac{\lambda_{F_{du}}}{\lambda_W} \\ -\frac{\lambda_W}{\lambda_{F_s}} & 1 & -\frac{\lambda_{F_{dd}}}{\lambda_{F_s}} & -\frac{\lambda_{F_{du}}}{\lambda_{F_s}} \\ -\frac{\lambda_W}{\lambda_{F_{dd}}} & -\frac{\lambda_{F_s}}{\lambda_{F_{dd}}} & 1 & -\frac{\lambda_{F_{du}}}{\lambda_{F_{dd}}} \\ -\frac{\lambda_W}{\lambda_{F_{du}}} & -\frac{\lambda_{F_s}}{\lambda_{F_{du}}} & -\frac{\lambda_{F_{dd}}}{\lambda_{F_{du}}} & 1 \end{bmatrix}$$

We can see that $\mathbf{A}$ only depends on the failure rates λ_{F_s}, $\lambda_{F_{dd}}$ and $\lambda_{F_{du}}$. The value of these parameters is given Table 2.2. For the sake of simplicity, we make the following approximations according to Table 2.2:

– For sensors (S), $\lambda_W^S \approx (\lambda_{F_s}^S + \lambda_{F_{dd}}^S)$, since $\lambda_{F_{du}}^S \ll \lambda_{F_{dd}}^S = \lambda_{F_s}^S$.

– For logic solvers (LS), $\lambda_W^{LS} \approx \lambda_{F_s}^{LS}$, since $\lambda_{F_{dd}}^{LS} \ll \lambda_{F_s}^{LS}$.

– For valves (V), $\lambda_W^V \approx \lambda_{F_s}^V$, since $\lambda_{F_{du}}^V \ll \lambda_{F_s}^V$.

Therefore, the coefficient matrices $\mathbf{A}^S$, $\mathbf{A}^{LS}$, $\mathbf{A}^V$ for sensors, logic solvers and valves are given below:

$$\mathbf{A}^S \approx \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} & 0 \\ -2 & 1 & 1 & 0 \\ -2 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.7}$$

$$\mathbf{A}^{LS} \approx \mathbf{A}^V \approx \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.8}$$

According to the formula in Eq.(5.32), it remains to calculate the conditional probabilities $\Pr\{w = y | v = d\}, d \in \text{dom}(v)$, to obtain the sensitivity factor

$\mathbf{Sen}(System = y, v = c)$. The algorithm of calculating conditional probabilities is given Section 5.3.4. It is implemented by our software **LatticeX**.

The calculation results of $\mathbf{Sen}(System = y, v = c)$ are attached Appendix A.4. These results exhibit many interesting features, which are probably resulted from the non-coherency of the model and the valuation of the operators $\oslash$ and $\|$. The explanation of these results is not provided this thesis but will be included in our future work.

6.3.2 A simplified train control system

The system used in this section is provided by Lei Jiang from his research on the train control system. More information can be found in his paper Jiang et al. (2018). Due to confidentiality requirement, the system is simplified and the parameters are slightly modified.

The objective of this section is to provide a concrete example of integrating time-dependent components/subsystems in FDMs following the idea presented Section 6.2.4.

System description

The system is pictured Figure 6.14. It is made of 9 hot-standby subsystems. The acronyms are given Table 6.7.

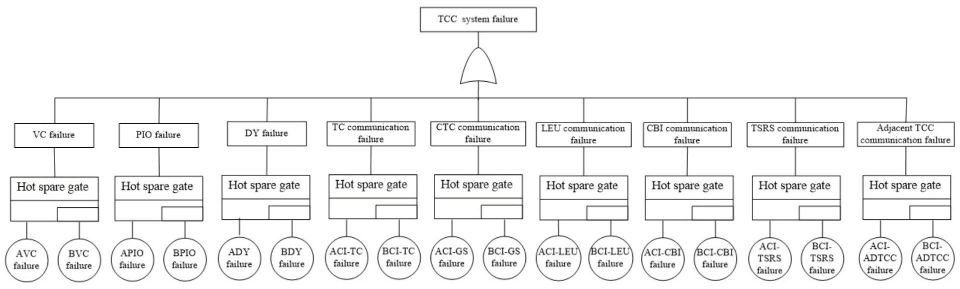


Figure 6.14: A train control system.

Table 6.7: Definitions of acronyms

TCC	Train Control Center	VC	Vital Computer
TC	Track Circuit	CTC	Centralized Traffic Control
LEU	Lineside Electronic Unit	CBI	Computer Based Interlocking
TSRS	Temporary Speed Restriction Server	ATCC	Adjacent TCC

These hot-standby subsystems are divided into two classes.

1. The first class consists of the hot-standby subsystems VC, DY, CBI and TSRS, whose failure will immediately cause the TCC system failure (i.e. the TOP event in Figure 6.14).
2. The second class consists of the hot-standby subsystems PIO, TC, CTC, LEU and ATCC, whose failure will only cause the degradation of the TCC system if none of the subsystems in the first class is failed.

Assume that each hot-standby subsystem satisfies the conditions made in Section 6.2.4. The failure and repair rates of each hot-standby subsystem are given Table 6.8.

Table 6.8: Failure and repair rates of each hot-standby subsystem.

	VC	DY	CBI	$TSRS$	PIO	TC	CTC	LEU	$ATCC$
$\lambda_{A/B}(\times 10^{-4})$	1.26	1.59	0.21	0.21	2.28	1.2	0.31	0.92	0.21
μ	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002

Modeling

The FDM of the TCC system is built as follows:

$$\mathcal{M}: \left\{ \begin{array}{rcl} VC & := & \wedge_{\mathbf{HS}}(A_{VC}, B_{VC}) \\ DY & := & \wedge_{\mathbf{HS}}(A_{DY}, B_{DY}) \\ CBI & := & \wedge_{\mathbf{HS}}(A_{CBI}, B_{CBI}) \\ TSRS & := & \wedge_{\mathbf{HS}}(A_{TSRS}, B_{TSRS}) \\ PIO & := & \wedge_{\mathbf{HS}}(A_{PIO}, B_{PIO}) \\ TC & := & \wedge_{\mathbf{HS}}(A_{TC}, B_{TC}) \\ CTC & := & \wedge_{\mathbf{HS}}(A_{CTC}, B_{CTC}) \\ LEU & := & \wedge_{\mathbf{HS}}(A_{LEU}, B_{LEU}) \\ ATCC & := & \wedge_{\mathbf{HS}}(A_{ATCC}, B_{ATCC}) \\ G1 & := & \vee(\vee(\vee(VC, DY), CBI), TSRS) \\ G2 & := & \vee(\vee(\vee(\vee(PIO, TC), CTC), LEU), ATCC) \\ TCC & := & \vee_{TCC}(G1, G2) \end{array} \right\} \tag{6.9}$$

For a hot-standby subsystem X, A_X and B_X are respectively the state variables of the main unit and the standby unit. The operation $\wedge_{\mathbf{HS}}$ is defined Section 6.2.4.

$G1$ and $G2$ are the flow variables representing respectively the state of the first class and the second class of the hot-standby subsystems described above. $\vee$ and $\wedge$ are the join and meet operators defined Section 6.2.2.

TCC is the flow variable representing the state of the whole TCC system. $\vee_{TCC}$ is an operator describing the relationship between $G1$ and $G2$. The valuation of $\vee_{TCC}$ is given Table 6.9.

Table 6.9: Valuation of $\vee_{TCC} : \mathbf{WF}^2 \twoheadrightarrow \mathbf{WDF}$

$\vee_{TCC}(u,v)$		v	
		W	F
u	W	W	D
	F	F	F

It is easy to verify that all the operators used in this model are coherent. Therefore, $\mathcal{M}$ is also coherent.

Assessment

The observers used in the assessment are $TCC = W$, $TCC = D$ and $TCC = F$.

For scenarios analysis, it is also possible to change the granularity of scenarios. For instance, we can calculate the scenarios at subsystem level, i.e. each scenario $(x_1, x_2, ..., x_9)$ represents a valuation of the 9 flow variables of the hot-standby subsystems, such that $x_1 = \sigma(VC)$, $x_2 = \sigma(DY)$, $x_3 = \sigma(CBI)$, $x_4 = \sigma(TSRS)$, $x_5 = \sigma(PIO)$, $x_6 = \sigma(TC)$, $x_7 = \sigma(CTC)$, $x_8 = \sigma(LEU)$ and $x_9 = \sigma(ATCC)$. The results of the number of scenarios and critical scenarios in this case are given Table 6.10.

Table 6.10: Number of scenarios and critical scenarios for TCC.

y	W	D	F
$\lvert\mathbf{Sce}(TCC = y)\rvert$	1	31	480
$\lvert\mathbf{MinSce}(TCC = y)\rvert$	1	1	4
$\lvert\mathbf{MaxSce}(TCC = y)\rvert$	1	5	1

The critical scenarios are listed below:

$$\begin{aligned}
\mathbf{MinSce}(TCC = W) &= \{(W, W, W, W, W, W, W, W, W)\}\\
\mathbf{MinSce}(TCC = D) &= \{(W, W, W, W, W, W, W, W, F),\\
&= (W, W, W, W, W, W, W, F, W),\\
&= (W, W, W, W, W, W, F, W, W),\\
&= (W, W, W, W, W, F, W, W, W),\\
&= (W, W, W, W, F, W, W, W, W)\}\\
\mathbf{MinSce}(TCC = F) &= \{(W, W, W, F, W, W, W, W, W),\\
&= (W, W, F, W, W, W, W, W, W),\\
&= (W, F, W, W, W, W, W, W, W),\\
&= (F, W, W, W, W, W, W, W, W)\}
\end{aligned}$$

$$\begin{aligned}
\mathbf{MaxSce}(TCC = W) &= \{(W, W, W, W, W, W, W, W, W)\}\\
\mathbf{MaxSce}(TCC = D) &= \{(W, W, W, W, F, F, F, F, F)\}\\
\mathbf{MaxSce}(TCC = F) &= \{(F, F, F, F, F, F, F, F, F)\}
\end{aligned}$$

From these critical scenarios, we can easily verify that the failure mechanism stated in the system description is correctly modeled.

For probabilistic calculation, we should first calculate the probability measures of the 9 hot-standby subsystems. As presented Section 6.2.4, they can be calculated by the Markov chain pictured Figure 6.11. The required parameters are given Table 6.8. Since this Markov chain is not complex, we solved it analytically and calculated the required probability measures in MATLAB. The results are stored in CSV files and uploaded to our software.

The results of the state probabilities for TCC are pictured Figure 6.15.

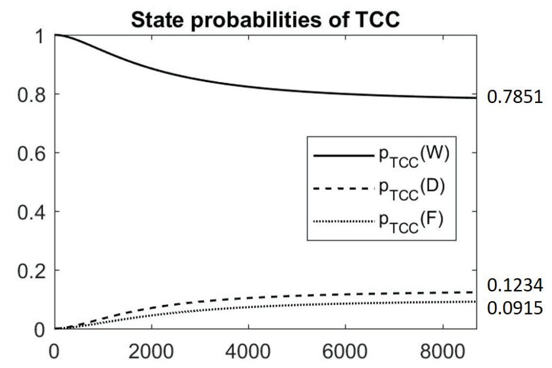


Figure 6.15: State probabilities for TCC within 8760 hours.

Chapter 7

Modeling Epistemic Space of Degradation Processes

7.1 Problem statement

7.1.1 Incomplete knowledge on states

The uncertainties in reliability and safety analyses can be categorized in two ways (Parry 1996, Helton and Burmaster 1996, Agarwal et al. 2004). First, they can be categorized into parametric uncertainty, modeling uncertainty and completeness uncertainty. In this categorization, parametric uncertainty addresses the uncertainty in the quantification of a model; modeling uncertainty can be seen as the uncertainty in the appropriateness of the structure or mathematical form of the model; and completeness uncertainty concerns the degree to which the required performances of the system are modeled or not.

Second, uncertainties can also be categorized into aleatory uncertainty and epistemic uncertainty. In this categorization, the aleatory aspect of uncertainty is addressed when the occurrence of an event or a phenomenon is modeled as a random variable in a stochastic manner. Therefore, aleatory uncertainty can be mathematically modeled using probability theory. The epistemic uncertainty is caused by the incomplete information and the lack of knowledge. Although probabilistic measure is also used to quantify epistemic uncertainty, it is interpreted — different from the probability of random variables — as a kind of subjective probability or belief that measures the analysts' confidence on a phenomenon. Therefore, the main difference between aleatory uncertainty and epistemic uncertainty is that the former is considered to be irreducible, while the latter is reducible when the knowledge of the system is getting more and more complete.

In this chapter, we focus on the epistemic uncertainty caused by the incomplete knowledge on the state of component (or group of components) in the system under study.

Generally, reliability and safety models are built under the condition that the knowledge on the current state of the system is complete, i.e. the state of each component takes exactly one certain value from its valuation domain. However, such condition may not be fulfilled all the time, i.e. that there can be some discrepancies between the diagnostic made on the state of component and its actual state. For instance, we may lose the state information of a component due to the failure of its monitoring sensor. In this case, the state of component becomes uncertain, i.e. that we don't know whether it is working or failed.

To model such uncertainty, a classical way is to regard this uncertain state as a new state and add it to the state space of the component. For instance, to take into account the uncertain state of a Boolean component (which can be either working or failed), we can add a new state "unknown" to its state space. As results, this component becomes a three-state component whose state space is {working, failed, unknown}. In this case, "working" means that we know that the component is working, "failed" means that we know that the component is failed and "unknown" means that we don't know the state of the component.

If there are $n \geq 1$ states in the state space of a component, then how many uncertain states it may have? The answer is $2^n - n - 1$. Take the case that $n = 3$. Assume that the state space of a component is $\{a, b, c\}$. An uncertain state can be "we don't know completely the state of the component" or it also can be "we know that it cannot be a, but we don't know whether it is b or c". We can see that an uncertain state is a possible combination of any two or three elements in $\{a, b, c\}$. Therefore, the uncertain states obtained from a state space with n elements are all the k-combinations of the n elements such that $2 \leq k \leq n$, since there should be at least two elements in the combination to make it uncertain. Finally, the total number of uncertain states is $\sum_{2\leq k\leq n} \binom{n}{k} = \sum_{0\leq k\leq n} \binom{n}{k} - \binom{n}{1} - \binom{n}{0} = 2^n - n - 1$.

Although this number is of exponential order, the determination of uncertain states is still operable since the number of states n of a component is usually small.

Once new uncertain states are added to the valuation domain of a state variable, the operations related to this variable should also be adjusted. Take the binary operation $\diamondsuit(u, v)$ as example. Assume that there are n states in the original valuation domain of u and v. If $2^n - n - 1$ new states are added to both of the domains dom(u) and dom(v), the total number of new valuations that should be added for

$\diamondsuit$ is $(2^n - n - 1)^2 + 2n \cdot (2^n - n - 1)$.

Table 7.1 lists the number of uncertain states and the required new valuations for a binary operation $\diamondsuit(u, v)$ where $|\text{dom}(u)| = |\text{dom}(v)| = n$. We can see that when $n > 2$, defining the required new valuations for a binary operation $\diamondsuit$ manually is however not operable.

Table 7.1: Number of uncertain states and the required new valuations for $\diamondsuit$.

n	Number of uncertain states $2^n - n - 1$	Number of new valuations $(2^n - n - 1)^2 + 2n \cdot (2^n - n - 1)$
1	0	0
2	1	5
3	4	40
4	11	209
5	26	936

In this chapter, we present a FDM-based modeling approach to model such epistemic uncertainty. The most highlighted part of this approach is that we don't need to manually define neither the new uncertain states nor the required new valuations. Instead, we transform the whole FDM into the epistemic space. This transformation can be mathematically defined. Accordingly, the uncertain states and the required valuations are automatically generated during this transformation. When dealing with multistate systems, this automatic generation will be more efficient, more effective and less erroneous than manual determinations.

7.1.2 Epistemic space

A proposition P is said to be *epistemically possible* (for an agent a) if there exists a *scenario* (or an epistemically possible world) at which P is true. There are also scenarios at which P is false. The space of all scenarios of P (at which P is either true or false) is called the *epistemic space* (for an agent a) (Bjerring 2014).

Let P be the predicate "$\sigma(v) \in X$", where v is a variable, σ is the valuation assignment of v (see Section 4.2) and $X \subseteq \text{dom}(v)$. Then, the epistemic space of P should contain all the scenarios satisfying and falsifying $\sigma(v) \in X$.

Take $\text{dom}(v) = \mathbf{WF}$ as example. Three propositions (or predicates) can be made in the form of $\sigma(v) \in X$. They are interpreted as different level of knowledge that we have on the valuation of v:

– $P_1 : \sigma(v) \in \{W\}$, i.e. it is known that the value of v is W;

– $P_2 : \sigma(v) \in \{F\}$, i.e. it is known that the value of v is F;

– $P_3 : \sigma(v) \in \{W, F\}$, i.e. it is known that the value of v is uncertain between W and F.

The set $\{\{W\}, \{F\}, \{W, F\}\}$ is regarded as the **epistemic space** built over the domain $\mathbf{WF}$. In this epistemic space, the scenarios — i.e. the possible valuations of X — that either satisfy or falsify $\sigma(v) \in X$ are all included.

The element in epistemic space is called **epistemic state**, representing an *epistemically possible* valuation of v. A distinction should be made between “epistemically possible” and “possible”. A possible valuation of v is an objective value that v can take in its valuation domain. An epistemically possible valuation of v is a subjective value representing how much we know about the valuation of v.

In the above example, W and F are the possible valuations of v, while $\{W\}$, $\{F\}$ and $\{W, F\}$ are the epistemically possible valuations of v. It is worth noting that the epistemic state $\{W, F\}$ doesn’t mean that v can take simultaneously both of values W and F. Instead, it only means that the two values W and F are both epistemically possible for v under our current knowledge on $\sigma(v)$.

To be more generic, let $\text{dom}(v) = \Theta$, where Θ is a non-empty finite set. An epistemically possible valuation of v is thus a subset $X \subseteq \Theta$. According to the closed-world assumption, X should contain at least one value in Θ, i.e. $X \neq \emptyset$. Therefore, the epistemic space built over Θ is $2^\Theta \backslash \{\emptyset\}$, where 2^Θ is the power set of Θ and $\backslash$ stands for the difference of sets.

In the epistemic space $2^\Theta \backslash \{\emptyset\}$, an epistemic state X is *certain* if $|X| = 1$; otherwise, X is uncertain. If $|\Theta| = n$, then $|2^\Theta \backslash \{\emptyset\}| = 2^n - 1$. Therefore, the number of uncertain epistemic states is $2^n - n - 1$, since the number of certain epistemic states is n.

7.2 FDSs in epistemic space

In FDMs, the valuation domain of each variable is not simply a set Θ but a FDS $\langle \Theta, \sqsubseteq, \perp, p \rangle$. The epistemic space built over Θ is $2^\Theta \backslash \{\emptyset\}$. In this section, we will define the degradation orders among the epistemic states in $2^\Theta \backslash \{\emptyset\}$ and the probability measures that can be assigned on $2^\Theta \backslash \{\emptyset\}$.

7.2.1 Degradation orders among epistemic states

Definition 7.2.1. We define that the epistemic space built over a meet-semi-lattice $\langle \Theta, \sqsubseteq, \perp \rangle$ is a new meet-semi-lattice $\langle \Omega/_\equiv, \sqsubseteq, \{\perp\} \rangle$, where:

- $\Omega = 2^{\Theta} \backslash \{\emptyset\}$ is the epistemic space built over the set Θ.
- $\Omega/_{\equiv}$ is the quotient set of Ω by $\equiv$ (equivalence). We will explain it later in this section.
- $\forall S, T \in \Omega, S \sqsubseteq T \Leftrightarrow (\forall y \in T, \exists x \in S, x \sqsubseteq y) \wedge (\forall x \in S, \exists y \in T, x \sqsubseteq y)$; otherwise, $S \sim T$.

The epistemic space Ω built over Θ is presented in the previous section. The quotient set $\Omega/_{\equiv}$ will be explained later in this section. As for the degradation order, we will use the following three examples to explain why it is defined as in the above definition.

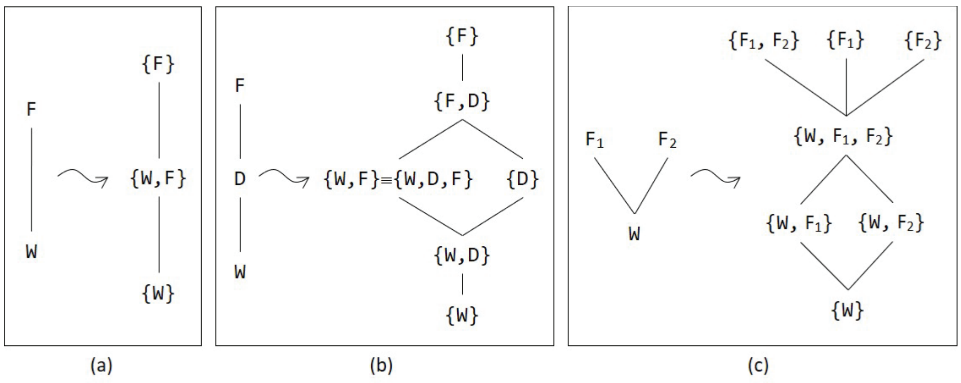


Figure 7.1: The epistemic space built over $\mathbf{WF}$, $\mathbf{WDF}$ and $\mathbf{W2F}$.

Example 7.2.1 (Epistemic space built over $\mathbf{WF}$)**.** The epistemic space built over $\mathbf{WF}$ is pictured Figure 7.1 (a). In this case, the epistemic states are:

$$2^{\{W,F\}} \backslash \{\emptyset\} = \{\{W\}, \{F\}, \{W, F\}\}$$

The degradation orders among these states are defined as:

$$\{W\} \sqsubset \{W, F\} \sqsubset \{F\}$$

Intuitively, $\{W\} \sqsubset \{W, F\}$ because comparing to $\{W\}$, $\{W, F\}$ has an additional possibility of being in a more degraded state F. Therefore, the degradation level of $\{W, F\}$ should be higher than $\{W\}$. Similarly, comparing to $\{F\}$, $\{W, F\}$ has an additional possibility to be in a less degraded state W. Therefore, the degradation level of $\{W, F\}$ should be lower than $\{F\}$, i.e. $\{W, F\} \sqsubset \{F\}$.

Example 7.2.2 (Epistemic space built over $\mathbf{WDF}$)**.** The epistemic space built over $\mathbf{WDF}$ is pictured Figure 7.1 (b). In this case, the epistemic states are:

$$2^{\{W,D,F\}}\backslash\{\emptyset\} = \{\{W\},\{D\},\{F\},\{W,D\},\{D,F\},\{W,F\},\{W,D,F\}\}$$

The degradation orders among these states are defined as follows:

$$\{W\} \sqsubset \{W,D\} \sqsubset \{D\} \sqsubset \{F,D\} \sqsubset \{F\} \tag{7.1}$$

$$\{W\} \sqsubset \{W,D\} \sqsubset \{W,F\} \equiv \{W,D,F\} \sqsubset \{F,D\} \sqsubset \{F\} \tag{7.2}$$

$$\{D\} \sim \{W,F\} \tag{7.3}$$

$$\{D\} \sim \{W,D,F\} \tag{7.4}$$

First, the linear orders in Eq.(7.1) and Eq.(7.2) can be understood similarly as Example 7.2.1. But in Eq.(7.2), there is an equivalence: $\{W,F\} \equiv \{W,D,F\}$.

Definition 7.2.2 (Equivalence)**.** If $\exists X, Y \in \Omega$ $(X \neq Y)$ such that $X \sqsubseteq Y$ and $Y \sqsubseteq X$, then they are *equivalent* with respect to the degradation order $\sqsubseteq$, denoted by $X \equiv Y$.

We can deduce that the equivalence occurs when there are more than three states that are ordered linearly in the original FDS.

Proposition 7.2.1. Let $\bot \sqsubset d_1 \sqsubset ... \sqsubset d_n \sqsubset \top$, $n \geq 1$ be a chain (i.e. a linearly ordered subset) of a meet-semi-lattice. Denote that $I = \{d_1, ..., d_n\}$. Then, $\forall X \subseteq I$ $(X \neq \emptyset)$, $X \in \Omega$ and the following relations hold according to the degradation orders defined in Definition 7.2.1:

$$\{\bot,\top\} \equiv \{\bot,\top\} \cup X \tag{7.5}$$

$$X \sim \{\bot,\top\} \cup X \tag{7.6}$$

$$X \sim \{\bot,\top\} \tag{7.7}$$

The equivalence in Eq.(7.5) indicates that if the two extremes $\bot$ and $\top$ are included in the epistemic state $\{\bot,\top\} \cup X$, then no matter how many intermediate states (in X) are epistemically possible, the degradation level of $\{\bot,\top\} \cup X$ is bounded by $\{\bot,\top\}$. For instance, the equivalence in Eq.(7.2) means that the degradation level of $\{W,F\}$ is equal to the one of $\{W,D,F\}$, i.e. knowing that D is not epistemically possible will not change the degradation level of the epistemic state of a component.

The incomparabilities in Eq.(7.6) and Eq.(7.7) indicates that if the two extremes $\perp$ and $\top$ are excluded from X, then X is incomparable with $\{\perp, \top\} \cup X$ and $\{\perp, \top\}$. For instance, in Eq.(7.6) and Eq.(7.7), $\{D\} \sim \{W, F\}$ and $\{D\} \sim \{W, D, F\}$.

Mathematically, the existence of $\equiv$ makes the partial order $\sqsubseteq$ become a preorder (since the antisymmetry in Definition 3.1.1 is not satisfied). In order to keep the degradation order $\sqsubseteq$ still being a partial order in the epistemic space, we shall merge the equivalent elements into quotients, e.g. the two epistemic states $\{W, D, F\}$ and $\{W, F\}$ are considered as one element with respect to $\sqsubseteq$.

Denote the quotient set of the equivalence $\equiv$ on a set Ω by $\Omega/_{\equiv}$. Then, the epistemic space $\langle \Omega/_{\equiv}, \sqsubseteq, \{\perp\}\rangle$ in Definition 7.2.1 is still a meet-semi-lattice, since $\sqsubseteq$ is a partial order over the quotients set $\Omega/_{\equiv}$ and it is easy to verify that $\{\perp\}$ is the least element in $\Omega/_{\equiv}$.

Example 7.2.3 (Epistemic space built over $\mathbf{W2F}$)**.** The epistemic space built over $\mathbf{W2F}$ is pictured Figure 7.1 (c). In this case, we have two incomparable states $F_1 \sim F_2$ so that the degradation orders are more complex.

– First, $\{W\} \sqsubset \{W, F_1\}$ and $\{W\} \sqsubset \{W, F_2\}$ can be understood similarly as Example 7.2.1.

– Then, for $\{W, F_1\}$ and $\{W, F_2\}$, they have both the possibility of being in W and the possibility of being in one of the two incomparable failed states F_1 and F_2. Since $F_1 \sim F_2$, then $\{W, F_1\} \sim \{W, F_2\}$.

– $\{W, F_1\}$ and $\{W, F_2\}$ are both less degraded than $\{W, F_1, F_2\}$ because comparing to $\{W, F_1\}$ and $\{W, F_2\}$, $\{W, F_1, F_2\}$ has an additional possibility of being the another failed state. It means that in $\{W, F_1, F_2\}$ the possibility of being in a failed state is enlarged. Therefore, $\{W, F_1\} \sqsubset \{W, F_1, F_2\}$ and $\{W, F_2\} \sqsubset \{W, F_1, F_2\}$.

– As for $\{F_1\}$, $\{F_2\}$ and $\{F_1, F_2\}$, they are all more degraded than $\{W, F_1, F_2\}$, since there is no possibility of being in W. However, $\{F_1\} \sim \{F_2\} \sim \{F_1, F_2\}$. In these three epistemic states, the component is sure to be failed but the difference is whether the failed state is certain (i.e. $\{F_1\}$ or $\{F_2\}$) or not (i.e. $\{F_1, F_2\}$). In this case, $\{F_1, F_2\}$ doesn't mean that there is an additional possibility to be in another failed state comparing to $\{F_1\}$ and $\{F_2\}$. Instead, it only means that the failed state is uncertain. This incomparability between $\{F_1\}$, $\{F_2\}$ and $\{F_1, F_2\}$ should be distinguished from the relation between $\{W, F_1\}$, $\{W, F_2\}$ and $\{W, F_1, F_2\}$.

7.2.2 Basic belief assignment of epistemic states

The probability measure applied in the epistemic space is not the probability of states but a subjective *belief* which measures the analysts' confidence on the occurrence of the epistemic states.

The belief functions are introduced by Dempster (Dempster 1967) and then reinforced by Shafer (Shafer 1976). In the Dempster-Shafer theory, the allocation of belief functions to uncertain phenomena is called *basic belief assignment* (BBA) or mass assignment.

Definition 7.2.3 (Mass assignment)**.** Let $\Omega = 2^{\Theta} \backslash \{\emptyset\}$ be the epistemic space built over a finite set Θ. The *mass assignment* on Ω is a function $m : \Omega \to [0, 1]$ such that $\forall X \in \Omega$, $m(X)$ is called the mass of X and $\sum_{X \in \Omega} m(X) = 1$.

In Shafer's original work, if $m(\emptyset) = 0$, then m is called *normalized*. The mass assignment in Definition 7.2.3 is always normalized since the empty set is removed from the Ω. The assumption $m(\emptyset) = 0$ is called the closed-world assumption, meaning that the possibility that the true valuation of a variable is not included in Θ is zero.

Moreover, the epistemic states in Ω are also *mutually exclusive* even though their intersection may not be empty. The epistemic states should be regarded as atomic states in Ω.

Probabilistic indicators

Let X be an epistemic state in Ω, then we can calculate the belief, the plausibility and the commonality of X.

The *belief* of X is denoted by $\mathbf{Bel}(X)$. It quantifies the mass of evidences supporting X. Mathematically, it can be calculated as follows:

$$\mathbf{Bel}(X) = \sum_{Y \in \Omega, Y \subseteq X} m(Y) \tag{7.8}$$

If m is normalized, then $\mathbf{Bel}(\Theta) = 1$.

The *plausibility* $\mathbf{Pl}(X)$ is the amount of belief not strictly committed to the complement $\Theta \backslash X$. It quantifies the mass potentially supports X, which can be calculated as follows:

$$\mathbf{Pl}(X) = \sum_{Y \in \Omega, Y \cap X \neq \emptyset} m(Y) \tag{7.9}$$

If m is normalized, then $\mathbf{Pl}(X) = 1 - \mathbf{Bel}(\Theta \backslash X)$.

Let p be a probability measure defined on Θ and P be a probability function such that $P(X) = \sum_{s \in X} p(s), \forall X \in \Omega$. Then, the belief and the the plausibility of X can be regarded respectively as a lower bound and an upper bound of $P(X)$, i.e.

$$\mathbf{Bel}(X) \leq P(X) = \sum_{s \in X} p(s) \leq \mathbf{Pl}(X) \tag{7.10}$$

The *commonality* $\mathbf{Com}(X)$ quantifies the mass committed to X from the epistemic states $Y \in \Omega$ such that $X \subseteq Y$. It is mathematically calculated as follows:

$$\mathbf{Com}(X) = \sum_{X \subseteq Y \in \Omega} m(Y) \tag{7.11}$$

Proposition 7.2.2 (Compatibility)**.** Let p be a probability measure defined on Θ and m be a mass function defined on Ω such that $\Omega = 2^{\Theta} \backslash \{\emptyset\}$. Then, p is said to be *compatible* with m if the following inequality holds for all $s \in \Theta$:

$$p(s) \leq \mathbf{Com}(\{s\}) \tag{7.12}$$

The above inequality indicates that the true probability $p(s)$ of being in state s is no greater than the sum of the mass of all epistemic states Y containing s, i.e. $p(s) \leq \sum_{s \in Y \in \Omega} m(Y)$.

Example 7.2.4. Assume that $\Theta = \{W, F\}$. According to Eq.(7.12), p is compatible with m if the following inequalities hold:

$$\begin{cases} p(W) \leq m(\{W\}) + m(\{W, F\}) \\ p(F) \leq m(\{F\}) + m(\{W, F\}) \end{cases}$$

7.2.3 Transformation of FDSs

Definition 7.2.4. The transformation of FDSs into the epistemic space is defined as the unary operation $(.)^u : \mathbf{FDS} \twoheadrightarrow \mathbf{FDS}$ such that for every FDS $\mathcal{L} : \langle \Theta, \sqsubseteq, \perp, p \rangle$,

$$(\mathcal{L})^u \overset{def}{=} \langle \Omega/_{\equiv}, \sqsubseteq, \{\perp\}, m \rangle$$

where,

– $\langle \Omega/_{\equiv}, \sqsubseteq, \{\perp\} \rangle$ is the epistemic space built over the meet-semi-lattice $\langle \Theta, \sqsubseteq, \perp \rangle$ (see Definition 7.2.1).

– $m : \Omega \rightarrow [0, 1]$ is a mass assignment on Ω (see Definition 7.2.3), which is compatible with p (see Eq.(7.12)).

It is worth mentioning that if the probability measure p is not defined in $\mathcal{L}$, then m can be directly assigned to Ω with appropriate values.

Since $\mathbf{FDS}$ is also closed under the transformation $(.)^u$, this transformation can thus be added to $\langle \mathbf{FDS}, \otimes, \Phi \rangle$ as the third type of operations, i.e. $\langle \mathbf{FDS}, \otimes, \Phi, (.)^u \rangle$.

In Section 7.2.2, we present the three indicators, i.e. $\mathbf{Bel}(X)$, $\mathbf{Pl}(X)$ and $\mathbf{Com}(X)$, which are defined in the evidence theory to quantify the epistemic states according to the mass assignment m.

But it might also be useful to quantify the states in Θ according to the mass assignment m in Ω. In this part, we present two probabilistic indicators defined in Θ, which are calculated by m and can be used to quantify the upper bound and the lower bound of the degradation level of states in Θ.

Definition 7.2.5. For each state $s \in \Theta$, denote the belief that the degradation level is in the best case to be s by $\mathbf{Best}(s)$ and the belief that the degradation level is in the worst case to be s by $\mathbf{Worst}(s)$. Then, they are defined and calculated as follows:

$$\mathbf{Best}(s) \overset{def}{=} \sum_{X \in \Omega, \{s\} \sqsubseteq X} m(X) \tag{7.13}$$

$$\mathbf{Worst}(s) \overset{def}{=} \sum_{X \in \Omega, X \sqsubseteq \{s\}} m(X) \tag{7.14}$$

Particularly, if s is the least element $\perp$ or the greatest element $\top$ of Θ, then:

$$\begin{cases} \mathbf{Best}(\perp) &= 1 \\ \mathbf{Best}(\top) &= m(\{\top\}) \\ \mathbf{Worst}(\perp) &= m(\{\perp\}) \\ \mathbf{Worst}(\top) &= 1 \end{cases}$$

Example 7.2.5. Take the domain $\mathbf{WDF}$ as example. Assume that the probability measure p defined on $\Theta = \{W, D, F\}$ is as follows:

$$\begin{cases} p(W) &= 0.7 \\ p(D) &= 0.2 \\ p(F) &= 0.1 \end{cases} \tag{7.15}$$

To obtain a compatible mass assignment m on its epistemic space Ω from p, consider the following allocation method.

Denote the proportion of $p(s)$ in $m(X)$ by $p(s)|_X$. If $s \notin X$, $p(s)|_X = 0$. Then, the following equality should hold for all $s \in \Theta$:

$$p(s) = \sum_{s \in X} p(s)|_X \tag{7.16}$$

If $p(s)|_X$ is determined for all $s \in \Theta$ and for all $X \in \Omega$, the mass $m(X)$ can be obtained as follows:

$$m(X) = \sum_{s \in \Theta} p(s)|_X \tag{7.17}$$

It is easy to verify that the mass assignment m obtained by such allocation always satisfies the inequality in Eq.(7.12), since $\mathbf{Com}(\{s\}) = \sum_{s \in X} m(X) = \sum_{s \in X} \sum_{t \in \Theta} p(t)|_X = p(s) + \sum_{s \in X} \sum_{t \in X, s \neq t} p(t)|_X \geq p(s)$.

Table 7.2 shows an example of such allocation. The probability proportions in each line satisfy the equality in Eq.(7.16) and the mass $m(X)$ is calculated by summing the proportions of the column X according to Eq.(7.17).

Table 7.2: Allocation example of $p(s)$ in $m(X)$ where $\Theta = \{W, D, F\}$.

X	$\{W\}$	$\{D\}$	$\{F\}$	$\{W,D\}$	$\{W,F\}$	$\{D,F\}$	$\{W,D,F\}$	Sum
$p(W)\|_X$	0.5	0	0	0.1	0.05	0	0.05	0.7
$p(D)\|_X$	0	0.15	0	0.03	0	0.01	0.01	0.2
$p(F)\|_X$	0	0	0.08	0	0.005	0.01	0.005	0.1
$m(X)$	0.5	0.15	0.08	0.13	0.055	0.02	0.065	1

According to the degradation orders pictured Figure 7.1 (b), we can thus calculate the two indicators **Best** and **Worst** in **WDF** according to the mass assignment m in Table 7.2. The results are given below:

$$\begin{cases} \mathbf{Best}(W) & = & 1 \\ \mathbf{Best}(D) & = & m(\{D\}) + m(\{W,D\}) + m(\{W\}) = 0.78 \\ \mathbf{Best}(F) & = & m(\{F\}) = 0.1 \\ & & \\ \mathbf{Worst}(W) & = & m(\{W\}) = 0.7 \\ \mathbf{Worst}(D) & = & m(\{D\}) + m(\{F,D\}) + m(\{F\}) = 0.25 \\ \mathbf{Worst}(F) & = & 1 \end{cases}$$

To summarize, a short comparison of the FDSs in state space and in epistemic space is given Table 7.3.

7.3 FDMs in epistemic space

In this section, we shall define mathematically the transformation of FDMs into its corresponding epistemic space. Technically, this transformation relies on the four types of transformations of the operations on FDSs.

Table 7.3: Comparison of FDSs in state space and in epistemic space.

	State space $\mathcal{L} : \langle \Theta, \sqsubseteq, \perp, p \rangle$	Epistemic Space $(\mathcal{L})^u : \langle \Omega/_{\equiv}, \sqsubseteq, \{\perp\}, m \rangle$
Elements	States in Θ	Epistemic states in $\Omega = 2^{\Theta} \backslash \{\emptyset\}$
Degradation orders	$x \sqsubseteq y \ (x, y \in \Theta)$	$S \sqsubseteq T \ (S, T \in \Omega/_{\equiv})$
Least element	$\perp \in \Theta$	$\{\perp\} \in \Omega$
	$p : \Theta \to [0, 1]$	$m : \Omega \to [0, 1]$
Probabilistic indicators	$\mathbf{Best} : \Theta \to [0, 1]$	$\mathbf{Bel} : \Omega \to [0, 1]$
	$\mathbf{Worst} : \Theta \to [0, 1]$	$\mathbf{Pl} : \Omega \to [0, 1]$

7.3.1 Transformation of operations

Consider the binary operation $\phi : \mathcal{A} \otimes \mathcal{B} \twoheadrightarrow \mathcal{C}$, where $\mathcal{A}, \mathcal{B}, \mathcal{C} \in \mathbf{FDS}$. If the input arguments of ϕ becomes uncertain, the output of ϕ should also be uncertain. The uncertainty is thus propagated from the domain of ϕ to the codomain of ϕ.

To model the propagation of uncertainty through an operation ϕ, we propose four transformations for ϕ, which are the left-transformation, right-transformation, inner-transformation and outer-transformation.

Definition 7.3.1. The *left*-transformation of ϕ is defined as follows:

$$\phi_L : (\mathcal{A})^u \otimes \mathcal{B} \twoheadrightarrow (\mathcal{C})^u$$

such that $\forall (X, y) \in (\mathcal{A})^u \otimes \mathcal{B}$,

$$\phi_L(X, y) = \{\phi(x, y) | x \in X\} \tag{7.18}$$

Definition 7.3.2. The *right*-transformation of of ϕ is defined as follows:

$$\phi_R : \mathcal{A} \otimes (\mathcal{B})^u \twoheadrightarrow (\mathcal{C})^u$$

such that $\forall (x, Y) \in \mathcal{A} \otimes (\mathcal{B})^u$,

$$\phi_R(x, Y) = \{\phi(x, y) | y \in Y\} \tag{7.19}$$

The left- and right-transformation of ϕ model respectively the propagation of uncertainties from the domain of the first and the second input argument into the codomain of ϕ.

Example 7.3.1. Take the Boolean join $\vee$ and meet $\wedge$ operators as example. The definition of these operators can be found Section 6.2.2. Their left- and right-transformation $\vee_L$, $\wedge_L$, $\vee_R$ and $\wedge_R$ are pictured Figure 7.2.

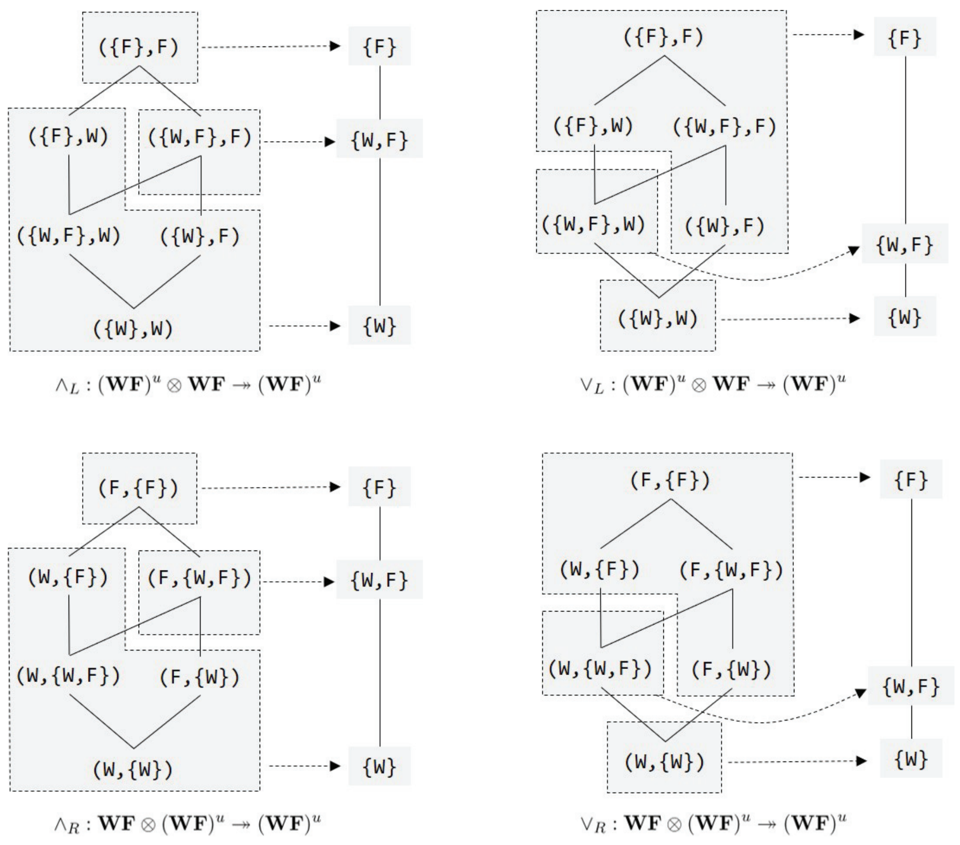

({F},F)
({F},W)
({W,F},F)
({W,F},W)
({W},F)
({W},W)
{F}
{W,F}
{W}
$\wedge_L : (\mathbf{WF})^u \otimes \mathbf{WF} \twoheadrightarrow (\mathbf{WF})^u$
$\vee_L : (\mathbf{WF})^u \otimes \mathbf{WF} \twoheadrightarrow (\mathbf{WF})^u$
(F,{F})
(W,{F})
(F,{W,F})
(W,{W,F})
(F,{W})
(W,{W})
$\wedge_R : \mathbf{WF} \otimes (\mathbf{WF})^u \twoheadrightarrow (\mathbf{WF})^u$
$\vee_R : \mathbf{WF} \otimes (\mathbf{WF})^u \twoheadrightarrow (\mathbf{WF})^u$

Figure 7.2: The left- and right-transformation of the Boolean join and meet operators.

Definition 7.3.3. The *inner*-transformation of ϕ is defined as follows:

$$\phi_u : (\mathcal{A})^u \otimes (\mathcal{B})^u \twoheadrightarrow (\mathcal{C})^u$$

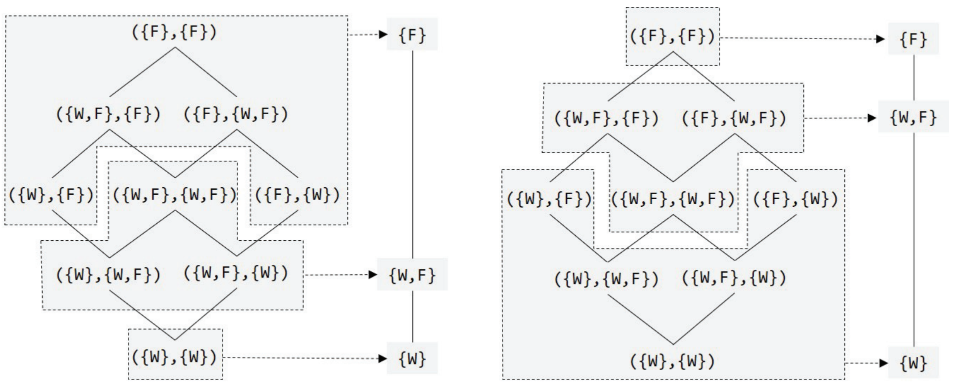
such that $\forall (X, Y) \in (\mathcal{A})^u \otimes (\mathcal{B})^u$,

$$\phi_u(X, Y) = \{\phi(x, y) | x \in X, y \in Y\} \tag{7.20}$$

The inner-transformation ϕ_u can be seen as a composition of ϕ_L and ϕ_R.

Example 7.3.2. Figure 7.3 shows the inner-transformation $\vee_u$ and $\wedge_u$ of the Boolean join $\vee$ and meet $\wedge$ operators.

Comparing to Figure 7.2, the range of uncertainties in the domain of the two operations in Figure 7.3 is enlarged, because the uncertainties are introduced to both of the two input domains.



$\vee_u : (\mathbf{WF})^u \otimes (\mathbf{WF})^u \twoheadrightarrow (\mathbf{WF})^u$ $\wedge_u : (\mathbf{WF})^u \otimes (\mathbf{WF})^u \twoheadrightarrow (\mathbf{WF})^u$

Figure 7.3: The inner-transformation of the Boolean join and meet operators.

Definition 7.3.4. The *outer*-transformation of ϕ is defined as follows:

$$\phi^u : (\mathcal{A} \otimes \mathcal{B})^u \twoheadrightarrow (\mathcal{C})^u$$

such that $\forall Z \in (\mathcal{A} \otimes \mathcal{B})^u$,

$$\phi^u(Z) = \{\varphi(x,y) | (x,y) \in Z\} \tag{7.21}$$

The outer-transformation is applied to the case where the uncertainties are introduced to the product $\mathcal{A} \otimes \mathcal{B}$. For instance, the operation $\wedge_{\mathbf{HS}} : \mathbf{WF} \otimes \mathbf{SWF} \twoheadrightarrow \mathbf{WF}$ for the hot-standby system presented Section 6.2.4 can be transformed using the outer-transformation. In this case, we can model the uncertainty that is directly introduced to the four reachable states in the product $\mathbf{WF} \otimes \mathbf{SWF}$.

Proposition 7.3.1. $\forall \mathcal{A}, \mathcal{B} \in \mathbf{FDS}$, there exists a weakly-coherent abstraction $\alpha_{\mathcal{AB}} : (\mathcal{A} \otimes \mathcal{B})^u \twoheadrightarrow (\mathcal{A})^u \otimes (\mathcal{B})^u$ such that,

- $\forall Z \in (\mathcal{A} \otimes \mathcal{B})^u$, $\alpha_{\mathcal{AB}}(Z) = (\{x | (x,y) \in Z\}, \{y | (x,y) \in Z\})$.
- $\forall Z_1, Z_2 \in (\mathcal{A} \otimes \mathcal{B})^u$, $Z_1 \sqsubseteq Z_2 \Rightarrow \alpha_{\mathcal{AB}}(Z_1) \sqsubseteq \alpha_{\mathcal{AB}}(Z_2)$.

The existence of $\alpha_{\mathcal{AB}}$ indicates that the abstraction level of $(\mathcal{A})^u \otimes (\mathcal{B})^u$ is higher than $(\mathcal{A} \otimes \mathcal{B})^u$.

Take the Boolean domain $\mathbf{WF}$ as example. Figure 7.4 pictures the coherent abstraction $\alpha_{\mathbf{WF}} : (\mathbf{WF} \otimes \mathbf{WF})^u \twoheadrightarrow (\mathbf{WF})^u \otimes (\mathbf{WF})^u$.

In this figure, we can see that the epistemic states excluded from the grey rectangle are one-to-one mapped from $(\mathbf{WF} \otimes \mathbf{WF})^u$ to $(\mathbf{WF})^u \otimes (\mathbf{WF})^u$. The seven

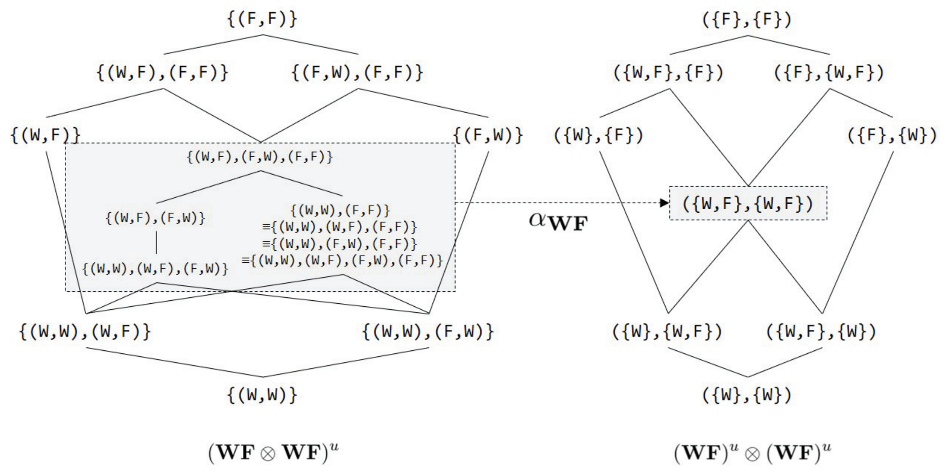


Figure 7.4: The coherent abstraction $\alpha_{\mathbf{WF}} : (\mathbf{WF} \otimes \mathbf{WF})^u \twoheadrightarrow (\mathbf{WF})^u \otimes (\mathbf{WF})^u$.

epistemic states in the grey rectangle of $(\mathbf{WF} \otimes \mathbf{WF})^u$ are abstracted into only one epistemic state $(\{W, F\}, \{W, F\})$ in $(\mathbf{WF})^u \otimes (\mathbf{WF})^u$.

Proposition 7.3.2 (Coherency)**.** If an operation ϕ is coherent, then its four transformations ϕ_L, ϕ_R, ϕ_u and ϕ^u are also coherent.

We will prove the coherency of ϕ_L as follows. The coherency if other transformations can be proved in a similar way.

Proof. First, for all $(X_1, y_1), (X_2, y_2) \in (\mathcal{A})^u \otimes \mathcal{B}$, we have:

$$(X_1, y_1) \sqsubseteq (X_2, y_2) \Rightarrow X_1 \sqsubseteq X_2 \wedge y_1 \sqsubseteq y_2 \tag{7.22}$$

Then, according to Definition 7.2.1,

$$X_1 \sqsubseteq X_2 \Rightarrow (\forall x_1 \in X_1, \exists x_2 \in X_2, x_1 \sqsubseteq x_2) \wedge (\forall x_2 \in X_2, \exists x_1 \in X_1, x_1 \sqsubseteq x_2) \tag{7.23}$$

According to Definition 7.3.1, the result of (X_1, y_1) and (X_2, y_2) under ϕ_L are defined as follows:

$$\begin{cases} \phi_L(X_1, y_1) &= \{\phi(x_1, y_1) | x_1 \in X_1\} \\ \phi_L(X_2, y_2) &= \{\phi(x_2, y_2) | x_2 \in X_2\} \end{cases} \tag{7.24}$$

Then, from Eq.(7.23) and Eq.(7.24), we can deduce that:

$$\begin{cases} \forall z_1 \in \phi_L(X_1, y_1), \exists z_2 \in \phi_L(X_2, y_2), z_1 \sqsubseteq z_2 \\ \forall z_2 \in \phi_L(X_2, y_2), \exists z_1 \in \phi_L(X_1, y_1), z_1 \sqsubseteq z_2 \end{cases} \tag{7.25}$$

where $z_1 = \phi(x_1, y_1)$ and $z_2 = \phi(x_2, y_2)$ such that $x_1 \sqsubseteq x_2$ (whose existence is guaranteed by Eq.(7.23)) and $y_1 \sqsubseteq y_2$ (which is given Eq.(7.22)). Since ϕ is coherent, $(x_1, y_1) \sqsubseteq (x_2, y_2) \Rightarrow \phi(x_1, y_1) \sqsubseteq \phi(x_2, y_2)$, i.e. $z_1 \sqsubseteq z_2$.

Finally, according to Eq.(7.25) and Definition 7.2.1, we obtain that $\phi_L(X_1, y_1) \sqsubseteq \phi_L(X_2, y_2)$, i.e. ϕ_L is coherent. $\square$

7.3.2 Reinterpretation of FDMs: a case study

This case study is extracted from Misuri et al. (2018), which aims at analyzing the security of a storage farm when an attack happens.

The system is pictured Figure 7.5. The main storage farm area (outlined in white) is composed of eight tanks (of different volumes) and a loading dock, where ships can charge and discharge chemicals and materials. For the sake of simplicity, only a single attack made with improvised explosive device (IED) was considered, which could be detonated with a remote controller. Two possible intrusion paths were considered, i.e. via ground and via water from the area of the loading dock.

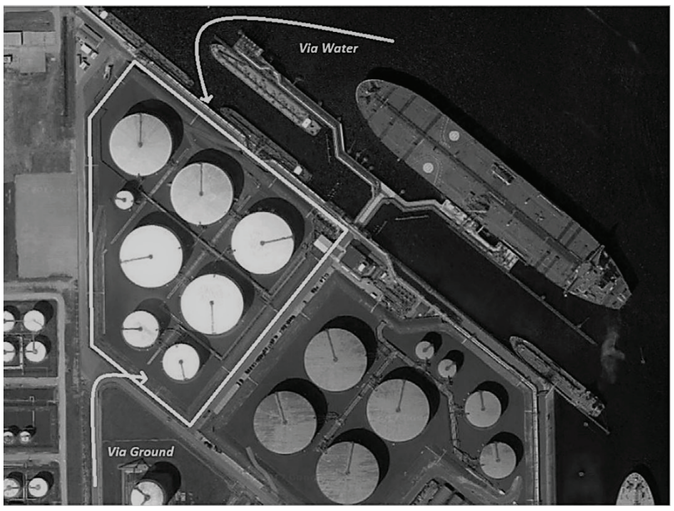


Figure 7.5: Case study in (Misuri et al. 2018), where the premises of the storage farm are outlined in white; the two intrusion paths considered, 'Via Ground' and 'Via Water' are reported as white arrows.

The original FDM of such system (without uncertainty) is given below:

$$\mathcal{M}: \left\{ \begin{array}{rcl} Attack\_OR & := & \vee(AG, AW) \\ Attack\_XOR & := & \sqcup_{XOR}(AG, AW) \\ AG & := & \wedge(UIG, Exp) \\ AW & := & \wedge(UIW, Exp) \\ UIG & := & \wedge(\wedge(FSL, CCTV), SF) \\ UIW & := & \wedge(\wedge(CCTV, SF), Doc) \\ Exp & := & \wedge(IED, Reg) \\ FSL & := & \wedge(\vee(MG, FF), Pat) \\ Doc & := & \wedge(Pat, DB) \end{array} \right\} \tag{7.26}$$

The acronyms are given Table 7.4.

Table 7.4: Definitions of acronyms

AG	Attack via Ground	AW	Attack via Water
UIG	Undetected Intrusion from Ground	UIW	Undetected Intrusion from Water
Exp	Explosion	FSL	First Security Layer
FF	First Fence	SF	Second Fence
Pat	Patrol	MG	Main Gate
Doc	Docking	DB	Docking Barriers
IED	Improvised Explosive Device	Reg	Regress

The valuation domain of each variable is the binary FDS $\mathbf{WF}$. But the meanings of W and F are different for different variables:

– For those variables representing a certain defense against the attack: W means that the defense is working and F means that the defense is failed. The variables belong to this kind are: Pat, DB, Doc, MG, FF, FSL, $CCTV$, SF.

– For those variables representing a certain attack: W means that the attack is failed and F means that the attack is succeeded. The variables belong to this kind are: IED, Reg, Exp, UIG, UIW, AW, AG, $Attack\_OR$, $Attacl\_XOR$.

In $\mathcal{M}$, there are two root variables: $Attack\_OR$ and $Attack\_XOR$. The former means that the attack is succeed if at least one of the attacks AG and AW is succeed. Therefore, we use the join operator $\vee$ to model $Attack\_OR$ from AG and AW. The latter means that the attack is succeed if only one of the attacks AG and AW is succeed. Therefore, we use the exclusive-OR operator $\sqcup_{XOR}$ to

Table 7.5: The valuation of $\sqcup_{XOR} : \mathbf{WF}^2 \twoheadrightarrow \mathbf{WF}$.

$\sqcup_{XOR}(u, v)$		v	
		W	F
u	W	W	F
	F	F	W

model $Attack\_XOR$ from AG and AW. The valuation of $\vee$ and $\wedge$ can be found Section 6.2.2. The valuation of $\sqcup_{XOR}$ is given Table 7.5.

In the sequel, two cases are presented to show the modeling of uncertain valuation of state variables. In case 1, the valuation of all state variables of $\mathbf{S} = \{MG, FF, Pat, DB, CCTV, SF, IED, Reg\}$ is considered to be uncertain. This case is the same as the one in Misuri et al. (2018). In case 2, we assume that only part of the state variables have uncertain valuation.

Case 1 Assume that the valuation of all state variables in $\mathbf{S}$ is uncertain. Then, for every state variable $v \in \mathbf{S}$, its valuation domain becomes $\text{dom}(v) = (\mathbf{WF})^u$. Accordingly, we should transform the operations in this model in correct forms using the transformations defined Section 7.3.1. The resultant model is given as follows:

$$\mathcal{M}_1 : \left\{ \begin{array}{rcl} Attack\_OR & := & \vee_u(AG, AW) \\ Attack\_XOR & := & (\sqcup_{XOR})_u(AG, AW) \\ AG & := & \wedge_u(UIG, Exp) \\ AW & := & \wedge_u(UIW, Exp) \\ UIG & := & \wedge_u(\wedge_u(FSL, CCTV), SF) \\ UIW & := & \wedge_u(\wedge_u(CCTV, SF), Doc) \\ Exp & := & \wedge_u(IED, Reg) \\ FSL & := & \wedge_u(\vee_u(MG, FF), Pat) \\ Doc & := & \wedge_u(Pat, DB) \end{array} \right\} \tag{7.27}$$

In this case,

$$[\![\mathcal{M}_1]\!] : ((\mathbf{WF})^u)^8 \twoheadrightarrow ((\mathbf{WF})^u)^9$$

Case 2 In this case, we assume that only the valuation of Pat and Reg is considered to be uncertain. As results, their valuation domains are first transformed into the epistemic space, i.e. $\text{dom}(Pat) = \text{dom}(Reg) = (\mathbf{WF})^u$. The valuation domain of other variables is still $\mathbf{WF}$. Similarly, we transform the operations in correct forms according to the transformations defined Section 7.3.1 and the resultant model is given as follows:

$$\mathcal{M}_2 : \left\{ \begin{array}{rcl} Attack\_OR & := & \vee_u(AG, AW) \\ Attack\_XOR & := & (\sqcup_{XOR})_u(AG, AW) \\ AG & := & \wedge_u(UIG, Exp) \\ AW & := & \wedge_u(UIW, Exp) \\ UIG & := & \wedge_L(\wedge_L(FSL, CCTV), SF) \\ UIW & := & \wedge_R(\wedge(CCTV, SF), Doc) \\ Exp & := & \wedge_R(IED, Reg) \\ FSL & := & \wedge_R(\vee(MG, FF), Pat) \\ Doc & := & \wedge_L(Pat, DB) \end{array} \right\} \qquad (7.28)$$

In this case,

$$[\![\mathcal{M}_2]\!] : \mathbf{WF}^6 \otimes ((\mathbf{WF})^u)^2 \twoheadrightarrow ((\mathbf{WF})^u)^9$$

7.3.3 Uncertainty analysis

Theoretically, the FDMs transformed in the epistemic space are still FDMs. It means that the scenarios analysis and the probability calculations presented Chapter 5 can also be applied to assess the FDMs in epistemic space.

Scenarios analysis

The scenarios analysis of the two models $\mathcal{M}_1$ and $\mathcal{M}_2$ in Eq.(7.27) and Eq.(7.28) is implemented. The observers used in the scenarios analysis are $Attack\_OR = \{W\}$, $Attack\_OR = \{W, F\}$, $Attack\_OR = \{F\}$, $Attack\_XOR = \{W\}$, $Attack\_XOR = \{W, F\}$ and $Attack\_XOR = \{F\}$.

The results of the number of scenarios and critical scenarios for $\mathcal{M}_1$ and $\mathcal{M}_2$ are given Table 7.6.

Table 7.6: Numbers of scenarios and critical scenariosfor $\mathcal{M}_1$ and $\mathcal{M}_2$.

	$\mathcal{M}_1$			$\mathcal{M}_2$		
Y	$\{W\}$	$\{W, F\}$	$\{F\}$	$\{W\}$	$\{W, F\}$	$\{F\}$
$\lvert\mathbf{Sce}(Attack\_OR = Y)\rvert$	5729	813	19	548	21	7
$\lvert\text{Min}\mathbf{Sce}(Attack\_OR = Y)\rvert$	1	3	3	1	3	3
$\lvert\text{Max}\mathbf{Sce}(Attack\_OR = Y)\rvert$	6	6	1	6	2	1
$\lvert\mathbf{Sce}(Attack\_XOR = Y)\rvert$	5734	821	6	551	21	4
$\lvert\text{Min}\mathbf{Sce}(Attack\_XOR = Y)\rvert$	1	3	3	1	3	3
$\lvert\text{Max}\mathbf{Sce}(Attack\_XOR = Y)\rvert$	1	7	2	1	2	2

We will not list all the critical scenarios here. But, we found that the scenarios in $\text{Min}\mathbf{Sce}(Attack\_OR = \{W\})$, $\text{Max}\mathbf{Sce}(Attack\_OR = \{W\})$, $\text{Min}\mathbf{Sce}(Attack\_XOR =$

$\{F\}$) and Max**Sce**$(Attack\_XOR = \{F\})$ obtained from $\mathcal{M}_1$ are equivalent to relevant critical scenarios obtained from $\mathcal{M}_2$. Moreover, we also perform the analysis of the original model $\mathcal{M}$ in Eq.(7.26). We finally found that those sets of scenarios are also equivalent to the ones obtained from $\mathcal{M}$. Therefore, we draw the conclusion that in this system, the introduction of epistemic uncertainties doesn't influence the critical scenarios of the observers where the valuation of $Attack\_OR$ and $Attack\_XOR$ is certain, i.e. $\{W\}$ or $\{F\}$.

To be concrete, we list the two sets of critical scenarios Max**Sce**$(Attack\_OR = \{W\})$ and Min**Sce**$(Attack\_OR = \{F\})$ obtained from $\mathcal{M}_1$ as follows:

$$\begin{array}{rcl}
\text{Max}\mathbf{Sce}(Attack\_OR = \{W\}) & = & \{(\{F\},\{F\},\{F\},\{F\},\{F\},\{F\},\{W\},\{F\}),\\
& & (\{F\},\{F\},\{F\},\{F\},\{F\},\{W\},\{F\},\{F\}),\\
& & (\{F\},\{F\},\{F\},\{F\},\{W\},\{F\},\{F\},\{F\}),\\
& & (\{F\},\{F\},\{F\},\{W\},\{F\},\{F\},\{F\},\{F\}),\\
& & (\{F\},\{F\},\{W\},\{F\},\{F\},\{F\},\{F\},\{F\}),\\
& & (\{W\},\{W\},\{F\},\{F\},\{F\},\{F\},\{F\},\{W\})\}\\
\text{Min}\mathbf{Sce}(Attack\_OR = \{F\}) & = & \{(\{W\},\{W\},\{F\},\{F\},\{F\},\{F\},\{F\},\{F\}),\\
& & (\{W\},\{F\},\{F\},\{F\},\{F\},\{F\},\{F\},\{W\}),\\
& & (\{F\},\{W\},\{F\},\{F\},\{F\},\{F\},\{F\},\{W\})\}
\end{array}$$

The valuation order of each scenario above is the same with the variable ordering in DD(f_{Attack_OR}) for $\mathcal{M}_1$, i.e. $MG \prec FF \prec Pat \prec CCTV \prec SF \prec IED \prec Reg \prec DB$.

Now, consider the observer $Attack\_OR = \{W, F\}$ where the valuation of $Attack\_OR$ is uncertain. Denote this observer by o.

For $\mathcal{M}_1$, the critical scenarios of o are listed as follows:

$$\begin{array}{rcl}
\text{Min}\mathbf{Sce}(o)_1 & = & \{(\{W,F\},\{W\},\{W,F\},\{W,F\},\{W,F\},\{W,F\},\{W,F\},\{W\}),\\
& & (\{W\},\{W,F\},\{W,F\},\{W,F\},\{W,F\},\{W,F\},\{W,F\},\{W\}),\\
& & (\{W\},\{W\},\{W,F\},\{W,F\},\{W,F\},\{W,F\},\{W,F\},\{W,F\})\}\\
\text{Max}\mathbf{Sce}(o)_1 & = & \{(\{F\},\{F\},\{W,F\},\{F\},\{F\},\{F\},\{F\},\{F\}),\\
& & (\{F\},\{F\},\{F\},\{W,F\},\{F\},\{F\},\{F\},\{F\}),\\
& & (\{F\},\{F\},\{F\},\{F\},\{W,F\},\{F\},\{F\},\{F\}),\\
& & (\{F\},\{F\},\{F\},\{F\},\{F\},\{W,F\},\{F\},\{F\}),\\
& & (\{F\},\{F\},\{F\},\{F\},\{F\},\{F\},\{W,F\},\{F\}),\\
& & (\{W,F\},\{W,F\},\{F\},\{F\},\{F\},\{F\},\{F\},\{W,F\})\}
\end{array} \tag{7.29}$$

For $\mathcal{M}_2$, the critical scenarios of o are listed as follows:

$$\begin{aligned}\text{Min}\mathbf{Sce}(o)_2 &= \{(W,W,\{W,F\},F,F,F,\{W,F\},F),\\ &\quad (W,F,\{W,F\},F,F,F,\{W,F\},W),\\ &\quad (F,W,\{W,F\},F,F,F,\{W,F\},W)\}\\ \text{Max}\mathbf{Sce}(o)_2 &= \{(F,F,\{F\},F,F,F,\{W,F\},F),\\ &\quad (F,F,\{W,F\},F,F,F,\{F\},F)\}\end{aligned} \tag{7.30}$$

Comparing the critical scenarios of o for $\mathcal{M}_1$ and $\mathcal{M}_2$, we can see that the two scenarios in Max$\mathbf{Sce}(o)_2$ are included in Max$\mathbf{Sce}(o)_1$ and the three scenarios in Min$\mathbf{Sce}(o)_1$ are less degraded than the three scenarios in Min$\mathbf{Sce}(o)_2$. To some extent, the latter indicates qualitatively that the range of uncertain scenarios is enlarged from $\mathcal{M}_2$ to $\mathcal{M}_1$. This enlargement can also be observed from the number of scenarios in Max$\mathbf{Sce}(o)_1$ and Max$\mathbf{Sce}(o)_2$.

Probabilistic indicators

As inputs, we provide both the probability measure p in the domain $\mathbf{WF}$ and the mass assignment m in the domain $(\mathbf{WF})^u$ of each state variable, see Table 7.7 and Table 7.8.

Table 7.7: Probability measure in the domain $\mathbf{WF}$ of state variables without uncertainty.

	MG	FF	Pat	DB	CCTV	SF	IED	Reg
$p(W)$	0.85	0.65	0.75	0.75	0.75	0.65	0.25	0.55
$p(F)$	0.15	0.35	0.25	0.25	0.25	0.35	0.75	0.45

Table 7.8: Mass assignment in the domain $(\mathbf{WF})^u$ of state variables with uncertainty.

	MG	FF	Pat	DB	CCTV	SF	IED	Reg
$m(\{W\})$	0.8	0.6	0.7	0.7	0.7	0.6	0.2	0.5
$m(\{W,F\})$	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
$m(\{F\})$	0.1	0.3	0.2	0.2	0.2	0.3	0.7	0.4

In this probabilistic assessment, the observers are the same as in scenarios analysis. The results are pictured Figure 7.6.

From Figure 7.6, we can see that the mass that the attack is failed (i.e. $m(\{W\})$) accounts for the largest proportion in the four diagrams. If we make a comparison between the results of $\mathcal{M}_1$ and $\mathcal{M}_2$, we can see that the range of uncertainties (i.e. $m(\{W,F\})$) is reduced from $\mathcal{M}_1$ to $\mathcal{M}_2$, while mass/belief that the attack is certain to be succeed (i.e. $m(\{F\})$) is enlarged from $\mathcal{M}_1$ to $\mathcal{M}_2$.

Based on the results of Figure 7.6, we can calculate the other probabilistic indicators, i.e. $\mathbf{Bel}$, $\mathbf{Pl}$, $\mathbf{Best}$ and $\mathbf{Worst}$. The results are given Table 7.9.

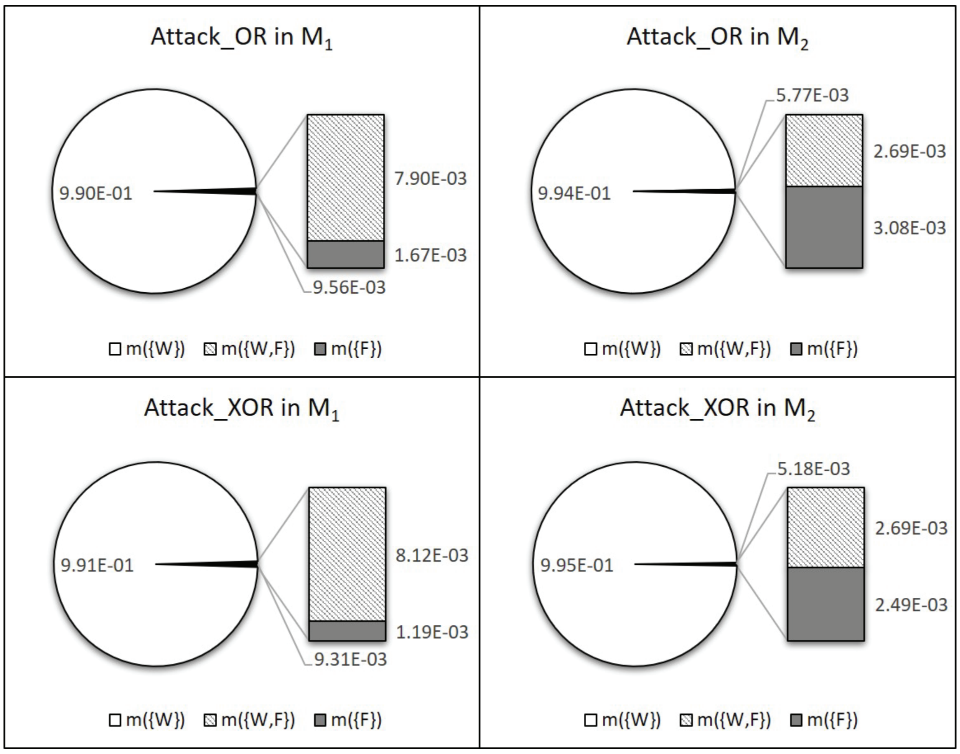


Figure 7.6: The mass assignment m in the domain of $Attack\_OR$ and $Attack\_XOR$ for $\mathcal{M}_1$ and $\mathcal{M}_2$.

As future work, we plan to apply the proposed modeling method to more complex systems, for instance the safety instrumented system presented Section 2.3.

Table 7.9: Results of $\mathbf{Bel}$, $\mathbf{Pl}$, $\mathbf{Best}$ and $\mathbf{Worst}$ in the domain of $Attack\_OR$ and $Attack\_XOR$ for $\mathcal{M}_1$ and $\mathcal{M}_2$.

	$\mathcal{M}_1$		$\mathcal{M}_2$	
	$Attack\_OR$	$Attack\_XOR$	$Attack\_OR$	$Attack\_XOR$
$\mathbf{Bel}(\{W\})$	9.904×10^{-1}	9.906×10^{-1}	9.942×10^{-1}	9.948×10^{-1}
$\mathbf{Bel}(\{W, F\})$	1.000	1.000	1.000	1.000
$\mathbf{Bel}(\{F\})$	7.895×10^{-3}	1.193×10^{-3}	3.075×10^{-3}	2.487×10^{-3}
$\mathbf{Pl}(\{W\})$	9.921×10^{-1}	9.987×10^{-1}	9.969×10^{-1}	9.975×10^{-1}
$\mathbf{Pl}(\{W, F\})$	1.000	1.000	1.000	1.000
$\mathbf{Pl}(\{F\})$	9.562×10^{-3}	9.918×10^{-1}	9.973×10^{-1}	9.973×10^{-1}
$\mathbf{Best}(W)$	1.000	1.000	1.000	1.000
$\mathbf{Best}(F)$	7.895×10^{-3}	1.193×10^{-3}	3.075×10^{-3}	2.487×10^{-3}
$\mathbf{Worst}(W)$	9.904×10^{-1}	9.906×10^{-1}	9.942×10^{-1}	9.948×10^{-1}
$\mathbf{Worst}(F)$	1.000	1.000	1.000	1.000

Chapter 8

Summary of main results and future work

8.1 Summary of main results

The primary purpose of this thesis is to propose a unified framework of reliability combinatorial models for probabilistic risk/safety analyses. A summary of the main results under this main objective are presented as follows.

8.1.1 Modeling framework of finite degradation structures

The proposed modeling framework is called finite degradation structures (FDSs). This framework extends formally the fault trees analysis from Boolean systems into multistate systems.

To realize such extension, we first define mathematically the algebraic framework of FDSs, i.e. $\langle \mathbf{FDS}, \otimes, \Phi \rangle$. Figure 8.1 shows a comparison between the Boolean algebra used for fault tree analysis and the algebraic framework of FDSs.

- **Valuation domain**: In fault tree analysis, the valuation domain of each variable is a Boolean lattice $\{0, 1\}$. Conventionally, 1 stands for the failed state and 0 stand for the working state. In FDSs, the valuation domain is changed to meet-semi-lattice. This change makes FDSs in essence more generic and more appropriate to model the state space of multistate component/system.

- **Orders**: The orders in the Boolean lattice $\{0, 1\}$ is theoretically defined as $0 < 1$. The interpretation of this order depends on its application field. For

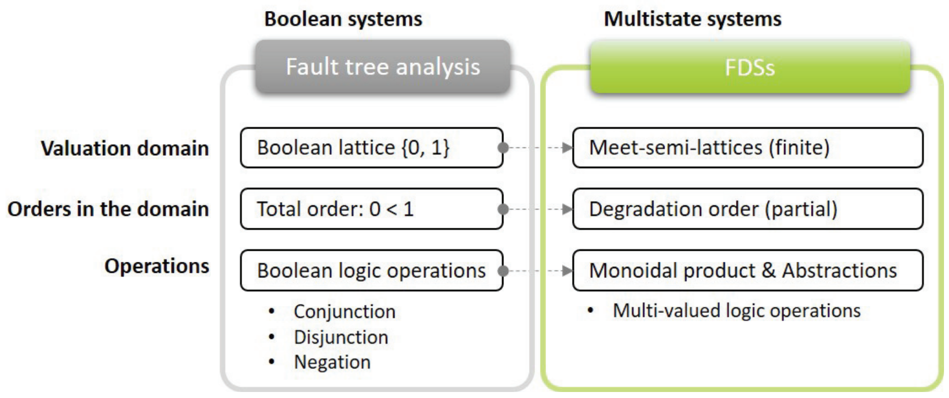


Figure 8.1: Comparison of the algebraic framework between fault trees and FDSs.

reliability and safety analyses, we interpret this order as the degradation order among states, i.e. 0 (working state) is less degraded than 1 (failed state). The orders in FDSs are also interpreted as the degradation order. Mathematically, the degradation order is a partial order that allows having incomparable states — i.e. whose degradation level is incomparable or undecidable — in a FDS.

- **Operations**: In classical fault trees, three operations can be used to model the system, which are the Boolean logic operations, i.e. conjunction, disjunction and negation. In FDSs, we propose two fundamental operations, i.e. the monoidal product $\otimes$ on FDSs and the abstractions Φ between FDSs. Based on these two operations, we can interpret any multi-valued logic operations into the framework of FDSs. This allows analysts to design and customize their own operations according to needs.

The main contribution of the algebraic framework of FDSs is that it breaks theoretically the binary assumption in fault tree analysis.

An important remark here is that in fault tree analysis, the binary assumption restricts not the probabilistic calculation but the determination of minimal cut/path sets. Theoretically, probability measures can always be defined and calculated in multistate systems, since there is never such a limitation that a random variable can only take two values.

The reason why the binary assumption impacts on the determination of minimal cut/path sets is that the notion of minimal cut/path sets in fault trees coincides with the notion of prime implicants in Boolean logic, and moreover, they both depend implicitly on the order $0 < 1$ in Boolean lattices. In other words, the minimality of a cut/path set is mathematically characterized by this order.

Therefore, after a careful mathematical demonstration, we found that to determine minimal cut/path sets in multistate systems, we should also order the states of the multistate component/system. For PRA/PSA, this order should be recognized as the degradation order among states. For this reason, we choose meet-semi-lattices as valuation domains of variables and equip each of them with a probability measure to support the probabilistic calculation in FDSs.

The second comparison is made between the models built on FDSs and the fault trees (as models built on Boolean algebra). The models built on the framework of FDSs are called finite degradation models (FDMs). FDMs can be seen as natural extension of fault trees in multistate realm. Figure 8.2 shows a comparison between fault trees and FDMs.

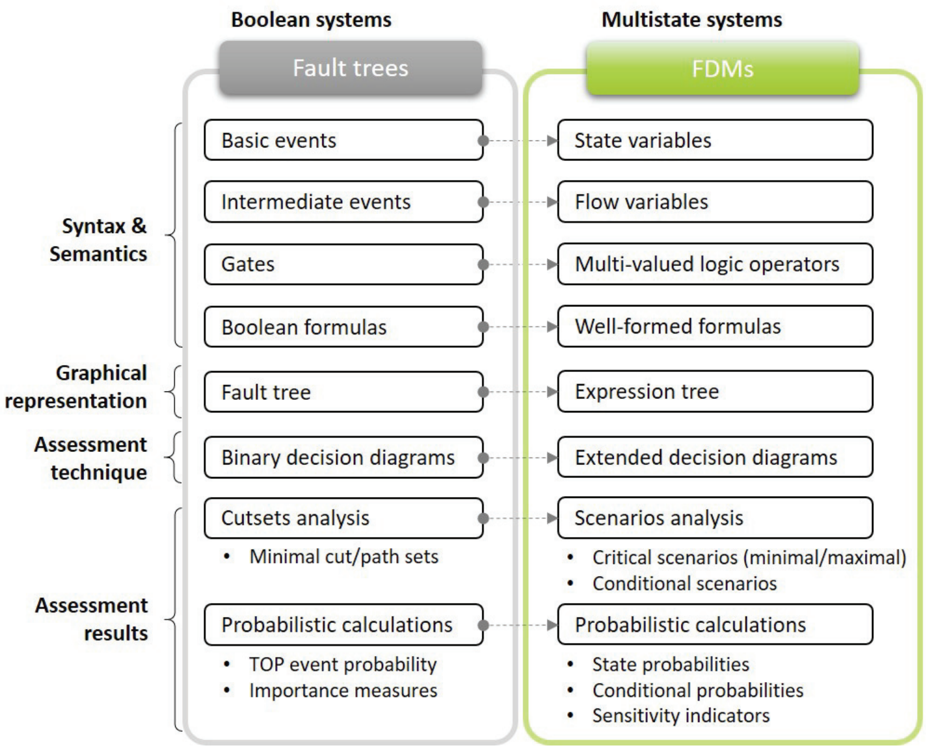


Figure 8.2: Comparison between fault trees and FDMs.

- **Syntax and semantics**: Syntactically, fault trees encode Boolean formulas and FDMs encode well-formed formulas. The former is interpreted as

Boolean functions, while the latter is interpreted as multi-valued logic functions. The basic events and the intermediate events in fault trees correspond respectively to the state variables and the flow variables in FDMs. The gates in fault trees correspond to the multi-valued logic operators in FDMs.

- **Graphical representation**: A fault tree itself is the graphical representation of the Boolean function it encodes. For FDMs, we also use a binary expression tree to encode its syntactic structure.
- **Assessment technique**: The most appropriate technique to assess logic functions is the decision diagram technique. Binary Decision Diagrams (BDDs) are used to extract minimal cutsets from fault trees. In FDMs, we modified the BDD to fit the multi-valued operations. A detailed comparison of these two types of decision diagrams will be given next section.
- **Assessment results**: As qualitative analysis, the cutsets analysis in fault trees can be fully covered by the scenarios analysis in FDMs. The notion of minimal cutsets is lifted up to the notion of minimal scenarios, which represent the minimal ways that the system degrades from an operation state to an undesired state. The notion of minimal path set is lifted up to the notion of maximal scenarios, which represent the maximal capability that the system can remain in a good state despite of the occurrence of failures. As quantitative results, the probabilistic indicators that can be calculated in fault trees can be calculated in FDMs. These indicators provide the mathematical bases based on which other reliability and safety indicators can be deduced.

From Figure 8.1 and Figure 8.2, we can see that the fault tree analysis is formally and fully extended into multistate systems. As a unified framework of combinatorial models in PRA/PSA, the modeling framework of FDSs can be applied in all cases where the system's behavior is modeled by the combination of behaviors of its components. Moreover, FDMs can also be used to abstract other models — e.g. state/transition models like Markov chains — to support the combinatorial and systematic analysis of their behaviors.

8.1.2 Calculation algorithms

Similar to fault tree analysis, the decision diagrams are also used to implement the required calculations in FDMs. A comparison between the BDDs and the extended decision diagrams for FDMs is given Figure 8.3.

- **Node definitions**: In BDDs, a terminal node is a constant node valued in the Boolean domain $\{0, 1\}$ and an internal node is a variable node (v, n_1, n_2)

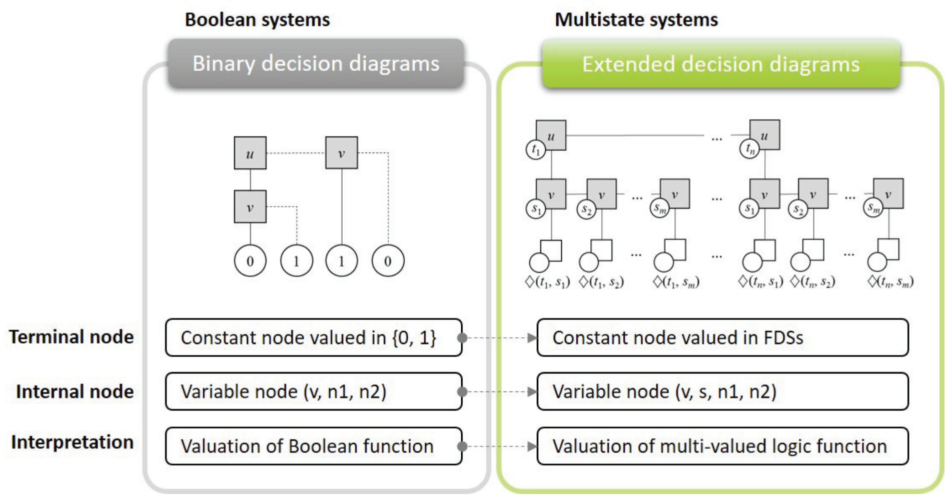

Boolean systems
Multistate systems
Binary decision diagrams
Extended decision diagrams
Terminal node
Internal node
Interpretation
Constant node valued in {0, 1}
Variable node (v, n1, n2)
Valuation of Boolean function
Constant node valued in FDSs
Variable node (v, s, n1, n2)
Valuation of multi-valued logic function

Figure 8.3: The binary decision diagrams and the extended decision diagrams for FDMs.

labeled with a variable v and pointing to the two child-nodes n_1 (then-child) and n_2 (else-child). In our extended decision diagrams, a terminal node is a constant node valued in a FDS and an internal node is a variable node (v, s, n_1, n_2) labeled with a variable v and a state constant s, and pointing to the two child-nodes n_1 (then-child) and n_2 (else-child).

- **Interpretation**: BDDs are used to encode the valuation of Boolean functions, while our extended decision diagrams are used to encode the valuation of multi-valued logic functions.

In our extended decision diagrams, we keep the binary connection of nodes so that existing algorithms that have been well applied in BDDs can be easily extended to our extended decision diagrams. To fit the syntax and the semantics of FDMs, these algorithms are well adjusted.

Here, we also want to mention that the multistate multi-valued decision diagram (MMDD) introduced Section 2.2.3 is theoretically an alternative to assess FDMs. But, the difference is that in our extended decision diagrams, we provide two orderings to organize the valuation paths. The primary ordering is the variable ordering similar to the one in BDDs and MMDDs. This ordering defines the valuation priority of different variables in the decision diagram. The secondary ordering is the ordering of the states in the valuation chain of a variable. We have noticed that if the secondary ordering is consistent with the degradation orders, the calculation efficiency can be improved considerably when the system is coherent. The use of this secondary ordering is not included in this thesis but can be found in our paper Rauzy and Yang (2019).

8.1.3 Software and modeling language

We developed a software called **LatticeX** in Python to realize the computer-based modeling and assessment of FDMs. The input of this software is the FDMs written in text files. As output, it can calculate the required scenarios, critical scenarios and state probabilities for a given observer.

The modeling of FDMs is realized by writing FDMs in text files using the modeling language FDS-ML. FDS-ML is an object-oriented language specifically designed to describe FDMs. Using FDS-ML, analysts can define their own FDSs and operators, assign (time-dependent) probabilities to variables and write equations to form FDMs.

The modeling of FDMs is separated from the assessment of FDMs. The latter is realized by a calculation engine in **LatticeX**. A compiler of FDS-ML is embedded in **LatticeX** to translate the input textual FDMs to the calculation engine to be assessed. The algorithms presented this thesis are currently used in this calculation engine. The current version of **LatticeX** is the first version and in this version we mainly focus on its functionality and correctness. The improvement of computational efficiency will be included in our future work.

8.2 Recommendations for future work

The modeling framework presented this thesis is a new tool for reliability and safety analyses. What we have done is just the first trial. The future work that can contribute to complete the proposed modeling framework in both theoretical and practical perspectives is summarized as follows.

Direction 1. Completing the theoretical framework

First, we want to include the determination of importance measures in FDMs. Importance measures can be used to identify the weakest component of a system and to support system improvement activities. For Boolean systems, many types of importance measures have been put forward, such as Birnbaum importance, risk achievement/deduction worth, criticality importance, Fussell-Vesely's measure, etc. For multistate systems, there are also many researches dedicated to the analysis of components' importance, see e.g. Chang et al. (2005), Si et al. (2012), Zaitseva et al. (2015) and Kvassay et al. (2017). Although the importance measures have not been included in FDMs yet, we have already defined the conditional probabilities and the sensitivity factors (i.e. partial derivative of probabilistic indicators), which can be used as the mathematical bases to define importance measures in FDMs.

Second, we want to put more efforts on the analyses of non-coherent systems. As presented Section 5.1.3, the scenarios analysis for coherent systems can be reduced to the analysis of critical scenarios, thanks to the order-similarity between the components and the system as a whole. But for non-coherent systems, such order-similarity doesn't exist anymore. In this case, what can be the good indicator to characterize the criticality of scenarios? A possible idea is to use the upper set as a coherent hull of the set of scenarios. This idea has been presented in our paper Rauzy and Yang (2019). Other solutions may also exist and will be included in our future work.

Direction 2. Enlarging the modeling library

This thesis provides a small modeling library of the operators that have been applied in reliability and safety analyses. This modeling library includes currently the logic conjunction, disjunction and negation operators; the join and meet operators; the k-out-of-n operator; the operator for hot-standby systems; and the series and parallel operators for safety instrumented systems. For future work, we may consider to enlarge this modeling library by adding more operators in different fields of applications.

Direction 3. Improving the calculation algorithms

The techniques that have been dedicated to improve the computational efficiency of algorithms on binary decision diagrams can be potentially applied to our calculation algorithms. For instance, we can use the cache technique to avoid repeated operations; we can minimize the size of diagram by defining suitable node reduction rules and optimizing the variable ordering; and we can use the secondary ordering — the ordering of states in a valuation chain of a variable — to cut off non-critical scenarios.

Direction 4. Upgrading the software $\mathbf{LatticeX}$

The current version of $\mathbf{LatticeX}$ is just a demonstration version, which contains only the basic functions of the proposed modeling framework. Since the main objective of this PhD is not the software development, we didn't put many efforts on its improvement yet. For future improvements (if required), the following directions can be considered: (1) integrate the modeling library in $\mathbf{LatticeX}$ so that the domains and the operators can be directly used in the model. (2) improve the modeling language FDS-ML, increase its expressive power and possibly add graphical user interface to visualize FDMs.

Direction 5. Supporting the synchronization of models between MBSE amd MBSA

To face the increasing complexity of technical systems, engineering disciplines contributing to the design and the operation of these systems are more and more relying on the so-called *model-based approach*. This is especially the case for system architectures and safety analyses. These two disciplines can be seen as the two faces of the same medal: *"systems architects are in charge of describing how the system works while safety analysts are in charge of studying how it may fail."*

For this reason, it is of primary importance to both ensure an effective communication between them and to keep their activities separated, so to avoid conflicts of interest. Although both consider the system from a holistic perspective, these two disciplines are of different natures and this difference reflects on the type of models that are designed. Models designed by system architects, using typically modeling languages such as SysML (Friedenthal et al. 2014), are pragmatic. They aim at supporting a seamless communication between the stakeholders. Models designed by safety analysts are formal in sense of mathematics. They aim at calculating risk indicators (Andrews 2002). It does not mean however that they should have nothing in common. Rather, their commonalities should be considered dynamically, i.e. by understanding that models are not designed once for all but evolve iteratively throughout the life cycle of systems. The problem should be thought in terms of *model synchronization*, i.e. of periodical rendezvous during which models are checked for consistency (Legendre et al. 2017).

As mentioned Section 2.2, PRA/PSA models can be roughly classified into two categories: combinatorial models and state automata. The former *abstract* away the latter by considering only states of the system not transitions between these states. In the framework of FDSs, we make it possible to formally define conditions under which a model is a correct abstraction of another one, via a suitable coherent operation. This makes FDSs a good candidate to be the mathematical ground on which models to synchronize *behavioral descriptions* for both system architecture and safety analyses are built, i.e. to support the interface between model-based systems engineering (MBSE) and model-based safety assessment (MBSA). This idea is presented in our paper Yang et al. (2018). For future work, we hope to find more practical case studies that can be used to concretely illustrate such synchronization supported by FDSs.

8.3 Closing remarks

In ancient times, the soldiers make weapons themselves. But now, few soldiers do so because our society is moving towards in a direction that the division of labour is more and more refined, i.e. there are more expertised people who make advanced weapons for soldiers.

When the war is won, it is hard to decide whether the soldiers or the people who made the weapons should be awarded the golden medal. In fact, they are not competitors. They are collaborators. They won the war together and their interdependence promotes both the development of advanced weapons and the training of more skilled soldiers.

"This division of labour, from which so many advantages are derived, is not originally the effect of any human wisdom, which foresees and intends that general opulence to which it gives occasion." — Adam Smith «The Wealth of Nations»

In engineering fields, there are engineers who work on the implementation of real systems, there are analysts who design models and perform the required analysis, and there are also researchers who design new methodology, models and tools to facilitate the work of them both. No one is and no one should be more dominant than the others, since they all contribute to the opulence of engineering.

The modeling framework proposed this thesis is like a new weapon for reliability and safety analysts. We hope that on the one hand, it can be useful in practical applications and on the other hand, it can also bring new ideas or inspirations to other similar researches.

There is no war in science. We are all contributors with the common faith:

"Kunnskap for en bedre verden." (Knowledge for a better world.) — NTNU vision

Bibliography

Adachi, M., Papadopoulos, Y., Sharvia, S., Parker, D., and Tohdo, T. (2011). An approach to optimization of fault tolerant architectures using hip-hops. *Software: Practice and Experience*, 41(11):1303–1327.

Agarwal, H., Renaud, J. E., Preston, E. L., and Padmanabhan, D. (2004). Uncertainty quantification using evidence theory in multidisciplinary design optimization. *Reliability Engineering and System Safety*, 85(1):281–294.

Akers, J. (1978). Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6):509–516.

Andrews, J. (2002). Reliability and risk assessment.

Barlow, R. E. and Wu, A. S. (1978). Coherent systems with multi-state components. *Mathematics of operations research*, 3(4):275–281.

Batteux, M., Prosvirnova, T., Rauzy, A., and Kloul, L. (2013). The altarica 3.0 project for model-based safety assessment. In *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, pages 741–746. IEEE.

Bjerring, J. (2014). Problems in epistemic space. *Journal of Philosophical Logic*, 43(1):153–170.

Boudali, H., Crouzen, P., and Stoelinga, M. (2007a). A compositional semantics for dynamic fault trees in terms of interactive markov chains. In *International Symposium on Automated Technology for Verification and Analysis*, pages 441–456. Springer.

Boudali, H., Crouzen, P., and Stoelinga, M. (2007b). Dynamic fault tree analysis using input/output interactive markov chains. In *37th Annual IEEE/IFIP In-*

ternational Conference on Dependable Systems and Networks (DSN'07), pages 708–717. IEEE.

Bouissou, M. and Bon, J.-L. (2003). A new formalism that combines advantages of fault-trees and markov models: Boolean logic driven markov processes. *Reliability Engineering and System Safety*, 82(2):149–163.

Bouissou, M., Bruyere, F., and Rauzy, A. (1997). Bdd based fault-tree processing: A comparison of variable ordering heuristics. In *Proceedings of European Safety and Reliability Association Conference, ESREL'97*.

Brace, K., Rudell, R., and Bryant, R. (1991). Efficient implementation of a bdd package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, DAC '90, pages 40–45. ACM.

Bruce, K. B. (2002). Foundations of object-oriented languages : types and semantics.

Bryant (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691.

Buchacker, K. (2000). Modeling with extended fault trees. In *Proceedings. Fifth IEEE International Symposium on High Assurance Systems Engineering (HASE 2000)*, volume 2000-, pages 238–246. IEEE.

Buchacker, K. et al. (1999). Combining fault trees and petri nets to model safety-critical systems. In *High performance computing*, pages 439–444. The Society for Computer Simulation International.

Chang, Y.-R., Amari, S. V., and Kuo, S.-Y. (2005). Obdd-based evaluation of reliability and importance measures for multistate systems subject to imperfect fault coverage. *IEEE Transactions on Dependable and Secure Computing*, 2(4):336–347.

Chaux, P.-Y., Roussel, J.-M., Lesage, J.-J., Deleuze, G., and Bouissou, M. (2013). Towards a unified definition of minimal cut sequences. *IFAC Proceedings Volumes*, 46(22):1–6.

Codetta-Raiteri, D. (2005). The conversion of dynamic fault trees to stochastic petri nets, as a case of graph transformation. *Electronic Notes in Theoretical Computer Science*, 127(2):45–60.

Dempster, A. P. (1967). Upper and lower probabilities induced by a multivalued mapping. *The Annals of Mathematical Statistics*, 38(2):325–339.

Dugan, J., Bavuso, S., and Boyd, M. (1990). Fault-trees and sequence dependencies. *Proceedings Annual Reliability And Maintainability Symposium*, (SYM):286–293.

Dugan, J., Bavuso, S., and Boyd, M. (1992). Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on Reliability*, 41(3):363–377.

El-Neweihi, E., Proschan, F., and Sethuraman, J. (1978). Multistate coherent systems. *Journal of Applied Probability*, 15(4):675–688.

Friedenthal, S., Moore, A., and Steiner, R. (2014). *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann.

Fussell, J. (1972). A new methodology for obtaining cut sets for fault trees. *Trans. Am. Nucl. Soc.*

Fussell, J. (1975). A review of fault tree analysis with emphasis on limitations. *IFAC Proceedings Volumes*, 8(1):552–557.

Griffith, W. S. (1980). Multistate reliability models. *Journal of Applied Probability*, 17(3):735–744.

Gudemann, M. and Ortmeier, F. (2010). A framework for qualitative and quantitative formal model-based safety analysis. In *2010 IEEE 12th International Symposium on High Assurance Systems Engineering*, pages 132–141. IEEE.

Helton, J. C. and Burmaster, D. E. (1996). Guest editorial: treatment of aleatory and epistemic uncertainty in performance assessments for complex systems.

Hirsch, W. M., Meisner, M., and Boll, C. (1968). Cannibalization in multicomponent systems and the theory of reliability. *Naval Research Logistics Quarterly*, 15(3):331–360.

Hochberg, M. (1973). Generalized multicomponent systems under cannibalization. *Naval Research Logistics Quarterly*, 20(4):585–605.

Huang, X. (1984). The generic method of the multistate fault tree analysis. *Microelectronics Reliability*, 24(4):617–622.

Ibañez-Llano, C. and Rauzy, A. (2008). Variable ordering heuristics for bdd based on minimal cutsets. In *Proceedings of the International Probabilistic Safety Assessment and Management Conference (PSAM 9) Hong Kong*, pages 18–23.

IEC61508 (2010). International iec standard iec61508 - functional safety of electrical/electronic/programmable safety-related systems (e/e/pe, or e/e/pes). Standard, International Electrotechnical Commission, Geneva, Switzerland.

International, S. (1996). *Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment*. SAE International.

ISO12489 (2013). Iso/tr 12489:2013 petroleum, petrochemical and natural gas industries – reliability modelling and calculation of safety systems. Standard, International Organization for Standardization, Geneva, Switzerland.

Janan, X. (1985). On multistate system analysis. *IEEE Transactions on Reliability*, R-34(4):329–337.

Jiang, L., Wang, X., and Liu, Y. (2018). Reliability evaluation of the chinese train control system level 3 using a fuzzy approach. *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, 232(9):2244–2259.

Jung, W. S., Han, S. H., and Ha, J. (2004). A fast bdd algorithm for large coherent fault trees analysis. *Reliability Engineering and System Safety*, 83(3):369–374.

Kaiser, B., Gramlich, C., and Förster, M. (2007). State/event fault trees—a safety analysis model for software-controlled systems. *Reliability Engineering and System Safety*, 92(11):1521–1537.

Kolmogorov, A. (1950). Foundations of the theory of probability.

Kumamoto, H. (1996). Probabilistic risk assessment and management for engineers and scientists.

Kvassay, M., Levashenko, V., and Zaitseva, E. (2016). Analysis of minimal cut and path sets based on direct partial boolean derivatives. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 230(2):147–161.

Kvassay, M., Zaitseva, E., and Levashenko, V. (2017). Importance analysis of multi-state systems based on tools of logical differential calculus. *Reliability Engineering & System Safety*, 165:302–316.

Langseth, H. and Lindqvist, B. H. (1998). Uncertainty bounds for a monotone multistate system. *Probability in the Engineering and Informational Sciences*, 12(2):239–260.

Lee, C.-Y. (1959). Representation of switching circuits by binary-decision programs. *The Bell System Technical Journal*, 38(4):985–999.

Legendre, A., Lanusse, A., and Rauzy, A. (2017). Toward model synchronization between safety analysis and system architecture design in industrial contexts. volume 10437, pages 35–49. Springer Verlag.

Levitin, G. (2004). A universal generating function approach for the analysis of multi-state systems with dependent elements. *Reliability Engineering and System Safety*, 84(3):285–292.

Levitin, G. (2005). *The Universal Generating Function in Reliability Analysis and Optimization*. Springer Series in Reliability Engineering. Springer London, London.

Li, S., Si, S., Dui, H., Cai, Z., and Sun, S. (2014). A novel decision diagrams extension method. *Reliability Engineering and System Safety*, 126:107–115.

Li, Y. Y., Chen, Y., Yuan, Z. H., Tang, N., and Kang, R. (2017). Reliability analysis of multi-state systems subject to failure mechanism dependence based on a combination method. *Reliability Engineering and System Safety*, 166:109–123.

Lindqvist, B. H. (2003). Bounds for the reliability of multistate systems with partially ordered state spaces and stochastically monotone markov transitions. *International Journal of Reliability, Quality and Safety Engineering*, 10(3):235–248.

Lisnianski, A. (2003). Multi-state system reliability : assessment, optimization and applications.

Lisnianski, A. (2007). Extended block diagram method for a multi-state system reliability assessment. *Reliability Engineering and System Safety*, 92(12):1601–1607.

Lisnianski, A., Frenkel, I., and Ding, Y. (2010). Multi-state system reliability analysis and optimization for engineers and industrial managers.

Liu, C., Chen, N., and Yang, J. (2015). New method for multi-state system reliability analysis based on linear algebraic representation. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 229(5):469–482.

Marsan, M. A., Balbo, G., Conte, G., Donatelli, S., and Franceschinis, G. (1995). *Modelling with generalized stochastic Petri nets*, volume 292. Wiley New York.

Matsuoka, T. and Kobayashi, M. (1988). Go-flow: a new reliability analysis methodology. *Nuclear Science and Engineering*, 98(1):64–78.

Merle, G., Roussel, J.-M., Lesage, J.-J., and Bobbio, A. (2010). Probabilistic algebraic analysis of fault trees with priority dynamic gates and repeated events. *IEEE Transactions on Reliability*, 59(1):250–261.

Miller, D. M. (1993). Multiple-valued logic design tools. In *[1993] Proceedings of the Twenty-Third International Symposium on Multiple-Valued Logic*, pages 2–11. IEEE.

Minato, S.-I. (1993). Zero-suppressed bdds for set manipulation in combinatorial problems. In *Proceedings of the 30th international Design Automation Conference*, DAC '93, pages 272–277. ACM.

Misuri, A., Khakzad, N., Reniers, G., and Cozzani, V. (2018). Tackling uncertainty in security assessment of critical infrastructures: Dempster-shafer theory vs. credal sets theory. *Safety Science*, 107:62–76.

Mo, Y., Liu, Y., and Cui, L. (2018). Performability analysis of multi-state series-parallel systems with heterogeneous components. *Reliability Engineering and System Safety*, 171:48–56.

Mo, Y., Zhong, F., Liu, H., Yang, Q., and Cui, G. (2013). Efficient ordering heuristics in binary decision diagram–based fault tree analysis. *Quality and Reliability Engineering International*, 29(3):307–315.

Murchland, J. (1975). Fundamental concepts and relations for reliability analysis of multi-state systems. In *Reliability and fault tree analysis*, pages 581–618.

Ohi, F. (2010). Multistate coherent systems. In *Stochastic Reliability Modeling, Optimization And Applications*, pages 3–34. World Scientific Publishing Co. Pte. Ltd.

Ohi, F. (2013). Lattice set theoretic treatment of multi-state coherent systems. *Reliability Engineering & System Safety*, 116:86–90.

Ohi, F. (2016). Stochastic evaluation methods of a multi-state system via a modular decomposition. *Journal of Computational Science*, 17:156–169.

Papadopoulos, Y. and McDermid, J. (1999). Hierarchically performed hazard origin and propagation studies. volume 1698, pages 139–152. Springer Verlag.

Parry, G. W. (1996). The characterization of uncertainty in probabilistic risk assessments of complex systems. *Reliability Engineering and System Safety*, 54(2-3):119–126.

Piriou, P.-Y., Faure, J.-M., and Lesage, J.-J. (2017). Generalized boolean logic driven markov processes: A powerful modeling framework for model-based safety analysis of dynamic repairable and reconfigurable systems. *Reliability Engineering and System Safety*, 163(C):57–68.

Portinale, L. and Codetta-Raiteri, D. (2011). Using dynamic decision networks and extended fault trees for autonomous fdir. In *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, pages 480–484. IEEE.

Prosvirnova, T. (2014). *AltaRica 3.0: a model-based approach for safety analyses*. PhD thesis.

Rasmussen, N. C. (1975). Reactor safety study: An assessment of accident risks in us commercial nuclear power plants. Technical report.

Rausand, M. (2004). System reliability theory : models, statistical methods, and applications.

Rauzy, A. (2001). Mathematical foundations of minimal cutsets. *IEEE Transactions on Reliability*, 50(4):389–396.

Rauzy, A. (2012). Anatomy of an efficient fault tree assessment engine. In *Proceedings of international joint conference PSAM*, volume 11.

Rauzy, A. (2018). Notes on computational uncertainties in probabilistic risk/safety assessment. *Entropy*, 20(3):162.

Rauzy, A. and Yang, L. (2019). Decision diagram algorithms to extract minimal cutsets of finite degradation models. *Information*, 10(12):368.

Rauzy, A. B. (2008). Guarded transition systems: A new states/events formalism for reliability studies. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 222(4):495–505.

Rauzy, A. B. (2011). Sequence algebra, sequence decision diagrams and dynamic fault trees. *Reliability Engineering and System Safety*, 96(7):785–792.

Ruijters, E. and Stoelinga, M. (2015). Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15-16(C):29–62.

Shafer, G. (1976). A mathematical theory of evidence.

Shrestha, A., Xing, L., and Coit, D. W. (2010). An efficient multistate multivalued decision diagram-based approach for multistate system sensitivity analysis. *IEEE Transactions on Reliability*, 59(3):581–592.

Si, S., Dui, H., Zhao, X., Zhang, S., and Sun, S. (2012). Integrated importance measure of component states based on loss of system performance. *IEEE Transactions on Reliability*, 61(1):192–202.

Signoret, J.-P., Dutuit, Y., Cacheux, P.-J., Folleau, C., Collas, S., and Thomas, P. (2013). Make your petri nets understandable: Reliability block diagrams driven petri nets.(report). *Reliability Engineering and System Safety*, 113:61.

Standard, I. (2011). Iso 26262 road vehicles–functional safety. Standard, International Organization for Standardization, Geneva, Switzerland.

Tang, Z. and Dugan, J. (2004). Minimal cut set/sequence generation for dynamic fault trees. *Annual Reliability And Maintainability Symposium, 2004 Proceedings*, pages 207–213.

Ushakov, I. (1986). A universal generating function. *Soviet Journal of Computer and Systems Sciences*, 24(5):118–129.

Ushakov, I. A. (1988). Reliability analysis of multistate systems by means of a modified generating function. *Journal of Information Processing and Cybernetics*, 24(3):131–135.

Verma, A. K., Srividya, A., and Karanki, D. R. (2010). Probabilistic safety assessment. *Reliability and Safety Engineering*, pages 323–369.

Walker, M. and Papadopoulos, Y. (2009). Qualitative temporal analysis: Towards a full implementation of the fault tree handbook. *Control Engineering Practice*, 17(10):1115–1125.

Watson, H. et al. (1961). Launch control safety study. *Bell labs*.

Wood, A. P. (1985). Multistate block diagrams and fault trees. *IEEE Transactions on Reliability*, R-34(3):236–240.

Xing, L. (2007). Efficient analysis of systems with multiple states. In *21st International Conference on Advanced Information Networking and Applications (AINA '07)*, pages 666–672. IEEE.

Xing, L. and Dai, Y. (2009). A new decision-diagram-based method for efficient analysis on multistate systems. *IEEE Transactions on Dependable and Secure Computing*, 6(3):161–174.

Yang, L. and Rauzy, A. (2018). Reliability modeling using finite degradation structures. In *2018 3rd International Conference on System Reliability and Safety (ICSRS)*, pages 168–175. IEEE.

Yang, L. and Rauzy, A. (2019a). Fds-ml: A new modeling formalism for probabilistic risk and safety analyses. In *International Symposium on Model-Based Safety and Assessment*, pages 78–92. Springer.

Yang, L. and Rauzy, A. (2019b). Model synthesis using boolean expression diagrams. *Reliability Engineering and System Safety*, 186:78–87.

Yang, L., Rauzy, A., and Haskins, C. (2018). Finite degradation structures: a formal framework to support the interface between mbse and mbsa. In *2018 IEEE International Systems Engineering Symposium (ISSE)*, pages 1–6. IEEE.

Yang, L., Rauzy, A., and Lundteigen, M. A. (2019). Finite degradation analysis of multiple safety instrumented systems. In *2019 European Safety and Reliability Conference (ESREL 2019)*.

Yu, K., Koren, I., and Guo, Y. (1994). Generalized multistate monotone coherent systems. *IEEE Transactions on Reliability*, 43(2):242–250.

Zaitseva, E. and Levashenko, V. (2013). Multiple-valued logic mathematical approaches for multi-state system reliability analysis. *Journal of Applied Logic*, 11(3):350–362.

Zaitseva, E., Levashenko, V., and Kostolny, J. (2015). Importance analysis based on logical differential calculus and binary decision diagram. *Reliability Engineering & System Safety*, 138:135–144.

Zang, X., Wang, D., Sun, H., and Trivedi, K. (2003). A bdd-based algorithm for analysis of multistate systems with multistate components. 52(12):1608–1618.

Zhang, H.-L., Zhang, C.-Y., Liu, D., and Li, R. (2011). A method of quantitative analysis for dynamic fault tree. In *2011 Proceedings - Annual Reliability and Maintainability Symposium*, pages 1–6. IEEE.

Zhao, J., Cai, Z., Si, W., and Zhang, S. (2019). Mission success evaluation of repairable phased-mission systems with spare parts. *Computers & Industrial Engineering*, 132:248–259.

Appendix A

Appendices

A.1 Attributes of the classes in LatticeX

```
class State:
    def __init__ (self,domain,name):
        self.name=name
        self.domain=domain
    ...

class Domain:
    def __init__(self,name):
        self.name=name
        self.states={}
        self.orders=[]
        self.bottom = None
    ...

class Distribution:
    def __init__(self,name):
        self.name = name
        self.datapoints = [] #(time,probability)
    ...
    def ReaderFromCSVFile(self,filename) ...
    def WriterToCSVFile(self,filename) ...
    ...

class  ProbabilityDistribution:
    def __init__(self,name):
        self.name = name
        self.distributions = {}
    ...

class FiniteDegradationStructure:
    def __init__(self,name):
        self.name = name
        self.domain = None
        self.PD = None #Probability distribution
    ...
```

Figure A.1: The attributes of the classes related to `FDS`.

A.2 Grammar of FDS-ML defined by EBNF

In this section, we provide the formal descriptions of the grammar of FDS-ML. The Extended Backus - Naur Form (EBNF) constructs listed Table A.1 are used.

First, the definition of `Identifier` is defined Figure A.3. In a language, an

```
class Variable:
    def __init__(self,name,FDS):
        self.FDS = FDS
        self.name = name
        self.order = None
    ...

class Operator:
    def __init__(self,name,A,B,C):
        self.name = name
        self.leftDomain = A
        self.rightDomain = B
        self.resultDomain = C
        self.valuations = {}
    ...

class FormulaNode:
    def __init__(self,nodeType,formula):
        self.nodeType = nodeType #'internal' or 'terminal'
        self.operator = None
        self.variable = None
        self.leftChild = None
        self.rightChild = None
        self.formula = formula
        self.number = 0
    ...

class Formula:
    def __init__(self,name,model):
        self.name = name
        self.variables = {}
        self.operators = {}
        self.root = None #Root of the formula tree
        self.model = model
        self.variableOrdering = [] #List of variable names
        self.DD = None
    ...
```

Figure A.2: The attributes of the classes related to `Formula`.

identifier is a sequence of letters, digits or the underscore character, which are used for naming various items in the language.

Then, the declaration of operator, domain and block are defined Figure A.5, Fig-

Table A.1: The constructs in EBNF and their meanings.

::=	definition
“ ... ”	terminal symbol
(...)	grouping
... ...	concatenation
...*	repetition (any number of times)
...+	repetition (at least once)
... | ...	alternative

ure A.4 and Figure A.6. These declarations form the main body of the textual FDM.

```
Identifier ::=
    ( Alphabet | "_" ) ( Digit | Alphabet | "_" )*

Alphabet ::=
    [a-zA-Z]

Digit ::=
    [0-9]
```

Figure A.3: Definition of identifier in FDS-ML.

```
DomainDeclaration ::=
    "domain" Identifier "{" States "}" "(" Orders ")"

States ::=
    Identifier ( "," Identifier )*

Orders ::=
    Order ( "," Order )*

Order ::=
    Identifier "<" Identifier
```

Figure A.4: Definition of domain in FDS-ML

```
OperatorDeclaration ::=
    "operator" Identifier "(" Dom "," Dom ")" "return" Dom
    ( MappingClause )*
    "end"

MappingClause ::=
    StateRef "," StateRef "->" StateRef

StateRef ::=
    Identifier

Dom ::=
    Identifier
```

Figure A.5: Definition of operator in FDS-ML

A.3 Textual model of the safety instrumented system

A.4 Sensitivity results of the safety instrumented system

```
BlockDeclaration ::=
    "block" Identifier
    ( VariableClause )+
    "assertion" ( AssertionClause )+
    ObserverDeclaration
    "end"

VariableClause ::=
    Dom Identifier "(" ProbabilityDistributions ")"

ProbabilityDistributions ::=
    Distribution ( "," Distribution )*

Distribution ::=
    StateRef "=" Probability

Probability ::=
    RealNumber
    | CSVFileName

AssertionClause ::=
    VariableRef ":=" Formula

VariableRef ::=
    Identifier

Formula ::=
    VariableRef
    | OperatorRef "(" Formula "," Formula ")"

ObserverDeclaration ::=
    "observer" Identifier "=" Formula
```

Figure A.6: Definition of block in FDS-ML

```
domain SIS {W, Fs, Fdd, Fdu} (W<Fs, W<Fdd, Fdd<Fdu)

operator series(SIS, SIS) return SIS
    *, W -> *
    *, Fs -> Fs
    *, Fdd -> Fdd
    W, Fdu -> Fdu
    Fs, Fdu -> Fdd
    Fdd, Fdu -> Fdd
    Fdu, Fdu -> Fdu
end

operator parallel(SIS, SIS) return SIS
    W, * -> W
    Fs, * -> Fs
    *, Fs -> Fs
    *, W -> W
    Fdd, Fdd -> Fdd
    Fdd, Fdu -> Fdu
    Fdu, Fdd -> Fdu
    Fdu, Fdu -> Fdu
end

block TA4
    SIS S1,S2,S3 (W='Prob_Sensor_W.csv', Fs=..., ...)
    SIS LS1,LS2 (W='Prob_LogicSolver_W.csv', Fs=..., ...)
    SIS V1,V2 (W='Prob_Valve_W.csv', Fs=..., ...)
    assertion
        Channel_1 := series(series(S1,LS1),V1)
        Group_S := parallel(S2,S3)
        Group_V := parallel(V1,V2)
        Channel_2 := series(series(Group_S,LS1),Group_V)
        System := parallel(Channel_1,Channel_2)
    observer output = System
end
```

Figure A.7: The textual model of system in Figure 2.2 written in FDS-ML.

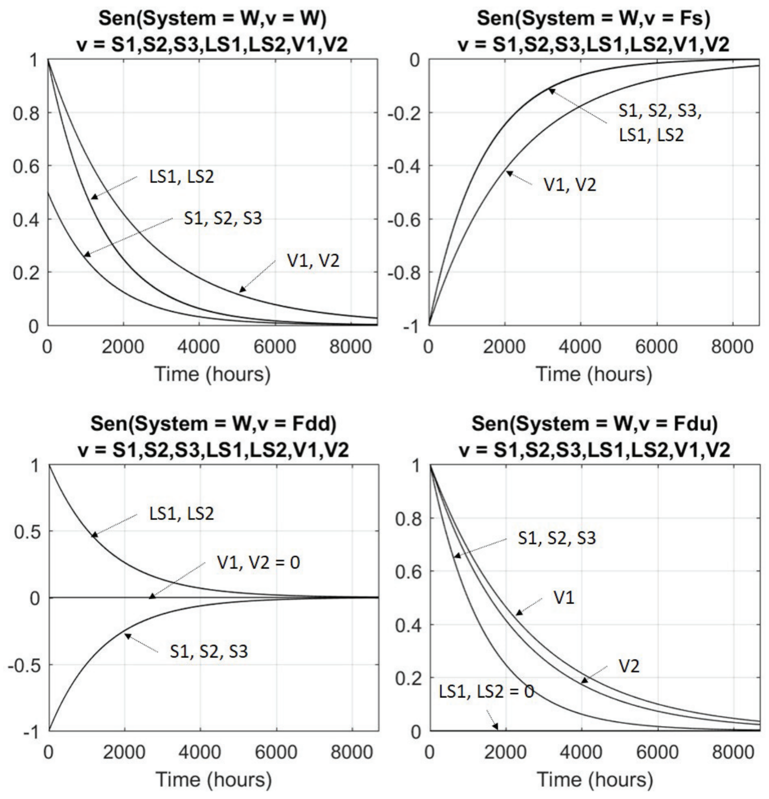


Figure A.8: Sensitivity $\mathbf{Sen}(System = W, v = c), \forall v \in \mathbf{S}, \forall c \in \mathbf{SIS}$.

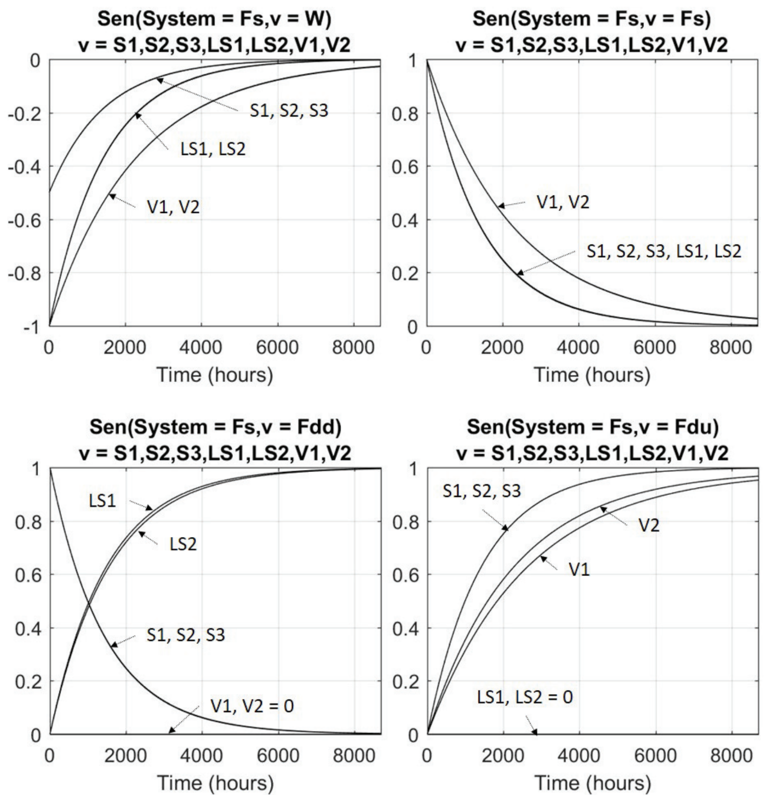


Figure A.9: Sensitivity $\mathbf{Sen}(System = F_s, v = c), \forall v \in \mathbf{S}, \forall c \in \mathbf{SIS}$.

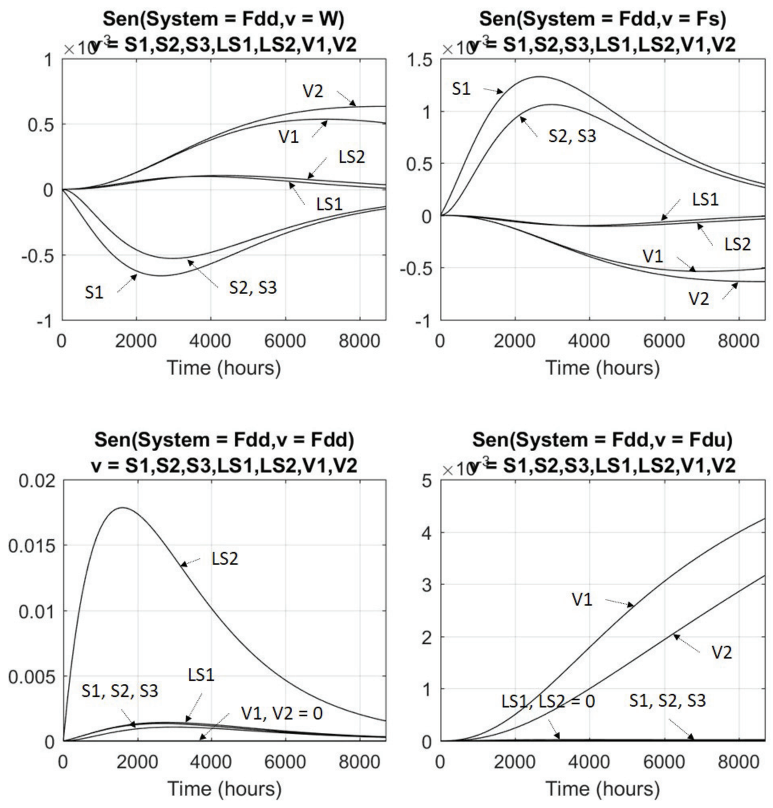


Figure A.10: Sensitivity $\mathbf{Sen}(System = F_{dd}, v = c), \forall v \in \mathbf{S}, \forall c \in \mathbf{SIS}$.

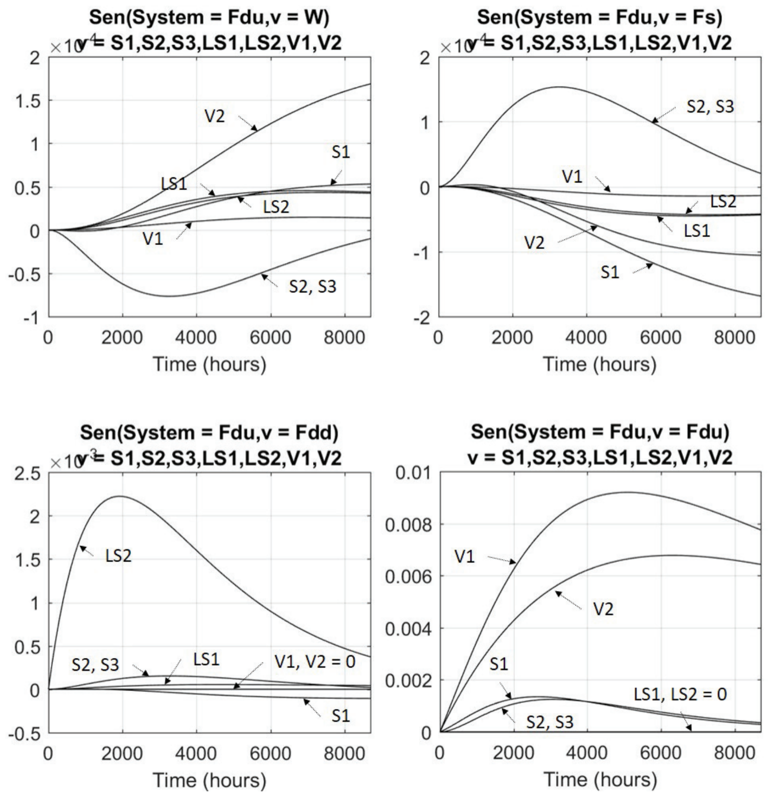


Figure A.11: Sensitivity $\mathbf{Sen}(System = F_{du}, v = c), \forall v \in \mathbf{S}, \forall c \in \mathbf{SIS}$.

Appendix B

Article on Model Synthesis

Reliability Engineering and System Safety 186 (2019) 78–87

Contents lists available at ScienceDirect

Reliability Engineering and System Safety

journal homepage: www.elsevier.com/locate/ress

Model synthesis using boolean expression diagrams

Liu Yang*, Antoine Rauzy

Department of Mechanical and Industrial Engineering, Norwegian University of Science and Technology, Trondheim 7491, Norway

ARTICLE INFO

Keywords:
Fault tree synthesis
Boolean expression diagrams
Boolean formulas
System architecture

ABSTRACT

In this article, we propose a new method for fault tree analysis, called model synthesis, which comes in addition to traditional assessment techniques. It consists in rewriting the fault tree under study, or the set of minimal cutsets extracted from this fault tree, so to make some relevant information emerge.

Our implementation of model synthesis relies on encoding Boolean formulas by means of zero-suppressed Boolean expression diagrams. Rewriting heuristics are efficiently implemented by means of local operations on these diagrams. A key feature of zero-suppressed Boolean expression diagrams is that they make it possible to perform partial normalization of Boolean formulas, avoiding in this way the exponential blow-up of calculation resources most of the other methods suffer from.

We show how to take advantage of the architecture of the systems under study to guide rewriting heuristics. We illustrate the principles of model synthesis and of its implementation by means of examples.

1. Introduction

Fault tree analysis is one of the prominent techniques for safety and reliability analyses [1–3]. It is applied in a wide range of industries [4]. Fault trees are classically assessed in two ways: by means of qualitative analyses, which consists in identifying failure scenarios via the extraction of minimal cutsets, and by means of quantitative analyses, which consists in calculating probabilistic indicators such as the top event probability, importance factors or safety integrity levels.

In this article, we propose a new assessment method, called model synthesis, which comes in addition to classical assessment techniques. It aims at making some relevant information emerge out of the fault tree under study. The idea behind model synthesis is to rewrite the original fault tree or the minimal cutsets extracted from this fault tree, into an equivalent formula that hopefully sheds a new light on the system under study. We use thus here the term model synthesis in a quite different way as several other authors who focus mainly on the automatic construction of fault trees either from high level models [5] or from reliability graphs [6,7].

Model synthesis in our sense can be useful in different contexts. First, it can be used to deal with large fault trees, like the ones involved in probabilistic safety analyses of nuclear power plants. In this context, model synthesis can help to better understand which parts of the model are influencing the results the most, i.e. eventually to determine who are the main contributors to the risk. Second, model synthesis can be applied onto fault trees that are automatically generated from higher level descriptions, as nowadays routinely done in avionic industry [8]. Automatically generated fault trees tend to be very different from those an expert would design by hand. Model synthesis can be used in this case to create more amenable models in view of certifying the system under study. Model synthesis can finally be used in any context to create synthetic view of failure scenarios. One can for instance factorize minimal cutsets according to basic events related to similar components in order to study the impact of these components on the risk as a whole [9].

Technically, synthesizing a model means rewriting a Boolean formula so to obtain a more hierarchical, compact and hopefully informative representation. The method we propose here relies on zero-suppressed Boolean expression diagrams, a variant of Boolean expression diagrams [10] to implement this rewriting process. Compared to classical binary decision diagrams [11] or zero-suppressed binary decision diagrams [12], zero-suppressed Boolean expression diagrams make it possible to implement partial rewritings. Partial normalizations are of a great interest in our context as the information we are seeking for is in general concentrated onto a small fraction of the basic events. The rewriting process can therefore be applied on these basic events only, leaving the remaining of the model unchanged. We provide mathematical justification for such partial normalizations as well as the principles of an efficient implementation.

Rewriting procedures are essentially heuristics: they may or may not give interesting results depending on where and when they are applied. We show here that it is possible to use the architecture of the system

* Corresponding author.
E-mail address: liu.yang@ntnu.no (L. Yang).

https://doi.org/10.1016/j.ress.2019.02.019
Received 16 August 2018; Received in revised form 5 January 2019; Accepted 12 February 2019
Available online 14 February 2019

under study as guideline for these heuristics. The idea is to keep related basic events close into the model. It ensures also that the fault tree stays consistent with the architecture so to make it understandable by the analyst [13].

The contribution of this article is eventually twofold. First, it proposes to complement traditional fault tree analysis with a new tool, model-synthesis. Second, it proposes an effective implementation of this new tool based on zero-suppressed Boolean expression diagrams.

The remainder of this article is organized as follows. Section 2 presents an illustrative example of what can be achieved by means of model synthesis. Section 3 introduces the zero-suppressed Boolean expression diagrams technology. Section 4 shows how implement model synthesis by means of partial operations on zero-suppressed Boolean expression diagrams. Section 5 reports the experimental results of the scalability test of the proposed method. Finally, Section 6 concludes this article.

2. Illustrative example

This section presents the case study we shall use throughout the article to illustrate ideas and techniques, namely the cooling system of a pressurized water nuclear reactor.

2.1. System description

The power of model synthesis stands primarily in its ability to make information emerges out of large models, typically such as those used to analyze the safety of cooling systems of nuclear power plants. Nevertheless, we keep here the description of the system as simple as possible, for pedagogical purposes.

Fig. 1 shows the principle of cooling systems of a pressurized water nuclear reactor. Nuclear reactions in the reactor vessel produce heat, heating the pressurized fluid in the primary coolant loop (in black on the figure). The hot primary coolant is pumped into the steam generator. Heat is transferred to a lower pressure secondary coolant loop (in dark grey on the figure) where the coolant evaporates into pressurized steam. The transfer of heat is accomplished without mixing the two fluids in order to prevent the secondary coolant from becoming radioactive. The pressurized steam is fed through a steam turbine which drives an electrical generator connected to the electric grid for transmission. After passing through the turbine the secondary coolant is cooled down and condensed in a condenser. The condenser converts the steam to a liquid so that it can be pumped back into the steam generator. The heat of the second loop is transferred to a third loop (in light grey on the figure) via the condenser. This third loop is usually made of a refrigerating tower and water which is typically pumped in a river.

We assume here that the cooling system is quadruplicate for the sake of production and safety, i.e. they are four independent circuits, each circuit consisting of the above three coolant loops. For technological reasons, these four circuits are however not fully independent as they share the same pressurizer.

To cool the nuclear reaction (and to produce electricity) the coolants must circulate into each of the three loops. This circulation is ensured by means of pumps which need to be powered. Normally, the pumps are powered by the power generated by the plant itself. In case the power of the plant does not suffice, they can be powered by the electric grid (off-site power). In case there is no off-site power, on-site diesel generators can be temporarily used to supply the required power.

2.2. Original model of the system

The fault tree for the cooling system is built classically top-down (see e.g. [1]), starting from the top-event LOCA (Loss of Coolant Accident). The cooling system is failed when all the four circuits are failed. A circuit is failed when at least one of its loops (primary, secondary and ternary) is failed. This process continues until the suitable level of decomposition. Table 1 gives the Boolean equations describing the original fault tree model we shall consider. Table 2 gives definitions of acronyms that are used in this fault tree.

Table 1
Fault tree describing the failures of the cooling system pictured Fig. 1.

$LOCA$	=	$LOCC_1 \cdot LOCC_2 \cdot LOCC_3 \cdot LOCC_4$
$LOCC_i$	=	$LOPC_i + LOSC_i + LOTC_i$
$LOPC_i$	=	$LOPP_i + FPR$
$LOSC_i$	=	$LOSP_i$
$LOTC_i$	=	$LOTP_i$
$LOPP_i$	=	$FPP_i + LOPS$
$LOSP_i$	=	$FSP_i + LOPS$
$LOTP_i$	=	$FTP_i + LOPS$
$LOPS$	=	$LOIP \cdot LOOP \cdot FDG$
$LOIP$	=	$LOPG_1 \cdot LOPG_2 \cdot LOPG_3 \cdot LOPG_4$
$LOPG_i$	=	$FCG_i + FTB_i + LOPC_i' + LOSC_i'$
$LOPC_i'$	=	$FPP_i + FPR$
$LOSC_i'$	=	FSP_i

2.3. Model synthesis

Model synthesis consists in rewriting the original model into equivalent formulas so to make information emerge. In other words, it makes it possible for the analyst to create different views on the model.

View 1: Role of the pressurizer and the power supply. Both the pressurized and the power supply have obviously a strong impact on the cooling system safety. An idea can thus be to rewrite the model (the set of equations given Table 1) so to visualize their role. Fig. 2 shows graphically the result of the factorization of the top event *LOCA* with respect to the two events *FPR* (failure of the pressurizer) and *LOPS* (loss of the power supply). This diagram can be interpreted as follows.

- If the pressurizer is failed (*FPR*), then the top event (*LOCA*) is realized.
- If the pressurizer is not failed but the power supply is lost (*LOPS*), then the top event is also realized.
- If neither the pressurizer is failed nor the power supply is lost, then the remaining scenarios to realize the top event are described by the formula $f_1 \cdot f_2 \cdot f_3 \cdot f_4$, where $f_i = FPP_i + FSP_i + FTP_i$, $1 \leq i \leq 4$.

Note that in this decomposition *FPR* is a basic event, but *LOPS* is an intermediate event. Note also that the choice and the order of these events depend on the needs of the analyst.

The model pictured Fig. 2 and the original model are equivalent. The former is indeed much more compact than the latter, but is probably not the one that the analyst would build upfront.

View 2: Role of primary pumps. The analyst may also want to study the role of primary pumps. The idea is thus to factorize the original model with respect to the *FPPi*'s events (failure of primary pump *i*, $1 \leq i \leq 4$). Fig. 3 shows a partial view of the result. It is possible to calculate an upper bound $\lceil p(\sigma) \rceil$ of the probability of a branch σ. For instance, any scenario under the upper branch of the diagram of Fig. 3, $\lceil p(FPP_1 \cdot FPP_2 \cdot FPP_3) \rceil = p(FPP_1) \times p(FPP_2) \times p(FPP_3)$. This upper bound may exceed a predefined threshold, meaning that the branch can be discarded. In our example, the probability of the simultaneous failure of three of the four primary pumps may be considered as too improbable to be reasonably considered.

The branch $FPP_1 \cdot FPP_2 \cdot \overline{FPP_3} \cdot \overline{FPP_4}$ involves only two failures of primary pumps. The analyst may want to check the formula corresponding to this branch. This formula may be however too large to be really informative. It can be nevertheless exploited, typically by extracting the basic events it involves, as shown on the figure. This is of interest in

Table 2
Definitions of acronyms.

LOCA	Loss of Coolant Accident	$LOCC_i$	Loss of Coolant Circuit *i*
$LOPC_i$	Loss of Primary Circuit *i*	$LOSC_i$	Loss of Secondary Circuit *i*
$LOTC_i$	Loss of Ternary Circuit *i*	$LOPP_i$	Loss of Primary Pump *i*
$LOSP_i$	Loss of Secondary Pump *i*	$LOTP_i$	Loss of Ternary Pump *i*
FPR	Failure of Pressurizer	FPP_i	Failure of Primary Pump *i*
FSP_i	Failure of Secondary Pump *i*	FTP_i	Failure of Ternary Pump *i*
LOPS	Loss of Power Supply	*LOIP*	Loss of Inside (on-site) Power
LOOP	Loss of Off-site Power	*FDG*	Failure of Diesel Generator
$LOPG_i$	Loss of Power Generation from circuit *i*	FCG_i	Failure of Current Generator *i*
FTB_i	Failure of Turbine *i*		

order to validate the model. In our case, the analyst can verify that no basic event issued from the first and second circuits shows up in this branch.

Note that the branches of the decision tree pictured Fig. 3 can be expanded on demand. Some branches may be unfolded further, while some other may be kept folded, typically because their exploration is made less relevant, thanks to symmetry arguments.

Table 3 summarizes the size of the original fault tree (using the number of basic and intermediate events), the size of ZBEDs (using the number of nodes) and the running time of getting the normalized ZBED. As comparison, *View 0* gives the result of a full normalization with an order of variables corresponding to a depth-first left-most traversal of the model.

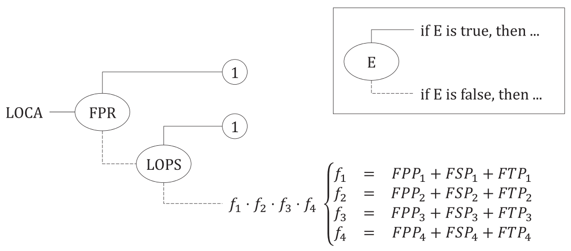

if E is true, then ...
E
if E is false, then ...
1
LOCA
FPR
1
LOPS
$f_1 \cdot f_2 \cdot f_3 \cdot f_4$
$f_1 = FPP_1 + FSP_1 + FTP_1$
$f_2 = FPP_2 + FSP_2 + FTP_2$
$f_3 = FPP_3 + FSP_3 + FTP_3$
$f_4 = FPP_4 + FSP_4 + FTP_4$

Fig. 2. View of the model obtained by factorizing events *FPR* and *LOPS*.

2.4. Discussion

In the previous section, we showed two examples of extraction of useful information from the model. The extraction process requires to rewrite the original model, typically by factorizing some of its basic and intermediate events. Several factorizations are possible, providing different views on the model.

Performing such rewritings by hand would be tedious and error prone. We shall show in the next sections that Zero-Suppressed Expression Diagrams provide a suitable algorithmic framework to implement them in a simple and efficient way.

3. Zero-suppressed boolean expression diagrams

3.1. Definition

Zero-suppressed Boolean expression diagrams result of ideas stemmed from Minato's zero-suppressed binary decision diagrams [12] and Andersen's Boolean expression diagrams [10]. A zero-suppressed Boolean expression diagram (ZBED) is a directed acyclic graph with three types of nodes:

- Constant nodes: leaves labeled with Boolean constants (1 and 0).
- Variable nodes: leaves labeled with basic events.
- Operator nodes: internal nodes with three out-edges. Such a node is denoted by $t = \langle u, v, w \rangle$, where *u* is the node pointed by the first out-edge, called the "if-edge", *v* is the node pointed by the second out-edge, called the "then-edge", and *w* is the node pointed by the third out-edge, called the "else-edge".

Each node of a ZBED is interpreted as a Boolean formula as follows.

- A constant node is interpreted by the constant it is labeled with.
- A variable node is interpreted by the basic event it is labeled with.
- An internal node $t = \langle u, v, w \rangle$ is interpreted by the formula $f{\cdot}g + h$, where *f, g* and *h* are the respective interpretations of nodes *u, v* and *w*.

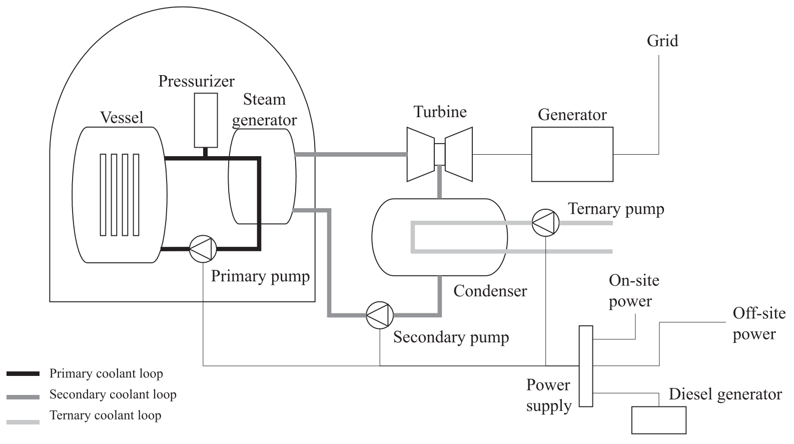


Fig. 1. A cooling system of the pressurized water nuclear reactor.

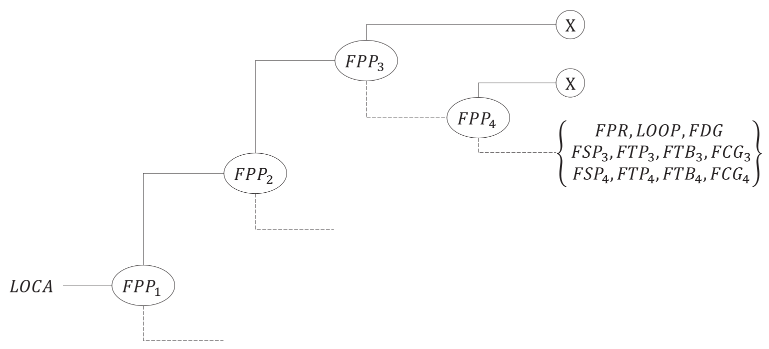


Fig. 3. View of the model obtained by factorizing events *FPPi*'s.

Table 3
Partial normalization comparing with a full normalization with an order of variables obtained from the depth-first-left-most traversal of the architecture in Fig. 16.

	# of fault tree events		# of ZBED nodes			Running time (s)	
	Basic	Intermediate	Original	Partial	Full	Partial	Full
View 0	23	39	81	–	215	–	0.657893
View 1	23	39	51	29	76	0.0024375	0.0207086
View 2	23	39	81	253	229	0.1364435	0.8062672

Table 4
Interpretation of the ZBED of Fig. 4 in terms of Boolean formulas.

Node	Definition	Interpretation	Formula
t_1	$\langle t_2, 1, t_7\rangle$	$t_2{\cdot}1 + t_7$	$t_2 + t_7$
t_2	$\langle t_3, 1, t_4\rangle$	$t_3{\cdot}1 + t_4$	$t_3 + t_4$
t_3	$\langle A, t_5, 0\rangle$	$A{\cdot}t_5 + 0$	$A \cdot t_5$
t_5	$\langle B, C, 0\rangle$	$B{\cdot}C + 0$	$B \cdot C$
t_4	$\langle A, t_6, 0\rangle$	$A{\cdot}t_6 + 0$	$A \cdot t_6$
t_6	$\langle B, D, 0\rangle$	$B{\cdot}D + 0$	$B \cdot D$
t_7	$\langle C, D, 0\rangle$	$C{\cdot}D + 0$	$C \cdot D$

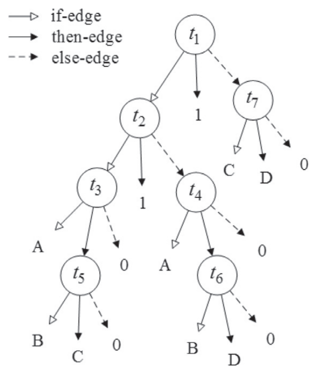


Fig. 4. An example of ZBED.

In the sequel, for the sake of the simplicity, we shall not make the distinction between nodes and their interpretation as formulas, i.e. we shall write simply $t = u{\cdot}v + w$. The set of basic events showing up in the ZBED rooted by the node t is denoted by *var*(t).

Any coherent Boolean formula can be easily encoded by means of ZBED. The binary operators "$\cdot$" (and) and "+" (or) can be written as $u{\cdot}v = \langle u, v, 0\rangle$ and $u + w = \langle u, 1, w\rangle$. Fig. 4 shows an example of the ZBED whose interpretation is given in Table 4. The formula at top level is thus written as $((A{\cdot}(B{\cdot}C)) + (A{\cdot}(B{\cdot}D))) + (C{\cdot}D)$.

$$\boxed{\begin{array}{lcl} \langle u,1,0\rangle & \rightarrow & u \\ \langle 1,v,w\rangle & \rightarrow & v \\ \langle 0,v,w\rangle & \rightarrow & w \\ \langle u,0,w\rangle & \rightarrow & w \\ \langle u,w,w\rangle & \rightarrow & w \end{array}}$$

Fig. 5. Rewriting rules used for node elimination.

3.2. Unicity of nodes

As for binary decision diagrams [11], ZBED nodes are managed in a unique table. Each time a node needs to be created, the table is looked-up to see whether a node with the same characteristics already exists. This technique is at the core of the efficiency of the decision diagrams technology.

Moreover, some simplifications are performed at node creation. Rewriting rules for eliminating equivalent nodes are given Fig. 5. The adjective "zero-suppressed" comes from the rule $\langle u, 0, w\rangle \rightarrow w$, which has been originally used by Minato for zero-suppressed binary decision diagrams [12]. In the sequel, we shall assume that these simplifications are systematically applied at node creation.

3.3. Indices

Variable nodes do not contain directly references to the variable they are labeled with. Rather, each variable is assigned an index and this index is used to label the node encoding this variable.

Both basic events and intermediate events are assigned an index. Nodes encoding intermediate events are not variable nodes, as they encode formulas. Nevertheless, as explained in the previous section, we may want to consider them as variable nodes, typically to factorize the ZBED. In this case, they are labeled with the index of the intermediate

$$
\begin{array}{rcll}
\texttt{cofactor}\ d\ E\ c & \rightarrow & d & \text{if } d \text{ is a Boolean constant} \\
\texttt{cofactor}\ E\ E\ c & \rightarrow & c & \\
\texttt{cofactor}\ F\ E\ c & \rightarrow & E & \text{if } F \text{ is a variable or a pseudo-variable node and } F \neq E \\
\texttt{cofactor}\ t\ E\ c & \rightarrow & t & \text{if } t.leastIndex > E.index \\
\texttt{cofactor}\ \langle u, v, w \rangle\ E\ c & \rightarrow & \langle u', v', w' \rangle & \text{where} \\
u' & = & \texttt{cofactor}\ u\ E\ c & \\
v' & = & \texttt{cofactor}\ v\ E\ c & \\
w' & = & \texttt{cofactor}\ w\ E\ c &
\end{array}
$$

Fig. 6. Recursive rewriting rules defining the cofactor algorithm.

$$
\begin{array}{rcll}
\texttt{factorize}\ c & \rightarrow & c & \text{if } c \text{ is a constant node} \\
\texttt{factorize}\ E & \rightarrow & E & \text{if } E \text{ is a variable or a pseudo-variable node} \\
\texttt{factorize}\ t & \rightarrow & \langle E, v, w \rangle & \text{if } t \text{ is an internal node and} \\
\multicolumn{4}{c}{E \text{ is the variable or pseudo-variable node of } t \text{ with the least index}} \\
g & = & \texttt{cofactor}\ t\ E\ 1 & \\
h & = & \texttt{cofactor}\ g\ E\ 0 & \\
v & = & \texttt{factorize}\ g & \\
w & = & \texttt{factorize}\ h &
\end{array}
$$

Fig. 7. Recursive rewriting rules defining the factorize algorithm.

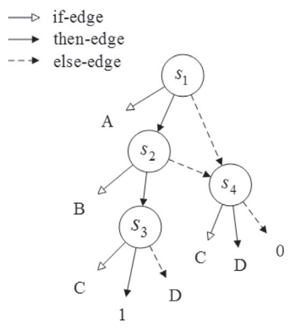


Fig. 8. Factorized version of the ZBED in Fig. 4 with $A<B<C<D$.

event they encode. In the sequel, we shall call such nodes pseudo-variable nodes.

Each node t embeds also the index $t.leastIndex$ of the least variable occurring in the ZBED rooted by t (for variable nodes and pseudo-variables nodes, $t.\ leastIndex = t.\ index$).

We shall explain the reason of this labeling principle in the next section.

3.4. Cofactors

Model synthesis relies heavily on factorization. Let f be formula and g be subformula of f. Factorizing f with respect to g consists in rewriting f as $g{\cdot}f|_{g=1}+\bar{g}{\cdot}f|_{g=0}$, where $f|_{g=c}$ denotes the formula f in which the constant c has been substituted for the subformula g. $f|_{g=0}$ and $f|_{g=1}$ are called respectively the negative cofactor and the positive cofactor of f with respect to g.

In model synthesis, we factorize formulas only with respect to variable or pseudo-variable nodes. Recursive rules defining this operation on ZBED are given Fig. 6. The fourth recursive equation makes clear the interest of the *leastIndex* field of ZBED nodes: when *t.leastIndex* $>$ *E.index*, we know for sure that the node *E* does not occur in the ZBED rooted by the node *t*. We can therefore stop here the exploration of this ZBED.

3.5. Caching

Most of operations on ZBED are defined recursively, similarly as the cofactor operation of the previous section. They take typically one or more ZBED as input and return a ZBED as output. The efficiency of

$$
\begin{array}{rcll}
\texttt{SoC}(0) & \overset{def}{=} & \emptyset & \\
\texttt{SoC}(1) & \overset{def}{=} & 1 & \\
\texttt{SoC}(E) & \overset{def}{=} & E & \text{If } E \text{ is a variable or a pseudo-variable node} \\
\texttt{SoC}(\langle E, v, w \rangle) & \overset{def}{=} & E \odot \texttt{SoC}(v) \cup \texttt{SoC}(w) &
\end{array}
$$

Fig. 9. Recursive rules to obtain SoC(t).

$$
\begin{array}{rcll}
\texttt{without}\ 0\ t & \rightarrow & 0 & \\
\texttt{without}\ 1\ t & \rightarrow & 1 & \text{if } t \neq 1 \\
\texttt{without}\ s\ 1 & \rightarrow & 0 & \\
\texttt{without}\ s\ 0 & \rightarrow & s & \\
\texttt{without}\ t\ t & \rightarrow & 0 & \\
\texttt{without}\ s\ t & \rightarrow & s & \text{if } s.leastIndex > t.index \\
\texttt{without}\ \langle E, s, t\rangle\ \langle E, v, w\rangle & \rightarrow & \langle E, s', t'\rangle & \text{where} \\
s'' & = & \texttt{without}\ s\ w & \\
s' & = & \texttt{without}\ s''\ v & \\
t' & = & \texttt{without}\ t\ w & \\
\texttt{without}\ \langle E, s, t\rangle\ \langle F, v, w\rangle & \rightarrow & \langle E, s', t'\rangle & \text{if } E.index < F.index \text{ and} \\
s' & = & \texttt{without}\ s\ \langle F, v, w\rangle & \\
t' & = & \texttt{without}\ t\ \langle F, v, w\rangle & \\
\texttt{without}\ \langle E, s, t\rangle\ \langle F, v, w\rangle & \rightarrow & \texttt{without}\ \langle E, s, t\rangle\ w & \text{if } F.index < E.index
\end{array}
$$

Fig. 10. Recursive rewriting rules defining the without algorithm.

$$
\begin{array}{rcll}
\texttt{minimize}\ c & \rightarrow & c & \text{if } c \text{ is a constant node} \\
\texttt{minimize}\ E & \rightarrow & E & \text{if } E \text{ is a variable or a pseudo-variable node} \\
\texttt{minimize}\ \langle E, s, t\rangle & \rightarrow & \langle E, v, w\rangle & \text{where} \\
u & = & \texttt{minimize}\ s & \\
w & = & \texttt{minimize}\ t & \\
v & = & \texttt{without}\ u\ w &
\end{array}
$$

Fig. 11. Recursive rewriting rules defining the minimize algorithm.

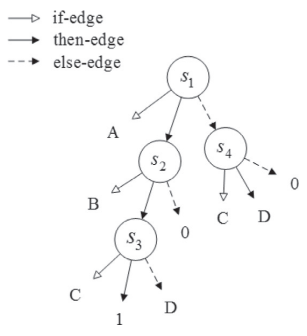


Fig. 12. The minimized ZBED of Fig. 8.

these operations can be significantly improved by using caching: each time the operation *op* must be performed on parameters $p_1,\ldots p_k$, the cache is looked up. If it contains an entry for *op* and $p_1,\ldots p_k$, the result is immediately returned. Otherwise, the operation is actually performed, then cached.

It is again Bryant &. al [11] who introduced this idea in the binary decision diagram technology.

In the sequel, we shall assume that all algorithms use such caching technique.

3.6. Factorization and ordering

It is possible to calculate a normal form for ZBED by factorizing them recursively with respect to basic events. A ZBED is factorized if either it is reduced a constant or a variable node, or it is rooted by a node ⟨*E, v, w*⟩ where *E* is a variable node and *v* and *w* are ZBED in which *E* does not show up.

Let ≺ be a total order over variables. A factorized ZBED is ordered with respect to ≺ if any of its node ⟨*E, v, w*⟩, any variable *F* showing up in ZBED in either *v* or *w* verifies *E*≺*F*.

The recursive rules defining the algorithm that rewrite a ZBED into an equivalent factorized and ordered one are given Fig. 7. To be more efficient, a node is not replaced by a new one, but rather transformed into a factorized one. In this way, all the nodes pointing to the node get the factorized version at no additional cost. This principle has been introduced for dynamic reordering of binary decision diagrams [14].

Fig. 8 shows the factorized version of the ZBED pictured Fig. 4 according to the ordering *A*≺*B*≺*C*≺*D*. The fully factorized ZBED ordered according to the order ≺ over the variables (basic events) of a formula *f* is isomorphic to the reduced ordered (according to ≺) binary decision diagram encoding *f*. The two diagrams differ however completely in the way they are obtained. The binary decision diagram associated with a formula *f* is built by composing the binary decision diagrams associated with the sub-formulas of *f*. The spaces (data structures) of formulas and binary decision diagrams are thus separated. With ZBED, both formulas

```
normalize $c$  $\rightarrow$  $c$             if $c$ is a constant node
normalize $E$  $\rightarrow$  $E$             if $E$ is a variable or a pseudo-variable node
normalize $t$  $\rightarrow$  $\langle E, v, w\rangle$   if $t$ is an internal node and
        $E$ is the variable or pseudo-variable node of $t$ with the least index
        $g$ $=$ cofactor $t$ $E$ $1$
        $h$ $=$ cofactor $t$ $E$ $0$
        $u$ $=$ normalize $g$
        $w$ $=$ normalize $h$
        $v$ $=$ without $u$ $w$
```

Fig. 13. Recursive rewriting rules defining the normalize algorithm.

```
p-normalize $c$ $q$  $\rightarrow$  $c$             if $c$ is a constant node
p-normalize $E$ $q$  $\rightarrow$  $0$             if $E$ is a variable or a pseudo-variable node and $q \times p(E) < \tau$
p-normalize $E$ $q$  $\rightarrow$  $E$             if $E$ is a variable or a pseudo-variable node and $q \times p(E) \geq \tau$
p-normalize $t$ $q$  $\rightarrow$  $t$             if $t.leastIndex \geq i_{max}$
p-normalize $t$ $q$  $\rightarrow$  $w$             if $t$ is an internal node and
        $E \in \mathcal{S}$ is the variable or pseudo-variable node of $t$ with the least index and $q \times p(E) < \tau$
        $h$ $=$ cofactor $t$ $E$ $0$
        $w$ $=$ p-normalize $h$ $q$
p-normalize $t$ $q$  $\rightarrow$  $\langle E, v, w\rangle$   if $t$ is an internal node and
        $E \in \mathcal{S}$ is the variable or pseudo-variable node of $t$ with the least index and $q \times p(E) \geq \tau$
        $g$ $=$ cofactor $t$ $E$ $1$
        $h$ $=$ cofactor $t$ $E$ $0$
        $u$ $=$ p-normalize $g$ $q \times p(E)$
        $w$ $=$ p-normalize $h$ $q$
        $v$ $=$ without $u$ $w$
```

Fig. 14. Recursive rewriting rules defining the p − normalize algorithm.

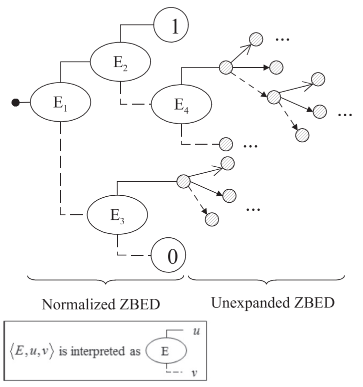


Fig. 15. Example of partially normalized ZBED.

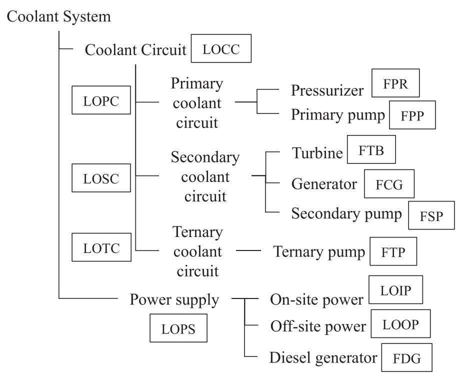


Fig. 16. An architecture of the coolant system in Fig. 1.

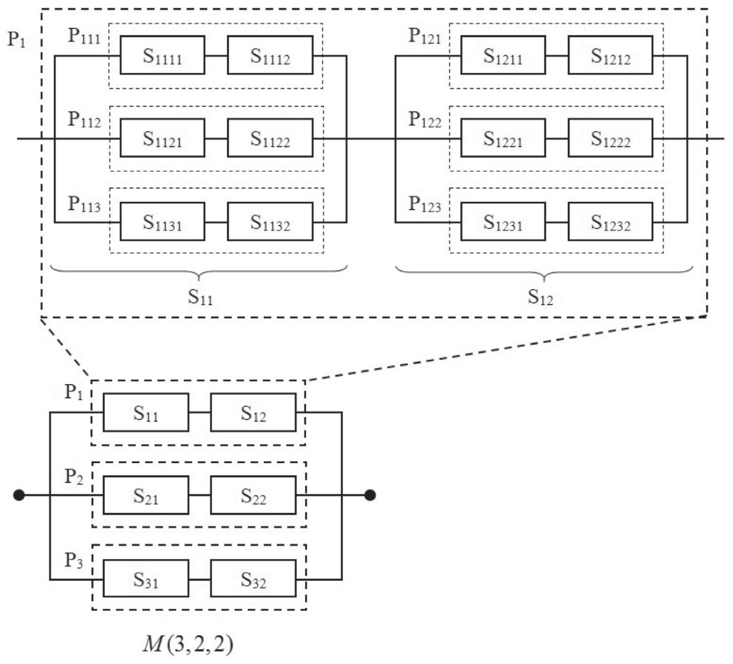


Fig. 17. Parametric model $M(3, 2, 2)$.

Table 5
Fault tree describing the failures of $M(3, 2, 2)$ in Fig. 17.

$TOP = P_1 + P_2 + P_3$	
$P_i = S_{i1} \cdot S_{i2}$	(i = 1,2,3)
$S_{ij} = P_{ij1} + P_{ij2} + P_{ij3}$	(j = 1,2)
$P_{ijk} = S_{ijk1} \cdot S_{ijk2}$	(k = 1,2,3)
$S_{ijkl} = C_{ijkl} + SS_i$	(l = 1,2)

and their normal forms are encoded within the same space.

3.7. Minimization

In general, once factorized, a ZBED becomes more compact. It is however often possible to make it even more compact by removing non-minimal branches. To explain this process, we shall first show how to interpret a ZBED as a set of cutsets, or equivalently as a sum-of-products.

Let E be a basic event and s be a set of cutsets. We define the product $E \odot s$ as follows:

$$E \odot s \overset{def}{=} \{E.\ \pi;\ \pi \in s\}$$

Let t be a factorized, ordered ZBED. $\mathrm{SoC}(t)$ denotes the set of cutsets interpretation of a t. Recursive rules to build $\mathrm{SoC}(t)$ are given Fig. 9. Consider again the ZBED pictured in Fig. 8. We have $\mathrm{SoP}(s1) = \{A.\ B.\ C, A.\ B.\ D, A.\ C.\ D, A.\ D\}$.

Note that $\mathrm{SoP}(s1)$ contains the non-minimal cutset $A.C.D$.

The method to minimize, i.e to remove non-minimal cutsets from a factorized ordered ZBED $\langle E, v, w\rangle$ works into two steps. First, it minimizes v into $v_{\min}$ and w into $w_{\min}$. Second, it removes from $v_{\min}$ all the cutsets π such that there exists a cutset ρ in $\mathrm{SoP}(w)$ such that $\rho \subseteq \pi$. This second operation is performed by the without algorithm introduced for zero-suppressed binary decision diagrams by one of the author in [15] and further improved in [16].

Figs. 10 and 11 give respectively recursive rewriting rules defining without and minimize algorithms adapted to ZBED. The minimized version of the ZBED pictured Fig. 8 is pictured Fig. 12. Note that the above minimization principle is similar to the one propose by Jung [17] to calculate minimal cutsets using zero-suppressed binary decision diagrams.

Table 6
Full normalization of $M(p, s, d)$ with order of variables corresponding to a depth-first left-most traversal of the model.

M			# of fault tree events		# of ZBED nodes		Running time (s)
p	*s*	*d*	Basic	Intermediate	Original	Fully normalized	
2	2	1	6	7	15	15	0.0005875
2	3	1	8	9	21	20	0.001016
2	4	1	10	11	27	25	0.0016374
3	2	1	9	10	22	29	0.0027617
3	3	1	12	13	31	39	0.0048888
3	4	1	15	16	40	49	0.005376
4	2	1	12	13	29	55	0.0059631
4	3	1	16	17	41	74	0.0257535
4	4	1	20	21	53	93	0.0238569
5	5	1	30	31	81	213	0.0929677
6	6	1	42	43	115	479	0.7946444
7	7	1	56	57	155	1067	8.5812105
8	8	1	72	73	201	2361	140.4506352
2	2	2	18	31	51	43	0.0105937
2	3	2	38	57	111	91	0.0275659
2	4	2	66	91	195	159	0.0564192
2	5	2	102	133	303	247	0.1878167
3	2	2	39	64	112	126	0.0375149
3	3	2	84	121	247	273	0.3376026
3	4	2	147	196	436	480	2.1421809
4	2	2	68	109	197	317	1.0885322
2	2	3	66	127	195	159	0.0922675
2	3	3	218	345	651	529	0.8691808

Table 7
Partial normalization (factorization of SS_i) comparing with a full normalization compatible with the factorization.

M			# of fault tree events		# of ZBED nodes			Running time (s)	
p	*s*	*d*	Basic	Intermediate	Original	Partial	Full	Partial	Full
2	2	1	6	7	15	14	15	0.0003807	0.0005609
2	3	1	8	9	21	18	20	0.0014846	0.0010145
3	3	1	12	13	31	31	39	0.001792	0.0035515
3	4	1	15	16	40	37	49	0.0027814	0.0315186
3	5	1	18	19	49	43	59	0.0034137	0.0202382
4	5	1	24	25	65	68	112	0.009577	0.0249844
5	5	1	30	31	81	109	213	0.0198436	0.0790558
6	6	1	42	43	115	194	479	0.0564447	0.778016
7	7	1	56	57	155	347	1067	0.1532794	8.763043
8	8	1	72	73	201	632	2361	0.463898	145.5475206
2	3	2	38	57	111	78	91	0.0050992	0.0220251
2	4	2	66	91	195	134	159	0.0099643	0.0960328
2	5	2	102	133	303	206	247	0.0183755	0.130448
3	2	2	39	64	112	85	126	0.0107254	0.0507543
3	3	2	84	121	247	175	273	0.0760447	0.3209596
3	4	2	147	196	436	301	480	0.0449491	1.7469204
4	2	2	68	109	197	156	317	0.033493	1.0738509
2	2	3	66	127	195	134	159	0.0148028	0.0839745
2	3	3	218	345	651	438	529	0.043567	0.8689405
2	4	3	514	731	1539	1030	–	0.1329544	–
3	2	3	219	388	652	445	–	0.0730657	–
3	3	3	732	1093	2191	1471	–	0.3639312	–
3	4	3	1731	2356	5188	3469	–	1.193553	–
3	5	3	3378	4339	10,129	6763	–	3.7578617	–
4	2	3	516	877	1541	1052	–	0.3506644	–
4	3	3	1732	2513	5189	3484	–	2.1515793	––
4	4	3	4100	5461	12,293	8220	–	9.837303	–
4	5	3	8004	10,105	24,005	16,028	–	31.5285081	–

3.8. Normalization

The normalization of a ZBED performs simultaneously both factorization and minimization, which is more efficient than applying one operation after the other. The recursive rewriting rules defining the normalization algorithm are given Fig. 13. The fully normalized ZBED ordered according to the order $\prec$ over the variables (basic events) of a formula *f* is isomorphic to the reduced ordered (according to $\prec$) zero-suppressed binary decision diagram encoding the minimal cutsets of *f*. Here again, the two diagrams differ in the way they are obtained.

4. Partial normalizations

The advantage of the ZBED technology over the (zero-suppressed) binary decision diagram technology stands in the ability to perform partial operations, including partial factorizations, minimizations and normalizations, thanks to the encoding of both formulas and their normal forms within the same data structures.

4.1. Algorithm

A ZBED can be seen as (a compact encoding of) a decision tree. As discussed Section 2, it is often sufficient to develop only partially such decision tree to get relevant information. More exactly, the partial development of a decision tree involves: first, a subset $\mathcal{S}$ of basic and intermediate events of interest, together with an order over these events; and second, a probability threshold τ under which branches can be discarded. $\mathcal{W}$ is called the care set. It contains the events on which is decomposition of the ZBED will be performed.

We shall thus modify algorithms presented in the previous section so to perform partial normalizations. This works as follows.

The first step consists in giving indices $1, 2, \ldots, i_{\max}$ to events of $\mathcal{S}$, including intermediate events, according to the order in which we want them to show up in the decision tree. The remaining basic events are given indices $i_{\max} + 1$, $i_{\max} + 2$, …. The ZBED is re-labeled according to these indices. This operation is linear in the size of the ZBED.

The second step consists in modifying the normalize so to take into account the set $\mathcal{S}$ and the threshold τ. Recursive rewriting rules defining the $\mathrm{p}-\mathrm{normalize}$ algorithm are given Fig. 14. The algorithm is initially called with its second parameter set to 1.0. In the last two rules, if the pivot variable *E* is pseudo-variable, *p*(*E*) is set to 1.0. It is possible to take the actual probability of *E* (or an approximation of this probability) only if *E* is a module, i.e. it shares no variable with the rest of the model.

A general representation of the partially normalized ZBED is given Fig. 15. In the normalized part of this ZBED is essentially similar to decision trees (or event trees) of Section 2.

4.2. Choice of the care set

In model synthesis, we suggest to use the system architecture as a guideline to achieve informative rewritings. As exemplified in Fig. 16, an architecture can be seen as a functional or physical decomposition of the system, in which events (marked in rectangle) can be assigned to their relevant functions or components. Based on such assignment, traversal algorithms (like the depth-first-left-most algorithm) can be implemented to order and group events automatically in the architecture.

Guiding by the system architecture, the ZBED can be rewritten in a way that maps the architecture, which means to create similarities in their way of decomposition between the ZBED and the architecture. With the help of this mapping, we provide a possibility to check the consistency between the fault tree model and the system design.

5. Experiments

In this section, we provide the scalability test of the proposed model synthesis method. The model used to perform the test is a multilevel parallel-series system, denoted by *M*(*p, s, d*), which is characterized by three parameters:

- p: the number of parts in parallel at each layer
- s: the number of parts in series at each layer
- d: the depth of alternation

Fig. 17 gives an example of $M(3, 2, 2)$. The last level of the hierarchy is made of a series of two components: a local independent component C and a support system SS dedicated to the first parallel line. For instance, the unit S_{ijkl} ($i = 1, 2, 3$; $j = 1, 2$; $k = 1, 2, 3$; $l = 1, 2$) in the last level of the model $M(3, 2, 2)$ is comprised by the series of C_{ijkl} and SS_i, where SS_i is dedicated to P_i.

The fault tree describing the failures of $M(3, 2, 2)$ is exemplified in Table 5.

Two kinds of experiments are implemented:

- A full normalization of the model with an order of variables corresponding to a depth-first left-most traversal of the model, of which the results are given in Table 6.
- A partial normalization (factorization of SS_i) comparing with a full normalization compatible with the factorization, of which the results are given in Table 7. For the last nine cases in Table 7, the full normalization is not proceeded since the running time is more than 2000 seconds. It also shows that partial normalizations are far more efficient when dealing with large models.

6. Conclusion

In this article, we propose a new method, called model synthesis, which consists in rewriting the fault tree under study so to make some relevant information emerge. The rewriting relies on an encoding of Boolean formulas by means of zero-suppressed Boolean expression diagrams. A key feature of zero-suppressed Boolean expression diagrams is that they make it possible to perform partial normalization of Boolean formulas. To get relevant information, it is often sufficient to develop only partially the diagram, which is also more efficient than a full factorization. For partial normalization, analyst can choose a care set guided by the system architecture and a probability threshold τ to decide under which branches can be discarded. As future work, we plan to extend this method to non-coherent systems and to support other quantitative analyses like the calculation of importance measures [18].

References

[1] Kumamoto H, Henley EJ. Probabilistic risk assessment and management for engineers and scientists. Piscataway, N.J., USA: IEEE Press; 1996. 978–0780360174.
[2] Andrews JD, Moss RT. Reliability and risk assessment. (2nd ed.) Materials Park, Ohio 44073-0002, USA: ASM International; 2002. 978–0791801833.
[3] Rausand M, Arnljot H, et al. System reliability theory: models, statistical methods, and applications. 396. John Wiley & Sons; 2004. https://doi.org/10.1002/9780470316900.
[4] Ruijters E, Stoelinga M. Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. ComputSciRev 2015;15:29–62. https://doi.org/10.1016/j.cosrev.2015.03.001.
[5] Xiang J, Yanoo K, Maeno Y, Tadano K. Automatic synthesis of static fault trees from system models. IEEE Publishing978-1-4577-0780-3; 2011. p. 127–36. https://doi.org/10.1109/SSIRI.2011.32.
[6] Camarda P, Corsi F, Trentadue A. An efficient simple algorithm for fault tree automatic synthesis from the reliability graph. Reliab IEEE Trans 1978;R-27(3):215–21. https://doi.org/10.1109/TR.1978.5220330.
[7] Elliott M. Computer-assisted fault-tree construction using a knowledge-based approach. Reliab IEEE Trans 1994;43(1):112–20. https://doi.org/10.1109/24.285124.
[8] Prosvirnova T, Rauzy A. Automated generation of minimal cutsets from altarica 3.0 models. Int J Crit Comput-Based Syst 2015;6(1):50–79. https://doi.org/10.1504/IJCCBS.2015.068852.
[9] Leblond A. Synthèse de coupes minimales fonctionnelles en coupes minimales composant. Actes du 19ième congrés Lambda-Mu. Dijon, France: Institut pour la Maîtrise des Risques; 2014.
[10] Andersen HR, Hulgaard H. Boolean expression diagrams. Information and computation 2002;179(2):194–212. https://doi.org/10.1006/inco.2001.2948.
[11] Brace KS, Rudell RL, Bryant RS. Efficient implementation of a BDD package. Proceedings of the 27th ACM/IEEE design automation conference. Orlando, Florida, USA: IEEE0-89791-363-9; 1990. p. 40–5. https://doi.org/10.1145/123186.123222.
[12] Minato S-I. Zero-suppressed BDDs for set manipulation in combinatorial problems. Proceedings of the 30th ACM/IEEE design automation conference, DAC'93. Dallas, Texas, USA: IEEE0-89791-577-1; 1993. p. 272–7. https://doi.org/10.1145/157485.164890.
[13] Getir S, Van Hoorn A, Grunske L, Tichy M. Co-evolution of software architecture and fault tree models: an explorative case study on a pick and place factory automation system. 1074. CEUR-WS; 2013. p. 32–9.
[14] Rudell RL. Dynamic variable ordering for ordered binary decision diagrams. In: Lightner M, Jess JAG, editors. Proceedings of IEEE International Conference on Computer Aided Design, ICCAD'93. Santa Clara, CA, USA: IEEE0-8186-4490-7; 1993. p. 42–7.
[15] Rauzy A. New algorithms for fault trees analysis. Reliab Eng Syst Saf 1993;05(59):203–11. https://doi.org/10.1016/0951-8320(93)90060-C.
[16] Rauzy A. Mathematical foundations of minimal cutsets. IEEE Trans Reliab 2001;50(4):389–96. https://doi.org/10.1109/24.983400.
[17] Jung WS, Han SH, Ha J. A fast bdd algorithm for large coherent fault trees analysis. Reliab Eng Syst Saf 2004;83(3):369–74. https://doi.org/10.1016/j.ress.2003.10.009.
[18] Aliee H, Borgonovo E, Glaß M, Teich J. On the boolean extension of the birnbaum importance to non-coherent systems. Reliab Eng Syst Saf 2017;160. http://search.proquest.com/docview/1944562495/?pq-origsite=primo