Reliable Systems

Fault Tree Analysis via Markov Reward Automata

Dennis Guck

Reliable Systems

Fault Tree Analysis via Markov Reward Automata

Dennis Guck

Composition of the Graduation Committee:

Chairman:	Prof.dr. P.M.G. Apers
Supervisor:	Prof.dr.ir. J.-P. Katoen, PDEng
Co-supervisor:	Dr. M.I.A. Stoelinga

Members:

Prof.dr.ir. B.R.H.M. Haverkort	University of Twente
Prof.dr.ir. L.A.M. van Dongen	University of Twente
Prof.dr. R.D. van der Mei	VU University Amsterdam
Prof.dr. A.K.I. Remke	University of Münster
Dr.ir. R.J.I. Basten	Eindhoven University of Technology
Dr. D. Giannakopoulou	NASA Ames Research Center
Dr. J. Romijn	Movares Netherlands

CTIT

CTIT Ph.D. Thesis Series No. 16-419

Centre for Telematics and Information Technology
University of Twente, The Netherlands
P.O. Box 217 – 7500 AE Enschede, The Netherlands

IPA Dissertation Series No. 2017-03

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

Stichting voor de Technische Wetenschappen

The work in this thesis was supported by the ArRangeer project (smARt Railroad maintenance eNGinEERing), funded by the STW-ProRail partnership program Explo-Rail under the project grant 122238.

ISBN: 978-90-365-4291-3

ISSN: 1381-3617 (CTIT Ph.D. Thesis Series No. 16-419)

DOI: 10.3990/1.9789036542913

<https://doi.org/10.3990/1.9789036542913>

Typeset with L^AT_EX. Printed by Ipskamp Drukkers.

Cover design by Dennis Guck.

Original image by Jack Moreh, available under a non-commercial license at <http://www.stockvault.net/photo/215073/fast-growing-business-with-rocket-launch>.

Copyright © 2017 Dennis Guck, Enschede, The Netherlands.

RELIABLE SYSTEMS

FAULT TREE ANALYSIS VIA MARKOV REWARD AUTOMATA

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus,
prof.dr. T.T.M. Palstra,
on account of the decision of the graduation committee,
to be publicly defended
on Thursday 23rd of March 2017 at 16:45

by

Dennis Guck

born on 22nd of May 1987
in Haan, Germany

This dissertation has been approved by:

Prof.dr.ir. J.-P. Katoen, PDEng (supervisor)

Dr. M.I.A. Stoelinga (co-supervisor)

Acknowledgements

WRITING this dissertation was the last step of a four year PhD journey. I would like to thank my supervisors Joost-Pieter Katoen and Mariëlle Stoelinga for the opportunity to start and finish this journey. Looking back to the start of my PhD, now about four and a half years ago, I learned a lot about research, computer science and collected a lot of memorable experiences.

Joost-Pieter, thank you for awakening my interest in formal methods during my studies in Aachen. Further, it was always insightful to have a discussion with you, and even you were not directly around the corner, you always found time for them. Be it while I visited Aachen, you were in Twente, during your sabbatical in Saarbrücken, or even when you were travelling. These discussions helped me a lot to form ideas and find a direction to focus my research on.

Marëlle, as my daily supervisor you were the go-to person for all my questions. Thank you for all the time and thoughtful discussions and advice throughout the years. You were always eager to taught me how to convey my research such that others will be able to find it interesting as well. Your quest for improving my writing, be it for presenting a clean theory or providing the proper motivation about my work, helped me a lot in writing papers and in the end this thesis. Also thank you for the nice personal chats during the years and the organisation of regular outings.

I really enjoyed my time at the Formal Methods & Tools (FMT) group. When I started at FMT, I knew that I was on the correct corridor when hearing Marks laughter. It was always a pleasure to work together with you and discuss ideas about Markov automata. Waheed, you were my longest office companion, thank you for the nice talks and also the invitations to experience delicious Pakistani food as well as your wedding. Enno, while working on the same project, thank you for always be available for discussions. Freark, thank you for the work on DFTCALC and all the discussions we had. Sebastian, who visited the FMT group for several month, thank you for the ongoing discussion over DFT semantics. Retrospectively, I've shared an office with a lot of people over the years, starting with Florian and Gerjan, and later on Waheed and Mark, extended by Enno, Marcus, Rajesh, Bugra, as well as temporarily with Freark, Sebastian, David and more. While the exact office constellation changed throughout the years between all those (and some more), I like to thank all of you for the time spent together. In total, the FMT group provided an excellent atmosphere for doing research, but also provided support, fun activities and friendships. I'm thankful to all the group members for being available to have

passionate discussions about science as well as the cosy chats and coffee breaks. Be it on a regular day or at the BOCOM on Friday evenings.

There were a lot of memorable experiences throughout the years at FMT like the outings with the whole group, including among others a trip to Schiermonnikoog or sailing on the IJsselmeer. Besides, there were a lot of different social events organised throughout the years. A great experience was to participate in the Inter-Actief Rially with Arend, Leslie and Freark. Also a recurring event over the years was the (former) floor five film festivals (FFFFF) organised by Arend, where we tried to watch as many movies as possible w.r.t. an always changing theme. Further, I really enjoyed the social dinners initiated by Tri and later on taken over by Tom. We discovered a lot of restaurants and always had a nice evening together. Besides, Bugra's initiative to participate in Pub-quizzes lead to many fun evenings, also if we never managed to come close to a top position. A highlight was also the yearly participation in the Batavierenrace. Thanks to Marina and Stefano who were always eager to motivate people to participate and initiated the Fast Moving Team.

One of the merits during my PhD journey were also the opportunities to participate in summer schools and conferences. The summer schools were a great way to meet a lot of other PhD students and to swap ideas as well as experiences, while also learning more about different fields of computer science. Further, while visiting conferences, I was able to present my work to people from around the world and have interesting discussions with them. Besides that, conference visits brought me the opportunity to explore different parts of the world. Visiting the Iguazu waterfalls on the Brazil/Argentina border with Joost-Pieter and Mariëlle before the QEST conference. Discovering Rome's nightlife and karaoke bars with Florian and Mark after the ETAPS conference. Taking a trip up the Australian east coast and diving at the great barrier reef with Christian after the ATVA conference. These and many more experiences, which I'm grateful for, were made possible throughout my years as PhD student.

In my final year I had the opportunity to spent time at the NASA Ames research center as part of an internship. I like to thank Dimitra and Johann for the great experience and willingness to work together. It was a pleasure to work with you, as well as to experience your hospitality. I also like to thank Freark for providing me lodging while I was in the US and entrusted me his BMW to drive around California. The road trip together to Las Vegas or the Yosemite falls were just some of the highlights during my stay.

I also like to thank all the people I met during the ROCKS meetings. Moreover, I like to thank all the people by ProRail, Movares and NedTrain that interacted with me over the years and made it possible to conduct my research and case studies. Further I like to thank all the committee members for approving my thesis and providing me with helpful comments.

Finally, I like to thank my family and friends for their understanding, support and encouragement throughout the years. Thank you for worrying about my progress, discussing problems, or just having a good time together.

TODAY'S society is characterised by the ubiquitousness of hardware and software systems on which we rely on day in, day out. They reach from transportation systems like cars, trains and planes over medical devices at a hospital to nuclear power plants. Moreover, we can observe a trend of automation and data exchange in today's society and economy, including among others the integration of *cyber-physical systems*, *internet of things*, and *cloud computing*. All these systems have one common denominator: they have to operate *safe* and *reliable*. But how can we trust that they operate safe and reliable?

Model checking is a technique to check if a system fulfils a given requirement. To check if the requirements hold, a model of the system has to be created, while the requirements are stated in terms of some logic formula w.r.t. the model. Then, the model and formula are given to a *model checker*, which checks if the formula holds on the model. If this is the case the model checker provides a positive answer, otherwise a counterexample is provided. Note that model checking can be used to verify hardware as well as software systems and has been successfully applied to a wide range of different applications like aerospace systems, or biological systems.

Reliability engineering is a well-established field with the purpose of developing methods and tools to ensure reliability, availability, maintainability and safety (RAMS) of complex systems, as well as to support engineers during the development, production, and maintenance to maintain these characteristics. However, with the advancements and ubiquitousness of new hardware and software systems in our daily life, also methods and tools for reliability engineering have to be adapted.

This thesis contributes to the realm of model checking as well as reliability engineering. On the one hand we introduce a *reward extension* to *Markov automata* and present algorithms for different reward properties. On the other hand we extend *fault trees* with *maintenance* procedures.

In the first half of the thesis, we introduce *Markov reward automata* (MRAs), supporting non-deterministic choices, discrete as well as continuous probability distributions and timed as well as instantaneous rewards. Moreover we introduce algorithms for reachability objectives for MRAs. In particular we define expected reward objectives for goal and time bounded rewards as well as for long-run average rewards.

In the second half of the thesis we introduce *fault maintenance trees* (FMTs). They extend *dynamic fault trees* (DFTs) with corrective and preventive main-

tenance models. The advantage of FMTs is that the maintenance strategies are directly defined on the level of the fault tree. Therefore the effect of maintenance is directly translated into the analysis and enables us to take a step towards finding smarter maintenance procedures.

In the end we introduce a tool-chain implementing our approach. Moreover we perform an industrial case study evaluating the capabilities of FMTs for modelling and analysing a realistic scenario. In particular we focus on a RAMS analysis for a railway trajectory in the Netherlands by investigating different corrective as well as preventive maintenance strategies.

Contents

1	Introduction	1
1.1	Reliability engineering	2
1.2	Problem statement	3
1.3	Fault trees	4
1.4	Maintenance	7
1.5	Model checking	8
1.5.1	Quantitative analysis	9
1.5.2	Markov models	10
1.6	Main contributions	12
1.7	Outline of the thesis	14
1.7.1	Thesis roadmap	14
I	Markov models	17
2	Markov reward automata	19
2.1	Markov automata	20
2.2	Behavioural notions of MAs	24
2.2.1	Structural properties of MAs	26
2.2.2	Open and closed behaviour	29
2.2.3	Subsumption of Markov models	30
2.3	Markov reward automata	31
2.4	The behaviour of MRAs	32
2.4.1	Paths	32
2.4.2	Traces	34
2.5	Schedulers	35
2.6	Parallel composition	37
2.6.1	Uniformisation	41
2.7	Bisimulations	41
2.7.1	Strong bisimulation	42
2.7.2	Weak bisimulation	42
2.8	Conclusion	48

3	Analysis of expected reward properties	49
3.1	Quantitative analysis	50
3.1.1	Prepossessing	51
3.2	Cumulative rewards	53
3.2.1	Goal-bounded reward	53
3.2.2	Time-bounded reward	58
3.3	Long run rewards	59
3.3.1	Step 2: Unichain MRA	60
3.3.2	Step 3: Arbitrary MRAs	66
3.4	Model checking of expected rewards	68
3.4.1	Continuous stochastic reward logic	68
3.5	Experiments	70
3.6	Related work	72
3.7	Conclusion	74
4	Markov models in the real world:	
	Airborne collision avoidance system	75
4.1	The ACAS X system	76
4.1.1	Inside ACAS X	77
4.2	The ACAS X model	78
4.3	Model Conformance	82
4.3.1	Conformance framework set up	83
4.3.2	Conformance relations	84
4.4	Analysing conformance issues	85
4.4.1	A non-conformance encounter	86
4.4.2	Step-wise conformance relations	88
4.5	Automatic Generation of Non-Conformance Encounters	89
4.5.1	The scenario generation environment	90
4.5.2	The reward function	90
4.5.3	Analysis of generated non-conformance encounters	93
4.6	Related work	94
4.7	Conclusion	96
II	Fault maintenance trees	97
5	Fault trees: The basics	99
5.1	Static fault trees	101
5.2	Dynamic fault trees	103
5.2.1	BEs in dynamic fault trees	104
5.2.2	Gates in dynamic fault trees	105
5.3	Formal definitions	108
5.3.1	Well-formedness	111
5.4	Key performance indicators for dependability	112
5.5	DFT analysis	116
5.5.1	Input output extension of MAs	116
5.5.2	Smart state space generation	117

5.6	Semantics for DFTs	119
5.6.1	BEs and static gates	120
5.6.2	Dynamic gates	122
5.6.3	Spare activation	125
5.7	Related work	128
5.7.1	DFT extensions	129
5.8	Conclusion	132
6	Fault maintenance trees	135
6.1	Maintenance	136
6.1.1	Preventive and corrective maintenance	137
6.1.2	Maintenance in fault trees	138
6.2	Maintainable components	138
6.2.1	Degradation	140
6.2.2	Maintenance signals	143
6.3	Inspections and repairs	146
6.3.1	Inspections	146
6.3.2	Repairs	148
6.4	Fault maintenance trees	150
6.4.1	Repair communication	151
6.5	Smart semantics	158
6.5.1	Aggregation of components	158
6.5.2	Effects on the semantics	159
6.6	Related work	163
6.7	Conclusion	164
7	Fault maintenance trees in practice	165
7.1	The tool architecture of DFTCALC	166
7.1.1	Architecture	167
7.1.2	Integrated tools	167
7.1.3	DFTCALC web interface	168
7.1.4	DFTCALC's program flow	170
7.1.5	FMT extension of DFTCALC	174
7.1.6	Alternative FMT specification	175
7.1.7	Use cases for the two toolchains	178
7.2	Analysis of classical DFTs	178
7.2.1	Classical DFT models	179
7.2.2	Results	180
7.3	RAMS in railways	185
7.3.1	Railway case study	186
7.3.2	Effects of corrective maintenance	189
7.3.3	Effects of preventive maintenance	190
7.4	Scalability of FMTs	192
7.4.1	Experimental set up	192
7.4.2	Scalability results	193
7.5	Conclusion	194

8 Conclusion	195
8.1 Summary	195
8.2 Discussion and Future work	197
Bibliography	199
List of publications by the author	199
References	204

CHAPTER 1

Introduction

*We can only see a short distance ahead, but we can see plenty there
that needs to be done.*

ALAN TURING

OUR daily life is characterised by the ubiquitousness of hardware and software systems. We rely on these systems day in, day out. Consider a visit to a physician for a health checkup. After the anamnesis and physical examination by the physician, further diagnostics follow. Most of these diagnostics rely on medical devices, and ranging from relatively simple ones like a blood pressure monitor to complex and even possibly harmful devices like an X-ray machine. Their correct functionality depends not only on the hardware but also on the software implementation. Therefore, for a correct and safe diagnostic with medical devices, the hardware in combination with the software implementation should be highly reliable, otherwise unforeseen accidents can happen. A classical example is the Therac-25 radiation therapy machine. A race condition in the control software of the Therac-25 led to accidents. As a result, six patients between 1985 and 1987 were given a massive overdose of radiation [LT93]. Hence if a problem in a medical device is not recognised early enough, it can have harmful impacts. According to the medical device recall report of 2003 to 2012 by the Food and Drug Administration, the most frequent recalls are related to device design, software, and non-conforming material or component issues [Foo14]. While these recalls prevent the distribution of possibly harmful medical devices, there is a chance that a device was already used despite the possible danger. To avoid such events, it is pivotal to be able to identify safety risks at the design phase.

Another driving factor in our today's society and economy is the trend of automation and data exchange, and is captured by the so-called fourth industrial revolution "Industry 4.0". It includes among others cyber-physical systems, internet of things, and cloud computing. All these will lead to "smart factories" with an interoperability between machines, devices, sensors, and people [LFKFH14]. Thus with the "Industry 4.0" current manufacturing technologies are transitioning more and more to automation. Therefore, mechanisation and automation in the work process will increase, reaching from assistance and support of humans with information and visualisations to the physical support of

humans by cyber-physical systems. These advancements are also carried over to different aspects of our infrastructure. Due to the increasing consumption of electricity as well as the demand for renewable resources, the load on power grids is changing. Therefore power grids have to be adapted with a variety of operational and energy measures, leading to smart grids [Ras10].

Other recent advancements in our daily life are for example consumer drones which can fly by themselves, or the emerge of self-driving cars. The acceptance of such automation heavily relies on the reliability of the system itself. Consider a disruption of service in a self-driving car. This would not just be an annoyance for the user but a safety risk for the user as well as bystanders. The same holds for a self-flying drone. From the consequential fact of the inter connectivity of hardware and software throughout the majority of our society and economy, new challenges arise in building reliable and secure systems. Thus, with the advancements and ubiquitousness of new hardware and software systems in our daily life, the methodologies to assure their reliability and safety become more important than ever.

1.1 Reliability engineering

Engineering a system can have many positive benefits, at the same time it most surely also includes risks. Especially for large or complex systems the question if one can rely on them is not straightforward to answer. *How can we trust that a power plant is reliable? How can we be certain that a train ride is safe? Can we rely on safety measures like airbags in our cars during an emergency?* Those are essential questions, and despite the number experts involved in the designing and engineering process of such systems, there is always a chance of failure. Farmer writes in the editorial to the first issue of the “Reliability Engineering & System Safety” journal [Far80] the following:

“Safety depends on reliability. This is a lesson learned through experience. The major growth of industrial technology went forward on learning gained through accidents. More has been learned from the failure of bridges, dams, turbines, etc., than from those that have not failed but we cannot afford to continue this pattern.”

Since failures do not only carry the risk of financial losses, but also the probability of environmental damages and casualties, there is an interest in ways to ensure reliability of systems a priori. Moreover, with the constant change in modern industry and increasing complexity of systems, methods and tools also have to be adapted. Thus there is a demand to apply advancements in academic research to practical use by which means the reliability of systems can be ensured.

Reliability engineering is a well-established field and has the purpose of de-

veloping methods and tools to ensure *reliability*, *availability*, *maintainability* and *safety* (RAMS) of complex systems, as well as to support engineers during the development, production, and maintenance to maintain these characteristics [KP14]. In the context of complex systems the term *dependability* is used to describe attributes like reliability, availability, maintainability, safety as well as security and survivability [ALR+01]. While the original definition of dependability is defined as the ability to provide a service that can justifiably be trusted, an alternate definition provides the criterion of deciding if a service is dependable [ALRL04]. That means, the dependability of a system is defined over the ability of avoiding failures that are more frequent and severe than acceptable. Considering the RAMS attributes, they can be assessed to determine the overall dependability of a system by using *quantitative* as well as *qualitative* measures. Following the definition of [ALRL04] the RAMS attributes can be described as follows:

- **Reliability.** The continuity of a correct service.
- **Availability.** The readiness of a correct service.
- **Maintainability.** The ability to be subject to modifications and repair.
- **Safety.** The absence of catastrophic events for the user and environment.

1.2 Problem statement

In this thesis we consider an important domain in the infrastructure: *railway systems*. They are pivotal in urban life, providing means of efficient transportation of freight as well as people while being environment friendly. However, railways require efficient maintenance to keep them reliable while ProRail, the company responsible for the Dutch railway infrastructure, has high ambitions w.r.t. increasing the availability and reliability of the Dutch railways. To do so they established together with STW a research program under the name “Explorail” with the aim of finding innovative solutions to prepare the railway infrastructure in the Netherlands for the future [SP11]. The program is split into two main themes, “*Whole system performance*” and “*Intelligent rail*”. The theme of the whole system performance is to enhance the performance of the railway system by innovative insights, new technologies and forms of cooperation. The theme of the intelligent rail is to exploit advanced maintenance concepts to determine how, when and where maintenance needs to be done to obtain sustainable rails. A crucial issue with maintenance is that it is a driving cost factor. Thus with a bad maintenance strategy the upkeep of the rail infrastructures can increase and therefore also affect e.g. the price of a train ticket for the general public. Therefore it is of utmost importance to find ways to conduct maintenance in a safe and reliable manner while still being economical. The main challenge for an *intelligent rail* is therefore determined by an *optimisation problem* regarding:

- maintenance benefits on the reliability and availability, and
- maintenance costs for the individual tasks like inspections and repairs.

The project “ArRangeer”, short for “smARt RAILroad maintenance eNGinEERING with stochastic model checking”, is part of the intelligent rail theme of the Explorail program and the initiator of this thesis. With the help of the ArRangeer project a step towards intelligent railway maintenance is taken by developing innovative concepts for tackling the maintenance optimisation problem. An overall challenge to resolve this problem can be formulated as follows.

Challenge 1. How to determine smart and cost effective maintenance procedures that enhance the reliability and availability of the railway infrastructure?

We take on this challenge by extending and combining methodologies from *reliability engineering* and *model checking* as illustrated in Figure 1.1.

- We extend the analysis model with costs/rewards such that it is possible to factor in different costs, e.g. for the maintenance. This is achieved through the introduction of *Markov reward automata* (MRAs) and expected reward algorithms.
- To take a step towards finding smarter maintenance procedures, a more integrated model is needed that combines risk analysis together with maintenance planning. We introduce *fault maintenance trees* (FMTs), an intuitive model for reliability engineers to describe a system’s failure behaviour and maintenance strategy.
- We combine both concepts into an analysis framework. This framework provides the key ingredients that can be deployed to solve maintenance optimisation problems. We use these ingredients to provide a tool that focuses on the benefits of maintenance w.r.t. reliability and availability.

In the next sections, we describe the main ingredients of the thesis, including fault trees, maintenance, and model checking.

1.3 Fault trees

Fault trees (FTs) are a wide-spread model within RAMS analysis and are used in the analysis of safety-critical systems. FTs were developed at Bell laboratories in the 1960s to evaluate the reliability of a complex missile launch system [EI99], the *Minuteman-I*. In the following years to the introduction of FTs, Boeing recognised their potential as a significant safety system analysis tool [Hix68]. Besides, at the System Safety Symposium of 1965 the first technical paper on *fault tree analysis* (FTA) was presented [Mea65]. By the 1970s FTA became more widespread and was adopted by other organisations, in particular by the nuclear power plant industry. Moreover, FTA has been enforced by several authorities, including the US Nuclear Regulatory Commission [VGRH81], the Federal Aviation Administration (FAA) [Fed00] as well as the National Aeronautics and Space Administration (NASA) [Sta+02]. Besides, FTA is standardised by the International Electrotechnical Commission [IEC61025]. Thus since the introduction of FTA, the method became wide-spread among organisations that have to deal with reliable systems like FAA, NASA, ESA, Airbus,

Figure 1.1: Building blocks of our approach.

Honeywell, etc. Further, FTs are also an active research field where recent advances include for example the automatic generation of FTs from a specification language [JVB07] or the synthesis of failure rates [VJK16].

An FT is used to describe the potential causes of a system failure. In particular, FTA follows a *top-down* approach by considering a system failure as top level event, which is refined into its originating causes down to the components. Therefore, the FT describes how component failures propagate throughout a system leading to the system failure. A complementary method to FTA is the *failure mode and effects analysis* (FMEA), which considers a *bottom-up* approach [LLL13; IEC60812]. Thus, it focuses on analysing the effects of individual component failures on the system. Historically, FMEA was one of the first systematical techniques for dependability analysis. The U.S. military standard MIL-STD-1629 introduced FMEA, which was standardized in 1974 and later updated by the standard MIL-STD-1629A [MIL-STD-1629A]. FMEA, and its extension with criticality FMECA, is still very popular in industry. The analysis offers a structured way to list possible failures together with their consequences, as well as possible countermeasures. In fact, since FME(C)A helps to determine component failures, creating an FME(C)A table is often the first step when constructing an FT [Sta+02].

The basic building blocks of an FT are the *root node*, *gates*, and *basic events*. The root node describes an undesired state of a system, gates capture how lower level events are affecting each other, and basic events are the lowest level and describe component failures. FTA is a top-down analysis method, tailored to determine and reduce potential risks that can lead to a system failure. Thus, FTA can help to understand how a system fails as well as provide metrics like

Figure 1.2: Example FT of a barrier failure.

the unreliability of a system. Use cases for FTA are among others:

- Help to understand what events lead to an undesired system state. This helps to understand the overall failure behaviour of a system.
- Assist in designing a system and avoid unnecessary risks. Hence, by laying out all potential failure causes with a FT, design flaws become more visible.
- Show compliance w.r.t. safety and reliability requirements. Therefore, by assigning failure distributions to components, the FT can be subject to quantitative analysis.
- Optimise resources by determining potential failure groups. For example, for critical components more redundancy can be added.
- Prioritise maintenance by determining high risk failures. Thus, by identifying high risk components, maintenance strategies can be adapted.

Example 1.1. Figure 1.2 depicts a small example of a FT for a barrier failure including some dynamic behaviour. The barrier is powered by a motor to go up and down, and if the motor fails the barrier cannot work anymore. As a backup, there is a second motor which can be switched on via a switching unit if the main motor fails. Thus the barrier will fail if the main motor and the spare motor fail, or if the switching unit fails first and then the main motor fails. The gate “Switching unit” represents the ordered failure propagation for the switch and motor, by only failing if the connected components fail from left to right.

As Example 1.1 shows, it is important to have the ability to describe dynamic behaviour in the FT to enable a more accurate description of the system’s failure behaviour. A well-established extension of FTs with such dynamic behaviour are *dynamic fault trees* (DFTs) [DBB92]. They support the modelling of priorities, spare management, as well as functional dependencies.

1.4 Maintenance

To ensure that systems stay reliable over time, it is crucial to conduct proper maintenance. Activities are required that will preserve the condition of components and therefore preserve the situation or state of the system. However, maintenance can also interfere with a system's operation and for example force a temporary unavailability and therefore induce extra costs for the stakeholder. To avoid such unwanted circumstances one has to consider multiple factors into the maintenance planning. The process of scheduling maintenance actions for a system can grow to a complex problem that involves a variety of different metrics.

While FTs describe a system's failure in terms of its component's failure behaviour, they do not directly regard the impact of maintenance on the components. While there exist FTs supporting repairs [RFIV04; BCR04], they cannot model more complex maintenance strategies. For example, a maintenance strategy could include the renewal of a component after a certain time which resets its failure distribution. When taking such behaviour into account while constructing and analysing the FT, a more accurate estimation about a system failure can be given. Further, maintenance strategies could be analysed with respect to their impact on the reliability within FTA. Thus, by including maintenance into FTA the class of properties that can be analysed is enriched, benefiting the significance of the FTA results.

Challenge 2. *How can advanced maintenance strategies be integrated into FTA to improve the analysis?*

The failure of a system is strongly related to its maintenance. For example, if a car is inspected regularly and the proper maintenance actions are performed, like changing the oil, the probability that a defect occurs between the inspection intervals is relatively low. However, if the inspection intervals are ignored and no maintenance at all is carried out, it is more likely that a failure will occur. Another factor that has an influence on the system's failure is the quality of the maintenance. For example, if the wrong type of maintenance is performed, instead of preventing a failure, more failures could appear. Considering the car maintenance, if wrong oil is used during the oil change, the car will probably undergo a failure before a new inspection is due. To be able to integrate the correlations between maintenance and the failure behaviour of the system, we will extend FTs with maintenance models and introduce *fault maintenance trees*. In particular, FTs will be extended with two kinds of maintenance procedures: *corrective* and *preventive* maintenance. While corrective maintenance basically changes a component when it is broken, e.g. repairs it, preventive maintenance inspects and changes components while they still work but have a degraded performance. For example, the oil change in a car would be a preventive maintenance procedure, while the change of a defective spark plug is a corrective maintenance procedure.

Figure 1.3: Overview of the model checking approach.

1.5 Model checking

The basic idea of *model checking* is to check if a system fulfils a given requirement. Figure 1.3 depicts an overview of the basic building blocks of the model checking approach. The starting point is a system and its requirements. To check if the requirements hold, a model of the system has to be created, while the requirements are stated in terms of some logic formula w.r.t. the model. Then, the model and formula are given to a *model checker*, which checks if the formula holds on the model. If this is the case the model checker provides a positive answer, otherwise a counterexample is provided. Note that model checking can be used to verify hardware as well as software systems. Moreover, it has been successfully applied to a wide range of different applications like aerospace systems [Boz+09], or biological systems [KNP08]. An extensive introduction to the principles of model checking is provided by Katoen and Baier [KB08].

The field of model checking was introduced in independent work by Clarke and Emerson [CE82] and by Queille and Sifakis [QS82]. One can say that they essentially discovered the idea of model checking at the same time, while the term “model checking” originates from [CE82]. Therefore, Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis obtained the ACM Turing award in 2007 [CES09]. A likely argument why the idea of model checking was due at that time is given by Clarke and Emerson in [CE82]:

“The task of proof construction can be quite tedious, and a good deal of ingenuity may be required. We believe that this task may be unnecessary in the case of finite state concurrent systems, and can be replaced by a mechanical check that the system meets a specification expressed in a propositional temporal logic.”

The problem statement, where also the term model checking originates from, is as follows:

Given a model M and a (temporal) formula f , determine whether M is a model of the formula f .

The model M is a finite-state model of the system in question, and the formula f is given by a (temporal) logic specifying a property that should hold on the system. To verify its validity an extensive search through the state space is con-

ducted. If f holds for M the model checker passes, otherwise a counterexample is generated.

1.5.1 Quantitative analysis

While *qualitative properties* in model checking give a clear “yes” or “no” answer, this is not suitable in all cases. Suppose we want to verify the throughput of a smart grid system. In this case we do not expect a simple “yes” or “no” answer if there is throughput, but rather a quantitative measure describing the system’s throughput. Including *quantitative properties* in model checking allows to check against a variety of performance and dependability measures. Especially w.r.t reliability engineering, quantitative measures are of utmost importance. Typical questions to answer are: *What is the failure probability of the system over a duration of 10 years? What is the expected time until a first failure in the system? What is the availability of the system in the long run?*

Further, the design of reliable systems involves many trade-offs: *Is the level of redundancy high enough to be available over 99% of the time? Is it cost effective to use multiple servers to increase availability and performance of a cloud service? What is the percentage change of availability if we reduce the battery size? How can maintenance be scheduled such that the operational costs are minimised?* Such optimisation questions not only need additional quantitative metrics to be answered, but are also subject to the following attributes:

- (1) (stochastic) timing to model speed or delay;
- (2) discrete probabilities to model random phenomena;
- (3) non-determinism to model choices;
- (4) rewards or costs to measure the quality of solutions.

Let’s consider the different applications for (1) - (4) while modelling a reliable system like the barrier failure, depicted as FT in Figure 1.2. The failure behaviour of the components, i.e. the switch and both motors, are dependent on their usage time. This can be represented using stochastic timing, such that with the ageing of the component the failure probability increases with an exponential rate. Recall that if the first motor fails, the switch will be used to activate the second motor. However, there could be a chance that this activation is not working. Hence, to represent this random phenomena, one would insert a probability distribution. Hence, with probability p the activation is successful, whereas with probability $1 - p$ the activation fails. Now consider the scenario that the switch and motor are affected by a common cause failure, such that either the switch is first deactivated and then the motor, or the motor is first deactivated and then the switch. Where the first combination will lead to a failure, in the second combination the switch can still activate the second motor. Since the failure order is unknown it would be represented by a non-deterministic choice. Moreover, to describe the impact of the individual and overall failure of the system one would assign costs to the failure events.

	discrete timing	stochastic timing
deterministic	DTMC	CTMC
non-deterministic	PA	IMC

Table 1.1: Markov models distinguished by timing and non-determinism.

1.5.2 Markov models

Markov models are a prominent model used in quantitative model checking that have support for (1) - (4). Moreover, they have the property that the future state only depends on the current state (Markov property). For example, consider to roll a six-sided dice. When rolling the dice, the chance to roll a four is one out of six. Now if the dice is rolled again, the outcome of the previous throw does not influence the chance of the current one. Thus, the chance to roll again a four is also one over six. Generally, this property allows to conduct analyses which would be otherwise intractable. Note that there exists a plethora of different Markov models that can be used to specify quantitative behaviour of a system. The most common distinctions between these Markov models are in their support of discrete and continuous timing as well as deterministic and non-deterministic behaviour.

Timing. In *probabilistic* models time is measured in discrete entities. Thus time is represented by a sequence of discrete steps where each step represents a time progression, usually identified with a natural number. On the contrary, *stochastic* models incorporate continuous timing. A step in a stochastic model is delayed by a random amount of time governed by a continuous probability distribution. Therefore, transitions are labelled by a positive real number representing the rate of a negative exponential distribution.

Non-determinism. The behaviour of a *deterministic* model is completely specified by its probability distribution. In contrast, the behaviour of a non-deterministic model is not fully specified by its probability distribution. Thus non-determinism specifies the uncertainty of a system. Hence, at some point the precise behaviour is unknown, however different outcomes can be specified.

Table 1.1 lists one example of a Markov model for each combination of timing and non-determinism: discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs), probabilistic automata (PAs), and interactive Markov chains (IMCs). DTMCs and PAs are modelling discrete probabilistic behaviours and CTMCs and IMCs are modelling stochastic probabilistic behaviour. Further, PAs and IMCs also support non-deterministic choices. Hence, they all have their own domain. However, in this thesis we want our model to be as general as possible, and thus to be able to cater to all these domains. Therefore, we focus on Markov automata (MA). They were introduced by Eisentraut et al. [EHZ10a] as a conservative extension of Segala's probabilistic automata

Figure 1.4: Example of an Markov automata.

(PAs) [Seg95] and Hermanns' interactive Markov chains (IMCs) [Her02]. Thus, they combine discrete and continuous probability distributions, as well as allow non-deterministic choices. Hence, a transition in an MA is either labelled with a positive real number representing the rate of a negative exponential distribution, or an action leading into a discrete probability distribution.

Example 1.2. Consider the MA depicted in Figure 1.4. It models a component degradation with either a repair or replacement. The component can be operational, i.e. it is in the *up* or *degraded* state, or the component is not working, i.e. it is in its *failed* state. The initial condition of the component is that it is operational and in its prime condition. Therefore, the initial state is the *up* state. While the component is up and running it can degrade with rate λ_1 or fail with rate λ_2 . If the component is degraded then it will fail with rate λ_3 . In case the component has failed, it can either be replaced or repaired. If the component is replaced, it will be up and running again. In the case the component is repaired, it is fully functional and up with probability p or it is running but degraded with probability $1 - p$.

Despite the simplicity of the component modelled in Example 1.2, it needs continuous probability distributions to model its degradation, discrete probability distributions to model the random repair behaviour as well as a non-deterministic choice for the uncertainty between choosing the repair and replacement. Since this model exhibits non-determinism, one can ask the question: *Is it better to replace or repair the component after a failure in the long run?* Therefore one can for instance compare the time spent in the *up* state as a distinguishing factor.

Besides timing and non-determinism, costs and rewards are important ingredients for many types of systems, modelling critical aspects like energy consumption, task completion, or repair costs. Considering Example 1.2, a question could be: *Is it more profitable to replace or repair a component?* To solve this kind of question, a notion of costs is needed in the model. This leads to the following challenge.

Challenge 3. *How can MAs be extended with costs and rewards and how can they be analysed against reward objectives?*

We approach this question by defining a *generic reward structure* on MAs and introducing *Markov reward automata* (MRAs). The reward structure allows

to assign *instantaneous rewards* to each transition as well as *timed rewards* to each state. Note that an instantaneous transition reward is associated to the action that is taken as well as the successor state. The timed reward, on the other hand, is assigned directly to the state.

Example 1.3. Consider the MA in Figure 1.4. Now we want to assign costs for the different maintenance actions. Since for the replace action only one transition exists, we assign only a cost to that transition. However, for the repair action we not only assign a cost for the repair action, but specifically assign a cost to the individual transitions. Thus, the transition going from the failed state to the degraded state can be regarded as less cost efficient and is assigned a high cost, while the transition going from the failed to the up state is assigned a lower cost. Moreover, one can assign rewards to the up and degraded state, representing the accumulation of revenue over time as long as the component is running.

Besides, to be able to argue over rewards in MRAs, we introduce algorithms for *expected reward objectives*. This enables us to give answers to questions like: *What are the expected costs to be operational again?* or *What is the long-run reward of the system?* For example, for the component we could ask what are the maximum and minimum long-run costs w.r.t. the repair and replacement.

1.6 Main contributions

Throughout the chapter we have posed different challenges w.r.t. the overall problem (see Challenge 1), fault tree analysis (see Challenge 2), as well as Markov models (see Challenge 3). If we reflect on these challenges, we can state our research objective in the following way:

Develop a framework that allows analysing system failures under different maintenance strategies. Further develop a general model that can be used to analyse timed and reward based properties.

Reliability engineering. The first part of the statement refers back to reliability engineering including RAMS analysis. The first question we need to answer is:

Question 1. *What is a suitable base model to analyse a system's failure behaviour?*

Since fault trees (FTs) are a wide-spread model in industry to perform reliability analysis w.r.t. a system's failure behaviour, we use this model as our basis. However, the expressiveness of FTs is not suitable for more complex models that have some dynamic behaviour. Therefore we decide to use dynamic fault trees (DFTs), an extension of FTs with priority failures, spare management and functional dependencies (see Chapter 5). This leads to the next question:

Question 2. *How can we integrate maintenance with DFTs and what is their semantics?*

Since we want to have an integrated model, including failure behaviour and maintenance procedures, we extend the DFT formalism with *repair* and *inspection* models and introduce fault maintenance trees (FMTs). While there exist different formalism's that extend DFTs with repairs, we allow more advanced maintenance behaviour, including inspection cycles and different repair strategies. Moreover, we provide a semantics that allows to introduce new behaviour with ease (see Chapter 6). However, this new model leaves us with another question:

Question 3. *What is the effect of FMTs on the reliability analysis and can they be applied in railway engineering?*

To answer this question we introduce a prototypical tool based on our FMT semantics. Besides, we provide benchmarks that demonstrate the tool's capabilities. Moreover, to demonstrate the usefulness of FMTs and our approach, we perform a real case study in the realm of railways (see Chapter 7).

Model checking. The second statement refers back to model checking and Markov models. The first question to answer is:

Question 4. *What is a suitable and generic Markov model with rewards?*

The recently introduced Markov automata (MA) provide a very generic Markov model, supporting non-deterministic choices, discrete probability distributions as well as continuous probability distributions. Therefore, we decided to extend this model with rewards and introduce *Markov reward automata* (MRA). This enriches the modelling with transition as well as state rewards (see Chapter 2). However, this leads to the next question:

Question 5. *How can we analyse MRAs w.r.t. reward objectives?*

As for MAs, we explore reachability objectives for MRAs, however in the context of rewards. Therefore we define expected reward objectives for goal and time bounded rewards as well as for long-run average rewards. Moreover, we provide algorithms to solve these and show their scalability with the help of some literature case study (see Chapter 3).

Besides, we explore the problem of validating Markov models w.r.t. real systems. The question is:

Question 6. *How can we determine how adequately a Markov model represents a real system?*

To answer this question, we conduct an extensive case study for the next airborne collision avoidance system. As a first step, we identify important conformance criteria between the model and the system and show how to analyse those. As a second step we provide a way to automatically generate scenarios to check the conformance criteria between the model and system (see Chapter 4).

1.7 Outline of the thesis

The outline of the thesis focuses on Questions 4, 5, and 6 in the first part, where the second part considers Questions 1, 2, and 3. An overview of how the thesis is structured is given in Figure 1.5. The remainder of the thesis is as follows:

- **Chapter 2** formally introduces Markov models. We first define Markov automata with their behavioural and structural properties, then we formally define Markov reward automata. Moreover, we discuss parallel composition as well as summarise different bisimulation relations w.r.t. the reward extension.
- **Chapter 3** presents algorithms for the analysis of expected reward properties of Markov reward automata. In particular, expected goal-bounded rewards as well as long-run average rewards. Moreover we discuss how these algorithms can be integrated in model checking.
- **Chapter 4** discusses the problem of model validation by means of the next airborne collision avoidance system. This includes a discussion of conformance relations as well as the topic of generation of test cases.
- **Chapter 5** introduces static and dynamic fault trees. We define their semantics in terms of Markov automata. Further, we introduce key performance indicators and their analysis.
- **Chapter 6** extends dynamic fault trees with maintenance and introduces fault maintenance trees. Therefore, we introduce new maintenance modules. Moreover we provide the semantics of fault maintenance trees in terms of Markov automata.
- **Chapter 7** discusses how to include fault maintenance trees in a prototypical implementation. Moreover, we conduct case studies and show the applicability of our approach.
- **Chapter 8** concludes the thesis by providing a discussion about the advantages and disadvantages of Markov reward automata and fault maintenance trees as well as provides directions for future research.

The origins of the chapters are given in their respective introduction.

1.7.1 Thesis roadmap

The thesis is meant to be read sequentially, however, variations are possible. A roadmap of the connections between the chapters is given in Figure 1.5. In total, the thesis is divided into two parts:

- (1) Markov models containing Chapters 2 to 4 and
- (2) Fault maintenance trees containing Chapters 5 to 7.

The first part focuses on Markov models, especially Markov (reward) automata in Chapters 2 and 3. Besides, Chapter 4 can be viewed as a spin-off which focuses on the relation between a Markov model and its real world application. The second part focuses on fault (maintenance) trees. Note that the chapters in the second part use concepts from the first part. In particular, Chapters 5 and 6 rely on definitions provided in Chapters 2 and 3.

Figure 1.5: Thesis roadmap.

Part I

Markov models

Markov reward automata

IN this chapter we introduce *Markov reward automata* (MRAs), a model that combines (a) stochastic timing, (b) discrete probabilities, (c) non-deterministic choices, and (d) rewards for states and transitions. MRAs are obtained by defining a new reward structure to the formalism of Markov automata (MAs) [EHZ10a]. We support two types of rewards: (1) *State rewards* modelling the reward per time unit while residing in a state, and (2) *transition rewards* which are obtained directly when taking a transition. Such reward extensions have been shown to be valuable in the past for less expressive models. For instance, rewards for DTMCs and CTMCs have lead to the implementation of the *markov reward model checker* (MRMC) [KZHHJ11] supporting among others model checking reward-based properties over DTMCs [AHK03] and CTMCs [HCHKB02] with rewards. Besides, with the MRA model we provide a natural combination of the EMPA [Ber97] and PEPA [Cla96] reward formalism.

By generalising MAs with rewards, MRAs provide a compositional formalism for concurrent real time systems. In fact, they inherit the MA application domain, ranging from the standardised *architecture analysis and design language* (AADL) [Int04] over *globally asynchronous locally synchronous* (GALS) hardware design [CHLS09] to *dynamic fault trees* (DFTs) [BCS10]. Moreover, MRAs are expressive enough to provide a natural semantics for generalised stochastic Petri nets (GSPNs) [MBCDF94]. Note that the traditional GSPN semantics yields a continuous time Markov chain (CTMC), i.e. an MRA without discrete probabilities and non-determinism. However, this semantics is restricted to *confusion free* GSPNs, i.e. excluding non-determinism. Traditionally, *confused* GSPNs are considered ambiguous and left out from any kind of analysis. Nevertheless, several semantics for higher level formalisms like AADL map onto GSPNs without ensuring that the GSPN is confusion free, and therefore possibly include confused models. Thus, by adapting the GSPN semantics to MAs, also those confused models can be represented. In fact Eisentraut et al. [EHKZ13] show that MAs, and therefore MRAs, are a natural semantics for *every* GSPN.

In this chapter we start with a general introduction to MAs, including standard notations and definitions that are used throughout the thesis. Afterwards we introduce MRAs and define their behaviour over paths and traces. For the resolution of non-deterministic choices we present a class of measurable schedulers. Moreover, we define parallel composition for MRAs as well as discuss the

lifting of bisimulation relations from MAs to MRAs.

Origins of the chapter. This chapter introduces MAs as presented in

- D. Guck, H. Hatefi, H. Hermanns, J.-P. Katoen, and M. Timmer. “Modelling, Reduction and Analysis of Markov Automata”. In: *Proceedings of the 10th International Conference on Quantitative Evaluation of Systems (QEST 2013)*. Vol. 8054. Lecture Notes in Computer Science. Springer Verlag

and MRAs based on

- D. Guck, M. Timmer, H. Hatefi, E. J. J. Ruijters, and M. I. A. Stoelinga. “Modelling and analysis of Markov reward automata”. In: *Proceedings of the 12th International Symposium on Automated Technology for Verification and Analysis (ATVA 2014), Sydney, NSW, Australia*. Vol. 8837. Lecture Notes in Computer Science. Springer Verlag.

Organisation of the chapter. In Section 2.1 we introduce Markov automata and describe their behavioural notions in Section 2.2. We continue with the definition of Markov reward automata in Section 2.3 and describe their paths and traces in Section 2.4. We continue with a description of schedulers on Markov reward automata in Section 2.5 and the parallel composition in Section 2.6. Finally we give an overview of several bisimulation relations in Section 2.7, first on Markov automata and then their extension to Markov reward automata. Section 2.8 concludes the chapter.

2.1 Markov automata

Markov automata (MAs) have been introduced in [EHZ10a] as a continuous-time version of Segalas probabilistic automata (PAs) [Seg95]. The idea of MAs is to have a compositional model supporting continuous-time as well as discrete probabilities. One can also view MAs as the union of interactive Markov chains (IMCs) [Her02] and PAs. Thus, as for IMCs, a transition in an MA is either

Figure 2.1: Markov automata and related models.

Figure 2.2: An example Markov automaton.

labelled with a positive real number representing the rate of a negative exponential distribution, or with an action. Moreover, as for PAs, a transition labelled with an action leads to a discrete probability distribution. Thus, MAs support stochastic timing, non-determinism as well as discrete probabilities. Therefore, MAs can model action transitions as in labelled transition systems (LTSs) including non-deterministic choices, probabilistic branching as in discrete-time Markov chains (DTMCs), as well as delays that are governed by an exponential distribution as in continuous-time Markov chains (CTMCs). Hence, MAs can be seen as a superset of these models. Figure 2.1 depicts a hierarchy of models that are covered by MAs. More details on how MAs are covering these models follows in Section 2.2.3.

Example 2.1. Consider the MA depicted in Figure 2.2. States are depicted as circles. Transitions labelled with a rate are represented by dashed lines. Transitions labelled with an action are represented by solid lines and lead into a black dot representing a probability distribution. From there transitions labelled with the corresponding probability lead to the successor states. The example itself models a simple repairable component. The component can be *up*, *degraded*, *failed* and *down*. In the beginning the component is fully functional (state s_0) and in its *up* state. After an exponential delay the component can fail (state s_2) or reach a certain level of degradation (state s_1). If the component is degraded, there exists a mechanism which detects the degradation after a certain time. If the degradation is detected (state s_4) an inspection will be executed and the component will be taken down (state s_4). If the component is down, it can be repaired, which will have the risk of still being degraded or it can be replaced such that it is new.

First of all we like to introduce distributions and their notations that are used throughout this chapter.

Definition 2.1 (Distributions). A *probability distribution* over a countable set S is a function

$$\mu: S \rightarrow [0, 1] \text{ such that } \sum_{s \in S} \mu(s) \leq 1.$$

We write $|\mu| = \sum_{s \in S} \mu(s)$ for the size of the probability distribution. Let

$$\text{supp}(\mu) = \{s \in S \mid \mu(s) > 0\}$$

be the *support* of μ . If $\text{supp}(\mu) = \{s'\}$ is a singleton, we call μ a *Dirac distribution* for s . We write $\mathbf{1}_s$ for the Dirac distribution over s , given by $\mathbf{1}_s(s') = 1$ and $\mathbf{1}_s(t) = 0$ for all $t \in S$ such that $t \neq s'$. We say μ is a

- *full distribution* if $|\mu| = 1$, and a
- *sub-distribution* if $|\mu| < 1$.

Let $\text{Distr}(S)$ and $\text{Subdistr}(S)$ denote the set of all distributions and sub-distributions over S , respectively.

We will introduce an Markov automaton as a 5-tuple. The state space is given by a finite set of states, including a dedicated initial state. The transition choices are given by a finite set of actions, including an invisible action denoted by τ . The transition relation is given by a set of action-labelled probabilistic transitions and a set of rate-labelled (Markovian) transitions.

Definition 2.2 (Markov automaton). A *Markov automaton (MA)* is a tuple $\mathcal{A} = \langle S, s^0, \text{Act}, \hookrightarrow, \rightsquigarrow \rangle$, where

- S is a finite set of *states*, where $s^0 \in S$ is the *initial state*;
- Act is a finite set of *actions*, including τ ;
- $\hookrightarrow \subseteq S \times \text{Act} \times \text{Distr}(S)$ is the *probabilistic transition relation*;
- $\rightsquigarrow \subseteq S \times \mathbb{R}_{>0} \times S$ is the *Markovian transition relation*;

If $(s, \alpha, \mu) \in \hookrightarrow$, we write $s \xrightarrow{\alpha} \mu$ and say that action α can be *executed* from state s , after which the probability to go to each $s' \in S$ is $\mu(s')$. If $(s, \lambda, s') \in \rightsquigarrow$, we write $s \xrightarrow{\lambda} s'$ and say that s moves to s' with rate λ .

Note that an MA can be extended by a finite set of *atomic propositions* AP (also called *state labels*) and a state labelling function $L : S \rightarrow \mathcal{P}(AP)$, where $\mathcal{P}(AP)$ is the power set of the set of AP . For instance the MA in Figure 2.2 has a state labelling assigned.

Example 2.2. The formal definition of the MA depicted in Figure 2.2 is given

by the tuple $\mathcal{A} = \langle S, s^0, Act, \hookrightarrow, \rightsquigarrow \rangle$ with

$$\begin{aligned} S &= \{s_0, s_1, s_2, s_3, s_4\}; \\ s^0 &= s_0; \\ Act &= \{fail, insp?, repair?, replace?\}; \\ \hookrightarrow &= \{(s_2, fail, \mathbb{1}_{s_4}), (s_3, insp?, \mathbb{1}_{s_4}) \\ &\quad (s_4, repair?, \{s_0 \mapsto 0.4, s_1 \mapsto 0.6\}), (s_4, replace?, \mathbb{1}_{s_0})\}; \\ \rightsquigarrow &= \{(s_0, 4, s_2), (s_0, 2, s_1), (s_1, 2, s_2), (s_2, 1, s_3), (s_3, 2, s_2)\}; \end{aligned}$$

and extended with the following atomic propositions and state labels

$$\begin{aligned} AP &= \{up, degraded, failed, down\}; \\ L(s_0) &= \{up\}; & L(s_1) &= \{degraded\}; & L(s_2) &= \{failed, down\}; \\ L(s_3) &= \{degraded\}; & L(s_4) &= \{down\}. \end{aligned}$$

Let $PT(s)$ be the set of probabilistic transitions of a state $s \in S$ and $MT(s)$ the set of Markovian transitions, respectively. We denote by PT and MT the set of all probabilistic and Markovian transitions, respectively. A state $s \in S$ that has at least one transition $s \xrightarrow{a} \mu$ is called *probabilistic*. A state that has at least one transition $s \rightsquigarrow s'$ is called *Markovian*. Note that a state could be both probabilistic and Markovian. Such states are called *hybrid*. We define the set of probabilistic states by

$$PS = \{s \in S \mid PT(s) \neq \emptyset \wedge MT(s) = \emptyset\},$$

the set of Markovian states by

$$MS = \{s \in S \mid MT(s) \neq \emptyset \wedge PT(s) = \emptyset\},$$

and the set of hybrid states by

$$HS = \{s \in S \mid MT(s) \neq \emptyset \wedge PT(s) \neq \emptyset\}.$$

The set of actions Act can be partitioned in a set of *external actions* Act^{ext} and *internal actions* Act^{int} , such that $Act = Act^{ext} \cup Act^{int}$ with $Act^{ext} \cap Act^{int} = \emptyset$. Note that τ is considered as an internal action which is *not observable*. We denote by $Act(s)$ the set of enabled actions in $s \in S$.

There are two types of non-determinism in an MA. The first type of non-determinism is about the choice over the enabled actions $Act(s)$ in state $s \in S$, known as *external non-determinism*. We say that a state contains non-determinism if $|Act(s)| > 1$. The second type of non-determinism is the choice over the enabled transitions induced by an enabled action $\alpha \in Act(s)$ in state $s \in S$. We say a state $s \in S$ contains *action non-determinism*, also known as *internal non-determinism*, if there exists an action $\alpha \in Act(s)$ such that $\deg(s, \alpha) > 1$, where $\deg(s, \alpha) = |\{(s, \alpha, \mu) \in PT(s) \mid \mu \in \text{Distr}(S)\}|$ denotes the degree of action non-determinism induced by α in $s \in S$.

The *rate* between two states $s, s' \in S$ and the *outgoing rate* of a state $s \in S$ is given by

$$\mathbf{R}(s, s') = \sum_{(s, \lambda, s') \in \rightsquigarrow} \lambda \quad \text{and} \quad E(s) = \sum_{s' \in S} \mathbf{R}(s, s'),$$

respectively. We require $E(s) < \infty$ for every state $s \in S$. If $E(s) > 0$, the *branching probability distribution* after this delay is denoted by $\mathbb{P}_s$ and defined by

$$\mathbb{P}_s(s') = \frac{\mathbf{R}(s, s')}{E(s)}$$

for every $s' \in S$. By definition of the exponential distribution, the probability of leaving a state s within t time units is given by $1 - e^{-E(s) \cdot t}$ (given $E(s) > 0$), after which the next state is chosen according to $\mathbb{P}_s$. We denote by $E(\mathcal{A}) = \{E(s) \mid s \in S\}$ the set of all exit rates in MA $\mathcal{A}$.

Example 2.3. Let $\mathcal{A}$ be the MA depicted in Figure 2.2. Consider the Markovian state $s_0 \in MS$ and its Markovian transition $s_0 \xrightarrow{4} s_2$ (depicted as dashed line). The transition's delay is exponentially distributed with rate $\lambda = \mathbf{R}(s_0, s_2) = 4$; thus it expires in the next $t \in \mathbb{R}_{\geq 0}$ time units with probability

$$\int_0^t \lambda e^{-\lambda t} dt = (1 - e^{-4t}).$$

As there exists another outgoing Markovian transition from state s_0 , both transitions are competing for execution. Hence, the MA will move along from state s_0 with the transition whose delay expires first. Therefore, the time in state s_0 has to be considered, which is determined by its exit rate $E(s_0) = 4 + 2 = 6$. Then the probability to move from s_0 to its successor s_1 or s_2 is equal to the probability that the corresponding Markovian transition wins the race. Thus we move to s_1 with $\mathbb{P}_{s_0}(s_1) = \frac{\mathbf{R}(s_0, s_1)}{E(s_0)} = \frac{1}{3}$ and to s_2 with $\mathbb{P}_{s_0}(s_2) = \frac{\mathbf{R}(s_0, s_2)}{E(s_0)} = \frac{2}{3}$.

Remark 2.1. Instead of having a single initial state, a probability distribution defining a set of initial states I could be used. Thus, we give an initial distribution $\iota : \text{Distr}(S)$ with $s \in I$ if $\iota(s) > 0$. Note that this behaviour can be mimicked in Definition 2.2 by defining a τ -transition from s^0 leading into ι .

2.2 Behavioural notions of MAs

The distinction of the action set in external and internal actions is done to differentiate which actions are visible to the outside, and thus can interact with the environment. Hence, in comparison to external actions, internal actions are not subject to any more synchronisation. Therefore, they only provide information about the resulting probability distribution to reach a state by performing a given action. It is assumed that internal actions in an MA *fire immediately*. Now consider a hybrid state $s \in HS$ with one probabilistic internal transition and one Markovian transition. The probabilistic transition will be fired immediately. However, the probability for the Markovian transitions to happen immediately is zero. Hence, given the transition $s \xrightarrow{\lambda} s'$, the probability to advance in $t = 0$ time units is given by $\mathbb{P}_s^{\leq 0}(s') = (1 - e^{-\lambda \cdot 0}) = 0$.

Definition 2.3 (Maximal progress assumption). In any MA, probabilistic transitions labelled with internal actions take precedence over Markovian transitions.

Thus, the *maximal progress assumption* prescribes internal transitions to never be delayed. Hence, a state that has at least one outgoing internal transition can never take a Markovian transition. For closed MAs it holds that $HS = \emptyset$ when applying the maximal progress assumption. Note that we will use the term τ -transition, and $s \xrightarrow{\tau} \mu$, as synonyms when we speak about internal action transitions.

To provide a uniform manner for dealing with both probabilistic and Markovian transitions in an MA we follow the concept of *extended transitions* introduced in [EHZ10b]. The extended transition relation is equivalent to the probabilistic transition relation, where Markovian rates are encoded as extended actions. Thus, a probabilistic transition is equivalent to an extended transition, whereas we have to lift all outgoing Markovian transitions from a state s to a single extended transition.

Definition 2.4 (Extended action set). Let $\mathcal{A} = \langle S, s^0, Act, \hookrightarrow, \rightsquigarrow \rangle$ be a MA, then the *extended action set* of $\mathcal{A}$ is given by

$$Act^X = Act \cup \{\chi(r) \mid r \in E(\mathcal{A})\}.$$

Given a state $s \in S$ and an action $\alpha \in Act^X$, we write $s \xrightarrow{\alpha} \mu$ if either

- $\alpha \in Act$ and $s \xrightarrow{\alpha} \mu$, or
- $\alpha = \chi(E(s))$, $E(s) > 0$, $\mu = \mathbb{P}_s$ and there is no μ' such that $s \xrightarrow{\tau} \mu'$.

A transition $s \xrightarrow{\alpha} \mu$ is called an *extended transition*. Let $ET(s)$ be the set of extended transitions of a state $s \in S$ and ET the set of all extended transitions.

Note that the actions $\chi(r)$ represent exit rates and are used to distinguish probabilistic and Markovian transitions. Further, the maximal progress assumption is directly encoded into the extended transitions. Thus, an extended transition with a $\chi(r)$ action from a state s is only defined if no τ -transition exits from that state. We denote with

$$Succ(s, \alpha) = \{s' \in S \mid \forall s \xrightarrow{\alpha} \mu \text{ with } \mu(s') > 0\}$$

the set of successors of state $s \in S$ according to action $\alpha \in Act^X$ and with

$$Succ(s, \alpha, \mu) = \{s' \in S \mid s \xrightarrow{\alpha} \mu \text{ with } \mu(s') > 0\}$$

the set of successors of an extended transition $s \xrightarrow{\alpha} \mu$.

Example 2.4. Consider the MA $\mathcal{A}$ depicted in Figure 2.3a. We now use Definition 2.4 and define the set of extended actions and transitions on $\mathcal{A}$. The corresponding MA with extended transitions is depicted in Figure 2.3b. Let $Act^X = Act \cup \{\chi(3), \chi(6)\}$, since $E(s_0) = 3$ and $E(s_1) = 6$. Now consider state s_0 and its two transitions $s_0 \xrightarrow{\tau} \{s_0 \mapsto 0.2, s_1 \mapsto 0.8\}$ and $s_0 \xrightarrow{3} s_1$. The probabilistic transition is kept as extended transition $s_0 \xrightarrow{\tau} \{s_0 \mapsto 0.2, s_1 \mapsto 0.8\}$,

Figure 2.3: An example of an extended transition MA.

Figure 2.4: An example transformation of an MA into a digraph.

whereas the Markovian transition is neglected due to the fact that there exists a τ -transition out of s_0 . Hence, the maximal progress assumption is applied. State s_1 has two outgoing Markovian transitions which will be represented by one extended transition $s_1 \xrightarrow{\chi(6)} \{s_0 \mapsto \frac{2}{3}, s_1 \mapsto \frac{1}{3}\}$.

2.2.1 Structural properties of MAs

When speaking about structural properties of an MA, we are interested in properties depending only on the abstract structure of the MA. In particular we are interested in the induced *underlying graph structure* of the MA, but not in the probability distributions or Markovian rates. Therefore, an MA can be represented as a *directed graph*, short *digraph*. The idea is to map each state $s \in S$ to a node $\langle s \rangle$ and introduce new nodes for each extended transition. Thus, given an extended transition $s \xrightarrow{\alpha} \mu$, we introduce a new node $\langle s, \alpha, \mu \rangle$ with an incoming transition from $\langle s \rangle$ and outgoing transitions to all $\langle s' \rangle$ with $\mu(s') > 0$.

Definition 2.5 (Digraph). Let $\mathcal{A} = \langle S, s^0, Act, \hookrightarrow, \rightsquigarrow \rangle$ be an MA. The corresponding directed graph (*digraph*) induced by $\mathcal{A}$ is given by $G_{\mathcal{A}} = (V, E)$ where

- $V = S \cup ET$ is the set of vertices;
- $E \subseteq V \times V$ with $(s, \langle s, \alpha, \mu \rangle) \in E$ if $(s, \alpha, \mu) \in ET(s)$ and $(\langle s, \alpha, \mu \rangle, s') \in E$ if $s' \in Succ(s, \alpha, \mu)$ for all $s \in S$.

We write $s \rightarrow_G s'$ if $(s, s') \in E$. Note that the digraph does not explicitly encode any probability distributions. However, the nodes introduced for each extended transition carry the probability distribution as an identifier.

Example 2.5. Figure 2.4 depicts the MA $\mathcal{A}$ and its corresponding digraph $G_{\mathcal{A}} = (V, E)$. Consider state s_0 and the extended transition $s_0 \xrightarrow{\beta} \mathbb{1}_{s_1}$. In the digraph $G_{\mathcal{A}}$ there exists now a node $\langle s_0, \beta, \mathbb{1}_{s_1} \rangle$ representing this transition. The full set of vertices for $G_{\mathcal{A}}$ is given by

$$V = \{\langle s_0 \rangle, \langle s_0, \alpha, \mu \rangle, \langle s_0, \beta, \mathbb{1}_{s_1} \rangle, \langle s_0 \rangle, \langle s_0, \chi(\lambda), \mathbb{1}_{s_0} \rangle\}$$

where $\mu(s_0) = 0.2$ and $\mu(s_1) = 0.8$. Now reconsider the extended transition $s_0 \xrightarrow{\beta} \mathbb{1}_{s_1}$. In $G_{\mathcal{A}}$ this transition is represented by the directed edge $\langle s_0 \rangle \rightarrow_G \langle s_0, \beta, \mathbb{1}_{s_1} \rangle$ and the directed edge $\langle s_0, \beta, \mathbb{1}_{s_1} \rangle \rightarrow_G \langle s_0 \rangle$. The full set of edges is given by

$$\begin{aligned} E = \{ & \langle s_0 \rangle \rightarrow_G \langle s_0, \alpha, \mu \rangle, \\ & \langle s_0 \rangle \rightarrow_G \langle s_0, \beta, \mathbb{1}_{s_1} \rangle, \\ & \langle s_0, \alpha, \mu \rangle \rightarrow_G \langle s_0 \rangle, \\ & \langle s_0, \alpha, \mu \rangle \rightarrow_G \langle s_1 \rangle, \\ & \langle s_0, \beta, \mathbb{1}_{s_1} \rangle \rightarrow_G \langle s_1 \rangle, \\ & \langle s_1 \rangle \rightarrow_G \langle s_1, \chi(\lambda), \mathbb{1}_{s_0} \rangle, \\ & \langle s_1, \chi(\lambda), \mathbb{1}_{s_0} \rangle \rightarrow_G \langle s_0 \rangle \}. \end{aligned}$$

When arguing over certain properties of an MA it is not always necessary to consider the whole MA. Instead, it can be sufficient to define a subset of the MA representing the section of interest. A naive way would be to define a new MA based on the section of interest. However, this would mean to duplicate the already existing definitions. Further, we would need to define a relation between the original MA and the new MA. Another more efficient way is to define a *sub-MA*, equivalent to [Alf99]. The idea is to define a pair describing the subset of states and the enabled transitions in those states. Then this pair describes a sub-MA over the MA by using its transition relations.

Definition 2.6 (Sub-MA). Let $\mathcal{A} = \langle S, s^0, Act, \hookrightarrow, \rightsquigarrow \rangle$ be an MA. A *sub-MA* of $\mathcal{A}$ is defined over a pair $\mathcal{S} = (S', T)$ where $S' \subseteq S$ with $S' \neq \emptyset$ and $T : S' \rightarrow \mathcal{P}(ET)$ is a function such that

- $\emptyset \neq T(s) \subseteq ET(s)$ for all $s \in S'$;
- $s \in S'$ and $(s, \alpha, \mu) \in T(s)$ implies $Succ(s, \alpha, \mu) \subseteq S'$.

We write $\mathcal{A}_{\mathcal{S}}$ for the sub-MA of $\mathcal{A}$ over $\mathcal{S}$.

Hence, a sub-MA describes a subset of states and transitions in an MA such that the post condition for all included states and transitions is fulfilled. Note that the set of states S' of a sub-MA $\mathcal{A}_{\mathcal{S}}$ may not contain the initial state s^0 . Thus, a sub-MA $\mathcal{A}_{\mathcal{S}}$ would induce an MA over $\mathcal{A}$ with state space S' where s^0 could be an arbitrary state in S' , and transition relations $\hookrightarrow'$ and $\rightsquigarrow'$ are according to T .

Based on the definitions of a digraph and a sub-MA we are now able to describe structural properties of an MA which will be helpful for reachability

Figure 2.5: An MA displaying different types of end components.

analysis. If we consider MAs with cyclic behaviour, it is of interest to find out which states and transitions could be visited infinitely often. We call the set of states and transitions that allow such a behaviour an *end component*. Hence, if we speak of an end component in an MA $\mathcal{A}$, it means that it is possible to cycle forever in this part of $\mathcal{A}$.

Definition 2.7 (End component). Let $\mathcal{A} = \langle S, s^0, Act, \hookrightarrow, \rightsquigarrow \rangle$ be an MA. An *end component* in $\mathcal{A}$ is a pair $\mathcal{S} = (S', T)$ such that the digraph $G_{\mathcal{A}_{\mathcal{S}}}$ induced by the sub-MA $\mathcal{A}_{\mathcal{S}}$ is strongly connected. We denote the set of end components of $\mathcal{A}$ by $EC(\mathcal{A})$.

Thus, if a sub-MA induces a strongly connected digraph, i.e. every state in the graph can be reached from every state, we speak of an end component. Since an end component can be included in another end component, there is a so called *maximal end component*, i.e. an end component that is not contained in any other end component.

Definition 2.8 (Maximal end component). Let $\mathcal{A} = \langle S, s^0, Act, \hookrightarrow, \rightsquigarrow \rangle$ be an MA. An end component $(S', T) \in EC(\mathcal{A})$ is called *maximal* if and only if there exists no end component $(S'', T') \in EC(\mathcal{A})$ such that $(S', T) \neq (S'', T')$ and $S' \subseteq S''$ and $T(s) \subseteq T'(s)$ for all $s \in S'$. We denote the set of maximal end components of $\mathcal{A}$ by $MEC(\mathcal{A})$.

By inspecting Definition 2.7 and 2.8, we see that for a state in an end component not all actions have to be contained. Thus, there could exist a way out of an end component. However, if for all states in an end component all actions are included, we speak of a *bottom strongly connected component*. Hence, if a state contained in such a component is reached, it is not possible to exit the component any more.

Definition 2.9 (Bottom strongly connected component). Let $\mathcal{A} = \langle S, s^0, Act, \hookrightarrow, \rightsquigarrow \rangle$ be an MA. A maximal end component $(S', T) \in MEC(\mathcal{A})$ is called *bottom strongly connected* if and only if for all states $s \in S'$ it holds that $T(s) = ET(s)$. We denote the set of bottom strongly connected components of $\mathcal{A}$ by $BSCC(\mathcal{A})$.

Example 2.6. Consider the MA $\mathcal{A}$ depicted in Figure 2.5. Given the pair $\mathcal{S} = (S', T)$ with

$$S' = \{s_3, s_4\} \quad \text{and} \quad T(s_3) = \{s_3 \xrightarrow{\alpha} s_4\} \quad \text{and} \quad T(s_4) = \{s_3 \xrightarrow{\alpha} s_4\}$$

we obtain a sub-MA $\mathcal{A}_{\mathcal{S}}$. The corresponding digraph $G_{\mathcal{A}_{\mathcal{S}}} = (V, E)$ is defined over

$$V = \{\langle s_3 \rangle, \langle s_3, \alpha, \mathbb{1}_{s_4} \rangle, \langle s_4 \rangle, \langle s_4, \alpha, \mathbb{1}_{s_3} \rangle\}$$

and

$$\begin{aligned} E = \{ & \langle s_3 \rangle \longrightarrow_G \langle s_3, \alpha, \mathbb{1}_{s_4} \rangle, \\ & \langle s_3, \alpha, \mathbb{1}_{s_4} \rangle \longrightarrow_G \langle s_4 \rangle, \\ & \langle s_4 \rangle \longrightarrow_G \langle s_4, \alpha, \mathbb{1}_{s_3} \rangle, \\ & \langle s_4, \alpha, \mathbb{1}_{s_3} \rangle \longrightarrow_G \langle s_3 \rangle \}. \end{aligned}$$

Since $G_{\mathcal{A}_{\mathcal{S}}}$ is a strongly connected component, the pair $\mathcal{S}$ describes an end component. However, $\mathcal{S}$ is not maximal, since it is contained in another end component. Let $\mathcal{S}' = (S'', T')$ with

$$\begin{aligned} S'' &= S' \cup \{s_1\}, \quad T'(s_3) = T(s_3), \\ T'(s_4) &= T(s_4) \cup \{s_4 \xrightarrow{\beta} s_1\} \quad \text{and} \quad T'(s_1) = \{s_1 \xrightarrow{\chi(\lambda_1 + \lambda_2)} \mu\} \end{aligned}$$

where $\mu(s_3) = \frac{\lambda_2}{\lambda_1 + \lambda_2}$ and $\mu(s_4) = \frac{\lambda_1}{\lambda_1 + \lambda_2}$ be the pair describing the maximal end component in Figure 2.5. For $\mathcal{S}$ it holds that $S' \subset S''$, $T(s_3) \subseteq T'(s_3)$ and $T(s_4) \subset T'(s_4)$. Thus, $\mathcal{S}$ is included in $\mathcal{S}'$ and therefore not maximal. Further, there exists no pair in $\mathcal{A}$ such that $\mathcal{S}'$ is included in it, and therefore $\mathcal{S}'$ is maximal. However, $\mathcal{S}'$ is not a bottom strongly connected component since $T(s_3) \neq ET(s_3)$ and we could leave $\mathcal{S}'$ in $\mathcal{A}$ by taking the extended transition $s_3 \xrightarrow{\beta} s_2$. A bottom strongly connected component in $\mathcal{A}$ is given by the pair $\mathcal{S}'' = (S''', T'')$ with

$$\begin{aligned} S''' &= \{s_5, s_6, s_7\}, \quad T(s_5) = ET(s_5), \\ T(s_6) &= ET(s_6) \quad \text{and} \quad T(s_7) = ET(s_7). \end{aligned}$$

Hence, after we enter $\mathcal{S}''$ in $\mathcal{A}$ we will stay forever in the sub-MA $\mathcal{A}_{\mathcal{S}''}$.

2.2.2 Open and closed behaviour

For *open* systems, i.e. systems that have visible actions, the precise behaviour is unknown and depends on the environment. In contrast to internal actions that always happen immediately, the firing of external actions is assumed to be dependent on the environment. Therefore external actions can be delayed until the environment is ready to synchronise on them.

We speak of an *open MA* if some actions are visible to the environment, and therefore can interact with it. Consider an MA modelling a server and a second MA modelling a client with access to the server. Now the server would wait for an input of the client before executing the requested task. This communication would be defined with respect to external actions.

Definition 2.10 (Open MA). Let $\mathcal{A} = \langle S, s^0, Act, \hookrightarrow, \rightsquigarrow \rangle$ be an MA. We say that $\mathcal{A}$ is *open* iff $Act^{ext} \neq \emptyset$.

To interpret the precise behaviour of a system represented by an open MA, we have to consider the environment. After all, the timing of a transition labelled with an external action is assumed to be dependent on the environment to synchronise on it.

We speak of a *closed MA* if no communication with the environment is possible. Hence, the MA will not contain any external actions. Therefore, all remaining actions are internal and not subject to any further synchronisation. In other words, the MA does not have to wait for any input from the environment.

Definition 2.11 (Closed MA). Let $\mathcal{A} = \langle S, s^0, Act, \hookrightarrow, \rightsquigarrow \rangle$ be an MA. We say that $\mathcal{A}$ is *closed* iff $Act^{ext} = \emptyset$.

Note that every open MA could behave like a closed MA by transforming the external actions used to synchronise with the environment into internal actions. The concept of this transformation is called *hiding*. The idea is to define a set of external actions which are not subject to any more interaction, and declare them as internal actions.

Definition 2.12 (Hiding). Given an MA $\mathcal{A} = \langle S, s^0, Act, \hookrightarrow, \rightsquigarrow \rangle$ and a set $H \in Act^{ext}$, we hide H in $\mathcal{A}$ such that $\mathcal{A}_{\setminus H} = \langle S, s^0, Act', \hookrightarrow, \rightsquigarrow \rangle$ with $Act' = Act^{ext'} \cup Act^{int'}$ with $Act^{ext'} = Act^{ext} \setminus H$ and $Act^{int'} = Act^{int} \cup H$.

Hence, by setting $H = Act^{ext}$ an open MA can be *closed*. Note that while hiding actions of hybrid states, also the maximal progress assumption can be applied (see Definition 2.3).

2.2.3 Subsumption of Markov models

MAs can be seen as a superset of several automata models since they feature discrete probabilities, non-determinism, as well as exponential delays. Figure 2.1 on page 20 depicts a diagram of what models are subsumed by MAs. We can give a definition for each model in Figure 2.1 using Definition 2.2 with an appropriate condition. Let $\mathcal{A} = \langle S, s^0, Act, \hookrightarrow, \rightsquigarrow \rangle$ be an MA.

Labelled transition system. A labelled transition system (LTS) contains non-deterministic choices, direct successors and action labelled transitions. An MA $\mathcal{A}$ is equivalent to an LTS iff $MT(s) = \emptyset$ for all $s \in S$ and for all $(s, \alpha, \mu) \in \hookrightarrow$ it holds that $\mu = \mathbf{1}_s$.

Discrete-time Markov chain. A probabilistic model in deterministic time is given by a discrete-time Markov chain (DTMC). An MA $\mathcal{A}$ is a DTMC iff $MT(s) = \emptyset$ and $|Act(s)| = 1$ for all $s \in S$.

Continuous-time Markov chain. A continuous-time variant of DTMCs is given by a continuous-time Markov chain (CTMC). An MA $\mathcal{A}$ is a CTMC iff $PT(s) = \emptyset$ for all $s \in S$.

Markov decision process. A probabilistic model with non-deterministic choices is given by an Markov decision process (MDP). An MA $\mathcal{A}$ is a MDP iff $MT(s) = \emptyset$ for all $s \in S$ and for all $s \in PS$ and $\alpha \in Act(s)$ it holds that $|\{(s, \alpha, \mu) \in \hookrightarrow \mid \mu \in \text{Distr}(S)\}| = 1$.

Probabilistic automaton. A probabilistic model with non-deterministic choices, including action non-determinism, is given by a probabilistic automaton (PA). A MA $\mathcal{A}$ is a PA iff $MT(s) = \emptyset$ for all $s \in S$.

Interactive Markov chain. By restricting the probabilistic transitions to Dirac distributions, i.e. for all $(s, \alpha, \mu) \in \hookrightarrow$ it holds that $\mu = \mathbb{1}_s$, the MA $\mathcal{A}$ is equivalent to interactive Markov chains (IMCs).

2.3 Markov reward automata

We now introduce the Markov reward automaton (MRA). For simplicity, we choose to define MRAs in terms of two separate reward functions. Moreover, instead of integrating rewards directly into the transition relation as we did in [GTHRS14b], we define a separate reward function over extended transitions.

Definition 2.13 (Markov reward automaton). A *Markov reward automaton* (MRA) is a tuple $\mathcal{M} = (\mathcal{A}, \rho, r)$, where

- $\mathcal{A}$ is a Markov automaton;
- $\rho: S \rightarrow \mathbb{R}_{\geq 0}$ is the *state-reward function*;
- $r: S \times Act^X \times \text{Distr}(S) \times S \rightarrow \mathbb{R}_{\geq 0}$ is the *transition-reward function*, such that for all $(s, \alpha, \mu, s') > 0$ there exists an extended transition $s \xrightarrow{\alpha, \mu} \mu$ with $\mu(s') > 0$.

The function ρ associates a non-negative real number to each state. This number may be zero. The state-based rewards are gained *while being in a state*, and are proportional to the duration of this stay. The function r associates a non-negative real number to a transition. The transition-based rewards are gained when *taking the transition* and are *instantaneous*. We write $s \xrightarrow{\alpha, \mu} r$ to denote the transition reward induced by taking a transition out of state s with action α and probability distribution μ , and r_s^Θ with $\Theta = (\alpha, \mu)$ as its shorthand notation. Further we write $r_s^\Theta(s')$ as a shorthand for $r(s, \alpha, \mu, s')$. In case s contains no internal non-determinism we just use α instead of Θ . Further, we omit the action subscript whenever clear from context.

Remark 2.2. Note that the transition-reward function r is not directly defined over the transition relations $\hookrightarrow$ and $\rightsquigarrow$. Instead the transition reward is defined over the extended transition relation plus a successor state. Assume we have a probabilistic transition $s \xrightarrow{\alpha} \mu$, then we can assign an individual reward for reaching a successor $s' \in S$ where $\mu(s') > 0$.

2.4 The behaviour of MRAs

To argue over the behaviour of MRAs we can define *paths* and *traces* through them. In the following we define concepts and notions for both. Generally speaking, a path describes a traversal through an MRA indicating which transitions were taken, including information about the action, timing and probability distribution. On the other hand, a trace represents the observable part of a path, i.e. all external actions and timing information.

2.4.1 Paths

As for traditional labelled transition systems (LTSs), we can define paths through an MRA. A *path* in $\mathcal{M}$ is a finite sequence

$$\pi^* = s_0 \xrightarrow{\alpha_0, \mu_0, t_0} s_1 \xrightarrow{\alpha_1, \mu_1, t_1} \dots \xrightarrow{\alpha_{n-1}, \mu_{n-1}, t_{n-1}} s_n$$

from some state s_0 to a state s_n ($n \geq 0$), or an infinite sequence

$$\pi^\omega = s_0 \xrightarrow{\alpha_0, \mu_0, t_0} s_1 \xrightarrow{\alpha_1, \mu_1, t_1} s_2 \xrightarrow{\alpha_2, \mu_2, t_2} \dots$$

with $s_i \in S$ for all $0 \leq i \leq n$ and all $0 \leq i$, respectively. One step $s_i \xrightarrow{\alpha_i, \mu_i, t_i} s_{i+1}$ denotes that after residing $t_i \in \mathbb{R}_{\geq 0}$ time units in s_i , the MRA has moved via action α_i and probability distribution μ_i to s_{i+1} . We use $\text{prefix}(\pi, t)$ to denote the prefix of path π up to and including time t , formally

$$\text{prefix}(\pi, t) = s_0 \xrightarrow{\alpha_0, \mu_0, t_0} \dots \xrightarrow{\alpha_{i-1}, \mu_{i-1}, t_{i-1}} s_i$$

such that $t_0 + \dots + t_{i-1} \leq t$ and $t_0 + \dots + t_{i-1} + t_i > t$. We use $\text{step}(\pi^*, i)$ to denote the transition $s_{i-1} \xrightarrow{\alpha_{i-1}} \mu_i$. When π is finite we define the length of a path by $|\pi| = n$ and its last state by $\pi \downarrow = s_n$. Further, we denote by π^j the path π up to and including state s_j and with $\pi[j] = s_j$ the state on path π on position j . Let Paths^* and Paths^ω denote the set of finite and infinite paths, respectively. Then, the set of all paths in an MRA $\mathcal{M}$ is given by $\text{Paths}(\mathcal{M}) = \text{Paths}^*(\mathcal{M}) \cup \text{Paths}^\omega(\mathcal{M})$.

Definition 2.14 (Elapsed time). Let $\mathcal{M} = (\mathcal{A}, \rho, r)$ be an MRA and $\pi \in \text{Paths}^*(\mathcal{M})$ a finite path in $\mathcal{M}$. The time elapsed on path π is given by

$$\Delta(\pi) = \sum_{i=0}^{|\pi|-1} t_i.$$

We write $\Delta(\pi, n)$ to denote the elapsed time on path π up to state $\pi[n]$ and $\Delta(\pi[i]) = \Delta(\pi, i) - \Delta(\pi, i-1)$ as the time spent in state s_i on path π . Further, we denote with $\pi @ t$ the states that π occupies at time point t . Formally, $\pi @ t \in (S^* \cup S^\omega)$ is the sequence of states that are visited on π during time point $t \in \mathbb{R}_{\geq 0}$. Let i be the smallest index, such that $t \leq \Delta(\pi)$. Then $\pi[i]$ is the first state on π that is visited at or after time point t . If such a state does not exist, then let $\pi @ t = \langle \rangle$. Otherwise, we distinguish two cases:

$t \leq \Delta(\pi, i)$	0	1	2	3	4	5	6	7	min j	max j	$\pi@t$
0	✓	✓	✓	✓	✓	✓	✓	✓	0	3	$\langle s_0 s_1 s_2 s_3 \rangle$
$t_3 - \epsilon$	×	×	×	×	✓	✓	✓	✓	4	-	$\langle s_3 \rangle$
t_3	×	×	×	×	✓	✓	✓	✓	4	5	$\langle s_4 s_5 \rangle$
$t_3 + \epsilon$	×	×	×	×	×	×	✓	✓	6	-	$\langle s_5 \rangle$
$t_3 + t_5$	×	×	×	×	×	×	✓	✓	6	7	$\langle s_6 s_7 \rangle$

Table 2.1: An example derivation of $\pi@t$.

1. If $t < \Delta(\pi, i)$, then $\pi@t = \langle s_{i-1} \rangle$;
2. If $t = \Delta(\pi, i)$, then there exists a largest index j such that $t = \Delta(\pi, j)$ and $\pi@t = \langle s_i \dots s_j \rangle$.

We write $s \in \pi@t$ if the state s is contained in the sequence $\pi@t$.

Example 2.7. Consider the path

$$\begin{aligned} \pi = s_0 &\xrightarrow{\alpha_0, \mu_0, 0} s_1 \xrightarrow{\alpha_1, \mu_1, 0} s_2 \xrightarrow{\alpha_2, \mu_2, 0} s_3 \xrightarrow{\chi(\lambda_3), \mu_3, t_3} s_4 \xrightarrow{\alpha_4, \mu_4, 0} \\ &s_5 \xrightarrow{\chi(\lambda_5), \mu_5, t_5} s_6 \xrightarrow{\alpha_6, \mu_6, 0} s_7. \end{aligned}$$

Let $0 < \epsilon < \min\{t_3, t_5\}$. The derivations for the sequence $\pi@0, \pi@(t_3 - \epsilon), \pi@t_3, \pi@(t_3 + \epsilon)$ and $\pi@(t_3 + t_5)$ are depicted in Table 2.1, where ✓ indicates that $t \leq \Delta(\pi, i)$, and × denotes the states where $t > \Delta(\pi, i)$. Further, min j describes the minimum path length and max j the maximum path length such that $t \leq \Delta(\pi, j)$. Hence, with min j , $\pi[j]$ describes the first state on path π for the sequence $\pi@t$, respectively for max j the last state.

Definition 2.15 (Path reward). Let $\mathcal{M} = (\mathcal{A}, \rho, r)$ be an MRA and $\pi \in \text{Paths}^*(\mathcal{M})$ a finite path in $\mathcal{M}$. We define the *total reward* of π by

$$\text{reward}(\pi) = \sum_{i=0}^{|\pi|-1} \left(\rho(\pi[i]) \cdot t_i + r(\pi[i], \alpha_i, \mu_i, \pi[i+1]) \right).$$

Rewards can be used to model many quantitative aspects of systems, like energy consumption, memory usage, deployment or maintenance costs, etc. The total reward of a path (e.g. total amount of energy consumed) is obtained by adding all rewards along that path, that is, all state rewards multiplied by the sojourn times of the corresponding states plus all action rewards on the path.

Time-abstract paths. If time is not of interest it suffices to consider paths with no timing information. Thus a *time-abstract* path in $\mathcal{M}$ is a finite sequence

$$\pi_{abs}^* = s_0 \xrightarrow{\alpha_0, \mu_0} s_1 \xrightarrow{\alpha_1, \mu_1} \dots \xrightarrow{\alpha_{n-1}, \mu_{n-1}} s_n$$

from some state s_0 to a state s_n ($n \geq 0$), or an infinite sequence

$$\pi_{abs}^\omega = s_0 \xrightarrow{\alpha_0, \mu_0} s_1 \xrightarrow{\alpha_1, \mu_1} s_2 \xrightarrow{\alpha_2, \mu_2} \dots$$

with $s_i \in S$ for all $0 \leq i \leq n$ and all $0 \leq i$, respectively. Let $Paths_{abs}^*$ and $Paths_{abs}^\omega$ denote the set of finite and infinite paths, respectively. We denote with $abs(\pi)$ the time-abstract path of $\pi \in Paths$.

Zeno behaviour. An undesired behaviour on infinite paths is that time converges. Consider a path, where the first sojourn time is $\frac{1}{2}$ time units and each subsequent sojourn time is half as long. Thus we would obtain a time sequence $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots = 1$. Hence, we are converging to a bounded time point on an infinite path. This is known as Zeno behaviour.

Definition 2.16 (Zeno path). Let $\pi \in Paths^\omega$ be an infinite path in MRA $\mathcal{M}$. We say π is a *Zeno path* in $\mathcal{M}$ if and only if

$$\lim_{n \rightarrow \infty} \sum_{i=0}^n t_i < \infty.$$

Hence, an infinite path π is non-Zeno whenever the time along the path diverges to infinity.

2.4.2 Traces

A *trace* is a finite or infinite alternating sequence of state labels, actions and times. Intuitively, it represents the observable part of a path. For every path we define its trace as a sequence obtained by removing all the action and time pairs for internal actions. Thus, given an MRA $\mathcal{M}$ we define for any $\pi \in Paths(\mathcal{M})$

$$trace(\pi) = L(s_0)\alpha_0 t_0 L(s_1)\alpha_1 t_1 \dots$$

as the sequence of observable state labels, actions and times. Further, we write

$$trace^\alpha(\pi) = \alpha_0 \alpha_1 \dots$$

$$trace^L(\pi) = L(s_0)L(s_1)\dots$$

as the sequence of observable actions and state labellings, respectively. Notice that an infinite sequence with only finitely many external actions has a finite action trace. We define by $traces^*$ and $traces^\omega$ the set of all finite and infinite traces, respectively.

Furthermore, the measurable space over $traces^* \cup traces^\omega$ is endowed with a σ -algebra generated by all subsets of $traces^*$ and by all cylinder sets within $traces^\omega$, analogously to the generators of $Paths$.

σ -algebra. We give a short introduction on the σ -algebra for MAs. First we recapitulate the concept of *compound transitions*. A compound transition is a triple of (t, α, μ, s) , which describes the behaviour of the MA when it waits in its current state for t time units then takes action α and finally evolves to the next state s with probability $\mu(s)$. The set of all compound transitions over action space Act^X and state space S is denoted by

$$CT = \mathbb{R}_{\geq 0} \times Act^X \times \text{Distr}(S) \times S.$$

As a path in an MA is composed of a sequence of compound transitions originating from an initial state, first we define a σ -algebra over compound transitions and then extend it over finite and infinite paths.

Let $\mathfrak{F}_{Act^X} = 2^{Act^X}$, $\mathfrak{F}_{Distr} = 2^{Distr(S)}$ and $\mathfrak{F}_S = 2^S$ be σ -algebras over Act^X , $Distr(S)$ and S respectively. We define the σ -algebra over compound transitions using the concept of Cartesian product of a collection of σ -algebras [ADD00], as

$$\mathfrak{F}_{CT} = \sigma(\mathfrak{B}(\mathbb{R}_{\geq 0}) \times \mathfrak{F}_{Act^X} \times \mathfrak{F}_{Distr} \times \mathfrak{F}_S),$$

where $\mathfrak{B}(\mathbb{R}_{\geq 0})$ is the Borel σ -algebra over non-negative reals. Furthermore, it can be extended to the σ -algebra over finite paths using the same technique as follows.

Let

$$\mathfrak{F}_{Paths^n} = \sigma\left(\mathfrak{F}_S \times \prod_{i=1}^n \mathfrak{F}_{CT}\right)$$

be the σ -algebra over finite paths of length n , then the σ -algebra over finite paths is defined as

$$\mathfrak{F}_{Paths^*} = \bigcup_{i=0}^{\infty} \mathfrak{F}_{Paths^n}.$$

The σ -algebra over infinite paths is defined using the standard cylinder set construction [ADD00]. We define the cylinder set of a given base $B \in \mathfrak{F}_{Paths^n}$ as

$$Cyl(B) = \{\pi \in Paths^\omega : prefix(\pi, n) \in B\}.$$

$Cyl(B)$ is measurable if its base B is measurable. The σ -algebra over infinite paths, $\mathfrak{F}_{Paths^\omega}$, is therefore the smallest σ -algebra over measurable cylinders. Finally the σ -algebra over the set of paths is the disjoint union of the σ -algebras over the finite paths and the infinite paths.

2.5 Schedulers

Some part of the behaviour of an MRA is probabilistic, however if there exists a state with more than one outgoing probabilistic transition, the choice of the transition taken is non-deterministic. This non-deterministic behaviour in an MRA is resolved by *schedulers*, also known as *policies* and *adversaries*. That is to say, a scheduler makes a choice over the outgoing probabilistic transitions of a state. Besides, schedulers can have influence on the behaviour, e.g. by delaying an action or randomising the choice over taking an action. In the following we introduce the concept of *measurable* schedulers for *closed* MRAs, as well as provide several meaningful scheduler classes, following concepts of Wolovick and Johr [WJ06].

Definition 2.17 (Generic measurable scheduler). A *generic scheduler* over MRA $\mathcal{M} = (\mathcal{A}, \rho, r)$ is a function

$$D: \text{Paths}^* \rightarrow \text{Distr}(ET)$$

such that for each path π , where $s_n = \pi \downarrow$, for all $\alpha \in \text{Act}(s_n)$ and $\mu \in \text{Distr}(S)$, $D(\pi)(s_n, \alpha, \mu) > 0$ implies $s_n \xrightarrow{\alpha} \mu$ is taken with a certain probability. A generic scheduler is measurable iff for all $(s, \alpha, \mu) \subseteq ET$, $D(\cdot)(s, \alpha, \mu): \text{Paths}^* \rightarrow [0, 1]$ is measurable.

We denote with *GM* the class of generic measurable schedulers. For a finite path π , a scheduler resolves the non-determinism by defining a distribution over the set of enabled extended transitions in the last state of π . Measurability of D means that the scheduler never resolves non-determinism in a way that induces a set of path that are not measurable. Thus, given a scheduler, the behaviour of an MRA is fully probabilistic.

Schedulers can be classified based on the level of information they use to resolve non-determinism. In [WJ06; Joh08; Mar10] a variety of scheduler classes for *continuous-time Markov decision processes* (CTMDPs) were introduced to resolve non-determinism for different objectives. Those scheduler classes can also be deployed on MRAs. The schedulers are classified on the level of time and history information they need to resolve non-determinism. Another criterion is if the scheduler resolves the non-determinism *deterministically* or *randomised*, i.e. the choice is a Dirac distribution or a probabilistic distribution, respectively.

History-dependent. A scheduler is *history-dependent* if the resolution of the non-determinism is dependent on the states visited on π up to $\pi \downarrow$.

Positional. A scheduler is *positional* if the resolution of the non-determinism is only made based on the last state of the path.

Hop counting. A scheduler is *hop counting* if all paths with the same length lead to the same resolution of the non-determinism.

Time-dependent. A scheduler is *time-dependent* if it utilises all timing information provided by the path, including the individual sojourn time in each state as well as the total time spend on the path up to the last state.

Total time-dependent. A scheduler is *total time-dependent* if it only uses the total time elapsed on a path up to the last state.

Time-abstract. A scheduler is *time-abstract* if no timing information is used.

The most general class of schedulers, *GM* schedulers, uses the whole history of the path up to the current state and randomly determines which extended transition to take. Therefore, the class of *GM* schedulers can be classified as *time- and history-dependent randomised* (THR) schedulers. To define the different scheduler classes we use the following abbreviations: TA (time-abstract), TT (total time-dependent), T (time-dependent), P (positional), HOP (hop counting), and H (history-dependent).

	Scheduler class	Scheduler signature
TA	<i>TAPR</i>	$D: S \rightarrow \text{Distr}(ET)$
	<i>TAHOPR</i>	$D: S \times \mathbb{N} \rightarrow \text{Distr}(ET)$
	<i>TAHR</i>	$D: \text{Paths}_{abs}^* \rightarrow \text{Distr}(ET)$
TT	<i>TTPR</i>	$D: S \times \mathbb{R}_{\geq 0} \rightarrow \text{Distr}(ET)$
	<i>TTHOPR</i>	$D: S \times \mathbb{N} \times \mathbb{R}_{\geq 0} \rightarrow \text{Distr}(ET)$
	<i>TTHR</i>	$D: \text{Paths}_{abs}^* \times \mathbb{R}_{\geq 0} \rightarrow \text{Distr}(ET)$
T	<i>TPR</i>	$D: S \times \mathbb{R}_{\geq 0} \rightarrow \text{Distr}(ET)$
	<i>THR</i>	$D: \text{Paths}^* \rightarrow \text{Distr}(ET)$

Table 2.2: Scheduler classes and their corresponding signatures.

Definition 2.18 (Scheduler classes). Given an MRA $\mathcal{M}$ and a GM scheduler D . Let π and π' range over $\text{Paths}^*(\mathcal{M})$, the scheduler classes are defined as follows:

Scheduler class	$D(\cdot) = D(\cdot)$ whenever
<i>TAPR</i>	$\pi \downarrow = \pi' \downarrow$
<i>TAHOPR</i>	$\pi \downarrow = \pi' \downarrow \wedge \pi = \pi' $
<i>TAHR</i>	$\pi \downarrow = \pi' \downarrow \wedge \Delta(\pi) = \Delta(\pi')$
<i>TTPR</i>	$\pi \downarrow = \pi' \downarrow \wedge \Delta(\pi) = \Delta(\pi')$
<i>TTHOPR</i>	$\pi \downarrow = \pi' \downarrow \wedge \pi = \pi' \wedge \Delta(\pi) = \Delta(\pi')$
<i>TTHR</i>	$abs(\pi) = abs(\pi') \wedge \Delta(\pi) = \Delta(\pi')$
<i>TPR</i>	$\pi \downarrow = \pi' \downarrow \wedge (\Delta(\pi[\pi]) = \Delta(\pi'[\pi']))$
<i>THR</i>	$\Delta(\pi) = \Delta(\pi') \wedge \pi = \pi' = n \wedge \forall 0 < i \leq n. \Delta(\pi[i]) = \Delta(\pi'[i])$

The respective scheduler signatures for each class is given in Table 2.2.

2.6 Parallel composition

A key advantage of MRAs is that they are compositional, allowing a large system to be modelled through smaller sub-systems. Hence, in order to model a large system, one can model individual MRAs and use external actions to synchronise the MRAs with each other.

We can generalise the definition of parallel composition from MAs as presented in [Tim13] to MRAs. The concept of parallel composition between two MRAs $\mathcal{M}_1$ and $\mathcal{M}_2$ is based on the synchronisation of mutual external actions in the style of Milner [Mil89]. Hence, if $\mathcal{M}_1$ and $\mathcal{M}_2$ have an external action α , they will synchronise on transitions with that action. First we provide the definition of the parallel composition for MAs.

2

$(A) \frac{s_1 \xrightarrow{\alpha} \mu_1 \quad \alpha \notin A}{(s_1, s_2) \xrightarrow{\alpha} \mu_1 \times \mathbb{1}_{s_2}}$	$(B) \frac{s_2 \xrightarrow{\alpha} \mu_2 \quad \alpha \notin A}{(s_1, s_2) \xrightarrow{\alpha} \mathbb{1}_{s_1} \times \mu_2}$
$(C) \frac{s_1 \xrightarrow{\alpha} \mu_1 \quad s_2 \xrightarrow{\alpha} \mu_2 \quad \alpha \in A}{(s_1, s_2) \xrightarrow{\alpha} \mu_1 \times \mu_2}$	
$(D) \frac{s_1 \xrightarrow{\lambda} s'_1 \quad s_1 \neq s'_1}{(s_1, s_2) \xrightarrow{\lambda} (s'_1, s_2)}$	$(E) \frac{s_2 \xrightarrow{\lambda} s'_2 \quad s_2 \neq s'_2}{(s_1, s_2) \xrightarrow{\lambda} (s_1, s'_2)}$
$(F) \frac{\lambda(s_1, s_2) > 0}{(s_1, s_2) \xrightarrow{\lambda(s_1, s_2)} (s_1, s_2)}$	

Table 2.3: Inference rules for the transitions of a parallel composition (if all conditions above the line of a rule hold, then so should the condition below) where $\lambda(s_1, s_2) = \mathbf{R}(s_1, s_1) + \mathbf{R}(s_2, s_2)$.

Definition 2.19 (Parallel composition for MAs). Let $\mathcal{A}_1, \mathcal{A}_2$ be two MAs such that $\mathcal{A}_1 = \langle S_1, s_1^0, Act_1, \hookrightarrow_1, \rightsquigarrow_1 \rangle$ and $\mathcal{A}_2 = \langle S_2, s_2^0, Act_2, \hookrightarrow_2, \rightsquigarrow_2 \rangle$, and action set $A \subseteq Act_1^{ext} \cap Act_2^{ext}$. Their parallel composition w.r.t. set A of actions is the system $\mathcal{A}_1 \parallel_A \mathcal{A}_2 = \langle S, s^0, Act, \hookrightarrow, \rightsquigarrow \rangle$, with

- $S = S_1 \times S_2$;
- $Act = Act_1 \cup Act_2$;
- $s^0 = (s_1^0, s_2^0)$;
- $\hookrightarrow$ is the smallest relation fulfilling the inference rules (A) - (C) in Table 2.3;
- $\rightsquigarrow$ is the smallest relation fulfilling the inference rules (D) - (F) in Table 2.3.

Let's have a closer look at the inference rules in Table 2.3. The first observation is, that MA $\mathcal{A}_1$ and $\mathcal{A}_2$ will only synchronise on shared external actions that are specified in the set A . Hence, a transition from a state in $s_1 \in S_1$ and $s_2 \in S_2$ synchronises when $\alpha \in A$ and α is enabled in both states, see inference rule (C). Otherwise, if $\alpha \notin A$ the transition moves independently, see inference rule (A) and (B). Additionally, Markovian transitions always move independently if they are not inducing a self-loop, see inference rule (D) and (E). Otherwise, inference rule (F) takes effect. Consider two states $s_1 \in S_1$ and $s_2 \in S_2$ with $s_1 \xrightarrow{\lambda} s_1$ and $s_2 \xrightarrow{\lambda} s_2$. If self-loops would be allowed in inference rule (D) and (E), the resulting parallel composition would only have the Markovian transition $(s_1, s_2) \xrightarrow{\lambda} (s_1, s_2)$. However, this would not reflect the behaviour that both systems would perform a self-loop with rate λ in parallel. However, with the self-loop restriction and inference rule (F) we obtain the correct behaviour with $(s_1, s_2) \xrightarrow{2\lambda} (s_1, s_2)$.

Now we can lift Definition 2.19 to MRAs. Therefore, we have to introduce the composition of state and transition rewards. The formal definition of the parallel composition is given below.

(A) $\frac{s_1 \xrightarrow{\alpha, \mu} r_1 \quad \alpha \in Act_1 \setminus A}{(s_1, s_2) \xrightarrow{\alpha, \mu} r_1}$	(B) $\frac{s_2 \xrightarrow{\alpha, \mu} r_2 \quad \alpha \in Act_2 \setminus A}{(s_1, s_2) \xrightarrow{\alpha, \mu} r_2}$
(C) $\frac{s_1 \xrightarrow{\alpha, \mu_1} r_1 \quad s_2 \xrightarrow{\alpha, \mu_2} r_2 \quad \alpha \in A}{(s_1, s_2) \xrightarrow{\alpha, \mu_1 \times \mu_2} r_1 + r_2}$	
(D) $\frac{s_1 \xrightarrow{\chi(E(s_1)), \mu} r_1 \quad MT(s_2) = \emptyset}{(s_1, s_2) \xrightarrow{\chi(E(s_1)), \mu} r_1}$	(E) $\frac{s_2 \xrightarrow{\chi(E(s_2)), \mu} r_2 \quad MT(s_1) = \emptyset}{(s_1, s_2) \xrightarrow{\chi(E(s_2)), \mu} r_2}$
(F) $\frac{s_1 \xrightarrow{\chi(E(s_1)), \mu_1} r_1 \quad s_2 \xrightarrow{\chi(E(s_2)), \mu_2} r_2}{(s_1, s_2) \xrightarrow{\chi(E(s_1) + E(s_2)), \mu_1 \times \mu_2} r_1 + r_2}$	

Table 2.4: Inference rules notation for transition rewards (if all conditions above the line of a rule hold, then so should the condition below).

Definition 2.20 (Parallel composition for MRAs). Let $\mathcal{M}_1, \mathcal{M}_2$ be two MRAs such that $\mathcal{M}_1 = \langle \mathcal{A}_1, \rho_1, r_1 \rangle$ and $\mathcal{M}_2 = \langle \mathcal{A}_2, \rho_2, r_2 \rangle$, and action set $A \subseteq Act_1^{ext} \cap Act_2^{ext}$. Their parallel composition w.r.t. set A of actions is the system $\mathcal{M}_1 \parallel_A \mathcal{M}_2 = \langle \mathcal{A}_1 \parallel_A \mathcal{A}_2, \rho, r \rangle$, with

- $\mathcal{A}_1 \parallel_A \mathcal{A}_2$ as in Definition 2.19;
- $\rho: S \rightarrow \mathbb{R}_{\geq 0}$ with $\rho(s_1, s_2) = \rho_1(s_1) + \rho_2(s_2)$;
- r is given by the smallest relation fulfilling the inference rules in Table 2.4.

The composition of the state reward is straightforward and sums up the state rewards of the composed states. The composition of the transition reward, however, is more complex. Let's have a closer look at the inference rules in Table 2.4. First recall that $s \xrightarrow{\alpha, \mu} r$ describes the transition reward function induced by the extended transition $s \xrightarrow{\alpha} \mu$. Let's consider two states $s_1 \in S_1$ and $s_2 \in S_2$ from $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. If both states s_1 and s_2 can take a probabilistic transition with shared external action α , then the sum of the transition rewards r_1 and r_2 will be the new transition reward, see inference rule (C). The same holds if both states induce a Markovian transition relation with extended actions $\chi(E(s_1))$ and $\chi(E(s_2))$, see inference rule (F). Otherwise, if the states are not synchronising over an action or only one state contains Markovian transitions, the individual transition rewards are carried over, see inference rules (A), (B), (D) and (E). Figure 2.6 depicts the parallel composition of two MRAs.

Example 2.8. Consider the MRAs $\mathcal{M}_1$ and $\mathcal{M}_2$ in Figure 2.6a and 2.6b, respectively. Note that both MRAs share action α . The parallel composition $\mathcal{M}_1 \parallel_{\{\alpha\}} \mathcal{M}_2$ is depicted in Figure 2.6c. Let's consider the initial state (s_0, t_0) . First of all, the new state reward $\rho(s_0, t_0) = \rho(s_0) + \rho(t_0) = 5$. Further, for state s_0 inference rule (A) for $s_0 \xrightarrow{\alpha} s_1$ and (F) for $s_0 \xrightarrow{\alpha} s_1$ from Table 2.3

Figure 2.6: Example of the parallel composition of two MRAs.

holds. Besides, inference rule (D) from 2.4 for the transition reward holds. However, state t_0 has only a shared action enabled and therefore wants to synchronise. Thus we obtain the Markovian transitions $(s_0, t_0) \xrightarrow{\mu} (s_1, t_0)$ with $r_{(s_0, t_0)}((s_1, t_0)) = 4$ and $(s_0, t_0) \xrightarrow{\lambda} (s_0, t_0)$ with $r_{(s_0, t_0)}((s_0, t_0)) = 2$. Continuing with state (s_1, t_0) both states s_1 and t_0 have a probabilistic transition with shared action α enabled. Hence, those probabilistic transitions will synchronise according to inference rule (C) of Table 2.3 and 2.4 and create the probabilistic transition $(s_1, t_0) \xrightarrow{\alpha} \mu$ in $\mathcal{M}_1 \parallel_{\{\alpha\}} \mathcal{M}_2$ with

$$\mu = \{(s_0, t_0) \mapsto 0.125, (s_1, t_0) \mapsto 0.125, (s_1, t_1) \mapsto 0.375, (s_0, t_1) \mapsto 0.375\}$$

and

$$r_{(s_1, t_0)} = \{(s_0, t_0) \mapsto 6, (s_1, t_0) \mapsto 3, (s_1, t_1) \mapsto 6, (s_0, t_1) \mapsto 9\}.$$

By carefully comparing the new probabilistic transition with the transitions in $\mathcal{M}_1$ and $\mathcal{M}_2$ we can see that $\mu = \mu_1 \times \mu_2$ and $r_{(s_1, t_0)} = r_{s_1} + r_{t_0}$. For state (s_1, t_1) and (s_0, t_1) no synchronisation happens and therefore the transitions are equivalent to the transitions in $\mathcal{M}_1$ and $\mathcal{M}_2$.

2.6.1 Uniformisation

The uniformisation of CTMCs is a well known method in the computation of transient probabilities [BHHK00]. The concept of the uniformisation step can also be translated to MRAs. First we define what a uniform MRA is and then how to transform a non uniform MRA into its corresponding uniform version.

Definition 2.21 (Uniform MRA). Let $\mathcal{M} = (\mathcal{A}, \rho, r)$ be an MRA. $\mathcal{M}$ is uniform iff $E(s) = e$ for all $s \in MS$ and some $e \in \mathbb{R}_{>0}$.

Now if the MRA is not uniform, one can apply a uniformisation. That is, all exit rates in the MRA are adjusted such that Definition 2.22 holds.

Definition 2.22 (Uniformisation). Let $\mathcal{M} = (\mathcal{A}, \rho, r)$ be finite. Let $e \in \mathbb{R}_{>0}$ such that $e \geq \max_{s \in MS} E(s)$. Then the uniform MRA is given by $unif(e, \mathcal{M})$ such that for all $s \in MS$ with $E(s) < e$ we add a Markovian self-loop (s, λ, s) with $\lambda = e - E(s)$ and $r(s, \lambda, \mu(s), s) = 0$.

Usually, the uniformisation rate e is chosen as the shortest mean residence time in MRA $\mathcal{M}$. By applying Definition 2.22 all rates will get normalised according to e . This has the effect that the probability distributions are altered in the following way:

$$\begin{aligned} \bar{\mathbb{P}}_s(s') &= \frac{E(s)}{e} \cdot \mathbb{P}_s(s') & \text{if } s \neq s' \\ \bar{\mathbb{P}}_s(s) &= \frac{E(s)}{e} \cdot \mathbb{P}_s(s) + 1 - \frac{E(s)}{e} & \text{otherwise.} \end{aligned}$$

2.7 Bisimulations

Equivalence relations. An *equivalence relation* is a binary relation $\mathcal{R}$ on a set X such that it is (a) *reflexive*, i.e. for every $x \in X$ it holds that $(x, x) \in \mathcal{R}$; (b) *symmetric*, i.e. $(x, y) \in \mathcal{R}$ implies $(y, x) \in \mathcal{R}$, and (c) *transitive*, i.e. if $(x, y) \in \mathcal{R}$ and $(y, z) \in \mathcal{R}$ then $(x, z) \in \mathcal{R}$.

Given an equivalence relation $\mathcal{R} \subseteq X \times X$, we write $[x]_{\mathcal{R}}$ for the *equivalence class* induced by x , formally $[x]_{\mathcal{R}} = \{y \in X \mid (x, y) \in \mathcal{R}\}$. The set of all such equivalent classes is denoted by $X/\mathcal{R}$.

Given two probability distributions μ, μ' over a set S and an equivalence relation $\mathcal{R}$ we write $\mu \equiv_{\mathcal{R}} \mu'$ to denote that μ and μ' assign the same probability to each equivalence class of S under $\mathcal{R}$:

$$\forall s \in S. \mu([s]_{\mathcal{R}}) = \mu'([s]_{\mathcal{R}}).$$

Obviously, $\equiv_{\mathcal{R}}$ is an equivalence relation itself.

2.7.1 Strong bisimulation

We define a notion of strong bisimulation for MRAs. As for LTSs, PAs, IMCs and MAs, it equates systems that are equivalent in the sense that every step of one system can be mimicked by the other, and vice versa.

Definition 2.23 (Strong bisimulation). Given an MRA $\mathcal{M} = \langle \mathcal{A}, \rho, r \rangle$, an equivalence relation $\mathcal{R} \subseteq S \times S$ is a *strong bisimulation* for $\mathcal{M}$ if for every $(s_1, s_2) \in \mathcal{R}$ and for all $\alpha \in A^\chi, \mu \in \text{Distr}(S)$, it holds that $\rho(s_1) = \rho(s_2)$ and

$$s_1 \xrightarrow{\alpha} \mu \implies \exists \mu' \in \text{Distr}(S) . s_2 \xrightarrow{\alpha} \mu' \wedge \mu \equiv_{\mathcal{R}} \mu' \\ \wedge \forall s \in S . r([s_1]_{\mathcal{R}}, \alpha, [s]_{\mathcal{R}}) = r([s_2]_{\mathcal{R}}, \alpha, [s]_{\mathcal{R}})$$

Two states $s_1, s_2 \in S$ are *strongly bisimilar* (denoted by $s_1 \approx s_2$) if there exists a strong bisimulation R for $\mathcal{M}$ such that $(s_1, s_2) \in R$. Two MRAs $\mathcal{M}, \mathcal{M}'$ are strongly bisimilar (denoted by $\mathcal{M} \approx \mathcal{M}'$) if their initial states are strongly bisimilar in their disjoint union.

Clearly, when setting all state-based and action-based rewards to 0, MRAs coincide with MAs. Additionally, our definition of strong bisimulation then reduces to the definition of strong bisimulation for MAs. Since it was already shown in [EHZ10b] that strong bisimulation for MAs coincides with the corresponding notions for all subclasses of MAs, this also holds for our definition. Hence, it safely generalises the existing notions of strong bisimulation. Moreover, as stated in [EHZ10b], strong bisimulation is a congruence for parallel composition.

Proposition 2.1 (Congruence). Let $\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}_3$ be MRAs such that $\mathcal{M}_1 \approx \mathcal{M}_2$, then

$$\mathcal{M}_1 \parallel \mathcal{M}_3 \approx \mathcal{M}_2 \parallel \mathcal{M}_3.$$

It is easy to see that the validity this proposition is not influenced by the introduction of the additional check for equal state rewards as well as equal transition rewards in our notion of strong bisimulation.

2.7.2 Weak bisimulation

In the following we introduce the definitions for *naive weak bisimulation* and *weak bisimulation* over distributions for MAs [EHZ10a] and propose a lifting to MRAs. Therefore, we first extend the concept of *transition trees* with rewards and then adapt the *naive weak bisimulation* and *weak bisimulation* with rewards for MRAs.

2.7.2.1 Weak transitions

Definition 2.24 (*L-labelled tree*). For $\sigma, \sigma' \in \mathbb{N}_{>0}^*$, let $\sigma \leq \sigma'$ if there exists a (possibly empty) $\Phi \in \mathbb{N}_{>0}^*$ such that $\sigma \circ \Phi = \sigma'$, where $\circ$ is the concatenation. Moreover, let $\sigma < \sigma'$ if $\sigma \leq \sigma'$ and $\sigma \neq \sigma'$. Let L be a (possibly uncountable) set. A partial function $\mathcal{T}: \mathbb{N}_{>0}^* \rightarrow L$, which satisfies

- if $\sigma \leq \sigma'$ and $\sigma' \in \text{dom}(\mathcal{T})$ then $\sigma \in \text{dom}(\mathcal{T})$
- if $\sigma \circ i \in \text{dom}(\mathcal{T})$ for $i > 1$, then also $\sigma \circ (i - 1) \in \text{dom}(\mathcal{T})$
- $\epsilon \in \text{dom}(\mathcal{T})$

is called an (infinite) *L-labelled tree*. The root of the tree $\mathcal{T}$ is called ϵ and $\sigma \in \text{dom}(\mathcal{T})$ is a node of $\mathcal{T}$. A node σ is called a leaf of $\mathcal{T}$ if there is no $\sigma' \in \text{dom}(\mathcal{T})$ such that $\sigma < \sigma'$.

We denote the set of children of a node $\sigma \in \text{dom}(\mathcal{T})$ by

$$\text{Children}(\sigma) = \{\sigma \circ i \mid \sigma \circ i \in \text{dom}(\mathcal{T})\}.$$

The set of all leaves of $\mathcal{T}$ is denoted by

$$\text{Leaf}_{\mathcal{T}} = \epsilon \cup \{\sigma \in \text{dom}(\mathcal{T}) \mid \text{Children}(\sigma) = \emptyset\}$$

and the set of all *inner nodes* of $\mathcal{T}$ by

$$\text{Inner}_{\mathcal{T}} = \{\sigma \in \text{dom}(\mathcal{T}) \mid \text{Children}(\sigma) \neq \emptyset\}.$$

Let $L = S \times \mathbb{R}_{\geq 0} \times (\text{Act}^X \cup \{\perp\}) \times \mathbb{R}_{\geq 0}$. Hence, a node in such an *L-labelled tree* corresponds to a state, the probability of reaching this node from the root, the chosen extended action to proceed (including a special bottom action $\perp$ for leaf nodes), and the accumulated reward. For a node σ we write $\text{Sta}_{\mathcal{T}}(\sigma)$ for the first component of $\mathcal{T}(\sigma)$ and $\text{Prob}_{\mathcal{T}}(\sigma)$ for the second component of $\mathcal{T}(\sigma)$, $\text{Act}_{\mathcal{T}}(\sigma)$ for the third component, and $\text{Rew}_{\mathcal{T}}(\sigma)$ for the fourth component.

Definition 2.25 (*Transition tree*). Let $\mathcal{M} = (\mathcal{A}, \rho, r)$ be an MRA. A *transition tree* $\mathcal{T}$ of $\mathcal{M}$ is a $(S \times \mathbb{R}_{\geq 0} \times (\text{Act}^X \cup \{\perp\}) \times \mathbb{R}_{\geq 0})$ -labelled tree that satisfies the following conditions:

1. $\text{Prob}_{\mathcal{T}}(\epsilon) = 1$,
2. $\forall \sigma \in \text{Leaf}_{\mathcal{T}} : \text{Act}_{\mathcal{T}}(\sigma) = \perp$,
3. $\forall \sigma \in \text{Inner}_{\mathcal{T}} \setminus \text{Leaf}_{\mathcal{T}} : \exists \mu : \text{Sta}_{\mathcal{T}}(\sigma) \xrightarrow{\text{Act}_{\mathcal{T}}(\sigma)} \mu$ and

$$\text{Prob}_{\mathcal{T}}(\sigma) \cdot \mu = \{(\text{Sta}_{\mathcal{T}}(\sigma'), \text{Prob}_{\mathcal{T}}(\sigma')) \mid \sigma' \in \text{Children}_{\mathcal{T}}(\sigma)\}$$

4. $\sum_{\sigma \in \text{Leaf}_{\mathcal{T}}} \text{Prob}(\sigma) = 1$,
5. $\text{Rew}_{\mathcal{T}}(\epsilon) = 0$,
6. $\forall \sigma, \sigma'. \sigma' \in \text{Children}_{\mathcal{T}}(\sigma)$
 $\Rightarrow \text{Rew}_{\mathcal{T}}(\sigma') = \text{Rew}_{\mathcal{T}}(\sigma) + r(\text{Sta}_{\mathcal{T}}(\sigma), \text{Act}_{\mathcal{T}}(\sigma), \text{Sta}_{\mathcal{T}}(\sigma')) \cdot \text{Prob}_{\mathcal{T}}(\sigma')$.

Figure 2.7: Example of an MRA and a corresponding transition tree.

By restricting Act^X to Act , a transition tree $\mathcal{T}$ corresponds to a *probabilistic execution fragment*. That is, it starts from $Sta_{\mathcal{T}}(\epsilon)$, and resolves the non-deterministic choices at every inner node of the tree with $Act_{\mathcal{T}}(\sigma)$. $Prob_{\mathcal{T}}(\sigma)$ is the probability of reaching a state $Sta_{\mathcal{T}}(\sigma)$ via immediate transitions in the MRA starting from state $Sta_{\mathcal{T}}(\epsilon)$ and $Rew_{\mathcal{T}}(\sigma)$ the accumulated transition reward w.r.t. the transition probability, respectively.

An *internal transition tree* $\mathcal{T}$ is a transition tree where each $Act_{\mathcal{T}}(\sigma)$ is either an internal action or $\perp$. The distribution associated with $\mathcal{T}$, denoted $\mu_{\mathcal{T}}$, is defined as

$$\mu_{\mathcal{T}} \stackrel{\text{def}}{=} \bigoplus_{\sigma \in \text{Leaf}_{\mathcal{T}}} \{(\text{Sta}_{\mathcal{T}}(\sigma), \text{Prob}_{\mathcal{T}}(\sigma)), \quad (2.1)$$

where $\bigoplus$ is the direct sum over a set of tuples. Further, we define the transition reward function associated with $\mathcal{T}$ as

$$r_{\mathcal{T}} \stackrel{\text{def}}{=} \bigoplus_{\sigma \in \text{Leaf}_{\mathcal{T}}} \{(\text{Sta}_{\mathcal{T}}(\sigma), \text{Rew}_{\mathcal{T}}(\sigma)/\text{Prob}_{\mathcal{T}}(\sigma))\}. \quad (2.2)$$

With the above introduced definitions we now can define a *weak transition*.

Definition 2.26 (Weak transitions). For $s \in S$ and $\mu \in \text{Distr}(S)$, let $s \rightsquigarrow \mu$ be a *weak transition* and $s \rightsquigarrow r$ a *weak reward relation* if μ and r are induced by some internal transition tree $\mathcal{T}$ with $Sta_{\mathcal{T}}(\epsilon) = s$. Let $\mu \in \text{Distr}(S)$ and r be a transition reward relation. If for every state $s_i \in \text{supp}(\mu)$, $s_i \rightsquigarrow \mu'_i$ for some μ'_i and $s_i \rightsquigarrow r'_i$ for some r'_i , then we write

$$\mu \rightsquigarrow \bigoplus_{s_i \in \text{supp}(\mu)} \mu(s_i) \mu'_i,$$

and

$$r \rightsquigarrow \bigoplus_{s_i \in \text{supp}(\mu)} \mu(s_i) \cdot r(s_i) + r'_i \cdot (\mu(s_i) \mu'_i).$$

Example 2.9. Consider the MRA $\mathcal{M}$ in Figure 2.7a and its corresponding transition tree $\mathcal{T}$ in Figure 2.7b. By inspecting $\mathcal{T}$ and each node, one can observe that each node contains the probability to reach the corresponding state from the root node as well as the gathered transition reward. For example $\langle s_3, \perp, \frac{1}{2}, 1\frac{1}{2} \rangle$ captures the time abstract path $s_0 \xrightarrow{\alpha, \mu_0} s_1 \xrightarrow{\alpha, \mu_1} s_3$ and the corresponding probability $\mu_0(s_1) \cdot \mu_1(s_3) = \frac{1}{2}$ and transition reward $\mu_0(s_1)r_{s_0}^\alpha(s_1) + \mu_0(s_1)\mu_1(s_3)r_{s_1}^\alpha(s_3) = 2\frac{1}{6}$. Note that each transition reward is weighted with the probability to reach the corresponding state.

Now we can create a *convex combination* of weak transitions and weak reward relations. Let

$$\mu \rightsquigarrow_C \gamma \text{ and } r \rightsquigarrow_C r'$$

be a convex combination of weak transitions and weak reward relations, respectively, such that:

- there exists a finite index set I ,
- $\mu \rightsquigarrow \gamma_i$ for every $i \in I$,
- $r \rightsquigarrow r'_i$ for every $i \in I$,
- with a factor $c_i, w_i \in (0, 1]$ such that $\sum_{i \in I} c_i = 1$ and $\sum_{i \in I} w_i = 1$, with
- $\gamma = \bigoplus_{i \in I} c_i \gamma_i$, and
- $r' = \bigoplus_{i \in I} w_i r'_i$.

Let the set of splittings of immediate successor sub-distributions be defined as

$$\text{split}(\mu) = \{(\mu_1, \mu_2) | \exists \mu' : \mu \rightsquigarrow_C \mu' \wedge \mu' = \mu_1 \oplus \mu_2\},$$

and the set of splittings of *weak reward relations* be defined as

$$\text{split}(r) = \{(r_1, r_2) | \exists r' : r \rightsquigarrow_C r' \wedge r' = r_1 \oplus r_2\}.$$

2.7.2.2 Weak bisimulation relations

Note that for an internal state s , i.e. a state where only internal actions are enabled such that $\text{Act}(s) \subseteq \text{Act}^{int}$, the fraction of time we spend in this state is 0 since an internal action will be triggered immediately. Hence, the accumulated state reward in such a state will be always 0. Therefore, we allow for weak bisimulations that the state reward for internal states does not have to match. Lets denote with $IS = \{s \in S \mid \text{Act}(s) \subseteq \text{Act}^{int}\}$ the set of internal states in an MRA $\mathcal{M}$.

Definition 2.27 (Naive weak bisimulation). A symmetric relation $\mathcal{R}$ over S is called a *naive weak bisimulation* if and only if whenever $s_1 \mathcal{R} s_2$ then $\rho(s_1) = \rho(s_2)$ if $s_1, s_2 \in S \setminus IS$ and for all $\alpha \in \text{Act}^X$:

- $s_1 \xrightarrow{\alpha} \mu$ implies $s_2 \rightsquigarrow_C^\alpha \mu'$ with $\mu(C) = \mu'(C)$ for all $C \in S/R$, and
- $s_1 \xrightarrow{\alpha} r$ implies $s_2 \rightsquigarrow_C^\alpha r'$ with $r_{s_1}(C) = r'_{s_2}(C)$ for all $C \in S/R$.

Note that we extended the naive weak bisimulation as presented in [EHZ10a] with the notion of rewards. That is, if two states are in a relation, also their state rewards have to be equivalent. However, for internal states $s \in IS$ the state reward can be disregarded. Besides, the weak transition relation as well as the weak reward relation have to be equivalent. This is a conservative extension of the strong bisimulation as presented in Definition 2.23. A coarser weak bisimulation definition is defined over sub-distributions instead of states. We will first define the weak bisimulation over MAs, and then extend it to MRAs.

Definition 2.28 (Weak bisimulation). A symmetric relation $\mathcal{R}$ on sub-distributions over S is called a weak bisimulation if and only if whenever $\mu_1 \mathcal{R} \mu_2$ then for all $\alpha \in Act^X$: $|\mu_1| = |\mu_2|$ and

1. for all $s \in \text{supp}(\mu_1)$ there exists $(\mu_2^{\rightarrow}, \mu_2^{\Delta}) \in \text{split}(\mu_2)$ and
 - (a) $\mu_1(s) \mathbb{1}_s \mathcal{R} \mu_2^{\rightarrow}$ and $(\mu_1 \ominus s) \mathcal{R} \mu_2^{\Delta}$
 - (b) whenever $s \xrightarrow{\alpha} \mu'_1$ for some μ'_1 then $\mu_2^{\rightarrow} \rightsquigarrow_C^{\alpha} \mu''$ and $(\mu_1(s) \cdot \mu'_1) \mathcal{R} \mu''$
2. for all $s \in \text{supp}(\mu_2)$ there exists $(\mu_1^{\rightarrow}, \mu_1^{\Delta}) \in \text{split}(\mu_1)$ and
 - (a) $\mu_2(s) \mathbb{1}_s \mathcal{R} \mu_1^{\rightarrow}$ and $(\mu_2 \ominus s) \mathcal{R} \mu_1^{\Delta}$
 - (b) whenever $s \xrightarrow{\alpha} \mu'_2$ for some μ'_2 then $\mu_1^{\rightarrow} \rightsquigarrow_C^{\alpha} \mu''$ and $(\mu_2(s) \cdot \mu'_2) \mathcal{R} \mu''$

Two sub-distributions μ and γ are weak bisimilar, denoted $\mu \approx \gamma$, if the pair (μ, γ) is contained in some weak bisimulation. MA $\mathcal{A}_1, \mathcal{A}_2$ are weak bisimilar, denoted $\mathcal{A}_1 \approx \mathcal{A}_2$, if their initial (Dirac) distributions are bisimilar in the direct sum.

Note that the weak bisimulation is defined over distributions instead of states. For MRAs we also have to define equivalence relations w.r.t. state and transition rewards. However, first have a closer look at Definition 2.28. First off all, two sub-distributions can only be bisimilar if they are of the same size. If this is the case, the two almost symmetric conditions (1) and (2) have to hold. Let's examine condition (1). Note that the behaviour of a sub-distribution is defined over the states in its support and their respective probabilities. Now for all states $s \in \text{supp}(\mu_1)$, μ_2 can only reach a sub-distribution over weak internal transitions, such that μ_2 can be split into two sub-distributions $\mu_2^{\rightarrow}$ and μ_2^{Δ} . Condition (a) says now that $\mu_2^{\rightarrow}$ behaves bisimilar to $\mu_1(s)$ and μ_2^{Δ} behaves bisimilar to the remainder of the distribution $\mu_1 \ominus s$ by neglecting state s . Condition (b) is then the usual bisimulation condition. Thus, whenever s is reaching a distribution μ'_1 over an action $\alpha \in Act^X$ then there exists a convex combination of weak transitions from μ_2^{Δ} to a sub-distribution μ'' , such that μ'' is bisimilar to $\mu_1(s) \mu'_1$. Condition (2) is then analogous.

To include reward conditions we adapt the weak bisimulation in the following way. For state rewards, we require that the state reward is equivalent for all states that are not internal. Note that an internal state only has τ -transitions and therefore the sojourn time will be 0. Hence, the state reward over time will be 0 despite the assigned state reward. Besides, we have to relate the transition rewards. This is equivalent to the relation of the probability distributions.

Figure 2.8: Example of two weak reward bisimilar MRAs.

However, to relate the transition rewards with each other, we have to include the respective sub-distributions. In the following we propose such an extension for a *weak reward bisimulation* of MRAs.

Definition 2.29 (Weak reward bisimulation). A symmetric relation $\mathcal{R}$ on sub-distributions over S is called a *weak reward bisimulation* if and only if whenever $\mu_1 \mathcal{R} \mu_2$ as defined in Definition 2.28 with $s_1 \xrightarrow{\alpha} \mu_1$ and $s_2 \xrightarrow{\alpha} \mu_2$ it holds that $\rho(s_1) = \rho(s_2)$ if $s_1, s_2 \in S \setminus IS$ and $r_1 \mathcal{R} r_2$ such that

1. for all $s \in \text{supp}(\mu_1)$ there exists $(r_2^{\rightarrow}, r_2^{\Delta}) \in \text{split}(r_2)$ and

(a) $r_1(s) \mathcal{R} r_2^{\rightarrow}$ and $(r_1 \ominus s) \mathcal{R} r_2^{\Delta}$

- (b) whenever $s \xrightarrow{\alpha} r'_1$ for some r'_1 then $r_2^{\rightarrow} \sim_C^{\alpha} r''$ and

$$\frac{r_1(s) \cdot \mu_1(s) \oplus r'_1 \cdot (\mu'_1 \mu_1(s))}{\mu'_1 \mu_1(s)} \mathcal{R} r''$$

2. an according condition for all $s \in \text{supp}(\mu_2)$ must hold.

Two sub-distributions μ and γ are weak reward bisimilar, denoted $\mu \approx_r \gamma$, if the pair (μ, γ) is contained in some weak reward bisimulation. MRA $\mathcal{M}_1, \mathcal{M}_2$ are weak reward bisimilar, denoted $\mathcal{M}_1 \approx_r \mathcal{M}_2$, if their initial (Dirac) distributions are bisimilar in the direct sum.

Note that the second condition for the relation between the transition rewards reflects the reward function as defined for the transition tree in Equation (2.2).

Example 2.10. Consider the two MRAs $\mathcal{M}_1$ and $\mathcal{M}_2$ in Figure 2.8. We will show that $\mathcal{M}_a \approx_r \mathcal{M}_b$. Thus, $\mu_{s_0} \approx_r \mu_{t_0}$ with

$$\begin{aligned} \mu_{s_0} &= \{(E, \tfrac{1}{2}), (s_2, \tfrac{1}{2})\} & \mu_{t_0} &= \{(E, \tfrac{1}{2}), (F, \tfrac{1}{3}), (G, \tfrac{1}{6})\} \\ r_{s_0} &= \{(E, 2), (s_2, 3)\} & r_{t_0} &= \{(E, 2), (F, 8.5), (G, 10)\} \end{aligned}$$

such that $\mu_{s_0} \mathcal{R} \mu_{t_0}$ and $r_{s_0} \mathcal{R} r_{t_0}$. We assume E, F and G are bisimilar to them-

selves. Lets consider the case for $s_0 \in \text{supp}(\mu_0)$. Now we split μ_{t_0} and r_{t_0} into

$$\begin{aligned}\mu_{t_0}^{\rightarrow} &= \{(F, \frac{1}{3}), (G, \frac{1}{6})\} & \mu_{t_0}^{\Delta} &= \{(E, \frac{1}{2})\} \\ r_{t_0}^{\rightarrow} &= \{(F, 8.5), (G, 10)\} & r_{t_0}^{\Delta} &= \{(E, 2)\}\end{aligned}$$

It is easy to see that $\mu_{s_0}(E) = \frac{1}{2}\mathcal{R}\mu_{t_0}^{\Delta}$ and $r_{s_0}(E) = 2\mathcal{R}r_{t_0}^{\Delta}$. The interesting part is to show $\mu_{s_0} \ominus (E) = \{(s_1, \frac{1}{2})\}\mathcal{R}\mu_{t_0}^{\rightarrow}$ and $r_{s_0} \ominus (E) = \{(s_1, 3)\}\mathcal{R}r_{t_0}^{\rightarrow}$, such that $r_{s_0}(s_1) = 3$. Now lets consider the transition $s_1 \xrightarrow{\tau} \mu'$ with $\mu' = \{(F, \frac{2}{3}), (G, \frac{1}{3})\}$ with $r'_{s_1} = \{(F, 4), (G, 1)\}$. Therefore, we will have a look at condition (b). One can directly see that $\mu_{s_0}(s_1)\mu' = \{(F, \frac{1}{3}), (G, \frac{1}{6})\}\mathcal{R}\mu_{t_0}^{\rightarrow}$. Moreover, for r_{s_1} it holds that

$$\frac{r_{s_0}(s_1) \cdot \mu_{s_0}(s_1) \oplus \mu_{s_0}(s_1)\mu'}{\mu_{s_0}(s_1)\mu'} = \{(F, 8.5), (G, 10)\}\mathcal{R}r_{t_0}^{\rightarrow}.$$

It is now routine to check the remaining conditions.

2.8 Conclusion

In this chapter we have presented Markov reward automata (MRA), an extension to Markov automata with state and transition rewards. Besides, we have presented several behavioural notions on MAs that are inherited by MRAs. For the resolution of the non-determinism in MRAs we presented a variety of measurable schedulers. Further, we lifted the parallel composition of MAs to MRAs. Finally, we proposed extensions of the strong and weak bisimulation of MAs to MRAs.

Analysis of expected reward properties

STOCHASTIC model checking is a state-of-the art analysis method for the verification of stochastic systems. It includes powerful techniques to analyse qualitative and quantitative properties over models with discrete probabilities and/or timing, as well as non-determinism. A plethora of powerful software tools such as PRISM [KNP11], MRMC [KZHHJ11], UPPAAL [Beh+06], NuSMV [Cim+02] and Spin [Hol04] are dedicated to model checking and have been applied in a wide range of applications. The traditional model checking approach is depicted in Figure 3.1. The basic idea is to construct a finite-state model of a system and to specify some properties in a logic. Those two ingredients are then input to the model checker which automatically tries to verify the validity of the properties on the model. In the case where all properties are satisfied by the model the process completes successfully, otherwise a counterexample is provided which helps to either improve the model or change the properties.

A typical field where stochastic model checking is crucial are cyber-physical systems (CPS), i.e. systems where mechanisms are controlled or monitored by computer based algorithms [Alu15]. CPSs are found among others in nuclear power plants, smart grids, medical monitoring, autonomous driving assistants, and avionics. An example property to model check in a CPS is whether the reliability of the safety mechanism lies over 99.99999999%.

This chapter focuses on the extension of analysis algorithms for expected rewards in MRAs. The idea to model check properties w.r.t. rewards — also interpretable as costs — typically arises when multiple objectives have to be verified. Think of a smart grid where the energy consumption should not pass a certain threshold, or a nuclear power plant which should operate safely, but also within a cost budget. We focus on analysing the cumulative expected reward until a property is fulfilled or a certain time point is reached as well as the expected long-run average reward that is gathered over a system’s runtime.

A challenge in solving such reward properties is that MRAs combine stochastic timing, probabilistic distributions and non-deterministic choices. Thus, we need algorithms that incorporate all these factors. Further, important for the applicability in model checking is how efficiently we are able to analyse the reward properties on MRAs. A key in defining efficient algorithms for the reward properties in MRAs is a stepwise reduction to classical numerical techniques, such as linear programming problems. By providing a way to reduce most parts

Figure 3.1: Overview of the model checking approach.

to well studied techniques, we are able to take advantage of already existing optimisations. We will show that the reward measures are suitable to be formulated as linear equations. In particular, the cumulative rewards can be reduced to Bellman equations, whereas the long-run average reward is a combination of an inequality system and reachability analysis. Those problems are well suited for value iteration algorithms as well as linear programming problems.

Origins of the chapter. This chapter is based on

- D. Guck, M. Timmer, H. Hafei, E. J. J. Ruijters, and M. I. A. Stoelinga. “Modelling and analysis of Markov reward automata”. In: *Proceedings of the 12th International Symposium on Automated Technology for Verification and Analysis (ATVA 2014), Sydney, NSW, Australia*. Vol. 8837. Lecture Notes in Computer Science. Springer Verlag

and focuses on the analysis of expected rewards for Markov reward automata. Further we extend this chapter by describing how to use the algorithms in the context of model checking.

Organisation of the chapter. We describe the reward objectives in Section 3.1 and describe a required preprocessing. In Section 3.2 we define the cumulative expected goal-bounded reward and the time-bounded variation followed by the long-run average reward in Section 3.3. Note that since we only discuss certain reachability properties, we first define our measures of interest in terms of random variables. In Section 3.4 we then explain how to express our reward properties using a logic and how to extend model checking to MRAs. Section 3.5 presents some case studies to show the applicability of the algorithms.

3.1 Quantitative analysis

A fundamental objective in quantitative analysis is the question of reachability. For example consider any system with a cost budget and a goal. Now it is

of interest to know the maximal cost of reaching that goal to validate if the cost budget is sufficient. Another important factor can be time. For example a system has to run for n hours. Now we want to know what is the maximal energy consumption during operation. Moreover, if the operation time of the system is indefinite it is of interest to know what the average costs are in the long-run. These properties can be represented by three common reward measures:

- (1) The expected cumulative reward until reaching a set of goal states,
- (2) the expected cumulative reward until a given time-bound, and
- (3) the long-run average reward.

Since we define these reward measures on MRAs, there may not be one optimal solution. This is due to the non-determinism within the model. Hence, depending on how the non-determinism is resolved different results are obtained. Thus, there may not exist one optimal solution, though we can give an interval by the minimal and maximal solution. Typical examples taking the minimum and maximum into account w.r.t. the reward measures are respectively: (1) to minimise the cumulative energy consumption of a mobile device; (2) to minimise the average maintenance cost of a railroad line over the first year of deployment; and (3) to maximise the yearly revenues of a data centre over a long time horizon.

Note that we will present the techniques w.r.t. the maximum solution. The minimum solution can be achieved analogously by arguing over the infimum. Moreover, we assume that the MRAs subject to analysis are closed, i.e. they are not subject to any further synchronisation.

3.1.1 Prepossessing

Throughout this chapter, we consider a fixed MRA $\mathcal{M}$ with finite state space S and goal states $G \subseteq S$. The set of goal states can be defined via the state labelling of $\mathcal{M}$. To facilitate the algorithms, we first perform three prepossessing steps.

- (1) We consider only closed MRAs, which are not subject to further interaction. Therefore, we hide all actions as described in Definition 2.12, focusing on their induced rewards.
- (2) Due to the maximal progress assumption, a Markovian transition will never be executed from a state with outgoing τ -transitions. Hence, we remove such Markovian transitions by using the extended action set as described in Definition 2.4.
- (3) To distinguish action non-determinism in a state $s \in PS$, we assume w.l.o.g. that the corresponding actions are numbered from 1 to n_s where n_s is the number of outgoing transitions with action α . We write $\mu_s^{\alpha_i}$ for the distribution induced by taking α_i in state s and we write $r_s^{\alpha_i}(s') = r(s, \alpha, \mu_s^{\alpha_i}, s')$ for the reward. For Markovian states we write $\mathbb{P}_s$ and $r_s(s')$, respectively.

Figure 3.2: An example transformation for action non-determinism.

Example 3.1. Consider the MRA $\mathcal{M}$ depicted in Figure 3.2. For s_0 it holds that $\deg(s_0, \alpha) = 3$. Therefore we extend Act^{ext} with the actions α_1, α_2 and α_3 and introduce the corresponding probability distributions and reward functions as follows:

$$\begin{aligned} \mu_{s_0}^{\alpha_1} &= \{s_1 \mapsto 0.3, s_2 \mapsto 0.7\} & r_{s_0}^{\alpha_1} &= \{s_1 \mapsto 3, s_2 \mapsto 2\} \\ \mu_{s_0}^{\alpha_2} &= \{s_1 \mapsto 1\} & r_{s_0}^{\alpha_2} &= \{s_1 \mapsto 4\} \\ \mu_{s_0}^{\alpha_3} &= \{s_2 \mapsto 0.4, s_3 \mapsto 0.6\} & r_{s_0}^{\alpha_3} &= \{s_2 \mapsto 1, s_3 \mapsto 3\}. \end{aligned}$$

3.1.1.1 Stochastic shortest path problem

Throughout the chapter we will use the non-negative stochastic shortest path (SSP) problem for MDPs. An SSP problem derives the expected cost to reach a set of goal states in an MDP [BT91]. SSPs can be solved efficiently and in the remainder of the chapter we will show that parts of the reward analysis for MRAs will boil down to SSP problems.

Definition 3.1 (SSP problem). A non-negative stochastic shortest path problem (SSP problem) is a tuple $\text{ssp} = (S, Act, \mathbb{P}, s^0, c, g)$ where

- $(S, Act, \mathbb{P}, s^0)$ is an MDP,
- $G \subseteq S$ is a set of goal states,
- $c : S \times Act \rightarrow \mathbb{R}_{\geq 0}$ is a cost function, and
- $g : G \rightarrow \mathbb{R}_{\geq 0}$ is a terminal cost function.

Recall that a path through an MDP is an alternating sequence $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \dots$ such that $\mathbb{P}(s_i, \alpha_i, s_{i+1}) > 0$, for all i . The accumulated cost along a path π through the MDP before reaching G , denoted by $C_G(\pi)$, is $\sum_{j=0}^{k-1} c(s_j, \alpha_j) + g(s_k)$ for all $i < k$ with $s_i \notin G$ and $s_k \in G$. If π does not reach G , then $C_G(\pi)$ equals ∞ .

As standard in MDPs [Put94], non-determinism between different actions in a state is resolved using schedulers; similar to the notion for MRAs, a stationary deterministic scheduler is a function $D: PS \rightarrow Act$. Let $cR^{\min}(s, \Diamond G)$ denote the minimum expected cost reachability of G in the SSP (under all schedulers) when starting from s . It is a well-known result that stationary schedulers suffice to achieve $cR^{\min}(s, \Diamond G)$ [BT91; Alf97]. Analogously it holds for the maximum expected cost $cR^{\max}(s, \Diamond G)$.

3.2 Cumulative rewards

3.2.1 Goal-bounded reward

We are interested in the minimal and maximal expected cumulative reward until reaching a set of goal states $G \subseteq S$. That is, we accumulate the state and transition rewards until a state in G is reached; if no state in G is reached, then we keep on accumulating rewards, resulting in a potentially infinite reward.

Definition 3.2 (Expected reward). Let $\mathcal{M} = (\mathcal{A}, \rho, r)$ be an MRA and $G \subseteq S$ a set of goal states. The random variable $V_G: Paths \rightarrow \mathbb{R}_{\geq 0}^{\infty}$ yields the accumulated reward before first visiting some state in G . For an infinite path $\pi \in Paths^{\omega}(\mathcal{M})$, we define

$$V_G(\pi) = \begin{cases} \text{reward}(\pi^j) & \text{if } \pi[j] \in G \wedge \forall i < j. \pi[i] \notin G \\ \text{reward}(\pi) & \text{if } \forall i. \pi[i] \notin G \end{cases}$$

such that the expected reward under a scheduler D is given by

$$\text{eR}^D(s, G) = \mathbb{E}_{s,D}(V_G) = \int_{Paths} V_G(\pi) \Pr_{s,D}(d\pi)$$

The maximal expected reward to reach G from $s \in S$ is then defined as

$$\text{eR}^{\max}(s, G) = \sup_{D \in GM} \mathbb{E}_{s,D}(V_G) = \sup_{D \in GM} \int_{Paths} V_G(\pi) \Pr_{s,D}(d\pi). \quad (3.1)$$

where D is an arbitrary scheduler on $\mathcal{M}$.

To compute $\text{eR}^{\max}$ we turn Equation (3.1) into a classical Bellman equation: For all goal states, no more reward is accumulated, so their expected reward is zero. For Markovian states $s \notin G$, the state reward of s is weighted with the expected sojourn time in s plus the expected reward accumulated via its successor states plus the transition reward to them. For a probabilistic state $s \notin G$, we select the action that maximises the expected cumulative reward. Note that, since the accumulated reward is only relevant until reaching a state in G , we may turn all states in G into absorbing Markovian states.

Figure 3.3: An example MRA.

Theorem 3.1 (Bellman equation). *The function $\mathbf{eR}^{\max}: S \rightarrow \mathbb{R}_{\geq 0}^{\infty}$ is the unique fixed point of the Bellman equation*

$$v(s) = \begin{cases} \frac{\rho(s)}{E(s)} + \sum_{s' \in S} \mathbb{P}_s(s') \cdot (r(s, \chi(E(s)), s') + v(s')) & \text{if } s \in MS \setminus G \\ \max_{\alpha \in Act(s)} \sum_{s' \in S} \mu_s^\alpha(s') \cdot (r(s, \alpha, s') + v(s')) & \text{if } s \in PS \setminus G \\ 0 & \text{if } s \in G. \end{cases}$$

A direct consequence of Theorem 3.1 is that the supremum in Equation (3.1) is attained by a stationary deterministic scheduler, as later shown in Theorem 3.2. Moreover, this result enables us to use standard solution techniques such as value iteration and linear programming to compute $\mathbf{eR}^{\max}(s, G)$. Note that by assigning $\rho(s) = 1$ to all $s \in MS$ and setting all other rewards to 0, we compute the expected time to reach a set of goal states. We denote the expected time to reach a set of goal states G under a scheduler D by $\mathbf{eT}^D(s, G)$.

Example 3.2. Consider the MRA $\mathcal{M}$ depicted in Figure 3.3. Lets define the set of goal states as $G = \{s_3\}$ and our initial state $s^0 = s_0$. Using Theorem 3.1 we can build a linear programming problem for the maximal expected goal-bounded reward as follows: Minimise $v(s_0) + v(s_1) + v(s_3) + v(s_4)$ according to

$$\begin{aligned} v(s_0) &\geq \frac{1}{5} + \frac{3}{5}v(s_1) + \frac{2}{5}v(s_2) \\ v(s_1) &\geq \frac{2}{3} + \frac{2}{3}v(s_2) + \frac{1}{3}v(s_3) \\ v(s_2) &\geq 0.7 \cdot \left(\frac{40}{7} + v(s_1)\right) + 0.3 \cdot \left(\frac{40}{3} + v(s_4)\right) \\ v(s_2) &\geq 0.8 \cdot \left(\frac{5}{4} + v(s_0)\right) + 0.2 \cdot (5 + v(s_4)) \\ v(s_3) &= 0 \\ v(s_4) &\geq \frac{1}{2} + v(s_3) \end{aligned}$$

where $v(s_i) \geq 0$ for $i = 1 \dots 4$. By solving the equations we obtain:

$$v(s_0) = 13.525 \quad v(s_1) = 11.4375 \quad v(s_2) = 16.1562 \quad v(s_3) = 0 \quad v(s_4) = 0.5$$

Thus, the maximal expected reward in $\mathcal{M}$ starting from s_0 and reaching G is given by $\mathbf{eR}^{\max}(s_0, G) = 13.525$.

Proof of Theorem 3.1. We show that Theorem 3.1 and Equation 3.1 coincide. We will distinguish three cases: $s \in G$, $s \in MS \setminus G$, and $s \in PS \setminus G$.

Let $s \in G$ and $\mathbf{eR}^{\max}(s, G)$ as defined in Equation (3.1). Now for every π starting in s we have $V_G(\pi) = \text{reward}(\pi^0) = 0$. Note, that since we are starting in a state from the set of goal states, no time will pass. Thus we have:

$$\mathbf{eR}^{\max}(s, G) = 0 = v(s).$$

Let $s \in MS \setminus G$ and $\mathbf{eR}^{\max}(s, G)$ as defined in Equation (3.1). Thus we have:

$$\begin{aligned} \mathbf{eR}^{\max}(s, G) &= \sup_{D \in GM} \int_{Paths} \text{reward}(\pi) \cdot \Pr_{s,D}(\mathrm{d}\pi) \\ &= \sup_{D \in GM} \int_{Paths} \left(\sum_{i=0}^{|\pi|-1} \rho(\pi[i]) \cdot t_i \right. \\ &\quad \left. + r(\pi[i], \alpha_i, \pi[i+1]) \right) \cdot \Pr_{s,D}(\mathrm{d}\pi) \end{aligned}$$

Now we can split the reward summation into the reward gained in $s = s_0$ and the rest of the path π . Further, we include the path probability into the equation and thus change the interval over all paths to an interval over the time horizon $[0, \infty]$:

$$\begin{aligned} &= \sup_{D \in GM} \int_0^\infty \rho(s) \cdot t \cdot E(s) \cdot e^{-E(s)t} + \sum_{s' \in S} \mathbb{P}_s(s') \cdot r(s, \chi(E(s)), s') \\ &\quad + \sum_{s' \in S} \mathbb{P}_s(s') \cdot \mathbb{E}_{s', D[s \xrightarrow{\chi(E(s)), \mathbb{P}_s(\cdot), t} s']} (V_G) \, \mathrm{d}t \end{aligned}$$

Note that $D[s \xrightarrow{\chi(E(s)), \mathbb{P}_s(\cdot), t} s']$ is the scheduler that resolves non-determinism for path π' starting from s' as D does it for $s \xrightarrow{\chi(E(s)), \mathbb{P}_s(\cdot), t} \pi'$, i.e.

$$D(s \xrightarrow{\chi(E(s)), \mathbb{P}_s(\cdot), t} \pi') = D[s \xrightarrow{\chi(E(s)), \mathbb{P}_s(\cdot), t} s'](\pi').$$

By solving the integral and applying Equation (3.1) we obtain

$$\begin{aligned} &= \frac{\rho(s)}{E(s)} + \sum_{s' \in S} \mathbb{P}_s(s') \cdot r(s, \chi(E(s)), s') + \sup_{D \in GM} \sum_{s' \in S} \mathbb{P}_s(s') \cdot \mathbb{E}_{s', D}(V_G) \\ &= \frac{\rho(s)}{E(s)} + \sum_{s' \in S} \mathbb{P}_s(s') \cdot (\mathbf{eR}^{\max}(s', G) + r(s, \chi(E(s)), s')). \end{aligned}$$

Hence, $v(s)$ satisfies Theorem 3.1 for $s \in MS$.

Let $s \in PS \setminus G$ and $\mathbf{eR}^{\max}(s, G)$ as defined in Equation (3.1). Note that if we start in a state $s \in PS$ no time will pass until we reach a state $s \in MS$. Thus in the first step only the scheduler decision is important. Therefore we can split the equation as follows:

$$\begin{aligned} \mathbf{eR}^{\max}(s, G) = & \sup_{D \in GM} \sum_{s \xrightarrow{\alpha, \mu_s^\alpha, 0} s'} D(s)(\alpha, \mu) \cdot \mu_s^\alpha(s') \\ & \cdot \left(\mathbb{E}_{s, D[s \xrightarrow{\alpha, \mu_s^\alpha, 0} s']} (V_G) + r(s, \alpha, s') \right) \end{aligned}$$

where $D[s \xrightarrow{\alpha, \mu_s^\alpha, 0} s']$ is the scheduler that resolves non-determinism for path π' starting from s' as D does it for $s \xrightarrow{\alpha, \mu_s^\alpha, 0} \pi'$, i.e. $D(s \xrightarrow{\alpha, \mu_s^\alpha, 0} \pi') = D[s \xrightarrow{\alpha, \mu_s^\alpha, 0} s'](\pi')$. Each action $\alpha \in Act(s)$ uniquely determines a distribution μ_s^α , such that the successor state s' , with $s \xrightarrow{\alpha, \mu_s^\alpha, 0} s'$, satisfies $\mu_s^\alpha(s') > 0$:

$$\alpha^* = \arg \max \left\{ \sup_{D \in GM} \sum_{s' \in S} \mu_s^\alpha(s') \cdot \mathbb{E}_{s', D}(V_G) \mid \alpha \in Act(s) \right\}$$

Hence, all optimal policies choose α^* with probability 1, i.e. $D(s)(\alpha^*, \mu_s^{\alpha^*}) = 1$ and $D(s)(\beta, \mu_s^\beta) = 0$ for all $\beta \neq \alpha^*$. Thus, we obtain

$$\begin{aligned} &= \max_{s \xrightarrow{\alpha} \mu_s^\alpha} \sup_{D \in GM} \sum_{s' \in S} \mu_s^\alpha(s') \cdot (\mathbb{E}_{s', D}(V_G) + r(s, \alpha, s')) \\ &= \max_{\alpha \in Act(s)} \sum_{s' \in S} \mu_s^\alpha(s') \cdot (\mathbf{eR}^{\max}(s', G) + r(s, \alpha, s')). \end{aligned}$$

Hence, $v(s)$ satisfies Equation (3.1) for $s \in PS$. $\square$

Theorem 3.1 allows us to reduce the computation of the expected goal-bounded reward to a non-negative SSP problem [BT91]. Therefore, we now define a SSP problem as follows.

Definition 3.3 (SSP for expected goal-bounded reward). The SSP of MRA $\mathcal{M} = (\mathcal{A}, \rho, r)$ for the expected goal-bounded reward of $G \subseteq S$ is

$$\text{ssp}_{\mathbf{eR}}(\mathcal{M}) = (S, Act^X, \mathbb{P}, s^0, c, g)$$

where for all $s \in S \setminus G$ and $\sigma \in Act^X$:

$$\mathbb{P}(s, \sigma, s') = \begin{cases} \frac{\mathbf{R}(s, s')}{E(s)} & \text{if } s \in MS \setminus G \\ \mu_s^\sigma(s') & \text{if } s \in PS \setminus G \\ 0 & \text{otherwise,} \end{cases}$$

and

$$c(s, \sigma) = \begin{cases} \frac{\rho(s)}{E(s)} + \sum_{s' \in S} \frac{\mathbf{R}(s, s') \cdot r_s^\sigma(s')}{E(s)} & \text{if } s \in MS \setminus G \\ \sum_{s' \in S} \mu_s^\sigma(s') \cdot r_s^\sigma(s') & \text{if } s \in PS \setminus G \\ 0 & \text{otherwise,} \end{cases}$$

and $\mathbb{P}(s, \chi(E(s)), s) = 1$, $c(s, \chi(E(s))) = 0$, and $g(s) = 0$ for all $s \in G$.

Theorem 3.2. For MRA $\mathcal{M}$, $\text{eR}^{\max}(s, G)$ equals $cR^{\max}(s, \diamond G)$ in $\text{ssp}_{\text{eR}}(\mathcal{M})$.

Thus it follows that there exists a stationary deterministic scheduler on $\mathcal{M}$ yielding $\text{eR}^{\max}(s, G)$. Moreover, the uniqueness of the solution of an SSP [BT91; Alf97] yields that $\text{eR}^{\max}(s, G)$ is the unique fixed point of the Bellman function in Theorem 3.1.

Proof. As shown in [BT91; Alf97], $cR^{\max}(s, \diamond G)$ is the unique fixed point of the Bellman function defined as

$$v(s) = \max_{\alpha \in \text{Act}^X(s)} c(s, \alpha) + \sum_{s' \in S \setminus G} \mathbb{P}(s, \alpha, s') \cdot v(s') + \sum_{s' \in G} \mathbb{P}(s, \alpha, s') \cdot g(s').$$

We show that the Bellman function defined in Definition 3.2 equals v' for $\text{ssp}_{\text{eR}}(\mathcal{M})$. Note that by definition $g(s) = 0$, $c(s, \cdot) = 0$ and $\mathbb{P}(s, \chi(E(s)), s) = 1$ for all $s \in G$. Thus

$$v'(s) = \max_{\alpha \in \text{Act}^X(s)} c(s, \alpha) + \sum_{s' \in S} \mathbb{P}(s, \alpha, s') \cdot v(s').$$

We distinguish three cases, $s \in MS \setminus G$, $s \in PS \setminus G$, and $s \in G$.

- (i) If $s \in MS \setminus G$, then $\text{Act}^X(s) = \chi(E(s))$ and therefore $\max_{\alpha \in \text{Act}^X(s)} c(s, \alpha) = c(s, \chi(E(s)))$. Further for all $s' \in S$, $\mathbb{P}(s, \chi(E(s)), s') = \frac{\mathbf{R}(s, s')}{E(s)}$. Thus

$$\begin{aligned} v'(s) &= \frac{\rho(s)}{E(s)} + \sum_{s' \in S} \frac{\mathbf{R}(s, s') \cdot r_s^\sigma(s')}{E(s)} + \sum_{s' \in S} \frac{\mathbf{R}(s, s')}{E(s)} \cdot v'(s') \\ &= \frac{\rho(s)}{E(s)} + \sum_{s' \in S} \frac{\mathbf{R}(s, s')}{E(s)} \cdot (r_s^\sigma(s') + v'(s')). \end{aligned}$$

- (ii) If $s \in PS \setminus G$, for each action $\alpha \in \text{Act}^X(s)$ and successor state s' , with $\mathbb{P}(s, \alpha, s') > 0$ it follows that $\mathbb{P}(s, \alpha, s') = \mu_\alpha^s(s')$. Further, $c(s, \alpha) = \sum_{s' \in S} \mu_\alpha^s(s') \cdot r_s^\sigma(s')$ for all $\alpha \in \text{Act}$. Thus

$$\begin{aligned} v'(s) &= \max_{\alpha \in \text{Act}^X(s)} \sum_{s' \in S} \mu_\alpha^s(s') \cdot r_s^\sigma(s') + \sum_{s' \in S} \mathbb{P}(s, \alpha, s') \cdot v'(s') \\ &= \max_{\alpha \in \text{Act}(s)} \sum_{s' \in S} \mu_\alpha^s(s') \cdot (r(s, \alpha, s') + v'(s')). \end{aligned}$$

- (iii) If $s \in G$, then by definition $|\text{Act}(s)| = 1$ with $\text{Act}(s) = \chi(E(s))$ and $\mathbb{P}(s, \chi(E(s)), s) = 1$ and $c(s, \chi(E(s))) = 0$. Thus

$$v'(s) = \sum_{s' \in S} \mathbb{P}(s, \alpha, s') \cdot v'(s') = 0.$$

□

Scheduler synthesis. To provide an insight view on the internal process of an MRA we would like to know how non-deterministic choices are resolved. Thus in addition to the result of the maximum or minimum expected goal-bounded reward, we would like to provide the corresponding scheduler. As shown in Theorem 3.2 a stationary deterministic scheduler is sufficient for the expected reward. Hence, we can provide the choice of each state to describe how the expected goal-bounded reward can be reached.

Example 3.3. Consider Example 3.2. The only state with a non-deterministic choice in the MRA depicted in Figure 3.3 is s_2 . Thus, to determine which choice has to be taken to obtain the expected goal-bounded reward, we just have to evaluate the equations of s_2 . Hence we obtain:

$$16.1562 \geq 0.7 \cdot \left(\frac{40}{7} + 11.4375 \right) + 0.3 \cdot \left(\frac{40}{3} + 0.5 \right) \quad (3.2)$$

$$16.1562 \geq 0.8 \cdot \left(\frac{5}{4} + 13.525 \right) + 0.2 \cdot (5 + 0.5) \quad (3.3)$$

where Equation (3.2) is true but (3.3) is false. Since Equation (3.2) is induced by taking α in s_2 , the stationary deterministic scheduler inducing the maximal expected goal-bounded reward will also choose α .

3.2.2 Time-bounded reward

The details about time-bounded reward are given in [GTHRS14b] and were worked out by Hassan Hatefi. In the following we only give a brief overview of the general idea.

A time-bounded reward is the reward gained until a time-bound t is reached. We denote the reward up to time point t by the random variable $\text{reward}(\cdot, t)$. For an infinite path π we first find the prefix of π up to time point t and then obtain the reward as defined in Definition 2.15:

$$\begin{aligned} \text{reward}(\pi, t) &= \text{reward}(\text{prefix}(\pi, t)) \\ &\quad + \rho((\downarrow \text{prefix}(\pi, t))) \cdot (t - \Delta(\text{prefix}(\pi, t))) \end{aligned} \quad (3.4)$$

The maximum time-bounded reward can then be described as the maximum expected-reward gained within some interval $I = [0, b]$, starting from some initial state s :

$$\mathcal{R}^{\max}(s, b) = \sup_{D \in \text{GM}} \int_{\text{Paths}} \text{reward}(\pi, b) \Pr_{s,D}(\text{d}\pi). \quad (3.5)$$

For Equation (3.6) there exists a fixed point characterisation (FPC) for computing the optimal reward within some interval of time, similar to time-bounded reachability probability [GHHKT14]. However, similar to time-bounded reachability, the FPC is not algorithmically tractable and needs to be discretised. Hence, the time horizon $[0, b]$ has to be divided into a (generally large) number of equidistant time steps. Each time step should be of length $0 < \delta \leq b$, such that $b = k\delta$ for some $k \in \mathbb{N}$.

First, $\mathcal{R}^{\max}(s, b)$ is expressed in terms of its behaviour in the first discretisation step $[0, \delta)$. Therefore, we partition the paths from s into the set $\mathcal{P}_1$ of paths that make their first Markovian jump in $[0, \delta)$ and the set $\mathcal{P}_2$ of paths that do not. We write $\mathcal{R}^{\max}(s, b)$ as the sum of

1. The expected reward obtained in $[0, \delta)$ by paths from $\mathcal{P}_1$
2. The expected reward obtained in $[\delta, b]$ by paths from $\mathcal{P}_1$
3. The expected reward obtained in $[0, \delta)$ by paths from $\mathcal{P}_2$
4. The expected reward obtained in $[\delta, b]$ by paths from $\mathcal{P}_2$

It turns out that the first three items, denoted by $A(s, b)$ can be grouped together and represent the FPC for the case that $s \in MS$ and $b \neq 0$. This will result into the equation

$$\mathcal{R}^{\max}(s, b) = A(s, b) + e^{-E(s)\delta} \mathcal{R}^{\max}(s, b - \delta). \quad (3.6)$$

Note that the exact computation of $A(s, b)$ is in general still intractable. However, by defining a sufficiently small discretisation constant δ at most one Markovian transition fires within a discretisation step with high probability. Hence, the reward gained for paths with multiple Markovian jumps within at least one such interval is negligible. By omitting those from the computation only a small error is introduced. However, this error can be defined a-priori, i.e. one can compute a discretisation constant that guarantees a given error bound.

3.3 Long run rewards

Next, we are interested in the average cumulative reward induced by a set of goal states $G \subseteq S$ in the long-run. Hence, all state and action rewards for states $s \in S \setminus G$ are set to 0.

Definition 3.4 (Long-run reward). Let $\mathcal{M} = (\mathcal{A}, \rho, r)$ be an MRA and $G \subseteq S$ a set of goal states. We define the random variable $\mathcal{L}_{\mathcal{M}}: Paths \rightarrow \mathbb{R}_{\geq 0}$ as the long-run reward over paths in $\mathcal{M}$. For an infinite path $\pi \in Paths^{\omega}(\mathcal{M})$ let

$$\mathcal{L}_{\mathcal{M}}(\pi) = \lim_{t \rightarrow \infty} \frac{1}{t} \cdot \text{reward}(\pi, t)$$

such that the long-run average reward under a scheduler D is given by

$$\text{LRR}_{\mathcal{M}}^D(s) = \mathbb{E}_{s,D}(\mathcal{L}_{\mathcal{M}}) = \int_{Paths} \mathcal{L}_{\mathcal{M}}(\pi) \Pr_{s,D}(d\pi).$$

Then, the maximal long-run average reward on $\mathcal{M}$ starting in state $s \in S$ is:

$$\text{LRR}_{\mathcal{M}}^{\max}(s) = \sup_{D \in GM} \mathbb{E}_{s,D}(\mathcal{L}_{\mathcal{M}}) = \sup_{D \in GM} \int_{Paths} \mathcal{L}_{\mathcal{M}}(\pi) \Pr_{s,D}(d\pi) \quad (3.7)$$

The computation of the expected long-run reward can be split into three steps:

Figure 3.4: Example of the underlying graph of an MRA.

- (1) Determine all maximal end components of MRA $\mathcal{M}$;
- (2) Determine $\text{LRR}_{\mathcal{M}_i}^{\max}$ for each maximal end component $\mathcal{M}_i$;
- (3) Reduce the computation of $\text{LRR}_{\mathcal{M}}^{\max}(s)$ to an SSP problem.

As formally defined in Definition 2.6 a sub-MRA $\mathcal{M}$ is a pair (S', T) with $S' \subseteq S$ and T is a non-empty set of extended transitions. Further, T assigns to all $s \in S'$ a non-empty set of extended transitions, such that for all $(s, \alpha, \mu) \in T(s)$ implies $\text{Succ}(s, \alpha, \mu) \in S'$. An *end component* (see Definition 2.7, page 28) is a sub-MRA whose underlying graph is strongly connected; it is maximal (a *MEC*) w.r.t. T if it is not contained in any other end component (S'', T') (see Definition 2.8, page 28).

For the first step we build the underlying graph $G_{\mathcal{M}}$ of MRA $\mathcal{M}$ as presented in Definition 2.5 on page 26. Then we can apply a graph-based algorithm [CH11; Alf97] to determine the maximal end components of $\mathcal{M}$. In the remainder of this section we will focus on the second and third step.

Example 3.4. Consider the MRA $\mathcal{M}$ depicted in Figure 3.4a. To obtain its maximal end components we create its underlying graph $G_{\mathcal{M}}$ (depicted in Figure 3.4b) according to Definition 2.5. Now we can use standard graph algorithms to determine the SCCs in $G_{\mathcal{M}}$, which are given by:

$$\begin{aligned} \text{SCC}_1 &= \{\langle s_1 \rangle, \langle s_1, \alpha, \mu_{s_1}^\alpha \rangle, \langle s_2 \rangle, \langle s_2, \chi(1), \mathbb{1}_{s_1} \rangle, \langle s_3 \rangle, \langle s_3, \beta, \mathbb{1}_{s_4} \rangle, \langle s_4 \rangle, \langle s_4, \chi(3), \mathbb{1}_{s_2} \rangle\} \\ \text{SCC}_2 &= \{\langle s_5 \rangle, \langle s_5, \chi(1), \mathbb{1}_{s_5} \rangle\}. \end{aligned}$$

According to Definition 2.8 both SCCs fulfil the condition for maximal end components. Thus by mapping the SCCs back to $\mathcal{M}$ we obtain the MECs:

$$\begin{aligned} \mathcal{M}_1 &= (\{s_1, s_2, s_3, s_4\}, \{s_0 \xrightarrow{\alpha} \mu_{s_1}^\alpha, s_2 \xrightarrow{\chi(1)} \mathbb{1}_{s_1}, s_3 \xrightarrow{\beta} s_4, s_4 \xrightarrow{\chi(3)} \mathbb{1}_{s_2}\}) \\ \mathcal{M}_2 &= (\{s_5\}, \{s_5 \xrightarrow{\chi(1)} \mathbb{1}_{s_5}\}). \end{aligned}$$

3.3.1 Step 2: Unichain MRA

A MEC can be seen as a *unichain* MRA: an MRA that yields a strongly connected graph structure under any stationary deterministic scheduler. Since the long-run average reward is dependent on the long-run average time spent in the corresponding states, we first give the definition of long-run average as presented in [GHHKT14]. Consider a state $s \in S$ on an infinite path π in an MRA $\mathcal{M}$ up to a time bound $t \in \mathbb{R}_{\geq 0}$. Then the fraction of time spent in $s \in S$ is given by

the random variable $A_{s,t}(\pi) = \frac{1}{t} \int_0^t \mathbb{1}_s(\pi @ u) du$. Note that $\mathbb{1}_s(\pi @ u) = 1$ if and only if $s \in MS$ and it is contained in the sequence $\pi @ u$, otherwise 0. By taking the limit $\lim_{t \rightarrow \infty}$ we obtain the random variable

$$A_s(\pi) = \lim_{t \rightarrow \infty} A_{s,t}(\pi) = \lim_{t \rightarrow \infty} \frac{1}{t} \int_0^t \mathbb{1}_s(\pi @ u) du. \quad (3.8)$$

The expectation of A_s for a scheduler D yields the corresponding long-run average time spent in $s \in S$:

$$\text{LRA}^D(s) = \mathbb{E}_D(A_s) = \int_{Paths} A_s(\pi) \cdot \Pr_D(d\pi). \quad (3.9)$$

Now we can use the long-run average time to define the long-run average reward. Therefore, we have to consider the frequency of passing through a state. For a Markovian state $s \in MS$ the frequency of taking a transition out of s can be determined by the long-run average time spent in s times its exit rate $E(s)$. For a probabilistic state $s \in PS$ the frequency of taking a transition out of s is equal to the frequencies of its incoming transitions. Note that since in an unichain MRA $\mathcal{M}$, for any two states $s, s' \in S$, $\text{LRR}_{\mathcal{M}}^{\max}(s)$ and $\text{LRR}_{\mathcal{M}}^{\max}(s')$ coincide, we omit the starting state and just write $\text{LRR}_{\mathcal{M}}^{\max}$.

Theorem 3.3. *For a unichain MRA $\mathcal{M}$, for each $s \in S$ the value of $\text{LRR}_{\mathcal{M}}^{\max}(s)$ equals*

$$\text{LRR}_{\mathcal{M}}^{\max} = \sup_D \sum_{s \in S} \left(\rho(s) \cdot \text{LRA}^D(s) + \nu^D(s) \cdot \sum_{s' \in S} \mu_s^{D(s)}(s') \cdot r(s, D(s), s') \right)$$

where ν is the frequency of taking a transition out of a state, defined by

$$\nu^D(s) = \begin{cases} \text{LRA}^D(s) \cdot E(s) & \text{if } s \in MS \\ \sum_{s'' \in S} \mu_{s''}^{D(s'')}(s) \cdot \nu^D(s'') & \text{if } s \in PS. \end{cases}$$

Hence, the long-run average reward gathered in an unichain MRA $\mathcal{M}$ is defined over its state rewards weighted with their long-run average time and the transition rewards weighted by the frequency of taking a transition out of a state. Note that probabilistic states are left immediately, so $\text{LRA}^D(s) = 0$ if $s \in PS$. Further, by assigning $\rho(s) = 1$ to all $s \in MS \cap G$ and setting all other rewards to 0, we compute the long-run average time spent in a set of goal states.

Proof of Theorem 3.3. We show that Theorem 3.3 and Equation (3.7) coincide. Therefore, we will distinguish two cases: $s \in MS$ and $s \in PS$. Note that we denote by $|\pi @ t|$ the index of the last state of the sequence $\pi @ t$, and with $\rho(\pi @ t)$ the state reward of the last state of the sequence $\pi @ t$. Further let $\Delta(\pi @ t) = \Delta(\text{prefix}(\pi, t))$ be the time on path π up to the last state of sequence $\pi @ t$. Let $\text{LRR}_{\mathcal{M}}^{\max}$ be defined as in Equation (3.7) and $\text{reward}(\pi, t)$ as in Equation (3.4). Now we have

$$\text{LRR}_{\mathcal{M}}^{\max} = \sup_{D \in GM} \int_{Paths} \lim_{t \rightarrow \infty} \frac{1}{t} \text{reward}(\pi, t) \Pr_D(d\pi)$$

$$\begin{aligned}
&= \sup_{D \in GM} \int_{Paths} \lim_{t \rightarrow \infty} \frac{1}{t} \left(\rho(\pi @ t) \cdot (t - \Delta(\pi @ t)) \right. \\
&\quad \left. + \sum_{i=0}^{|\pi @ t|-1} \left(\rho(\pi[i]) \cdot t_i + r(\pi[i], \alpha_i, \pi[i+1]) \right) \right) \Pr_D(d\pi)
\end{aligned}$$

Note, that we can include the Dirac distribution of visiting a state s on path π into the equation. This will result into the following:

$$\begin{aligned}
&= \sup_{D \in GM} \int_{Paths} \lim_{t \rightarrow \infty} \frac{1}{t} \left(\sum_{s \in S} \left(\mathbb{1}_s(\pi @ t) \cdot \rho(s) \cdot (t - \Delta(\pi @ t)) \right. \right. \\
&\quad \left. \left. + \sum_{i=0}^{|\pi @ t|-1} \left(\mathbb{1}_s(\pi[i]) \cdot \left(\rho(s) \cdot t_i + r(s, \alpha_i, \pi[i+1]) \right) \right) \right) \right) \Pr_D(d\pi)
\end{aligned}$$

By applying several transformation steps we can separate the time, state reward, and transition reward and obtain

$$\begin{aligned}
&= \sup_{D \in GM} \int_{Paths} \lim_{t \rightarrow \infty} \frac{1}{t} \left(\sum_{s \in S} \left(\mathbb{1}_s(\pi @ t) \cdot (t - \Delta(\pi @ t)) + \sum_{i=0}^{|\pi @ t|-1} \mathbb{1}_s(\pi[i]) \cdot t_i \right) \cdot \rho(s) \right. \\
&\quad \left. + \sum_{i=0}^{|\pi @ t|-1} \left(\mathbb{1}_s(\pi[i]) \cdot r(s, D(s), \pi[i+1]) \right) \right) \Pr_D(d\pi)
\end{aligned}$$

Now by inspecting the term

$$\mathbb{1}_s(\pi @ t) \cdot (t - \Delta(\pi @ t)) + \sum_{i=0}^{|\pi @ t|-1} \mathbb{1}_s(\pi[i]) \cdot t_i$$

one can see that it describes the time spent in state s on path π up to time t . Hence, we can substitute the term by $\int_0^t \mathbb{1}_s(\pi @ u) du$. With this substitution and several transformation steps we obtain

$$\begin{aligned}
&= \sup_{D \in GM} \sum_{s \in S} \left(\rho(s) \cdot \int_{Paths} \lim_{t \rightarrow \infty} \frac{1}{t} \int_0^t \mathbb{1}_s(\pi @ u) du \Pr_D(d\pi) \right. \\
&\quad \left. + \int_{Paths} \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{i=0}^{|\pi @ t|-1} \left(\mathbb{1}_s(\pi[i]) \cdot r(s, \alpha_i, \pi[i+1]) \right) \Pr_D(d\pi) \right) \\
&= \sup_{D \in GM} \sum_{s \in S} \left(\rho(s) \cdot \text{LRA}^D(s) \right. \\
&\quad \left. + \int_{Paths} \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{i=0}^{|\pi @ t|-1} \left(\mathbb{1}_s(\pi[i]) \cdot r(s, D(s), \pi[i+1]) \right) \Pr_D(d\pi) \right)
\end{aligned}$$

Now we include the Dirac distribution of being in state s' into the second part of the equation and obtain

$$= \sup_{D \in GM} \sum_{s \in S} \left(\rho(s) \cdot \text{LRA}^D(s) \right)$$

$$\begin{aligned}
& + \sum_{s' \in S} \int_{Paths} \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{i=0}^{|\pi @ t|-1} \left(\mathbb{1}_s(\pi[i]) \cdot \mathbb{1}_{s'}(\pi[i+1]) \cdot r(s, D(s), s') \right) \Pr_D(d\pi) \\
& = \sup_{D \in GM} \sum_{s \in S} \left(\rho(s) \cdot \text{LRA}^D(s) \right. \\
& \quad \left. + \sum_{s' \in S} r(s, D(s), s') \cdot \int_{Paths} \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{i=0}^{|\pi @ t|-1} \left(\mathbb{1}_s(\pi[i]) \cdot \mathbb{1}_{s'}(\pi[i+1]) \right) \Pr_D(d\pi) \right)
\end{aligned}$$

Note that $\mathbb{1}_s(\pi[i]) \cdot \mathbb{1}_{s'}(\pi[i+1]) = 1$ iff $\mu_s(s') > 1$, otherwise 0. This allows us to rewrite the equation and obtain

$$\begin{aligned}
& = \sup_{D \in GM} \sum_{s \in S} \left(\rho(s) \cdot \text{LRA}^D(s) \right. \\
& \quad \left. + \sum_{s' \in S} r(s, D(s), s') \cdot \mu_s(s') \cdot \int_{Paths} \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{i=0}^{|\pi @ t|-1} \mathbb{1}_s(\pi[i]) \Pr_D(d\pi) \right)
\end{aligned}$$

Let $\mathcal{F}_s : S \rightarrow \mathbb{R}_{\geq 0}$ be the random variable defining the frequency of taking transitions out of a state s such that

$$\mathcal{F}_s(\pi) = \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{i=0}^{|\pi @ t|-1} \mathbb{1}_s(\pi[i])$$

is the frequency of taking transitions out of s on path π . Then we can define the frequency of taking a transition out of a state s under scheduler D as

$$\nu^D(s) = \int_{Paths} \mathcal{F}_s(\pi) \Pr_D(d\pi).$$

Thus we obtain

$$= \sup_{D \in GM} \sum_{s \in S} \left(\rho(s) \cdot \text{LRA}^D(s) + \nu^D(s) \cdot \sum_{s' \in S} \mu_s(s') \cdot r(s, D(s), s') \right)$$

Now we solve $\nu^D(s)$ for $s \in MS$ and $s \in PS$. First let $s \in MS$, then we extend the equation with the long-run average random variable $A_s(\pi)$ from Equation (3.8) and obtain

$$\nu^D(s) = \int_{Paths} A_s(\pi) \cdot \frac{\mathcal{F}_s(\pi)}{A_s(\pi)} \Pr_D(d\pi)$$

Note that $A_s(\pi)$ denotes the long-run average time spent in s on path π and $\frac{\mathcal{F}_s(\pi)}{A_s(\pi)}$ is the ratio of the frequency of taking a transition out of s on path π divided by $A_s(\pi)$. Hence, it describes the exit rate of s on path π and will be denoted by $E_s(\pi)$. Now let $P = \{\pi | \pi \in Paths \wedge A_s(\pi) = \text{LRA}^D(s)\}$ be the set of paths such that $A_s(\pi)$ is equivalent to the long-run average time spent on s . Thus we can split the interval into two and obtain

$$\nu^D(s) = \int_P A_s(\pi) \cdot E_s(\pi) \Pr_D(d\pi) + \int_{Paths \setminus P} A_s(\pi) \cdot E_s(\pi) \Pr_D(d\pi)$$

Let $A_s(\pi, k) = \lim_{t \rightarrow k} A_{s,t}(\pi)$. Let $P_\epsilon(k) = \{\pi | \pi \in Paths \wedge |A_s(\pi, k) - LRA^D(s)| \leq \epsilon\}$ such that $P = \lim_{\epsilon \rightarrow 0} \lim_{k \rightarrow \infty} P_\epsilon(k)$. For all paths $\pi \in Paths$ it holds that $\forall \epsilon > 0$. $\lim_{k \rightarrow \infty} \Pr(|A_s(\pi, k) - LRA^D(s)| \geq \epsilon) = 0$. Thus, $\forall \epsilon > 0$. $\lim_{k \rightarrow \infty} \Pr(\pi \notin P_\epsilon(k)) = 0$, and $\lim_{\epsilon \rightarrow 0} \lim_{k \rightarrow \infty} \Pr(\pi \in P_\epsilon(k)) = 1$. Hence, $\Pr(P) = 1$ and $\Pr(Paths \setminus P) = 0$. Thus it follows

$$\begin{aligned} \nu^D(s) &= \int_P A_s(\pi) \cdot E_s(\pi) \Pr_D(d\pi) \\ &= \int_P LRA^D(s) \cdot E_s(\pi) \Pr_D(d\pi) \\ &= LRA^D(s) \cdot \int_P E_s(\pi) \Pr_D(d\pi) \end{aligned}$$

Further, solving the integral over the exit rate for s over all paths inducing $LRA^D(s)$ will result in the exit rate of s :

$$\nu^D(s) = LRA^D(s) \cdot E(s).$$

For the second case let $s \in PS$. Note that $\Pr_D(\pi) = \sum_{s'' \in S} \Pr(\pi')$ with $\pi' = s''\pi$

$$\begin{aligned} \nu^D(s) &= \sum_{s'' \in S} \int_{Paths} \mathcal{F}_s(\pi') \Pr_D(d\pi') \\ &= \sum_{s'' \in S} \int_{Paths} \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{i=0}^{|\pi'@t|-1} \mathbb{1}_{s''}(\pi'[i]) \cdot \mathbb{1}_s(\pi'[i]) \Pr_D(d\pi') \end{aligned}$$

Note that $\mathbb{1}_{s''}(\pi'[i]) \cdot \mathbb{1}_s(\pi'[i]) = 1$ if and only if $\mu_{s''}(s) > 0$, otherwise 0. Thus we can transform the equation into

$$\begin{aligned} &= \sum_{s'' \in S} \mu_{s''}(s) \cdot \int_{Paths} \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{i=0}^{|\pi'@t|-1} \mathbb{1}_{s''}(\pi'[i]) \cdot \Pr_D(d\pi') \\ &= \sum_{s'' \in S} \mu_{s''}(s) \cdot \nu^D(s'') \end{aligned}$$

□

To solve the equation from Theorem 3.3 it would be too expensive to compute the long-run average time as well as the frequency of the transitions for each state and for all possible schedulers. Instead, we can compute $LRR_{\mathcal{M}}^{\max}$ by solving a system of linear inequations following the concepts of [Alf97]. Given a unichain MRA $\mathcal{M}$, let k denote the optimal average reward accumulated in the long-run and executing the optimal policy. Then, for all $s \in S$ there is a function $h(s)$ that describes a differential cost per visit to state s , such that a system of inequations can be constructed as follows:

Minimise k subject to:

$$\begin{cases} h(s_i) \geq \frac{\bar{\rho}(s_i)}{E(s_i)} - \frac{k}{E(s_i)} + \sum_{s_j \in S} \mathbb{P}_{s_i}(s_j) \cdot h(s_j) & \text{if } s_i \in MS \\ h(s_i) \geq \sum_{s_j \in S} \mu_{s_i}^\alpha(s_j) \cdot (r(s_i, \alpha, s_j) + h(s_j)) & \text{if } s_i \in PS \wedge \forall \alpha \in Act(s_i) \end{cases}$$

where the state and action reward of Markovian states are combined as $\bar{\rho}(s_i) = \rho(s_i) + (r_{s_i} \cdot E(s_i))$ where $r_{s_i} = \sum_{s_j \in S} \mathbb{P}_{s_i}(s_j) \cdot r(s_i, \chi(E(s_i)), s_j)$. Standard linear programming algorithms, e.g. the simplex method [Wun96], can be applied to solve the above system of linear equations.

Example 3.5. Consider the MRA $\mathcal{M}$ depicted in Figure 3.4a and the MECs computed in Example 3.4. Lets focus on MEC $\mathcal{M}_1$. We can build the following system of linear equations:

Minimise k subject to

$$\begin{aligned} h(s_1) &\geq 0.4h(s_2) + 0.6h(s_3) \\ h(s_2) &\geq 2 - k + h(s_1) \\ h(s_3) &\geq h(s_4) \\ h(s_4) &\geq 1 - \frac{k}{3} + h(s_2) \end{aligned}$$

where $k \geq 0$. This system of linear equations can also be represented as follows:

$$\left(\begin{array}{ccccc|c} -1 & 0.4 & 0.6 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & -1 & -2 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & -\frac{1}{3} & -1 \end{array} \right)$$

By solving this under-determined matrix we obtain $k = 2\frac{1}{6}$ and therefore $\text{LRR}_{\mathcal{M}_1}^{\max} = 2\frac{1}{6}$.

Theorem 3.4. *The long-run average reward of an unichain MRA coincides with the limit of the time-bounded expected cumulative reward, such that:*

$$\text{LRR}^D(s) = \lim_{t \rightarrow \infty} \frac{1}{t} \mathcal{R}^D(s, t).$$

Proof. We show a sketch proof that the definition of LRR coincides with the definition of the cumulative reward in the long-run.

$$\begin{aligned} \text{LRR}_{\mathcal{M}}^D(s) &= \mathbb{E}_{s,D}(\mathcal{L}_{\mathcal{M}}) = \int_{Paths} \mathcal{L}_{\mathcal{M}}(\pi) \Pr_{s,D}(\mathrm{d}\pi) \\ &= \int_{Paths} \lim_{t \rightarrow \infty} \frac{1}{t} \cdot \text{reward}(\pi, t) \Pr_{s,D}(\mathrm{d}\pi) \end{aligned}$$

Let $\sum_{k=1}^n g(\pi, k) = \text{reward}(\pi, n)$ with $g(\pi, 1) = \text{reward}(\pi, 1)$ and $g(\pi, k) = \text{reward}(\pi, k) - \text{reward}(\pi, k-1)$ for $k > 1$.

$$\begin{aligned}
&= \int_{Paths} \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^t g(\pi, k) \Pr_{s,D}(d\pi) \\
&= \lim_{t \rightarrow \infty} \frac{1}{t} \int_{Paths} \sum_{k=1}^t g(\pi, k) \Pr_{s,D}(d\pi) \\
&= \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^t \int_{Paths} g(\pi, k) \Pr_{s,D}(d\pi) \\
&= \lim_{t \rightarrow \infty} \frac{1}{t} \int_{Paths} \text{reward}(\pi, t) \Pr_{s,D}(d\pi) \\
&= \lim_{t \rightarrow \infty} \frac{1}{t} \cdot \mathcal{R}^D(s, t).
\end{aligned}$$

□

3.3.2 Step 3: Arbitrary MRAs

To obtain the long-run average reward in an arbitrary MRA, we have to weigh the obtained long-run rewards in each maximal end component with the probability to reach those from s . This is equivalent to the third step in the long-run average computation of [GHHKT13b]. Let $\mathcal{M}$ be an MRA with initial state s^0 and maximal end components $\{\mathcal{M}_1, \dots, \mathcal{M}_n\}$ for $n > 0$ where MRA $\mathcal{M}_j$ has state space S_j .

Theorem 3.5. *For MRA $\mathcal{M} = (\mathcal{A}, \rho, r)$ with MECs $\{\mathcal{M}_1, \dots, \mathcal{M}_n\}$ with state spaces $S_1, \dots, S_n \subseteq S$, and a set of goal states $G \subseteq S$*

$$\text{LRR}^{\max}(s^0, G) = \sup_{D \in \text{GM}} \sum_{i=1}^n \text{LRR}_i^{\max}(G) \cdot \Pr_{s^0, D}(\Diamond \Box S_i),$$

where $\Pr_{s^0, D}(\Diamond \Box S_i)$ is the probability to eventually reach and continuously stay in some states in S_i from s^0 under scheduler D and $\text{LRR}_i^{\max}(G)$ is the long-run reward of $G \cap S_i$ in unichain MRA $\mathcal{M}_i$.

Therefore, the computation of the long-run reward for an arbitrary MRA is split into computing the long-run reward in each MEC and solving a non-negative SSP problem using the LRR results as goal costs. Hence, in MRA $\mathcal{M}$ we replace each maximal end component $\mathcal{M}_i$ by two fresh states $q_i, u_i \notin S$. Intuitively, q_i is representing $\mathcal{M}_i$ whereas u_i encodes the gate to and from $\mathcal{M}_i$. Let U denote the set of u_i states and Q the set of q_i states.

Figure 3.5: Example for Definition 3.5.

Definition 3.5 (SSP for long-run reward). Let $\mathcal{M}$ be an MRA with maximal end components $\mathcal{M}_1$ through $\mathcal{M}_k$ with S_i the state space of $\mathcal{M}_i$. The SSP of MRA $\mathcal{M}$ for the LRA in $G \subseteq S$ is

$$\text{ssp}_{\text{LRR}}(\mathcal{M}) = ((S \setminus \bigcup_{i=1}^k S_i) \cup U \cup Q, \text{Act}^X \cup \{\perp\}, \mathbb{P}', s_0, Q, c, g),$$

where $g(q_i) = \text{LRA}_i^{\max}(G)$ for $q_i \in Q$ and $c(s, \sigma) = 0$ for all s and $\sigma \in \text{Act}^X$. $\mathbb{P}'$ is defined as follows. $\mathbb{P}'(s, \cdot, s')$ equals $\mathbb{P}(s, \cdot, s')$ for all $s, s' \in S'$ where $S' = S \setminus \bigcup_{i=1}^k S_i$. For $u_j \in U$:

$$\mathbb{P}'(u_j, \sigma, s') = \mathbb{P}(S_j, \sigma, s') \quad \text{if } s' \in S' \setminus S_j \quad \text{and} \quad \mathbb{P}'(u_i, \sigma, u_j) = \mathbb{P}(S_i, \sigma, S_j) \quad \text{for } i \neq j.$$

Finally, we have: $\mathbb{P}'(q_j, \perp, q_j) = 1 = \mathbb{P}'(u_j, \perp, q_j)$ and $\mathbb{P}'(s, \sigma, u_j) = \mathbb{P}(s, \sigma, S_j)$.

Here, $\mathbb{P}(s, \alpha, S')$ is a shorthand for $\sum_{s' \in S'} \mathbb{P}(s, \alpha, s')$; similarly, we use $\mathbb{P}(S', \alpha, s')$ = $\sum_{s \in S'} \mathbb{P}(s, \alpha, s')$. The terminal costs of the new q_i -states are set to $\text{LRA}_i^{\max}(G)$.

Example 3.6. Consider the MRA $\mathcal{M}$ from Figure 3.5a, having MECs $\mathcal{M}_1$ with $S_1 = \{s_1, s_2, s_3, s_4\}$ and $\mathcal{M}_2$ with $S_2 = \{s_5\}$. By Definition 3.5, $\text{ssp}_{\text{LRR}}(\mathcal{M})$ is defined as follows. As $k=2$, $U = \{u_1, u_2\}$ and $Q = \{q_1, q_2\}$. Hence, $S_{\text{ssp}} = \{s_0, u_1, u_2, q_1, q_2\}$. First consider $s, s' \in S'$. Since, $S' = \{s_0\}$ and there exists no transition from s_0 to s_0 we can omit the first rule. Now consider all outgoing transitions from MECs. For $\mathcal{M}_1$ there exists a transition from s_3 to s_5 in the underlying MRA, where $s_3 \in S_1$ and $s_5 \in S_2$. It follows that $\mathbb{P}'(u_1, \alpha, u_2) = \mathbb{P}(s_3, \alpha, S_2) = 1$. Now consider all states in U and Q and add new transitions with $\mathbb{P}(u_i, \perp, q_i) = \mathbb{P}(q_i, \perp, q_i) = 1$ for $i = 1, 2$. Finally, consider all states $s \in S_{\text{ssp}} \cap S$ with a transition into a MEC. Hence, $\mathbb{P}'(s_0, \perp, u_1) = \mathbb{P}(s_0, \perp, s_1) = 1$. The MDP of $\text{ssp}_{\text{LRR}}(\mathcal{M})$ is depicted in Figure 3.5b.

Note that the scheduler from Theorem 3.5 does not have to induce the maximal long-run average reward when applied on the corresponding MRA $\mathcal{M}$. This is since the scheduler is only concerned to stay in a MEC and therefore could induce a multi chain or a non optimal value. Hence, we have to adjust the scheduler D from Theorem 3.5 such that the optimal LRR is computed in the corresponding MECs of $\mathcal{M}$.

Lemma 3.1. Let D be the scheduler inducing $\text{LRR}^{\max}$ on $\mathcal{M}$ as in Theorem 3.5. Let D induce a unichain MEC with non-optimal $\text{LRR}^{\max}$. Then there exists a stationary deterministic scheduler D' inducing a unichain MEC such that the long-run ratio is at least as good as for D .

Proof. By the limit in the long run reward ratio $\mathcal{L}_{\mathcal{M}}$ it follows that for every $i \geq 0$ the prefix of a path π up to position i can be ignored. Hence, $\mathcal{L}_{\mathcal{M}}(\pi) = \mathcal{L}_{\mathcal{M}}(\pi_i)$ where π_i is the path π from the i -th position onwards. Now given a scheduler D leading into a maximal end component $\mathcal{M}$ we can construct a scheduler D' inducing the optimal long-run ratio as follows: Let D' fix the recurrent class S' of $\mathcal{M}$ with the maximum value induced by D , and for the minimum long-run average reward with the minimum value, respectively. For states outside of S' , D' is a policy that reaches S' with probability 1. $\square$

3.4 Model checking of expected rewards

Consider we are given an MRA and a high-level performability requirement. How can we describe this performability requirement and compute a satisfying set of states in the MRA? The first step is to translate the performability requirement into a logic. The second step is to compute the satisfaction set in the MRA by recursively breaking down the logical formulae. We will provide a broad overview of the model checking capabilities of MRAs based on [HH12]. Therefore, we extend the logic with reward properties based on our reward measures. Furthermore, we shortly discuss how to apply the algorithms presented in Section 3.2 and Section 3.3 to verify requirements w.r.t. the logic.

3.4.1 Continuous stochastic reward logic

Continuous Stochastic Logic [BHHK03] (CSL) is suitable to represent performability requirements on MRAs, which can express a broad range of performance and dependability measures. CSL is an extension of Probabilistic Computation Tree Logic (PCTL) [HJ94; BA95] to continuous-time Markov models. Moreover, Continuous Stochastic Reward Logic (CSRL) extends CSL with rewards and originates from [BHHK00] where it was defined for CTMCs with rewards.

Let $\mathcal{J}$ be the set of all nonempty non negative real intervals with real bounds, then Continuous Stochastic Reward Logic (CSRL) for MRAs is defined as follows.

Definition 3.6 (CSRL Syntax). Let $a \in AP$, $p \in [0, 1]$, $t \in \mathbb{R}_{\geq 0}^{\infty}$, $r \in \mathbb{R}_{\geq 0}$, $I \in \mathcal{J}$ an interval and $\sqsubseteq \in \{<, \leq, \geq, >\}$, CSL state and path formulae are described by

$$\begin{aligned} \Phi &::= a \mid \neg \Phi \mid \Phi \wedge \Phi \mid \mathcal{P}_{\sqsubseteq p}(\phi) \mid \mathcal{E}_{\sqsubseteq r}^t(\Phi) \mid \mathcal{L}_{\sqsubseteq r}(\Phi) \\ \phi &::= \mathcal{X}^I \Phi \mid \Phi \mathcal{U} \Phi \mid \Phi \mathcal{U}^I \Phi \end{aligned}$$

Except for the last two operators of the state formulae this logic corresponds to the CSL logic defined in [ZN10]. Note that

- $\mathcal{P}_{\leq p}(\phi)$ denotes the probability of the set of paths that satisfy ϕ ;
- $\mathcal{E}_{\leq r}^t(\Phi)$ denotes the expected reward to reach states satisfying Φ and,
- $\mathcal{L}_{\leq r}(\Phi)$ denotes the average long-run reward spent in states satisfying Φ .

Given an infinite path $\pi \in Paths^\omega$, π satisfies $\mathcal{X}^I \Phi$ if the first transition of π occurs within time interval I and leads to a state that satisfies Φ . Similarly, the bounded until formula $\Phi \mathcal{U}^I \Psi$ is satisfied by π if π visits states that satisfy formula Φ until it reaches a state that satisfies formula Ψ within the time interval I . In contrast to the bounded until, an unbounded until formula does not constrain the time at which π may visit a state which satisfies Ψ . This corresponds to the interval $[0, \infty)$.

We denote with $\gamma(\pi, n)$ the time interval during which a given path π stays in its n -th state. More formally, it equals $[\Delta(\pi, n), \Delta(\pi, n + 1)]$ if $\Delta(\pi, n) < \Delta(\pi, n + 1)$, and $\{\Delta(\pi, n)\}$ otherwise. Further, we have the random variable

$$V_\Phi(\pi, t) = \begin{cases} \{V_G(\pi) \mid \exists i \in \mathbb{N}. \pi[i] \in G \wedge \pi[i] \models \Phi\} & \text{if } t = \infty \\ \text{reward}(\pi, t) & \text{otherwise,} \end{cases}$$

where $t = \infty$ corresponds to an expected goal-bounded reward, and otherwise to a time-bounded reward. Further we have the random variable

$$\mathcal{L}_\Phi(\pi) = \mathcal{L}_\mathcal{M}(\pi)$$

for the long-run average reward, such that for all $s \in S$ in $\mathcal{M}$ with $s \not\models \Phi$ it holds that $\rho(s) = 0$ and $r(s, \cdot, \cdot) = 0$. The formal semantics of CSRL formulae is then defined as follows.

Definition 3.7 (CSRL Semantics). Let $\mathcal{M} = (\mathcal{A}, \rho, r)$ be a state-labelled MRA, $s \in S$, $a \in AP$, $p \in [0, 1]$, $t \in \mathbb{R}_{\geq 0}$, $r \in \mathbb{R}_{\geq 0}$, $I \in \mathcal{I}$, $\leq \in \{<, \leq, \geq, >\}$, and $\pi \in Paths^\omega$. We define the satisfaction relation $\models$ for state formulae: $s \models a$ iff $a \in L(s)$, $s \models \neg \Phi$ iff $s \not\models \Phi$, $s \models \Phi \wedge \Psi$ iff $s \models \Phi \wedge s \models \Psi$, and

$$\begin{aligned} s \models \mathcal{P}_{\leq p}(\phi) & \quad \text{iff } \forall D \in GM. \Pr_{s,D}(\{\pi \in Paths \mid \pi \models \phi\}) \leq p \\ s \models \mathcal{E}_{\leq r}^t(\Phi) & \quad \text{iff } \forall D \in GM. \int_{Paths} V_\Phi(\pi, t) \Pr_{s,D}(d\pi) \leq r \\ s \models \mathcal{L}_{\leq r}(\Phi) & \quad \text{iff } \forall D \in GM. \int_{Paths} \mathcal{L}_\Phi(\pi) \Pr_{s,D}(d\pi) \leq r \end{aligned}$$

For path formulae:

$$\begin{aligned} \pi \models \mathcal{X}^I \Phi & \quad \text{iff } \pi[1] \models \Phi \wedge \Delta(\pi, 1) \in I \\ \pi \models \Phi \mathcal{U}^I \Psi & \quad \text{iff } \exists n \in \mathbb{N}_0. \gamma(\pi, n) \cap I \neq \emptyset \wedge \pi[n] \models \Psi \wedge \forall k = 0 \dots n-1. \pi[k] \models \Phi \\ \pi \models \Phi \mathcal{U} \Psi & \quad \text{iff } \exists n \in \mathbb{N}_0. \pi[n] \models \Psi \wedge \forall k = 0 \dots n-1. \pi[k] \models \Phi \end{aligned}$$

Example 3.7. Consider a system that can be up and running or be down because of a failure, represented by the atomic propositions *up* and *down*, respectively. If the system is down, then costs will occur. Now we are interested

Figure 3.6: A polling system consisting of a server and two stations.

that the average cost over the long-run is smaller than 1000 units due to possible system failures. This property can be represented by the CSRL long-run average operator $\mathcal{L}_{\leq 1000}(\text{down})$. The formula is satisfied if we accumulate on average only 1000 units of reward while in *down* states. To check this property, we can run $\text{LRR}^{\min}$ and $\text{LRR}^{\max}$ as presented in Section 3.3.

Besides the long-run average costs, we are interested if the expected reward gained in the *up* states covers our average down time cost before the system reaches a *down* state. This condition can be expressed by the CSRL formula $\mathcal{E}_{\geq 1000}^{\infty}(\text{up } \mathcal{U} \text{ down})$. To check this property, we can run $\text{eR}^{\min}$ and $\text{eR}^{\max}$ as presented in Section 3.2.

3.5 Experiments

To assess the algorithms in practice, we provide two case studies: A server polling system based on [Sri91], and a fault-tolerant workstation cluster based on [HHK00]. Both case studies were extended with rewards. Further, we computed the expected-goal bounded reward, expected-time bounded reward and the long-run average reward, as presented in Section 3.2 and Section 3.3, respectively. The experiments were conducted on a 2.2 GHz Intel® Core™ i7-2670QM processor with 8 GB RAM, running Linux. The experiments were performed with the MaMa tool-chain [GTHRS14b]. MaMa allows the modelling of MRAs with MAPA [Tim13] as well as their analysis based on the algorithms presented in this chapter.

Polling system. Figure 3.6 shows an MRA of the polling system. It consists of two stations, each providing a job queue of length $Q = 1$ for $N = 1$ job types, and one server. When the server polls a job from a station, there is a 10% chance that it will erroneously remain in the queue. An impulse reward of

Figure 3.7: Schematic of the workstation cluster.

0.1 is given each time a server takes a job, and a reward of 0.01 per time unit is given for each job in the queue. The rewards are meant to be interpreted as costs in this example. Thus the polling system induces costs for having a job processed and for taking up server memory, respectively.

Table 3.1, Table 3.2 and Table 3.3 show the results obtained by the MaMa tool-chain when analysing for different queue sizes Q and different numbers of job types N . The goal states for the expected reward are those when both queues are full. The error-bound for the time-bounded reward analysis was set to 0.1.

The tables show that the minimal reward does not depend on the number of job types, while the maximal reward does. The long-run reward computation is, for this example, considerably slower than the expected reward, and both increase more than linear with the number of states. The time-bounded reward is more affected by the time bound than the number of states, and the computation time does not significantly differ between the maximal and minimal queries.

Workstation cluster. The second case study is based on a fault-tolerant workstation cluster, described as a GSPN in [Mar10]. The schematic of the workstation cluster is depicted in Figure 3.7. Using the GEMMA [Bam12] tool, the GSPN was converted into a MAPA specification.

The workstation cluster consists of two groups of N workstations, each group connected by one switch. The two groups are connected to each other by a backbone. Workstations, switches and the backbone experience exponentially distributed failures, and can be repaired one at a time. If multiple components are eligible for repair at the same time, the choice is non-deterministic. The overall cluster is considered operational if at least Q workstations are operational and connected to each other. Rewards have been added to the system to simulate the costs of repairs and downtime. Repairing a workstation has cost 0.3, a switch costs 0.1, and the backbone costs 1 to repair. If fewer than Q workstations are operational and connected, a cost of 1 per unit time is incurred.

Tables 3.4, 3.5, and 3.6 show the analysis results for this example. The goal states for the expected reward are the states where not enough operational workstations are connected. The error bound for the time-bounded reward analysis was 0.1. For this example, the long-run rewards are quicker to compute than the expected rewards. The long-run rewards do not vary much with the

Q	N	T_{lim}	Time-bounded reward			
			min	$T(\text{min})$	max	$T(\text{max})$
2	3	1	0.626	0.46	0.814	0.46
2	3	2	0.914	1.64	1.389	1.66
2	3	10	1.005	161.73	2.189	166.59
3	3	1	0.681	4.90	0.893	4.75
3	3	2	1.121	16.69	1.754	17.11
3	3	10	1.314	1653	4.425	1687

Table 3.1: Time-bounded rewards for the polling system.

Q	N	$ S $	$ G $	Goal-bounded reward			
				min	$T(\text{min})$	max	$T(\text{max})$
2	3	1159	405	0.735	0.28	2.110	0.43
2	4	3488	1536	0.735	0.93	3.227	2.01
3	3	11122	3645	1.034	3.14	4.752	8.14
3	4	57632	24576	1.034	31.80	8.878	95.87
4	2	5706	1024	1.330	3.12	4.199	3.12
4	3	102247	32805	1.330	63.24	9.654	192.18

Table 3.2: Goal-bounded rewards for the polling system.

Q	N	$ S $	$ G $	Long-run average reward			
				min	$T(\text{min})$	max	$T(\text{max})$
2	3	1159	405	0.731	0.61	1.048	0.43
2	4	3488	1536	0.731	3.76	1.119	2.21
3	3	11122	3645	0.750	95.60	1.107	19.14
3	4	57632	24576	0.750	5154.6	1.198	705.8
4	2	5706	1024	0.769	38.03	0.968	5.73
4	3	102247	32805	Timeout(2h)			

Table 3.3: Long-run average rewards for the polling system.

scheduler, since multiple simultaneous failures are rare in this system. This also explains the large expected rewards when Q is low: many repairs will occur before the cluster fails. The time-bounded rewards also show almost no dependence on the scheduler.

3.6 Related work

Cloth [Clo06] presented model checking algorithms for Markov reward models (MRMs). MRMs are CTMCs paired with state rewards, and are therefore included in our MRA model. Besides Cloth covered concepts and algorithms needed to model check MRMs with respect to CSRL.

N	Q	T_{lim}	Time-bounded reward			
			min	$T(\text{min})$	max	$T(\text{max})$
4	3	10	0.0467	1.16	0.0467	1.09
4	3	20	0.0968	8.54	0.0968	8.23
4	3	50	0.2481	125.4	0.2481	123.6
4	3	100	0.5004	989.8	0.5004	991.8
4	5	10	0.0454	1.276	0.0454	1.209
4	5	20	0.0929	9.297	0.0929	9.218
4	5	50	0.2333	141.9	0.2333	143.8
4	5	100	0.4610	1123	0.4610	1132

Table 3.4: Time-bounded rewards for the workstation cluster.

Q	N	$ S $	$ G $	Goal-bounded reward			
				min	$T(\text{min})$	max	$T(\text{max})$
4	3	1439	1008	5335	337.9	5348	297.1
4	5	1439	621	6.848	0.4111	6.848	0.4095
4	8	1439	1438	0	0.00019	0	0.00018
8	6	4876	3584	16460	4502	16514	4124
8	8	4876	4415	254.0	55.57	254.0	53.73
8	10	4883	4783	13.70	2.941	13.70	2.904
8	16	4895	4894	0	0.00059	0	0.00061

Table 3.5: Goal-bounded rewards for the workstation cluster.

Q	N	$ S $	$ G $	Long-run average reward			
				min	$T(\text{min})$	max	$T(\text{max})$
4	3	1439	1008	0.00504	0.0272	0.00505	0.143
4	5	1439	621	0.00857	0.00787	0.00864	0.217
4	8	1439	1438	0.01655	0.00709	0.0166	0.182
8	6	4876	3584	0.00983	0.258	0.00984	1.875
8	8	4876	4415	0.00997	0.0920	0.0100	1.992
8	10	4883	4783	0.0134	0.0463	0.0134	2.064
8	16	4895	4894	0.0294	0.0351	0.0294	2.134

Table 3.6: Long-run average rewards for the workstation cluster.

Baier et al. discussed model checking for MRMs [BCHHK10] as well as for CTMDPs with rewards [BHHK08]. They exploited a duality result by transforming the time-bounded reward property into a time-bounded probability property, that allowed the reuse of existing algorithms. However, this result is only applicable for models with state rewards.

Buchholz et al. [BS11] provided a numerical analysis for CTMDPs with state rewards over a finite horizon. Their method is based on the uniformisation of CTMDPs and computes an optimal policy and the resulting reward with a pre defined error bound.

Fu [Fu14] discussed multi-dimensional maximal cost-bounded reachability probabilities over CTMDPs with rewards. Fu proves that the class of measurable deterministic cost-positional scheduler is sufficient to obtain the maximal cost-bounded reachability probability. Besides numerical approximation algorithm for maximal cost-bounded reachability probability are introduced.

Hatefi et al. [Hat+15] and Braitling et al. [Bra+15] focused on improving the scalability of analysing MRAs to make them more appealing, e.g. for large real-life systems. They presented an abstraction technique for MRAs based on stochastic games. In their experiments they achieved some significant speed-ups compared to our direct analysis methods.

3.7 Conclusion

In this chapter we introduced algorithms for the expected reward analysis of MRAs that consider both state-based and transition-based rewards (or, equivalently, costs). In particular we presented algorithms for computing the expected reward until reaching a set of goal states, and for computing the long-run average reward as well as gave an overview for computing the expected reward until reaching a time-bound. This allows for a wide variety of systems featuring non-determinism, discrete probabilistic choice, continuous stochastic timing and transition-based and state-based rewards to be analysed. Moreover, we showed how to adapt CSRL for MRAs including our expected reward properties, allowing the inclusion of MRAs in model checking.

Markov models in the real world: Airborne collision avoidance system

ADVANCEMENTS in technology lead to more and more autonomous systems, leading from autopilots in aircraft, over home automation systems to autonomous driving cars. Hence, tasks that are normally done by a human are executed by an advanced algorithm. Put simply, to automate a task the algorithm has to acquire information from its environment, interpret this information and make a decision based on this interpretation. Therefore, autonomy requires models for decision making, adaptation and self-healing.

Advanced algorithms for decision-making often rely on the use of models, i.e. abstract representations of knowledge that the algorithms need in order to operate correctly. Such algorithms appear increasingly in safety critical systems with the introduction of autonomy, and the need to make decisions, initiate mitigation actions, and adapt to changing environments and unanticipated situations.

In this chapter we describe an encounter with such algorithms in the context of ACAS X, the neXt generation Airborne Collision Avoidance System [KC11; Koc15]. The current collision avoidance standard, TCAS (traffic collision avoidance system) [KD07], is required on all large passenger and cargo aircraft worldwide, and has been successful in preventing mid-air collisions. However, its deterministic logic limits robustness in the presence of unanticipated circumstances. According to the predictions by the Federal Aviation Administration (FAA), air traffic will increase by an average annual rate of 2.5 percent over the next 30 years [Fed15]. To address the increase of air traffic and the growing needs in the safety of the airspace, the FAA is developing and testing ACAS X. As shown in [KHC12], ACAS X promises a number of potential improvements over TCAS including a reduction in collision risk while simultaneously reducing the number of false alerts. To increase robustness, *ACAS X uses a probabilistic model to represent uncertainty, in particular a Markov decision process*. Simulation studies with recorded radar data have confirmed that this novel approach leads to a significant improvement in safety and operational performance. ACAS X has also been the target of formal verification efforts [Jea+15; EG14; EG16].

This chapter focuses on the problem of identifying quality metrics for models used in decision-making. Since a model is an abstraction of information

the question that stands out is: *How can we determine whether this model is satisfactory?* In the context of ACAS X, decisions may look correct in terms of avoiding aircraft collisions, but those decisions are not necessarily supported by the model. How does one establish trust that a model is good enough for a safety-critical system like ACAS X, and that the important decisions made by the associated algorithms are based on correct information?

To address these questions, our research aims at establishing criteria of model quality. These criteria should be measurable, and should be helpful in developing certification standards for algorithms that use models. Moreover, model quality should be directly associated with overall system quality. In other words, poor model quality should indicate or predict violations of system requirements. In this chapter we discuss model conformance relations that we established for ACAS X and the (sometimes unanticipated) results of their application. Moreover, we show techniques that stress-test ACAS X by generating test cases that may exhibit poor model quality with respect to our defined relations.

Origins of the chapter. The work presented in this chapter is based on

- D. Giannakopoulou, D. Guck, and J. Schumann. “Exploring Model Quality for ACAS X”. in: *Proceedings of the 21st International Symposium on Formal Methods (FM 2016)*. Vol. 9995. Lecture Notes in Computer Science. Springer International Publishing

and was carried out during a research internship at the NASA Ames research center. We focus on ACAS X and define relations between the probabilistic model and the real world.

Organisation of the chapter. Section 4.1 describes the ACAS X system and motivates this work, while Section 4.2 provides more details about the probabilistic model on which ACAS X is based. Section 4.3 discusses the relations that we have defined for evaluating model quality. The application of these relations to ACAS X is presented in Section 4.4. In Section 4.5 we focus on the generation of data for stress-testing ACAS X, and present the results of our analysis. Section 4.6 lists related work and Section 4.7 closes the chapter with conclusions and future work.

4.1 The ACAS X system

The aim of a aircraft collision avoidance system is to prevent mid-air collisions while minimising unnecessary pilot alerting and evasive manoeuvres. Uncertainties such as sensor noise or errors, aircraft intent, and pilot behaviour make it challenging to design such systems.

In the context of collision avoidance algorithms, we use the term *loss of horizontal separation* (LHS) to describe the situation where two aircraft are within 500 ft from each other, ignoring their altitude difference. A *near mid-air collision* (NMAC) occurs when the altitude difference between the two aircraft

Figure 4.1: Principled architecture of the ACAS X modelling process (top) and on-board software architecture (bottom).

is at most 100 ft when LHS occurs. We refer to the aircraft equipped with a collision avoidance system as the *ownship*, and the other aircraft as the *intruder*.

The current collision avoidance standard, TCAS [KD07], uses several sources to estimate the current state of the aircraft on which it is deployed and other aircraft in its vicinity. If another aircraft is a potential threat, then TCAS issues a traffic advisory, which gives the pilots of the ownship an audio announcement “Traffic, Traffic” and highlights the intruder on a traffic display. This serves as a warning to raise the pilot’s awareness for the potential need to manoeuvre.

If a manoeuvre becomes necessary, then the system will issue a resolution advisory (RA) instructing the pilot to climb or descend in order to maintain a safe distance. After the encounter is resolved, TCAS issues a “Clear of Conflict” (COC). Only advisories for vertical manoeuvres (climb, descend, and maintain altitude) are given, together with the target rate. For example, advisory DES1500 stands for descend with rate 1500 ft/min. Preventive advisories to avoid climbing or descending may also be provided, as described in [Koc15].

The TCAS system has been implemented as a traditional piece of software with conditional branches that model all the different situations. Several years of research have resulted in the development of the ACAS X system. Although the interface of ACAS X to the pilot is the same as TCAS, the underlying collision avoidance algorithm is dramatically different [Koc15; KC11].

4.1.1 Inside ACAS X

In ACAS X, a Markov decision process (MDP) models a coarse abstraction of how an encounter between two aircraft progresses. A transition in the MDP

represents a time passing of one second after applying an advisory. Based on a reward function and this MDP, dynamic programming techniques are used to generate a look-up table (LUT). This table associates each encounter state in the MDP with a cost for each possible ACAS X advisory. The LUT is deployed on-board the aircraft. Every second, ACAS X uses sensors and other information to compute a probability distribution of the states in which an encounter may be at the current time t . ACAS X interpolates the state estimate within the discrete states of the LUT, and calculates the advisory that bears the lowest cost. Detailed information of how the MDP is generated is given in [Koc15].

Figure 4.1 shows the overall architecture of the ACAS X system and its development process. The top of the panel shows the development of the probabilistic MDP model and the generation of the look-up table using dynamic programming techniques. This LUT comprises the core of the ACAS X software that is running on-board the aircraft (bottom part of Figure 4.1). During each 1-second update, new estimates of the aircraft involved and uncertainties are calculated based upon sensor measurements and transponder responses. Based upon this update, the LUT provides weights for the advisories w.r.t. the aircraft estimate.

In essence, the cost of an advisory computed by dynamic programming is based on how the MDP expects an encounter to evolve from the current state. Therefore, there is a *trade-off* between the MDP model: (1) being accurate enough for the advisories to be appropriate for collision avoidance in reality; and (2) abstract enough for enabling a compact LUT to fit in memory on-board the aircraft. The ACAS X system is based on a relatively simple dynamic model of how aircraft behave. As explained in [Koc15], a simple model is easier to understand and validate, makes the dynamic programming problem more tractable, and results in a smaller controller, which is easier to fit into memory on-board an aircraft. On the other hand, smaller models are less accurate since they have to abstract more. Hence, the key question is: *How can we establish that this simple model is acceptable for ACAS X?* Even though the resulting system is tested extensively with independent high fidelity simulations and flight data, it is hard to establish whether the behaviour of such optimisation algorithms is as expected.

In the remainder of this chapter we establish measurable criteria that directly address model quality. The approaches are all based on establishing relationships between the evolution of encounters (1) as expected by the MDP model and (2) as recorded by high fidelity simulations and actual flight data. Therefore, we define and analyse model conformance relations. Moreover, we show how to generate interesting scenarios based on these relations.

4.2 The ACAS X model

The formulation of the collision avoidance problem used for ACAS X consists of two aircraft, the ownship and the intruder, on a collision course. Note that ACAS X also handles cases with multiple aircraft. However, this chapter will focus on scenarios involving two aircraft. In this section we will describe the

Figure 4.2: Illustration of the state variables describing the ownship and intruder relation.

main ingredients of the MDP model. However, we do not provide a description of the detailed construction of the MDP; this can be found in [Koc15].

The ACAS X model keeps record of the altitude of the intruder relative to the ownship, the aircraft climb rates, the produced advisory and the pilot response.

Definition 4.1 (State variables). The state of an aircraft is represented by the following state variables:

1. $z_{\text{rel}} \in [-8000, 8000]$ ft, the altitude difference between the two aircraft;
2. $dz_o \in [-2500, 2500]$ ft/min, the ownship's climb rate;
3. $dz_i \in [-2500, 2500]$ ft/min, the intruder's climb rate;
4. sRA, encoding the ACAS X advisory produced one second earlier and the advisory the pilot is following, thus modelling pilot delay.

The variables z_{rel} , dz_o and dz_i are also called *geometric* variables.

Figure 4.2 illustrates the first three state variables z_{rel} , dz_o and dz_i .

The advisories available in ACAS X are summarised in Table 4.1. They can be categorised into *preventive* and *corrective* advisories. When the vertical rate is within the minimum/maximum range when the advisory is issued it is called preventive, otherwise corrective. Note that the availability of the advisories in a state depends upon the current advisory. For example, a strengthening can only be issued if the previous advisory was of the same type. Hence, a **SCLI1500** can only be issued after an **MCL** or **CLI1500**. Therefore, not all advisories are enabled in each state of the MDP. This constraint of issuing advisories was already present in the TCAS design.

Note that the sRA state variable encodes the ACAS X advisory given out to the pilot and the pilot response, e.g. if ACAS X issues a **DND** and the pilot is following this advice then $\text{sRA} = \text{DND} - \text{DND}$. Note, that **COC** as well as no pilot response are represented by the term **NONE**. Moreover, the pilot response either matches the given advisory or is **NONE**. Hence, sRA is a set of 33 advisory pilot response actions.

Recall the Definition 2.2 on page 22 where we defined MAs and the subsumption of an MDP in Section 2.2.3 on page 30.

Definition 4.2 (MDP for ACAS X). Let $\mathcal{M} = \langle S, Act, \hookrightarrow \rangle$ be an MDP where:

- S is a finite set of states, such that each state describes the state of the aircraft;
- Act is a finite set of actions, consisting of the ACAS X advisories;
- $\hookrightarrow: S \times Act \times \text{Distr}(S)$ is the probabilistic transition relation.

Since the state space for ACAS X is continuous, the MDP discretises the state variables to keep the model tractable. Hence, ranges are set for each variable, describing the state of the aircraft. Note, that only z_{rel} , dz_o and dz_i have to be discretised. The MDP discretises each state variable with a resolution that depends on the proximity between the aircraft. Previous work [EG16] has shown that the discretisation resolution constitutes an important trade-off between accuracy and the size of the LUT, which is important for implementation on-board the aircraft.

Definition 4.3 (MDP state variables). For a state variable x , we denote its discretised value by $\hat{x}$. Thus, a state in the MDP $\mathcal{M}$ is given by $\hat{s} = \langle \hat{z}_{\text{rel}}, \hat{dz}_o, \hat{dz}_i, \text{sRA} \rangle$ where

1. z_{rel} is discretised in 45 points over the range of ± 8000 ft;
2. dz_o is discretised in 25 points over the range of ± 10000 ft/min;
3. dz_i is discretised in 25 points over the range of ± 10000 ft/min;

with finer discretisation steps near 0. Moreover, to better reflect that a transition in the MDP captures a time step of one second, the values of $\hat{dz}_o$ and $\hat{dz}_i$ are converted to ft/s.

Advisory	Vertical rate		Aural annunciation
	Min (ft/min)	Max (ft/min)	
COC	$-\infty$	∞	Clear of conflict
DNC2000	$-\infty$	2000	Monitor vertical speed
DND2000	-2000	∞	Monitor vertical speed
DNC1000	$-\infty$	1000	Monitor vertical speed
DND1000	-1000	∞	Monitor vertical speed
DNC500	$-\infty$	500	Monitor vertical speed
DND500	-500	∞	Monitor vertical speed
DNC	$-\infty$	0	Level-off, Level-off (or Monitor vertical speed)
DND	0	∞	Level-off, Level-off (or Monitor vertical speed)
MDES	∞	current	Maintain vertical speed, Maintain
MCL	current	∞	Maintain vertical speed, Maintain
DES1500	$-\infty$	-1500	Descend, Descend
CLI1500	1500	∞	Climb, Climb
SDES1500	$-\infty$	-1500	Descend, Descend NOW, Descend, Descend NOW
SCLI1500	1500	∞	Climb, Climb NOW, Climb, Climb NOW
SDES2500	$-\infty$	-2500	Increase Descent, Increase Descent
SCLI2500	2500	∞	Increase Climb, Increase Climb

Table 4.1: ACAS X advisories, their vertical rate range, and their aural annunciation [Koc15].

Figure 4.3: Example of state variable changes in the MDP.

Given a discrete state $\hat{s} = \langle \hat{z}_{\text{rel}}, \hat{dz}_o, \hat{dz}_i, \text{sRA} \rangle$ and an advisory adv , the MDP provides probabilistic state transitions into new states $\hat{s}'$, i.e. $\hat{s} \xrightarrow{\text{adv}} \mu$ with successor states $\hat{s}'$ such that $\mu(\hat{s}') > 0$. We say that a state $\hat{s}$ reaches $\hat{s}'$ by performing adv with state transition probability $p = \mu(\hat{s}')$. Figure 4.3 depicts such a transition. Consider the state $(200, -33, -25, \text{DES1500} - \text{NONE})$ and advisory DES1500 . Now the MDP can transition with probability k into state $(250, -50, -25, \text{DES1500} - \text{DES1500})$. The detailed construction of the probabilistic transition relation for ACAS X is given in [Koc15].

Since the state space of the MDP is discretised, we have to interpolate the actual state variables to the discretised values of the MDP. Figure 4.4 illustrates the interpolation for a two-dimensional space. For a geometrical state of ACAS X, we have to interpolate in three dimensions: z_{rel} , dz_o and dz_i . Note that the variable sRA stays the same. Hence, when we map the current ACAS X state to the model $\mathcal{M}$, we obtain a set of states and a probability distribution over those.

Figure 4.4: Example of the interpolation in a two-dimensional space.

The goal of ACAS X is to avoid NMAC encounters while minimising the amount of advisories issued. Hence, in order to obtain an optimal controller, the individual advisories are associated with a cost. For example, a clear-of-conflict (COC) carries a reward, whereas alerting the pilot through climb and descend advisories carry a small cost to avoid unnecessary pilot alerting. An NMAC situation has an extremely high cost.

τ -distribution. Although the MDP in itself is not aware of timing throughout the encounter, the ACAS X system uses a temporal distribution (τ -distribution) to model the temporal sequence of the encounter and avoidance strategy. The τ -distribution is a weighted distribution over the estimated time to LHS of the aircraft. The time horizon that ACAS X is expected to operate is within 40–50 seconds to provide an adequate alert time before a NMAC situation may occur. Intuitively, the τ -distribution predicts the likelihood of a LHS w.r.t. discrete time points. For example, consider that the aircraft is in state s . The τ -distribution provides a weight w for each time point t , where t is the time to LHS for state s .

Recall Figure 4.1 and the on-board ACAS X implementation. Each second, new estimates of the aircraft state and τ -distribution are calculated based upon sensor measurements and transponder responses. Then those are used to consult the LUT to come up with an appropriate advisory. In addition to the LUT, on-line corrections are also taken into account in selecting advisories (see [Koc15] for details).

4.3 Model Conformance

As discussed, the MDP model used by ACAS X captures the expected evolution of flight encounters as a result of time passing and of the application of resolution advisories. To ensure safe operation, the model and the actual system behaviour must in some way “match up”. In other words, we need to be able to justify that the model is appropriate for the decision making that it is used for.

To this aim, we introduce several notions of *conformance* to characterise model quality. All of these notions capture desired relations between the behaviour or states of a model $\mathcal{M}$ and the actual system $\mathcal{A}$ that uses this model. For example, conformance might require that whenever an encounter results in an NMAC in the actual ACAS X system, then the model of this encounter also results in an NMAC. Or when the actual system $\mathcal{A}$ produces an advisory adv , then the model will produce an identical or compatible advisory.

Similarly, one would expect that all situations that occur in practice in the system will be reflected in the model (within the bounds of the model abstraction). This notion is related to over-approximation in formal methods. Imagine, for example, that during flight, an encounter reaches a state s' from a state s , a transition that the model does not anticipate (i.e. in the model s cannot directly transition to s'). ACAS X may still work appropriately or may produce wrong or unsafe results.

Even though such notions of conformance appear to be relatively simple, defining them in the context of ACAS X is non-trivial due to the presence of probabilistic reasoning and state discretisation. In the following sections we describe how we incorporate these characteristics into our model quality criteria.

4.3.1 Conformance framework set up

As discussed in Section 4.1 and 4.2, during ACAS X deployment, the (geometric) state of an encounter is represented by a weighted distribution of states. At each second, the Cartesian product of the geometric state estimate with the τ -distribution (weighted distribution of estimated time to LHS) forms the current state estimate. This state estimate is then interpolated within the discrete states of the LUT in order to calculate a resolution advisory. Hence, given z_{rel} , dz_o , dz_i as the geometric information and sRA for the previous advisory and pilot response, the geometric variables are subject to the interpolation. Further, the τ -distribution is extended with the weights of the interpolation.

To establish model conformance in this context, our framework needs to set up the model and the actual system appropriately. The model $\mathcal{M}$ is ACAS X as deployed within the MDP model. In fact, if we speak about $\mathcal{M}$, we refer to the MDP. Note that the change in the geometric state distribution in $\mathcal{M}$ is provided by the transitions in the MDP. The actual system $\mathcal{A}$ is ACAS X as deployed and using the LUT within a real flight environment or within a high-fidelity simulator. The change in the geometric states is based on sensor information. To establish model conformance we set up a common initial state for $\mathcal{M}$ and $\mathcal{A}$ and compare their evolution's according to conformance criteria. More specifically, we test the conformance between $\mathcal{M}$ and $\mathcal{A}$ relatively to an encounter $\mathcal{E}$. Therefore, the initial state of $\mathcal{M}$ is set to the initial state of $\mathcal{A}$ for each encounter $\mathcal{E}$ to be analysed for conformance.

There are two options for comparing the evolution of encounters. One is a *stepwise* synchronisation, which means that we synchronise the state of $\mathcal{M}$ and $\mathcal{A}$ once every second, namely every time ACAS X is invoked. The second is an *initial-state* synchronisation, where we start from the same state but then let the two systems evolve independently.

For $\mathcal{A}$, the geometric state and the corresponding τ -distribution are obtained by the state estimation component of ACAS X. Recall Figure 4.1 on page 77. The “surveillance and tracking module” processes sensor information which is used to provide the state estimation. Then, this state estimation is subject to the previously described interpolation, resulting in the set of geometric states $\hat{S}_{\mathcal{A}}$. Further, recall that for $\mathcal{M}$ the time to LHS decreases by one with every step. The state distribution in $\mathcal{M}$ after one step is obtained as follows. Let us assume that at some point in encounter $\mathcal{E}$, the MDP moves from state set $\hat{S}_{\mathcal{M}}$ to state set $\hat{S}'_{\mathcal{M}}$ based on an advisory adv . Hence,

$$\hat{S}'_{\mathcal{M}} = \{s' \in S \mid s \in \hat{S}_{\mathcal{M}} \wedge s \xrightarrow{\text{adv}} \mu \wedge \mu(s') > 0\}.$$

Then the probability $p_{\mathcal{M}}(s')$ of each state $s' \in \hat{S}'_{\mathcal{M}}$ is obtained as follows. For each $s \in \hat{S}_{\mathcal{M}}$ let $p_{\mathcal{M}}(s, s')$ denote the probability associated with the transition

Figure 4.5: State conformance between ACAS X and the MDP.

from s to s' with advisory adv in the MDP:

$$p_{\mathcal{M}}(s, s') = p_{\mathcal{M}}(s) \cdot \mu(s') \text{ with } s \xrightarrow{\text{adv}} \mu.$$

Note that $p_{\mathcal{M}}(s, s') = 0$ means that s cannot transition to s' in a single step. Then the probability to be in state s' after one step is given by

$$p_{\mathcal{M}}(s') = \sum_{s \in \hat{S}_{\mathcal{M}}} p_{\mathcal{M}}(s, s').$$

Decision making, i.e. the advisory that is issued, is based on the LUT. The LUT used by $\mathcal{M}$ is identical to that used by $\mathcal{A}$ for ACAS X deployment. For this reason, we compare states of $\mathcal{M}$ and $\mathcal{A}$ in terms of their interpolated states within the LUT. Figure 4.5 illustrates an example of a stepwise synchronisation between $\mathcal{M}$ and $\mathcal{A}$. $\mathcal{M}$ and $\mathcal{A}$ start at the same set $\hat{S}_{\mathcal{A}}$ of geometric states, which has been estimated during operation of $\mathcal{A}$. Hence, the initial set of states in the MDP is given by $\hat{S}_{\mathcal{M}} = \hat{S}_{\mathcal{A}}$. Note that we only illustrate in this example the possible states in which the system may be, without including the probabilities associated with those states. At the next ACAS X cycle, one second later, the grey state set $\hat{S}'_{\mathcal{M}}$ is computed based on MDP transitions starting from $\hat{S}_{\mathcal{A}}$, whereas the blue set of states $\hat{S}'_{\mathcal{A}}$ is computed based on sensor and transponder information.

4.3.2 Conformance relations

Based on the above setup, we can define expectations for model and actual system behaviour. If $\mathcal{M}(\mathcal{E})$ is the behaviour of the model $\mathcal{M}$ on encounter $\mathcal{E}$, and $\mathcal{A}(\mathcal{E})$ the behaviour of ACAS X on $\mathcal{E}$, conformance establishes a relation $\mathcal{C}$ between the two behaviours: $\mathcal{C}(\mathcal{M}(\mathcal{E}), \mathcal{A}(\mathcal{E}))$. For the ACAS X system, conformance can be defined at three distinct levels: NMAC conformance, advisory conformance, and state conformance.

The NMAC conformance metric is the coarsest one and just focuses on the dangerous NMAC states: whenever $\mathcal{A}(\mathcal{E})$ encounters an NMAC, $\mathcal{M}(\mathcal{E})$ also has to encounter an NMAC, and vice versa.

At the next level of detail, we focus on the interactions of ACAS X with the outside world, namely the sequence of advisories. Model and implementation are conformant if the sequence of advisories issued are the same or compatible, where the notion of compatibility has to be specified. For example, we may tolerate small deviations in the time at which an advisory is issued.

A state conformance requires that, at each point in time, the state of $\mathcal{M}$ and $\mathcal{A}$ are compatible, where compatibility may take into account state distributions. For simplicity, we first consider the geometric states of $\mathcal{M}$ and $\mathcal{A}$, without taking into account their associated probabilities, as illustrated in Figure 4.5. Each state can be viewed as a point in a high-dimensional state space, where each dimension corresponds to one of the state variables in the geometric state.

One measure of conformance is to expect that the system states are fully contained in the model states, which we call *full state conformance* (left box in Figure 4.5):

$$\hat{S}'_{\mathcal{A}} \cap \hat{S}'_{\mathcal{M}} = \hat{S}'_{\mathcal{A}}.$$

Intuitively, this means that the model anticipates all the possible evolution's of the encounter in a single step.

Whereas full conformance represents the ideal situation, a partial overlap might also be acceptable, leading to a notion of *partial state conformance* (centre box in Figure 4.5):

$$\hat{S}'_{\mathcal{A}} \cap \hat{S}'_{\mathcal{M}} \neq \emptyset.$$

For this particular case, and given that states are weighted, partial conformance relations may also take these weights into account in order to evaluate the significance of the overlap.

Finally, if the model and system states are disjoint, we have an undesirable *no state conformance* situation (right box in Figure 4.5):

$$\hat{S}'_{\mathcal{A}} \cap \hat{S}'_{\mathcal{M}} = \emptyset.$$

Decision making in ACAS X is based on how the MDP expects encounters to evolve. Such an extreme mismatch between the MDP and the actual system may result in providing advisories that will not prevent an NMAC from occurring. Figure 4.5 (bottom part) illustrates these three types of conformance.

Note that “no state conformance” need not directly result in a violation of ACAS X requirements for the encounter that is being studied; it is possible that NMAC just happens to be avoided. However, detecting non-conformance in a tested encounter is still valuable since it indicates a mismatch that may result in dangerous behaviour in other encounters that are not included in the testing. As such, conformance relations might be useful for predictive runtime analysis of systems.

4.4 Analysing conformance issues

We have applied a variety of conformance metrics to ACAS X test data that we obtained from the ACAS X team as part of the ACAS X Run 13 distribution. We ran both initial-state and stepwise variants of these metrics. Overall, we

observed several conformance issues that need to be studied more carefully to determine their significance. For example, Figure 4.6b illustrates three bars of advisories at the bottom. The first bar is for the actual system, the second for the interpolated system states (i.e. ACAS X without on-line advisory corrections) and the third one for the model with a stepwise synchronisation of the MDP and the system. We can observe that the same advisories are produced but in the third case the descend advisory is issued much earlier.

Among all the observations we made through our experiments, the cases that we believe need the most immediate investigation are cases of stepwise non-conformance. In fact, we did not anticipate that such mismatches between model and system might be possible within a single step. In this section, we analyse one of the cases of non-conformance.

4.4.1 A non-conformance encounter

Let us consider as an example encounter 86 within the official ACAS X distribution referred to as Run 13. Figure 4.6a shows a 3D representation of the flight path for each of the involved aircraft, where the ownship is represented by the blue line and the intruder by the red line, respectively. The encounter starts when the two aircraft are still safely apart (locations marked by small circles). As soon as the ACAS X system on the ownship detects a potentially dangerous development (at $t = 19$ s) into the encounter, an alarm “Descend, Descend!” is annunciated in the cockpit and an advisory to descend with 1500 ft/min (DES1500) is issued (Figure 4.6b, red line). After a short delay, the pilot reacts and follows that advisory and the vertical velocity of the ownship becomes negative. The vertical speed of both aircraft are shown in Figure 4.6b, middle panel. At time $t = 29$ s, ACAS X advises the pilot to not climb (DNC, Figure 4.6B, green line) and the pilot levels off. As soon as the danger of an NMAC has been averted, a COC advisory is given ($t = 45$ s) and the encounter ends successfully.

We checked state conformance on this encounter. Recall that state conformance synchronises the geometric model state to that of the system at each step, and then compares the geometric states of the system and model in the next step. Given that the value of sRA is always the same in this encounter, we focus on the remaining three state variables: z_{rel} , dz_o , and dz_i . The upper panel of Figure 4.6c displays the relative state conformance, i.e. the degree of overlap for $\hat{S}'_{\mathcal{A}} \cap \hat{S}'_{\mathcal{M}}$ with:

- $C_{\text{rel}} = 1$: full state conformance;
- $0 < C_{\text{rel}} < 1$: partial state conformance;
- $C_{\text{rel}} = 0$: no state conformance.

The bottom panel of Figure 4.6c displays the weighted state conformance, i.e. the probability mass of the overlapping states. A higher value of C_w represents a more precise prediction by the model.

Figure 4.6d-4.6f illustrates the set of states for model $\mathcal{M}$ as a grey cloud and the set of states for system $\mathcal{A}$ as a blue cloud, respectively. Each cloud contains

Figure 4.6: Visualisation of non-conformance encounter: (a) trajectories of ownship (blue) and intruder (red). Circles mark the starting-points; (b) ACAS X data showing (top to bottom) relative altitude z_{rel} over time, vertical velocities (dz_o and dz_i) and sequence of produced advisories by ACAS X, interpolated system states, stepwise synchronised MDP. (c) Conformance metrics C_{rel} (top) and C_w (bottom). Small red x signs indicate time points $t = 11$ s, 23 s, 34 s, associated with panels (d)-(f); (d)-(f) 3D projections of $\hat{S}'_{\mathcal{M}}$ (gray) and $\hat{S}'_{\mathcal{A}}$ (blue) at $t = 11$ s, 23 s, 34 s.

the projection of states on the three geometric variables z_{rel} , dz_o and dz_i , where each figure represents a different point in the encounter. Due to uncertainties in

movement of the intruder and pilot reactions, size and shape of the overlapping parts of the blue and gray clouds vary during the encounter. Figure 4.6d and Figure 4.6e both show a partial coverage at time point $t = 11$ s and $t = 23$ s, respectively. Figure 4.6f, however, illustrates a non-conformance situation at time point 34 s in the encounter. The red plane in the figure clearly separates the two clouds. Moreover, one can see that the variable dz_o separates the two clouds. Hence, the model predicted a different change in the vertical rate of the ownship.

4.4.2 Step-wise conformance relations

To analyse non-conformance situations such as the one illustrated in Figure 4.6f, it is sometimes helpful to analyse the conformance with respect to individual variables of the state. The reason is that non-conformance may be due to the way a particular variable is modelled, which enables us to give precise and helpful information to developers. For example, the non-conformance situation of Figure 4.6 is due to variable dz_o . Note that this does not necessarily occur in all non-conformance situations. It may be that variables are covered individually, but their combination is not.

We define a metric of relative state conformance w.r.t. $\hat{S}'_{\mathcal{A}}$ and $\hat{S}'_{\mathcal{M}}$ as follows:

Definition 4.4 (Relative state conformance). Let $\hat{S}'_{\mathcal{A}}$ be the set of geometric states (after interpolation) after a step in ACAS X $\mathcal{A}$ and $\hat{S}'_{\mathcal{M}}$ be the set of states in the MDP $\mathcal{M}$ after one step. The *relative state conformance* after one step between $\mathcal{A}$ and $\mathcal{M}$ is given by

$$C_{\text{rel}} = \frac{|\hat{S}'_{\mathcal{A}} \cap \hat{S}'_{\mathcal{M}}|}{|\hat{S}'_{\mathcal{A}}|}.$$

Hence, $C_{\text{rel}} \in [0, 1]$ where $C_{\text{rel}} = 0$ describes a non-conformance situation and $C_{\text{rel}} = 1$ full conformance. Figure 4.6c (top) shows C_{rel} for the example shown in Figure 4.6a. The non-conformance at $t = 34$ s is clearly visible.

In order to study more accurately cases of partial conformance i.e. $0 < C_{\text{rel}} < 1$, we want to consider additional information perusing the fact that $\hat{S}'_{\mathcal{A}}$ and $\hat{S}'_{\mathcal{M}}$ are distributions. Informally speaking, the higher $p_{\mathcal{A}}(s)$ of some $s \in \hat{S}'_{\mathcal{A}}$, the more weight it carries for decision making. If those important states are well represented in the MDP (with a high $p_{\mathcal{M}}$) then conformance is very good and the metric should be high. In other words, we wish to measure the similarity between the MDP and actual system information. In an ideal situation, the model and the system will contain the exact same pairs of states and corresponding weights.

Let us assume that we are comparing $\hat{S}'_{\mathcal{M}}$ and $\hat{S}'_{\mathcal{A}}$ for state conformance, where each state $s_{\mathcal{M}} \in \hat{S}'_{\mathcal{M}}$ and $s_{\mathcal{A}} \in \hat{S}'_{\mathcal{A}}$ is associated with probability $p_{\mathcal{M}}(s)$ and $p_{\mathcal{A}}(s)$, respectively. We then define a weighted state conformance metric in terms of the sum of the absolute differences between the probabilities of the states in $\hat{S}'_{\mathcal{A}}$ and $\hat{S}'_{\mathcal{M}}$ as follows:

Definition 4.5 (Weighted state conformance). Let $\hat{S}'_{\mathcal{A}}$ be the set of geometric states (after interpolation) after a step in ACAS X $\mathcal{A}$ and $\hat{S}'_{\mathcal{M}}$ be the set of states in the MDP $\mathcal{M}$ after one step. The *weighted state conformance* after one step between $\mathcal{A}$ and $\mathcal{M}$ is given by

$$C_w = 1 - C_{\text{diff}}$$

with

$$C_{\text{diff}} = \frac{1}{2} \sum_{s \in (\hat{S}'_{\mathcal{A}} \cup \hat{S}'_{\mathcal{M}})} |p_{\mathcal{A}}(s) - p_{\mathcal{M}}(s)|.$$

The sum is divided by 2, which represents the maximum possible divergence (*mpd*) between the sets. Indeed, in the presence of non-conformance, the probability differences will add up to 1 for the model, and 1 for the system, for a total of 2:

$$\begin{aligned} mpd &= \sum_{s \in (\hat{S}'_{\mathcal{A}} \cup \hat{S}'_{\mathcal{M}})} |p_{\mathcal{A}}(s) - p_{\mathcal{M}}(s)| \\ &= \sum_{s \in \hat{S}'_{\mathcal{A}}} |p_{\mathcal{A}}(s) - p_{\mathcal{M}}(s)| + \sum_{s \in \hat{S}'_{\mathcal{M}}} |p_{\mathcal{A}}(s) - p_{\mathcal{M}}(s)| \\ &\stackrel{(a)}{=} \sum_{s \in \hat{S}'_{\mathcal{A}}} p_{\mathcal{A}}(s) + \sum_{s \in \hat{S}'_{\mathcal{M}}} p_{\mathcal{M}}(s) \\ &= 1 + 1 = 2 \end{aligned}$$

where for (a) it holds that $p_{\mathcal{M}}(s) = 0$ for $s \in \hat{S}'_{\mathcal{A}}$ and $p_{\mathcal{A}}(s) = 0$ for $s \in \hat{S}'_{\mathcal{M}}$ if $\hat{S}'_{\mathcal{A}} \cap \hat{S}'_{\mathcal{M}} = \emptyset$. Note that when a state does not belong to a set of states, we represent it as its probability being 0 for that set. $C_w = 1$ corresponds to $C_{\text{diff}} = 0$, i.e. the sets have the same states with the same associated probabilities. Full conformance situations only have a high value of C_w if the probability mass of the model lies mostly within the subset that is covered by the actual system, and if the states in that subset are weighted similarly to their corresponding ones in the actual system. Figure 4.6c (bottom) shows C_w for our example encounter.

Since the case of non-conformance seemed the most urgent to report to the ACAS X team, we focused on studying non-conformance cases, aiming at providing useful information for the developers to examine the issue. Since non-conformance is very rare in the test data that we received as part of the ACAS X release, we decided to focus on generating additional encounters exhibiting non-conformance, using machine learning.

4.5 Automatic Generation of Non-Conformance Encounters

The complexity of the ACAS X input domain makes it hard to explore it systematically. We therefore based our initial experiments on test encounters prepared

by the developers of ACAS X that were included with the ACAS X distribution. When measuring state conformance on those, we encountered only a handful of situations where non-conformance exists. However, those situations are a hint that there are discrepancies between the real world evolution based on ACAS X and the corresponding MDP model. For a more thorough analysis of this phenomenon a much larger data set is required.

In this section, we propose techniques for the automated generation of non-conformance scenarios. In addition to generating such scenarios, we want to be able to provide constraints that further filter the encounters to the most interesting and safety-critical cases. For example, we wish to focus on situations where an actual advisory is issued because of the close proximity of the aircraft.

4

4.5.1 The scenario generation environment

Our approach extends the RLESCAS (Reinforcement Learning Encounter Simulator for Collision Avoidance Systems) package for adaptive stress testing of airborne collision avoidance systems [LKMB015]. RLESCAS uses Monte Carlo Tree Search (MCTS) to automatically generate two-aircraft NMAC encounters.

Figure 4.7 illustrates how our framework extends RLESCAS for the case where a single aircraft is equipped with ACAS X. Our framework implements two main changes to the original framework. First, it introduces the MDP in order to compute information needed for the conformance relations. Second, it modifies the reward function to favour the generation of low-conformance encounters.

The original framework relies on a simulator for aircraft encounters; in our case one of the aircraft is equipped with ACAS X. Inputs to this simulation environment are basic simulator controls, like initialise, and a seed. These are the only variables that the framework is able to manipulate in order to target specific types of encounters.

Our extension intercepts the operation of the ACAS X component within the simulation to obtain interpolated states that can be fed to the MDP. Then the MDP and the LUT are used as described in the previous sections to calculate conformance data. The output of the framework is the likelihood of the current transition, a variable describing the conformance of the current state, and the weighted conformance metric. The outputs are subjected to our modified reward function used by the MCTS algorithm. The result of the MCTS is in turn used to choose the seed and control inputs for the simulator. A detailed description of the original RLESCAS and the MCTS algorithm can be found in [LKMB015].

4.5.2 The reward function

The encounter generation framework aims at maximising the reward function. Therefore, our reward function $\mathcal{R}$ must be designed to reward the occurrence of non-conformance, but it must also be able to provide guidance on how to evaluate the current situation.

We start with the reward function given in [LKMB015] and gradually modify it to serve the purposes of our framework. The objective of the function in

Figure 4.7: Encounter generation framework. The white box at the top is RLESCAS for one aircraft equipped with ACAS X, and the orange box is our extension.

Figure 4.8: RLESCAS extension for the encounter generation.

[LKMB015] is to find high probability encounters that contain NMAC events:

$$\mathcal{R}(s_t, s_{t+1}) = \begin{cases} 0 & \text{if } s_t \in \text{NMAC}, \\ -\infty & \text{if } s_t \notin \text{NMAC}, t \geq T, \\ \log(p(s_t, s_{t+1})) & \text{if } s_t \notin \text{NMAC}, t < T. \end{cases} \quad (4.1)$$

The reward for going from a state s_t to s_{t+1} depends on two main events: 1) if an NMAC occurs ($s_t \in \text{NMAC}$), and 2) if the maximum simulation time has been reached ($t \geq T$). Here T is set to the time horizon of ACAS X, which is 50s in the RLESCAS framework. Reaching T therefore indicates a terminal state in our framework; all NMAC situations, if any, have to occur by that time.

The first two conditions of $\mathcal{R}$ represent the NMAC occurrence constraint. If an NMAC occurs a maximal reward is issued, whereas if the time horizon T has been reached and no NMAC has occurred, an infinite penalty is issued. In all other cases a reward based on the probability to be in the current state is issued to maximise the likelihood of the encounter. Note that this function assigns negative rewards, in other words, penalises undesirable situations to a higher or lower degree, and assigns 0 to the desired outcome.

To adapt the reward function for the generation of non-conformance encounters we investigate variations of Equation (4.1). As a first attempt, our objective is similar to that of the original reward function, but for non-conformance (NC) instead of NMAC events. Our reward function then infinitely penalises the learner when no non-conformance event is encountered.

However, we introduce a change for the evaluation of intermediate situations, because we want to generally encourage mismatches between the system and the MDP. We do so by rewarding small intersections between system and model states, i.e. partial conformance (the smaller the intersection the better):

$$\mathcal{R}(s_t, s_{t+1}) = \begin{cases} nc & \text{if } s_t \in \text{NC}, \\ 0 & \text{if } s_t \notin \text{NC}, t \geq T, \\ (1 - C_{\text{rel}}) \cdot pc & \text{if } s_t \notin \text{NC}, t < T. \end{cases} \quad (4.2)$$

Hence, $\mathcal{R}$ is geared towards finding encounters with a non-conformance event (NC) and with a low conformance metric throughout the encounter. Note that instead of using negative rewards as in Equation (4.1) this function uses positive rewards. We define two positive reward parameters nc and pc representing the reward for non-conformance events and partial conformance events, respectively. We parameterise the reward function in this fashion in order to be able to experiment with different levels of relative importance to the two aspects of the targeted encounters. Parameter pc is weighted by $(1 - C_{\text{rel}})$, which represents the ratio of system states that are not covered by the model. If no non-conformance event occurred, a reward of 0 is issued.

Applying the reward function given in Equation (4.2) in the MCTS algorithm enables us to generate non-conformance encounters. However, we observed that in many cases, the altitude difference between the two aircraft remained high, so ACAS X never issued any advisories other than COC. Such encounters are not very interesting for our study.

EC #	time point	$C_{\text{rel} z_{\text{rel}}}$	$C_{\text{rel} dz_o}$	$C_{\text{rel} dz_i}$	$C_{\text{rel} sRA}$
1	24	16/16	16/16	0/16	16/16
	49	26/30	30/30	0/30	30/30
6	19	10/12	12/12	0/12	12/12
	33	24/28	28/28	0/28	28/28
	35	28/28	28/28	4/28	28/28
	45	20/20	20/20	0/20	20/20
9995	35	40/40	40/40	0/40	40/40
	40	24/28	28/28	0/28	28/28
	42	24/24	12/24	4/24	24/24

Table 4.2: Subset of generated non-conformance encounters, including conformance information per state variable.

The natural next step is then to find a reward function that combines objectives from Equations (4.1) and (4.2). The objective of our new reward function is to trigger non-conformance, involve low conformance, and minimise the altitude difference between the aircraft at the time of closest approach.

$$\mathcal{R}(s_t, s_{t+1}) = \begin{cases} nc & \text{if } s_t \in \text{NC}, \\ fc & \text{if } s_t \in \text{FC}, \\ (1 - C_w) \cdot pc & \text{if } s_t \in \text{PC}, t < T, \\ -z_{\text{rel}} & \text{if } s_t \notin \text{NMAC}, t \geq T. \end{cases} \quad (4.3)$$

Like before, nc and pc are positive parameters, and we introduce parameter fc , which is negative or 0. Function $\mathcal{R}$ now includes both positive and negative rewards. It penalises encounters with no NMAC with the relative distance, whereas positively rewards partial and non-conformance. Partial conformance (PC) is weighted by $(1 - C_w)$ to encourage a low probability match between the model and system states. Full conformance (FC) receives a negative or 0 reward. Note that, in order to increase the likelihood of such encounters, one could add to the reward the probability to be in the current state, similarly to Equation (4.1).

4.5.3 Analysis of generated non-conformance encounters

We used reward function (4.3) in our framework to generate 18 encounters with a total of 33 non-conformance events. Table 4.2 displays three of these encounters and decomposes non-conformance events into conformance of individual state variables, to identify whether mismatches are associated with particular variables, as discussed in Section 4.4.

We use $\mathcal{P}_{z_{\text{rel}}}(S)$, $\mathcal{P}_{dz_o}(S)$, $\mathcal{P}_{dz_i}(S)$ and $\mathcal{P}_{sRA}(S)$, to denote the sets obtained by projecting each state $s \in S$ onto its variable z_{rel} , dz_o , dz_i , and sRA , respectively. Then for system state $\hat{S}_{\mathcal{A}}$ and model state $\hat{S}_{\mathcal{M}}$, conformance relative to each state variable x is defined as:

$$C_{\text{rel}|x} = \frac{|\mathcal{P}_x(\hat{S}_{\mathcal{M}}) \cap \mathcal{P}_x(\hat{S}_{\mathcal{A}})|}{|\mathcal{P}_x(\hat{S}_{\mathcal{A}})|}$$

Figure 4.9: Encounter 9995 generated with reward function from Equation (4.3). Thin red lines indicate non-conformance at $t = 35$ s, 40 s, 42 s.

Table 4.2 and Table 4.3 indicates that for the wide majority of non-conformance events, the states deviate in variable dz_i . We further analysed some of these encounters to gain intuition of the types of characteristics that may be causing non-conformance. Consider Figure 4.9b visualising the encounter of two aircraft w.r.t. their altitude over time, for example. The corresponding non-conformance time points of this encounter are at 35, 40 and 42 seconds into the encounter. Inspecting the altitude changes of the intruder in the interval of $[35, 42]$ reveals a sudden change from descend to a relatively strong climb. This sudden and steep altitude change is not reflected in the MDP.

Even though in such encounters the behaviour of ACAS X appears reasonable, it is important to study non-conformance occurrences closely, in case they trigger problematic decision-making under potentially rare circumstances. We are currently in the process of examining these results together with the ACAS X team in order to determine whether model fine-tuning may be beneficial, and whether the root cause is in the continuous model or its discretisation.

4.6 Related work

Essen and Giannakopoulou developed the Verica tool and applied probabilistic verification and synthesis to an early version of ACAS X [EG14; EG16]. Their aim was to study the impact of design issues such as model discretisation and the selection of costs for the dynamic programming.

Jeannin et al. [Jea+15] analysed ACAS X using hybrid models including the dynamic aircraft behaviour and discrete pilot decisions. To verify if ACAS X is safe under certain assumptions they performed analysis using hybrid systems theorem proving techniques. Therefore, they used the KeYmaera tool to compute safe regions for restricted types of encounters and for a single advisory. Safe regions characterise the types of advisories that are safe for the corresponding

EC #	time point	$C_{\text{rel}} z_{\text{rel}}$	$C_{\text{rel}} dz_o$	$C_{\text{rel}} dz_i$	$C_{\text{rel}} sRA$
1	24	16/16	16/16	0/16	16/16
	49	26/30	30/30	0/30	30/30
2	35	28/32	32/32	0/32	32/32
	49	16/16	16/16	0/16	16/16
3	36	10/10	10/10	0/10	10/10
4	36	20/24	24/24	0/24	24/24
	41	28/28	28/28	0/28	28/28
6	19	10/12	12/12	0/12	12/12
	33	24/28	28/28	0/28	28/28
	35	28/28	28/28	4/28	28/28
	45	20/20	20/20	0/20	20/20
9	33	16/16	16/16	0/16	16/16
	48	24/28	28/28	0/28	28/28
10	34	16/16	16/16	0/16	16/16
	41	12/16	16/16	0/16	16/16
11	45	16/16	16/16	0/16	16/16
9100	28	10/10	10/10	0/10	10/10
	34	16/16	16/16	0/16	16/16
9101	38	22/30	30/30	0/30	30/30
9904	42	20/20	20/20	0/20	20/20
9931	23	14/14	14/14	0/14	14/14
9982	41	16/16	8/16	0/16	16/16
9995	35	40/40	40/40	0/40	40/40
	40	24/28	28/28	0/28	28/28
	42	24/24	12/24	4/24	24/24
9969	42	8/8	8/8	0/8	8/8
	49	8/8	8/8	0/8	8/8
9997	24	10/10	10/10	0/10	10/10
9999	46	16/24	24/24	0/24	24/24
10000	22	8/8	8/8	0/8	8/8
	31	12/14	14/14	0/14	14/14
	44	10/10	10/10	0/10	10/10
86 (run 13)	34	40/40	0/40	40/40	40/40

Table 4.3: Full set of generated non-conformance encounters, including conformance information per state variable.

encounter. ACAS X advisories for specific encounters can then be compared against their corresponding safe regions. The advantage of taking a hybrid approach is that it does not require discretisation. However, the entire hybrid model for ACAS X is prohibitively large, which forced the authors to work with a restricted number of scenarios.

Other researchers [LL97; TPS98; PC09; LRP13; Gho+14; GMA07] have investigated hybrid techniques and theorem proving for other collision avoidance

systems. Some researchers have developed testing frameworks for automated air-traffic control [GBSEH11; Gia+14; DG15]. In order to evaluate the performance of ACAS X, the ACAS X team heavily relies on the simulation of a large number of encounters including recorded flight data.

More broadly, our work is related to several disciplines, such as model-based design and testing, runtime monitoring, abstraction, and model validation. Model-based design uses models to describe, analyse, and generate code for a software system. For example Tretmans [Tre08] expresses models as labelled transition systems (LTSs) and compliance in the *ioco* implementation. Thus the implementation under test is represented by the model describing the required behaviour and tested for compliance defined in *ioco*. This approach lead to a testing framework for LTSs introduced by Timmer et al. [TBS11] and implemented in the JTorX tool by Belinfante [Bel14].

In our work, models are abstractions of the real-world that software algorithms use for decision-making. Our conformance relations can be viewed as requirements that can be monitored at runtime, and be used for predictive analysis. These requirements are aimed particularly at establishing model quality. With respect to abstraction and model validation, our work develops metrics for validation of an abstraction in the context of its use for decision making. In other words, our conformance relations set application-specific requirements for an abstraction.

4.7 Conclusion

Autonomy requires models for decision making, adaptation and self-healing. Safety-critical autonomous systems demand the development of new methods for establishing trust in these models.

In this chapter, we explored and applied several model conformance criteria in the context of the ACAS X collision avoidance system, and discovered some rare cases of non-conformance. We used machine learning to automatically generate additional encounters that exhibit non-conformance, and were able to identify potential causes for this issue.

In the context of ACAS X it is important to develop approaches that help to prioritise the issues that our techniques report. Moreover, additional reward functions for encounter generation can help to narrow the set of non-conformance findings based on criticality. Further, statistical learning techniques for the analysis of conformance issues can be applied to help developers with debugging their models.

Part II

Fault maintenance trees

Fault trees: The basics

INNOVATIONS in IT systems like autonomous cars, the Internet-of-Things and robotics rapidly increase our dependence on computer-based systems. That is, systems in our daily life are controlled as well as monitored by computer-based algorithms and are known as *Cyber Physical Systems* (CPSs) [Alu15]. They are ubiquitous in our daily life and govern critical systems like nuclear power plants, smart grids and medical devices. The common denominator of all CPSs is that they have to operate in a safe and reliable manner. Hence, they have to be dependable. Moreover, stakeholders require substantial facts and figures regarding system performance, risks and costs to support asset management as well as to demonstrate compliance. *Reliability engineering* is a field that provides methods, tools and techniques to evaluate and mitigate the risks related to complex systems [RH04]. One of the most important techniques in that field is fault tree analysis (FTA) [VGRH81; Sta+02], a prominent assessment method in industries ranging from aerospace over railways to nuclear power plants. Moreover, it is standardised by the IEC [IEC61025] and companies and institutions like Airbus, ESA, NASA, FAA, Honeywell, etc. are deploying FTA as a crucial tool in their risk management process.

Fault trees (FTs) specify the failure behaviour of a system in terms of its components and their failure propagation throughout the system. The leaves of the tree are called *basic events* (BE) and represent component failures; The other nodes are called *gates* and indicate how component failures propagate through the system, and lead to system level failures. Standard (or: static) FTs (SFTs) contain basic logic gates (excluding negation), like AND and OR, which makes them easy to use and to analyse. However, they are also limited in expressiveness. One can say that SFTs represent the bare-bones of FTs, such that they have to be adapted to better represent the area of application in industry. For example, to cater for more sophisticated dependability patterns, like spare management and causal dependencies, a number of extensions to FTs have been proposed. Dynamic fault trees (DFTs) are the most common extension to FTs and widely used. They were presented by Dugan et. al. [DBB92] to model fault-tolerant computer systems. Since then a plethora of variants of fault trees have been proposed in literature and [RS15] provides a thorough overview of several FT formalisms.

FTs provide an easy to use framework for engineers to graphically model

possible system failures through a top to bottom design. Hence, when modelling a FT one starts with the system failure of interest and gradually unfolds the failure causes and connections down to the level of single component failures. The creation of a FT during the design phase of a system can already expose vulnerabilities of the system such that precautions can be taken. Moreover, one can apply qualitative as well as quantitative analysis techniques on the FT. The former can be used to identify high risk components in the system, while the latter can be used to calculate important key performance indicators, like the reliability of a system.

This chapter provides a basic introduction to static fault trees (SFTs) and dynamic fault trees (DFTs). Furthermore we give a detailed overview of all DFT elements and provide a formal definition. Besides, we define structural properties on DFTs to prevent undesired behaviour. The first question we will answer is: *What can we do with a DFT?* Therefore, we introduce important key performance indicators of dependable systems and how they relate to DFTs. For example the reliability w.r.t. a DFT can be described by the probability to reach the top level node. In particular, we provide information about three important stochastic measures: the mean time to failure; the reliability; and the availability. These measures lead us to the following question: *How can we analyse those stochastic measures on a DFT?* The approach we take is to extract an underlying stochastic model from the DFT. There exists a variety of different approaches to obtain a stochastic model from a DFT, e.g. Bayesian networks [BD06] or stochastic Petri nets [CR05]. In this work we follow the approach of Boudali et al. [BCS10], where we use input/output Markov automata (I/O-MA). One feature of I/O-MA is that they are compositional, i.e. the I/O-MA model of a DFT can be constructed by composing smaller I/O-MA models representing the various DFT elements. Thus, we will show how to translate a DFT into an I/O-MA in a compositional way. Moreover, we show how algorithms presented in Section 3.2 can be utilised to compute stochastic metrics on DFTs.

Summarising, this chapter presents the basic concepts of fault trees and their dynamic extension. Therefore, we first give an informal overview of the various gates and basic components and explain their behaviour. Subsequently we provide formal definitions for DFTs as well as formulate conditions that a DFT should not violate to be eligible to be formally analysed. After formally defining DFTs we present how to utilise algorithms of I/O-MAs to analyse key performance metrics on DFTs, e.g. the reliability for a specified operation time. After establishing what we can analyse, we give a detailed description over the actual semantics of DFTs in terms of I/O-MAs. Thus, we provide I/O-MAs for all DFT components as well as describe how they are composed with each other. Finally we discuss related work and show how the I/O-MA semantics can be adapted for various DFT extensions.

Origins of the chapter. This chapter introduces SFTs and DFTs as well as their semantics in terms of I/O-MAs and is partly based on

- F. Arnold, A. F. E. Belinfante, F. I. Van der Berg, D. Guck, and M. I. A. Stoelinga. “DFTCalc: a tool for efficient fault tree analysis”. In:

Proceedings of the 32nd International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2013). Vol. 8153. Lecture Notes in Computer Science. Springer Verlag.

Organisation of the chapter. We introduce fault trees in Section 5.1 and dynamic fault trees in Section 5.2. Section 5.3 provides a formal definition of DFTs and Section 5.4 introduces key performance indicators for dependability. In Section 5.5 we describe how to analyse DFTs. Section 5.6 introduces the semantics of DFTs in terms of I/O-MAs followed by related work and discussion of different extensions for DFTs in Section 5.7.

5.1 Static fault trees

A *fault tree* (FT) is a tree, or rather a directed acyclic graph, describing how a system fails depending on some basic component failures in the system. In other words, an FT ranges over a set of *basic events* (BEs) that typically describe some low level system failures like “Battery is empty” or “CPU overheat”. The FT then encodes the system failure by connecting the BEs via *gates* to a *top-level event*, e.g. “Cloud service failure”. Hence, a gate expresses how component failures induce a system failure and consist of one or more inputs, and one output.

Figure 5.1: FT for a simplified laptop failure.

Example 5.1. The FT in Figure 5.1 represents a simplified failure behaviour of a laptop, consisting of three subsystems: the power supply, the graphical output and the processing unit. The laptop will fail if one of the three subsystems fail. The power supply fails if the battery as well as the external power fails. The external power fails if the power adaptor fails or there is a power outage. The graphical output fails if the monitor or the GPU have a failure. For a failure of the processing unit the CPU has to fail or both memory modules need to fail.

Figure 5.2: Static fault tree gates.

BEs in static fault trees. A BE can be in two different modes, *active* and *failed*. Initially a BE is in active mode and after a failure occurs it switches into its failed mode. To describe the failure, BEs are usually equipped with a *failure rate*, i.e. the parameter of a negative exponential distribution indicating the probability for the component to fail within a time bound T . Thus, the probability that a component with failure rate λ fails within time bound T is given by

$$\mathbb{P}[\text{Fail} \leq T] = 1 - e^{-\lambda T}. \quad (5.1)$$

Gates in static fault trees. *Static* fault trees (SFTs) feature three types of gates, depicted in Figure 5.2, OR, AND and VOT(k).

- (a) The OR gate fails when at least one input fails.
- (b) The AND gate fails when all of its inputs fail.
- (c) The VOT(k) gate fails when at least k out of n inputs fail.

Often, SFTs also provide an INHIBIT gate to describe a condition under which the failure would occur. For example, the failure of a laptop is only discovered when it is used. Thus, an INHIBIT gate fails if the input occurs as well as the attached condition is satisfied. Since its behaviour is equivalent to the AND gate, we will not treat it as a separate gate.

The root of a FT is called the *top level event* (TLE) and represents the failure condition of interest, e.g. the laptop failure from Example 5.1. A FT fails if the TLE fails. For SFTs the failure of the TLE is determined by an unordered set of failed BEs.

Analysis of SFTs. The tree structure of an SFT provides a fast visual feedback of the system's failure behaviour. Therefore, by modelling an SFT during the design phase it can help to identify design flaws. *Qualitative* analysis techniques are used to detect vulnerable parts of the SFT by providing more insights on the structure. The most prominent qualitative analysis technique is to compute *minimal cut sets* (MCSs) [LGTL85]. A *cut set* is a set of BEs that can cause the system to fail. A cut set is minimal, i.e. a MCS, if no proper sub set is a cut set. A small MCS, i.e. a MCS with a small number of BEs, or a MCS with BEs that have a high failure probability, indicates unreliable parts in the fault

tree. Hence, if an SFT contains such MCSs, they can be used to gain better insights on which part of the system is more likely to fail and how to improve the system.

Example 5.2. Consider the SFT in Figure 5.1 and the cut set {Power, Power adaptor, Battery}. If this cut set fails, then the top level event of “Laptop failure” will occur. However, the cut set is not minimal, since it contains true subsets that already let the top level event fail. These minimal cut sets are given by {Power, Battery} and {Power adaptor, Battery}.

For more information about qualitative analysis we refer to the classical methods by Lee et al. [LGTL85] and a survey of state of the art FTA by Ruijters and Stoelinga [RS15].

Moreover, SFTs can also be subject to *quantitative* analysis techniques, i.e. techniques that compute metrics like the reliability of the system. More information about quantitative analysis techniques is given in Section 5.5.

5.2 Dynamic fault trees

While SFTs appeal to be very simplistic by using simple logic gates and only focus on the failure of components but not their order, they are a useful modelling tool. However, they lack several important failure patterns required for safety-critical systems, such as:

- *Order-dependent failures*, i.e. failures that have to occur in a specified order. For example, consider a short circuit in a pump. This failure occurs after a valve failure leads to a leakage. However, this must occur before a power outage to cause the short circuit.
- *Spare management and spare modules*. A spare module is a part in a system which can be used to replace a failed element, e.g. a spare car tyre. A spare module can be a *single component* or a *whole sub-system*. Moreover, a spare module is inactive when not in use. Depending on the spare module its failure rate can be reduced while inactive, i.e. it has a *passive* failure behaviour.
- *Common cause failures and feedback loops*, which invoke failures of a dependent events based on some trigger event. Although it is possible to model systems with feedback loops with SFTs, it requires verbose workarounds and makes modelling error prone [Sta+02].

Dynamic fault trees (DFTs) introduce three additional gates, namely the PAND, SPARE and FDEP gate, extending SFTs with the above mentioned behaviour. A major difference in the failure behaviour to SFTs is that components, in particular spare modules, can be inactive. Accordingly, if a spare module — like a spare car tyre — is not in use, it is considered passive and can have a different failure rate. Since DFTs support passive failure behaviour, the failure modes of BEs is also extended.

Figure 5.3: Railway crossing DFT.

Example 5.3. The DFT in Figure 5.3 represents a (simplified) failure behaviour of a railway level crossing [GKSLR14], consisting of three subsystems: the sensors, the barriers and the controller. The crossing fails if either of these subsystems fails, as indicated by the top level OR gate “Level crossing failure”. The sensor system fails if at least two out of the four redundant sensors fail, as modelled by the VOT(2,4)-gate “Sensors failure”. Furthermore, there can be a detection problem due to a disconnection of the cables, making all sensors unavailable. This is modelled by the FDEP-gate “No detection”. The trigger “Disconnection” causes the failure of its dependent events Sensor₁–Sensor₄. Finally, the barriers fail if either the main and spare motor fail, modelled by the SPARE-gate “Motors”, or if the switch and then a motor fails, modelled by the PAND-gate “Switching unit”. Thus, the barrier failure encodes a spare management with two motors as well as a order-dependent failure with the switch. In more detail, the PAND gate “Switching unit” describes the order-dependency of the motor and switch failures and the SPARE gate Motors describes the spare management of the motors. We will elaborate the dynamic gates in Section 5.2.2.

5.2.1 BEs in dynamic fault trees

The failure rate for a BE in a DFT is equivalent to the failure rate in a SFT. However, the BE of a DFT can also be inactive and therefore be in a *dormant* mode. This is typical for spare components like a spare tyre in the trunk of a car. Hence there are three different modes for a BE in a DFT: *dormant*, *active* and *failed*. Equivalent to SFTs, the component is in active mode when it is in use and it is in failure mode when it breaks down. The component is in dormant mode if it is not in use. In this mode, the failure rate is decreased by a dormancy factor $\alpha \in [0, 1]$. Hence, if the failure rate of a BE is λ its dormant failure rate is given by $\alpha \cdot \lambda$. Recall the component failure of a BE given in Equation (5.1). Thus, the probability that a dormant component fails within time bound T is given by

$$\mathbb{P}[\text{fail} \leq T] = 1 - e^{-\alpha\lambda T}. \quad (5.2)$$

Depending on the dormancy factor a dormant BE is categorised into a *cold* BE, *hot* BE or *warm* BE. A cold BE cannot fail and is represented with the dormancy

Figure 5.4: Dynamic fault tree gates.

factor $\alpha = 0$. On the other hand, a hot BE behaves equivalent to an active BE and has dormancy factor $\alpha = 1$. A warm BE has a reduced failure rate induced by a dormancy factor $\alpha \in (0, 1)$.

5.2.2 Gates in dynamic fault trees

DFTs feature all SFT gates and three additional gates, depicted in Figure 5.4, the PAND gate, SPARE gate and FDEP gate.

- (a) The **PAND** gate fails when all of its inputs fail from left to right.
- (b) The **SPARE** gate consists of a primary input and one or more spare inputs. At system start, the primary is active, and the spares are in dormant mode. When the primary input fails, one of the spare inputs is activated and replaces the primary. If no more spares are available, the SPARE gate fails. Note that a spare component can be shared among several spare gates.
- (c) The **FDEP** (functional dependency) gate consists of one trigger event and several dependent events. When the trigger event occurs, all dependent events fail. The FDEP has a dummy output, which is represented by a dotted line and ignored in calculations.

An overview of all gates considered throughout this chapter is given in Table 5.1. In the following we introduce the behaviour of the dynamic gates in more detail.

Order-dependent failures. The PAND gate, short for *priority-and gate*, is an AND gate where the failure output is dependent on the order of its children. This means that it only emits a failure if its children fail from left to right. Hence, the PAND gate enables the modelling of order-dependent failures in systems. The PAND gate can be distinguished by a *strict* and *weak* ordering of failures. Therefore, if two inputs (in the currently correct order) fail simultaneously the PAND gate fails if it has a weak ordering but not if it has a strict ordering. We consider PANDs to have a strict order, however, we consider the weak ordering in the following way: if there is a simultaneous failure of two or more inputs (in the currently correct order) we translate the failure in all possible strict orders.

Gate	Symbol	Failure	Additional behaviour	Special inputs
OR		$1/n$	—	—
AND		n/n	Can be used as INHIBIT gate.	—
VOT(k)		k/n	$k = 1$: OR gate $k = n$: AND gate	—
PAND		$1, 2, \dots, n$	Failures appear in a strict order, where weak orders are translated in all possible strict orders.	—
SPARE		n/n	Activates the next spare module after an input fails.	Primary and spare modules can be sub-trees.
FDEP		—	Dependent events fail if the trigger fails.	Dependent events can be BEs and gates. Trigger event can be a sub-tree.

Table 5.1: Main gate attributes of DFTs in this thesis.

Thus, we obtain $2^{(n-1)}$ choices for n simultaneous failures, where one of the choices will lead to a failure of the PAND gate.

Consider the PAND gate in Figure 5.3. The intention of the PAND gate “Switching unit” is to model a realistic change of the motors. That is, if a motor fails, the barrier has to be connected to the second motor by the switch. However, if this process was already conducted, the switch has fulfilled its purpose. Therefore, a failure of the switch after the start up of the second motor will not lead to a failure of the switching unit. Conversely, if the switch fails and then the first motor fails, the switching unit fails, since it cannot connect the second motor to the barrier. Now consider the simultaneous failure of the switch and the first motor. By a strict order, this would mean that the “switching unit” will not fail, although, most probably the second motor could not be activated anymore by the switch. However, by translating the weak order into all possible strict orders, we represent both scenarios: (1) the switch was still able to activate the second motor and (2) the switch could not activate the second motor anymore and therefore the “Switching unit” fails.

Spare management. The SPARE gate models the spare management of possibly shared spare modules. Originally, spare modules were limited to only BEs, but these restrictions have been relaxed in e.g. [BCS10]. We allow spare modules to be *independent sub-trees* and therefore lift the notion of cold, warm and hot BEs to spare modules:

- (a) All BEs contained in a *cold spare module* have a dormancy factor of $\alpha = 0$;
- (b) All BEs contained in a *warm spare module* have a dormancy factor of $0 < \alpha < 1$;
- (c) All BEs contained in a *hot spare module* have a dormancy factor of $\alpha = 1$.

A spare module is considered shared, if it is connected to more than one SPARE gate. However, since we require independent sub-trees, only the complete spare module can be shared and not sub parts of the spare module.

Figure 5.5: Example of a shared car tyre.

The first child of a SPARE gate is the *primary* and all subsequent children are *spares*. Initially the SPARE gate uses the primary. After the failure of the primary, the SPARE gate will attempt to switch to an available spare, i.e. a spare that has not failed yet and is not in use by another SPARE gate. We refer to this behaviour as *claiming*. This behaviour is repeated until no more spares are available. In case a spare module is shared, only one SPARE gate is able to claim it. For example, consider the sub-DFT depicted in Figure 5.5 representing four SPARE gates where each has a primary car tyre and shares the one spare tyre. If for example “tyre₃” fails, the corresponding SPARE gate will claim the spare “Spare”. Thus, the spare component becomes unavailable to all other SPARE gates. In the unlikely event that the currently active components of two or more SPARE gates that share a spare component fail at the same time, we would have a *spare race*. This will result in a non-deterministic choice, where each choice represents another SPARE gate that claimed the spare component first.

Since spare modules are dormant while not in use, the SPARE gate is responsible to activate the spare module after claiming. Hence, all BEs in the spare module have to change their state from dormant to active. This process is called *activation*. Recall that a spare module can be a sub-tree, and therefore a DFT gate can be a direct child of a SPARE gate. Hence, instead of directly activating a BE, the SPARE gate will activate the gate. Then the activation has to be propagated subsequently down the sub-tree to the BEs.

Common cause failures and feedback loops. The FDEP gate, short for *functional dependency gate*, describes a common cause failure, i.e. component failures that result from a shared cause. The first input to a FDEP gate is called *trigger* and describes the common cause. All subsequent children are *dependent events*. If the trigger fails, then also all dependent events fail. For example consider the FDEP gate “No detection” in Figure 5.3. The common cause that

all four sensors fail is a disconnection. Hence, each sensor has an individual failure behaviour, however, they fail simultaneously in case of a disconnection. Note that the output of the FDEP gate is denoted by a dashed line. This is the case, since the FDEP gate has no failure output. Hence, the FDEP gate emits no failure and is therefore not an input to any other gate and cannot be the top level event.

Another application where FDEP gates are utilised for are *feedback loops*. For example consider the DFT depicted in Figure 5.6a describing a simple failure of a power adaptor. The feedback loop of the power adaptor failure is as follows: If the thermal unit fails, it causes an overheating of the power supply unit, and a failure of the power supply unit will cause the failing of the thermal unit. A simplified representation of the feedback loop is given in Figure 5.6b. Here, the FDEP gate is substituted by a dotted arrow, where the source of the arrow corresponds to the trigger and the destination to the dependent event.

Figure 5.6: Feedback loop for a power adaptor.

5.3 Formal definitions

In the following we provide formal definitions describing how DFT gates and BEs are connected as well as to assign the failure behaviour to BEs. Recall Table 5.1 for a short summary of the main attributes of the DFT gates. We refer to DFT nodes as *elements* where they are partitioned into *gates* and *leaves*. Note that every element has a type from the following set:

$$\begin{aligned} \text{Gates} &= \{\text{OR}, \text{AND}, \text{PAND}, \text{SPARE}, \text{FDEP}\} \cup \{\text{VOT}(k) \mid k \in \mathbb{N}\}, \text{ and} \\ \text{Leaves} &= \{\text{BE}\}. \end{aligned}$$

We write $\text{Elements} = \text{Gates} \cup \text{Leaves}$ for all types of an DFT. Notice that only BEs can be leaves in a DFT. However, the set can be extended by special failure events, e.g. sinks that either have already failed or never fail. Formally a DFT is defined as follows.

Definition 5.1 (Dynamic fault tree). A dynamic fault tree (DFT) is a tuple $\mathcal{F} = (V, top, Tp, \delta)$ where

- V is a finite set of *nodes*;
- $top \in V$ is the unique top level element;
- $\delta: V \rightarrow V^*$ maps each node to its (ordered) *children*, also called *inputs*;
- $Tp: V \rightarrow \text{Elements}$ is a *type mapping* of nodes. We require that leaf nodes are mapped to **Leaves** and other nodes to **Gates**, i.e. $Tp(v) = \text{BE} \Leftrightarrow \delta(v) = \epsilon$.

For a node $v \in V$ of $\mathcal{F}$ we use $v \in K$ and say “ v is a K (gate)” meaning that the type of v is K , i.e. $Tp(v) = K$. Further we write $V_K = \{v \in V \mid Tp(v) = K\}$ for the set of all nodes of type K . Let $(\delta(v))_i$ denote the i -th child of $v \in V$ with $i \in \mathbb{N}$. Further for $v, v' \in V$ we write that $v' \in \delta(v)$ if $v' = (\delta(v))_i$ for some $i \in \mathbb{N}$. Let

$$\text{children}(v) = \{v' \in V \mid v' \in \delta(v)\}$$

be the set of all children of a node $v \in V$ and

$$\text{parents}(v) = \{v' \in V \mid v \in \delta(v')\}$$

the set of all parents of v . Similarly, we define the set of all reachable elements from $v \in V$, i.e. all elements that can be reached by going down the DFT, as

$$\text{descendants}(v) = \{v' \in V \mid v' \in \delta^*(v)\}$$

and the set of all previous elements, i.e. all elements that can be reached by going up the DFT, as

$$\text{ancestors}(v) = \{v' \in V \mid v \in \delta^*(v')\}$$

where δ^* is the transitive closure of δ . To assign the failure rate and dormancy factor to a **BE** we define an attachment function.

Definition 5.2 (Attachment function). Let $\mathcal{F} = (V, top, Tp, \delta)$ be a DFT, then

$$\Omega: V_{\text{BE}} \rightarrow \mathbb{R}_{>0} \times [0, 1]$$

is an *attachment function*.

Hence, the attachment function assigns a failure rate as well as a dormancy factor to each **BE** in a DFT $\mathcal{F}$. Thus for $v \in \text{BE}$ we have that $\Omega(v) = (\lambda, \alpha)$ with $\lambda \in \mathbb{R}_{>0}$ and $\alpha \in [0, 1]$.

Example 5.4. The formal definition of the DFT depicted in Figure 5.3 is given by the tuple $\mathcal{F} = (V, top, Tp, \delta)$ with

- $V = \{v_1, v_2, \dots, v_{15}\}$;
- $top = v_1$;

- $Tp = \{$

$v_1 \mapsto \text{OR},$	$v_2 \mapsto \text{FDEP},$	$v_3 \mapsto \text{VOT}(2, 4),$
$v_4 \mapsto \text{OR},$	$v_5 \mapsto \text{PAND},$	$v_6 \mapsto \text{SPARE},$
$v_7 \mapsto \text{BE},$	$v_8 \mapsto \text{BE},$	$v_9 \mapsto \text{BE},$
$v_{10} \mapsto \text{BE},$	$v_{11} \mapsto \text{BE},$	$v_{12} \mapsto \text{BE},$
$v_{13} \mapsto \text{BE},$	$v_{14} \mapsto \text{BE},$	$v_{15} \mapsto \text{BE}$

 $\};$
- $\delta = \{$

$v_1 \mapsto \langle v_3, v_4, v_7 \rangle,$	$v_2 \mapsto \langle v_8, v_9, v_{10}, v_{11}, v_{12} \rangle,$	$v_3 \mapsto \langle v_9, v_{10}, v_{11}, v_{12} \rangle,$
$v_4 \mapsto \langle v_5, v_6 \rangle,$	$v_5 \mapsto \langle v_{13}, v_{14} \rangle,$	$v_6 \mapsto \langle v_{14}, v_{15} \rangle,$
$v_7 \mapsto \emptyset,$	$v_8 \mapsto \emptyset,$	$v_9 \mapsto \emptyset,$
$v_{10} \mapsto \emptyset,$	$v_{11} \mapsto \emptyset,$	$v_{12} \mapsto \emptyset,$
$v_{13} \mapsto \emptyset,$	$v_{14} \mapsto \emptyset,$	$v_{15} \mapsto \emptyset$

 $\}$

where for $v_5 \in \text{PAND}$ it holds that $\delta(v_5, 1) = v_{13}$ and $\delta(v_5, 2) = v_{14}$, and for $v_6 \in \text{SPARE}$ it holds that $\delta(v_6, 1) = v_{14}$ is the primary spare component. Further, an example attachment function could be

- $\Omega = \{$

$v_7 \mapsto (3, 0),$	$v_8 \mapsto (1, 0),$	$v_9 \mapsto (2, 0),$
$v_{10} \mapsto (2, 0),$	$v_{11} \mapsto (2, 0),$	$v_{12} \mapsto (2, 0),$
$v_{13} \mapsto (1, 0),$	$v_{14} \mapsto (3, 0.5),$	$v_{15} \mapsto (3, 0.5)$

 $\}.$

Activation of components. Since spare modules can be sub-trees, we introduce *activation propagation* throughout a sub-tree. Therefore, the output of the gates will allow an activation input, whereas the inputs of a gate will allow for an activation output. If a **SPARE** gate claims a spare module then it will activate it by activating the connected child. Subsequently, the activation will be propagated through the sub-tree to all descendants, such that each gate activates its children down to the **BEs**. Note that all **BEs** that are not contained in a spare module are active from the start. We will use the set **Active** to denote all active **BEs** of a DFT. Further, we introduce a set containing all DFT elements that can be activated.

Definition 5.3 (Activation elements.). Let $\mathcal{F} = (V, \text{top}, Tp, \delta)$ be a DFT. We define the set of elements of $\mathcal{F}$ which can be activated as:

$$\text{Activation} = \{v \in V \mid v' \in \text{ancestors}(v) \wedge Tp(v') = \text{SPARE}\}.$$

Example 5.5. Consider the DFT $\mathcal{F}$ in Figure 5.3 on page 104 and defined in Example 5.3 on page 104. By applying Definition 5.3 on $\mathcal{F}$ we obtain the set $\text{Activation} = \{\text{Motor}_1, \text{Motor}_2\}$. Note, that Motor_1 will be directly activated by the SPARE gate.

5.3.1 Well-formedness

DFTs provide a lot of freedom, however, it is important to apply some restrictions to ensure a coherent behaviour. For example, a spare module should not be able to use components of the primary module. We divide these restrictions in three categories:

- (1) General DFT restrictions;
- (2) FDEP restrictions; and
- (3) SPARE gate restrictions.

The well-formedness conditions are formalised in Definition 5.4. In the following we elaborate those conditions.

First of all a DFT should not contain any cyclic behaviour, i.e. a descendent node should not have any ancestor node as child, as well as a node should not be input to itself, see condition (1). Further, a $\text{VOT}(k)$ gate should have at least k inputs, otherwise it could never fail, see condition (2). This well-formedness condition is included to simplify the semantics presented in Section 5.6.

FDEP gates distinguish themselves from other gates whilst they do not have any output. Hence, an FDEP gate should not be a top level event to describe a system failure, see condition (3). Moreover, since FDEP gates do not propagate any failure, they should not be an input to any other gate, see condition (4).

The primary of a SPARE gate should not be shared with any other SPARE gate, see condition (5). This restriction is needed since a SPARE gate describes the spare management of a unique module. Hence, the primary of a SPARE gate is claimed from the start whereas spare modules are not directly claimed, and therefore can be shared. However, when a spare module is claimed by a SPARE gate, it will become unavailable to the other SPARE gates it is shared with. For example consider two devices running on a battery. They cannot share a single battery at the same time, however, a spare battery can be available to both devices. After the battery of one device is empty, this device can claim the spare battery and therefore the battery becomes unavailable to the other device. Another restriction for SPARE gates is that primary and spare modules are independent, i.e. they do not share any components among themselves, see condition (6). Hence, a component can not be used over multiple spare modules.

Definition 5.4 (Well-formedness). A DFT $\mathcal{F} = (V, top, Tp, \delta)$ is well-formed if it fulfils the following conditions:

- (1) If $v \in V$, then $\text{descendants}(v) \cap \text{ancestors}(v) = \emptyset$ and $v \notin \text{children}(v)$;
- (2) If $v \in \text{VOT}(k)$ then $k \leq n$;
- (3) $top \notin \text{FDEP}$;
- (4) If $v \in \text{FDEP}$ then $\text{parents}(v) = \emptyset$;
- (5) If $v \in \text{SPARE}$ then for $v' = (\delta(v))_1$ and for all $v'' \in \text{parents}(v')$ with $v'' \in \text{SPARE}$ it holds that $v'' = v$;
- (6) If $v \in \text{SPARE}$ then for all $v' \in \text{children}(v)$ and for all $v'' \in V \setminus \text{descendants}(v')$ it holds that $\text{descendants}(v') \cap \text{descendants}(v'') = \emptyset$;

5

5.4 Key performance indicators for dependability

Key performance indicators (KPIs) are used to measure the performance of a system. For example, for a data centre the number of possible connections could be a KPI. In terms of a DFTs a KPI revolves around the reachability of the top-level event. For example, for a given mission time the probability of reaching the top-level event should not exceed 0.1. In the following we define measures on DFTs for three important KPIs:

1. The *mean time to failure*;
2. The *reliability*;
3. The *availability*.

Those three KPIs are standard measures in FTA [Sta+02]. In the following we will use MAs — the underlying stochastic model we use for DFTs — to define these KPIs. In Section 5.6 we will introduce the semantics of the underlying MA in detail.

Definition 5.5 (MA of a DFT). Let $\mathcal{F}$ be a DFT, and $\mathcal{A}_{\mathcal{F}}$ be the underlying MA of $\mathcal{F}$. Let s^0 be the initial state in $\mathcal{A}_{\mathcal{F}}$ describing the operational start of the system, and Fail the set of states in $\mathcal{A}_{\mathcal{F}}$ describing a system failure in $\mathcal{F}$.

Example 5.6. Consider the DFT $\mathcal{F}$ and its corresponding MA $\mathcal{A}_{\mathcal{F}}$ depicted in Figure 5.7. The DFT $\mathcal{F}$ will fail if the primary module A as well as the spare module B fail, where A has a failure rate of λ_A and B has a failure rate of λ_B and a dormancy factor of $\alpha > 0$. The corresponding MA $\mathcal{A}_{\mathcal{F}}$ is created as follows. Initially we start in a state where A and B have not failed yet. Now there is a race between the failure of A and B. If the transition labeled with rate λ_A fires first, then A has failed. Otherwise B has failed in its dormant mode with rate $\alpha\lambda_B$. If A has failed and B is still available, the SPARE gate claims

Figure 5.7: Example of a SPARE gate with one primary and one spare and the corresponding I/O-MA.

Figure 5.8: Example of the unreliability and MTTF of the railway crossing DFT.

and activates B , represented by the transition labeled with $act_B!$. When B is activated its failure rate is λ_B . If both, A and B have failed the MA sends a failure signal with the transition labeled with action $fail!$.

Note that the MA can have non-deterministic behaviour. This can arise from the structural behaviour of the DFT. Consider a DFT with a PAND gate with BEs “A” and “B”. Further, “A” and “B” are dependent events of the same FDEP gate. Now if the trigger of the FDEP gate fails, it induces the failure of “A” and “B”. The PAND gate will only fail if it receives the failure signal from “A” before the failure signal of “B”. However, the precise order of which BE communicates first is unknown and therefore the underlying MA models both possibilities. Since non-determinism can be present, we define all metrics in terms of their minimum and maximum values.

Mean time to failure. The mean time to failure (MTTF) in a DFT describes the expected time until the first system failure appears from the moment the system became operational. For example for the DFT in Figure 5.3 one may ask “What is the mean time until we can expect a failure of the level crossing

after taking it in operation?”. The plot in Figure 5.8 shows the MTTF for the level crossing with varying failure rates for the components. The dashed lines at the x-axis show the time point when to expect a first failure, and the dashed lines to the y-axis the corresponding probability that the system failed when running up to that point.

Technically, the MTTF in a DFT $\mathcal{F}$ is equivalent to the expected time ($\mathbf{eT}$) to fail in the underlying MA $\mathcal{A}_{\mathcal{F}}$. In particular, all states in $\mathcal{A}_{\mathcal{F}}$ that correspond to the system failure in $\mathcal{F}$ are collected in the set **Fail**. Then the expected time to reach states in the set **Fail** from the initial state s_0 , i.e. from when the system is fully operational, is computed. Since it is possible that non-determinism is present, we have to consider a scheduler D that resolves this choice. Hence, we obtain the MTTF for $\mathcal{F}$ under scheduler D .

Definition 5.6 (Mean time to failure). Let $\mathcal{F}$ be a DFT and $\mathcal{A}_{\mathcal{F}}$, s^0 , and **Fail** are defined as in Definition 5.5. Then the *mean time to failure* in $\mathcal{F}$ under a scheduler D on $\mathcal{A}_{\mathcal{F}}$ is defined as:

$$\text{MTTF}_{\mathcal{F}}^D = \mathbf{eT}^D(s^0, \text{Fail})$$

where $\mathbf{eT}^D(s^0, \text{Fail})$ is the expected time to reach a state in **Fail** from s^0 under scheduler D . The mean time to failure of $\mathcal{F}$ is given by the tuple:

$$\text{MTTF}_{\mathcal{F}} = \left(\min_D \text{MTTF}_{\mathcal{F}}^D, \max_D \text{MTTF}_{\mathcal{F}}^D \right).$$

Note that $\mathbf{eT}^D(s^0, \text{Fail})$ is equivalent to $\mathbf{eR}^D(s^0, \text{Fail})$ in Equation (3.1) on page 53, such that all transition rewards are set to zero and all state rewards are set to one.

Reliability. The reliability of a DFT is the probability that the DFT operates for a given time without failure. Hence, the *reliability until a given mission time* T is given by the probability that no failure occurs within time T . For example for the DFT in Figure 5.3 on page 104 one may ask “What is probability that the level crossing is still in operation after 5 years?”. The plot in Figure 5.8 shows the reliability for the level crossing with four and six sensors. The case with four sensors is as in Figure 5.3, i.e. if two out of four sensors fail the “Sensors failure” is triggered. In case of six sensors, four out of six sensors have to fail to trigger the “Sensors failure”. The red and blue line in Figure 5.8 show the probability that the system fails over time, where the dotted lines show the MTTF. Not that in both cases the minimum and maximum is equivalent, since we do not encounter non-determinism in the railway crossing. Remark that in industry the common notion of reliability often refers to the expected number of failures per time unit. However, this notion depends on repair times within the model.

Technically, the reliability in a DFT $\mathcal{F}$ for a mission time T is equal to the probability to fail in the underlying MA up to time T . Therefore, as for the MTTF, all states as in $\mathcal{A}_{\mathcal{F}}$ that correspond to the system failure in $\mathcal{F}$ are collected in the set **Fail**. Then the probability to eventually reach a state in **Fail** in mission time T from the initial state s_0 under a scheduler D is computed.

Hence, the reliability in a DFT $\mathcal{F}$ is directly correlated to the probability to fail in $\mathcal{A}_{\mathcal{F}}$.

Definition 5.7 (Reliability). Let $\mathcal{F}$ be a DFT, $\mathcal{A}_{\mathcal{F}}$ the underlying MA of $\mathcal{F}$, and $T \in \mathbb{R}_{\geq 0}$ a mission time. Let **Fail** be the set of states in $\mathcal{A}_{\mathcal{F}}$ describing a system failure in $\mathcal{F}$. Then the *reliability* of $\mathcal{F}$ under a scheduler D on $\mathcal{A}_{\mathcal{F}}$ is defined as:

$$\text{RELY}_{\mathcal{F}}^D = 1 - \Pr_{s^0, D}(\diamond^{\leq T} \text{Fail}),$$

where $\Pr_{s^0, D}(\diamond^{\leq T} \text{Fail})$ is the probability to reach a state in **Fail** from s^0 under scheduler D in time T . The reliability of $\mathcal{F}$ is given by the tuple:

$$\text{RELY}_{\mathcal{F}} = \left(\min_D \text{RELY}_{\mathcal{F}}^D, \max_D \text{RELY}_{\mathcal{F}}^D \right).$$

Note that $\Pr_{s^0, D}(\diamond^{\leq T} \text{Fail})$ is equivalent to the timed reachability objective as presented in [GHHKT14].

Availability. The availability of a DFT can be described as the probability that a system is available at a given time. Given a time interval, the availability is defined as the fraction of time that the system is operational. In general, availability is an interesting metric when considering that the DFT can be repaired. For example, after a system failure a repair could be conducted while the system is out of service. Then one may ask “What is the percentage of time that the system is operational during its life time?”. Remark that in industry the time interval for the availability is the lifetime of the system.

Technically, the availability in a DFT $\mathcal{F}$ is equal to the long-run average time spent in the underlying MA $\mathcal{A}_{\mathcal{F}}$. In particular, all states in $\mathcal{A}_{\mathcal{F}}$ that correspond to the system failure in $\mathcal{F}$ are collected in the set **Fail**. Then the unavailability, i.e. the percentage of time spent in the **Fail** states over the long-run, is computed. Thus, the availability is the remaining percentage of time spent in all other states.

Definition 5.8 (Availability). Let $\mathcal{F}$ be a DFT, and $\mathcal{A}_{\mathcal{F}}$ the underlying MA of $\mathcal{F}$. Let **Fail** be the set of states in $\mathcal{A}_{\mathcal{F}}$ describing a system failure in $\mathcal{F}$. Then the *availability* of $\mathcal{F}$ under a scheduler D on $\mathcal{A}_{\mathcal{F}}$ is defined as:

$$\text{AVAIL}_{\mathcal{F}}^D = 1 - \text{LRA}^D(\text{Fail}),$$

where $\text{LRA}^D(\text{Fail})$ is the long-run average time spent in states of **Fail** under scheduler D . The availability of $\mathcal{F}$ is given by the tuple:

$$\text{AVAIL}_{\mathcal{F}} = \left(\min_D \text{AVAIL}_{\mathcal{F}}^D, \max_D \text{AVAIL}_{\mathcal{F}}^D \right).$$

Note that $\text{LRA}^D(\text{Fail})$ is equivalent to LRR^D in Equation (3.7) on page 59, such that all transition rewards are set to zero and all state rewards for $s \in \text{Fail}$ are set to one and otherwise to zero.

Figure 5.9: Example DFT with non failing minimal cut sequences.

5.5 DFT analysis

DFT analysis can be divided into two categories: *qualitative* and *quantitative*. Since in DFTs the order of failures matters, cut sets are extended to ordered tuples of BEs. Hence a MCS for DFTs is a *minimal cut sequence* [TD04; LX-ZLL07]. However, minimal cut sequences are insufficient to fully characterise the failure behaviour of DFTs [JGKS16]. For example, a DFT does not necessarily fail if one of its MCSs fails. For example, in the presence of a PAND gate a failure may require BEs out of the cut sequence to not fail. Consider the DFT in Figure 5.9. The MCSs are given by “AB” and “AC”. Now consider the sequence “BAC” which will not lead to a failure, despite the MCS “AC”. Hence, non-failures of BEs have to be considered together with the cut sequences. In the rest of this section we focus on quantitative analysis techniques.

Quantitative analysis techniques compute numerical values over the fault tree. Those can be *importance measures*, e.g. indicating the criticality of components, or *stochastic measures*, e.g. providing failure probabilities. We focus on the KPIs presented in Section 5.4, i.e. the *mean time to failure*, the *reliability* and *availability*. In the following we describe the process of how to apply quantitative analysis on a DFT.

5.5.1 Input output extension of MAs

Recall Section 2.1 on page 20 and the set of actions in MAs. They distinguish between internal and external actions, where external actions can be delayed and be subject to synchronisation. Therefore, two MAs can communicate with a handshake, such that they synchronise on an action α when both MAs are ready to perform the α action, see Section 2.6 on page 37. To describe this handshake in a more natural way, we extend MAs to *input-output* MAs (I/O-MAs). By distinguishing input and output actions, the handshake is modified, such that an input action has to wait for an output action. For example, a BE would have an output action indicating a failure, while a gate has an input action to listen for the BEs failure. Hence, extending the handshake with the I/O notation gives us a more concise and natural way to model the communication in DFTs.

Definition 5.9 (I/O-MA). Let $\mathcal{A} = \langle S, s^0, Act, \hookrightarrow, \rightsquigarrow \rangle$ be an MA as defined in Definition 2.2 where $Act^X = Act^I \cup Act^O$ is partitioned in a set of input actions Act^I and output actions Act^O . Further $\mathcal{A}$ is *input-enabled*, such that $\forall s \in S$ it holds that $Act^I \subseteq Act(s)$.

Note that the input enabling defines that each state in an I/O-MA has to be able to receive any input signal. Now the elements of a DFT are translated into I/O-MAs and composed to a single I/O-MA $\mathcal{A}$. More details about the I/O-MAs follow in Section 5.6. We write $\mathcal{A}_{\mathcal{F}}$ for the I/O-MA of DFT $\mathcal{F}$.

5.5.2 Smart state space generation

An important part in the analysis of a DFT is the transformation into its underlying stochastic model. In our case, how to generate the I/O-MA $\mathcal{A}_{\mathcal{F}}$ of DFT $\mathcal{F}$. However, the *state space generation* of the I/O-MA is one of the main bottlenecks prior to the analysis. This is due to the internal state of the DFT elements. For example, for a VOT(k) gate one needs to keep track of how many children already have failed and for a PAND gate in which order the children fail. Moreover, considering a SPARE gate one has to observe the status of spare components, i.e. are they shared and in use by another SPARE gate or even failed while dormant. This complexity may yield large state spaces. An effective technique to keep the state space generation at bay is to generate the I/O-MA in a *compositional* manner. The idea of applying a compositional state space generation to DFTs was introduced by Boudali et al. [BCS10]. The key ingredient in the state space generation is the principal of *compositional aggregation*. *Aggregation* describes a parallel composition where the resulting model is subject to a minimisation, and compositional aggregation is the incremental technique of aggregation over several models.

Compositional aggregation provides an intelligent way of translating the DFT into its corresponding I/O-MA while keeping the state space small. An overview of the steps is given in Figure 5.10. First of all, instead of generating the state space of a DFT at once, the state space of each individual DFT element is generated in terms of an I/O-MA. Since only the behaviour of a single element has to be described by an I/O-MA, these state spaces are relatively small. To obtain the single state space of the DFT, the individual I/O-MAs are composed in parallel. Moreover, each composition is subject to a minimisation algorithm to keep the state space as small as possible. Figure 5.10 depicts the process flow of the compositional aggregation approach. The overall approach consists of the following steps:

- (1) Specifying the DFT as input;
- (2) Transformation of DFT elements into the underlying I/O-MA;
- (3) Parallel composition of two I/O-MAs;
- (4) Minimisation of the parallel composition;
- (5) Iterate step (3) and (4) until one I/O-MA remains;
- (6) Analysis of the I/O-MA w.r.t. input metrics.

Figure 5.10: Graphical overview of the compositional aggregation of a DFT. (a) DFT specification as input; (b) Transformation into the underlying I/O-MA for each DFT element; (c) composition of two I/O-MAs; (c) minimisation of the composition; (e) the resulting I/O-MA.

In the following we discuss the individual steps in more detail.

The first step is to define the DFT structure describing the system failure of interest (Figure 5.10a). Besides the DFT structure also the failure rates and dormancy factors of the BEs have to be specified.

The second step (Figure 5.10b) is the translation of the DFT elements into their corresponding I/O-MAs. Each DFT gate is translated w.r.t. its type and number of inputs and each DFT leaf is translated w.r.t. its failure rate and dormancy factor. Moreover, the structure of the DFT is represented over the I/O-actions of the I/O-MAs.

Example 5.7. Consider a DFT consisting out of an AND-gate “A” with BEs “B” and “C” as inputs. Now the I/O-MA representing “A” will have input actions for the failure signal of “B” and “C”, whereas the I/O-MAs of “B” and “C” have an output action for their own failure.

The third and fourth step represent the core of the compositional aggregation, the composition and minimisation (Figure 5.10c and Figure 5.10d). Therefore, two I/O-MAs are composed in parallel, as described in Definition 2.19 on page 38, while hiding all actions that are no longer subject to further synchronisation. Subsequently the resulting I/O-MA is minimised based on weak bisimulation relation as described in Section 2.7. This process is iterated until a single I/O-MA remains (Figure 5.10e). The order of the aggregation process heavily influences the maximal number of intermediate states, and is determined by a smart heuristic introduced by Crouzen and Lang [CL11]. The heuristic focuses on finding an order such that the intermediate state space is kept as small as possible. Consider a set of n models that have to be composed, then we have $2^n - n - 1$ possibilities to select subsets of models to compose [CH10]. Hence, the number grows exponentially with the number of components. Thus, evaluating all subsets quickly to find an optimal order becomes infeasible for large n . Note that under different heuristics the resulting number of states at the end is the same.

Finally, we obtain the I/O-MA $\mathcal{A}_{\mathcal{F}}$ of DFT $\mathcal{F}$ which is input to the analysis together with the key performance metrics of interest.

Action	Signal	Context
<i>act</i>	Activation	Communication signal for the activation of DFT elements. Send from parents to children.
<i>fail</i>	Failure	Communication signal for the failure of DFT elements. Send from children to parents.

Table 5.2: DFT signals in an I/O-MA.

5.6 Semantics for DFTs

Reliability engineering requires models and analysis techniques to design and maintain dependable systems. The ubiquity of CPSs in our daily life and the growing complexity of those systems demands for scalable and state-of-the-art analysis techniques. However, to perform any kind of analysis we need to represent the system in a formalism that can be analysed. The high level formalism of DFTs appears to be simplistic and is appealing to use with existing handbooks on their interpretation [VGRH81; Sta+02] and norms for their analysis [IEC61025] available. Besides, while DFTs provide a high level formalism that is easy to understand, their actual meaning and therefore correct analysis is non trivial. Note that with the introduction of DFTs [DBB92] there was no proper formalisation. This had the effect that the meaning of specific fault trees was unclear [CSD00]. Moreover, to be able to formally reason over a DFT, there should be a semantics. Since the initial formalisation of DFTs, several different semantics has been introduced. These semantics, however, are not necessarily compatible with each other. Recall that quantitative analysis of a DFT relies on extracting an underlying stochastic model. As introduced in Section 5.5 we define the reliability, MTTF and availability of a DFT in terms of MAs. Moreover, we have introduced the input-output extension to I/O-MAs in Section 5.5.1. Henceforth, the semantics of a DFT will be given in terms of I/O-MAs.

In this section we introduce an I/O-MA for each DFT element, following the approach of Boudali et al. [BCS10]. Those I/O-MAs are then used to build the underlying stochastic model of the DFT via compositional aggregation, as described in Section 5.5.2. Note that the I/O distinction is given by the input represented with a “?” and the output represented with a “!”. We have two types of signals within the DFT, the failure as well as the activation. The interpretation of the input and output communication in DFTs is as follows:

Failure: DFT gates have input actions for the failure of each of their children, while DFT elements have an output action for their own failure. If a DFT element fails, then it will send its failure via an output action to its parents, while the parents wait to receive the failure notification via their input action.

Activation: DFT elements have an input action for their own activation, while DFT gates also have an output action for the activation of their children. If the children of a DFT gate should be activated, the gate will send out

the activation over its output action to all of its children, while the children wait to receive the activation notification over their input action.

To represent the activation and failure signal we use the action “*act*” and “*fail*”, respectively. Table 5.2 provides a quick overview of the two types of actions. Note that for gates with n inputs we describe the corresponding input actions with a subscript $i \in [1, n]$ and write “*act_i*?” and “*fail_i*?”. Moreover, all DFT gates are defined without activation. In Section 5.6.3, we introduce how to extend DFT gates with activation if they are contained in a spare module. Note that when depicting an I/O-MA we omit the self-loops required to make the I/O-MA input enabled. Besides, while we provide a graphical representation of the semantics, in Chapter 7 we use Lotos NT [Cha+11] as well as MAPA [Tim13] to describe the general cases of the I/O-MAs.

5.6.1 BEs and static gates

Basic events. Recall that a BE can be in three different modes: dormant, active and failed. We will add a fourth mode, namely *down*, representing that the BE has propagated its failure signal. Therefore, the I/O-MA of a BE consists of four states: a dormant state, an active state, a failed state, and a down state. Figure 5.11 shows the I/O-MAs of a BE such that (a) is a cold BE with dormancy factor $\alpha = 0$; (b) is a warm BE with dormancy factor $\alpha \in (0, 1)$; (c) is a hot BE with dormancy factor $\alpha = 1$. Initially, the I/O-MA is in its dormant state

Figure 5.11: The I/O-MAs modelling a cold, warm and hot BE.

where it waits to be activated (or fails). Therefore a transition labelled with “*act*?” goes into the active state. Moreover, if the BE has a dormancy factor $\alpha > 0$, then a Markovian transition with rate $\alpha\lambda$ goes from the initial to the failed state. Besides, from the active state a Markovian transition with rate λ goes into the failed state. From the failed state a transition labelled with “*fail*!” goes into the down state. Note that all I/O-MAs for DFT gates will be defined as already active. However, we introduce a separate activation module which will be combined with the I/O-MAs for the DFT gates.

Note that if a BE is not part of a spare module, then the dormancy factor of the BE is irrelevant since the component will be active from the start. Hence, we introduce an always active BE that has a active, failed and down state, as depicted in Figure 5.12. Note that the hot BE has the same behaviour in its dormant and active state and can also be represented by the active BE. Then due to the input enabling of I/O-MAs, the BE would just stay in the current state after receiving an activation signal.

Figure 5.12: The I/O-MA modelling an active BE.

OR gate. An OR gate fails if at least one of its inputs fails. Figure 5.13 depicts the I/O-MA for an OR gate with n inputs. In the leftmost state the I/O-MA waits for the failure signal “ $fail_i?$ ” from one of its children. If a failure signal is received, then the OR gate will emit its own failure with “ $fail!$ ”.

Figure 5.13: I/O-MA of an OR gate.

AND gate. An AND gate fails if all of its inputs fail. Figure 5.14 depicts the I/O-MA for an AND gate with n inputs. Since the input failure signals can be received in any order, the I/O-MA models all combinations. This leads to a total of $\sum_{i=0}^n n^i(n-i)$ transitions ready to receive a “ $fail_i?$ ” signal. After n fail signals are received, the AND gate emits its own failure with “ $fail!$ ”.

Figure 5.14: I/O-MA of an AND gate.

Note that the I/O-MA in Figure 5.14 provides a clean translation of the AND gate by keeping track of which input already failed. However, by omitting this information, we can give a more compact representation of the AND gate

as depicted in Figure 5.15 where in the n -th state the “fail!” signal is send. In this definition the state will not encode which input already failed and only count the number of failures. Therefore, we reduce the state space such that for n inputs only $(n + 1)$ states are required.

Figure 5.15: I/O-MA of a reduced AND and VOT(k) gate.

VOT(k) gate. The VOT(k) gate fails if k out of n inputs fail. Figure 5.16 depicts the I/O-MA for a VOT(2,3) gate. In general, the I/O-MA for a VOT(k) gate is a combination of the I/O-MAs of the AND and OR gate. Therefore the I/O-MA of the VOT(k) gate can be defined in two parts. The first part is an AND gate with n inputs up to the $(k - 1)$ th fail signal and the second part is an OR gates with $n - (k - 1)$ inputs. Thus, after k failure signals are received, the VOT(k) gate will emit its own failure signal “fail!”. Note that the AND and OR gate are special occurrences of the VOT(k) gate with $k = n$ and $k = 1$, respectively.

Figure 5.16: I/O-MA of a VOT(2,3) gate.

Equivalent to the AND gate, we can simplify the complexity of the I/O-MA for the VOT(k) gate by omitting the information which input already failed. Therefore, we obtain an I/O-MA as for the reduced AND gate in Figure 5.15 where in the k -th state the “fail!” signal is send.

5.6.2 Dynamic gates

PAND gate. The PAND gate fails if its inputs fail from left to right, i.e. in the right order. Figure 5.17 defines the I/O-MA for a PAND gate with n inputs. If all n inputs fail in order from 1 up to n , then the I/O-MA emits a “fail” signal. Otherwise, the I/O-MA stays in a sink state. As an example for a PAND gate

consider the “Switching unit” in Figure 5.3. It is part of drive-train of the boom-barrier in a railroad crossing. There are two motors of which one is necessary to operate the drive train. Initially one motor is connected to the barrier, and as soon as this motor fails, the second motor can take over. However, it requires the switch to establish the connection to the barrier. The failure of the switch, however, is only relevant if it occurs before the first motor fails. Hence, the barrier failure only occurs if either both motors fail or if the switch fails and afterwards the first motor fails.

Figure 5.17: I/O-MA of a PAND gate.

SPARE gate. The SPARE gate fails if its primary module and all spare modules fail. Moreover, if the current module that is in use by the SPARE gate fails the next available spare becomes active. The I/O-MA for a SPARE gate with a primary and two spare modules is depicted in Figure 5.18. The distinguishing factor of a SPARE gate is that the spare modules are initially dormant and will be activated when utilised. Hence, after receiving the failure signal “ $fail_1$ ” of the primary module, the SPARE gate sends an activation signal “ act_2 ” to its first spare module. If the primary and all spare modules have sent a “ $fail$ ” signal, then the SPARE gate will emit its failure. Note that dormant modules can also fail. Thus the spare gate always listens for the failure of all modules.

Since SPARE gates have the capability of activating modules, we have to define the behaviour of a shared spare module, i.e. a spare module that is connected to two different SPARE gates. For example consider a car with a spare wheel. If one of the four tires will fail, it can be replaced with the spare tire. However, if any other wheel will fail afterwards, then no spare tire would be available anymore. Figure 5.19a illustrates this using the DFT notation. Thus, the activation of the shared spare module by one SPARE gate makes it unavailable for the other SPARE gates. With respect to the I/O-MA, this is equivalent to the failing of the spare module for those SPARE gates. Hence, if there exists a shared spare, the I/O-MA has not only to listen if the shared spare fails, but also whether it is activated by another SPARE gate. Figure 5.19b depicts the I/O-MA representing a SPARE gate with one primary module and a shared spare.

FDEP gate. The FDEP gate models the dependency between a trigger event and its dependent events. Hence, after the trigger event fails, the FDEP gate will send failure signals for its dependent events. Figure 5.20 models the I/O-MA

Figure 5.18: I/O-MA of a SPARE gate with one primary and two spare modules.

Figure 5.19: Shared SPARE gate.

of the FDEP gate where the order of how the failure signals are sent out is non-deterministic. In case the dependent events are connected to a PAND gate, the resolution of the non-deterministic choices in the I/O-MA determines whether the PAND gate fails or not. Although the failure of the dependent events are at the same moment, due to the propagation of the signals an order is induced.

If this is an undesired behaviour it is possible to use an alternative I/O-MA of the FDEP gate, depicted in Figure 5.21. In this case, the FDEP gate has a fixed order in which the signals are sent out to its dependent events.

Figure 5.20: I/O-MA of a FDEP gate.

Figure 5.21: I/O-MA of a FDEP gate with ordering.

5.6.3 Spare activation

Since we allow whole sub-trees as spare modules, we have to extend the I/O-MAs for the DFT gates with activation capabilities to make BEs contained in the spare module switch from dormant to active. We will introduce two different activation classes:

1. Full activation for OR, AND, VOT(k), and PAND gates;
2. Partial activation for SPARE gates.

Note that since an FDEP gate is independent, i.e. it has no parent, the FDEP gate is not considered to be part of a spare module, and its trigger event is assumed to be active.

To model the activation we use an *activation sequence*. This sequence will be initiated by an activation signal “act!”. In the following we explain how to adapt the introduced I/O-MAs with activation signals.

Figure 5.22: I/O-MA of the activation module.

Full activation. If a gate should perform a full activation it means that the activation signal should be propagated to all its children. This is the case for OR, AND, VOT(k) and PAND gates that are contained in a spare module. To extend the corresponding I/O-MAs with activation an additional I/O-MA for the full activation is introduced, called *activation module*. Figure 5.22 depicts the I/O-MA of an activation module that activates n children after the reception of an activation signal “ act ”. The I/O-MAs of the OR, AND, VOT(k) and PAND gates will be extended with the activation module via parallel composition. Note that we can omit the activation procedure if the gate is already in its failed or down state.

We illustrate how the activation module is composed with the help of an AND gate with two inputs, depicted in Figure 5.23. The activation states are filled white. Moreover, we depict the failure signals out of the activation states with a dashed arrow for “ $fail_1?$ ” and with a dotted arrow for “ $fail_2?$ ”. The failure sequence for an inactive and active AND gate is equivalent and therefore the I/O-MA will not change regarding its failure behaviour. From all states where the gate has not failed yet, the activation sequence will be induced when receiving the activation signal “ act ”. Notice that with the last activation signal that is send out the I/O-MA is back in the source state where it received the activation signal (if no failure occurred in the mean time).

Figure 5.23: I/O-MA for the AND(2) gate with activation.

Partial activation. The SPARE gate has to be treated differently due to the fact that it only activates its spare modules when they are used. Hence, the SPARE gate will partially carry out the activation of its children. Figure 5.24 depicts the activation pattern for a SPARE gate. In the inactive mode, the

SPARE gate will only listen for failure signals of its inputs and register them. If an activation signal is received, the SPARE gate will activate only the leftmost module and move to the corresponding active state. To illustrate this more clearly, Figure 5.25 depicts the I/O-MA for a SPARE gate with one primary and one spare module including the activation pattern. The SPARE gate waits in its initial inactive state and listens for its activation or any failure of its inputs. If the activation is received before any input has failed, the SPARE gate forwards the activation to its primary and moves into the branch with all spares in working order. In the case the spare module already failed, the SPARE would move into the branch where only the primary is in working order. In the case the primary already failed, the activation will be forwarded to the spare module. If both primary and spare already failed before the activation signal is received, the SPARE gate will emit its failure.

Figure 5.24: Pattern of the I/O-MA for the SPARE gate activation.

Figure 5.25: I/O-MA for the SPARE gate activation with one primary and one spare.

5.7 Related work

Dynamic fault trees were introduced as a way to model fault tolerant computer systems by Dugen et. al. [DBB92]. Moreover based on this methodology *DIFTree*, a prototypical software-tool for the reliability analysis of DFTs, was developed [DVG97] and evolved into *Galileo* [SDC99]. Note that *Galileo* implements a modular approach combining different static and dynamic analysis approaches based on binary decision diagrams and Markov models. In particular, *Galileo* uses CTMCs for the quantitative analysis of DFTs. However, the translation of a DFT into a CTMC as done by *Galileo* leads usually to a large state space, and is subject to the state space explosion problem. Besides, the DFT formalism restricts spares to BEs. To overcome the limitation of having only BEs as spares as well as to handle the state space explosion problem, Boudali et al. introduced a framework for DFTs based on I/O-IMCs [BCS10]. In particular they translate each DFT element into a small I/O-IMC and use compositional aggregation to obtain the smallest possible I/O-IMC representing the DFT. Note that this framework is the basis of our I/O-MA semantics. Further, Volk et al. [VJK16] focuses on a state space generation approach by aggressively exploiting the DFT structure. In particular they have taken the *Galileo* approach and combined it with several state space reduction techniques from model checking, such as symmetry and partial-order reduction.

Bayesian networks (BN) [Pea88] are a popular method for the quantitative analysis of DFTs. Commonly DFTs are translated into discrete time Bayesian networks (DTBN) [BD05], continuous time Bayesian networks (CTBN) [BD06], or dynamic Bayesian networks (DBN) [MPBCR06]. The general idea in transforming a fault tree into a BN is to introduce random variables for each DFT event. Therefore, random variables will represent gates such that they are conditionally dependent on the random variables representing their children. The main difference between DTBN, CTBN and DBN is their representation of time. While CTBN consider continuous time, DTBNs and DBNs discretise the time. DTBNs are slicing a given time interval into n possible equidistant time intervals, such that a failure event occurs during an interval instead of a time point. Contrary, DBNs assume discrete time points for each event. For DBNs there exists tool-support by *DBNet* [MPBVCR06] and *Radyban* [MPBCR08] for reliability analysis of DFTs. Note that a clear restriction that all BN formalism's have is that feedback-loops are not allowed, since they would yield a cyclic BN. Moreover, sharing of spares is in DTBNs and CTBNs not considered as well as only BEs can be spares. Besides, DBNs also restrict warm spares to BEs.

Petri nets (PNs) [Pet77] are a well known formalism in the analysis of distributed systems. Besides, they are also used in the quantitative analysis of DFTs. In particular, there exist formalisms translating DFTs to stochastic well-formed coloured nets (SWN) [BCR04] and to generalised stochastic Petri nets (GSPNs) [CR05]. Similar to our approach, the Petri net formalisms translate the DFT element wise. In case of SWNs, each DFT element is modelled as a small Petri net and then the input and outputs are merged according to the structure of the DFT. Contrary to SWNs, a graph transformation is used for an element wise-reduction of a DFT to a GSPN. Besides, both formalisms are

using the *GreatSPN* tool [MBCDF94] to analyse the resulting Petri nets, which resolves into a reduction to CTMCs. Note that the transformation to SWN in [BCR04] also provides an extension to DFTs with repairs. However, there exist several limitations in the expressiveness of the DFT. The SWN formalism only support BEs as spares and only share spares amongst symmetric SPARE gates. Besides, FDEP gates also only allow BEs as triggers and dependent events. For the GSPN formalism similar restrictions hold.

Note that different formal semantics can lead to various DFT dialects. Hence, they differ for example in the type of supported elements and their failure propagation. This has the effect that different DFT semantics can lead to different interpretations to syntactically identical DFTs. Further, classical concepts like minimal cut sets are difficult to generalise for DFTs. In conclusion, despite the seemingly simplicity of DFTs, they are complex objects. Therefore, engineers should be aware of the various nuances of interpretation of DFTs to accurately interpret the analysis results. An extensive discussion about such differences is given in our work of uncovering DFTs [JGKS16].

5.7.1 DFT extensions

While DFTs were introduced to overcome limitations of SFTs, they still can have a shortage in expressiveness to describe a systems failure behaviour. Hence, over time, several extensions to DFTs were proposed to overcome special needs. In particular some specialised gates were introduced with variations of orderings and dependencies. We illustrate in the following that the I/O-MA semantics is flexible enough to accommodate for a variety of new gates.

Sequence enforcer. A *sequence enforcer* (SEQ) as presented in [DBB92] guarantees that inputs fail from left to right. Hence, components connected to the SEQ can only fail in their input order. Figure 5.26a shows a DFT with a SEQ (represented as a box with an arrow and no parent). The DFT describes a power outage in a pump where first a leakage has to happen before a short circuit failure can occur. Hence, the SEQ enforces that only the failure order of leakage and then short circuit can appear, but not the other way around.

Figure 5.26: Sequence enforcing .

Some papers mimic a SEQ with a cold SPARE gate [BCS10]. By applying this

assumption we obtain an equivalent DFT for the power outage with a SPARE gate, as depicted in Figure 5.26b, where the dormancy factor of the short circuit failure is set to 0. However, as shown in [JGKS16] this substitution is not always possible. Thus, instead of substituting a SEQ by a SPARE gate with cold spare modules a proper semantics has to be defined. Therefore, we can define a new I/O-MA for the SEQ. In the following we introduce a simple solution for the case that only BEs can be connected to a SEQ.

The I/O-MA for a SEQ with n BEs is depicted in Figure 5.27a and will make a BE failable by sending an output action “failable”. Hence, also the I/O-MA of the BE has to be extended. Figure 5.27b depicts a warm BE with the failable extension. Two additional states are added in front of the initial state, an active and inactive *fail safe* state. Now, if the “failable” action is received, the I/O-MA moves into the standard inactive or active state, respectively. Initially, the BE will be in its inactive fail safe state (or active fail safe state if it is not in a spare module).

Figure 5.27: I/O-MAs for a SEQ with only BEs.

Priority or gate. Temporal fault trees [WP09; WP10; EWG12] are SFTs with priority gates. Hence they extend the set of SFT gates, with a PAND, SAND, and POR gate. The PAND and SAND gate are equivalent to the priority and sequence gates in the set of DFT gates. Where the PAND gate is as such available as a DFT gate, the SAND gate, short for *sequential and* gate, is equivalent to a SPARE gate with cold spare modules. The POR gate, short for *priority or* gate, however, introduces a new behaviour. A POR gate fails if the first child fails before any other child fails. Extending the I/O-MA semantics with a POR gate is straight forward. The gate resembles the first step of the PAND gate, where after the failure of the left most child, the I/O-MA sends out the failure signal for the POR gate. The I/O-MA for a POR gate in terms of an I/O-MA is depicted in Figure 5.28.

Example 5.8. Lets consider an example provided in [JGKS16] for a FT using a POR gate. The corresponding fault tree is depicted in Figure 5.29. The system consists of two devices connected to a data link and the system is operational as

Figure 5.28: I/O-MA of the POR gate.

long as one device is operational and no device blocks the data link. If a device is blocking the data link, it is called “*babbling idiot*”, i.e. a device constantly sending messages over a data link and therefore blocking communication for other devices. A device failure is caused by a processor failure and a data link failure will turn a device into a babbling idiot. The TLE fails if either both devices fail or a babbling idiot occurs. A babbling idiot occurs if the data link failure occurs but the devices processor is still operational. This is the part where the POR gate is used.

Figure 5.29: POR gate babbling idiot example.

Probabilistic and rate dependency gate. The FDEP gate describes functional dependencies in systems and can be used to create feedback loops. However, there may be cases such that a trigger event only causes with a certain probability a failure, or just accelerates the failure of its dependent events. Therefore the *probabilistic dependency* (PDEP) gate [MPBCR06; MPBCR08; MPBVC06] and the *rate dependency* (RDEP) gate [RGvS16; RGDPS16] were introduced.

The PDEP gate extends the FDEP with a probability factor p describing the probability that the dependent events fail given that the trigger has failed.

Hence, if the trigger fails, the PDEP sends out failure signals to its dependent events with probability p . If $p = 1$ the PDEP gate is equivalent to the FDEP gate, and if $p = 0$ the failure signal is almost surely not send out and makes the PDEP gate superfluous. Figure 5.30 depicts an I/O-MA of the PDEP gate with an ordered failure propagation. After the trigger fails, the I/O-MA moves into a probabilistic state where it advances with probability p to a state initiating the failure propagation to the dependent events. Otherwise, the I/O-MA moves into a sink state.

Figure 5.30: I/O-MA of an ordered PDEP gate.

The RDEP gate extends the FDEP such that the dependent events will not fail, but rather their failure gets accelerated. Therefore, a rate acceleration factor $r \in \mathbb{R}_{\geq 0}$ is assigned to the dependent events of the RDEP gate. Hence, if the trigger event fails, the failure rate of the dependent events gets accelerated with the factor r . To realise this behaviour in our I/O-MA semantics, we would need to introduce a new acceleration signal and extend the dependent BEs with the accelerated failure. Figure 5.31 depicts an I/O-MA for a dependent BE of an RDEP gate with acceleration factor r . The RDEP gate is equivalent to the FDEP gate, where the “fail” signals for the dependent events are replaced with “accelerate” signals.

Figure 5.31: I/O-MA of a dependent BE of an RDEP gate.

5.8 Conclusion

In this chapter we explored the expressiveness of DFTs. In particular, we described a semantics for DFTs based on I/O-MAs. This semantics has the characteristic of being compositional and flexible. That is, the DFT is build up through small models that are composed with each other. Further, the semantics is flexible in the sense that it can be easily extended based on the demands of the system.

Moreover, we have shown how to compute important key performance metrics for safety. In particular, the MTTF, reliability, and availability. For the analysis we have defined those metrics in terms of MAs. Hence, we exploit algorithms for MAs to compute KPIs of DFTs. Furthermore, this allows us to extend analysis of DFTs also to the realm of costs, e.g. by extending the model to MRAs and exploiting the expected reward algorithms from Chapter 3.

Fault maintenance trees

RISK management helps to identify, assess and prioritise risks that can occur in all kinds of systems. Organisations can face consequences, economically as well as environmental and social, when risks are not handled properly. Therefore, the ISO 31000 standard [ISO31000] provides basic guidelines on how to implement risk management. Especially for safety-critical systems like nuclear power plants, smart grids, air traffic, railways, and many more, a failure can lead to a catastrophic event. To be able to contain risks, organisations are incorporating reliability, availability, maintenance, and safety (RAMS) analysis as a tool into their risk management process. RAMS requirements are of utmost importance in safety-critical systems and are often imposed by law or government regulations. As introduced in Chapter 5, FTA is a widely applied industry standard in reliability engineering to analyse the risks related to complex systems and henceforth also an integral part of most RAMS analysis.

Where FTA focuses on reliability analysis and identifying safety risks, it does not explicitly consider maintenance. However, the maintainability of a system can have a high impact on the whole system performance. Proper maintenance, consisting of timely inspections, renewals, repairs, and spare management, reduces the number of failures and extends the system's life time. FTA can only implicitly consider maintenance by including the effects into the failure rates of the components. Besides, DFTs support elementary maintenance aspects like spare management. However, more advanced maintenance like condition-based strategies for prevention, corrective, clock- and age-based maintenance, inspection cycles, etc. are not trivial to include into a components failure rate.

While RAMS analysis is used to show the system's compliance of certain requirement, it is also a tool for asset management. For example, one would like to find the optimal balance between maintenance and system dependability that minimises risks. An important trend in infrastructural asset management is reliability-centered maintenance (RCM) [Mou97]. There the goal is to obtain optimal maintenance policies by maintaining crucial objects more intensively than less crucial ones. Hence, RCM focuses on finding an optimal balance between maintenance and system dependability, by placing maintenance effort where it matters most. To be able to make such decisions, RCM requires a good insight in the effect of a maintenance policy on the system dependability.

Since maintenance is not fully represented within DFTs, also FTA is limited

in its usage. For example, FTA is not the best option for the RAMS analysis of systems that highly depend on maintenance changes, or to analyse different RCM strategies. While there exists a notion of repair for components in FTs if they are not affecting the operations of other components [Sta+02], this simple repair model is not sufficient to express more advanced maintenance. To overcome these limitations, FTs as well as DFTs were augmented with more advanced repairs [RFIV04; BCR04]. Nevertheless, they still do not fully represent more advanced maintenance procedures like inspection cycles. Thus, to overcome this limitation and assess the impact of different maintenance strategies on a system, DFTs have to be intertwined with maintenance.

In this chapter we extend DFTs with *maintenance* and introduce *fault maintenance trees* (FMTs). First, we differentiate between *corrective* and *preventive* maintenance. Subsequently we discuss how these different maintenance regimes can be included in DFTs. In particular, we focus on three main extensions to achieve this: (1) the actual maintenance of basic components; (2) extensions describing the maintenance strategy; and (3) maintenance communication w.r.t. all DFT elements. Therefore, we start with introducing maintainable components including degradation phases, inspections and repairs. Further, we describe different maintenance signals and modules extending DFTs with preventive and corrective maintenance. In particular we introduce *inspection modules* and *repair units*. Finally we describe FMTs and the effects and changes on the I/O-MA semantics, as well as discuss some simplifications.

Origins of the chapter. This chapter presents repairs and inspections for DFTs based on

- D. Guck, J. J. Spel, and M. I. A. Stoelinga. “DFTCalc: Reliability centered maintenance via fault tree analysis (tool paper)”. In: *Proceedings of the 17th International Conference on Formal Engineering Methods (ICFEM 2015)*. Vol. 9407. Lecture Notes in Computer Science. Springer Verlag

and introduces fault maintenance trees and their I/O-MA semantics.

Organisation of the chapter. Section 6.1 discusses maintenance procedures and their relation to DFTs. Section 6.2 introduces maintenance at the level of basic components and Section 6.3 introduces maintenance models for DFTs. In Section 6.4 we introduce FMTs and extend the I/O-MA semantics and provide smart improvements on the semantics in Section 6.5. Section 6.6 discusses related work and Section 6.7 concludes the chapter.

6.1 Maintenance

Maintenance of critical assets is crucial to ensure and improve system dependability [HMS+02]. By performing inspections in a timely manner, conduct repairs and renewals of worn out components, the lifespan and reliability of a system can be significantly improved. A trend in asset management is to apply

reliability-centred maintenance (RCM) [Mou97] which has the goal of optimising maintenance planning by maintaining critical assets more rigorous than less critical ones. Therefore it is important to have a good understanding of the effects of maintenance policies on the system's dependability w.r.t. several key performance indicators (KPIs) such as reliability, availability, mean time to failure, costs, and many more.

Maintenance comprises a combination of inspections, repairs, renewals and spare management. In the following we distinguish between two types:

- (1) *Preventive* maintenance, referring to the prevention of component failures. Hence components are inspected, and based on their condition, (partial) renewals or repairs are performed, putting the component in a better condition.
- (2) *Corrective* maintenance, which is carried out after a failure has occurred, replacing or repairing the broken component.

In the following we elaborate more on both maintenance principles.

6.1.1 Preventive and corrective maintenance

Preventive maintenance. The goal of preventive maintenance is, as the name implies, to prevent the failure of components. Therefore, it is crucial to be able to predict how components degrade.

Example 6.1. Lets consider a car engine. Motor oil is used in engines among others to reduce wear. Hence, the degradation of the engine is in part dependent on the motor oil. With a bad oil the engine could degrade faster and by changing the oil the degradation could be slowed down again. Hence, maintenance of the oil can be done to slow down the degradation of the engine.

Preventive maintenance can be further divided into

- (1) *condition-based* maintenance, and
- (2) *usage-based* maintenance.

Condition-based maintenance factors in the degradation of a component. For example, a car tire will be replaced if its profile is too low. On the other hand, usage-based maintenance factors in a usage interval. For example, for a car a specified set of maintenance actions should be performed after every 10,000 miles.

Since parameters for condition- as well as usage-based maintenance do not have to be the same for the complete lifetime of a component, there exists the sub class of *adaptive* maintenance. Hence, the preventive maintenance strategy can be adapted to a new context or situation. For example, if a component was already renewed several times based on a usage-based maintenance, then the usage-condition could be adapted.

Corrective maintenance. Corrective maintenance is a maintenance action that is carried out on a component after a failure is detected. The goal of corrective maintenance is to repair or replace the failed component such that the intended behaviour is restored and the component can perform as intended.

Example 6.2. Recall Example 6.1, however we consider that the preventive maintenance is not carried out. This will lead eventually to the wear out of a component within the engine. While carrying out corrective maintenance on the engine, the failed component or components are identified and repaired.

Note that corrective maintenance may trigger preventive maintenance. For example, if the car engine is repaired, additional inspections are carried out. Hence, in Example 6.2 the preventive maintenance action of changing oil would be performed.

6.1.2 Maintenance in fault trees

DFTs describe how a system fails depending on the failure of basic components in the system. As described in Chapter 5, DFTs already incorporate elementary maintenance aspects via spare gates. Hence, one can model spare management within the DFT and even specify dormant failure rates to inactive components. However, they lack more advanced maintenance aspects such as: (1) The specification of different maintenance regimes, like corrective and preventive maintenance; (2) The modelling of inspection intervals with appropriate maintenance actions; (3) The modelling of sophisticated repair strategies. To equip DFTs with such capabilities is a non-trivial process. There exist formalisms where fault trees are augmented with repairs, for example by equipping leaves with repair times [VGRH81] or repair boxes [BCR04]. However, these approaches are not containing any actual maintenance strategies.

Built upon the compositional semantics presented in Section 5.6 we will introduce maintenance to DFTs by:

- (1) Extending basic components with more expressiveness such as degradation;
- (2) Introducing new modules for inspections and repairs;
- (3) Extending DFT gates with repair signals.

These features will enable us to include various maintenance strategies into DFTs. For example, one can specify a periodic inspection of components and initiate a preventive maintenance action of a component if its condition is below a certain threshold. In this chapter we will introduce these three building blocks by formulating their semantics in terms of I/O-MAs.

6.2 Maintainable components

Maintenance in a system starts at the basic components. Hence, while maintenance will improve the whole system performance, maintenance actions like

detailed inspections, repairs, and renewals are performed on the level of the system components. Thus, including maintenance into FTs also starts at the BEs.

Recall the description of BEs in Section 5.2.1 on page 104. A BE can be either functional, i.e. it is dormant or active, or it is failed. However, considering a BE is maintainable, there should be additional states representing degradation. To overcome this issue, the *condition* of a component is modelled via degradation phases, similar to extended fault trees [Buc00]. Moreover, since the BE can be maintained and repaired, it can improve its degradation phase. However, a component does not necessarily support all maintenance features. To distinguish between different maintainable BEs we introduce additional types:

RBE: Repairable basic event;

MBE: Maintainable basic event;

RMBE: Repairable and maintainable basic event.

Hence, the leaves of a maintainable DFT are represented by:

$$\text{Leaves} = \{\text{BE}, \text{RBE}, \text{MBE}, \text{RMBE}\}.$$

To represent these new leaves, we extend the BE semantics with the following attributes:

- (1) Degradation phases;
- (2) Repair signals;
- (3) Maintenance signals;
- (4) Inspection thresholds.

The degradation phases will model the different degradation states of the BE as well as represent its failure distribution. The inspection threshold, represented by action “*threshold*”, indicates the degradation phase where the component is below its required quality and should be maintained. The maintenance signal, represented by action “*maintain*”, models the successful maintenance of a component. The repair signal, represented by actions “*repair*” and “*repaired*”, model the successful repair of a component. An overview of the actions is given in Table 6.1.

Example 6.3. Consider the railway crossing DFT from Figure 5.3 on page 104. The failure of the motors can be described by some degradation phases like “small wear” and “high wear”. Moreover, an inspection based on the wear could be conducted every quarter, such that if the motor shows high wear a maintenance procedure is performed. On the other hand, for the disconnection and sensor failures a dedicated repair unit could be deployed. The repair unit would monitor the status of the sensors and the connection, and induce a repair if a failure occurs.

In the following we give the semantics for the new BE classes in terms of I/O-MAs.

Action	Signal	Context
<i>act</i>	Activation	Communication signal for the activation of DFT elements. Send from parents to children.
<i>fail</i>	Failure	Communication signal for the failure of DFT elements. Send from children to parents.
<i>threshold</i>	Inspection threshold	Indication that a BE reaches a degradation phase where an inspection has an effect. Send from a BE to an inspection module.
<i>maintain</i>	Maintenance action	Communication signal for a successful maintenance action. Send from an inspection module to a BE.
<i>repair</i>	Repair request	Communication signal to request a repair. Send from a BE to a repair unit.
<i>repaired</i>	Repair action	Communication signal for a successful repair. Send from a repair unit to a BE.

Table 6.1: Signals for DFTs with maintenance.

6.2.1 Degradation

Phase-types. The general idea of modelling degradation in BEs is to represent the failure distribution not only by a single rate, but by a degradation process represented by a *phase-type* distribution (PHD). A PHD is a probability distribution constructed out of several exponential distributions. Basically a PHD is a distribution representing the time until absorption represented by a CTMC with an absorbing state. The advantage of PHDs is, that they can approximate arbitrarily closely other probability distribution [JT88]. Consider we replace the failure rate with a PHD in our BEs. How can we specify the degradation we are in? Due to the fact that the CTMC for a PHD can have cycles, it is not possible to specify the degree of degradation per state. A solution is to limit the PHDs to *acyclic phase type distributions* (APHDs). Then to every state a degree of degradation can be assigned such that the successor state has a higher degradation. Note that $\text{APHD} \leq_{\infty} \text{PHD}$. That means, that all finite state APHDs can be represented by an appropriate finite state PHD, whereas there exists at least one finite state PHD that can only be represented by an infinite state APHD [TBT06]. *But how are we obtaining an appropriate APHD for the failure of a component?* The component failures of FTs are often based on empirical data [BP75]. Thus, the empirical data can be used as input to a fitting algorithm that fits the data set to a APHD [HT02]. However, the task of fitting a general data set to a APHD is non-trivial. Thummler et al. [TBT06] presented a novel fitting approach which is numerical more effective and stable than previous methods. The idea was to reduce the search space by restricting the class of PHDs to a smaller class still sufficiently general to represent any non-

(a) PDF of Erlang($k, 2$) distribution for $k = 1, \dots, 7$. (b) CDF of Erlang($k, 2$) distribution for $k = 1, \dots, 7$.Figure 6.1: Erlang($k, 2$) distribution for $k = 1, \dots, 7$.

negative distribution. Therefore, they chose *hyper Erlang distributions* (HErD), and showed that $\text{HErD} \leq_{\infty} \text{APHD}$. Thus, by representing the degradation of a component with a HErD, the BE can efficiently be equipped with an approximation of any non-negative failure distribution. Further, each state represents a degree of degradation such that the successor is more degraded. However, there is no distinction w.r.t. the rate between any two degraded states. Note that the number of degradation phases of a BE is dependent on the fitting of the data set to a HErD.

Figure 6.2: Fitting a ErD to a data set.

Degraded BEs. First of all we have a closer look at the Erlang distribution (ErD). Figure 6.1 shows the *probability density function* (pdf)

$$f(x)(k, \lambda) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!}$$

as well as the *cumulative distribution function* (cdf)

$$F(x)(k, \lambda) = 1 - \sum_{n=0}^{k-1} \frac{1}{n!} e^{-\lambda x} (\lambda x)^n$$

for the ErD $\text{Erlang}(k, \lambda)$ with phases $k = 1, \dots, 7$ and rate $\lambda = 2$. By inspecting the different pdf and cdf curves one can see that with a larger k , i.e. more degradation phases, the time point where the BE probably fails shifts.

Example 6.4. Consider Figure 6.2. The boxes represent a collection of failure data for a component, i.e. the bar represents the percentage of failures of a component class after n years of operation. In this case, the majority of component failures is between 5 and 6 years. A good approximation for this dataset is given by the ErD $\text{Erlang}(7, 1.2)$. Thus, the probability density function of the ErD aligns with the data set.

Figure 6.3 depicts a BE with three degradation phases. The first phase represents the component in perfect condition, whereas subsequent phases represent degraded conditions, until the component has failed. The BE degrades according to an $\text{Erlang}(k, \lambda)$ distribution, where $k = 3$ is the shape of the ErD representing the degradation phases, and λ is the failure rate of a negative exponential distribution.

A HErD consists of several separate ErDs with an initial probability distribution over them. Note that the expressiveness of MAs allows for incorporating also HErDs into the BE semantics. Figure 6.4 depicts a BE with a HErD

consisting of three ErDs: Erlang(2, λ_1) with probability p_1 , Erlang(4, λ_2) with probability p_2 , and Erlang(3, λ_3) with probability p_3 , where $p_1 + p_2 + p_3 = 1$. To accommodate the new failure behaviour of BEs we extend the attachment function to HErDs.

Definition 6.1 (Attachment function with degradation). Let $\mathcal{F} = (V, top, Tp, \delta)$ be a DFT. Then

$$\Omega: V \rightarrow \text{Distr}(\mathbb{N} \times \mathbb{R}_{>0} \times [0, 1]).$$

is an attachment function with $\text{dom}(\Omega) = \{\text{BE}, \text{RBE}, \text{MBE}, \text{RMBE}\}$.

Hence, the attachment function with degradation assigns a distribution over ErDs and dormancy factors to each BE. Thus if $v \in V$ with $Tp(v) \in \text{dom}(\Omega)$, then $\Omega(v) = \text{Distr}(k, \lambda, \alpha)$ with $k \in \mathbb{N}$, $\lambda \in \mathbb{R}_{>0}$ and $\alpha \in [0, 1]$.

6.2.2 Maintenance signals

The inclusion of maintenance on the level of BEs requires the introduction of additional signals. For example, after a BE is repaired it should be able to inform its parents about the change. Moreover, the BE has to encode when maintenance actions will have an impact. For example, a maintenance action on a BE has only an effect if already a certain level of degradation is reached. Note that in reality a maintenance action is often not a 100% return to the initial condition. Thus the maintained component can still have a certain degree of degradation. However, for simplicity of the model we assume that a maintenance action induces a full recovery of the component. In the following we introduce how to model RBEs, MBEs and RMBEs.

Repair. To define RBEs we introduce two signals representing the repair procedure: (a) “*repair*” and (b) “*repaired*”. The “*repair*” signal is an input signal of the RBE. Hence, the RBE waits for a successful repair after its failure. The “*repaired*” signal is an output signal and is used to inform the parents of the RBE that it is functional again. Figure 6.5 depicts the I/O-MA of an active RBE including three degradation phases. After the RBE has failed, it can receive a “*repair*” signal which indicates that the repair of the component was successful. Afterwards the RBE switches into its repaired state and sends out a “*repaired*” signal to communicate to its parents that it is active again.

Inspection. To define MBEs we introduce two signals representing the maintenance procedure: (a) “*threshold*” and (b) “*maintain*”. The “*threshold*” signal

Figure 6.3: Active BE with degradation phases.

Figure 6.4: Active BE with several degradation phases.

Figure 6.5: Active RBE with degradation phases.

is an output signal indicating when a MBE is below the quality for its intended use. Thus, after this point an inspection will trigger a maintenance action. The “*maintain*” signal is an input signal to the MBE. Hence, the MBE waits for a successful maintenance action to return into its non-degraded state. Figure 6.6 depicts the I/O-MA of an active MBE including an inspection threshold. When the MBE is in its first degraded state, i.e. after the first degradation phase, it sends out a “*threshold*” signal. From this point on a maintenance action will have an effect on the MBE. After the threshold the MBE can receive a “*maintain*” signal which has the effect that the MBE returns into its initial state. The combination of a RBE and MBE results into a RMBE as depicted by the I/O-MA in Figure 6.7. To describe when an inspection of an MBE would induce a maintenance procedure, the threshold is included in the attachment function.

Definition 6.2 (Attachment function with threshold). Let $\mathcal{F} = (V, top, Tp, \delta)$ be a DFT. Then

$$\Omega: V \rightarrow \text{Distr}(\mathbb{N} \times \mathbb{R}_{>0} \times [0, 1] \times \mathbb{N}).$$

is an attachment function with $\text{dom}(\Omega) = \{\text{BE}, \text{RBE}, \text{MBE}, \text{RMBE}\}$.

Hence, the attachment function with threshold extends Definition 6.1 with a threshold. Thus, if $v \in V$ with $Tp(v) \in \text{dom}(\Omega)$, then $\Omega(v) = \text{Distr}(k, \lambda, \alpha, q)$ with $k \in \mathbb{N}$, $\lambda \in \mathbb{R}_{>0}$, $\alpha \in [0, 1]$ and $q \in \mathbb{N}_0$ such that $q \leq k$. Note, that q

describes the number of degradation phases after which a “*threshold*” is issued. If $q = 0$, then no maintenance is conducted.

Figure 6.6: Active MBE with inspection threshold.

Figure 6.7: Active RMBE with inspection threshold.

Inactive mode. Recall that the dormancy factor α describes the inactive failure behaviour of a BE. Hence, if a BE is part of a spare module, the dormancy factor adjusts the failure rate while the spare module is not in use (see Section 5.2.2 on page 105). Hence, also RBEs, MBEs and RMBEs need to support this feature. Therefore, we distinguish between an inactive and an active mode. Note that if a BE in active mode fails, it is considered to switch into the inactive mode. Figure 6.8 depicts the I/O-MA for a RMBE with inactive and active mode. The inactive mode is equivalent to the I/O-MA depicted in Figure 6.8 where the failure rate is weighted by the dormancy factor. While the RMBE has not failed it can be activated, i.e. after an “*act*” signal is received, the I/O-MA switches from the inactive to active mode. In active mode the behaviour of the RMBE is equivalent to the active RMBE. However, with the transition from the last degradation phase, the RMBE returns to the inactive mode, since it

is a spare. Recall that spares can be sub-trees. Therefore, a spare component will also get deactivated, if the spare-module fails. To model the deactivation, the RMBE listens for the failure of the root of the spare module. Hence, if the spare module fails, but the RMBE is still operational, it switches into its inactive mode.

Figure 6.8: Spare RMBE with an active and an inactive mode.

6.3 Inspections and repairs

The second step in including maintenance into DFTs is to introduce new models that enable the inclusion of different maintenance strategies. In particular we introduce two new models: (a) *inspection modules* and (b) *repair units*. Those models are no traditional gates in a FT, but rather a specification of the maintenance strategy over components of the FT. For example, an inspection module specifies which component is inspected in what interval, and a repair unit describes in what order components are repaired and how long the repair takes. Note that we write BE instead of RBE, MBE and RMBE whenever clear from the context. In the following we describe inspection modules and repair units and present their semantics in terms of I/O-MAs.

6.3.1 Inspections

To evaluate the quality of components they have to be inspected. Hence, during an inspection the quality of components is evaluated, and based on that a maintenance action may be performed. Usually, inspections are made in a specified

interval. Moreover, during an inspection usually more than one component is inspected. To specify inspections in DFTs, we introduce an inspection module (IM). An IM describes the inspection interval as well as which BEs are inspected.

Definition 6.3 (Inspection module). Let $\mathcal{F} = (V, top, Tp, \delta)$ be a DFT. We define $\mathcal{I} = (V', \lambda, phases)$ as an inspection module for $\mathcal{F}$ such that

- $V' \subseteq V$ where for all $v \in V'$ it holds that $Tp(v) \in \{\text{MBE}, \text{RMBE}\}$;
- $\lambda \in \mathbb{R}_{>0}$ is the inspection rate;
- $phases \in \mathbb{N}$ is the number of Erlang phases until an inspection is carried out.

Note that the inspection interval of an IM is described by an Erlang distribution, and therefore can approximate a fixed time interval. Further, we assume that the time of a maintenance action is neglectable and therefore will be performed instantaneous in the model. Note that an instantaneous maintenance action is not realistic, however it is a sensible way of modelling it. For example, consider that an inspection is conducted every quarter while a maintenance action of a component may take a couple of hours. In such a case one can assume that these couple of hours are covered sufficiently in the time interval of the inspection cycle.

Figure 6.9 depicts an I/O-MA of an IM with a single BE and n phases. The I/O-MA starts at the beginning of an inspection cycle, i.e. the first phase of the Erlang distribution. In each phase the I/O-MA listens if the BE sends its inspection threshold. Hence, the inspection of a BE by an IM is modelled indirectly. If during one inspection cycle the BE has not reached its threshold the IM is reset, represented by the τ -transition. Otherwise, if the BE has sent the “threshold” signal, the IM will issue the maintenance of the component and sends out the “maintain!” signal. This will also lead to the start of a new inspection cycle.

If an IM covers multiple BEs then it has to keep track of which BEs reached their inspection threshold, such that the IM only issues maintenance actions to components that are intended to be maintained. To keep the maintenance signalling simple, the IM sends the “maintain” signals out in a certain order. For example, an IM is assigned to three BEs and the first and third BE reach their inspection threshold before the end of the inspection cycle. Then the IM will send out a “maintain” signal for the first BE and then for the third BE, regardless of which BE first passed its inspection threshold. Therefore, an IM with n assigned BEs will have 2^n different maintenance behaviours. Note that this IM keeps track over which components have reached their threshold and have to be maintained. This information is crucial to answer questions like: *How many components are maintained during an inspection cycle?*, or *What is the number of individual maintenance actions per component in one cycle?* However, if one is only interested in the reliability of the system, then this kind of information is not needed and one can use a simplified IM. The simplified IM with n assigned BEs is defined in the following way: Take the IM from Figure 6.9 as the base model and add for each BE a threshold transition with the same state

change. Further, if one component has send a threshold signal, send out the maintain signal for all components. If a component did not reach the threshold, it still can synchronise on the maintain action due to the input enabling, and will stay in its current degradation state.

Figure 6.9: IM for a single BE with inspection rate λ and n phases.

6.3.2 Repairs

Performing corrective maintenance in a DFT contains the repair of failed components. Hence, if a BE has failed a corrective maintenance strategy should define how and when the BE will be repaired. To describe repair strategies in DFTs we use repair units (RUs). A RU can be assigned to several BEs and determines in which order the BEs are repaired. Further, the RU assigns a repair time, described by an exponential distributed delay, to the individual BEs or a combination of BEs. We distinguish between two types of repair behaviour of RUs: (a) a group repair; and (b) an individual repair.

Definition 6.4 (Repair unit). Let $\mathcal{F} = (V, top, Tp, \delta)$ be a DFT. We define $\mathcal{R} = (V', \Theta, \phi)$ as a repair unit for $\mathcal{F}$ such that

- $V' \subseteq V$ where for all $v \in V'$ it holds that $Tp(v) = BE$;
- $\Theta = (\text{Group} | \text{FCFS})$ defines the strategy of the repair unit;
- $\phi = \begin{cases} V' \rightarrow \mathbb{R}_{>0} & \text{such that } \phi(v') = \lambda \text{ for each } v' \in V' \text{ if } \Theta = \text{FCFS} \\ \lambda \in \mathbb{R}_{>0} & \text{if } \Theta = \text{Group.} \end{cases}$

Note that the definition allows to add several more repair strategies. For example one could chose a priority queue or a round robin scheduling.

6.3.2.1 Group repairs

Figure 6.10 depicts a RU for two BEs with a group repair. Thus, the RU waits until all assigned BEs failed and only then starts with the repair, described by the dashed transition with rate μ . Afterwards the RU sends out “repair” signals in order to communicate a successful repair of the assigned BEs.

Figure 6.10: Deterministic group repair unit.

6.3.2.2 FCFS repairs.

Figure 6.11 depicts a RU for two BEs with a first come first serve (FCFS) policy. Thus, the BEs are repaired individually and in the order of their failure. Therefore, each BE has a dedicated repair rate. After a BE is successfully repaired he RU sends out the “repair” signal. For the FCFS RU every possible failure order has to be explicitly modelled. ’

Figure 6.11: First come first serve repair unit.

6.4 Fault maintenance trees

We include maintenance in the FT framework by redefining the BEs behaviour and the communication within the gates. This class of fault trees is defined as *fault maintenance trees* (FMTs) and can handle condition-based maintenance, inspections, repairs, and spare management.

Definition 6.5 (Fault maintenance tree). Let $\mathcal{T} = (\mathcal{F}, Im, Ru)$ be a fault maintenance tree (FMT) such that

- $\mathcal{F}$ is a DFT;
- Im is a finite set of inspection modules $\mathcal{I}$ for $\mathcal{F}$;
- Ru is a finite set of repair units $\mathcal{R}$ for $\mathcal{F}$.

Impact of maintenance on DFTs. Including maintenance into the fault tree framework also affects how the fault tree has to be modelled and interpreted. Whereas inspections and the resulting maintenance action have an impact on the failure behaviour of the BEs, they do not change the structural properties. However, by including corrective maintenance, i.e. repairs, the structural behaviour of the DFT can change. In the following we provide a small example.

Example 6.5. Consider the railway crossing DFT from Figure 5.3 on page 104, in particular the sub-tree describing the barrier failure. A barrier failure occurs if the switching unit fails, i.e. the switch fails before the first motor, or if both motors fail. Now consider the switch and both motors can be repaired such that a barrier failure can be prevented or reversed. Recall that if the first motor fails, the switching unit will switch over to the second motor, and after this the failure of the switching unit is irrelevant. However, since the first motor could be repaired while the second motor is in use, the switching unit would be needed to switch back to the first motor if the spare motor fails. By inspecting the DFT, we can see that the change to the first motor is possible with a failed switch, since there is no dependency between the second motor and the switch. Hence, to model this dependency, the switching unit has to also be connected to the second motor. Figure 6.12 depicts this extension. Note that if the motors have a dormant failure behaviour there is a chance that the switch fails and then the second motor fails while the first motor is still active, leading to a barrier failure. The DFT in Figure 6.13 accommodates for that. However, compared to the simple sub-tree in the beginning with 3 gates and 3 BEs, the repairable sub-tree has 6 gates and 5 BEs.

This example shows that repairs in DFTs with PAND gates lead to non coherent behaviour. Therefore, additional checks are needed to prevent such unwanted behaviour.

Figure 6.12: Barrier failure.

Figure 6.13: Barrier failure with repairs including dormant modes.

6.4.1 Repair communication

The inclusion of repairs in DFTs has an influence on the total system failure. Consider an AND gate as the top level event, and assume that all children emitted a failure signal. Hence, the top level event failed and therefore the system has failed. However, if one child gets repaired, the condition of the AND gate is not fulfilled anymore. This means, the AND gate would not be in a failed state anymore, and therefore the system would be back up running.

Including repairs into the DFT framework opens up a plethora of different possibilities of how to represent the system behaviour. For example, if the top level event has failed, would this mean that all still active but not failed components should switch to a dormant mode? Lets consider a car with a flat tire. The car would be out of service until the tire is repaired or changed. During this time, the other tires would not be actively used, and therefore dormant. However, the period of time until the repair of the flat tire is finished compared to the life span of the other tires would not make a big impact. Moreover, the stress induced on the remaining tires would be equivalent to a short stop and not equal to e.g. the spare tire in the trunk, which is truly dormant. This

behaviour is present in a majority of systems where repairs can be conducted. In the following, we assume that the still active BEs keep being active after a system failure.

In the following we do not treat activation and assume all FMT components are active. However, the extension with activation is equivalent to the concept presented in Section 5.3. Moreover, the repair semantics of the FMT gates can be seen as a conservative extension of the DFT I/O-MA semantics. Thus the failure behaviour is equivalent, however, the I/O-MAs have to keep track over what children already failed or were repaired.

6.4.1.1 OR gate

The OR gate fails if one of its children fails. Conversely, it will be repaired if all its children are repaired. This has the effect that the repairable OR gate has to track which children already failed and which children were repaired.

Figure 6.14 illustrates the OR(2) gate with repair tracking. The main change w.r.t. the standard OR gate is that each failure will spawn a separate failure signal and failed state. The reason for that is due to the fact that the OR gate has to keep track of the children that fail before a repair will be conducted. Hence, the OR gate has two symmetrical parts, one in its operational state, and one in its failed state. If all children are successfully repaired, the OR gate will emit a repair signal “repaired”.

Failed: If 1 out of n children fails.

Repaired: If n out of n failed children are repaired.

6.4.1.2 AND gate

The AND gate fails if all its children fail. Therefore, after the AND gate has failed it is sufficient when one child gets repaired to be operational again. Note that the repairable AND gate has the same structure as the repairable OR gate, where only the failure and repaired signals of the gate itself are at different positions. In particular, an AND(n) gate is equivalent to an OR(n) gate where the “fail!” transitions for all states with $1 < k < n$ failures are reversed and labelled with “repaired!”.

Figure 6.15 shows an AND(2) gate with repairs. Consider the initial state and assume we receive a failure signal from the first component. Then the AND gate moves into the upper failure branch. If now the first component sends a repair signal before the failure of the second component appears, the AND gate moves back into its initial state. If all components fail, then the AND gate will send out a failure signal and moves into its failed state. Now any component can be repaired to trigger a repaired signal. Then the AND gate can move back into its active mode where $n - 1$ out of n components are failed.

Figure 6.14: OR (2) gate with repairs.

Failed: If n out of n children failed.

Repaired: If 1 out of n failed children is repaired.

6

Figure 6.15: AND (2) gate with repairs.

6.4.1.3 VOT(k) gate

The VOT(k) gate fails if k out of n children fail. Therefore, it will be operational when $(n - (k - 1))$ out of n failed children get repaired. Note that the repairable VOT(k) gate has the same structure as the repairable OR and AND gate. Moreover, the “fail!” and “repaired!” transitions are equivalent to the repairable AND(n) gate up to $(k - 1)$ component failures and to the repairable OR(n) gate from k up to n component failures.

Figure 6.16 illustrates a repairable VOT(2) gate with 3 children. By inspecting the I/O-MA, we can see that the failure signal “fail!” is only emitted after the second child has failed. In particular, if we limit us in the initial state to the signals of component 1 and 2, the resulting sub-I/O-MA we obtain is equivalent to the AND(2) gate. The same holds for the combination of 1 and 3 as well as 2 and 3. Moreover, if we consider the state where already $k - 1$ failures were received as initial state, while limiting the transitions to the children that have not yet failed, we obtain a repairable OR(2) gate as sub I/O-MA. Thus a VOT(k) gate with n children contains n AND(k) gates as well as n OR(k) gates.

Failed: If k out of n children failed.

Repaired: If $(n - (k - 1))$ out of n failed children are repaired.

6.4.1.4 PAND gate

The PAND gate fails if all n children fail in the correct order, i.e. from left to right. The PAND gate is repaired and moves to a non failed state if $k \geq 1$ out of n components are repaired in the reversed failure order. Interestingly, due to repairs, the PAND gate does not have an actual sink state anymore. That means, if the components are not failing in the correct order, instead of going into an actual sink state from which the PAND gate cannot fail anymore, the failing order can be corrected due to repairs. Consider the switching unit from Figure 6.12a. In the not repairable case, the PAND gate would fail if the switch fails before the motor would fail, but not if the motor fails before the switch fails. In the repairable case, however, after the motor failed before the switch failed, the motor can be repaired again and therefore the PAND gate is back in a valid failing order.

Figure 6.17 depicts a repairable PAND(2) gate. The figure clearly shows how the repair of components can bring the PAND gate back to its correct failure behaviour instead of residing in a sink state. If initially the second component fails first, the PAND gate condition is violated. However, if the second component is repaired before the first component fails we can go back into the initial state. If the first component also failed there are two possible options. On the one hand, the first component gets repaired before the second such that the PAND gate is in its previous state. On the other hand, if the second component gets repaired, then the failing order is restored.

Figure 6.16: Depiction of a repairable VOT(2) gate with 3 children.

Failed: If n children fail in the correct order.

Repaired: If $k \geq 1$ out of n failed children are repaired in the reversed order.

6.4.1.5 SPARE gate

The **SPARE** gate fails if its primary as well as all spares fail. Thus if no components are available, i.e. the primary and all spare-modules are in their fail states, the **SPARE** gate sends its fail signal. In case that either component, regardless if it is the primary or a spare component, gets repaired, the **SPARE** gate can again activate and utilise that repaired component. Note that all BEs that are part of a primary or spare sub-tree switch into their inactive mode after a failure. *The SPARE gate with repairs considers the primary and spare module as equal.* Therefore, also if the primary is repaired, the **SPARE** module will not switch back to the primary directly. On the contrary, the primary behaves now equivalent to a spare module and will only get activated when needed. However, the order of the spare modules will stay the same. Consider a **SPARE** gate with one primary and two spare-modules. If the primary fails, the first spare-module is

Figure 6.17: PAND(2) gate with repairs.

6

activated, assuming it is available. If the primary is repaired in the mean-time and the first spare module fails, then the second spare module will get activated, and not the primary. Only after the second spare module fails, i.e. the last spare module, the primary gets activated again.

Figure 6.18 depicts a repairable **SPARE** gate with one primary and one spare component. After the primary failed, and the spare got activated, it is possible that the primary gets repaired before the spare fails. In this case, the **SPARE** gate can activate the primary again, after the spare fails. If the primary as well as the spares fail, the **SPARE** gate is in its failed state. However, if any of the spares or the primary gets repaired, the **SPARE** gate can activate the module again and is in its active mode. Note that the currently used module will be utilised until it fails and only then the next available module will be activated. For example consider the case in Figure 6.18 where the primary failed and the spare got activated. Now the primary could be repaired before the spare fails. However, the primary will stay inactive until the spare failed. Only then the **SPARE** gate will activate the primary again.

Failed: If the primary and n out of n spares fail.

Repaired: If the primary or 1 out of n spares is repaired.

Figure 6.18: SPARE gate with one primary and one spare and repairs.

6.4.1.6 FDEP gate

Since the FDEP gate does not emit an actual failure signal to a parent, but rather lets its dependent components fail, it will also not emit a repaired signal. Moreover, if the trigger of an FDEP gate gets repaired, this does not mean that its dependent events get repaired. For example, the trigger event could be a oil leakage in a motor, which would have the effect of a motor failure. However, if the oil leakage is repaired, this does not automatically repairs the motor. Hence, the dependent events could still be failed after the trigger got repaired.

Figure 6.19 depicts the repairable FDEP gate (with ordered failure propagation). The addition to the non repairable FDEP gate is the repair transition for the trigger event. Hence, after the trigger event has failed and induced the failure of the dependent events, the trigger can be repaired. This will send the FDEP gate back to its initial status and stops the failure propagation. Note that the repair of the dependent events is not induced or tracked by the FDEP gate.

Failed: If the trigger event failed.

Repaired: If the trigger event is repaired.

Figure 6.19: Ordered FDEP gate with repairs.

6.5 Smart semantics

For the state space generation of an FMT we apply the same concept as for DFTs described in Section 5.5.2. However, due to the increased complexity of the individual I/O-MAs for the FMT elements, the intermediate state space before minimisation can be significantly high. In the following we present a concept that exploits redundancies in BEs which allows one to define “smarter” I/O-MA models for certain FMT elements. The idea is to find equivalent BEs that are connected to the same gate and *aggregate* them. In the following we provide more details about when such an aggregation is possible and how it changes the I/O-MA semantics.

6.5.1 Aggregation of components

During the modelling of a FT lot of redundancy is added to understand the systems behaviour. For example, consider the railway crossing DFT from Figure 5.3 on 104 with the system failure OR gate “Level crossing failure” and the connected OR gate “Barrier failure”. The barrier failure OR gate could be semantically omitted by connecting the “Switching unit” and “Motors” directly to the top level OR gate. However, the “Barrier failure” OR gate is introduced to describe a specific failure group. This is important for the human perception of the FT, but for the reliability analysis this behaviour can be simplified.

To reduce DFTs Junges et al. [JGKRS15] presented a graph transformation approach. In case of FMTs, however, we have to consider not only the graph structure of the FT but also the inspection and repair structures. In [GKSLR14] we presented a *smart semantics* approach regarding pattern matching in the FT and finding equivalent behaviour in the components. In the following we explain how this can be applied to FMTs.

The aggregation of equivalent BEs can significantly reduce the state space and is achieved by applying *counting abstraction* [GS92]. Lets consider a simplified BE with only two states, active and failed. By composing n such BEs in parallel this would result in 2^n configurations. However, if those BEs describe the same failure as well as repair behaviour, we can reduce the possible configurations down to $n + 1$. Figure 6.20 depicts a comparison of the composition of two simplified BEs in Figure 6.20a and the aggregated BE in Figure 6.20b.

In the first case, all possible combinations are created. However, if the BEs will have the same behaviour, then it is sufficient enough to only count how many BEs already failed. Thus, the aggregated BE just counts the number of failures and repairs, represented by the action “ $fail^{i/n}$ ” for the i -th failure out of n BEs and “ $repaired^{i/n}$ ”, respectively.

Figure 6.20: Comparison of two simplified BEs.

6.5.2 Effects on the semantics

To be able to incorporate the smart aggregation of BEs, the I/O-MA semantics has to be adapted. In the following we do not treat activation and assume all FMT components are active. To obtain the inactive versions, the same concept as in Section 6.2 has to be applied. Note, that the aggregation is limited to BEs that are children of static gates, i.e. AND, OR, and VOT(k) gates.

Figure 6.21: I/O-MA of n aggregated BEs with repairs.

Aggregated BEs. First let's consider the I/O-MA for n aggregated BEs as depicted in Figure 6.21. Counting the number of failed BEs is represented by the action “ $fail^{i/n}$ ”, where i/n stands for i out of n BEs have failed, and vice versa “ $repaired^{i/n}$ ” for the number of repaired/functional BEs. Moreover, the aggregated BE has to combine the failure rates of the individual BEs. Since we only aggregate BEs with the same failure and repair behaviour, the failure rate of all BEs will be equal. Therefore, we can lump the failure rates for n BEs with failure rate λ together to $n \cdot \lambda$. Note that with each failure the number of BEs decreases until all n BEs have failed.

Definition 6.6 (Aggregation of BEs). Let $\mathcal{T} = (\mathcal{F}, Im, Ru)$ be an FMT and $B \in V$ a set of components such that $Tp(b) \in \text{Leaves}$ for all $b \in B$. The set of components B can be aggregated to one component if and only if for all $b_i, b_j \in B$ it holds that:

- $Tp(b_i) = Tp(b_j)$;
- $\Omega(b_i) = \Omega(b_j)$;
- $\text{parents}(b_i) = \text{parents}(b_j)$ and $\text{parents} \in \{\text{AND}, \text{OR}, \text{VOT}(k)\}$;
- for all IM $\mathcal{I} = (V', \lambda, \text{phases}) \in Im$ with $b_i \in V'$ it follows that $b_j \in V'$, and vice versa;
- for all RU $\mathcal{R} = (V', \Theta, \phi) \in Ru$ with $b_i \in V'$ it follows that $b_j \in V'$, and vice versa.

Note that the aggregation of BEs with degradation phases is not as straight forward as without. In the following we provide an example of aggregating BEs with degradation. Figure 6.22b depicts an I/O-MA for two aggregated BEs with two degradation phases and failure rate λ . For comparison, Figure 6.22a depicts the result of composing two BEs with two degradation phases. Besides, an aggregated version with inspections and repairs is depicted in Figure 6.22c. Recall that only BEs with the same failure rate λ as well as the same number of degradation phases k should be aggregated. The failure rate for the first n phases in the aggregated BE is given by $n \cdot \lambda$ where n is the number of BEs to aggregate. After k degradation phases, however, there is a chance that the degradation phases differ for the individual BEs. Thus the aggregated BE will have a probabilistic decision describing the probability of the different degradation combinations for the n BEs. The probability and number of combinations depends on the number of BEs and degradation phases. The number of combinations for the different degradations is determined by the number of partitions $P(k)$ up to size n [And98]. After one out of n BEs has failed, the failure rate will change to $(n - 1) \cdot \lambda$. If the BEs can be maintained, their inspection threshold t has to be equivalent, such that the threshold action can be added after t degradations.

Example 6.6. Consider 3 BEs such that each BE has 5 degradation phases. The number of combinations of non-negative integers that sum up to 5 is given by $P(5) = 7$ such that:

$$\begin{aligned}
 5 &= 5 \\
 4 + 1 &= 5 \\
 3 + 2 &= 5 \\
 3 + 1 + 1 &= 5 \\
 2 + 2 + 1 &= 5 \\
 2 + 1 + 1 + 1 &= 5 \\
 1 + 1 + 1 + 1 + 1 &= 5.
 \end{aligned}$$

(a) I/O-MA of two BEs with component degradation.

(b) I/O-MA of two aggregated BEs with component degradation.

(c) I/O-MA of two aggregated BEs with inspection and repair.

Figure 6.22: Aggregation of BEs with degradation phases.

Figure 6.23: Smart I/O-MAs for the repairable AND, OR, and VOT(k) gate.

However, we are only interested in the summations with at most three integers, denoted by $P_3(5)$. Thus, we have $P_3(5) = 5$ combinations of how three BEs are degraded after five degradation phases. Note that with a chance of 20% one of the three BEs will fail after 5 degradation phases. Now let's consider the partition (2, 2, 1). Thus two BEs are in the second degradation phase where one BE is in its first. Now, after three more degradations, there is a chance that one of the BEs fails. Thus the aggregated BE would perform three more degradation phases resulting in $P_3(3) = 3$ combinations. Thus the resulting degradation phases for the three BEs would be given by (5, 2, 1), (4, 3, 1), and (3, 3, 3).

Aggregated Gates. If a gate has children that can be aggregated, then also the gate has to be adapted to be able to synchronise on the failure actions of the aggregated BE. Figures 6.23a to 6.23c depict the smart I/O-MA semantics of the OR, AND, and VOT(k) gate, respectively. Therefore, the I/O-MAs listen for 1, n or k out of n failures, respectively. However, if not all children of a gate can be aggregated, a composition between a smart and normal I/O-MA has to be made.

Example 6.7. Consider the partial depiction of the I/O-MA $\mathcal{A}$ in Figure 6.24 for an AND(3) gate with two identical BEs. Since only two BEs are identical we cannot combine all three BEs to one. Nonetheless the smart aggregation of

Figure 6.24: Smart I/O-MA of an AND(3) gate with 2 aggregated BEs.

BEs can be applied. Therefore we have to combine a smart AND(2) gate with a normal AND(2) gate. This has the effect that the general structure is equivalent to an AND(2) gate, while inserting $n - 1$ extra states for n aggregated BEs in each branch of the functional as well as the failed part.

6.6 Related work

Bouissou and Bon [BB03] introduced boolean logic driven Markov processes (BDMPs) as a formalism combining the advantages of fault trees and Markov models. In particular they extended the fault tree formalism with shortcuts to model complex dynamic behaviours. They also support failure distributions via phases as well as exploit symmetry to aggregate similar BEs. Further, they included repairable components, however, only directly on the BE level without further repair strategies than a repair time. Besides, the application field where BDMPs are currently tailored to is the analysis of safety and security [PCB10; KBCHPC14] compared to the reliability and maintainability approach of FMTs.

Bobbio and Codetta-Raiteri [BCR04] introduced parametric DFTs with repair boxes. The repair boxes are equivalent to the RUs in FMTs modelling corrective maintenance. However, no preventive maintenance is supported by these models. Moreover, there exist several limitations in the expressiveness of their formalism. For example, only BEs are supported as spares and only symmetric SPARE gates can share spares.

Raiteri et al. [RFIV04] introduced repairable fault trees (RFT). They assign to each repair action a trigger condition as well as a repair policy. The trigger

event for a repair action can be (a) the failure of a basic event, or (b) a condition expressed as a function of a set of basic events. Where (a) represents corrective maintenance, (b) is useful to model preventive maintenance. However, their framework only allows option (a).

6.7 Conclusion

In this chapter we included preventive as well as corrective maintenance into DFTs. Therefore, we extended the behaviour of basic components with degradation phases as well as the capability to get maintained as well as repaired. To model preventive maintenance, we introduced IMs. They can be assigned to several BEs and will initiate a maintenance procedure for the BEs that reached a certain level of degradation during an inspection interval. Corrective maintenance is modelled via RUs, which allow to specify different repair strategies. We included a group repair as well as a FCFS strategy. As for IMs, several BEs can be assigned to a RU. With this ingredients we defined FMTs.

To define the semantics of FMTs, we extended the I/O-MA semantics from Chapter 5. First of all, we defined new I/O-MAs for the basic components as well as I/O-MAs for the IM and RUs. Then we extended the I/O-MA semantics of the DFT gates with repairs. Finally, we proposed some smart I/O-MAs that exploit properties of the FMT to further reduce the state-space.

Fault maintenance trees in practice

CHAPTERS 5 and 6 have shown the importance of reliability, availability, maintainability and safety (RAMS) analysis for dependable systems. Moreover we introduced fault maintenance trees (FMTs), an extension of dynamic fault trees (DFTs) with preventive as well as corrective maintenance procedures. While we laid out the advantages of FMTs, a pending aspect is to show their applicability. This ranges from tool support and scalability to how FMTs can be applied to a real world scenario. In this chapter we aim to establish FMTs as a modelling and analysis framework within RAMS and show their capabilities in practice.

A pressing aspect to adapt FMTs into the FTA process within industry is the availability of tool support. For example, a company that works with a certain FTA tool will not just adapt to a new model and tool only on its promises. One important aspect therefore is that the work-flow of a company is optimised on the tool of their choice. Besides, using a commercial tool also provides a support guarantee over a certain time frame. Then, how can FMTs be introduced into industry? To make FMTs tempting for industry, they have to be presented within practical use cases and show an advantage over the current FTA methods. However, to be able to do so, we first need to have tool support for FMTs. Since commercial tools like Isographs FaultTree+ [Iso16] are closed source, we cannot just add FMT support to them. Hence, we have to implement a prototypical tool that supports FMTs and can be used for case studies. Besides the implementation of a prototypical tool, the applicability of FMTs has to be shown. Therefore, it is important to investigate how FMTs and their analysis can have a positive effect on existing RAMS analysis techniques.

In the first half of this chapter we introduce a toolchain for the analysis of FMTs. In particular we focus on DFTCALC [ABVGS13b] and its extension to FMTs. Besides, we introduce an alternative toolchain for the analysis of FMTs based on MAPA [Tim13] as an experimental test bed. Moreover for the interaction with DFTCALC, especially for third parties, we introduce a web-based GUI for the main tool functionality. Finally, we conduct several benchmarks for classical DFTs presenting the capabilities of DFTCALC.

In the second half we show a proof of concept of analysing FMTs and show their capabilities for modelling and analysing a realistic case study. For the analysis we use DFTCALC as well as the alternative toolchain based on MAPA. In

particular we focus on a RAMS analysis for a railway trajectory in the Netherlands provided by the the engineering consultancy company Movares Nederland. We will investigate different corrective as well as preventive maintenance strategies. Moreover we show how the use of FMTs shows a direct effect of the different strategies within the FTA.

Origins of the chapter. This chapter presents how fault maintenance trees can be used in practice and provides tool-support and case studies as presented in:

- D. Guck, J.-P. Katoen, M. I. A. Stoelinga, T. Luiten, and J. Romijn. “Smart railroad maintenance engineering with stochastic model checking”. In: *Proceedings of the 2nd International Conference on Railway Technology: Research, Development and Maintenance (Railways 2014)*. Vol. 104. Civil-Comp Proceedings. Civil-Comp Press.
- D. Guck, J. J. Spel, and M. I. A. Stoelinga. “DFTCalc: Reliability centered maintenance via fault tree analysis (tool paper)”. In: *Proceedings of the 17th International Conference on Formal Engineering Methods (ICFEM 2015)*. Vol. 9407. Lecture Notes in Computer Science. Springer Verlag.

Organisation of the chapter. In Section 7.1 we discuss the tool support for FMTs and in Section 7.2 we provide classical cases studies for DFTs. Section 7.3 discusses RAMS in railways and provides a practical case study for FMTs in the context of railway engineering. Section 7.4 discusses the scalability of analysing FMTs and Section 7.5 concludes the chapter.

7

7.1 The tool architecture of DFTCalc

The main functions our tool should support are:

- The translation of a FMT specification into I/O-MAs for each FMT element;
- The generation of the I/O-MA $\mathcal{A}_{\mathcal{F}}$ representing FMT $\mathcal{F}$ using compositional aggregation;
- The analysis of the I/O-MA with respect to:
 - The reliability of the system;
 - The mean time to failure;
 - The availability of the system.

As a basis of our tool architecture for FMTs we build upon DFTCALC [ABVGS13b]. It takes as input an FMT description and generates the corresponding I/O-MA which then can be analysed against several reliability metrics. The DFTCALC toolchain was developed as the successor of the CORAL tool

Figure 7.1: DFTCALC toolchain.

[BCS07], the implementation of the extensible and compositional framework for DFTs as introduced by Boudali et al. [BCS10]. Thus, the DFTCALC framework includes the analysis of DFTs with respect to the compositional aggregation approach as presented in Section 5.5.2 on page 117. In the following we give a detailed overview of DFTCALC and its support of FMTs.

7.1.1 Architecture

DFTCALC is implemented in C++. It integrates several tools and has just over 10000 lines of code.¹ The architecture of DFTCALC is depicted in Figure 7.1 and can be partitioned into three parts:

- (1) Translation;
- (2) Generation;
- (3) Analysis.

The *translation* is responsible for parsing the DFT input and create the DFT in the internal data structure of DFTCALC. Moreover, the translation creates descriptions of all DFT elements and their communication. The *generation* is responsible for generating the state space for each DFT element and then applying the compositional aggregation. The last part, the *analysis*, computes the desired reliability metric. Section 7.1.4 provides a more in depth description of these parts.

7.1.2 Integrated tools

DFTCALC combines dedicated code with several existing analysis tools. In particular it utilises the CADP toolbox, MRMC, and IMCA. In the following we give a short overview about these tools and their function in DFTCALC.

CADP. CADP (Construction and Analysis of Distributed Processes) is a toolbox for the design of asynchronous concurrent systems developed at INRIA [GLMS12]. CADP supports the generation of Markov models from Lotos NT [Cha+11] specifications. Moreover, CADP supports parallel composition as well as several minimisation algorithms. In particular, the CADP engine is able to apply the compositional aggregation approach.

¹DFTCALC is open source and available at <https://github.com/utwente-fmt/dftcalc>

MRMC. MRMC (Markov reward model checker) is a model checker for discrete-time and continuous-time Markov reward models [KZHHJ11]. It supports the verification of properties expressed by the logic's PCTL and CSL as well as their reward extensions CSRL and PCTRL. There is also a CTMDP extension available which provides analysis techniques based on [BHHZ11]. In particular, MRMC provides fast algorithms for the reliability analysis in case no non-determinism is present and the I/O-MA of the DFT. Then the I/O-MA can be reduced to a CTMC.

IMCA. IMCA (interactive Markov chain analyser) is a tool for the quantitative analysis of MRAs [GHHKT14]. In particular, it supports the analysis of MAs against unbounded reachability, time- and interval-bounded reachability, expected-time objectives, and long-run average objectives. Moreover it supports the analysis of MRAs against expected-reward, time-bounded expected rewards and long-run average reward objectives. In particular, IMCA focuses on calculating the minimum and maximum values if non-determinism is present in the I/O-MA of the DFT. Moreover, it is used for the mean time to failure and availability computations.

7.1.3 DFTCalc web interface

The full functionality of DFTCALC is available via a command line interface (CLI). For the ease of use by third parties, we extended DFTCALC with a GUI via a web interface. The web-interface is realised with the use of PUPTOL [BR13], a framework to publish a command line tool on the web. The web interface provides the main functions of DFTCALC as well as an extra plot function. The web interface is depicted in Figure 7.2 and works as follows:

- (1) The user can choose from a set of predefined examples or
- (2) define his/her own DFT using the text box.

Then the user can choose between the dependability metrics. This can be

- (a) the reliability for one or more mission times T , or
- (b) the probability to fail within a given interval $[T_1, T_2]$, or
- (c) the mean time to failure.

The default analysis back-end of DFTCALC for (a) is MRMC, and IMCA for (b) and (c). However, DFTCALC is also able to use IMCA for (a). Therefore the user can enforce the use of IMCA for (a). Besides, it is possible to set BEs already to failed in the *evidence* field. Before the user starts the analysis, he can also choose the level of precision as well as if he wants to compute the minimum or maximum values in case of non-determinism. When all options are set, the user can execute DFTCALC via the “Show Result” button and obtain the result, or the “Show Plot (and store dataset)” button and also obtain a plot (and save the data). The plots can display the reliability over time as well as the MTTF.

(select example) 1

DFT:

```

toplevel "crossing";

"crossing" or "sensors" "barrier" "controller" "disconnection";
"sensors" 2of4 "sensor1" "sensor2" "sensor3" "sensor4";
"barrier" or "switching" "motors";
"switching" pand "switch" "motor1";
"motors" wsp "motor1" "motor2";
"detection" fdep "disconnection" "sensor1" "sensor2" "sensor3" "sensor4";

"disconnection" lambda=0.01 dorm=0;
"sensor1" lambda=0.1 dorm=0;
"sensor2" lambda=0.1 dorm=0;
"sensor3" lambda=0.1 dorm=0;
"sensor4" lambda=0.1 dorm=0;
"controller" lambda=0.05 dorm=0;
"switch" lambda=0.2 dorm=0;
"motor1" lambda=0.5 dorm=0.5;
"motor2" lambda=0.5 dorm=0.5;

```

2

1
2

a ☐ Compute unreliability in interval [0,T],
for mission times T (T>0), given as
☐ list of values
☐ range, from: to: step:

Model checker: ☐ MRMC ☐ IMCA

b ☐ Compute unreliability in interval [T1,T2]
T1: T2:

c ☐ Compute MTTF
(for plot: to: step:)

Evidence:

Error bound: Prob: Time: DFT:

Version: Verbosity: ☐ Coloured output ☐ No pointmarks

Data set name:

Powered by [puptol](#). Visit your [puptol session overview](#).

Figure 7.2: DFTCALC web-interface.

Metric	Options
Reliability	Probability that the DFT fails in time interval $[0, T]$. Probability that the DFT fails within the time interval $[T_1, T_2]$ without failing before.
Mean time to failure	Expected time of reaching the top-level event of the DFT.
Availability	Long-run average time in an up state of the DFT.

Table 7.1: Metrics for DFTCALC

Type	Key	FMT element
Gates	or	OR gate
	and	AND gate
	kofn	VOT(k,n) gate
	pand	PAND gate
	wsp	SPARE gate
	fdep	FDEP gate
Leaves	lambda	failure rate
	dorm	dormancy factor
	phases	number of Erlang phases
	threshold	inspection threshold
	repair	repair rate
RU	ru	FCFS repair unit
IM	kinspn	Inspection module with Erlang(k, n) as inspection cycle.

Table 7.2: Main keywords for the DFTCALC Galileo input language.

```

1  Toplevel    =  toplevel "tle";\n
2              Gates
3  Gates      =  "gate" type "child1" ... "childn";\n
4              Gates | RU | IM | Leaves
5  RU         =  "ru" ru "be1" ... "ben";\n
6              RU | IM | Leaves
7  IM         =  "im" kinspn "be1" ... "ben";\n
8              IM | Leaves
9  Leaves     =  BE | RBE | MBE | RMBE | \eof
10 BE         =  "be"      lambda= $\lambda$  dorm= $\alpha$ ;\n
11              Leaves
12              |  "be"      lambda= $\lambda$  dorm= $\alpha$  phases= $n$ ;\n
13              Leaves
14 RBE        =  "rbe"      lambda= $\lambda$  dorm= $\alpha$  repair= $\mu$ ;\n
15              Leaves
16              |  "rbe"      lambda= $\lambda$  dorm= $\alpha$  phases= $n$  repair= $\mu$ ;\n
17              Leaves
18 MBE        =  "mbe"      lambda= $\lambda$  dorm= $\alpha$  phases= $n$  threshold= $k$ ;\n
19              Leaves
20 RMBE       =  "rmbe"     lambda= $\lambda$  dorm= $\alpha$  phases= $n$  threshold= $k$  repair= $\mu$ ;\n
21              Leaves

```

Figure 7.3: Galileo FMT file structure.

7.1.4 DFTCalc's program flow

In the following we provide more information about the program flow of DFT-CALC. Therefore, we describe in more detail the processes involved from the FMT input up to the analysis result.

```

1  toplevel "crossing";
2
3  "crossing" or "sensors" "barrier" "controller" "disconnection";
4  "sensors" 2of4 "sensor1" "sensor2" "sensor3" "sensor4";
5  "barrier" or "switching" "motors";
6  "switching" pand "switch" "motor1";
7  "motors" wsp "motor1" "motor2";
8  "detection" fdep "disconnection" "sensor1" "sensor2" "sensor3" "sensor4";
9
10 "inspection" 4insp1 "motor1" "motor2";
11 "repair" ru "disconnection" "sensor1" "sensor2" "sensor3" "sensor4";
12
13 "disconnection" lambda=0.01 dorm=0 repair=0.5;
14 "sensor1" lambda=0.1 dorm=0 repair=1;
15 "sensor2" lambda=0.1 dorm=0 repair=1;
16 "sensor3" lambda=0.1 dorm=0 repair=1;
17 "sensor4" lambda=0.1 dorm=0 repair=1;
18 "controller" lambda=0.05 dorm=0;
19 "switch" lambda=0.2 dorm=0;
20 "motor1" lambda=0.5 dorm=0.5 phases=3 threshold=2;
21 "motor2" lambda=0.5 dorm=0.5 phases=3 threshold=2;

```

Figure 7.4: Galileo specification of the crossing DFT.

7.1.4.1 Translation from Galileo FMT to DFTCALC

The input to DFTCALC is a FMT in Galileo's textual format [SDC99]. This intuitive format describes a FMT top-down from its root to the basic components. Each sub-tree is identified by a name, logically connected with other sub-trees by gates, and then refined down to the basic components. The main key words for the FMT input specification are given in Table 7.2 and Figure 7.3 describes the file structure to specify FMTs in Galileo.

Example 7.1. Figure 7.4 depicts the Galileo specification for the railway crossing DFT from Figure 5.3 on page 104 in Chapter 5. Further it includes the maintenance aspects described in Example 6.3 on page 139 in Chapter 6.

This specification is then input to the DFTCALC toolchain.

The first step of the translation is the parsing of the FMT Galileo specification. The `dft2lntc` tool within DFTCALC will obtain the specification and parses the FMT. After `dft2lntc` successfully parses the DFT it generates the following output:

- (a) Files for all FMT elements (`.lnt`) describing the behaviour of each element;
- (b) One file (`.exp`) for the communication between all FMT elements;
- (c) One file (`.svl`) describing the composition.

In particular, each element of the FMT has a Lotos NT [Cha+11] specification template describing its behaviour as presented in Section 5.6.

Example 7.2. Consider the `.lnt` file depicted in Figure 7.5. It is a template for a BE with active and inactive mode. The communication channels available for a BE are the activation `ACTIVATE`, the failure `FAIL`, as well as the failure rate `RATE_FAIL`. The block from line 17-21 describes the activation procedure of the

```

1  module TEMPLATE_BEAPH(TEMPLATE_COMMON) is
2  process BEproc
3  [FAIL : NAT_CHANNEL, ACTIVATE : NAT_BOOL_CHANNEL, RATE_FAIL : NAT_NAT_CHANNEL]
4  (cold: BOOL, initial: STATUS, phases: NAT) is
5  var
6      status : STATUS,
7      nr : NAT
8  in
9      status := initial;
10     nr := 0;
11     loop
12     select
13     (
14         * When this BE is activated, it will set the status to ACTIVE
15         * but only if it was DORMANT (not active and not failed)
16         *)
17         ACTIVATE (?nr,FALSE) where (nr==0 of NAT);
18         if (status == DORMANT) then
19             status := ACTIVE
20         end if;
21     nr := 0
22     (
23         * When this BE is in the DORMANT state, it can fail
24         * with the failure rate associated with this state.
25         * The rate is later introduced by renaming of RATE_FAIL!!1.
26         * If the failure event is triggered, go into the FAILING state.
27         *)
28     []
29     if ((not (cold)) and (status == DORMANT)) then
30         RATE_FAIL (!1 of NAT,1 of NAT);
31         phases := phases-1;
32         if (phases < 1) then
33             status := FAILING
34         end if
35     end if
36     (
37         * When this BE is in the ACTIVE state, it can fail
38         * with the failure rate associated with this state.
39         * The rate is later introduced by renaming of RATE_FAIL!!2.
40         * If the failure event is triggered, go into the FAILING state.
41         *)
42     []
43     if (status == ACTIVE) then
44         RATE_FAIL (!1 of NAT,2 of NAT);
45         phases := phases-1;
46         if (phases < 1) then
47             status := FAILING
48         end if
49     end if
50     (
51         * When this BE is in the FAILING state (caused by one
52         * of the failure rates), signal this to the system
53         * using FAIL !0. After this, go into the FAILED state.
54         *)
55     []
56     if (status == FAILING) then
57         FAIL (!0 of NAT);
58         status := FAILED
59     end if
60     end select
61     end loop
62     end var
63     end process
64 end module

```

Figure 7.5: LotosNT specification of a BE.

BE. The block from line 29-35 and line 42-49 describe the failure behaviour of the BE in inactive and active mode, respectively. Finally the last block from line 55-59 describes the failure of the BE.

The communication between the components, the hiding of signals as well

```

1  (* Number of rules: 6*)
2  hide
3    and_be1,
4    and_be2,
5    f_be1,
6    f_be2
7  in
8    label par using
9      (* and          be1          be2          *)
10     "ACTIVATE!0!FALSE" * _          * _          -> ACTIVATE,
11     "ACTIVATE!1!TRUE"  * "ACTIVATE!0!FALSE" * _          -> and_be1,
12     "ACTIVATE!2!TRUE"  * _          * "ACTIVATE!0!FALSE" -> and_be2,
13     "FAIL!0"           * _          * _          -> FAIL,
14     "FAIL!1"           * "FAIL!0"    * _          -> f_be1,
15     "FAIL!2"           * _          * "FAIL!0"    -> f_be2
16   in
17     "and_p1_c2.bcg"
18   ||
19   total rename "RATE_FAIL!1!2" -> "rate 0.5" in "be_p1_cold.bcg" end rename
20   ||
21   total rename "RATE_FAIL!1!2" -> "rate 0.5" in "be_p1_cold.bcg" end rename
22 end par
23 end hide

```

Figure 7.6: Example of a .exp communication description.

as the application of the failure behaviour to BEs is specified in the .exp file. An example of the communication of an AND gate with two BEs is given in Example 7.3. Then, the .svl file (Script Verification Language) specifies to perform compositional aggregation based on the description given in the .exp file.

7.1.4.2 Generation of I/O-MAs

The second step is the generation of the I/O-MA. This is done with the help of the CADP tool-set [GLMS12]. In particular the .svl file is executed by CADP, which leads to the compositional aggregation of the individual elements as described in Section 5.5.2 on page 117.

The first step of the generation is the generation of the individual I/O-MAs. Therefore CADP creates each I/O-MA from its Lotos NT specification. Hence, the .lnt specifications are interpreted by CADP and for each a .bcg file is created, i.e. a binary file format for the I/O-MA. Note, that if the FMT has identical elements, e.g. several AND gates with three inputs, only one .bcg file has to be created. To reduce further unnecessary computation, DFTCALC creates a cache of already generated .bcg files. Thus, with incremental use, DFTCALC builds a library of .bcg files for FMT elements.

The second step is the compositional aggregation. Therefore, the .bcg files are composed according to the .exp description. Note, that the order of how the I/O-MAs are composed is not necessarily the order of the .bcg files as listed in the .exp file. CADP decides the order based on a heuristic with the goal of keeping the intermediate state space of the composition as small as possible. For the minimisation, CADP hides the actions specified in the .exp file, as well as includes the rates for the BEs.

Example 7.3. Consider the .exp file as depicted in Figure 7.6. Assume the AND gate “and” and the first BE “be1” are composed. The AND gate will send

an activation to the BE (line 11), which will be represented by action `and_be1` in the composition. Besides, the BE will send a fail signal to the AND gate (line 14), which will be represented by `f_be1`. As defined in line 3 and 5, those two actions are subject to hiding. Moreover, line 19 specifies the renaming of action `RATE_FAIL !1 !2` into a rate with $\lambda = 0.5$.

Note that due to the use of the CADP tool-set, there are some limitations w.r.t. the failure distribution. In particular, CADP only supports MAs without discrete probability distributions, i.e. interactive Markov chains. Thus, tool support restricts the failure distributions of BEs to Erlang distributions.

The third step in the generation is to convert the generated I/O-MA from the `.bcg` format into the corresponding format for the analysis back-end. In case the FMT should be analysed with MRMC, the `.bcg` is transformed with the `imc2ctmdp` tool into a `.ctmdp` and `.lab` file. In particular, the I/O-MA is translated into a CTMDP according to [Joh08]. Note that this is valid if the I/O-MA has only Dirac probability distributions. Moreover, in case there exists no non-determinism, the transformation yields a CTMC. In case IMCA is the chosen analysis back-end, the `.bcg` file is translated into an `.ma` file with the `bcg2imca` without further transformation.

7.1.4.3 Analysis

The third and final step in DFTCALC is the analysis of the FMT, i.e. the computation of metrics. Therefore, two back ends are connected. Depending on the resulting I/O-MA and the chosen metric, MRMC or IMCA is utilised. An overview of the metrics supported by DFTCALC is given in Table 7.1. MRMC supports the reliability analysis, whereas IMCA supports all metrics, i.e. the reliability, the mean time to failure, and the availability. The default option for the reliability analysis for the failure up to a given mission time T is MRMC. In case the resulting I/O-MA is in fact a CTMC, MRMC provides the faster algorithms to calculate the reliability, since IMCA only facilitates an algorithm tailored to continuous-time models with non-determinism. Note that for reliability questions like “*What is the reliability over 10 years and how is the probability of failure for each year?*”, DFTCALC can provide intermediate results for each year during the reliability computation of the final mission time.

7.1.5 FMT extension of DFTCalc

The clear distinction between local component and global system information together with the compositional semantics of I/O-MAs makes DFTCALC highly flexible: New components can be added or existing components adapted by specifying their behaviour I/O-MA semantics and including the corresponding Lotos NT specification to the tool’s library. Thus, to add the capability of analysing FMTs in DFTCALC several new Lotos NT specifications as well as new signal communication processes had to be included.

Table 7.2 lists all main keywords in DFTCALC, where `threshold`, `repair`, `ru` and `ninspk` are keywords only for FMTs, whereas the other keywords can be used in DFTs. The standard RU strategy used in DFTCALC is the first come first

Figure 7.7: Prototypical FMT toolchain with MAPA.

serve strategy. The keyword for the IM also encodes the number of inspection phases as well as their rate, i.e. `kinspn` corresponds to $\text{Erlang}(k, n)$.

The stable version of DFTCALC only includes repair communication for the static gates. This decision was made on the basis of an experimental evaluation of DFTCALC w.r.t. FMTs. That is, we incrementally included the FMT elements needed w.r.t. the demands of the case-studies presented in Section 7.3.1. Note that the inclusion of the new communication channels in the I/O-MAs as presented in Chapter 6, as well as the Erlang distributions for inspections and the BEs, lead to bigger state spaces for the individual I/O-MAs of the FMT elements. This has the effect that also the intermediate state space during the compositional aggregation increases.

7.1.6 Alternative FMT specification

As an addition to the FMT generation via CADP, we defined an approach using the Markov automata process algebra (MAPA) [Tim13]. MAPA is a process-algebraic language for specifying MAs. It was introduced for easy specification of large MAs in a concise manner. Moreover, several types of reduction techniques have been defined for the MAPA language [TKPS16]. They are implemented in the tool SCOOP with the goal of optimising the specifications and decreasing the state space of the corresponding MAs while staying bisimilar [Tim11]. Moreover, LTSMin [Kan+15] also supports the MAPA language as well as the on-the fly confluence reduction.

Similarly to the CADP tool in DFTCALC, MAPA is used to generate the corresponding I/O-MA of the specified FMT, which is then input to MRMC or IMCA. Figure 7.7 depicts the process of the prototypical FMT toolchain with MAPA. Note that this approach is not included in the DFTCALC toolchain and is used as a standalone. This design decision was made since DFTCALC is build around the compositional aggregation approach, where the MAPA toolchain differs from that. The input is a MAPA specification of the FMT.

MAPA FMT. The MAPA FMT specification combines the specification of the FMT elements with the FMT description. That is, each FMT element is described as a process in MAPA, equivalent to the Lotos NT specifications for CADP. In particular, the MAPA specification can be divided into three parts:

- The communication;
- The FMT elements;
- The FMT structure.


```

1  type Components = {0..n}
2
3  comm (sendActivateComponent, receiveActivateComponent, activateComponent),
4      (sendFailComponent, receiveFailComponent, failComponent),
5      (sendInUse, receiveInUse, inUse),
6      (sendBroken, receiveBroken, broken),
7      (sendFailSpareGate, receiveFailSpareGate, failSpareGate)
8
9  encap sendActivateComponent, receiveActivateComponent,
10      sendFailComponent, receiveFailComponent,
11      sendInUse, receiveInUse,
12      sendBroken, receiveBroken,
13      sendActivateSpareGate, receiveActivateSpareGate,
14      sendFailSpareGate, receiveFailSpareGate
15
16  hide break, activateComponent, failComponent, inUse, broken, failSpareGate
17
18  reach fail
19
20  System(active:Bool) =
21      not(active) => ( sendActivateComponent(1).System[active := T]
22                      ++ receiveBroken(1).fail . Deadlock[]
23                      ++ receiveInUse(1).System[active := T])
24      ++ active      =>      receiveFailComponent(0, 1).fail.Deadlock[]
25
26  Deadlock = F => dead.Deadlock[]

```

Figure 7.8: Example MAPA specification for the communication of DFTs.

The communication describes the input/output communication as well as the system initialisation and end state. Figure 7.8 depicts an example MAPA specification for the communication of a DFT. Line 3-7 describes the communication for the activation and failure. Line 9-14 describes the encapsulation of actions, and line 16 which actions are subject to hiding.

Example 7.4. Consider the activation procedure of the MAPA specification in Figure 7.9. Line 3 says that the output signal `sendActivateComponent` communicates with the input signal `receiveActivateComponent` and results in the new action `activateComponent`. Line 9 then defines the encapsulation of the output and input action `sendActivateComponent` and `receiveActivateComponent` respectively. Moreover line 16 defines the hiding of the resulting `activateComponent` action.

Note that there are some extra communication channels besides the activation and failure. Those are needed to properly define the gates in MAPA. Besides, the fail states are defined via line 18, such that all states that are reached via the action `fail` are considered as a top-level event failure. Line 20-26 then defines the initialisation of the system as well as the failure of the top-level event.

The FMT elements are described by individual processes in the MAPA specification. Figure 7.9 depicts an example MAPA specification for a BE with an exponential failure distribution. Note that the process `BasicComponent` describes the behaviour of the BE, and `Baisc` is calls this process and simplifies the definition for the user.

Example 7.5. Consider the process `activateComponent` in Figure 7.9. Line 2 describes the initialisation of the BE with an identifier. Moreover, the state of a BE is defined by the variable `inUse`, i.e. if the BE is claimed, as well as `broken`, i.e. if the BE failed. Further, a dormant as well as active failure rate is assigned to the BE.

```

1 BasicComponent
2   (i:Components, inUse:Bool, broken:Bool, failRateSlow:Nat, failRateFast:Nat) = not(
3       inUse & not(broken)
4           => receiveActivateComponent(i).BasicComponent[inUse :=
5               T]
6       ++ not(inUse) & not(broken) => <failRateSlow>.BasicComponent[broken := T]
7       ++ inUse & not(broken)      => <failRateFast>.BasicComponent[broken := T]
8       ++ inUse & not(broken)      => sendInUse(i).BasicComponent []
9       ++ broken                   => sendBroken(i).BasicComponent []
10      ++ broken                    =>
11      sum(parent:Components, sendFailComponent(parent, i).BasicComponent [])
12
13 Basic(i:Components, failRateSlow:Nat, failRateFast:Nat) =
14   BasicComponent[i, F, F, failRateSlow, failRateFast]

```

Figure 7.9: Example MAPA specification for a BE.

```

1 init System[F] || (* FMT initialization *)
2 Or[1, 2;3;4;5] || (* crossing *)
3 Vot[2, 2, 6;7;8;9] || (* sensors *)
4 Or[3, 10;11] || (* barrier *)
5 Pand[4, 12;13] || (* switching *)
6 Spare[5, 13;14;15] || (* motors *)
7 Fdep[16, 5;6;7;8;9] || (* detection *)
8 Basic[5, 0, 0.01] || (* disconnection *)
9 Basic[6, 0, 0.1] || (* sensor1 *)
10 Basic[7, 0, 0.1] || (* sensor2 *)
11 Basic[8, 0, 0.1] || (* sensor3 *)
12 Basic[9, 0, 0.1] || (* sensor4 *)
13 Basic[12, 0, 0.05] || (* controller *)
14 Basic[13, 0, 0.2] || (* switch *)
15 Basic[14, 0.5, 0.5] || (* motor1 *)
16 Basic[15, 0.5, 0.5] || (* motor2 *)

```

Figure 7.10: Example MAPA specification for the crossing DFT.

The FMT structure is the last part of the MAPA specification. Figure 7.10 depicts the specification for the railway crossing DFT (see Figure 7.4) in MAPA. Note, that the children are specified via a queue of their respective identifiers. Besides, the first entry of each element assigns the identifier to the element. Consider line 2 of Figure 7.10. The OR gate “crossing” has the identifier 1 and the elements with identifier 2, 3, 4 and 5 as children.

SCOOP/LTSMIn. To generate the I/O-MA from the MAPA specification two tools can be used, SCOOP (implemented in Haskell) as well as LTSMIn (implemented in C++). Note that instead of the compositional aggregation, both tools generate the whole state space at once while using an on the fly confluence reduction. Besides, where SCOOP is a dedicated tool for the generation of MAs from MAPA, LTSMIn is a complete tool set with model-checking capabilities, that includes the functionality of SCOOP via `mapa2lts`. Moreover, LTSMIn also supports lumping of CMTCs via `ltsmin-reduce`. Thus, if the resulting I/O-MA can be reduced to a CTMC, LTSMIn can further reduce the state space.

	DFTCalc	MAPA FMT
input	Galileo DFT	MAPA specification
Generation method	Compositional aggregation	On the fly confluence reduction
license requirements	CADP	-
automated toolchain	✓	×
Tailored GUI	web-based	-
FMTs embedded in toolchain	✓	×
MRA ready	×	✓

Table 7.3: Differences between the two toolchains.

7.1.7 Use cases for the two toolchains

The two introduced toolchains have several similarities but are different in their core functionality. Table 7.3 lists a comparison between the DFTCALC toolchain and the MAPA toolchain for FMTs. First of all, both tools aim to generate the I/O-MA for an FMT based on the semantics presented in Chapter 6. Moreover, as an analysis back-end, both toolchains use MRMC as well as IMCA. However, the tools are using a different method for generating the I/O-MA that is the input to the analysis tools. DFTCALC implements the compositional aggregation approach, as presented in Section 5.5.2 on page 117, using the CADP toolset. Therefore, a CADP license is required to be able to use DFTCALC. In contrast, with the MAPA FMT toolchain the generation is realised via parallel composition with an on the fly confluence reduction based on [Tim13] and implemented in SCOOP. Another difference is the user-friendliness of the tools. DFTCALC is implemented as a fully automated toolchain, supporting the Galileo DFT input format as well as provides a web-based GUI to explore the main functionality of the tool. This is not given for the MAPA FMT toolchain. There, the input is directly in the MAPA language and the process from the generation to the analysis is not automated. Hence, in DFTCALC the FMT specifications are internal, where for MAPA FMT the FMT specification is included in the input. Since the MAPA language supports also MRAs with action rewards [GTHRS14a], it can be used to experiment with reward/cost extensions of the FMTs. In the case of DFTCALC this is not possible at the moment. In conclusion, the DFTCALC toolchain is better suited for the overall usage of analysing FMTs, where the MAPA FMT toolchain can be regarded as an experimental test bed.

7.2 Analysis of classical DFTs

To demonstrate the performance and capabilities of DFTCALC, we first conduct several case studies of classical DFTs. In the experiments we focus on the following points:

- (1) Comparison between CORAL[BCS07] and DFTCALC;
- (2) Benchmark tests on DFTCALC;

- (3) Influence of activation procedures in the generation;
- (4) Interpretation of DFTCALC's results.

Note that the comparison of CORAL and DFTCALC was conducted on a different machine than the other experiments, due to restrictions of CORAL.

7.2.1 Classical DFT models

In the following we use some classical DFT models from literature: The multi processor computing system (MCS) [BD05], the cardiac assist system (CAS) [BCS10], and the fault-tolerant parallel processor (FTPP) [BCS10]. In the following we describe those examples in more detail.

Multiprocessor computing system (MCS). Figure 7.11a depicts the physical description of a multiprocessor computing system (MCS). The system consists of two computing modules CM_i (i=1 or 2). They are connected via a bus, powered by a power supply (PS) and share a spare memory module M3. Each computing module consists of a processor P_i, a memory module M_i and two hard drives, a primary Di1 and a backup Di2. The system fails, if either both computing modules fail, the communication bus fails or the power supply fails. A computing module fails, if both hard drives fail, or the processor fails or the memory fails (where the shared spare memory is not available or also fails). The corresponding DFT is depicted in Figure 7.11b. We also analysed an extension of the MCS (4CM) which contains two more copies of the computing modules connected to the same bus with an own power source and a separate spare memory. In the following we provide a more in-depth description of the MCS, CAS and FTTP DFTs.

The cardiac assist system (CAS). Figure 7.12 depicts a DFT representing a cardiac assist system (CAS) consisting of three subsystems: the CPU, the motor and pump units. If either one of these subsystems fails, then the entire CAS fails, as modelled by the top level OR gate. The CPU unit consists of a primary (P) and a backup (B) CPU, as indicated by the SPARE gate. The primary and backup CPU are subject to a common cause failure, modelled by the CPU FDEP gate: if either the crossbar switch (CS) or the system supervisor (SS) fails, the primary and backup CPU become unavailable. The motor unit consists of a primary (MA) and a backup (MB) motor. If the primary fails, the motor switching component (MS) will turn on the backup motor. Because of the PAND gate the failure of the switching component can then be ignored. Finally, the pump unit consists of two pumps (PA and PB), which share a common cold spare (PS).

Fault-tolerant parallel processor (FTPP). The FTTP-*n* models a redundant computer system consisting of four groups of *n* processors and is depicted for *n* = 4 in Figure 7.13. The system consists of 4*n* processors divided into four logical groups. Each group is equipped with a shared cold spare. Furthermore,

Figure 7.11: The multiprocessor computing system (MCS).

Figure 7.12: The CAS case study.

a network element physically connects one processor of each group to the system. If a network element fails, all connected processors become unavailable. In our set-up, all network elements have a failure rate equal to 0.017, and all processors have a failure rate equal to 0.11.

7.2.2 Results

We present the results of three different sets of experiments. The first part shows the initial DFTCALC implementation in comparison to its predecessor CORAL [BCS07] by comparing the MCS, CAS, and FTPP case studies w.r.t. the reliability of the system. The second set of experiments provides a benchmark

Figure 7.13: The FTTP-4 case study.

w.r.t. the current implementation of DFTCALC and focuses on the generation of the I/O-MA. The third set of experiments shows the difference in the state space generation of DFTCALC w.r.t. the activation within DFTs. Furthermore, we show how to interpret the results provided by DFTCALC.

Comparison. The comparison between DFTCALC and CORAL was conducted on a single core of a 2.7 GHz Intel Core2Duo processor with 2GB RAM running on Linux. The difference between the two tools is shown in Table 7.4. We compared the MCS for 2 and 4 CMs, the CAS, and the FTTP- n for $n = 4$ and $n = 5$. For each case study we computed the reliability in terms of the probability that the system fails until a given mission time t , with $t = 10000$ for MCS and CAS, and $t = 1$ for FTTP. The computation time includes each analysis step, i.e. the translation of the DFT, the generation of the I/O-MA as well as the analysis. In this experiment we used the model checker MRMC. The average speed up of DFTCALC to CORAL is approximately a factor of 2. Moreover, DFTCALC has a better reduction w.r.t. DFTs containing FDEP gates.

DFTCalc benchmarks. The following experiments were conducted on a 2.5 GHz Intel Core i5 processor with 16GB RAM running on OSX with IMCA as analysis back-end. We performed benchmarks on variations of the MCS and CAS DFT with a total set of 27 different DFTs.

Recall the MCS DFT from Figure 5.9 on page 116. We consider variations

based on this DFT in the following way: Let m be the number of MCSs of which k have to be operational. Moreover, each MCS contains n CMs, where each CM has either a single power supply (sp) or a redundant power supply (dp). Recall the MCS DFT from Figure 5.9. For the CAS we changed the number of motors needed as well as included extra functional dependencies for the motor and pump failures. Note that the resulting I/O-MAs had only deterministic choices. Figure 7.14 depicts different benchmark results:

- (a) The number of DFT elements against the number of states of the resulting I/O-MA;
- (b) The number of DFT elements against the number of transitions in the resulting I/O-MA;
- (c) The number of states in the I/O-MA against the maximum number of states during the generation of the I/O-MA;
- (d) The number of transitions in the I/O-MA against the maximum number of transitions during the generation of the I/O-MA;
- (e) The number of DFT elements against the memory peak consumption of the generation process;
- (f) The number of DFT elements against the runtime of DFTCALC.

For (a) - (f) we used a log scale on the x-axis and the y-axis. Plot (a) and (b) show that the growth of the I/O-MA is relatively linear with the number of DFT elements. Plot (c) and (d) show that also the reduction of the state space of the I/O-MA is almost linear. However, the maximal state space can vary depending on the DFT structure. The CAS DFTs show this very well. While the state spaces for the standard CAS as well as the CAS with additional FDEPs is similar, the maximum stat-space during generation is much larger for the CAS DFTs with additional FDEPs. Plot (d) depicts the maximum memory consumption of DFTCALC during the generation. One can observe that the memory consumption growth is more exponential than proportional to the number of DFT elements. Plot (e) depicts the execution time of DFTCALC, i.e. the time for the translation, generation and analysis. The runtime increases linear with the number of DFT elements.

Model	Tool	Time (s)	P(fail)	States	Transitions	Speedup
MCS 2CMs	CORAL	131.492	0.998963	18	55	1
	DFTCALC	55.395	0.998963	18	55	2.37371
MCS 4CMs	CORAL	339.752	0.997927	151	992	1
	DFTCALC	201.461	0.997927	151	992	1.68644
CAS	CORAL	135.155	0.0460314	16	50	1
	DFTCALC	51.267	0.0460314	16	50	2.64794
FTPP-4	CORAL	491.114	0.0192186	142	923	1
	DFTCALC	234.905	0.0192186	72	386	2.09069
FTPP-5	CORAL	730.761	0.0030616	2167	27438	1
	DFTCALC	603.630	0.0030616	400	3369	1.21061

Table 7.4: Results of the case studies.

Figure 7.14: Benchmark results for DFTCALC in log-scale.

Model	Gates	BEs	Activation	States	Transitions	Max. States	Max. Transitions
CAS	10	10	spares	16	36	84	304
			all	14	34	49	133
MCS	10	11	spares	18	37	6438	32202
			all	12	31	220	803
FTPP-4	21	20	spares	72	312	45823	230596
			all	66	306	7020	32200
CPS	5	12	spares	39	71	918	3140
			all	38	70	134	291
SF	6	7	spares	15	36	383	1500
			all	14	35	64	138

Table 7.5: Differences DFTCALC w.r.t. the activation procedure.

Activation in DFTCalc. Note that in the first version of the DFT semantics and the corresponding DFTCALC implementation, the activation procedure described in Section 5.3 on page 110 was done from the root node. Thus all elements were equipped with activation signals and had an inactive and active behaviour. However, if an element is not part of a spare module, and therefore active from the start, all the inactive behaviour can be discarded, which reduces the state space of the component. Besides the introduced case studies we also use the cascaded PAND system (CPS) [BCS10] and an instance of a Sensorfilter (SF) [Boz+11] case study in this comparison.

Table 7.5 shows the effect of differentiating between active and inactive elements in the generation of the model. The table gives a quick overview for the MCS, CAS, FTTP-4, CPS and SF case study with a complete activation as well as only spare module activation. While the final state spaces are small, it is the size of the largest intermediate models that matter, since these determine the amount of memory required. By comparing the results for this five DFTs, one can already observe that the intermediate state space is significantly reduced by omitting the activation of the complete DFT. In particular, for these cases the maximal state space reduction for the intermediate state space during generation lies between 72% and 96%, and the state space reduction for the final state space lies between 3% and 33%. Note, that the final state space differs between the two modes, due to the omission of the initial activation procedure. This points the significance of distinguishing the active and inactive DFT part beforehand.

Interpretation of results. In the following we show how the analysis results obtained from DFTCALC can help in understanding a system failure. In particular we have a look at the MCS and CAS case study. Figures 7.15a and 7.15b are depicting the failure probability over time as well as the mean time to failure for the MCS and CAS case study, respectively.

Figure 7.15a provides the failure probability until a mission time of 100000 hours for the 2CM and the 4CM system. One can observe that the probability to fail for the system with 4CMs is lower in the beginning and converges to 1

Figure 7.15: Reliability and MTTF of the (a) MCS and (b) CAS DFTs.

after 10000h, as the MCS with 2CMs. Further, the mean time to failure is ca. 1000h later for a MCS with 4CMs.

Figure 7.15b depicts the failure probability for a fully functional CAS system and for one with a broken motor switch until a mission time of 40000 hours. Comparing the failure probabilities, a defect of a motor switch in a CAS has a significant impact on its failure probability. Where for a fully functional CAS the mean time to failure lies at about 9000h, an initial defect of the motor switch shifts the mean time to failure to about 6000h.

7.3 RAMS in railways

FTs and therefore FTA is often used in railway engineering. For example, in the guide for RAMS analysis by the Dutch Railinfrastructure manager ProRail [Pro10], FTA is one of the methods described for reliability analysis of railways. In the following we want to investigate how the use of FMTs in railway engineering can benefit from the FTA results.

Railway systems are bound by high performance standards. In the Netherlands for instance, the government poses severe financial consequences for violating performance objectives to the companies involved in railway transportation and maintenance. These performance objectives are stated as quantitative requirements on the number of delayed or cancelled train services.

Moreover, safety objectives are of utmost importance. This includes, among others, the safety of passengers, train/maintenance personnel, residents and passers-by. As for performance requirements, quantitative requirements are imposed by the Dutch government on all safety-critical systems and subsystems, such as automatic train protection. Therefore, the engineering of systems includes a demonstration of compliance with the quantitative safety requirements. Hence, without a convincing quantitative proof, new systems are not allowed to be implemented and used in the Dutch railways.

To assess the performance of a railway system or subsystem, *FTA* is a popular method. As described by the guide for RAMS analysis by ProRail [Pro10], *FTA* is suitable for reliability analysis. However, since ProRail's biggest interest lies in maintaining the infrastructure, *FTA* is limited in its expressiveness. That means *FTA* may not yield trustworthy and truly useful performance expectancy results in the context of railway engineering. In particular, there is a strong relationship between

- (1) preventive maintenance and failure rates, and
- (2) corrective maintenance and the repair rate.

The likelihood of a failure occurring is closely related to the amount and quality of preventive maintenance being performed. For failure data that is being monitored, this is usually unknown. Moreover, preventive maintenance practice may vary, e.g. leading to more failures when replacement intervals become longer. Likewise, the repair rate is closely related to corrective maintenance and depends on spare parts, reachability of the components, repair or replacement strategies, etc.

In the remainder of this chapter, we focus on applying FMTs to a railway case study. In particular, we conduct experiments that show the effect of using FMTs to analyse corrective as well as preventive maintenance strategies.

7.3.1 Railway case study

To investigate how FMTs can be used in practice, we analyse a set of fault tree models constructed by Movares to carry out RAMS analysis of a major railroad corridor. The actual data of the FTs from Movares are confidential and cannot be displayed. Therefore we have used anonymised data and altered minor details of the FTs.

The main focus of the case study lies on the RAMS analysis on the railway safety systems of a railway trajectory at one of the major crossing-points in the Netherlands between north-south and east-west railway traffic. The goal of the analysis is to verify that the rail trajectory fulfils the railway system specifications. Here, the focus lies on the availability of the system on the rail trajectory. The failure occurrences are defined by three failure categories:

Category I. Severe disruption in *both directions*, such that no train can ride.

Category II. Severe disruption in *one direction*, such that no train can ride.

Category III. Minor disruption which leads to unpunctuality.

Our experiments have been conducted on several DFTs, describing major functionalities of the safety system of the railroad trajectory. The analysis includes rail-side safety equipment such as signalling, train detection, relay cabinets, high voltage cabinets and similar. Note that we use the following abbreviations in the fault tree descriptions:

Category	Failure described by the FT
I	Failure of several relay cabinets Failure of several switches Disruption of a level crossing
II	Failure of a switch
III	Failure in the authorisation

Table 7.6: FTs of the railway case study.

- *ATB*: short for “automatische treinbeïnvloeding” which is a Dutch automatic train protection system. It will override the driver’s controls if it detects a failure in reducing speed at a caution signal or a failure to observe the speed limit.
- *GRS*: which is a track circuit system by “General Railway Section Company”. It is used to detect the absence of trains on tracks.
- *PSSSL*: short for “prikspanningspoorstroomloop” which is a peak voltage track circuit system. It is mainly used at stations as well as rusty rails.

In this case study we focus on *different strategies for corrective and preventive maintenance*, expressed with FMTs. Note that the FMTs used in this case study are all static, hence only containing OR, AND and VOT(k) gates. We provide two sets of experiments, one focusing on corrective maintenance w.r.t. availability and on focusing on preventive maintenance w.r.t. reliability. The first set concentrating on corrective maintenance has the following properties:

- Corrective maintenance under two different maintenance strategies:
 - Dedicated repair units for each component;
 - Pooling of components assigned to repair units.
- Availability of the system as metric.
- Analysis with the MAPA FMT tool chain.

The second set focusing on preventive maintenance has the following properties:

- Preventive maintenance with pooling of component inspections and different inspection cycles.
- Comparison with corrective maintenance with pooling of components and different repair times.
- Reliability of the system as metric.
- Analysis with DFTCALC.

Moreover, we test the scalability of analysing FMTs with DFTCALC with focus on the generation of the I/O-MA. Before discussing the results we have a closer look at the FT structure of the different failure categories. An overview of the different FTs is given in Table 7.6.

7.3.1.1 Category I failure

Failure of several relay cabinets. This failure is induced by a disruption of several relays on an operated track. This can happen by an actual defect of either the relay cabinets, a defect of the high voltage cabinets or a defect in the power cables. A failure on the operated track will occur if either two different failures occur, e.g. a cable fails and a relay cabinet fails, or if two of the same

Figure 7.16: Category I FT of the severe relay cabinet failure.

category are failing, e.g. two high voltage cabinets are down. The whole FT consists of 30 BEs connected by 4 VOT(k) gates and 4 OR gates. There exists also another FT with 66 BEs. An abstract illustration of this FT is given in Figure 7.16.

Failure of several switches. The failure of several switches will induce a category I failure. The FT lifts the failures of the previous described FT to the context of switches. Hence the failures leading to a failure of several switches are given by: The GRS detection is failing, or the PSSSL detection is failing, or a cable fails. The whole FT consists of 26 BEs connected by 4 VOT(k) gates and 4 OR gates.

Disruption of a level crossing. The failure of a level crossing will induce a category I failure since the whole rail trajectory must be closed in this situation. The level crossing will fail if the operating system fails as well as an erroneous activation of the crossing occurs. An erroneous activation can be induced by several detection or component failures. In total they are grouped in five events. The first event is a GRS detection failure. The second event is a PSSSL detection failure. The third event is a relay cabinet outage. The fourth event is a failure of one of the several cables. The fifth event is induced by a defect of the level crossing installation itself. All events can either occur in a station or on the midsection. The whole FT consists of 58 BEs connected by 8 OR gates and one AND gate. A second variant of this FT with an extended amount of security cables which increases the number of BEs to 350.

7.3.1.2 Category II failure

Failure of a switch. The failure of a single switch on the rail trajectory will block one rail direction and thus induces a category II failure. The corresponding FT consists of three different OR events:

- The first event is the unavailability of a currently operated switch. A failure of a switch is again divided in three occurrences. The first occurrence is a failure on the rail trajectory on a currently used switch. The second occurrence is a failure in one of the stations on a currently used switch. The third occurrence is the failure of the cables in the station connected to a currently operated switch.
- The second event is the unreliability of a switch. This is similar to the first event, with the difference that all currently not operated switches are considered.
- The third event describes a failure of a release approval for shunting. This happens if either one of the control knobs fails or one of the cables fails.

The whole FT consists of 27 BEs connected by 9 OR gates.

7.3.1.3 Category III failure

Failure in the authorisation. This failure can be induced by several operational failures on the rail trajectory. Those can be divided in five different failure events connected by OR gates:

- The first event is induced by a signalling failure on the rail trajectory. This can be caused by any signal light failing or by failure of the cables.
- The second event is an unjust occupation. This failure is induced by wrong occupation signals due to a failure of the train detection.
- The third event is an impossibility to operate a switch, where it is in correct position. This failure is induced by a failure of the train detection.
- The fourth event is an unjust closing of a level crossing. This failure is induced by a failure of detection, a relay cabinet, or of a high voltage cabinet.
- The fifth event is a faulty ATB code. This failure is induced by a failure of the ATB functionality of a railway section.

The whole FT consists of 40 BEs connected by 6 OR gates.

7.3.2 Effects of corrective maintenance

The following experiments were conducted on a dual core Linux machine with 2GB of RAM with the MAPA tool-chain depicted in Figure 7.7, using SCOOP for the generation, LTSMin for the lumping of CTMCs and IMCA for the unavailability analysis. For investigating effects of corrective maintenance of the FTs, we use *two different repair strategies*:

- (1) In the first strategy, each component has a dedicated repair procedure, i.e. each component can be repaired at any time by having a dedicated RU. This is the strategy Movares has considered for their analysis.
- (2) In the second strategy we consider one RU per group of components, i.e. polling a set of similar BEs together and assign one RU to them. This is

System	#BEs	#States	Strategy 1 Dedicated repair	Strategy 2 Grouped repair
(a)	40	12,393	$2.8 \cdot 10^{-6}$	$2.8 \cdot 10^{-6}$
(b)	27	90,720	$1.926 \cdot 10^{-4}$	$1.926 \cdot 10^{-4}$
(c)	30	46,656	$8.71 \cdot 10^{-12}$	$1.2 \cdot 10^{-11}$
	66	2985983	$4.44 \cdot 10^{-11}$	$6.3 \cdot 10^{-11}$
(d)	26	14,400	$4.44 \cdot 10^{-9}$	$7.15 \cdot 10^{-9}$
(e)	58	200,447	$6.8 \cdot 10^{-8}$	$6.8 \cdot 10^{-8}$
	350	11653631	$2 \cdot 10^{-7}$	$2 \cdot 10^{-7}$

Table 7.7: Unavailability results for (a) Failure in the authorisation; (b) Failure of a switch; (c) Failure of several relay cabinets; (d) Failure of several switches; (e) Disruption of a level crossing.

more realistic in practice: dedicated repairs are expensive, so in general repair units (personnel, spare parts, etc) are allocated to groups of similar components.

Note that we consider an average repair time of 2 hours per component. Further, the FMTs were applicable for the aggregation of several BEs as presented in Section 6.5 on page 158. Thus, the BEs assigned to a RU with the same behaviour were translated into the aggregated I/O-MA model.

The results of our computations are listed in Table 7.7. Strategy 1 refers to the model where all BEs have a dedicated repair and strategy 2 where a repair is assigned to a group of BEs. Further, the unavailability numbers are fractions, e.g. for the failure of a switch the system will be unavailable in 0.01926% of the time in the long run. By observing the unavailability column of strategy 1 and 2, it stands out that the results only differ for 3 FTs. Further the variance is rather low. However, this shows that even with less possible repairs, the availability of the system is only affected marginally. In total, under both strategies, the system is available during 99.98% of the time in the long run. Thus, for the considered rail trajectory even *a more relaxed maintenance strategy in terms of component repairs suffice to guarantee the same availability of the system.*

7.3.3 Effects of preventive maintenance

A key question in maintenance is to investigate how inspections can have an impact on the analysis. Therefore, we rerun the experiments on the failure of the relay cabinet as well as on the failure of a switch by including inspections. We have run the experiments on a single core of a 2 GHz Intel Xeon with 22GB RAM running on Linux with the FMT extension of DFTCALC and used MRMC as the analysis back-end.

In the following we investigate several maintenance strategies w.r.t. corrective as well as preventive maintenance in separation. To investigate the effect of corrective and preventive maintenance, each group of components is assigned to either a RU or a IM, following the following strategies:

- (1) without maintenance;
- (2) corrective maintenance with repair times of one, two and seven days;

Figure 7.17: System reliability over time for different inspection intervals.

- (3) preventive maintenance with inspection frequencies of once, twice, and four times a year.

Note that each BE was assigned an Erlang(2, λ) distribution where the failure rate λ varies for each group of BEs. Further, the inspection threshold is sent out after the first phase. Moreover, the repair strategy of the RU is a FCFS strategy. In total we analysed 14 FMTs based on the relay cabinet FT and the switch FT from Section 7.3.1.1.

We calculated the system reliability for a mission time of 10 years for each FMT and depict the results in Figure 7.17 with a logarithmic scale on the y-axis. As basis for the evaluation, we also calculated the reliability without maintenance. As expected, the probability to fail over 10 years is the highest without any maintenance for both cases. In total, Figures 7.17a and 7.17b show that increasing the inspection frequency as well as lowering the repair times have a positive effect on the reliability. Moreover, one can observe that with increasing life-time of the system, the failure probability increases more when only conducting corrective maintenance compared to preventive maintenance.

Example 7.6. Consider the corrective maintenance strategy where the repair is conducted within a week and the preventive maintenance strategy, where the components are inspected twice a year and renewed if their threshold is violated. While the failure probability is lower for the corrective maintenance strategy in the beginning, the preventive maintenance strategies failure probability growth is slower. This can be seen in Figures 7.17a and 7.17b. This change in the reliability over time is connected to the degradation of components. With the initial deployment of the system the components are not yet degraded and the probability to fail is very low. Moreover, an inspection probably does not trigger

a maintenance action based on the current degradation phase. However, if a component fails the repair is conducted very fast. Over time, more components degrade, and an inspection will trigger the maintenance of several components improving their degradation state. This will slow down the probability that the system fails. However, when only focusing on repairs, more and more components fail over time that have to be repaired. Thus, due to the higher amount of component failures, the probability of a system failure increases more rapidly compared to the inspection approach.

For the relay cabinets in Figure 7.17a the preventive strategy with two inspections a year surpasses the corrective strategy with a repair time of one week after 6 years, while for the switch in Figure 7.17b after 9 years. Similar results hold for the strategy with 4 inspections a year against a repair in one or two days. Those results show how corrective and preventive maintenance affect the reliability of a system over time. Moreover, one can deduct that additional inspections will improve the reliability over the long-run for both systems more than just decreasing the repair time.

7.4 Scalability of FMTs

In the previous section we have analysed FMTs with respect to corrective and preventive maintenance strategies. There we have seen how comparing strategies by analysing FMTs provides helpful insights w.r.t. their impact on reliability as well as availability. In the following we provide more information on the scalability of the analysis of FMTs with DFTCALC.

7.4.1 Experimental set up

The first step in the analysis of an FMT is the generation of its corresponding I/O-MA. Only if the I/O-MA can be generated successfully, an analysis of the FMT is possible. In the following we test the scalability with the FMT from Figure 7.17a. In particular we focus on two factors in the generation of the I/O-MA for FMTs:

- (1) The number of BEs that are subject to inspection on the state space generation;
- (2) The number of phases in the failure distribution of the BEs as well as the inspection distribution of the IMs.

Note that the FMT has an IM for each group of BEs, i.e. 6 IMs in total. For (1) each IM has an inspection phase of $k = 1$, whereas for (2) we have IMs with $k = 1, 2, 3$, and 4. Moreover, each BE has a $\text{Erlang}(k, \lambda)$ failure distribution, where λ varies for each group of BEs and $k = 2$ for (1) and $k = 2, 3$ and 4 for (2). Besides, the inspection threshold is send out after the first phase for all BEs.

BEs per group	# States	# Max. States	# Transitions	# Max. Transitions	memory (kb)	time (min)
4	89,854	131,484	476,107	937,084	2,109.5	8.15
6	418,374	1548,474	2,461,195	13,554,146	32,692.3	79.25
8	1,346,222	4,837,840	8,427,243	73,635,168	144,443	114.25
10	-	-	-	-	-	168.34

Table 7.8: Effects of different numbers of BEs on the analysis.

BE/IM phases	# States	# Max. States	# Transitions	# Max. Transitions	memory (kb)	time (min)
2/1	89,854	131,484	476,107	937,084	2,109.5	8.15
2/2	2,747,801	3,225,763	15,710,001	29,936,003	72,802.5	39.44
2/3	20,509,024	23,240,199	118,583,647	225,324,831	560,943.5	357
2/4	-	-	-	-	-	71.19
3/1	9,979,566	10,525,427	69,669,467	133,437,163	313,962.8	647.61
4/1	-	-	-	-	-	398.8

Table 7.9: Effects of different phases for BEs and IMs.

7.4.2 Scalability results

We have run the experiment on a 2.5 GHz Intel i5 with 16GB RAM running on OSX with the FMT extension of DFTCALC. Note that our main focus was the generation of the I/O-MA and not the actual analysis. Table 7.8 depicts the result for different number of BEs for each group. The number of total elements of the respective FMTs are: 38, 50, 62 and 74. Table 7.9 depicts the results for different number of phases for the FMT with 4 BEs in each group. The graphs in Figure 7.18 visualise the results from the tables while using a log-scale on both axes.

Figure 7.18a visualises the effect of increasing the number of BEs in the FMT. Note that for the FMT with 10 BEs per group the generation process of the I/O-MA was killed after ca. 3 hours by CADP. The graphs show that the state space increases linear with the number of BEs. Besides, the number of maximal states and transitions during the generation increases faster than the number of states and transitions of the final I/O-MA. This has the effect that also the memory consumption during the generation increases with the rate of the maximal number of states and transitions. However, the difference in the maximal states and transitions to the final states and transitions also increases. This approach keeps the growth of size of the final I/O-MA at a lower rate.

Figure 7.18a visualises the effect of increasing the number of phases within the IMs of the FMT. The graphs also show a linear increase with the number of phases. However, compared to adding additional BEs, the growth of the number of states and transitions for the maximal and final case is higher. Hence, adding one additional phase per IM has a bigger effect than adding additional BEs to the FMT and therefore to the IM. Moreover, the difference between the maximal number of states and transitions during the generation and the final size of the I/O-MA is also much lower. In particular the reduction in the number of states

Figure 7.18: Effects of inspections on the analysis.

is marginal. The same effect results from increasing the number of phases of the failure distribution of the BEs. Note that for the FMT where each BE has an Erlang(4, λ) distribution with varying λ the generation process of the I/O-MA was killed after ca. 6 hours by CADP.

7.5 Conclusion

In this chapter we have presented two prototypical tool chains for the evaluation of FMTs, DFTCALC and MAPA for FMTs. For DFTCALC we presented a web based interface as well as provided benchmarks with respect to traditional DFTs. The introduction of the MAPA tool chain for FMTs showed how the I/O-MA semantics can be used with an alternative specification language and generation method.

For evaluating FMTs within an industrial context we conducted an experiment based on a RAMS analysis for a railway trajectory in the Netherlands, provided by Movares. We specified various FMTs with different corrective as well as preventive maintenance strategies and analysed them w.r.t. the systems unavailability and reliability. We can conclude that the use of FMTs shows a direct effect of different maintenance strategies within the FTA, which can help to optimise maintenance strategies. However, we also showed current limitations for the scalability of our analysis w.r.t. the size of the FMT as well as more complex failure and inspection distributions.

Conclusion

*If you find that you're spending almost all your time on theory,
start turning some attention to practical things; it will improve your
theories.*

DONALD E. KNUTH

THE project that led to this thesis was “ArRangeer”, which stands for “smARt RAIlroad maintenance eNGinEERING with stochastic model checking”. The Dutch railway infrastructure manager ProRail has strong ambitions in taking a step towards an intelligent railway maintenance. With respect to that goal, the ArRangeer project set out to investigate methods for advanced maintenance planning and engineering with the help of dynamic fault trees and stochastic model checking. In particular, we were interested in how to combine maintenance with dynamic fault trees and how this affects the analysis. Besides, we investigated how to adapt the analysis model, such that it is able to handle more complex structures as well as can be used to investigate costs.

8.1 Summary

Lets recall our research objective from Chapter 1:

Develop a framework that allows to analyse system failures under different maintenance strategies. Further develop a general model that can be used to analyse timed and reward based properties.

In the first part of this thesis we focused on the second part of the objective. Therefore, we introduced Markov reward automata, an extension to Markov automata with state and transition rewards. Moreover, we provided algorithms to analyse expected reward reachability objectives as well as long-run reward objectives. Besides, we discussed the problem of model conformance between a real world system and its abstracted model representation. In particular, we provided the following results:

- We introduced transition and state rewards to Markov automata and defined Markov reward automata (MRA). In particular the definition of transition rewards in MRAs is kept very generic. That is, a reward can be assigned to each transition and not only to an action or probability distribution. Besides the introduction of MRAs, we adapted the parallel composition to rewards as well as strong and naive weak bisimulation. Moreover, we proposed a version of the weak bisimulation over distributions including reward relations.
- For the analysis of MRAs we presented algorithms for expected reachability objectives. In particular we focused on the expected reward for goal and time bounded reachability as well as the long-run average reward. We showed how to reduce parts of the analysis to a stochastic shortest path problem as well as linear programming problems. Moreover we proved that these reductions are correct. Further, we discussed how to adapt CSRL model checking to include MRAs and the expected reward and long-run reward metrics. Finally, we provided experiments presenting the scalability of the algorithms. Thus with MRAs a wide variety of systems that feature non-determinism, discrete probabilistic choices, continuous stochastic timing and transition-based and state-based rewards can now be analysed.

In the second part of this thesis we focused on the first part of the objective, i.e. on providing a framework capable of analysing system failures under different maintenance strategies. We introduced fault maintenance trees (FMTs), a maintenance extension to dynamic fault trees. They allow to incorporate preventive as well as corrective maintenance strategies into the FTA. Moreover we provided a prototypical tool to perform FTA on FMTs.

- We introduced FMTs, a model consisting of a DFT, inspection modules (IMs) and repair units (RUs). This allows to specify a DFT enriched with corrective and preventive maintenance strategies. The advantage is that the maintenance strategies are directly defined on the level of the fault tree. Moreover, their effect is directly translated into the analysis. Further, we defined a semantics for FMTs using I/O-MAs such that each FMT element is described by its I/O-MA. The semantics is a natural extension to the I/O-MA semantics for DFTs. Therefore also the compositional aggregation is applicable to the FMT semantics. Besides, we provided smarter I/O-MAs that exploit similar BEs in FMTs by using counting abstraction.
- For evaluating FMTs in practice, we conducted a RAMS case study for a railway trajectory in the Netherlands provided by Movares. We used the MAMA tool chain for analysing different corrective maintenance strategies w.r.t. the unavailability, and the DFTCALC tool chain for comparing the effects of preventive and corrective maintenance w.r.t. the reliability. The case study has shown that with the use of FMTs the effect of different maintenance strategies is directly reflected within the FTA. However, we have also shown the current limitations w.r.t. the analysis of FMTs. For the industrial use of FMTs further optimisation's are needed. While we are able to analyse medium size FMTs, increasing the number of BEs assigned

to IMs as well as increasing the failure and inspection phases limits the generation of the I/O-MA and therefore the analysis. Henceforth, more aggressive reductions is needed to keep the state space smaller.

8.2 Discussion and Future work

In addition to the results presented in this thesis, we performed more in depth case studies on FMTs in cooperation with industry partners. Together with ProRail we used FMTs to investigate the electrically insulated railway joints in the Netherlands [RGvS16]. Further, we used FMTs together with NedTrain to evaluate maintenance strategies for a train pneumatic compressor [RGDPS16; RGDS16]. However, while the modelling of both systems is based on FMTs, the analysis approach we used is different from the one presented in this thesis. We used simulation based techniques, in particular statistical model checking (SMC) with Uppaal [Beh+06]. This enabled us to obtain analysis results on models that did not scale well w.r.t. the methods provided in this thesis. The case studies showed that FMTs are a useful framework to investigate maintenance optimisation problems from an industrial point of view. In particular, FMTs proved to be a convenient model for the cases of our industry partners, such that they provide sufficient expressive power to capture complex maintenance aspects, and are able to produce predictive analysis results. However, it is also of interest to be able to have a numerical solution instead of only be able to perform simulations. Hence, we would like to have an integrated framework supporting the analysis of FMTs with MRAs as well as with the statistical model checking approach.

The results presented in this thesis open up a variety of potential research topics and applications. In the following we provide a discussion about the challenges that arise with the modelling and analysis of FMTs as well as possible research directions and advancements.

Modelling. As shown in Section 6.4 on page 150, the inclusion of corrective maintenance can have the effect that the structure of the tree has to be changed. Hence, if a DFT is used as a basis for a FMT, the structure has to be reevaluated after specifying the RUs. Moreover, the specification of the phases and inspection threshold for the BEs connected to an IM involves additional effort. As described in Section 6.2 on page 138, the degradation phases can be specified by fitting historical data sets to a (hyper) Erlang distribution. Otherwise, for components with no available data, the failure distributions has to be estimated by experts. Moreover, the threshold of the component has to be defined. Hence, one has to define the phase after which an inspection should be performed for each component. However, the threshold is bound to the phases described by the failure distribution. For example, if the best fit for the failure of a component is an exponential distribution, there exists no degraded state besides the failed state. Hence no threshold can be defined for such a component, and therefore an inspection could not have an impact. On the other hand, if the failure distribution is a Erlang(k, λ) distribution with $k \geq 2$, the phase with

the best fit has to be chosen for the threshold. Hence, specifying FMTs not only involves a careful consideration of the impact of failures w.r.t. the possible repairs, but also the evaluation of when and how components can and should be inspected w.r.t. their respective failure distribution.

Analysis. The approach presented in this thesis could not fully cope with the case studies by ProRail and NedTrain. The main reason for that lies in the mild scalability w.r.t. generation of the I/O-MA for more complex FMTs. There are two main starting points we would suggest to tackle this problem. The first one is on the level of the FMT itself. In [JGKRS15] we have shown how the rewriting of DFTs via graph-transformation has a positive effect on the generation within DFTCALC. However, the results can not directly be applied to FMTs, since repairs change the structure of the fault tree and therefore also the underlying graph. Nevertheless, we think that this concept can also be applied to FMTs with the effect of reducing the model and therefore improving the generation process. Furthermore, additional reduction on the level of the I/O-MA are needed to keep the state space small. Where the compositional aggregation helps in reducing the state space during composition, the intermediate state space can nevertheless grow very large due to the individual I/O-MAs. This comes from the additional communication of the inspection and repairs as well as the increase of Markovian transitions based on the inspection and degradation phases. Thus, more concepts like the aggregation of BEs as shown in Section 6.5 have to be included prior to the generation. Furthermore, we think that a combination of the techniques by Volk, Junges, and Katoen [VJK16] combined with parallel composition can drastically improve the generation.

Advancements. A desired advancement for FMTs and their analysis is to be able to extract the optimal maintenance strategies w.r.t. certain cost and reliability bounds. Moreover, a popular strategy in asset management is reliability centered maintenance (RCM), which focuses on reducing costs while keeping the level of risks low [Mou97]. Thus the incorporation of costs into the FMT framework would make them also attractive for the assessment of RCM. In the statistical model checking approach [RGDS16] we already included costs for the top level failure as well as repair and inspection actions. This proved to be valuable in analysing maintenance strategies with the focus of optimising the costs while adhering to certain reliability bounds. However, only given strategies can be analysed. With the introduction of MRAs, the analysis w.r.t. costs is also possible in the I/O-MA framework for FMTs. For example, one can assign transition rewards to transitions labelled with an inspection or repair action, as well as assign state rewards to states that represent a top level failure. This would allow one to obtain results w.r.t. the costs of inspection and repair actions, as well as the average cost of a system failure. However, to be able to analyse FMTs using MRAs and their algorithms, optimisation's are needed. For example, as shown in [Bra+15] abstraction improves the runtime of the algorithms for MRAs. In addition, the efficient generation, composition and minimisation of MRAs has to be studied.

Outlook. In conclusion, we think that FMTs as well as MRAs provide a powerful modelling and analysis framework with further potential. For a start, MRAs as well as FMTs provide interesting challenges for research. For example, finding efficient ways to analyse multi objective properties on MRAs as well as including parameter synthesis. In addition, optimising the generation of MRAs from FMTs as well as defining coarser equivalence relations on MRAs that still adhere to our expected reward metrics. Moreover, the results obtained with our current framework already show that FMTs provide extra value to the analysis of reliable systems. Further, with a more long-term perspective we see the potential that FMTs are integrated in the design and optimisation process of reliable systems.

List of publications by the author

Conferences

- [1] F. Arnold, A. F. E. Belinfante, F. I. Van der Berg, D. Guck, and M. I. A. Stoelinga. “DFTCalc: a tool for efficient fault tree analysis”. In: *Proceedings of the 32nd International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2013)*. Vol. 8153. Lecture Notes in Computer Science. Toulouse, France: Springer Verlag, 2013, pp. 293–301. URL: http://dx.doi.org/10.1007/978-3-642-40793-2_27.
- [2] D. Giannakopoulou, D. Guck, and J. Schumann. “Exploring Model Quality for ACAS X”. In: *Proceedings of the 21st International Symposium on Formal Methods (FM 2016)*. Ed. by J. Fitzgerald, C. Heitmeyer, S. Gnesi, and A. Philippou. Vol. 9995. Lecture Notes in Computer Science. Limassol, Cyprus: Springer International Publishing, 2016. ISBN: 978-3-319-48989-6. DOI: 10.1007/978-3-319-48989-6_17. URL: http://dx.doi.org/10.1007/978-3-319-48989-6_17.
- [3] D. Guck, H. Hatefi, H. Hermanns, J.-P. Katoen, and M. Timmer. “Modelling, Reduction and Analysis of Markov Automata”. In: *Proceedings of the 10th International Conference on Quantitative Evaluation of Systems (QEST 2013)*. Ed. by K. R. Joshi, M. Siegle, M. I. A. Stoelinga, and P. R. D’Argenio. Vol. 8054. Lecture Notes in Computer Science. Buenos Aires, Argentina: Springer Verlag, 2013, pp. 55–71. URL: http://dx.doi.org/10.1007/978-3-642-40196-1_5.
- [4] D. Guck, T. Han, J.-P. Katoen, and M. Neuhäüßer. “Quantitative timed analysis of interactive Markov chains”. In: *Proceedings of the 4th International Symposium on NASA Formal Methods (NFM 2012)*. Vol. 7226. Lecture Notes in Computer Science. Norfolk, VA, USA: Springer Verlag, 2012, pp. 8–23. URL: http://dx.doi.org/10.1007/978-3-642-28891-3_4.

-
- [5] D. Guck, J.-P. Katoen, M. I. A. Stoelinga, T. Luiten, and J. Romijn. “Smart railroad maintenance engineering with stochastic model checking”. In: *Proceedings of the 2nd International Conference on Railway Technology: Research, Development and Maintenance (Railways 2014)*. Ed. by J. Pombo. Vol. 104. Civil-Comp Proceedings. Ajaccio, Corsica, France: Civil-Comp Press, 2014, p. 299. URL: <http://dx.doi.org/10.4203/ccp.104.299>.
- [6] D. Guck, J. J. Spel, and M. I. A. Stoelinga. “DFTCalc: Reliability centered maintenance via fault tree analysis (tool paper)”. In: *Proceedings of the 17th International Conference on Formal Engineering Methods (ICFEM 2015)*. Vol. 9407. Lecture Notes in Computer Science. Paris, France: Springer Verlag, 2015, pp. 304–311. URL: http://dx.doi.org/10.1007/978-3-319-25423-4_19.
- [7] D. Guck, M. Timmer, H. Hatefi, E. J. J. Ruijters, and M. I. A. Stoelinga. “Modelling and analysis of Markov reward automata”. In: *Proceedings of the 12th International Symposium on Automated Technology for Verification and Analysis (ATVA 2014), Sydney, NSW, Australia*. Vol. 8837. Lecture Notes in Computer Science. Sydney, NSW, Australia: Springer Verlag, 2014, pp. 168–184. URL: http://dx.doi.org/10.1007/978-3-319-11936-6_13.
- [8] S. Junges, D. Guck, J.-P. Katoen, A. Rensink, and M. I. A. Stoelinga. “Fault Trees on a Diet - Automated Reduction by Graph Rewriting”. In: *Proceedings of the 1st International Symposium on Dependable Software Engineering: Theories, Tools, and Applications (SETTA 2015)*. Ed. by X. Li, Z. Liu, and W. Yi. Vol. 9409. Lecture Notes in Computer Science. Nanjing, China: Springer Verlag, 2015, pp. 3–18. URL: http://dx.doi.org/10.1007/978-3-319-25942-0_1.
- [9] S. Junges, D. Guck, J.-P. Katoen, and M. I. A. Stoelinga. “Uncovering Dynamic Fault Trees”. In: *Proceedings of the 46th annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2016)*. Toulouse, France: IEEE Computer Society, 2016, pp. 299–310. DOI: 10.1109/DSN.2016.35.
- [10] E. J. Ruijters, D. Guck, P. Drolenga, M. Peters, and M. I. A. Stoelinga. “Maintenance Analysis and Optimization via Statistical Model Checking”. In: *Proceedings of the 13th International Conference on Quantitative Evaluation of Systems (QEST 2016)*. Ed. by G. Agha and B. Van Houdt. Quebec City, QC, Canada: Springer International Publishing, 2016, pp. 331–347. ISBN: 978-3-319-43425-4. DOI: 10.

- 1007/978-3-319-43425-4_22. URL: http://dx.doi.org/10.1007/978-3-319-43425-4_22.
- [11] E. J. Ruijters, D. Guck, P. Drolenga, and M. I. A. Stoelinga. “Fault maintenance trees: reliability centered maintenance via statistical model checking”. In: *Proceedings of the Annual Reliability and Maintainability Symposium (RAMS 2016)*. IEEE, 2016, pp. 1–6. DOI: 10.1109/RAMS.2016.7447986.
- [12] E. J. J. Ruijters, D. Guck, M. van Noort, and M. I. A. Stoelinga. “Reliability-centered maintenance of the Electrically Insulated Railway Joint via Fault Tree Analysis: A practical experience report”. In: *Proceedings of the 46th annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2016)*. Toulouse, France: IEEE Computer Society, 2016, pp. 662–669. DOI: 10.1109/DSN.2016.67.

Journals

- [13] D. Guck, H. Hatefi, H. Hermanns, J.-P. Katoen, and M. Timmer. “Analysis of timed and long-run objectives for Markov automata”. In: *Logical Methods in Computer Science* 3 (2014), p. 17. ISSN: 1860-5974. URL: [http://dx.doi.org/10.2168/LMCS-10\(3:17\)2014](http://dx.doi.org/10.2168/LMCS-10(3:17)2014).
- [14] S. Junges, D. Guck, J.-P. Katoen, A. Rensink, and M. I. A. Stoelinga. “Fault Trees on a Diet - Automated Reduction by Graph Rewriting”. In: *Formal Aspects of Computing*. accepted for publication. 2016.

Workshops

- [15] F. Arnold, D. Guck, R. Kumar, and M. I. A. Stoelinga. “Sequential and Parallel Attack Tree Modelling”. In: *Proceedings of the SAFECOMP 2015 Workshops, ASSURE, DEC-SoS, ISSE, ReSA4CI, and SASSUR*. Ed. by F. Koornneef and C. Van Gulijk. Vol. 9338. Lecture Notes in Computer Science. Delft, The Netherlands: Springer Verlag, 2015, pp. 291–299. ISBN: 3-319-24249-0. URL: http://dx.doi.org/10.1007/978-3-319-24249-1_25.
- [16] D. Guck, M. Timmer, and S. C. C. Blom. “Extending Markov Automata with State and Action Rewards”. In: *Proceedings of the 12th Workshop on Quantitative Aspects of Programming Languages and Systems (QAPL 2014)*. Ed. by N. Bertrand and L. Bortolussi. Grenoble, France: Inria, 2014. URL: <http://eprints.eemcs.utwente.nl/24693/>.

- [17] R. Kumar, D. Guck, and M. I. A. Stoelinga. “Time dependent analysis with dynamic counter measure trees”. In: *Proceedings of the 13th Workshop on Quantitative Aspects of Programming Languages and Systems (QAPL 2015)*. London, England: Inria, 2015, pp. 1–5. URL: <http://arxiv.org/pdf/1510.00050.pdf>.

Technical reports

- [18] F. Arnold, A. F. E. Belinfante, F. I. Van der Berg, D. Guck, and M. I. A. Stoelinga. *DFTCalc: a tool for efficient fault tree analysis (extended version)*. Technical Report TR-CTIT-13-13. Enschede: Centre for Telematics and Information Technology, University of Twente, 2013. URL: <http://eprints.eemcs.utwente.nl/23404/>.
- [19] D. Guck, H. Hatefi, H. Hermanns, J.-P. Katoen, and M. Timmer. *Modelling, Reduction and Analysis of Markov Automata (extended version)*. Technical Report arXiv:1305.7050. Ithaca, NY, USA: Cornell University, 2013. URL: <http://arxiv.org/abs/1305.7050v1>.
- [20] D. Guck, M. Timmer, H. Hatefi, E. J. J. Ruijters, and M. I. A. Stoelinga. *Modelling and analysis of Markov reward automata (extended version)*. Technical Report TR-CTIT-14-06. Enschede: Centre for Telematics and Information Technology, University of Twente, 2014. URL: <http://eprints.eemcs.utwente.nl/24800/>.

Survey

- [21] F. Arnold, D. Gebler, D. Guck, and H. Hatefi. “A tutorial on interactive Markov chains”. In: *Stochastic Model Checking. Rigorous Dependability Analysis Using Model Checking Techniques for Stochastic Systems*. Ed. by A. K. I. Remke and M. I. A. Stoelinga. Vol. 8453. Lecture Notes in Computer Science. Vahrn, Italy: Springer Verlag, 2014, pp. 26–66. URL: http://dx.doi.org/10.1007/978-3-662-45489-3_2.

References

- [ABVGS13b] F. Arnold, A. F. E. Belinfante, F. I. Van der Berg, D. Guck, and M. I. A. Stoelinga. “DFTCalc: a tool for efficient fault tree analysis”. In: *Proceedings of the 32nd International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2013)*. Vol. 8153. Lecture Notes in Computer Science. Toulouse, France: Springer Verlag, 2013, pp. 293–301. URL: http://dx.doi.org/10.1007/978-3-642-40793-2_27.
- [ADD00] R. Ash and C. Doléans-Dade. *Probability & Measure Theory*. Academic Press, 2000. ISBN: 9780120652020.
- [AHK03] S. Andova, H. Hermanns, and J.-P. Katoen. “Discrete-Time Rewards Model-Checked”. In: *Proceedings of the 1st International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2003)*. Ed. by K. G. Larsen and P. Niebert. Vol. 2791. Lecture Notes in Computer Science. Marseille, France: Springer Berlin Heidelberg, 2003, pp. 88–104. ISBN: 978-3-540-40903-8. DOI: 10.1007/978-3-540-40903-8_8. URL: http://dx.doi.org/10.1007/978-3-540-40903-8_8.
- [ALR+01] A. Avizienis, J.-C. Laprie, B. Randell, et al. *Fundamental concepts of dependability*. University of Newcastle upon Tyne, Computing Science, 2001.
- [ALRL04] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. “Basic concepts and taxonomy of dependable and secure computing”. In: *IEEE Transactions on Dependable and Secure Computing* 1.1 (2004), pp. 11–33. ISSN: 1545-5971. DOI: 10.1109/TDSC.2004.2.
- [Alf97] L. de Alfaro. “Formal Verification of Probabilistic Systems”. PhD thesis. Stanford University, 1997.
- [Alf99] L. Alfaro. “Proceedings of the 10th International Conference on Concurrency Theory (CONCUR 1999)”. In: ed. by J. C. M. Baeten and S. Mauw. Eindhoven, The Netherlands: Springer Berlin Heidelberg, 1999. Chap. Computing Minimum and Maximum Reachability Times in Probabilistic Systems, pp. 66–81. ISBN: 978-3-540-48320-5. DOI:

- 10.1007/3-540-48320-9_7. URL: http://dx.doi.org/10.1007/3-540-48320-9_7.
- [Alu15] R. Alur. *Principles of cyber-physical systems*. MIT Press, 2015.
- [And98] G. E. Andrews. *The theory of partitions*. 2. Cambridge university press, 1998.
- [BA95] A. Bianco and L. de Alfaro. “Model Checking of Probabilistic and Nondeterministic Systems”. In: *Proceedings of the 15th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 1995)*. Vol. 1026. Lecture Notes in Computer Science. Bangalore, India: Springer Berlin Heidelberg, 1995, pp. 499–513. ISBN: 978-3-540-49263-4. DOI: 10.1007/3-540-60692-0_70. URL: http://dx.doi.org/10.1007/3-540-60692-0_70.
- [BB03] M. Bouissou and J.-L. Bon. “A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes”. In: *Reliability Engineering & System Safety* 82.2 (2003), pp. 149–163.
- [BCHHK10] C. Baier, L. Cloth, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. “Performability assessment by model checking of Markov reward models”. In: *Formal Methods in System Design* 36.1 (2010), pp. 1–36. ISSN: 1572-8102. DOI: 10.1007/s10703-009-0088-7. URL: <http://dx.doi.org/10.1007/s10703-009-0088-7>.
- [BCR04] A. Bobbio and D. Codetta-Raiteri. “Parametric fault trees with dynamic gates and repair boxes”. In: *Annual Symposium Reliability and Maintainability (RAMS 2004)*. IEEE, 2004, pp. 459–465. DOI: 10.1109/RAMS.2004.1285491.
- [BCS07] H. Boudali, P. Crouzen, and M. I. A. Stoelinga. “CORAL-a tool for compositional reliability and availability analysis”. In: *ARTIST workshop. Presented at the 19th international conference on Computer Aided Verification*. 2007.
- [BCS10] H. Boudali, P. Crouzen, and M. Stoelinga. “A Rigorous, Compositional, and Extensible Framework for Dynamic Fault Tree Analysis”. In: *IEEE Transactions on Dependable and Secure Computing* 7 (2 2010), pp. 128–143.
- [BD05] H. Boudali and J. B. Dugan. “A discrete-time Bayesian network reliability modeling and analysis framework”. In: *Reliability Engineering & System Safety* 87.3 (2005), pp. 337–349.
- [BD06] H. Boudali and J. B. Dugan. “A continuous-time Bayesian network reliability modeling and analysis framework”. In: *IEEE Transactions on Reliability* 55.1 (2006), pp. 86–97.

- [BHHK00] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. “On the Logical Characterisation of Performability Properties”. In: *Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP 2000)*. Ed. by U. Montanari, J. D. P. Rolim, and E. Welzl. Vol. 1853. Lecture Notes in Computer Science. Geneva, Switzerland: Springer Berlin Heidelberg, 2000, pp. 780–792. ISBN: 978-3-540-45022-1. DOI: 10.1007/3-540-45022-X_65. URL: http://dx.doi.org/10.1007/3-540-45022-X_65.
- [BHHK03] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. “Model-Checking Algorithms for Continuous-Time Markov Chains”. In: *IEEE Transactions on Software Engineering* 29.6 (2003), pp. 524–541.
- [BHHK08] C. Baier, B. R. Haverkort, H. Hermanns, and J. Katoen. “Reachability in continuous-time Markov reward decision processes”. In: *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*. 2008, pp. 53–72.
- [BHHZ11] P. Buchholz, E. M. Hahn, H. Hermanns, and L. Zhang. “Model checking algorithms for CTMDPs”. In: *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV 2011)*. Ed. by G. Gopalakrishnan and S. Qadeer. Vol. 6806. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 225–242. ISBN: 978-3-642-22110-1. DOI: 10.1007/978-3-642-22110-1_19. URL: http://dx.doi.org/10.1007/978-3-642-22110-1_19.
- [BP75] R. E. Barlow and F. Proschan. *Statistical theory of reliability and life testing: probability models*. Tech. rep. DTIC Document, 1975.
- [BR13] A. Belinfante and A. Rensink. *Publishing your prototype tool on the web: PUPTOL, a framework*. 2013.
- [BS11] P. Buchholz and I. Schulz. “Numerical analysis of continuous time Markov decision processes over finite horizons”. In: *Computers & Operations Research* 38.3 (2011), pp. 651–659. ISSN: 0305-0548. DOI: <http://dx.doi.org/10.1016/j.cor.2010.08.011>. URL: <http://www.sciencedirect.com/science/article/pii/S0305054810001759>.
- [BT91] D. P. Bertsekas and J. N. Tsitsiklis. “An analysis of stochastic shortest path problems”. In: *Mathematics of Operations Research* 16.3 (1991), pp. 580–595.
- [Bam12] R. Bamberg. “Non-deterministic generalised stochastic Petri nets modelling and analysis”. MA thesis. University of Twente, 2012.

- [Beh+06] G. Behrmann et al. “UPPAAL 4.0”. In: *Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems (QEST 2006)*. IEEE, 2006, pp. 125–126. DOI: 10.1109/QEST.2006.59.
- [Bel14] A. F. E. Belinfante. “JTorX: exploring model-based testing”. IPA Dissertation series no. 2014-09. PhD thesis. Enschede, 2014. URL: <http://doc.utwente.nl/91781/>.
- [Ber97] M. Bernardo. “An Algebra-Based Method to Associate Rewards with EMPA Terms”. In: *Proceedings of the 24th International Colloquium on Automata, Languages and Programming, (ICALP 1997)*. Ed. by P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela. Vol. 1256. Lecture Notes in Computer Science. Bologna, Italy: Springer Berlin Heidelberg, 1997, pp. 358–368. ISBN: 978-3-540-69194-5. DOI: 10.1007/3-540-63165-8_192. URL: http://dx.doi.org/10.1007/3-540-63165-8_192.
- [Boz+09] M. Bozzano et al. “The COMPASS Approach: Correctness, Modelling and Performability of Aerospace Systems”. In: *Proceedings of the 28th International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2009)*. Ed. by B. Buth, G. Rabe, and T. Seyfarth. Vol. 5775. Lecture Notes in Computer Science. Hamburg, Germany: Springer Berlin Heidelberg, 2009, pp. 173–186. ISBN: 978-3-642-04468-7. DOI: 10.1007/978-3-642-04468-7_15. URL: http://dx.doi.org/10.1007/978-3-642-04468-7_15.
- [Boz+11] M. Bozzano et al. “Safety, Dependability and Performance Analysis of Extended AADL Models”. In: *The Computer Journal* 54 (2011), pp. 754–775. DOI: 10.1093/comjnl/bxq024.
- [Bra+15] B. Braitling et al. “Abstraction-Based Computation of Reward Measures for Markov Automata”. In: *Proceedings of the 16th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2015)*. Ed. by D. D’Souza, A. Lal, and K. G. Larsen. Vol. 8931. Lecture Notes in Computer Science. Mumbai, India: Springer Berlin Heidelberg, 2015, pp. 172–189. ISBN: 978-3-662-46081-8. DOI: 10.1007/978-3-662-46081-8_10. URL: http://dx.doi.org/10.1007/978-3-662-46081-8_10.
- [Buc00] K. Buchacker. “Modeling with extended fault trees”. In: *Proceedings of the 5th IEEE International Symposium on High Assurance Systems Engineering (HASE 2000)*. IEEE, 2000, pp. 238–246.

- [CE82] E. M. Clarke and E. A. Emerson. “Design and synthesis of synchronization skeletons using branching time temporal logic”. In: *Logics of Programs: Workshop, Yorktown Heights, New York*. Ed. by D. Kozen. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, pp. 52–71. ISBN: 978-3-540-39047-3. DOI: 10.1007/BFb0025774. URL: <http://dx.doi.org/10.1007/BFb0025774>.
- [CES09] E. M. Clarke, E. A. Emerson, and J. Sifakis. “Model checking: algorithmic verification and debugging”. In: *Communications of the ACM* 52.11 (2009), pp. 74–84.
- [CH10] P. Crouzen and H. Hermanns. “Aggregation Ordering for Massively Compositional Models”. In: *Proceedings of the 10th International Conference on Application of Concurrency to System Design (ACSD 2010)*. IEEE, 2010, pp. 171–180. DOI: 10.1109/ACSD.2010.28.
- [CH11] K. Chatterjee and M. Henzinger. “Faster and Dynamic Algorithms for Maximal End-Component Decomposition and Related Graph Problems in Probabilistic Verification”. In: *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2011)*. San Francisco, CA, USA: Society for Industrial and Applied Mathematics, 2011, pp. 1318–1336.
- [CHLS09] N. Coste, H. Hermanns, E. Lantreibeacq, and W. Serwe. “Towards Performance Prediction of Compositional Models in Industrial GALS Designs”. In: *Proceedings of the 21st International Conference on Computer Aided Verification (CAV 2009)*. Ed. by A. Bouajjani and O. Maler. Vol. 5643. Lecture Notes in Computer Science. Grenoble, France: Springer, 2009, pp. 204–218. ISBN: 978-3-642-02658-4. DOI: 10.1007/978-3-642-02658-4_18. URL: http://dx.doi.org/10.1007/978-3-642-02658-4_18.
- [CL11] P. Crouzen and F. Lang. “Smart reduction”. In: *Proceedings of the 11th International Conference on Fundamental Approaches to Software Engineering (FASE 2011)*. Ed. by D. Giannakopoulou and F. Orejas. Vol. 6603. Lecture Notes in Computer Science. Saarbrücken, Germany: Springer Berlin Heidelberg, 2011, pp. 111–126. ISBN: 978-3-642-19811-3. DOI: 10.1007/978-3-642-19811-3_9. URL: http://dx.doi.org/10.1007/978-3-642-19811-3_9.
- [CR05] D. Codetta-Raiteri. “The Conversion of Dynamic Fault Trees to Stochastic Petri Nets, as a case of Graph Transformation”. In: *Proceedings of the Workshop on Petri Nets and Graph Transformations (PNGT 2004)*. Vol. 127. 2. Elsevier, 2005, pp. 45–60. DOI: <http://dx.doi.org/10.1016/j.entcs.2005.02.005>.

- [CSD00] D. Coppit, K. J. Sullivan, and J. B. Dugan. “Formal semantics of models for computational engineering: A case study on dynamic fault trees”. In: *Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE 2000)*. IEEE, 2000, pp. 270–282.
- [Cha+11] D. Champelovier et al. “Reference manual of the LOTOS NT to LOTOS translator (version 5.4)”. In: *Inria/Vasy* 149 (2011).
- [Cim+02] A. Cimatti et al. “NuSMV 2: An OpenSource Tool for Symbolic Model Checking”. In: *Proceedings of the 14th International Conference on Computer Aided Verification (CAV 2002)*. Ed. by E. Brinksma and K. G. Larsen. Vol. 2404. Lecture Notes in Computer Science. Copenhagen, Denmark: Springer Berlin Heidelberg, 2002, pp. 359–364. ISBN: 978-3-540-45657-5. DOI: 10.1007/3-540-45657-0_29. URL: http://dx.doi.org/10.1007/3-540-45657-0_29.
- [Cla96] G. Clark. “Formalising the Specification of Rewards with PEPA”. In: *PAPM*. 1996, pp. 139–160.
- [Clo06] L. Cloth. “Model checking algorithms for Markov reward models”. PhD thesis. 2006.
- [DBB92] J. B. Dugan, S. J. Bavuso, and M. A. Boyd. “Dynamic fault-tree models for fault-tolerant computer systems”. In: *IEEE Transactions on reliability* 41.3 (1992), pp. 363–377.
- [DG15] M. Dimjasevic and D. Giannakopoulou. “Test-case generation for runtime analysis and vice versa: verification of aircraft separation assurance”. In: *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA 2015)*. Baltimore, MD, USA: ACM, 2015, pp. 282–292. DOI: 10.1145/2771783.2771804. URL: <http://doi.acm.org/10.1145/2771783.2771804>.
- [DVG97] J. B. Dugan, B. Venkataraman, and R. Gulati. “DIFtree: a software package for the analysis of dynamic fault tree models”. In: *Proceedings of the Annual Reliability and Maintainability Symposium (RAMS 1997)*. 1997, pp. 64–70. DOI: 10.1109/RAMS.1997.571666.
- [EG14] C. von Essen and D. Giannakopoulou. “Analyzing the Next Generation Airborne Collision Avoidance System”. In: *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2014)*. Lecture Notes in Computer Science. Grenoble, France: Springer Berlin Heidelberg, 2014, pp. 620–635. DOI: 10.1007/978-3-642-54862-8_54. URL: http://dx.doi.org/10.1007/978-3-642-54862-8_54.

- [EG16] C. von Essen and D. Giannakopoulou. “Probabilistic verification and synthesis of the next generation airborne collision avoidance system”. In: *International Journal on Software Tools for Technology Transfer* 18.2 (2016), pp. 227–243. ISSN: 1433-2787. DOI: 10.1007/s10009-015-0388-8. URL: <http://dx.doi.org/10.1007/s10009-015-0388-8>.
- [EHKZ13] C. Eisentraut, H. Hermanns, J.-P. Katoen, and L. Zhang. “A Semantics for Every GSPN”. In: *Petri Nets*. Vol. 7927. Lecture Notes in Computer Science. Springer, 2013, pp. 90–109. ISBN: 978-3-642-38696-1.
- [EHZ10a] C. Eisentraut, H. Hermanns, and L. Zhang. “On Probabilistic Automata in Continuous Time”. In: *Proceedings of 25th Annual IEEE Symposium on Logic in Computer Science (LICS 2010)*. 2010, pp. 342–351. DOI: 10.1109/LICS.2010.41.
- [EHZ10b] C. Eisentraut, H. Hermanns, and L. Zhang. “Concurrency and Composition in a Stochastic World”. In: *Proceedings of the 21th International Conference on Concurrency Theory (CONCUR 2010)*. Ed. by P. Gastin and F. Laroussinie. Vol. 6269. Lecture Notes in Computer Science. Paris, France: Springer Berlin Heidelberg, 2010, pp. 21–39. ISBN: 978-3-642-15375-4. DOI: 10.1007/978-3-642-15375-4_3. URL: http://dx.doi.org/10.1007/978-3-642-15375-4_3.
- [EI99] C. A. Ericson II. “Fault Tree Analysis—A History”. In: *Proceedings of the 17th International System Safety Conference*. Seattle, Washington, 1999.
- [EWG12] E. Edifor, M. Walker, and N. Gordon. “Quantification of Priority-OR Gates in Temporal Fault Trees”. In: *Proceedings of the 31st International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2012)*. Ed. by F. Ortmeier and P. Daniel. Vol. 7612. Lecture Notes in Computer Science. Magdeburg, Germany: Springer Berlin Heidelberg, 2012, pp. 99–110. ISBN: 978-3-642-33677-5. DOI: 10.1007/978-3-642-33678-2_9. URL: http://dx.doi.org/10.1007/978-3-642-33678-2_9.
- [Far80] F. Farmer. “Editorial”. In: *Reliability Engineering* 1.1 (1980), pp. 1–2. ISSN: 0143-8174. DOI: [http://dx.doi.org/10.1016/0143-8174\(80\)90009-8](http://dx.doi.org/10.1016/0143-8174(80)90009-8). URL: <http://www.sciencedirect.com/science/article/pii/0143817480900098>.
- [Fed00] Federal aviation administration. *System Safety Handbook*. Tech. rep. 2000.
- [Fed15] Federal aviation administration. *FAA aerospace forecast fiscal year 2016-2036*. Tech. rep. 2015.
- [Foo14] Food and Drug Administration. *Medical Device Recall Report FY2003 to FY2012*. Tech. rep. 2014.

- [Fu14] H. Fu. “Maximal Cost-Bounded Reachability Probability on Continuous-Time Markov Decision Processes”. In: *Proceedings of the 17th International Conference on Foundations of Software Science and Computation Structures (FOS-SACS 2014)*. Ed. by A. Muscholl. Grenoble, France: Springer Berlin Heidelberg, 2014, pp. 73–87. ISBN: 978-3-642-54830-7. DOI: 10.1007/978-3-642-54830-7_5. URL: http://dx.doi.org/10.1007/978-3-642-54830-7_5.
- [GBSEH11] D. Giannakopoulou, D. H. Bushnell, J. Schumann, H. Erzberger, and K. Heere. “Formal testing for separation assurance”. In: *Annals of Mathematics and Artificial Intelligence* 63.1 (2011), pp. 5–30. DOI: 10.1007/s10472-011-9224-3. URL: <http://dx.doi.org/10.1007/s10472-011-9224-3>.
- [GGS16] D. Giannakopoulou, D. Guck, and J. Schumann. “Exploring Model Quality for ACAS X”. In: *Proceedings of the 21st International Symposium on Formal Methods (FM 2016)*. Ed. by J. Fitzgerald, C. Heitmeyer, S. Gnesi, and A. Philippou. Vol. 9995. Lecture Notes in Computer Science. Limassol, Cyprus: Springer International Publishing, 2016. ISBN: 978-3-319-48989-6. DOI: 10.1007/978-3-319-48989-6_17. URL: http://dx.doi.org/10.1007/978-3-319-48989-6_17.
- [GHHKT13b] D. Guck, H. Hatefi, H. Hermanns, J.-P. Katoen, and M. Timmer. “Modelling, Reduction and Analysis of Markov Automata”. In: *Proceedings of the 10th International Conference on Quantitative Evaluation of Systems (QEST 2013)*. Ed. by K. R. Joshi, M. Siegle, M. I. A. Stoelinga, and P. R. D’Argenio. Vol. 8054. Lecture Notes in Computer Science. Buenos Aires, Argentina: Springer Verlag, 2013, pp. 55–71. URL: http://dx.doi.org/10.1007/978-3-642-40196-1_5.
- [GHHKT14] D. Guck, H. Hatefi, H. Hermanns, J.-P. Katoen, and M. Timmer. “Analysis of timed and long-run objectives for Markov automata”. In: *Logical Methods in Computer Science* 3 (2014), p. 17. ISSN: 1860-5974. URL: [http://dx.doi.org/10.2168/LMCS-10\(3:17\)2014](http://dx.doi.org/10.2168/LMCS-10(3:17)2014).
- [GKSLR14] D. Guck, J.-P. Katoen, M. I. A. Stoelinga, T. Luiten, and J. Romijn. “Smart railroad maintenance engineering with stochastic model checking”. In: *Proceedings of the 2nd International Conference on Railway Technology: Research, Development and Maintenance (Railways 2014)*. Ed. by J. Pombo. Vol. 104. Civil-Comp Proceedings. Ajaccio, Corsica, France: Civil-Comp Press, 2014, p. 299. URL: <http://dx.doi.org/10.4203/ccp.104.299>.

- [GLMS12] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. “CADP 2011: A toolbox for the construction and analysis of distributed processes”. In: *International Journal on Software Tools for Technology Transfer* (2012), pp. 1–19.
- [GMA07] A. L. Galdino, C. Muñoz, and M. Ayala-Rincón. “Formal Verification of an Optimal Air Traffic Conflict Resolution and Recovery Algorithm”. In: *Proceedings of the 14th International Workshop on Logic, Language, Information and Computation (WoLLIC 2007)*. Ed. by D. Leivant and R. de Queiroz. Vol. 4576. Lecture Notes in Computer Science. Rio de Janeiro, Brazil: Springer Berlin Heidelberg, 2007, pp. 177–188. DOI: 10.1007/978-3-540-73445-1_13. URL: http://dx.doi.org/10.1007/978-3-540-73445-1_13.
- [GS92] S. M. German and A. P. Sistla. “Reasoning about systems with many processes”. In: *Journal of the ACM (JACM)* 39.3 (1992), pp. 675–735.
- [GSS15] D. Guck, J. J. Spel, and M. I. A. Stoelinga. “DFTCalc: Reliability centered maintenance via fault tree analysis (tool paper)”. In: *Proceedings of the 17th International Conference on Formal Engineering Methods (ICFEM 2015)*. Vol. 9407. Lecture Notes in Computer Science. Paris, France: Springer Verlag, 2015, pp. 304–311. URL: http://dx.doi.org/10.1007/978-3-319-25423-4_19.
- [GTHRS14a] D. Guck, M. Timmer, H. Hatefi, E. J. J. Ruijters, and M. I. A. Stoelinga. *Modelling and analysis of Markov reward automata (extended version)*. Technical Report TR-CTIT-14-06. Enschede: Centre for Telematics and Information Technology, University of Twente, 2014. URL: <http://eprints.eemcs.utwente.nl/24800/>.
- [GTHRS14b] D. Guck, M. Timmer, H. Hatefi, E. J. J. Ruijters, and M. I. A. Stoelinga. “Modelling and analysis of Markov reward automata”. In: *Proceedings of the 12th International Symposium on Automated Technology for Verification and Analysis (ATVA 2014)*, Sydney, NSW, Australia. Vol. 8837. Lecture Notes in Computer Science. Sydney, NSW, Australia: Springer Verlag, 2014, pp. 168–184. URL: http://dx.doi.org/10.1007/978-3-319-11936-6_13.
- [Gho+14] K. Ghorbal et al. “Hybrid Theorem Proving of Aerospace Systems: Applications and Challenges”. In: *Journal of Aerospace Information Systems* 11.10 (2014), pp. 702–713. DOI: 10.2514/1.I010178. URL: <http://dx.doi.org/10.2514/1.I010178>.

- [Gia+14] D. Giannakopoulou et al. “Taming test inputs for separation assurance”. In: *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, (ASE 2014)*. Vasteras, Sweden: ACM, 2014, pp. 373–384. DOI: 10.1145/2642937.2642940. URL: <http://doi.acm.org/10.1145/2642937.2642940>.
- [HCHKB02] B. R. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, and C. Baier. “Model Checking Performability Properties”. In: *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2002)*. IEEE, 2002, pp. 103–112. ISBN: 0-7695-1597-5. DOI: 10.1109/DSN.2002.1028891.
- [HH12] H. Hatefi and H. Hermanns. “Model Checking Algorithms for Markov Automata”. In: *Electronic Communications of the ECEASST* 53 (2012). DOI: 10.14279/tuj.eceasst.53.783. URL: <http://journal.ub.tu-berlin.de/eceasst/article/view/783>.
- [HHK00] B. R. Haverkort, H. Hermanns, and J.-P. Katoen. “On the use of model checking techniques for dependability evaluation”. In: *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS 2000)*. IEEE, 2000, pp. 228–237. DOI: 10.1109/RELDI.2000.885410.
- [HJ94] H. Hansson and B. Jonsson. “A Logic for Reasoning about Time and Reliability”. In: *Formal Aspects of Computing* 6.5 (1994), pp. 512–535.
- [HMS+02] L. R. Higgins, R. K. Mobley, R. Smith, et al. *Maintenance engineering handbook*. McGraw-Hill, 2002.
- [HT02] A. Horváth and M. Telek. “PhFit: A General Phase-Type Fitting Tool”. In: *Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS 2002)*. Vol. 2324. Lecture Notes in Computer Science. London, UK: Springer Berlin Heidelberg, 2002, pp. 82–91. ISBN: 978-3-540-46029-9. DOI: 10.1007/3-540-46029-2_5. URL: http://dx.doi.org/10.1007/3-540-46029-2_5.
- [Hat+15] H. Hatefi et al. “Cost vs. Time in Stochastic Games and Markov Automata”. In: *Proceedings of the 1st International Symposium on Dependable Software Engineering: Theories, Tools, and Applications (SETTA 2015)*. Ed. by X. Li, Z. Liu, and W. Yi. Vol. 9409. Lecture Notes in Computer Science. Nanjing, China: Springer International Publishing, 2015, pp. 19–34. DOI: 10.1007/978-3-319-25942-0_2. URL: http://dx.doi.org/10.1007/978-3-319-25942-0_2.

- [Her02] H. Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*. Vol. 2428. Lecture Notes in Computer Science. Springer, 2002.
- [Hix68] A. Hixenbaugh. *Fault tree for safety*. Tech. rep. DTIC Document, 1968.
- [Hol04] G. J. Holzmann. *The SPIN model checker: Primer and reference manual*. Vol. 1003. Addison-Wesley Reading, 2004.
- [IEC60812] *Analysis techniques for system reliability Procedure for failure mode and effects analysis*. Tech. rep. NEN-EN-IEC 60812:2006. 2006.
- [IEC61025] *Fault tree analysis (FTA)*. Tech. rep. IEC 60050:2006. 2007.
- [ISO31000] International Organization for Standardization. *Risk management Principles and guidelines*. Standard ISO 31000:2009. Geneva, Switzerland, 2009.
- [Int04] International Society of Automotive Engineers. *Architecture Analysis and Design Language (AADL)*. SAE Standard AS5506. 2004.
- [Iso16] Isograph. *Fault Tree+*. Tech. rep. 2016.
- [JGKRS15] S. Junges, D. Guck, J.-P. Katoen, A. Rensink, and M. I. A. Stoelinga. “Fault Trees on a Diet - Automated Reduction by Graph Rewriting”. In: *Proceedings of the 1st International Symposium on Dependable Software Engineering: Theories, Tools, and Applications (SETTA 2015)*. Ed. by X. Li, Z. Liu, and W. Yi. Vol. 9409. Lecture Notes in Computer Science. Nanjing, China: Springer Verlag, 2015, pp. 3–18. URL: http://dx.doi.org/10.1007/978-3-319-25942-0_1.
- [JGKRS16] S. Junges, D. Guck, J.-P. Katoen, A. Rensink, and M. I. A. Stoelinga. “Fault Trees on a Diet - Automated Reduction by Graph Rewriting”. In: *Formal Aspects of Computing*. accepted for publication. 2016.
- [JGKS16] S. Junges, D. Guck, J.-P. Katoen, and M. I. A. Stoelinga. “Uncovering Dynamic Fault Trees”. In: *Proceedings of the 46th annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2016)*. Toulouse, France: IEEE Computer Society, 2016, pp. 299–310. DOI: 10.1109/DSN.2016.35.
- [JT88] M. A. Johnson and M. R. Taaffe. *The Denseness of Phase Distributions*. School of Industrial Engineering Research Memoranda 88-20. Purdue University, 1988.

- [JVB07] A. Joshi, S. Vestal, and P. Binns. “Automatic generation of static fault trees from AADL models”. In: *Workshop on Architecting Dependable Systems of The 37th Annual IEEE/I-FIP Int. Conference on Dependable Systems and Networks, Edinburgh, UK*. 2007.
- [Jea+15] J. Jeannin et al. “A Formally Verified Hybrid System for the Next-Generation Airborne Collision Avoidance System”. In: *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2015)*. Ed. by C. Baier and C. Tinelli. Vol. 9035. Lecture Notes in Computer Science. London, UK: Springer Berlin Heidelberg, 2015, pp. 21–36. DOI: 10.1007/978-3-662-46681-0_2. URL: http://dx.doi.org/10.1007/978-3-662-46681-0_2.
- [Joh08] S. Johr. “Model checking compositional Markov systems”. PhD thesis. 2008.
- [KB08] J.-P. Katoen and C Baier. *Principles of model checking*. The MIT Press, 2008.
- [KBCHPC14] S. Kriaa, M. Bouissou, F. Colin, Y. Halgand, and L. Pietre-Cambacedes. “Safety and Security Interactions Modeling Using the BDMP Formalism: Case Study of a Pipeline”. In: *Proceedings of the 33rd International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2014)*. Ed. by A. Bondavalli and F. Di Giandomenico. Florence, Italy: Springer International Publishing, 2014, pp. 326–341. ISBN: 978-3-319-10506-2. DOI: 10.1007/978-3-319-10506-2_22. URL: http://dx.doi.org/10.1007/978-3-319-10506-2_22.
- [KC11] M. J. Kochenderfer and J. P. Chryssanthacopoulos. *Robust Airborne Collision Avoidance through Dynamic Programming*. Project Report ATC-371. Massachusetts Institute of Technology, Lincoln Laboratory, 2011.
- [KD07] J. Kuchar and A. C. Drumm. “The traffic alert and collision avoidance system”. In: *Lincoln Laboratory Journal* 16.2 (2007), p. 277.
- [KHC12] M. J. Kochenderfer, J. E. Holland, and J. P. Chryssanthacopoulos. “Next generation airborne collision avoidance system”. In: *Lincoln Laboratory Journal* 19.1 (2012), pp. 17–33.
- [KNP08] M. Kwiatkowska, G. Norman, and D. Parker. “Using Probabilistic Model Checking in Systems Biology”. In: *ACM SIGMETRICS Performance Evaluation Review* 35.4 (2008), pp. 14–21. ISSN: 0163-5999. DOI: 10.1145/1364644.1364651. URL: <http://doi.acm.org/10.1145/1364644.1364651>.

- [KNP11] M. Kwiatkowska, G. Norman, and D. Parker. “PRISM 4.0: Verification of Probabilistic Real-Time Systems”. In: *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV 2011)*. Ed. by G. Gopalakrishnan and S. Qadeer. Vol. 6806. Lecture Notes in Computer Science. Snowbird, USA: Springer Berlin Heidelberg, 2011, pp. 585–591. ISBN: 978-3-642-22110-1. DOI: 10.1007/978-3-642-22110-1_47. URL: http://dx.doi.org/10.1007/978-3-642-22110-1_47.
- [KP14] K. C. Kapur and M. Pecht. *Reliability engineering*. John Wiley & Sons, 2014.
- [KZHHJ11] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. “The ins and outs of the probabilistic model checker MRMC”. In: *Performance Evaluation* 68.2 (2011). Advances in Quantitative Evaluation of Systems (QEST 2009), pp. 90–104. ISSN: 0166-5316. DOI: <http://dx.doi.org/10.1016/j.peva.2010.04.001>. URL: <http://www.sciencedirect.com/science/article/pii/S0166531610000660>.
- [Kan+15] G. Kant et al. “LTSmin: high-performance language-independent model checking”. In: *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2015)*. Vol. 9035. Lecture Notes in Computer Science. London, UK: Springer Berlin Heidelberg, 2015, pp. 692–707. ISBN: 978-3-662-46681-0. DOI: 10.1007/978-3-662-46681-0_61. URL: http://dx.doi.org/10.1007/978-3-662-46681-0_61.
- [Koc15] M. J. Kochenderfer. *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.
- [LFKFH14] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann. “Industry 4.0”. In: *Business & Information Systems Engineering* 6.4 (2014), pp. 239–242. ISSN: 1867-0202. DOI: 10.1007/s12599-014-0334-4. URL: <http://dx.doi.org/10.1007/s12599-014-0334-4>.
- [LGTL85] W.-S. Lee, D. L. Grosh, F. A. Tillman, and C. H. Lie. “Fault Tree Analysis, Methods, and Applications — A Review”. In: *IEEE transactions on reliability* 34.3 (1985), pp. 194–203.
- [LKMB015] R. Lee, M. J. Kochenderfer, O. J. Mengshoel, G. P. Brat, and M. P. Owen. “Adaptive stress testing of airborne collision avoidance systems”. In: *Proceedings of the 34th IEEE/AIAA Digital Avionics Systems Conference (DASC 2015)*. IEEE, 2015, pp. 6C2–1.

- [LL97] J. Lygeros and N. Lynch. “On the formal verification of the TCAS conflict resolution algorithms”. In: *Proceedings of the 36th IEEE Conference on Decision and Control*. IEEE, 1997, pp. 1829–1834. DOI: 10.1109/CDC.1997.657846.
- [LLL13] H.-C. Liu, L. Liu, and N. Liu. “Risk evaluation approaches in failure mode and effects analysis: A literature review”. In: *Expert systems with applications* 40.2 (2013), pp. 828–838.
- [LRP13] S. M. Loos, D. W. Renshaw, and A. Platzer. “Formal verification of distributed aircraft controllers”. In: *Proceedings of the 16th international conference on Hybrid systems: computation and control (HSCC 2013)*. Philadelphia, PA, USA: ACM, 2013, pp. 125–130. DOI: 10.1145/2461328.2461350. URL: <http://doi.acm.org/10.1145/2461328.2461350>.
- [LT93] N. G. Leveson and C. S. Turner. “An investigation of the Therac-25 accidents”. In: *Computer* 26.7 (1993), pp. 18–41. ISSN: 0018-9162. DOI: 10.1109/MC.1993.274940.
- [LXZLL07] D. Liu, W. Xing, C. Zhang, R. Li, and H. Li. “Cut sequence set generation for fault tree analysis”. In: *Proceedings of the 3rd International Conference on Embedded Software and Systems (ICESS 2007 2007)*. Ed. by Y.-H. Lee et al. Vol. 4523. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 592–603. ISBN: 978-3-540-72685-2. DOI: 10.1007/978-3-540-72685-2_55. URL: http://dx.doi.org/10.1007/978-3-540-72685-2_55.
- [MBCDF94] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with generalized stochastic Petri nets*. John Wiley & Sons, Inc., 1994.
- [MIL-STD-1629A] U.S. Department of Defense. *MIL-STD-1629A: Procedures for Performing a Failure Mode, Effects and Criticality Analysis*. Department of Defense, 1980.
- [MPBCR06] S. Montani, L. Portinale, A. Bobbio, and D. Codetta-Raiteri. “Automatically translating dynamic fault trees into dynamic Bayesian networks by means of a software tool”. In: *Proceedings of the 1st International Conference on Availability, Reliability and Security (ARES 2006)*. IEEE, 2006, pp. 804–810. DOI: 10.1109/ARES.2006.34.
- [MPBCR08] S. Montani, L. Portinale, A. Bobbio, and D. Codetta-Raiteri. “Radyban: A tool for reliability analysis of dynamic fault trees through conversion into dynamic Bayesian networks”. In: *Reliability Engineering & System Safety* 93.7 (2008), pp. 922–932. ISSN: 0951-8320. DOI: <http://dx.doi.org/10.1016/j.ress.2007.03.013>.

- [MPBVCR06] S. Montani, L. Portinale, A. Bobbio, M. Varesio, and D. Codetta-Raiteri. “A tool for automatically translating dynamic fault trees into dynamic Bayesian networks”. In: *Proceedings of the Annual Reliability and Maintainability Symposium (RAMS 2006)*. IEEE, 2006, pp. 434–441. DOI: 10.1109/RAMS.2006.1677413.
- [Mar10] R. N. Martin. “Model Checking Nondeterministic and Randomly Timed Systems”. PhD thesis. PhD thesis, RWTH Aachen, 2010.
- [Mea65] A. Mearns. “Fault tree analysis - the study of unlikely events in complex systems (Fault tree analysis as tool to identify component failure as probable cause of undesired event in complex system)”. In: *System Safety Symposium*. Seattle, Washington, 1965, p. 1965.
- [Mil89] R. Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989. ISBN: 978-0-13-115007-2.
- [Mou97] J. Moubray. *Reliability centered maintenance*. Industrial Press, 1997.
- [PC09] A. Platzer and E. M. Clarke. “Formal Verification of Curved Flight Collision Avoidance Maneuvers: A Case Study”. In: *FM 2009: Formal Methods, Second World Congress*. Ed. by A. Cavalcanti and D. R. Dams. Eindhoven, The Netherlands, 2009, pp. 547–562. DOI: 10.1007/978-3-642-05089-3_35. URL: http://dx.doi.org/10.1007/978-3-642-05089-3_35.
- [PCB10] L. Piètre-Cambacédès and M. Bouissou. “Beyond Attack Trees: Dynamic Security Modeling with Boolean Logic Driven Markov Processes (BDMP)”. In: *Proceedings of the European Dependable Computing Conference (EDCC 2010)*. IEEE, 2010, pp. 199–208. DOI: 10.1109/EDCC.2010.32.
- [Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., 1988. ISBN: 0-934613-73-7.
- [Pet77] J. L. Peterson. “Petri Nets”. In: *ACM Computing Surveys* 9.3 (1977), pp. 223–252. ISSN: 0360-0300. DOI: 10.1145/356698.356702. URL: <http://doi.acm.org/10.1145/356698.356702>.
- [Pro10] ProRail. *Leidraad voor RAMSHE - LMC-studie, HDL00032*. Tech. rep. (in Dutch). 2010.
- [Put94] M. L. Puterman. *Markov Decision Processes*. Wiley, 1994.

- [QS82] J. P. Queille and J. Sifakis. “Specification and verification of concurrent systems in CESAR”. In: *Proceedings of the 5th International Symposium on Programming*. Ed. by M. Dezani-Ciancaglini and U. Montanari. Turin, Italy: Springer Berlin Heidelberg, 1982, pp. 337–351. ISBN: 978-3-540-39184-5. DOI: 10.1007/3-540-11494-7_22. URL: http://dx.doi.org/10.1007/3-540-11494-7_22.
- [RFIV04] D. C. Raiteri, G. Franceschinis, M. Iacono, and V. Vittorini. “Repairable fault tree for the automatic evaluation of repair policies”. In: *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2004)*. IEEE, 2004, pp. 659–668. DOI: 10.1109/DSN.2004.1311936.
- [RGDPS16] E. J. Ruijters, D. Guck, P. Drolenga, M. Peters, and M. I. A. Stoelinga. “Maintenance Analysis and Optimization via Statistical Model Checking”. In: *Proceedings of the 13th International Conference on Quantitative Evaluation of Systems (QEST 2016)*. Ed. by G. Agha and B. Van Houdt. Quebec City, QC, Canada: Springer International Publishing, 2016, pp. 331–347. ISBN: 978-3-319-43425-4. DOI: 10.1007/978-3-319-43425-4_22. URL: http://dx.doi.org/10.1007/978-3-319-43425-4_22.
- [RGDS16] E. J. Ruijters, D. Guck, P. Drolenga, and M. I. A. Stoelinga. “Fault maintenance trees: reliability centered maintenance via statistical model checking”. In: *Proceedings of the Annual Reliability and Maintainability Symposium (RAMS 2016)*. IEEE, 2016, pp. 1–6. DOI: 10.1109/RAMS.2016.7447986.
- [RGvS16] E. J. J. Ruijters, D. Guck, M. van Noort, and M. I. A. Stoelinga. “Reliability-centered maintenance of the Electrically Insulated Railway Joint via Fault Tree Analysis: A practical experience report”. In: *Proceedings of the 46th annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2016)*. Toulouse, France: IEEE Computer Society, 2016, pp. 662–669. DOI: 10.1109/DSN.2016.67.
- [RH04] M. Rausand and A. Høyland. *System reliability theory: models, statistical methods, and applications*. Vol. 396. John Wiley & Sons, 2004.
- [RS15] E. Ruijters and M. I. A. Stoelinga. “Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools”. In: *Computer Science Review* 15 (2015), pp. 29–62.
- [Ras10] D. Rastler. *Electricity energy storage technology options: a white paper primer on applications, costs and benefits*. Electric Power Research Institute, 2010.

- [SDC99] K. J. Sullivan, J. B. Dugan, and D. Coppit. “The Galileo fault tree analysis tool”. In: *Proceedings of the 29th International Symposium on Fault-Tolerant Computing (FTCS 1999)*. IEEE. 1999, pp. 232–235.
- [SP11] STW and ProRail. *STW - ProRail Partnership Program Plan ExploRail 2011 - 2016*. Tech. rep. 2011.
- [Seg95] R. Segala. “Modeling and verification of randomized distributed real-time systems”. PhD thesis. 1995.
- [Sri91] M. M. Srinivasan. “Nondeterministic polling systems”. In: *Management Science* 37.6 (1991), pp. 667–681.
- [Sta+02] M. Stamatelatos et al. *Fault tree handbook with aerospace applications*. 2002.
- [TBS11] M. Timmer, H. Brinksma, and M. I. A. Stoelinga. “Model-Based Testing”. In: *Software and Systems Safety: Specification and Verification*. Ed. by M. Broy, C. Leuxner, and C. A. R. Hoare. Vol. 30. NATO Science for Peace and Security Series D: Information and Communication Security. Amsterdam: IOS Press, 2011, pp. 1–32.
- [TBT06] A. Thummler, P. Buchholz, and M. Telek. “A Novel Approach for Phase-Type Fitting with the EM Algorithm”. In: *IEEE Transactions on Dependable and Secure Computing* 3.3 (2006), pp. 245–258. ISSN: 1545-5971.
- [TD04] Z. Tang and J. B. Dugan. “Minimal cut set/sequence generation for dynamic fault trees”. In: *Proceedings of the Annual Reliability and Maintainability Symposium (RAMS 2004)*. IEEE, 2004, pp. 207–213.
- [TKPS16] M. Timmer, J.-P. Katoen, J. van de Pol, and M. I. A. Stoelinga. “Confluence reduction for Markov automata”. In: *Theoretical Computer Science* (2016).
- [TPS98] C. Tomlin, G. J. Pappas, and S. Sastry. “Conflict resolution for air traffic management: A study in multiagent hybrid systems”. In: *IEEE Transactions on Automatic Control* 43.4 (1998), pp. 509–521. ISSN: 0018-9286. DOI: 10.1109/9.664154.
- [Tim11] M. Timmer. “SCOOP: A tool for symbolic optimisations of probabilistic processes”. In: *Proceedings of the 8th International Conference on the Quantitative Evaluation of Systems (QEST 2011)*. IEEE. 2011, pp. 149–150. DOI: 10.1109/QEST.2011.27.
- [Tim13] M. Timmer. “Efficient Modelling, Generation and Analysis of Markov Automata”. PhD thesis. Enschede: University of Twente, 2013.

- [Tre08] J. Tretmans. “Formal Methods and Testing”. In: ed. by R. M. Hierons, J. P. Bowen, and M. Harman. Berlin, Heidelberg: Springer-Verlag, 2008. Chap. Model Based Testing with Labelled Transition Systems, pp. 1–38. ISBN: 3-540-78916-2, 978-3-540-78916-1. URL: <http://dl.acm.org/citation.cfm?id=1806209.1806210>.
- [VGRH81] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. *Fault tree handbook. Technical Report NUREG-0492*. 1981.
- [VJK16] M. Volk, S. Junges, and J.-P. Katoen. “Advancing Dynamic Fault Tree Analysis - Get Succinct State Spaces Fast and Synthesise Failure Rates”. In: (2016). Ed. by A. Skavhaug, J. Guiochet, and F. Bitsch. DOI: 10.1007/978-3-319-45477-1_20. URL: http://dx.doi.org/10.1007/978-3-319-45477-1_20.
- [WJ06] N. Wolovick and S. Johr. “A Characterization of Meaningful Schedulers for Continuous-Time Markov Decision Processes”. In: *Proceedings of the 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2006)*. Ed. by E. Asarin and P. Bouyer. Vol. 4202. Lecture Notes in Computer Science. Paris, France: Springer Berlin Heidelberg, 2006, pp. 352–367. ISBN: 978-3-540-45031-3. DOI: 10.1007/11867340_25. URL: http://dx.doi.org/10.1007/11867340_25.
- [WP09] M. Walker and Y. Papadopoulos. “Qualitative temporal analysis: Towards a full implementation of the Fault Tree Handbook”. In: *Control Engineering Practice* 17.10 (2009), pp. 1115–1125. ISSN: 0967-0661. DOI: <http://dx.doi.org/10.1016/j.conengprac.2008.10.003>.
- [WP10] M. Walker and Y. Papadopoulos. “A Hierarchical Method for the Reduction of Temporal Expressions in Pandora”. In: *Proceedings of the 1st Workshop on Dynamic Aspects in Dependability Models for Fault-Tolerant Systems (DYADEM-FTS 2010)*. Valencia, Spain: ACM, 2010, pp. 7–12. ISBN: 978-1-60558-916-9. DOI: 10.1145/1772630.1772634.
- [Wun96] R. Wunderling. “Paralleler und objektorientierter Simplex Algorithmus”. PhD thesis. Technische Universität Berlin, 1996.
- [ZN10] L. Zhang and M. Neuhäüßer. “Model Checking Interactive Markov Chains”. In: *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2010)*. Ed. by J. Esparza and R. Majumdar. Vol. 6015. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 53–68. ISBN: 978-3-642-12002-2. DOI: 10.1007/978-3-642-12002-2_5.

Titles in the IPA Dissertation Series since 2014

- J. van den Bos.** *Gathering Evidence: Model-Driven Software Engineering in Automated Digital Forensics.* Faculty of Science, UvA. 2014-01
- D. Hadziosmanovic.** *The Process Matters: Cyber Security in Industrial Control Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-02
- A.J.P. Jeckmans.** *Cryptographically-Enhanced Privacy for Recommender Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-03
- C.-P. Bezemer.** *Performance Optimization of Multi-Tenant Software Systems.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2014-04
- T.M. Ngo.** *Qualitative and Quantitative Information Flow Analysis for Multi-threaded Programs.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-05
- A.W. Laarman.** *Scalable Multi-Core Model Checking.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-06
- J. Winter.** *Coalgebraic Characterizations of Automata-Theoretic Classes.* Faculty of Science, Mathematics and Computer Science, RU. 2014-07
- W. Meulemans.** *Similarity Measures and Algorithms for Cartographic Schematization.* Faculty of Mathematics and Computer Science, TU/e. 2014-08
- A.F.E. Belinfante.** *JTorX: Exploring Model-Based Testing.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-09
- A.P. van der Meer.** *Domain Specific Languages and their Type Systems.* Faculty of Mathematics and Computer Science, TU/e. 2014-10
- B.N. Vasilescu.** *Social Aspects of Collaboration in Online Software Communities.* Faculty of Mathematics and Computer Science, TU/e. 2014-11
- F.D. Aarts.** *Tomte: Bridging the Gap between Active Learning and Real-World Systems.* Faculty of Science, Mathematics and Computer Science, RU. 2014-12
- N. Noroozi.** *Improving Input-Output Conformance Testing Theories.* Faculty of Mathematics and Computer Science, TU/e. 2014-13
- M. Helvensteijn.** *Abstract Delta Modeling: Software Product Lines and Beyond.* Faculty of Mathematics and Natural Sciences, UL. 2014-14
- P. Vullers.** *Efficient Implementations of Attribute-based Credentials on Smart Cards.* Faculty of Science, Mathematics and Computer Science, RU. 2014-15
- F.W. Takes.** *Algorithms for Analyzing and Mining Real-World Graphs.* Faculty of Mathematics and Natural Sciences, UL. 2014-16
- M.P. Schraagen.** *Aspects of Record Linkage.* Faculty of Mathematics and Natural Sciences, UL. 2014-17
- G. Alpár.** *Attribute-Based Identity Management: Bridging the Cryptographic Design of ABCs with the Real World.* Faculty of Science,

Mathematics and Computer Science, RU. 2015-01

A.J. van der Ploeg. *Efficient Abstractions for Visualization and Interaction.* Faculty of Science, UvA. 2015-02

R.J.M. Theunissen. *Supervisory Control in Health Care Systems.* Faculty of Mechanical Engineering, TU/e. 2015-03

T.V. Bui. *A Software Architecture for Body Area Sensor Networks: Flexibility and Trustworthiness.* Faculty of Mathematics and Computer Science, TU/e. 2015-04

A. Guzzi. *Supporting Developers' Teamwork from within the IDE.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-05

T. Espinha. *Web Service Growing Pains: Understanding Services and Their Clients.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-06

S. Dietzel. *Resilient In-network Aggregation for Vehicular Networks.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-07

E. Costante. *Privacy throughout the Data Cycle.* Faculty of Mathematics and Computer Science, TU/e. 2015-08

S. Cranen. *Getting the point — Obtaining and understanding fixpoints in model checking.* Faculty of Mathematics and Computer Science, TU/e. 2015-09

R. Verduft. *The (in)security of proprietary cryptography.* Faculty of Science, Mathematics and Computer Science, RU. 2015-10

J.E.J. de Ruiter. *Lessons learned in the analysis of the EMV and TLS security protocols.* Faculty of Science, Mathematics and Computer Science, RU. 2015-11

Y. Dajsuren. *On the Design of an Architecture Framework and Quality Evaluation for Automotive Software Systems.* Faculty of Mathematics and Computer Science, TU/e. 2015-12

J. Bransen. *On the Incremental Evaluation of Higher-Order Attribute Grammars.* Faculty of Science, UU. 2015-13

S. Picek. *Applications of Evolutionary Computation to Cryptology.* Faculty of Science, Mathematics and Computer Science, RU. 2015-14

C. Chen. *Automated Fault Localization for Service-Oriented Software Systems.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-15

S. te Brinke. *Developing Energy-Aware Software.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-16

R.W.J. Kersten. *Software Analysis Methods for Resource-Sensitive Systems.* Faculty of Science, Mathematics and Computer Science, RU. 2015-17

J.C. Rot. *Enhanced coinduction.* Faculty of Mathematics and Natural Sciences, UL. 2015-18

M. Stolikj. *Building Blocks for the Internet of Things.* Faculty of Mathematics and Computer Science, TU/e. 2015-19

D. Gebler. *Robust SOS Specifications of Probabilistic Processes.* Faculty of Sciences, Department of Computer Science, VUA. 2015-20

- M. Zaharieva-Stojanovski.** *Closer to Reliable Software: Verifying functional behaviour of concurrent programs.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-21
- R.J. Krebbers.** *The C standard formalized in Coq.* Faculty of Science, Mathematics and Computer Science, RU. 2015-22
- R. van Vliet.** *DNA Expressions – A Formal Notation for DNA.* Faculty of Mathematics and Natural Sciences, UL. 2015-23
- S.-S.T.Q. Jongmans.** *Automata-Theoretic Protocol Programming.* Faculty of Mathematics and Natural Sciences, UL. 2016-01
- S.J.C. Joosten.** *Verification of Interconnects.* Faculty of Mathematics and Computer Science, TU/e. 2016-02
- M.W. Gazda.** *Fixpoint Logic, Games, and Relations of Consequence.* Faculty of Mathematics and Computer Science, TU/e. 2016-03
- S. Keshishzadeh.** *Formal Analysis and Verification of Embedded Systems for Healthcare.* Faculty of Mathematics and Computer Science, TU/e. 2016-04
- P.M. Heck.** *Quality of Just-in-Time Requirements: Just-Enough and Just-in-Time.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2016-05
- Y. Luo.** *From Conceptual Models to Safety Assurance – Applying Model-Based Techniques to Support Safety Assurance.* Faculty of Mathematics and Computer Science, TU/e. 2016-06
- B. Ege.** *Physical Security Analysis of Embedded Devices.* Faculty of Science, Mathematics and Computer Science, RU. 2016-07
- A.I. van Goethem.** *Algorithms for Curved Schematization.* Faculty of Mathematics and Computer Science, TU/e. 2016-08
- T. van Dijk.** *Sylvan: Multi-core Decision Diagrams.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2016-09
- I. David.** *Run-time resource management for component-based systems.* Faculty of Mathematics and Computer Science, TU/e. 2016-10
- A.C. van Hulst.** *Control Synthesis using Modal Logic and Partial Bisimilarity – A Treatise Supported by Computer Verified Proofs.* Faculty of Mechanical Engineering, TU/e. 2016-11
- A. Zawedde.** *Modeling the Dynamics of Requirements Process Improvement.* Faculty of Mathematics and Computer Science, TU/e. 2016-12
- F.M.J. van den Broek.** *Mobile Communication Security.* Faculty of Science, Mathematics and Computer Science, RU. 2016-13
- J.N. van Rijn.** *Massively Collaborative Machine Learning.* Faculty of Mathematics and Natural Sciences, UL. 2016-14
- M.J. Steindorfer.** *Efficient Immutable Collections.* Faculty of Science, UvA. 2017-01
- W. Ahmad.** *Green Computing: Efficient Energy Management of Multi-processor Streaming Applications via Model Checking.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-02

D. Guck. *Reliable Systems – Fault tree analysis via Markov reward automata.* Faculty of Electrical En-

gineering, Mathematics & Computer Science, UT. 2017-03

DE hedendaagse maatschappij laat zich kenmerken door de alomverteenwoordiging van hardware en software systemen waarop we dagelijks vertrouwen. Van vervoermiddelen zoals auto's, treinen en vliegtuigen tot medische apparatuur in ziekenhuizen tot kerncentrales. Bovendien kunnen we een trend van automatisering en gegevensoverdracht observeren in de hedendaagse maatschappij en economie, onder andere bij de opkomst van *cyber-physical systems*, *internet of things* en *cloud computing*. Deze systemen hebben een eis gemeen: ze moeten *veilig* en *betrouwbaar* functioneren. Maar hoe kunnen we hierop vertrouwen?

Model checking is een techniek om te bepalen of een systeem aan een eis voldoet. Om dit te bepalen, wordt er een model gemaakt van het systeem en de eisen omgezet naar logische formules. De *model checker* kan vervolgens bepalen of de formules correct zijn met betrekking tot het model. Wanneer een formule niet correct is, geeft de model checker een tegenvoorbeeld. Belangrijk is dat model checking zowel op hardware als op software toegepast kan worden en is succesvol ingezet ten behoeve van verscheidene toepassing zoals ruimtevaartsystemen en biologische systemen.

Reliability engineering is een reeds lang gevestigd vakgebied met als doel het ontwikkelen van tools ten behoeve van de betrouwbaarheid, beschikbaarheid, onderhoudbaarheid en veiligheid (RAMS¹) van complexe systemen en ook ondersteuning bieden bij het ontwikkelen, produceren en onderhouden van deze eigenschappen. Echter, met de hedendaagse alomverteenwoordiging van hardware en software is het nodig dat ook methoden en tools voor *reliability engineering* aangepast worden.

Deze dissertatie bevordert het gebieden van model checking en reliability engineering. Aan de ene kant introduceren wij een *beloningsextensie* op *Markov-automaten* en presenteren algoritmes voor verschillende beloningseigenschappen. Aan de andere kant breiden wij *foutbomen* uit met *onderhoudprocedures*.

In de eerste helft van deze dissertatie introduceren wij *Markov-beloning-automaten* (MRAs), met ondersteuning voor non-deterministische keuzes, zowel discrete als continue kansverdeling en zowel onmiddellijke beloningen als beloningen met tijd. Bovendien introduceren wij algoritmes voor bereikbaarheidsdoeleinden voor MRAs. In het bijzonder definiëren wij de verwachte beloningsdoeleinden voor doel- en tijdgelimiteerde beloningen en voor de gemiddelde beloning over langere tijd.

¹reliability, availability, maintainability and safety

In de tweede helft introduceren wij *fout-onderhoudsbomen* (FMTs). Dit is een uitbreiding op *dynamische foutbomen* (DFTs) met modellen voor onderhoud ten behoeve van verbetering en voorkoming. Het voordeel van FMTs is dat de onderhoudstrategieën direct op het niveau van de foutboom is gedefinieerd. Hierdoor is het effect van onderhoud direct vertaalbaar naar een analyse en dit brengt ons in staat om slimmere onderhoudprocedures te vinden.

Ten slotte introduceren wij een tool die onze aanpak implementeert. Bovendien hebben wij onderzoek gedaan in de praktijk om de kwaliteiten van FMTs te evalueren om een realistisch scenario te modelleren en te analyseren. In het bijzonder leggen wij de focus op RAMS analyse van een treinspoortraject in Nederland door verschillende onderhoudstrategieën ten behoeve van verbetering en voorkoming.