Otto-von-Guericke-Universität Magdeburg

Qualitative and Quantitative Formal Model-Based Safety Analysis

– Push the Safety Button –

Dissertation
zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von: Dipl.-Inf. Matthias Güdemann
geb. am 11.06.1980 in Augsburg

Gutachter: Jun.-Prof. Dr. Frank Ortmeier
Prof. Dr. Jean-Jacques Lesage
Prof. Dr. Rudolf Kruse

Ort und Datum des Promotionskolloquiums: Magdeburg, 29.09.2011

Acknowledgments

I would like to express my gratitude to my advisor and colleague Jun.-Prof. Dr. Frank Ortmeier, whose expertise in the areas of safety analysis, formal methods and mathematics and whose willingness to share his ideas and opinion in many discussions, contributed greatly to the successful completion of my dissertation thesis. I would like to thank Prof. Dr. Jean-Jacques Lesage and Prof. Dr. Rudolf Kruse for taking time from their busy schedule to serve as reviewers for my dissertation thesis.
My special thanks go to Prof. Dr. Wolfgang Reif, who encouraged and supported me from the very first semester of my university studies and continued to do so throughout my academic career. I learned a lot at the time at his chair at the University of Augsburg and the opportunity to work with many students in lectures, seminars and different exercises provided me with very valuable experiences.
I would also like to thank all my colleagues and now friends, both at Augsburg and Magdeburg for all the discussions, support and fun we had together – at the workplace, as well as in the leisure time.
My deepest gratitude goes to my family for the support they provided me through my life, my parents who encouraged me to follow whatever I wanted to do and in particular to my partner Agnes for showing me so much I would never have experienced without her.

Zusammenfassung

In vielen Anwendungsbereichen wird Software mehr und mehr zum Hauptinnovationsfaktor. Immer größere Teile der Funktionalität von Systemen werden durch Software implementiert, die auf generischer Hardware läuft. Dafür hat sich der Begriff der software-intensiven Systeme etabliert. Mittlerweile sind solche Systeme auch in sicherheitskritischen Bereichen weit verbreitet. Mit ihrer Verwendung geht eine enorme Erhöhung der Komplexität einher, welche den Nachweis der funktionalen Sicherheit immer schwieriger macht. Ein solcher Nachweis ist in sicherheitskritischen Bereichen jedoch notwendig und wird von den entsprechenden Zertifizierungsstellen gefordert. Die genauen Anforderungen dafür sind in domänenspezifischen Normen und Standards spezifiziert.

Die Verwendung formaler Methoden zur modellbasierten Sicherheitsanalyse kann den Sicherheitsnachweis für solche Systeme unterstützen. Dazu wird ein gemeinsames formales Systemmodell erstellt, welches sich der Entwickler und der Sicherheitsingenieur teilen. Dieses Modell besteht dabei aus einem abstrakten Modell des funktionalen Systems, einem Modell des physikalischen Umweltverhaltens, sowie einem Modell des Fehlverhaltens. Ein solches Modell kann in einer Sprache mit formaler Semantik ausgedrückt werden. Dies erlaubt dann eine Analyse mit automatischen Modellprüfern und unterstützt so den Sicherheitsanalyseprozess für komplexe Systeme. Der Vorteil gegenüber bisherigen Verfahren liegt dabei einmal in der Verwendung eines gemeinsamen Modells, was den notwendigen Aufwand bei Designänderungen verringert. Der zweite Vorteil liegt in der erhöhten Automatisierung, wodurch ein Sicherheitsnachweis effizienter durchgeführt werden kann.

Die Ergebnisse dieser Dissertation verbessern die bisher existierenden modellbasierten Analysemethoden wesentlich. Hauptaspekte dabei sind einmal die Erweiterung der analysierbaren Systemklasse sowie die Erweiterung der analysierbaren Eigenschaften. Desweiteren wurde eine neue, probabilistische Sicherheitsanalysemethodik geschaffen, die wesentlich genauere Ergebnisse liefern kann als dies mit bisherigen Analysen möglich war. Die Basis dazu bildet die formale Beschreibungssprache SAML (Safety Analysis and Modeling Language). Für diese wurde eine prototypische Werkzeugunterstützung geschaffen, die es erlaubt, SAML Modelle durch Modelltransformationen mit verschiedenen Verifikationstools zu analysieren. Dadurch profitiert der Ansatz von jeder Erweiterung der unterstützten Analysetools. Dieser Ansatz erlaubt eine Kombination verschiedener Analysemethoden und bildet die Basis für eine toolunabhängige Analyseplattform. Der Ansatz wird mit der Analyse von drei Fallstudien illustriert und bildet die Basis für das DFG Einzelforschungsprojekt “ProMoSA” (Probabilistic Models for Safety Analysis).

Abstract

Software is becoming the main innovation factor in many domains. Every more functionality is implemented in software running on relatively generic hardware. For this the notion of software-intensive systems has been established. By now such systems are already common in safety-critical domains. Their application causes an increase of complexity which makes the assurance of functional safety ever harder. Such evidence of safety is required in safety-critical domains and is required by the responsible certification authorities. The exact requirements are specified in domain-specific standards.
Using formal methods for model-based safety analysis can support the safety assurance of such systems. The basis is the construction of a common formal system model which is shared between the developer and the safety engineer. Such a model generally consists of an abstract model of the system, a model of the physical behavior of the environment and a model of the possible faults and failure modes. A model expressed in a language with formal semantics allows for the analysis using automatic verification tools and can therefore support the safety analysis process of complex systems. Compared to more traditional approaches the advantages are firstly that using a common system model requires less effort in case of design changes and secondly in the increased automation which make more efficient safety analysis possible.
The results of this dissertation thesis significantly advance existing safety analysis methods. Firstly, the class of analyzable systems is extended and secondly the set of analyzable properties is extended. In addition, a new probabilistic safety-analysis method was developed which produces much more accurate results than possible using existing methods. The basis is the formal specification language SAML (Safety Analysis and Modeling Language). A prototypical tool support with model transformations was developed which allows for analysis of SAML models with different verification tools. Therefore the approach benefits from all advancement in the development of the supported analysis tools. This allows the combination of different analysis methods and forms the basis for a tool-independent analysis framework. The approach is illustrated with three case studies and is the foundation for a new research project “ProMoSA” (Probabilistic Models for Safety Analysis) founded by the German Research Foundation (DFG).

Contents

List of Figures

1. Introduction

If there's more than one possible outcome of a job or task, and one of those outcomes will result in disaster or an undesirable consequence, then somebody will do it that way.

(Edward A. Murphy, Jr.)

The last decades have seen a very strong increase in the use of software-intensive systems in all areas of life. Embedded systems are becoming ubiquitous and software is one of the main innovation factors in most technical domains. This development also leads to application of software-intensive systems in many safety-critical areas where a malfunctioning poses major risks, not only in terms of economic loss, but also often endangers people. This makes safety assessment of these systems ever more important, but at the same time the complexity of these systems is increasing which steadily makes this more difficult.

This increase in complexity and the accompanying increased difficulty of conducting traditional safety analyses have already led to several tragic accidents. Examples are the Therac-25 accidents in the years 1985 to 1987 which are described by Leveson [LT93]. The Therac-25 was a device for radiation cancer therapy which – due to software errors – could deliver far too high a dose of radiation under certain circumstances. This led to three deaths and three severe injuries. The ultimate cause was erroneous software. However, the main reason for the consequences being so hazardous was that the original safety analysis system did not even consider software, and that the software was written without any method of self-checking or error correction [LT93].

One approach to prevent this kind of software errors is the use of programming languages specifically designed for the development of safety-critical systems. One very prominent example is the Ada language which was developed by Honeywell Bull in response to a request for proposals from the Department of Defense (DoD) of the USA and became standardized as ISO/IEC 8652 [Te95]. Its main features are strict static typing, various run-time checks and exceptions for different common programming errors. As a subset of Ada, SPARK was developed with even more properties for the development of software for safety-critical systems. Its formal definition allows for several static checks and encourages design by contract implementations with well defined pre- and postconditions. A further step in the development of correct software for safety-critical systems is Esterel Technologies' SCADE Suite [Est11]. It allows model-based graphical software development and provides a verification tool that can even prove some of the dynamic properties of the software and also facilitates traceability of requirements. Today, Ada, SPARK and SCADE are widely and successfully used to develop safety-critical software, especially in the domain of avionics, where malfunctioning of an important device potentially endangers hundreds of people.

Nevertheless, these approaches concentrate on software correctness only, but this alone cannot guarantee the overall safety of a system. Software is not used in isolation and therefore is not the only possible cause of a hazard. This can be seen from the accident during the first flight of Ariane 5. A well-tested and reliable software implementation became problematic, as it was used in an inappropriate environment. In 1996, the first flight of the rocket lead to the loss of the carrier and its satellite payload, marking an enormous economic loss. The accident is described by Lions [Lio96]. The software had been developed for the Ariane 4 which had very different flight properties. The different flight trajectories of the Ariane 5 led to the overflow of a variable which ultimately triggered the self-destruction mechanism of the rocket. More detailed analyses of the causes of the accident are given by LeLann [LL96] and Dowson [Dow97]. This illustrates that an isolated concentration on the software is not sufficient to guarantee safety.

A way to tackle this problem is to use a model-based approach which considers the complete system – including software, hardware and its surrounding physical environment – for safety analysis. The model-based approach proposed in this dissertation is based on the construction of a system model consisting of both the nominal behavior according to the system specification, as well as a model of potential erroneous behavior.

Such a model is used for safety analysis, both to find bugs that prevent correct functioning of the system and to analyze the behavior of the system following the occurrence of failures. The advantage of this is that model-based safety analysis is possible in the early phases of system design, when the elimination of design flaws is less expensive than in later phases.

Model-based safety analysis can partially be automated using automatic verification tools. These tools can deliver counterexamples if the specification is not fulfilled. The information from the counterexamples can be further examined and the results used for later tests of the implemented systems. A safety engineer can use such counterexamples to decide which failure mode combinations can lead to an overall system hazard and then refine the system design with risk-reducing measures. This is normally done until a desired level of failure tolerance is reached. Of course for every system there will always be a combination of failures which cannot be compensated for and the system can no longer work according to its specification. Whether this is tolerable for a specific system is most often decided by considering the *probability* that such a situation occurs.

The results of this dissertation allow to combine different safety analysis methods which provide information about the effect of failure occurrences and analysis methods which provide accurate quantitative measurements. This is a big step forward in design and development of safety-critical systems. Its application allows the early discovery of design flaws and also quantitative comparison of different system designs.

1.1. Main Contribution

This dissertation presents an approach for model-based safety analysis which for the first time allows for a combined consideration of both qualitative and quantitative aspects as well as the consideration of different types of failure modes. The main new contributions to the field of analysis of safety- critical systems are the following:

- Definition of the formal safety analysis and modeling language (SAML) for the convenient specification and analysis of safety-critical systems unifying existing and newly-developed methods in a semantically well-founded way.
- Extension of failure mode modeling methodologies with probabilistic aspects that allow a combination of per-time and per-demand failure mode types.
- Extension of existing qualitative model-based safety analysis methods, by making a broader class of systems analyzable and by making additional properties analyzable.
- Development of a new, quantitative model-based safety analysis method which allows much more accurate assessment of hazard probabilities than existing methods.

- Prototypical implementation of SAML.
- Prototypical implementation of semantically founded transformations for different formal verification tools.
- Application of the developed techniques to several realistic case studies to show the feasibility and validate the benefit of the developed modeling and analysis techniques.

The work presented in this dissertation resulted in 16 peer-reviewed publications that were presented at international conferences and workshops. The results developed in this dissertation form the basis for the project Probabilistic Models in Safety Analysis (ProMoSA) which is funded by the German Research Foundation (DFG). This project aims at creating an integrated model-driven development process for safety critical-systems and to use quantitative safety analysis to optimize systems.

1.2. Outline of the Dissertation

The outline of the dissertation is as follows: Chapter 2 gives an overview of different existing safety analysis methods which are used in different application domains. These are general related work from the field of safety analysis; specific related work topics are presented at the end of each of the following chapters. Chapter 3 presents formal basics, in particular syntax and semantics of SAML and temporal logic properties. This forms the basis for the following Chapters 4, 5 and 6.

Chapter 4 presents modeling guidelines for safety-critical systems analyzed with SAML. This includes, in particular, accurate failure mode modeling and physical behavior modeling in SAML. How such models can be analyzed correctly using different temporal logic proof obligations is presented in Chapter 5. The semantically founded transformation of SAML models into different state-of-the-art verification tools is explained in Chapter 6. The application of the results of these chapters to the safety analysis of case studies is shown in Chapter 7. Chapter 8 concludes the dissertation by giving a summary and an outlook of promising further research. All the proofs for the presented theorems and corollaries can be found in Appendix A. Appendix B lists the resulting peer-reviewed publications.

2. Safety Analysis Overview

Essentially, all models are wrong, but some are useful.

(George Box)

This chapter presents the definitions of the basic concepts that are used in this dissertation. This includes safety, hazard and failure modes. Their respective definitions are based on their usage in the scientific literature. An overview of other approaches to safety analysis is presented – although no overview can be complete, it includes the approaches most widely used in industry today as well as the most relevant related work – along with a classification of the proposed formal qualitative and quantitative model-based safety analysis.

Section 2.1 introduces the concepts used for safety analysis. Section 2.2 discusses a number of structured safety analysis approaches. Section 2.3 discusses failure logic modeling as a first step in the direction of automatic failure analysis, while Section 2.4 discusses some failure-injection based safety analysis methods. An overview of the proposed approach of this dissertation and its relation to existing approaches is given in Section 2.5.

2.1. Motivation and Concepts

In the area of safety analysis, the relevant concepts are often used with a slightly different meaning. Therefore the definitions used throughout this dissertation are given in the following section, which are based on the available scientific literature and standards of this domain.

Concepts

In [Lap95], Laprie defines the *safety* of a system as the "non-occurrence of catastrophic consequences on the environment" and lists safety as one of the different aspects of the broader concept of dependability. The other aspects are reliability, maintainability, confidentiality, integrity and availability. The meaning of safety as used in this dissertation is based on this definition. The safety of a system is quantified as a measure of dependability of the system in a context where its malfunction poses a risk and possibly endangers lives.

Such a situation is called a *hazard* which in accordance with Ladkin [Lad08] is characterized as a "potential source of harm". A bit more strict definition of a hazard is given by Leveson in [Lev11] (p. 157) as a "system state or set of conditions that, together with a particular set of worst-case environmental conditions, will lead to an accident" where accident is defined as leading to a specified level of loss. For the formal safety analysis presented in this dissertation, a hazard is considered to be a system state in which the system *potentially* causes harm and the system itself cannot prevent this any more. This means that a hazard is dangerous, as it may lead to an accident *in the worst case*.

A hazard can be caused by the malfunctioning of one or more components of the system under consideration. Such malfunctioning is the result of the occurrence of a *failure mode*. Not every occurrence of a failure mode must have such an effect. For example a system component might not be active at the moment of the occurrence or the failure mode might be of a transient nature. On the other hand, every malfunction of a system component is always caused by the occurrence of one or more specific failure modes.

In this dissertation, safety is measured quantitatively as the occurrence probability of a hazard. This is the probability that the system itself cannot prevent the occurrence of a potential source of harm resulting from the occurrence of failure modes in the system components. Safety is measured qualitatively as the combinations of failure modes that must occur in combination as a necessary requirement for the occurrence of a hazard.

Standards and Norms

The basis for many standards in the field of safety analysis is the IEC 61508 [Int98] standard ("Functional safety of electrical / electronic / programmable electronic safety-related systems"). In this standard, safety is considered as the "freedom from unac-

ceptable risk", where risk means the "combinations of the probability of occurrence of harm and the severity of harm" [Lad08]. The severity of the potential harm caused by hazard is not measured, but determined a-priori. When the occurrence probability of the occurrence of harm is known, it can then be checked whether the posed risk is unacceptable, i.e. the system is considered unsafe or if it is acceptable and the system is therefore considered safe.

The notion of risk as defined above is still slightly unclear. To clarify this, guidelines exist which define certain maximal occurrence probabilities for a hazard, depending on the severity of the potential harm caused. Based on this, risk is classified into three areas (see [Lad08]):

- Acceptable: So low that it can be ignored for all practical purposes.
- Intolerable: So high that is it unacceptable in all circumstances.
- ALARP region: Between acceptable and intolerable, the developer is required to reduce the risk to "as low as reasonably possible" (ALARP).

Depending on the potential severity of a hazard, maximal hazard occurrence probabilities are specified. In IEC 61508, the probabilities are defined as safety integrity levels (SIL) and the developer must provide evidence that the system fulfills the required threshold probabilities. Similar concepts exist in domain-specific standards derived from IEC 61508, such as ISO 26262 [Int09] for automotive or DO 178-B [RTC92] for avionics systems.

To which extend the required quantitative results can accurately be computed is an ongoing debate in the domain of safety analysis, in particular if software is to be quantified (e.g. see Butler and Finelli [BF93] or Alexander and Kelly [AK09]). Many of the existing methods that were developed to determine whether a system fulfills the imposed requirements for threshold probabilities, rely on assumptions like stochastic independence which are often not fulfilled.

Model-Based Safety Analysis

The main concept of *model-based* safety analysis is defined by Joshi et al. in [JMWH05] as using a common system model by both the system developer and the safety engineer. In practice the degree of sharing varies greatly and many different variations of this rather basic principle are used in safety analysis. The common factor between all is that there exists a model of the failure modes and how they may cause a system hazard.

The variants of model-based safety analysis span from a manually specified model of the failure effects to fully integrated automatic combination of both nominal and failure behavior in a single model. Here the classification is roughly as follows: *structured approaches* use an additional manually created model of the failure behavior which mainly considers the components of the system but not the structure of the system. Approaches

based on *failure logic modeling* often directly use the structure of the analyzed system to model the possible propagation of failure effects. This allows for semi-automatic deduction of possible failure effects behavior from a structural model the system. *Failure injection* based approaches model the failure effects behavior directly into the nominal system model and use various (semi-) automatic analysis techniques to compute possible causes for a system hazard. Of course for some approaches, an exact classification is not possible and they could fit in different classes.

The following section presents an overview, ranging from more traditional but widely-used approaches to new developments in the area of model-based safety analysis techniques.

2.2. Structured Approaches

Structured approaches generally consist of the manual creation of a model for safety analysis. Therefore they rely heavily on the experience and skill of the safety engineer. Furthermore, if a system changes, very often the whole analysis must be conducted again. The identification of only the relevant changes is often very difficult as the connection between the system and the safety analysis model is not clear.

2.2.1. Fault Tree Analysis

Fault tree analysis (FTA) [VDF+02, Int06] is widely used in industry for safety analysis. It is a structural approach, in which a complex system hazard is broken down in events that may lead to this hazard. Such events may be *intermediate* events which are further broken down into *basic events*. Fault tree gates connect the intermediate or basic events, resulting in a tree with the hazard at the root as the *top event* and the basic events on the leaves. The following description is adapted from [GOR08]. A simple fault tree with basic events (here failure modes) fm_i (i=1...5) and hazard H is shown in Fig. 2.1.

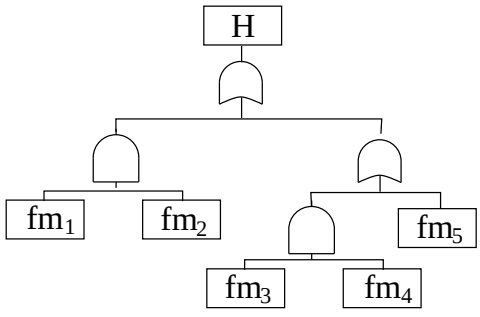


Figure 2.1.: Example Fault Tree [GOR08]

The fault tree gates which connect basic and intermediate events are most often simply the Boolean *AND* and *OR* gates as shown in Fig. 2.2. More complex variants of gates,

e.g. *INHIBIT*, also exist [STR02], but tool support for fault trees is most often limited to Boolean gates. The informal semantics of the Boolean gates is that for the *AND* gate, *all* connected events must occur, for the *OR* gate, *one* of the connected event is enough to trigger the top level event.

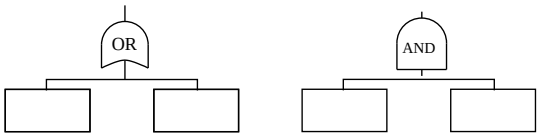


Figure 2.2.: Boolean Fault Tree Gates

When the desired accuracy in the form of basic events which are not broken further down has been reached, then *minimal cut sets* can be computed automatically from the fault tree. Every such set contains a combination of basic events which may cause the occurrence of the hazard if they appear together. This combination is *inclusion minimal* in the sense that if at least one basic event from every minimal cut set can be prevented, the hazard cannot occur. This fact is called the *minimal cut set theorem*. An example fault tree with corresponding minimal cut sets is shown in Fig. 2.3. It contains 3 minimal cut sets marked by dashed lines, two of size 2 and one with a single point of failure (minimal cut set of size one).

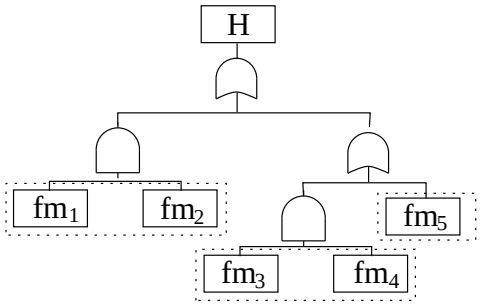


Figure 2.3.: Minimal Cut Sets [GOR08]

Using the Boolean logic semantics, it is possible to compute the minimal cut sets of very large fault trees. In practice, symbolic representations as binary decision diagrams (BDD) are used for this as proposed by Sinnamon and Andrews in [SA97]. This is generally implemented in tools supporting the modeling and construction of fault trees.

2.2.2. Failure Modes And Effects Analysis

Another structured approach which is widely used for safety analysis in industrial practice is the failure modes and effects analysis (FMEA) (e.g. see McDermott et al. [MMB96]). It basically consists of the following three steps: identification of the

failure modes, determination of the causes for the failure modes and the number of times they occur and definition of detection methods for the failure modes.

Using these concepts, a FMEA table is constructed (most often manually). In this table, the *severity* is noted for each failure mode. This ranges from 1 (no danger) to 10 (critical). The *occurrence rating* is noted from 1 (very seldom) to 10 (very often) and the *detection rate* is also noted from 1 (easy to detect) to 10 (impossible to detect). From these ratings, the risk priority number (RPN) is calculated. It is used to determine the sequence in which the failure modes must be prevented or their effect limited by risk-reducing mechanisms. The associated risk of the entries of the FMEA table with the higher RPN are considered with higher priority.

An extension of basic FMEA is the failure modes, effects and criticality analysis (FMECA) [MIL]. It extends FMEA with a notion of criticality that connects the probability of failure modes with the severity of their effects. This probability can either be given directly as a failure rate or via levels defined by upper and lower bounds similar to the safety integrity levels (SIL) of IEC 61508. Recently, Grunske et al. [GLYW05] developed an automatic way to deduce FMEA tables from a system model. This method has been extended by Grunske et al. in [GCW07] to also support the computation of hazard probabilities and can therefore also be used for criticality analysis. An application of this method to the safety analysis of an airbag system is described by Aljazzar et al. in [AFG+09].

2.2.3. Why-Because Analysis

Another structured approach to finding the cause of accidents is the *Why-Because Analysis* (WBA) described by Ladkin [Lad01]. Its main focus is to determine the causal relations between recorded events and states of an accident. The basic concept is the Why-Because Graph (WBG) in which events and states are connected if one is the *causal factor* [Lad99] of the other. Being a causal factor is formulated as: "A is a causal factor of B, in which A, respectively B, is either an event or state" if "in the nearest possible world in which A did not happen, neither did B" [Lad99] (based on Lewis [Lew73]). This means that if everything else is the same, but the state or event A being absent, B would not have happened. This is formulated in the modal logic EL [Lad01] (pp. 295-320) as the *Counterfactual Test* and a hierarchical proof scheme is applied to the WBG that can prove that the explanation of the causality in the WBG is correct [Lad01] (pp. 339-378).

Paul-Stüve defines in [PS05] the following steps for a full WBA: gathering information, determination of the facts, creation of a list of facts, creation of a why-because list, creation of an auxiliary list of facts, determination of the top node (most often the accident itself), determination of the necessary causal factors and quality assurance and correction of the WBG. Several tools to support this structured approach have been developed and WBA has been successfully applied to several real accidents, in particular aircraft accidents as described by Ladkin in [Lad00].

A big advantage of WBA over many other analysis approaches is the absence of the

"closed-world assumption"[1]. This is not needed for WBA, as the list of states and events is constructed a-posteriori from all facts that are available from an accident. On the other hand this applies only if it is applied to explain the causes of an accident that has already happened. It does not solve the problem of the closed-world assumption in a-priori safety assessment of a system.

2.2.4. System-Theoretic Analysis Model and Processes

A very interesting approach which aims at an a-posteriori explanation of accidents is the Systems-Theoretic Analysis Model and Processes (STAMP) [Lev04b, Lev04a] and STAMP based Process Analysis (STPA) [Lev03] developed by Leveson. It differs considerably from the approaches discussed before. STPA is based on a systems theoretic approach to safety-analysis and considers safety an emergent property of a system which addresses the increasing complexity of systems and the problems that purely analytic safety analysis approaches can have. Leveson argues in [Lev92] that many of the approaches based on causal consequence analysis have simply been adapted from mechanical systems, but are not well suited for modern software-intensive systems which are inherently much more complex.

The three main concepts of STAMP are: constraints, hierarchical levels of control and process models. In this approach, every system is viewed as a hierarchical structure, where each level imposes constraints on the possible behavior of the level beneath. An accident is then not viewed as a result of events in a given order, but as a result from insufficient control. The process model in STAMP is typically a process-control loop with an automated controller and a human supervisor for this controller. This model views accidents as a failure to adequately satisfy the systems goal condition, action condition, model condition or observability condition [Lev04b].

STAMP (and STPA) has successfully been applied to the different case studies, e.g. the Comair accident by Nelson [Nel08], the U.S. Ballistic Missile Defense System by Pereira [PLH06] and an unmanned space transfer vehicle by Ishimatsu [ILT+10].

2.3. Failure Logic Modeling

Another class of safety analysis approaches use an explicit modeling of the propagation of the effects of failure modes. The information about possible causes for a system hazard is then (semi-) automatically deduced from the structural model of a system. These methods are called failure logic modeling approaches in the classification of Lisagor and McDermid [LM06].

[1] A full list of important events and states is known.

2.3.1. Failure Propagation and Transformation Notation

The Failure Propagation and Transformation Notation (FPTN) by Fenelon et al. [FM93, FM92] is a graphical notation which describes the structure of a system and the generation and propagation of failure dependent on this structure. Components are connected via inputs and outputs to other components, which defines the possible propagation of failures in a system.

An enhancement of FTPN is the Failure Propagation and Transformation Calculus (FPTC) described by Wallace [Wal05] which uses a formalized approach. Structural models of systems are expressed as real-time networks (RTN). The nodes of these networks represent components of the analyzed system and analogous to basic FPTN, the possible propagation of failures is defined via inputs and outputs that connect the nodes. Different types of failures can be defined: late and early for timing failures and omission or value for data failures. Using the formal transformation rules described in [Wal05] this can then be used to analyze the potential effect of the failures. A newer extension of FTPC is the Probabilistic Failure Transformation and Propagation Analysis (PFTPA) developed by Ge et al. [GPM09]. It allows for the specification of failure occurrence probabilities of the single failure modes. The overall hazard occurrence probability of the system is then computed in an automatic way.

2.3.2. Hierarchically Performed Hazard Origin and Propagation Studies

Hierarchically Performed Hazard Origin and Propagation Studies (Hip-Hops) developed by Papadopoulos et al. [PM91, PPM99] is a safety analysis technique that allows for automatic generation of fault trees and of FMEA tables, based on a structural system model. It describes the structure of the system, in which the basic elements are the system components. Components can be connected via input and output ports which model the dataflow through the system. The failure behavior is specified as the failure of system components, failure effects can then propagate along the defined connections to other components.

An advantage over the FTPN approach is the tool support which allows for automatic generation of fault trees from such a model. Another advantage of Hip-Hops is that the hierarchical system structure is reflected in the generated fault tree, in contrast to the flat fault trees[2] often extracted from other approaches, based on failure injection (see Section 2.4). Based on the synthesis of the fault trees, an automated FMEA table generation is possible [WWGP10]. Extensions of the Hip-Hops technique allow for automatic extraction of dynamic fault tree information as described by Walker et al. [WBP07, WP07]. Dynamic fault trees are a generalization which introduces additional gates that require for example an ordering on the occurrence of the basic events.

[2]Which have only a top event, one layer of Boolean logic fault tree gates and then one layer of basic events.

The Hip-Hops approach has also successfully been used to develop an automatic, optimal allocation of safety functions for the automotive domain [PP07]. In this case the structure of the system is exploited to reach a target overall automotive safety integrity level (ASIL) of a system, by breaking it down to the necessary ASIL assignment to sub-components. In [PWP+10], Papadopoulos et al. show how this can be used to optimize the structure of a system.

2.3.3. AltaRica

AltaRica is a formal dataflow language for hierarchically structured safety-critical systems. It was described by Arnold et al. in [APGR99]. Its main design goal was the specification of the behavior of concurrent systems and it is often used to specify fault occurrence in such systems as observed in the overview of Joshi et al. [JWH05]. It allows for state based system behavior modeling of parallel system nodes. A failure is an event that can affect the state of a node. Several formal analysis tools have been developed for the analysis of AltaRica models, for example the Mec 5 model-checker by Griffault and Vincent [GV04]. Bieber describes in [BCS02] a tool for the automatic generation of fault trees from Altarica models. Current development is on the integration of different analysis methods in the Altarica Studio.

Modeling in AltaRica sometimes has the problem that the failure propagation specification is cyclic and therefore the model is invalid [BBC+04]. On the other hand it provides excellent support for the creation of a library of components that can graphically be used to construct a AltaRica model [ea03], which is seen as a great advantage in the application of AltaRica to case studies, see e.g. Bieber et al. [BBC+04]. Current research focuses on extension of model-checking capabilities in AltaRica Studio and the elimination of the limitations to acyclic models in AltaRica Next Generation by a new execution model based on fix points [PPRd10].

2.4. Failure-Injection Based Analysis Techniques

In failure-injection based safety analysis [LM06], a functional system model is constructed first on which the nominal system behavior can be verified. After then, the effects of different failure modes are successively injected into the model and the nominal behavior is tried to be verified. If this fails, the injected failure modes are considered to be critical. A system model which contains the modeling of the effects of failure modes is commonly called the *extended system model* by Ortmeier et al. [ORS06] and Joshi et al. [JMWH05, JH07]. The combination of the functional behavior and the failure model into the extended system model can either be done manually or automatically.

2.4.1. ESACS and ISAAC Project

In the Enhanced safety assessment for complex systems (ESACS) project [ea03], the FSAP / NuSMV-SA tool was developed by Bozzano et. al [BV03b]. It contains a library of possible failure modes (bit-inversion, bit-stuck) for Boolean state variables. The failure model can then automatically be integrated into a nominal system model. This integration is done on the syntactical level, the resulting models are analyzed with the NuSMV model-checking tool developed by Cimatti et al. [CCG+02]. It computes flat fault trees and can extract some temporal ordering information from the system models [BV03a]. The flat fault trees are used to estimate the overall system hazard probability based on the assumption of stochastic independence of the failure mode occurrence.

In the ISAAC project [rBB+06] the successor of ESACS, two main approaches for safety analysis were implemented. The first one was he further development of the FSAP / NuSMV-SA tools. The other main approach was to use failure injection in SCADE models as described by Abdulla et al. [ADS+04]. In contrast to FSAP / NuSMV-SA, the failure models were integrated manually into the system model and the extended system model was analyzed with the SCADE Design Verifier. A similar approach is described by Joshi and Heimdahl in [JH05, JH07] where Matlab / Simulink models were analyzed. The concrete analysis was conducted in SCADE (or Lustre) via the Simulink import function of the tool. This method allows to find counterexamples if a safety property was not fulfilled, but analogous to our own work in [GOR07], verification was not possible. The formal analysis tool built into SCADE is not very efficient if compared to other state of the art tools as it is also observed by Moy in his dissertation [Moy05] (p. 142).

2.4.2. COMPASS Project

In the Correctness, Modeling and Performance of Aerospace Systems (COMPASS) research project [BCK+09a], a subset of the architecture analysis and design language (AADL) [SA04, GH08] and its error annex [SA06] was formalized in the SLIM language as described by Bozzano et al. in [BCK+09b, BCR+09]. This allows for the combination of continuous state variables and probabilistic failure mode specifications. Continuous data is supported via the SMT solver MathSAT developed by Bruttomesso et al. [BCF+08]. Probabilistic models are analyzed with the probabilistic model-checker MRMC developed by Katoen et al. [KKZ05, KZH+10]. The already existing FSAP / NuSMV-SA tool which was developed in the ESACS and ISAAC project was further extended and now allows for the specification and automatic inclusion of probabilistic failure modeling. In [Boz11], Bozzano showed how this also allows for calculation of FMEA tables directly from SLIM models for safety analysis.

The approach has successfully been applied to industrial case studies from the aerospace domain [BCK+09a]. The developed tool [BCK+10a] allows for the specification of differ-

ent kinds of failure mode effects, similar to the library offered in the FSAP / NuSMV-SA tool, but extended with probabilistic behavior. System properties are specified via patterns which aim at enabling users who are not experienced with temporal logic. This is similar to the safety properties specification described by Bitsch [Bit01] and quantitative safety properties specification described by Grunske [Gru08].

2.4.3. AVACS Project

In the automatic verification and analysis of complex systems (AVACS) project [AVA03], different approaches for the analysis of critical systems are developed. The main focus is the development of new verification algorithms that can analyze a broader range of systems.

This includes mainly hybrid systems which allow for using continuous data variables. In this project, first steps for the verification of probabilistic hybrid systems have been made by Zhang et al. [ZSR+10]. In contrast to the safety analysis in COMPASS, this would allow a probabilistic analysis of model with hybrid variables. Other topics include the usage of symbolic methods for probabilistic model-checking which allows for the analysis of even larger systems as described by Herbstritt et al. [HWP+06]. The application of this technique has been used by Böde et al. [BPRW08a, BPRW08b] to rank critical failure combinations according to their relative influence of the overall hazard probability of a safety-critical system. In general the modeling is conducted in variants of Statemate Statecharts [BDW00, HN96] and different tools for state space reduction and bisimulation are applied on the resulting labeled transition systems.

2.5. Formal Model-Based Safety Analysis

The proposed model-based safety analysis approach described in this dissertation is based on failure-injection. The considered extended system models include both nominal and failure behavior. Critical failure mode combinations can be computed automatically and in contrast to the other approaches the result is provably correct and complete. The overall hazard occurrence probabilities are also computed automatically. An advantage over most of the related model-based approaches is that probabilistic failure mode behavior is supported for different types of failure modes, in particular per-time and per-demand. To my knowledge the only other approach that directly supports these different failure mode types are Bouissou's Boolean Logic Driven Markov Processes [Bou07, BB03], which are mainly a failure logic modeling approach.

The foundations are the results described in Ortmeier's dissertation [Ort06]. Ortmeier describes a formal safety analysis based on the formal analysis of Statemate Statecharts, expressed as Kripke structures in the specification language of the Cadence SMV [McM90] model-checker. It is a purely qualitative method which allows for the automatic computation of (inclusion) minimal sets of failure modes that can cause a

hazard. This is done with the deductive cause-consequence analysis (DCCA) (see Section 5.2.1).

These qualitative analysis results are used to optimize the system under consideration. This uses an orthogonal, simple mathematical model of the system, based on probabilistic distribution functions as optimization goal. These distribution functions have application-specific free parameters, the variation of these parameters allows for balancing a system between its safety goals (an upper bound for the hazard probability) and other objectives, like cost or delay times, this safety optimization approach is described by Ortmeier et al. in [OSR04, OR04].

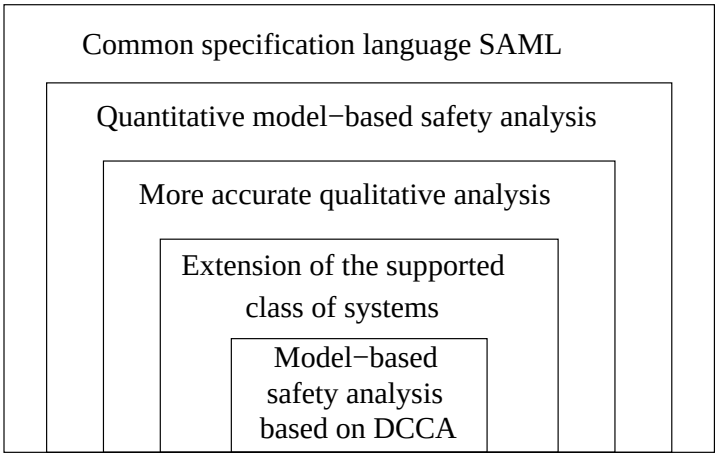


Figure 2.4.: Extension of Existing Safety Analysis Approach

The safety analysis approach presented in this dissertation extends the existing one in several ways (see Fig. 2.4). It renders the qualitative analyses much more accurate, allows for the analysis of a broader range of systems and enables quantitative analyses directly on the system model. More detailed, the extensions are the following:

- Definition of a formal specification language (SAML) which allows for the specification of both qualitative and quantitative system behavior.
- Deductive Failure Ordering analysis allows for the automatic deduction of temporal dependencies of critical failure mode combinations.
- Formal safety analysis of self-healing systems which can recover after a temporal loss of function.
- Accurate probabilistic failure mode and physical environment modeling based on the formal specification language SAML.
- Model-based quantitative safety analysis that allows for direct computation of hazard probabilities on the system model.

- Concrete implementation of model transformations of SAML models into the specification languages of different formal analysis tools.

Because of the implemented model transformations, the approach is tool-independent and therefore benefits from any increased efficiency of the already supported analysis tools. In addition, the integration of new analysis tools, e.g. those developed in the AVACS project, is possible by implementing the corresponding model transformations. SAML is designed to be well suited for the actual model specification as it offers a convenient textual representation of both qualitative and quantitative behavior of a system. The increased accuracy of the computed hazard probabilities also make it an ideal candidate in a completely model-based safety optimization approach as proposed for the ProMoSA project [OG10], replacing the approximations via distribution functions previously developed by Ortmeier et al. for safety optimization [OSR04, OR04].

Summary

This chapter presented the definition of the basic concepts of safety, hazard and failure mode as used in this dissertation. It gave an overview of different existing approaches for safety analysis, namely purely structured approaches, failure propagation based approaches and failure-injection based approaches. It outlined the benefits of the approach proposed in this dissertation which greatly extends the existing qualitative safety analysis approach based on DCCA and allows for much more precise quantitative analyses than possible before.

3. Formal Basics

Make things as simple as possible, but not simpler.

(Albert Einstein)

This chapter introduces the formal foundations of the qualitative and quantitative formal model-based safety analysis. At first the syntax of the Safety Analysis and Modeling Language (SAML) is defined which serves as modeling framework. Its formal semantics is based on Markov chain models from probability theory.

Properties of SAML models are formulated as temporal logic proof obligations. The semantics of different temporal logics are presented, qualitative properties are specified in computational tree logic (CTL) and linear time logic (LTL), quantitative properties in a probabilistic temporal logic (PCTL).

The motivations for the design of SAML are explained in Section 3.1. Its syntax is presented in Section 3.2 together with a small example model. Section 3.3 describes the semantics of the underlying formal models by providing its formal definition and also an illustration of the semantics of the introduced example. In Section 3.4 temporal logics for both qualitative and probabilistic assertions are introduced which are used in later chapters to specify the various formal analyses. Section 3.5 introduces a convenient graphical representation of SAML modules. Related work is discussed in Section 3.6.

3.1. Motivation

A specification framework for the analysis of safety-critical systems has several strong requirements. The most important aspects are the following:

1. Feasibility of formal analysis by using formal semantics
2. Efficient transformation of models into the specification frameworks of existing state of the art formal analysis tools
3. Convenient system modeling directly in the framework
4. Expressibility of probabilistic *and* non-deterministic behavior

Having formal semantics and being able to transform the language into existing formal analysis tools are somehow antagonistic to the requirement that direct modeling is convenient. On the one hand, if too many high level modeling artifacts are introduced, their expression as a formal model can get very complicated and lead to state explosion which makes formal analysis unfeasible for all but very simple models. On the other hand, it is often very difficult to create models in a specification language that is very close to the analysis model of a tool, comparable to programming in assembly language.

Therefore a compromise between convenient modeling and formal aspects is needed. SAML allows for the specification via parallel finite state automata. These can easily be visualized which is very useful to visualize models and to analyze analysis results. A formal semantics is defined by specifying a parallel composition of the finite state automata and the semantics of the resulting product automaton. Finite state automata are also the basis for the analysis model for most state of the art analysis tools. This allows for the efficient transformation of SAML models into models suitable for these tools.

For accurate quantitative model-based safety *analysis*, a possibility to model probabilistic behavior as well as non-deterministic behavior is necessary. Not only the isolated correctness of a modeled system itself is of interest, but its behavior in a real environment. This environment includes failing components as well as non-exact or unforeseen physical behavior. A lot of this type of behavior can be described in concepts of probabilities, e.g. as failure rates or via probabilistic distributions. On the other hand, if the correct probabilities are not known or a behavior is inherently not probabilistic, then non-deterministic modeling is often more adequate. Therefore both non-deterministic and probabilistic modeling is possible in SAML and an explicit timing model is specified for accurate environment modeling.

Its design allows SAML to be both a modeling language in which safety-critical systems and their environment can be expressed and analyzed, as well as a potential intermediate language. SAML is tool-independent and supports the transformation of models into input specifications of different formal analysis tools. Therefore any language that can be transformed into SAML benefits from this, as its models can also

be analyzed with all tools supported in the SAML framework. SAML has been first presented in [GO10a]. Transformations for widely used verification tools is explained in Chapter 6. In the outlook in Chapter 8, some further ideas for language extensions built on top of SAML will be discussed.

3.2. Syntax of the Formal Models

The most important aspects of the grammar for SAML models is shown in Extended Backus Naur Form (EBNF) in Fig. 3.1. Keywords and symbols are presented in bold font, symbols are enclosed between ‘ for better visibility. The keywords are: **constant, formula, int, double, bool, true, false, module, endmodule, init and choice**. IDENT is a general lexer rule for identifiers, DOUBLE and INT are lexer ruler for double, respectively integral numbers.

Lexer rules start in uppercase, parser rules in lowercase. The actual implementation of SAML is realized as a grammar specification for the ANTLR parser generator developed by Parr [Par07] with Java as target language.

SAML Language Concepts

The syntax of SAML is basically an extended subset of the input language of the PRISM model-checker developed by Kwiatkowska et al. [KNP02b]. The most prominent differences are the omission of explicit synchronization labels and the explicitness of non-deterministic choices with the `choice` keyword. The explicit indication of non-determinism has the advantage that it is less likely to be missed when reading such a model specification. It also prevents some modeling which is not in accordance with the proposed modeling approach for safety critical systems presented in the next chapter.

SAML Model A SAML model consists of various definitions of constants and formulas and of one or more SAML modules. These modules represent finite state automata that are executed in a synchronously parallel manner.

Constants A constant has an associated name, the identifier, a type and an optional value. The values can either specified directly or as an arithmetic expression. The type of a constant is either *double*, *int* or *bool*.

Value A value is either an integral value, a Boolean value or a floating point value. Floating point values can be expressed in scientific notation.

Formulas A formula is comprised of its name and a Boolean *condition*. It is generally used to specify an abbreviation of a more complex Boolean expression.

```
saml-model :  (constant | formula)* module+ ;
constant :  constant (int | double | bool) IDENT (':=' value)?  ;
formula :  formula IDENT ':=' condition ;

condition :  ( condition )
             | '!' condition
             | condition '&' condition
             | condition '|' condition
             | predicate ;

predicate :  IDENT ('=' |'<' |'>' |'>=' |'<=') state_expr
             | IDENT | true | false;

module :  module IDENT declaration+ update_rule+ endmodule;
declaration :  IDENT ':' '[' INT '..' INT ']' init INT ;

update_rule :  condition '->' non-det_assigns
               | prob_assigns ;

non-det_assigns :  non-det_assign
                   ('+' non-det_assign)* ;

non-det_assign :  choice '(' prob_assigns ')' ;
prob_assigns :  prob_assign ('+' prob_assign)* ;

prob_assign :  probability :  nextstate_assign
                              ('&' nextstate_assign)* ;

probability :  IDENT | DOUBLE | arith_expr;
nextstate_assign :  '(' IDENT"'" '=' state_expr ')' ;

state_expr :  IDENT | INT | ( state_expr )
              | state_expr ('+' | '-') state_expr ;

arith_expr :  INT | DOUBLE | IDENT | '(' arith_expr ')'
              | arith_expr ('+' | '-' | '/' | '*') arith_expr
```

Figure 3.1.: Basic SAML Syntax

Condition A condition is a Boolean expression in propositional logic. It is comprised of negation, conjunction or disjunction of conditions or predicates.

Predicate A predicate is a Boolean truth value or a predicate of expressions over the state variables or a reference to a formula identifier.

Modules Each module declaration has a name and is contained between the keywords `module` and `endmodule`. Within a module at least one state variable is declared and at least one update rule is specified.

Declaration State variables have an associated identifier as name, are of integral type, have a single initial value and represent a range with lower and upper bound.

Update Rules An update rule is comprised of a Boolean *activation condition* and has at least one non-deterministic choice for assignments or a single probabilistic assignment. Each non-deterministic choice is characterized by the `choice` keyword. If several such non-deterministic choices are specified, they are written as a sum.

Non-Deterministic Assignment Each non-deterministic assignment consists of one or more probabilistic assignments.

Probabilistic Assignment Each single probabilistic assignment starts with its probability. This probability can be given directly as value of type double, as a floating point constant or as an arithmetic expression. The probability is then followed by parallel assignments of new values to state variables.

Parallel Assignments Each assignment of a new value refers to the name of the state variable and a state expression. The assignments are separated from each other with `&`. For a state variable `v`, the name for the next value is written as `v'`. A state expression defines this new value `v'` which can be either an integral value, the name of a constant or an arithmetic expression. It is required, that in each parallel assignment all variables of the respective module get a new value assigned.

State Expressions State expressions are generally an expression over the value of the states. As states are of integral value, state expressions allow to specify the sum or difference of the value of states. They can also consist directly of an identifier referring to a state variable, an integral constant or directly an integral value.

Arithmetic Expressions An arithmetic expression can be the sum, difference, quotient or product of integral values or doubles, specified as constants or directly via values. The interpretation of sum, difference, quotient and product is the standard one.

Example SAML Model

Fig. 3.2 shows a small SAML example model which is used to illustrate the different available modeling artifacts. It consists of two modules: `a` and `b`. In the first part of the model, several constants of type double are declared directly with a specified value of type double. The keyword `formula` declares an abbreviation of a Boolean formula, here the name `case3` refers to the condition `v_a = 0 & !(v_b1 = 0 & v_b2 = 0 | v_b1 = 1 & v_b2 = 1)` and can be used interchangeably anywhere in the model.

```
constant double p_a := 0.9;
constant double p_b1 := 0.9;
constant double p_b2 := 0.09;
constant double p_b3 := 0.01;

formula case3 := v_a = 0 & !(v_b1 = 0 & v_b2 = 0 | v_b1 = 1 & v_b2 = 1);

module a
       v_a : [0..2] init 0;

       v_a = 0 & v_b1 = 0 & v_b2 = 0 ->
               choice (p_a : (v_a' = 0) + (1 - p_a) : (v_a' = 1));
       v_a = 0 & v_b1 = 1 & v_b2 = 1 -> choice (1 : (v_a' = 2));
       case3 -> choice (1 : (v_a' = 1));
       v_a = 1 -> choice (1 : (v_a' = 1));
       v_a = 2 -> choice (1 : (v_a' = 2));
endmodule

module b
       v_b1 : [0..1] init 0;
       v_b2 : [0..1] init 0;

       true -> choice:(p_b1 : (v_b1' = 0) & (v_b2' = 0) +
                       p_b2 : (v_b1' = 1) & (v_b2' = 0) +
                       p_b3 : (v_b1' = 1) & (v_b2' = 1))
             + choice:(   1 : (v_b1' = 1) & (v_b2' = 1));
endmodule
```

Figure 3.2.: Example SAML Model

Module `a` contains one state variable named `v_a` with a domain ranging from 0 to 2 and an initial value of 0. The module contains 5 update rules. The first update rule has a single probabilistic distribution, where the next value of `v_a` is 0 with probability p_a and 1 with probability $1 - p_a$. The other update rules assign the new value for `v_a` each with probability 1.

Module `b` contains two state variables, `v_b1` and `v_b2` both with a range from 0 to 1 and an initial value of 0. The module contains exactly one update rule with the activation condition *true*, i.e. which is always active. It consists of a non-deterministic

choice between two different probabilistic distributions, the first assigns the values 0 to both `v_b1` and `v_b2` with probability p_{b1}, 1 to `v_b1` and 0 to `v_b2` with probability p_{b2} and 1 to both with p_{b3}. The second non-deterministic choice assigns 1 to both state variables with probability 1.

The example is used only for illustration purpose and is deliberately modeled in a very abstract way. It is large enough to exhibit most SAML modeling artifacts but still small enough to construct its whole state-space manually. In Chapter 4, detailed modeling guidelines for SAML models for safety analysis are explained with a larger example.

In Table 3.1 the composition of the update rule of the module `b` from the example model is shown. The whole update rule consists of the *activation condition* and two *non-deterministic choices*. The non-deterministic choices consist of *probability distributions* and each probability distribution consists of one or more *probabilistic assignment*. Each such probabilistic assignment consists of a *probability* and *parallel assignments* and parallel assignments finally consist of one or more *single assignments*.

Update Rule									
Activation Condition Condition		Non-Deterministic Choice							
			Probabilistic Assignment						
			Probability		Parallel Assignments				
					Single Assignment				
`true`	`->`	`choice : (`							
			`p_b1`	`:`	`(v_b1' = 0)`	`&`	`(v_b2' = 0) +`	}	Probability Distribution
			`p_b2`	`:`	`(v_b1' = 1)`	`&`	`(v_b2' = 0) +`		
			`p_b3`	`:`	`(v_b1' = 1)`	`&`	`(v_b2' = 1)`		
		`)`							
	`+`	`choice : (`	`1`	`:`	`(v_b1' = 1)`	`&`	`(v_b2' = 1));`		

Table 3.1.: Composition of an Update Rule

3.3. Semantics of the Formal Models

The semantics of SAML is defined for a single module. As a SAML model typically consists of several modules, a *syntactical* parallel composition which constructs a single module from several parallel ones is specified. A very important factor for SAML was the ability to express both quantitative and qualitative aspects. Therefore, the semantics for both interpretations of SAML models is given. Firstly the more general quantitative semantics is described and afterwards the qualitative one. Finally the connection between these two is discussed.

3.3.1. Parallel Composition

The parallel composition transforms a SAML model with several modules into an equivalent model with a single module. The semantics of a SAML model is defined in Section 3.3.2 based on this single module. For the parallel composition of two modules, the parallel composition operator || is used analogously to the synchronous parallel composition described by Norman et al. in [NPK10].

Every update rule of a SAML model contains parallel assignments of new values to the state variables. For a state variable v, v' denotes its value for the next time step. The assigned value is an expression over the other state variables, constants and integers. The common interpretation of addition and subtraction for $+$ and $-$ is used. Parallel assignments to several state variables are denoted by the operator & in the form:

$$v'_1 = expr_1^{Var} \& \cdots \& v'_n = expr_n^{Var}$$

Each possible variable assignment has an corresponding probability p and is of the form:

$$p : \left(v'_1 = expr_1^{Var} \& \cdots \& v'_n = expr_n^{Var}\right)$$

Several of these parallel variable assignments form a discrete probability distribution, which requires that $\sum_i p_i = 1$:

$$\begin{aligned} p_1 : (v'_1 = expr_{11} \& v'_2 = expr_{12} \& \cdots \& v'_m = expr_{1m}) + \\ \cdots \\ p_n : (v'_1 = expr_{n1} \& v'_2 = expr_{n2} \& \cdots v'_m = expr_{nm})) \end{aligned}$$

Each update rule of a SAML model is comprised of one or more non-deterministic choices, where each such choice corresponds to exactly one of these discrete probability distributions of transitions. Together with the Boolean activation condition ϕ_i, the general form of an update rule is as follows:

$$\begin{aligned} \phi_i \rightarrow \quad & \text{choice}_1^i : (\quad p_{11}^i : (v'_1 = expr_{111} \& v'_2 = expr_{112} \& \cdots \& v'_m = expr_{11m}) + \cdots \\ & \qquad\qquad p_{1n}^i : (v'_1 = expr_{1n1} \& v'_2 = expr_{1n2} \& \cdots v'_m = expr_{1nm})) \\ & \qquad\qquad \vdots \\ + \quad & \text{choice}_k^i : (\quad p_{k1}^i : (v'_1 = expr_{k11} \& v'_2 = expr_{k12} \& \cdots \& v'_m = expr_{k1m}) + \cdots \\ & \qquad\qquad p_{kn}^i : (v'_1 = expr_{kn1} \& v'_2 = expr_{kn2} \& \cdots v'_m = expr_{knm})) \end{aligned} \tag{3.1}$$

For simplicity it is assumed that all the requirements for a proper specification are fulfilled. This means that the probabilities specify correct probability distributions and

that the variables are uniquely named, i.e. there are no modules that contain a variable with the same name.

This notation is chosen to simplify the usage of update rules in the parallel composition definition. The single update rule of the module `b` (see table 3.1) of the example above would be written in the following way:

$$\begin{aligned} true \rightarrow \quad & \text{choice} : (\quad p_{b1} : (v'_{b1} = 0 \& v'_{b2} = 0) + \\ & \qquad\qquad\quad p_{b2} : (v'_{b1} = 1 \& v'_{b2} = 0) + \\ & \qquad\qquad\quad p_{b3} : (v'_{b1} = 1 \& v'_{b2} = 1)) \\ + \quad & \text{choice} : (\quad 1 : (v'_{b1} = 1 \& v'_{b2} = 1)) \end{aligned}$$

Using this notation, update rules can be written using the mathematical notation for sums and the parallel composition of two SAML modules with the operator || can be defined as in Def. 1 which is adapted from Kwiatkowska et al. [NPK10]. For two modules M_i and M_j, each probability distribution of each of the non-deterministic choices of the first module (Eq. (3.2)) is combined with each probability distribution of each of the non-deterministic choices of the second module (Eq. (3.3)). The result is an update rule with an activation condition which is the conjunction of the activation conditions of the original update rules. Each of these choices is comprised of a new probability distribution and parallel assignments of the state variables.

Definition 1. Parallel Composition of SAML Modules

For $M = M_i || M_j$ set $stateVars(M) := stateVars(M_i) \cup stateVars(M_j)$ and create for each update rule

$$\phi_i \rightarrow \sum_{k=1}^{c} choice_k^i \left(\sum_{l=1}^{d_k} p_{kl}^i : u_{kl}^i \right) \tag{3.2}$$

of M_i and

$$\phi_j \rightarrow \sum_{m=1}^{e} choice_m^j \left(\sum_{n=1}^{f_m} p_{mn}^j : u_{mn}^j \right) \tag{3.3}$$

of M_j a product update rule for M of the form

$$\phi_i \wedge \phi_j \rightarrow \sum_{k=1}^{c} \sum_{m=1}^{e} choice \left(\sum_{l=1}^{d_k} \sum_{n=1}^{f_m} \left(p_{kl}^i \cdot p_{mn}^j : u_{kl}^i \& u_{mn}^j \right) \right)$$

This parallel composition is well-defined and unique. The variable ordering in the parallel assignments does not change the semantics of the result. Reordering of variables in the parallel assignments or in the definition of state variables creates an isomorphic state space. Therefore the semantics of the underlying Markov chain model is invariant

under ordering of state variable definitions or parallel assignments[1]. If a SAML model has more than two modules, the parallel composition is conducted iteratively. A *complete* SAML model is defined as the parallel composition of its modules (Def. 2).

Definition 2. Complete Model
For a SAML model which consists of the single modules $M_1, \ldots M_n$*, the complete model* M *is defined as* $M := M_1 || \cdots || M_n$.

The complete model generated from the parallel composition `a_b` := `a`||`b` of the two modules of the example of Fig. 3.2 is shown in Fig. 3.3. The complete model is comprised of 5 updates rules, resulting from 5 update rules of module `a` and 1 update rule from module `b`. Each update rule of the complete model consists of two choices, resulting from the 2 choices of module `b` and 1 choice of module `a`. In the first update rule of the product module `a_b`, the first choice has 6 possible transitions, resulting from the product of 2 possibilities in the first choice of module `a` and 3 possibilities in the first choice of module `b`. The second update rule of `a_b` has 2 possible transitions, resulting from the 2 possibilities of module `a` and only 1 possibility of module `b`. The rest of the update rules are constructed analogously.

From the length and complexity of the complete model resulting from the parallel composition of the rather simple modules of the originating SAML model, it becomes clear that direct specification of the complete module is not practical. Therefore in general multiple parallel modules are specified in SAML and the construction of the single module is left to the specific analysis tool.

The informal semantics of a SAML model is as follows: All the modules of the original model are executed synchronously in parallel. At each time step, all activation conditions of each module are evaluated. The models are constructed in such a way that there is always exactly one active update rule per module per time-step. In every module, one probability distribution of the active update rule is chosen non-deterministically. This probability distribution is a set of assignments of new values for all the state variables of the module. One of these assignments is then chosen probabilistically and the state variables get new values assigned in parallel. The parallel composition retains this behavior, but combines all the modules into one single module in which the update rules are the enumeration of all possible combinations of updates rules of the original model.

3.3.2. Quantitative Formal Models

The basis for the formal semantics of SAML is a variant of Markov chains which can express both probabilistic and non-deterministic behavior. Markov chain models are often used for modeling probabilistic behavior. Different variants exist which are appropriate for different modeling applications. Markov models basically describe a system as discrete states with probabilistic transitions between these states.

[1] The internal representation of the state space often uses variable reordering to achieve a more compact representation.

```
module a_b
       v_a : [0..2] init 0;
       v_b1 : [0..1] init 0;
       v_b2 : [0..1] init 0;

       true & v_a = 0 & v_b1 = 0 & v_b2 = 0 ->
              choice (p_b1 * p_a : (v_a' = 0) & (v_b1' = 0) & (v_b2' = 0)
                    + p_b1 * (1 - p_a) : (v_a' = 1) & (v_b1' = 0) & (v_b2' = 0)
                    + p_b2 * p_a : (v_a' = 0) & (v_b1' = 1) & (v_b2' = 0)
                    + p_b2 * (1 - p_a) : (v_a' = 1) & (v_b1' = 1) & (v_b2' = 0)
                    + p_b3 * p_a : (v_a' = 0) & (v_b1' = 1) & (v_b2' = 1)
                    + p_b3 * (1 - p_a) : (v_a' = 1) & (v_b1' = 1) & (v_b2' = 1))
            + choice (1 * p_a : (v_a' = 0) & (v_b1' = 1) & (v_b2' = 1)
                    + 1 * (1 - p_a) : (v_a' = 1) & (v_b1' = 1) & (v_b2' = 1));

       true & v_a = 0 & v_b1 = 1 & v_b2 = 1 ->
              choice (p_b1 * 1 : (v_a' = 2) & (v_b1' = 0) & (v_b2' = 0)
                    + p_b2 * 1 : (v_a' = 2) & (v_b1' = 1) & (v_b2' = 0)
                    + p_b3 * 1 : (v_a' = 2) & (v_b1' = 1) & (v_b2' = 1))
            + choice (1 * 1 : (v_a' = 2) & (v_b1' = 1) & (v_vb2' = 1));

       true & v_a = 0 & !(v_b1 = 0 & v_b2 = 0 | v_b1 = 1 & v_b2 = 1) ->
              choice (p_b1 * 1 : (v_a' = 1) & (v_b1' = 0) & (v_b2' = 0)
                    + p_b2 * 1 : (v_a' = 1) & (v_b1' = 1) & (v_b2' = 0)
                    + p_b3 * 1 : (v_a' = 1) & (v_b1' = 1) & (v_b2' = 1))
            + choice (1 * 1 : (v_a' = 2) & (v_b1' = 1) & (v_vb2' = 1));

       true & v_a = 1 ->
              choice (p_b1 * 1 : (v_a' = 1) & (v_b1' = 0) & (v_b2' = 0)
                    + p_b2 * 1 : (v_a' = 1) & (v_b1' = 1) & (v_b2' = 0)
                    + p_b3 * 1 : (v_a' = 1) & (v_b1' = 1) & (v_b2' = 1))
            + choice (1 * 1 : (v_a' = 1) & (v_b1' = 1) & (v_vb2' = 1));

       true & v_a = 2 ->
              choice (p_b1 * 1 : (v_a' = 2) & (v_b1' = 0) & (v_b2' = 0)
                    + p_b2 * 1 : (v_a' = 2) & (v_b1' = 1) & (v_b2' = 0)
                    + p_b3 * 1 : (v_a' = 2) & (v_b1' = 1) & (v_b2' = 1))
            + choice (1 * 1 : (v_a' = 2) & (v_b1' = 1) & (v_vb2' = 1));
endmodule
```

Figure 3.3.: Parallel Composition of the SAML Model of Fig. 3.2

The most important difference between different Markov models is their model of passing time. It is either modeled as a continuous real-valued variable or as a discrete fixed amount of time. In the continuous model, transitions between two states have a parameter[2], which describes the mean time in which the system changes from the current state to the state the transition leads to. In the discrete model, after every fixed amount of time, there is a change of state and the transition parameter describes the probability that a specific transition is taken. The sum of these transition parameters must always be 1, in order to form a proper discrete probability distribution.

Hermanns et al. note in [HKMKS00] that continuous time Markov models are well suited to describe asynchronous, interleaved system behavior, whereas discrete time Markov models are better suited to describe parallel synchronous behavior. Many safety-critical systems are developed using clocked systems or processing units with synchronous parallel components[3]. Therefore the underlying formal model for SAML is a Markov decision process (MDP) which is a discrete-time Markov chain model with additional non-deterministic behavior. The following definitions (3)-(13) for the formal definitions of MDPs are based on the descriptions by Hansson and Jonsson [HJ94], Ciesinski and Grösser [CG04] and de Alfaro et al. [dAFH$^+$05].

Definition 3. Markov Decision Process
A Markov Decision Process (MDP) *is a tuple*

$$\tau_{MDP} = (S, \mathbf{Steps}, AP, L, s_0)$$

where

- S *is a finite set of states*
- $\mathbf{Steps} : S \to 2^{Idx \times Dist(S)}$ *is the transition probability function, where* Idx *is a set of indices and* $Dist(S)$ *is the set of discrete probability distributions over the set* S*, i.e. each pair* $(i, p) \in Idx \times Dist(S)$ *defines a discrete probability distribution* $p : S \times S \to [0, 1]$ *such that*
  - $\forall s, t \in S : p(s, t) \geq 0$
  - $\forall s \in S : \Sigma_{t \in S} p(s, t) = 1$
- AP *is a finite set of atomic propositions*
- L *is a labeling function* $L : S \to 2^{AP}$ *that labels each state* $s \in S$ *with a subset of the atomic propositions that hold in this state*
- $s_0 \in S$ *is the initial state*

[2]This corresponds to the parameter λ of an exponential probability density function.
[3]For example SCADE Suite which is based on the synchronous data-flow language LUSTRE developed by Halbwachs [HCRP91] is widely used in industry for the development of safety-critical systems.

A MDP as in Def. 3 represents a labeled transition system with a finite set of states. Each state $s \in S$ corresponds to a valuation of all state variables of a *single* SAML module, in general the complete model of parallel SAML modules . The initial valuation of the state variables in the model correspond to the initial state s_o of the MDP. Each state s is labeled with the set of atomic propositions (AP) that hold in s by the labeling function L. By the definition of L, all $\psi \in AP : \psi \notin L(s)$ do not hold in s. In SAML, these atomic propositions correspond to the set of `conditions`, i.e. are Boolean expressions over the values of the state variables of the model.

The update rules of the SAML model describe the possible transitions of the MDP. Each activation condition ϕ describes a subset $S' \subseteq S$ of all states such that for each state $s \in S'$, its label $L(s)$ implies ϕ. For each state s, $\mathbf{Steps}(s)$ maps s to a set of pairs of indices and discrete probability distributions. These describe the probability of reaching a successor state s' from s, where s' corresponds to a new valuation of the state variables of the model. The index specifies which probability distribution is chosen. The probability distribution specifies the probability to reach successor states.

The state space of the parallel composition of the example in Fig. 3.2 is shown in Fig. 3.4. Each node corresponds to a state of the MDP. Each state represents a valuation of the state variables, written inside the nodes as a vector of the form `(v_a,v_b1,v_b2)`. There is an edge from state s to s' if there exists an index and probability distribution pair (i, p) in $\mathbf{Steps}(s)$, such that $p(s, s') > 0$, i.e. the state s' is reachable from state s with non-zero probability.

The edges represent these probability distributions. They are labeled in the form $j : p_j$, where j is the index of the possible probability distributions in $\mathbf{Steps}(s)$ and p_j is the probability to reach the state s' from s if the probability distribution p is chosen, i.e. $p_j = p(s, s')$.

The paths of the MDP are all infinite sequences of states (starting from the initial state `(0,0,0)`) and probabilities such that for each successor state s' of a state s there exists a directed edge from s to s' in the graph representation of the state space.

This describes the semantics of a SAML module as a MDP. In order to be interpretable in such a way, a SAML module must fulfill the following constraints on the activation conditions and probabilities of the update rules:

1. $\sum_i p_i = 1$ for each probabilistic assignment and $p_i > 0$ for each probability
2. $\bigvee_i \phi_i \equiv true$ for all activation conditions ϕ_i
3. $\forall i \neq j : \phi_i \wedge \phi_j \equiv false$ for each pair of activation conditions

The first constraint states that every probabilistic assignment must define a proper discrete probability distribution, by demanding that the probabilities to reach a new state sum to 1. The second constraint specifies that there is always an activation condition that holds. This means, in every state s there exists at least one element in $\mathbf{Steps}(s)$. These two constraints assure that there is always a successor state that can

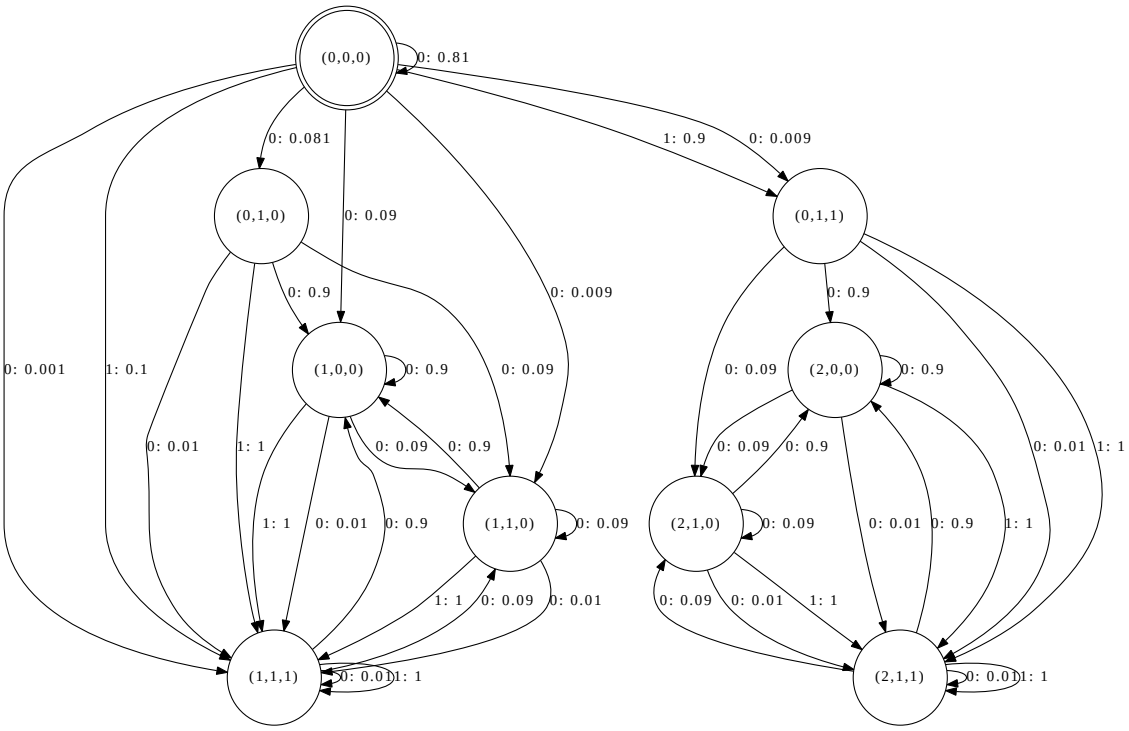


Figure 3.4.: State Space and Transitions of the Complete Model of Fig. 3.3

be reached with a positive probability. A proper SAML model therefore has no deadlock state[4]. The third constraint assures, that there are no overlapping activation conditions and in every state exactly one activation condition holds.

If overlapping activation conditions are allowed, normalization of the probability distributions would be required in order to retain a proper MDP (see [NPK10]). This easily leads to unintended behavior of the model, as the specified probabilities are different from those in the analyzed model. Such a behavior is not desired in the context of safety analysis.

A run of a MDP is a sequence of states that are reached, starting from the initial state. This sequence together with the non-deterministically chosen probability distributions and the probabilities to reach a successor states is called a path of the MDP (see Def. 4).

Definition 4. Path of a MDP

A finite or infinite path of a MDP is a sequence of states and pairs of indices and probability distributions such that for a sequence $s_0(j_0, p_0)s_1(j_1, p_1)\ldots$

$$(j_i, p_i) \in Steps(s_i) \text{ and } \forall i \geq 0 : p_i(s_i, s_{i+1}) > 0$$

holds.

[4]Of course the successor state to s can be s itself. A deadlock would be no successor at all.

When all states of a path are labeled with their labeling function, the resulting sequence is called a trace of the MDP (see. Def. 5). Properties of the MDP are most often formulated based on the set of possible traces.

Definition 5. Trace of a MDP
A trace σ *of a path* $\omega = s_0(j_0, p_0)s_1(j_1, p_1)\ldots$ *of a MDP Model* τ_{MDP} *is the word over the alphabet* 2^{AP}*, generated from*

$$\sigma(s_0(j_0, p_0)s_1(j_1, p_1)\ldots) = L(s_0)L(s_1)\cdots$$

In a path of a MDP, every state s is followed by a pair (j, p). This describes an index j and a probability p. Here j is the index of the non-deterministically chosen probability distribution P of $\mathbf{Steps}(s)$. This distribution describes a probability $p = P(s, s')$ as the probability to make the step from s to s' in the MDP. By definition $p > 0$ holds if s' is a successor of s on a path of the MDP. A valid sequence of indices and probability distributions is described by an adversary (see Def. 6).

Definition 6. Adversary / Scheduler
An adversary A of an MDP τ_{MDP} *is a function that maps all finite paths* $\omega = s_0(j_0, p_0)s_1 \ldots s_n$ *to one element of* $\mathbf{Steps}(s_n)$*.*

An adversary describes an infinite sequence of non-deterministic choices of probability distributions. A specific adversary resolves the non-determinism by defining which probability distribution is chosen from $\mathbf{Steps}(s_n)$ for every finite path of the MDP.

In general there are infinitely many adversaries possible in a MDP. Of most interest are the "worst-case" and "best-case" adversaries, which always make the "worst" or "best" decision. This will be discussed in more detail in Section 3.4. Here, the adversaries are used to eliminate non-determinism from the model (an adversary consists of concrete choices) by projecting the non-deterministic choices onto one specific deterministic model. In order to get a probability measure for MDP using adversaries, a formal model of such *deterministic* probabilistic behavior is needed (see. Def. 7).

Definition 7. Discrete Time Markov Chain
A discrete time Markov chain (DTMC) *is a tuple*

$$\tau_{DTMC} = (S, p, AP, L, s_0)$$

where

- S *is a finite or infinite, non-empty set of states*
- $p : S \times S \to [0, 1]$ *is a function such that* $\forall s \in S$*:*
  - $\forall s, t \in S : p(s, t) \geq 0$
  - $\Sigma_{t \in S} p(s, t) = 1$

- *AP is a finite set of atomic propositions*
- *L is a labeling function* $L : S \to 2^{AP}$ *that labels each state* $s \in S$ *with a subset of the atomic propositions that hold in this state*
- $s_0 \in S$ *is the initial state*

A DTMC is a special case of a MDP that has a constant **Steps** function, where only a single global probability distribution exists to choose from. Paths (Def. 8) and traces (Def. 9) are defined analogously to paths and traces of a MDP. The difference here is that a path is comprised of a sequence of only states.

Definition 8. Path of a DTMC
A path π *of a DTMC model* τ_{DTMC} *is a finite or infinite sequence of states* $s_i \in S$ *of the form* $\pi = s_0, s_1, \ldots$ *and* $p(s_i, s_{i+1}) > 0$.

Definition 9. Trace of a DTMC
A trace σ *of a path* $\pi = s_0, s_1, \ldots$ *of a DTMC Model* τ *is the word over the alphabet* 2^{AP}*, generated from*

$$\sigma(\pi) = L(s_0)L(s_i)\cdots$$

Every finite path π of a DTMC has an associated set of infinite paths called the basic cylinder (Def. 10). Each infinite path in the basic cylinder has π as a prefix, i.e. the elements of the basic cylinder of π are all infinite postfixes of π.

Definition 10. Basic Cylinder of a DTMC
For any finite path π *of a DTMC, the set* $Path_{infin}(\pi)$ *of infinite postfixes of* π*, the* basic cylinder *is defined as*

$$\Delta(\pi) = \{\rho \in Path_{infin}(\pi) | \pi \text{ is prefix of } \rho\}$$

This basic cylinder is used to define the probability space for a DTMC as in Def. 11. It defines a probability measure for the infinite continuations of a finite path of a DTMC.

Definition 11. Probability Space of DTMC Model
For $\tau_{DTMC} = (S, p, AP, L, s_o)$ *and* $s \in S$*, the probability space* Υ_{dtmc} *is defined as:*

$$\Upsilon_{dtmc} = (\Delta(s), \Delta^s, prob_s)$$

such that

- Δ^s *is the* σ*-algebra generated by the empty set and the basic cylinders over* S *that are contained in* $\Delta(s)$.
- $prob_s$ *is the uniquely induced probability measure which satisfies the following:* $prob_s(\Delta(s_0)) = 1$ *and for all basic cylinders* $\Delta(s_0, s_1, \ldots, s_n)$ *over* S*:*

$$prob_s(\Delta(s_0, s_1, \ldots, s_n)) = p(s_0, s_1) \cdot p(s_1, s_2) \cdot \ldots \cdot p(s_{n-1}, s_n)$$

The probability measure is defined over the basic cylinders of finite paths. The path starting in s_0 of length one, therefore has probability 1. For every finite path, the probability of its basic cylinder is defined as the product of the probabilities to reach the respective successor states on the path in the sequence they appear on it.

This definition is sound, as from the definition of a DTMC it is known that from every reachable state there is always a successor state. This means that from every reachable state s there exists a continuation with probability 1 and no deadlock is possible. So for a finite path ending in state s_n, the basic cylinder $\Delta(s_n)$ is the set of all possible infinite continuations. Therefore the probability that a successor of s_n exists on any of those continuations is 1.

Definition 12. Probability Measure for Paths of an Adversary

For each possible adversary A, a probability measure Pr_s^A can be defined by considering a variant of DTMC with infinitely many states $\tau_{DTMC\infty} = (Path_{fin}^A(s), T, Pr_s^A, AP, L, s)$ where

- *the set of states $Path_{fin}^A(s)$ are the finite paths starting in state s*
- *initial state s, the path starting in s with length 0*
- *$Pr_s^A(\omega, \omega') = p(s_n, s)$ if $\omega = s_0(a_0, p_0)s_1 \ldots s_n$, $\omega' = \omega(a, p)s$ and $A(\omega) = (a, p)$*
- *$Pr_s^A(\omega, \omega') = 0$ otherwise*

This probability measure for paths of an adversary is used to specify the probability space of a MDP. Every adversary corresponds to an infinite sequence of non-deterministic choices which projects a MDP to an infinite state DTMC. The states of this DTMC represent the finite paths starting in the state s. As there is an infinite number of finite paths in the MDP, there are infinitely many states in the DTMC. If s_n is the last state of such a finite path, then the probability to reach a state s' from s_n is defined as $P(s_n, s')$ where P is the probability distribution chosen by the adversary at step n.

Definition 13. Probability Space of a MDP

For $\tau_{MDP} = (S, \mathbf{Steps}, AP, L, s_0)$, an adversary A and $s \in S$, the probability space Υ_{mdp}^A is defined as:

$$\Upsilon_{mdp}^A = (Path_{infin}^A(s), \Sigma_s^A, Pr_s^A)$$

such that

- *Σ_{MDP}^A is the σ-algebra generated by the empty set and the infinite paths of the associated (countable) infinite state DTMC $\tau_{DTMC\infty} = (Path_{fin}^A(s), Pr_s^A, AP, L, s)$ in Def. (12)*
- *Pr_s^A is the uniquely induced probability measure as defined in Def. (12) such that*

– $Pr_s^A(Path_{infin}^A(s)) = 1$

– *for a finite path* $\omega = s_0(j_0, p_0) \cdots s_n(j_n, p_n)s_{n+1}$:

$$Pr_s^A(Path_{infin}^A(\omega)) = \prod_{i=0}^{n-1} A(\omega)(s_{i+1})$$

So to compute the probability of the paths of a MDP, at first *all* non-deterministic choices are resolved. This is done by choosing a specific adversary. This results in an infinite-state DTMC for which a probability measure can be defined as shown in Def. 11. As there are infinitely many possible adversaries, it is not possible to compute the probability for all. Realistically only two choices for adversaries are of interest, the "worst-case" and the "best-case" adversary[5]. A more detailed explanation about this is given in Section 3.4.

3.3.3. Qualitative Formal Model

For some analyses, probabilistic information is not relevant or desired. Reasons for this are that the analysis itself does not make use of the probabilistic information or that the computation of the associated probabilities does not provide an advantage. An analysis without probabilistic information is called qualitative. An example for such a qualitative analysis would be the question: "Is a state s reachable from the initial state of a SAML model?". This is the same as computing the probability to reach the state s from the initial state and using only the information that the probability is non-zero[6].

One prominent advantage of qualitative in contrast to quantitative analyses is that they are easier and therefore less costly to compute. The reason is mainly that no arithmetic operations must be conducted and the state space can be represented in a more compact way. Therefore it is reasonable not to use quantitative analyses if an equivalent qualitative analysis method exists. This is supported in SAML by exploiting the fact that every MDP has an associated Kripke structure which forms the basis for qualitative analyses. The following definitions (14)-(16) are adapted from Clarke et al. [CGP00] (pp. 14-15).

Definition 14. Kripke Structure

A Kripke structure *over a set AP of atomic propositions is a tuple*

$$\tau_{Kripke} = (S, S_0, T, L, AP)$$

with

[5]What the worst and best case is depends of course on the type of probability that is computed. For safety analysis the worst case is a higher probability of the occurrence of a hazardous situation.

[6]There are cases where an infinite path exists but has probability 0, this will be discussed in Section 3.4.

- S *a finite, non-empty set of states*
- $S_0 \subseteq S$ *a set of initial states*
- $T \subseteq S \times S$ *a* total *transition relation, i.e. for each state* $s \in S$ *there is a state* $s' \in S$ *such that* $(s, s') \in T$ *holds*
- $L : S \to 2^{AP}$ *is a mapping that labels each state* $s \in S$ *with a subset of the atomic propositions that hold in this state (all others are false)*

The composition of a Kripke structure is similar to a MDP. It consists of a finite set of states, a labeling function and a set of atomic propositions. The difference here is the existence of a transition relation instead of a **Steps** function. The transition relation eliminates the probability distributions and only non-deterministic behavior is retained.

Due to this similarity, paths (Def. 15) and traces (Def. 16) of a Kripke structure can be defined analogously as for MDPs. The only difference is the absence of the probability distributions in the definition of a path.

Definition 15. Path of a Kripke Structure

A path π *of a Kripke structure* τ_{Kripke} *is a finite or infinite sequence of states* $s_i \in S$, $s_0 \in S_o$ *with* $(s_i, s_{i+1}) \in T$ *of the form*

$$\pi = s_0, s_1 \dots$$

Definition 16. Trace of a Kripke Structure

A trace σ *of path* $\pi = s_0, s_1 \dots$ *of a Kripke structure* τ_{Kripke} *is the word over the alphabet* 2^{AP}*, generated from*

$$\sigma(\pi) = L(s_0)L(s_1)\cdots$$

Every SAML model has an embedded Kripke structure, which has the same state space and transitions as the underlying MDP. This Kripke structure defines the *qualitative semantics* of a SAML model. It is obtained by eliminating the probabilistic but keeping the non-deterministic behavior of the SAML model.

Definition 17. Embedded Kripke Structure

Let $\kappa(\tau_{MDP})$ *be a mapping of the MDP*

$$\tau_{MDP} = (S, \mathbf{Steps}, AP, L, s_0)$$

to a tuple $(S, \{s_0\}, T, L, AP)$ *with*

$$T := \{(s,t) | s,t \in S \wedge \exists (j,p) \in \mathbf{Steps}(s) : p(s,t) > 0\}$$

then $\tau_{Kripke} = (S, \{s_0\}, T, L, AP)$ *is called the* embedded Kripke structure *of* τ_{MDP}.

The embedded Kripke structure of a MDP is constructed by abstracting from the probabilities by the introduction of additional non-deterministic choices. All those pairs of states s, s' for which a probability distribution exists such that $p(s, s') > 0$, the pair (s, s') is included in the transition relation of the embedded Kripke structure.

Formally every discrete probability distribution is therefore replaced by a set of probability distributions. For each probability distribution p' in this set $p'(s, s') = 1$ for which $p(s, s') > 0$ held in the original MDP and $p'(s, t) = 0$ for every state t for which no probability distribution with $p(s, t) > 0$ existed in the MDP. Formally, this results in a set of probability distributions where each successor state is reached with probability 1. The choice which probability distribution is used and therefore which successor state is reached is made non-deterministically. When this transformation is conducted for each probability distribution, all probabilistic behavior is transformed to non-deterministic behavior resulting in a proper Kripke structure (Lemma 1).

Lemma 1. Embedded Kripke Structure

Let $\tau_{Kripke} = \kappa(\tau_{MDP})$ be the tuple $\tau_{Kripke} = (S, \{s_0\}, T, L, AP)$ as defined in Def. 17, then τ_{Kripke} is a Kripke structure.

proof see p. 167

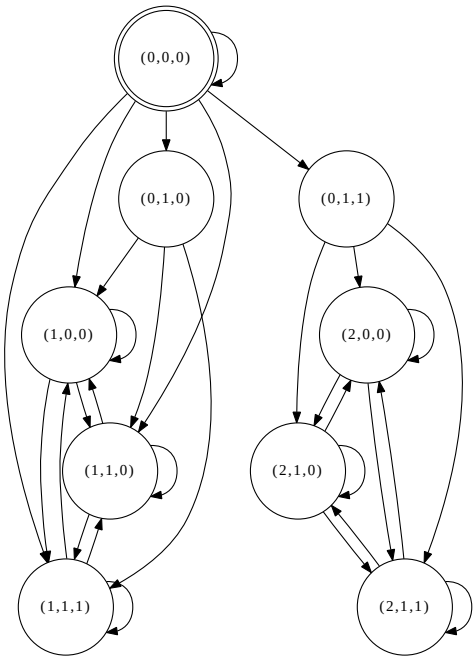


Figure 3.5.: Mapping of the state space of the MDP in Fig. 3.4 to a Kripke structure

An example of the mapping of the example SAML model shown in Fig. 3.3 onto its embedded Kripke structure is presented in Fig. 3.5. Unsurprisingly its structure is

basically equivalent to the state space of the MDP in Fig. 3.4. The differences are the edge labels without probabilities and the merging of transitions between the same states but from different probability distributions.

This structural similarity is exploited in Lemma 2 which shows that the set of paths generated by a MDP τ_{MDP} when projected onto the state sequences is equivalent to the set of paths of the embedded Kripke Structure.

Lemma 2. Embedded Kripke Structure Path Equivalence

Let ρ be the projection of a path of the MDP of the form $\omega = s_0(j_0, p_0)s_1 \ldots$ to a state sequence π of the form $\pi = s_0 s_1 \ldots$.

Then the diagram in Fig. 3.6 is commutative, i.e. $Paths(\kappa(\tau_{MDP})) = \rho(Paths(\tau_{MDP}))$.

proof see p. 167

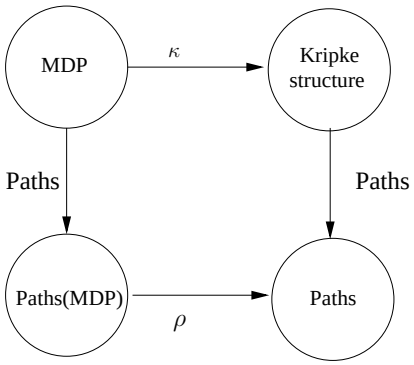


Figure 3.6.: Mapping of MDP Paths to Kripke Structure Paths

This means that for each path of the MDP when it is projected onto its sequence of states, there exists an analogous path in the set of paths of its embedded Kripke structure. Conversely, every path π of the embedded Kripke structure corresponds to a path of the MDP whose projection onto the state sequence is equivalent to π. A corollary of this is that the set of traces generated by a MDP and its embedded Kripke structure are the same.

Corollary 1. Embedded Kripke Structure Trace Equivalence

A MDP τ_{MDP} and its embedded Kripke structure $\tau_{Kripke} = \kappa(\tau_{MDP})$ are isomorphic wrt. their set of traces.

proof see p. 168

Therefore every qualitative assertion on the sequence of labellings of a trace of the MDP holds if and only if there exists a trace of its embedded Kripke structure on which it holds. This forms the sound basis to conduct a qualitative analysis of properties of the MDP on its embedded Kripke structure. These properties can often be formulated conveniently using temporal logic formulas.

3.4. Temporal Logics

Temporal logic is well applicable to formalize properties of a MDP or a Kripke structure in such a way that they can be checked automatically. Temporal logic was introduced by Pnueli in [Pnu77]. Since then, different variants have been developed, varying in their degree of expressiveness and suitability for automatic proving. Temporal logic allows to specify properties of systems which change over time. The two most important classes of properties for temporal logics (originally introduced by Lamport in [Lam77]) which are used for analyses in SAML are:

- *Liveness* properties: "Something will eventually happen"
- *Safety* properties: "Something will never happen"

Liveness properties are often of importance where functional properties of a system should be verified. Safety properties are used for model-based safety analysis in the form of: something bad will never happen. For the SAML framework, only decidable logics are of interest which can efficiently be checked using automatic analysis techniques. More expressive logics which are often undecidable can be used with interactive proof systems for state-based models like SAML. One example using a first order variant of Interval Temporal Logic (ITL) is described by Thums et al. in [TOWS04]. Nevertheless, these are often of limited practical value, as the resulting proof obligations are too numerous even for small examples. In addition, to conduct such a proof, an expert in the domain of interactive theorem proving is needed and the goal of SAML is the integration into industrial tools and development processes which requires automation as much as possible.

Similar to the definition of the semantics of SAML models, different temporal logics for the specification of qualitative and quantitative properties are employed.

3.4.1. Syntax and Semantics of CTL*

Analogous to the traces generated by the labeling of the paths of a MDP, temporal logic formulas also describe traces. The qualitative computation tree logic CTL* formalizes the behavior of a system as a computation tree with the initial state at its root. Whenever a system has different possibilities to continue on a path, there are several successor child nodes in the tree. A CTL* formula holds in a MDP τ_{MDP}, if the traces that τ_{MDP} describes are a subset of the traces that the formula describes. CTL* uses the *modal operators* "Always" (**A**) and "Exists" (**E**) and the *temporal operators* "Globally" (**G**), "Finally" (**F**) and "Next" (**X**). The following definitions (18)-(21) for the syntax and semantics are adapted from Clarke et al. [CGP00] (pp. 27-32).

Definition 18. Syntax of CTL*

- *If $p \in AP$, then p is a state formula.*

- *If ϕ_i and ϕ_2 are state formulas, then $\neg\phi_i$, $\phi_i \vee \phi_j$ and $\phi_i \wedge \phi_j$ are state formulas.*
- *If ϕ is a path formula, then $\mathbf{E}\,\phi$ and $\mathbf{A}\,\phi$ are state formulas.*
- *If ϕ is a state formula, then ϕ is also a path formula.*
- *If ϕ_i and ϕ_j are path formulas, then $\neg\phi_i$, $\phi_i \vee \phi_j$, $\phi_i \wedge \phi_j$, $\mathbf{X}\phi_i$, $\mathbf{G}\,\phi_i$ and $\phi_i\mathbf{U}\,\phi_j$ are path formulas.*

In the case of SAML, the atomic propositions of the set AP generally are Boolean expressions over the values of the state variables of SAML modules. Note that in contrast to SAML models, there is currently no parser developed for the proof obligations and these have to be specified manually in the required format. Nevertheless, such an implementation would be rather straightforward.

Definition 19. Semantics of CTL*

Let τ_{Kripke} be a Kripke structure, ϕ denote state and ψ path formulae and π^i denote the suffix of the path π starting in the state s_i, then the CTL semantics is defined as:*

$$
\begin{aligned}
\tau_{Kripke}, s \models p &\Leftrightarrow p \in L(s) \\
\tau_{Kripke}, s \models \neg\phi &\Leftrightarrow \tau_{Kripke}, s \not\models \phi \\
\tau_{Kripke}, s \models \phi_1 \vee \phi_2 &\Leftrightarrow \tau_{Kripke}, s \models \phi_1 \text{ or } \tau_{Kripke}, s \models \phi_2 \\
\tau_{Kripke}, s \models \phi_1 \wedge \phi_2 &\Leftrightarrow \tau_{Kripke}, s \models \phi_1 \text{ and } \tau_{Kripke}, s \models \phi_2 \\
\tau_{Kripke}, s \models \mathbf{E}\,\psi &\Leftrightarrow \text{there is a path } \pi \text{ from } s \text{ such that } \tau_{Kripke}, \pi \models \psi \\
\tau_{Kripke}, s \models \mathbf{A}\,\psi &\Leftrightarrow \text{for every path } \pi \text{ starting from } s : \tau_{Kripke}, \pi \models \psi \\
\tau_{Kripke}, \pi \models \phi &\Leftrightarrow s \text{ is the first state of } \pi \text{ and } \tau_{Kripke}, s \models \phi \\
\tau_{Kripke}, \pi \models \neg\psi &\Leftrightarrow \tau_{Kripke}, \pi \not\models \psi \\
\tau_{Kripke}, \pi \models \psi_i \vee \psi_j &\Leftrightarrow \tau_{Kripke}, \pi \models \psi_i \text{ or } \tau_{Kripke}, \pi \models \psi_j \\
\tau_{Kripke}, \pi \models \psi_i \wedge \psi_j &\Leftrightarrow \tau_{Kripke}, \pi \models \psi_i \text{ and } \tau_{Kripke}, \pi \models \psi_j \\
\tau_{Kripke}, \pi \models \mathbf{X}\psi &\Leftrightarrow \tau_{Kripke}, \pi^1 \models \psi \\
\tau_{Kripke}, \pi \models \mathbf{F}\,\psi &\Leftrightarrow \exists i \geq 0 : \tau_{Kripke}, \pi^i \models \psi \\
\tau_{Kripke}, \pi \models \mathbf{G}\,\psi &\Leftrightarrow \forall i \geq 0 : \tau_{Kripke}, \pi^i \models \psi \\
\tau_{Kripke}, \pi \models \psi_1\mathbf{U}\,\psi_2 &\Leftrightarrow \exists i \geq 0 : \tau_{Kripke}, \pi^i \models \psi_2 \\
&\quad \wedge \forall j < i : \tau_{Kripke}, \pi^j \models \psi_1
\end{aligned}
$$

The distinction of state and path formulas in CTL* allows for the specification of two commonly used subsets, namely CTL (computation tree logic) and LTL (linear time logic). The advantage of using one of these logics is that for both these logics, automatic model-checking algorithms and tool implementations exist. These are more efficient than full CTL* model-checking. More details of the algorithms can be found for example in McMillan [McM90] (pp. 35-40) or Clarke et al. [CGP00] (pp. 35-51 and pp. 61-97).

Definition 20. Computation Tree Logic (CTL)
CTL is the subset of CTL in which each modal operator is immediately followed by a temporal operator and each temporal operator is preceded directly by a modal operator.*

Definition 21. Linear Time Logic (LTL)
LTL is the subset of CTL in which no modal operators are allowed.*

These two logics do have a common subset of properties, but for each there exist properties which can be specified in one but not in the other and vice versa. In addition there are properties that are expressible in CTL* but in neither LTL or CTL alone. For a more detailed discussion about this see for example Maidl [Mai00].

For the usage for the safety analysis methods presented in Chapter 5, either LTL or CTL is used. Full CTL* is only used to prove certain properties of the analysis techniques, but not for the automatic proofs for safety analysis. These are checked using efficient model-checking tools for safety analysis in SAML.

3.4.2. Syntax and Semantics of PCTL

The previous logics allow for the specification of qualitative properties of a system, i.e. refer to the semantics of the embedded Kripke structure of a MDP. In order to specify quantitative properties, the probabilistic variant *probabilistic computation tree logic* (PCTL) is used. The syntax and semantics definitions (22)-(24) are derived from Hansson and Jonsson [HJ94]. PCTL allows to formalize properties like:

- "The probability that something will eventually happen is higher than p."
- "The probability that something will never happen is at least p."

Definition 22. Syntax of PCTL

- *If $p \in AP$, then p is a state formula.*
- *If ϕ_i and ϕ_2 are state formulas, then $\neg\phi_i$, $\phi_i \vee \phi_j$ and $\phi_i \wedge \phi_j$ are state formulas.*
- *If ϕ is a path formula, then $[\phi]_{\ p}$ with $\in \{\leq, <, \geq, >\}$ and $p \in [0,1]$ is a state formula.*
- *If ϕ is a state formula, then ϕ is also a path formula.*
- *If ϕ_i and ϕ_j are path formulas, then $\neg\phi_i$, $\phi_i \vee \phi_j$, $\phi_i \wedge \phi_j$, $\mathbf{X}\phi_i$, $\mathbf{G}\,\phi_i$, $\phi_i \mathbf{U}\,\phi_j$ and $\phi_i \mathbf{U}^{\leq k} \phi_j$ are path formulas.*

Definition 23. Semantics of PCTL

Let τ_{MDP} be a MDP, ϕ denote state and ψ path formulae and π^i denote the suffix of the path π of the MDP starting in state s_i, then the PCTL semantics is defined as:

$$
\begin{aligned}
\tau_{MDP}, s \models p &\Leftrightarrow p \in L(s) \\
\tau_{MDP}, s \models \neg\phi &\Leftrightarrow \tau_{Kripke}, s \not\models \phi \\
\tau_{MDP}, s \models \phi_1 \vee \phi_2 &\Leftrightarrow \tau_{Kripke}, s \models \phi_1 \text{ or } \tau_{Kripke}, s \models \phi_2 \\
\tau_{MDP}, s \models \phi_1 \wedge \phi_2 &\Leftrightarrow \tau_{Kripke}, s \models \phi_1 \text{ and } \tau_{Kripke}, s \models \phi_2 \\
\tau_{MDP}, s \models P[\psi]_{\sim p} &\Leftrightarrow \text{for all adversaries } A\text{: } Prob^A(s,\phi) \sim p \\
&\qquad Prob^A(s,\psi) := Pr_s^A\{\pi \in Paths_{inf}(s) : \\
&\qquad\qquad \tau_{MDP}, \pi \models \psi\} \\
&\qquad \text{with } \sim \in \{>, \geq, <, \leq\} \qquad (3.4) \\
\tau_{MDP}, \pi \models \phi &\Leftrightarrow s \text{ is the first state of } \pi \text{ and } \tau_{MDP}, s \models \phi \\
\tau_{MDP}, \pi \models \neg\psi &\Leftrightarrow \tau_{MDP}, \pi \not\models \psi \\
\tau_{MDP}, \pi \models \psi_i \vee \psi_j &\Leftrightarrow \tau_{MDP}, \pi \models \psi_i \text{ or } \tau_{MDP}, \pi \models \psi_j \\
\tau_{MDP}, \pi \models \psi_i \wedge \psi_j &\Leftrightarrow \tau_{MDP}, \pi \models \psi_i \text{ and } \tau_{MDP}, \pi \models \psi_j \\
\tau_{MDP}, \pi \models \mathbf{X}\phi &\Leftrightarrow \tau_{MDP}, \pi^1 \models \phi \\
\tau_{MDP}, \pi \models \mathbf{F}\,\psi &\Leftrightarrow \exists i \geq 0 : \tau_{MDP}, \pi^i \models \psi \\
\tau_{MDP}, \pi \models \mathbf{G}\,\psi &\Leftrightarrow \forall i \geq 0 : \tau_{MDP}, \pi^i \models \psi \\
\tau_{MDP}, \pi \models \psi_1 \mathbf{U}\,\psi_2 &\Leftrightarrow \exists i \geq 0 : \tau_{MDP}, \pi^i \models \psi_2 \\
&\qquad \wedge \forall j < i : \tau_{MDP}, \pi^j \models \psi_1 \\
\tau_{MDP}, \pi \models \psi_1 \mathbf{U}^{\leq t} \psi_2 &\Leftrightarrow \exists i, 0 \leq i \leq t : \tau_{MDP}, \pi^i \models \psi_2 \wedge \\
&\qquad\qquad \forall j < i : \tau_{MDP}, \pi^j \models \psi_1 \qquad (3.5)
\end{aligned}
$$

The main difference to the qualitative logic CTL is the quantification over paths. In CTL, there exist the modal operators $\mathbf{A}$ and $\mathbf{E}$. In PCTL these are generalized by a probability threshold. These quantitative aspects of a PCTL formula can be computed as shown in Eq. (3.4). It is defined that with probability $p' := Prob^A(s,\phi)$ a given formula holds in the MDP and for a threshold value p, a relation of the form $p' \sim p, (\sim \in \{>, \geq, <, \leq\})$ holds for *all possible* (infinitely many) adversaries A. As already mentioned, the computation of infinitely many is not possible. Fortunately it is possible to compute the explicit probabilities that a PCTL formula holds for two very important adversaries. These two adversaries are the "best-case" and "worst-case" and lead to the minimal and maximal probabilities that a given PCTL formula holds.

Definition 24. Minimal / Maximal Probability of a PCTL Formula

$$P_{min}[\phi] := min\{Pr^A\{\pi \in Paths_{inf}(s_0) : \tau_{MDP}, \pi \models \phi\},\ \text{adversary } A\}$$

$$P_{max}[\phi] := max\{Pr^A\{\pi \in Paths_{inf}(s_0) : \tau_{MDP}, \pi \models \phi\},\ adversary\ A\}$$

The logics CTL and PCTL are closely related and a lot of properties are expressible in both logics. An important difference are formulas of the form $\mathbf{E}\,\mathbf{G}\,\phi$ which are not directly translatable into PCTL as $P_{max}[\mathbf{G}\,\phi]_{>0}$, although this seems quite obvious.

The reason for this is that is possible that a path exists on which ϕ holds globally, but this path has a probability of 0. An example for this is the MDP in Fig. 3.7. Assume the proposition q holds in state 0 but not in state 1, i.e. $q \in L(0)$ but $q \notin L(1)$. Then $\mathbf{E}\,\mathbf{G}\,q$ holds as there exists a loop from state 0 to itself. But if $p < 1$ the probability to stay in state 0 forever is $\lim_{n\to\infty} p^n = 0$, therefore $P_{max}[\mathbf{G}\,\phi]_{>0}$ does not hold.

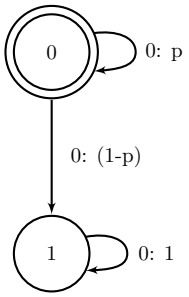


Figure 3.7.: Example MDP for $\mathbf{E}\,\mathbf{G}\,\phi \not\equiv P_{max}[\mathbf{G}\,\phi]_{>0}$

Further discussions on different temporal logics for quantitative and qualitative aspects of MDPs and their relation to each other are described for example by Huth and Kwiatkowska [HK98] and Chatterjee et al. [CdAFL09]. For the safety analysis methods introduced in this dissertation, CTL/LTL and PCTL are sufficient. They have the advantage that efficient and robust formal analysis tools exist for property verification.

3.5. Graphical Representation of SAML Models

Although the formal semantics is based on the textual representation of SAML models, it is often very convenient and more comprehensible to represent a model in a graphical way. MDPs and Kripke structures have already been represented in a similar way in Fig. 3.4 and Fig. 3.5.

These diagrams showed the complete state-space of a model. As SAML is specified using parallel finite state automata, an informal graphical notation for parallel automata can be used to represent the parallel modules. This notation is well suited for a first conceptual expression, before a SAML model is developed, or to visualize such a model.

A SAML model is visualized as its parallel single modules. The state of each module is represented as a node of a directed graph, each node represents one valuation of all state variables of the module. A directed edge from one state s to a small filled intermediate node corresponds to a non-deterministic choice of a probability distribution. The labeling

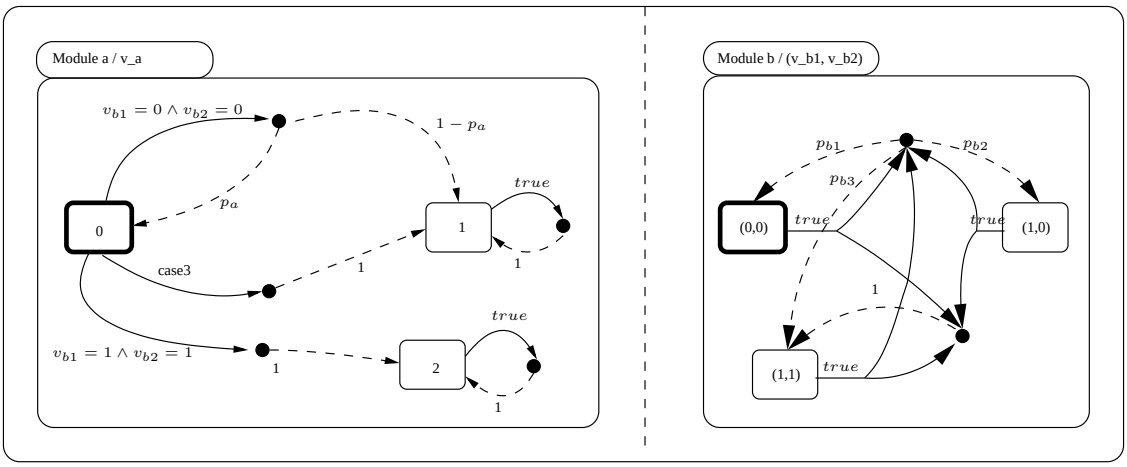


Figure 3.8.: Graphical Representation of the Example Model

of the edges is the corresponding activation condition[7]. The directed dashed edges from an intermediate node to a state s' are labeled with the probability to reach the state s' from state s.

In Fig. 3.8 the graphical representation of the SAML model of Fig. 3.2 is shown. It consists of the two parallel finite state modules `a` and `b`. Each possible state of a module is represented as a node of a directed graph. As the module `a` consists of only a single state variable, each state holds only one value. Module `b` consists of two state variables, its states therefore hold a vector of values of the form `(v_b1, v_b2)`. The initial state of `a` is indicated via the thicker boundary and has the value 0, the initial state of `b` is $(0, 0)$. For this graphical notation it is often more convenient to use symbolic names for the values of a state variable instead of integers. This notation will be used in many of the presented examples in the following chapters.

3.6. Related Work

The TopCased project is a large open source approach to build a modeling framework for different modeling languages for software and systems modeling. The approach is described by Vernadat et al. in [VPF+06]. Target input languages are SysML [Sys08], UML [Gro06] and the Architecture analysis and design language (AADL) [SA04]. One aspect of TopCased is to allow for the formal analysis of these models. The approach here is to use the language FICARE as an intermediate language which is described by Fairail et al. [FGP+08]. Models specified in the modeling framework are then transformed into FIACRE as intermediate language. From there models are transformed into the input

[7]The parts of the activation condition that refer to the values of the state variables of the current module are left out. This constraint is implicitly fulfilled by the source node of an edge.

language of different analysis tools like the CADP toolbox developed at INRIA and described by Fernandez et al. [FGK+96] or the TINA petri net analyzer described by Berthomieu et al. in [BaFV04].

Some of the problems arising from the approach taken in the TopCased project are described by Farail et al. [FGP+08]. The modeling framework provides so many modeling artifacts, that the resulting FIACRE models are often too complex for the analysis. Effectively only very small examples can be analyzed at the moment. Mainly structural aspects can be verified. SAML has deliberately been kept as simple as possible to prevent such a situation.

Harel's Statemate state-charts [HN96] provide a well-defined formal semantics and have a reduced set of modeling artifacts compared to TopCased models. They can therefore be used for qualitative formal analysis of the modeled systems. In Thums et al. [TOWS04] the interactive theorem prover KIV is used for verification. In Chan et al. [CAB+01] and Clarke and Heile [CH00], automatic model-checking techniques are used. In the AVACS [AVA03] project, Böde et al. [BPRW08a] developed a state-chart extension with probabilistic behavior and used it for safety analysis. The resulting model is transformed into a labeled transition system and minimized with bisimulation techniques using the CADP [FGK+96] toolbox. The final transition system is then analyzed with the MRMC [KKZ05] probabilistic model-checker of Katoen et al. [KKZ05].

Maybe the most elaborate framework especially for the specification of safety-critical systems and model-based safety analysis is developed in the Correctness, Modeling and Performance of Aerospace Systems (COMPASS) project [BCK+09a] described by Bozzano et al. [BCK+10b]. It combines both qualitative and quantitative modeling capabilities via a formalization of a subset of AADL [SA04] and its error annex [SA06]. The quantitative analysis is then conducted using the MRMC model checker.

The developed language is called SLIM [BCK+09b]. Models in SLIM can contain hybrid continuous behavior, the failure modeling can contain continuous time failure rates. There is a strict discrimination between the nominal behavior (without occurrence of failures) and the failure behavior. One drawback here is that it is not possible to specify probabilistic nominal behavior, but only probabilistic failure behavior. Probabilistic behavior can often be used to specify realistic environment models which is very important for safety analysis as safety-critical system cannot be examined in isolation. In Chapter 4 the usage of probabilistic behavior in SAML to model realistic environment for both nominal and failure behavior will be presented.

Both approaches use a continuous time model which, according to Hermanns et al. [HKMKS00], is well suited for asynchronous interleaved systems. Many safety-critical systems are developed using synchronous parallel components, SAML is based on discrete time synchronous parallel semantics. Of course, modeling asynchronicity is possible in SAML , but must be done explicitly. In addition, probabilistic discrete time modeling is an advantage if different failure types must be modeled. This is explained in more detail in Chapter 4.

Summary

This chapter introduced the syntax and formal semantics of the SAML framework. It describes synchronous parallel automata which can have a combination of non-deterministic and probabilistic transitions. This allows for the expression of both quantitative models and also purely qualitative ones. The underlying formal semantics are Markov Decision Processes (MDP). Properties are expressed as temporal logics formulas. Qualitative properties in LTL or CTL and quantitative properties in PCTL. Any SAML model can be abstracted to a purely qualitative model by the transformation into its embedded Kripke structure. This allows for the analysis of both qualitative and quantitative properties of a SAML model.

4. SAML Modeling for Safety Analysis

On two occasions I have been asked: "Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?"
I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.

(Charles Babbage)

This chapter describes how models for safety analysis can be constructed using SAML. These models generally consist of modeling of hardware, software and the surrounding environment and are called extended system models. Correct environment modeling is very important for accurate safety analysis, as the safety of a system cannot be understood in isolation of its surroundings. Important aspects of the environment modeling are general physical behavior and especially accurate modeling of different kinds of failure modes and their effects.

Section 4.1 gives a motivation for a general guideline for the creation of SAML system models for safety analysis. Section 4.2 introduces a case study (taken from safety analysis literature) which will be used for illustration purposes throughout the dissertation and to illustrate SAML modeling for safety analysis. Section 4.3 shortly describes the approaches for hardware and software modeling in SAML. Section 4.4 describes the approach for realistic probabilistic environment modeling in SAML and Section 4.5 describes correct and accurate failure mode modeling which is essential for safety analysis. Some related work is discussed in Section 4.6.

4.1. Motivation

To conduct a realistic model-based safety analysis, a meaningful modeling of the system under consideration is necessary. Especially if an accurate *quantitative* analysis is desired, the precision of the analysis result depends heavily on the accuracy of the modeling. Safety analysis cannot be conducted in isolation, but the environment into which the system is embedded always has to be considered too, as it often strongly influences the possible behavior.

In general, a formal system model of a software-intensive system will include software and hardware components as well as a model of the environment. Hardware components are often sensors and actors that the software uses to get information about the current environment and to make the system react to it. The environment model specifies the behavior of the surroundings of the system that influence it directly or indirectly.

When a model is built consisting of these aspects, it is possible to verify the functional correctness of a system. This means that the system works as intended and fulfills its specification. For a safety-critical system, the analysis of functional correctness alone is not sufficient. The assumption that all components work according to their specification is in general wrong as different components may fail. Therefore the most important question for safety analysis is:

> "How safe is the system, if one or more components do *not* behave according to their specification?"

To answer this question, the potential occurrence of faulty behavior of components and also the effects of this misbehavior must also be modeled accurately and be integrated into the formal system model. When the faulty behavior is integrated into a SAML model, it is called the extended system model. Qualitative and quantitative safety analyses can be conducted on an extended system model expressed in SAML. With the construction presented in this chapter, it is possible to integrate different types of failure modes while preserving the functional correctness of the system model.

Fig. 4.1 shows the general structure of an extended system model for model-based safety analysis. The software model interacts with the hardware model, reading and processing inputs and emitting control commands. The interaction of the software with the environment is generally indirectly via the hardware model. Actors and sensors are influenced by the physical environment model. All hardware components can be influenced by the occurrence of component failures and their effects specified by the failure model.

Very often there will not be a one-to-one correspondence between SAML modules and one of these aspects. But on the conceptual level, all those aspects are required for an accurate model-based safety analysis. This allows for giving some general modeling guidelines for both environmental and failure mode modeling in SAML which will often be helpful in the analysis of safety critical systems.

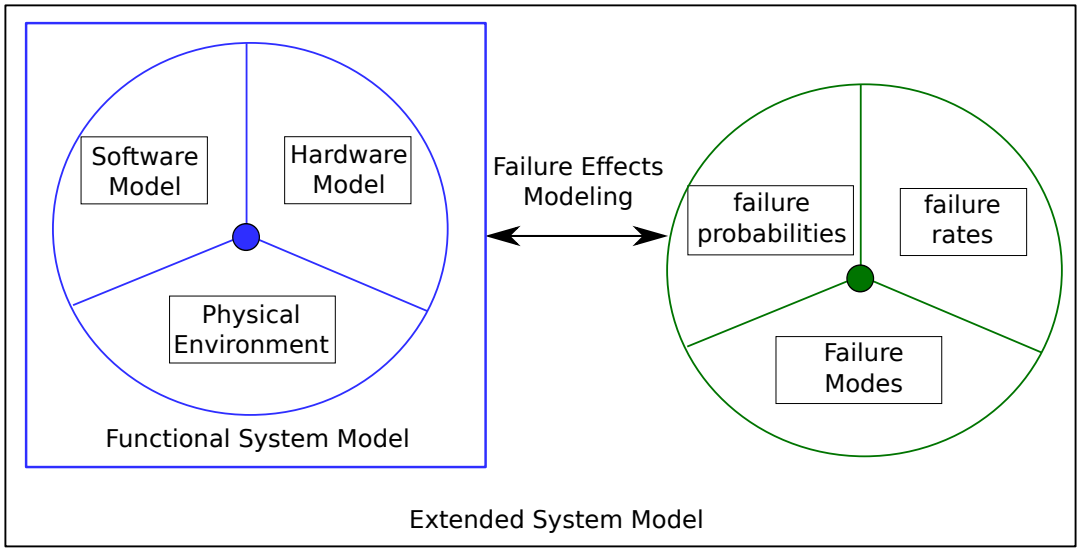


Figure 4.1.: Structure of a SAML Model for Safety Analysis

The second part of the extended system model is the modeling of the failure modes and erroneous behavior of the system. The occurrence of the failure modes is specified by occurrence probabilities (for per-demand failure modes) or failure rates (for per-time failure modes). The effect of the failure mode occurrence is modeled as additional possible behavior of the system on top of the functional model.

4.2. Example Case Study

The construction of SAML models for model-based safety analysis is illustrated with a case study originally proposed by Walker et al. [WBP07]. It will be used to illustrate the necessary steps to create an extended system model and will serve to illustrate the formal safety analysis presented in Chapter 5.

The case study is a generic system with built-in hot-spare redundancy. If the primary system operation is not possible anymore, the system enters a degraded mode as secondary operational mode. The system processes a measured input signal and produces a resulting output signal. Redundancy is used to increase the dependability of the system. A first purely qualitative modeling and analysis of this case study has been presented in [GOR08], quantitative modeling and analysis in [GO10b]. A schematic view of the case study is depicted in Fig. 4.2.

The case study consists of two redundant input sensors (S1 and S2) measuring the input signal (I). This signal is then processed in an arithmetic unit to generate the desired output signal. Two arithmetic units exist, a primary unit (A1) and its backup unit (A2). The primary unit reads the input signal from both input sensors, the backup unit only from one of the two sensors. If the primary unit produces no output signal,

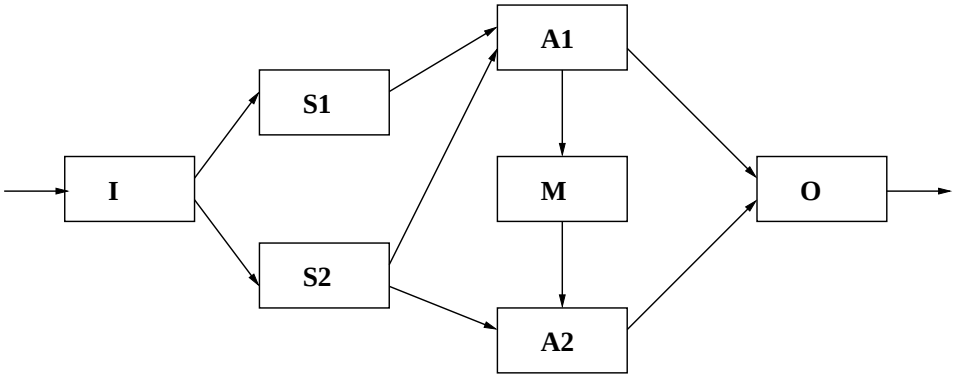


Figure 4.2.: Schematic View of Example Case Study [GOR08]

then a monitoring unit (M) switches to the backup unit for the generation of the output signal. The backup unit will only produce an output signal if it has been triggered by the monitor. If the secondary unit is triggered, the primary unit will get switched off.

The system is functionally correct if it delivers a correct output signal. It is easy to imagine that if such a system is used in a safety-critical environment, a malfunctioning (omission of values) could be very dangerous and the system can become safety-critical. Therefore the redundancy and the degraded mode have been integrated.

4.3. Hardware and Software Modeling

For safety analysis, the first step is to construct a model of the functional behavior of the system. This functional system model specifies the intended behavior of the system under consideration and is used to verify the functional correctness of the system, proving that it works as intended. The modeling of hardware and software forms the basis for such functional formal models expressed in SAML. It defines the basic behavior of the modeled system and its reaction to and its influence on its surrounding environment. To get the most accurate analysis results, it is important to analyze a system model which behaves as similar as possible as the system under consideration.

4.3.1. Software Modeling

For formal model-based safety analysis, most often software will be specified in an abstract way. Not every detail of the algorithms will be modeled, only the most important concepts. The reason for this is firstly the fact that fine-grained software specifications would increase the state space enormously, making an analysis very difficult or even impossible. Secondly, the correctness of the algorithms and the software specification can better be assured using other techniques than model-checking, e.g. interactive theorem proving, as this is often undecidable or requires infinite state spaces. Thirdly and

most important, the main interest in safety analysis is the combination of hardware, software and its behavior in a (possibly failing) environment. For such an analysis the implementation details are not of interest, but only the *important effects*[1] of the implemented software. Therefore abstract specifications of software are generally used in SAML models which show primarily these intended effects.

4.3.2. Hardware Modeling

The hardware model of a formal SAML model will most often represent the interface of the system under consideration to its surrounding physical environment. This typically consists of sensors and actors. Sensors are used to get feedback from the physical environment of the system. Depending on this feedback, the system changes its internal state according to its software specification. Actuators are then used to influence the environment according to the changed system state.

Depending on the system under consideration, the hardware model may also consist of additional aspects. Examples for this are aspects which define the spatial structure of a system and therefore the possible interactions of different components. Other possible aspects are for example general technical modeling of aspects like communication channels and component interfaces.

4.3.3. Case Study Model

One component of the example case study which reads input from hardware (sensors) and acts according to a (simple) software specification is the model of the secondary arithmetic unit (A2). Its modeling is shown in Fig. 4.3. Initially it is in state *idle*, i.e. a hot stand-by state where no output is produced. It stays in this state until it gets activated (predicate *activate* is true) by the monitoring unit. It then is in state *sig*, as long as there is data available (the predicate *signal* holds which means sensor S2 produces data). If there is no data available, the unit enters the state *noSig*, as no signal can be produced. If the sensor starts delivering data again, A2 switches back to state *sig*. Here the software controls the different states of the A2 unit, the hardware model consists of the sensors that deliver the data and the production of the output signal of the system from the arithmetic units.

The whole functional system model of the case study consists of 5 parallel SAML modules. One for each of the two sensors that indicates whether the sensor delivers a signal, one for the primary arithmetic (A1) unit and one for the monitoring unit that detects when A1 fails to deliver a signal and activates A2. The functional system model is then the parallel composition of these five SAML modules. The functional correctness of the system can be verified by proving that there is always an output signal produced once the system enters its normal operating mode.

[1] Most often the change of the internal state of the system.

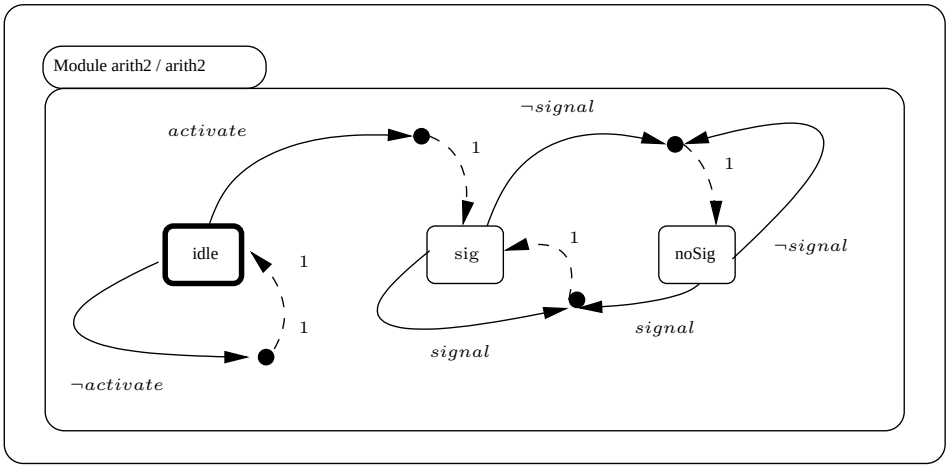


Figure 4.3.: SAML Module of Second Arithmetic Unit

4.4. Physical Environment Modeling

The accuracy of the physical environment modeling is very important for the accuracy of the analysis of a SAML model. Physical behavior can often be described in ways of measurable units and the rate of change of these units. In mathematical terms, measurable units are described as functions and the rate of change of these units is described as the derivations of these functions. An often used example of this in physical environment modeling is the function from time to space which describes the position at a given time. Its derivation is the velocity and its second derivation is the acceleration (for an example using this type of environment modeling see Section 7.1). Other physical behavior can often be modeled analogously, but it is important to note that SAML models are discrete and values can only use finite domains, so all continuous values have to be discretized.

4.4.1. Temporal Resolution

To model physical behavior realistically, the correct interpretation of the passing of time and its correct integration into the model is of biggest importance. In SAML, a discrete time model is used. In such a context, there exists a basic time unit which passes whenever the system performs a step (i.e. all parallel finite state machines execute an update rule).

This means that whenever the state variables change, the same amount of time passes for all modules. This basic unit is called the *temporal resolution* Δt of the system. In synchronous parallel systems this will usually be a basic system clock[2]. All time-

[2]This basic system clock is not equal to the CPU clock of a system, but most often a (large) multiple thereof.

dependent SAML modeling, physical behavior modeling of the environment model and especially the failure mode modeling is done relative to Δt.

Definition 25. Temporal Resolution of a SAML model
The amount of time Δt that passes at each time-step of a SAML model is called the temporal resolution. In general its unit is given in seconds or fractions thereof.

The value of Δt is very much system-dependent. As it defines the basic system clock, it must be chosen accordingly to the abstraction level of the software and hardware modeling (e.g. the sensor reading frequency). Some discussion on the effect of Δt for a formal system model is presented in Section 4.5.2 on formal failure modeling and in Section 5.3.3 on the interpretation of quantitative analysis results.

4.4.2. Case Study Model

In the functional system model of the case study, the core aspect of the physical modeling is the frequency that input signals are read, processed and sent to the output. The temporal resolution of the system is modeled as $\Delta t = 10ms$, which means that the signal frequency is $100Hz$. The frequency of the hardware in the processing units of A1 and A2 would of course be higher, but as already mentioned, Δt does not necessarily correspond to the CPU clock of the system but is an abstract basic clock.

4.5. Failure Mode Modeling

After the construction of the functional system model and the modeling of the physical environment, the most important step from the point of model-based *safety analysis* is conducted: the accurate modeling of the failure modes. The basic idea is to integrate the effect of the failure mode occurrence into the functional model. It is important to do this in such a way, that the original behavior is still retained. For the formal model this means that the functional correctness of the system model is retained although it is extended with possible defective behavior. This assures that the later safety analysis reflects properly the behavior of the system under consideration in the case of component failures. This makes the presented modeling and integration of the failure modes *conservative*. This concept was introduced by Ortmeier et al. in [ORS05], some extensions can be found in [OGR07]. But until now this integration was limited to purely qualitative failure mode modeling.

The identification of possible failure modes for a given system is not discussed here, the existence of a set of relevant failure modes is assumed. Such a set is often given directly for components for which the possible failure modes are known. Methods to systematically find possible failure modes can also be applied, for example HaZop as described by Kletz [Kle86].

4.5.1. Qualitative Formal Failure Modeling

The basic concept is to separate the occurrence pattern of a failure mode from its direct effects (based on [OGR07]). This allows for convenient modeling of failure modes that affect different system components. The occurrence pattern of a failure mode describes *when* a failure mode can appear and when or if it disappears again. It is modeled orthogonally to the system model as a parallel SAML module. The effect of a failure mode is modeled as the local effect of its occurrence and is integrated into the system model itself, most often as new states and transitions. This general failure mode modeling forms the basis for the quantitative failure mode modeling described in Section 4.5.2.

The occurrence patterns are modeled as failure modules that signal the occurrence or absence of a failure mode via the value of their state variable. This modeling can be expressed in SAML by representing the qualitative failure module as an additional parallel SAML module with non-deterministic behavior for each modeled failure mode. Fig. 4.4 shows an example for a simple failure mode occurrence pattern as graphical representation of a SAML module.

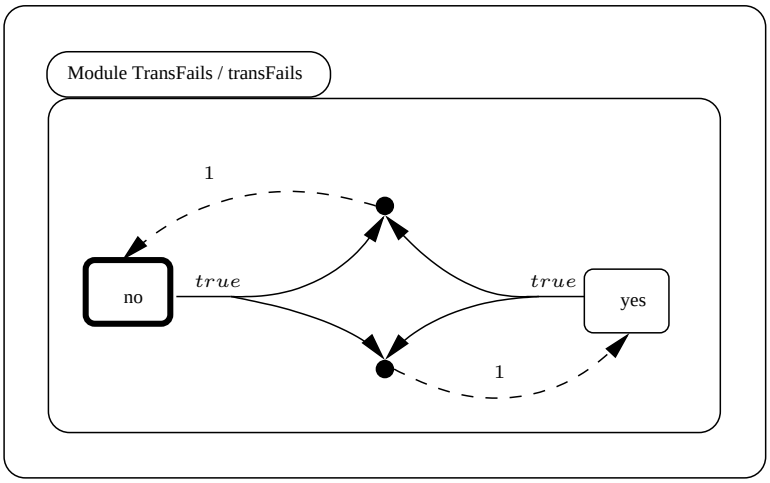


Figure 4.4.: Transient Failure Mode Occurrence Pattern

The state variable *transFails* signals the occurrence or absence of the failure mode. The state variable has two possible values, *no* for the absence of the failure mode and *yes* for its presence. Initially the failure mode is absent[3]. The failure mode in Fig. 4.4 is an example of a transient failure mode, i.e. the failure mode can become absent again after its occurrence. This is realized by adding a non-deterministic choice with activation condition *true* to both states of the state variable. This non-deterministic choice consists of two probability distributions, both with exactly one successor state which is reached with probability 1.

[3]If this is not intended, an additional *start* state must be modeled for all system modules

A transient failure mode modeling is appropriate for example for sensor failures. A sensor failure often results from a temporary disturbance which is reversible and therefore the sensor can function correctly again afterwards.

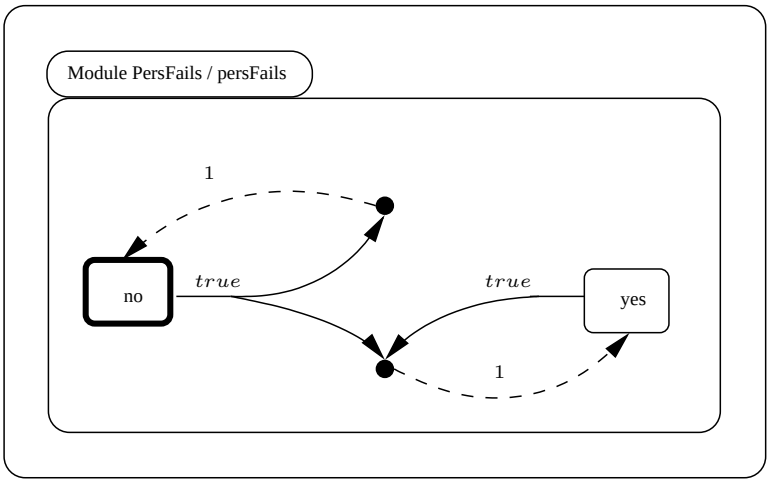


Figure 4.5.: Persistent Failure Mode Occurrence Pattern

In Fig. 4.5 a simple persistent failure mode modeling is shown. Once the failure occurred, there is no possibility for the failure mode to disappear. This is realized by allowing only a single non-deterministic choice from state *yes* which has only itself as possible successor state. Such a persistent modeling is for example appropriate for broken hardware which cannot become operational by itself again.

More complex qualitative failure mode models are possible using this modeling technique. Possible variants are repairable failures that get absent again after either some time passes or an external repair occurs. Another possibility is to have failure modes triggering others, thus modeling dependent failure modes like common-cause failures. The drawback of this previous formal failure modeling described in [OGR07] is, that it is purely qualitative. Any quantitative analysis has to be done a-posteriori using coarse approximations. A big benefit of SAML is the possible extension to also include probabilistic behavior directly in the failure modules. This allows for much more accurate modeling and therefore analysis results than previously possible. The foundations for quantitative failure mode modeling have first been introduced in [GO10b].

4.5.2. Quantitative Failure Mode Modeling

For accurate quantitative model-based safety analysis, probabilistic modeling of the failure occurrence pattern is used in the extended system model. Accurate modeling of the occurrence probabilities of the failure modes is essential as the accuracy of the safety analysis results depends heavily on it. Two main types of failure probabilities exist. The

first is *per-demand* failure probability, which is the probability that a system component fails to deliver its function at a given demand (comparable to *low demand mode* in IEC 61508 [Lad08]). The second is *per-time* failure probability which is specified by a *rate* of failures in relation to time (comparable to *high demand or continuous mode* in IEC 61508).

Which type of failure probability is best fitting for a given failure mode can only be decided on a case-by-case basis. Transient sensor failures will very often be modeled as a *per-time* failure mode, as they are active the whole time and a failure can always occur. Other failure modes, like the activation of a mechanical device, will often be modeled as a *per-demand* failure, as a clear moment of activation exists. Per-demand failures often are of persistent nature[4].

Per-Time Failure Mode Modeling

For per-time failure modes, the occurrence probability is specified by a failure rate. This per-time failure rate λ specifies the parameter for the exponential distribution as shown in Eq. (4.1). It is the expected value of occurrences of the failure mode in a given time interval. $P(X \leq t)$ as in Eq. (4.1) describes the probability that the failure mode appears before or at time t.

$$P(X \leq t) = \int_0^t e^{-\lambda t} \mathrm{d}t = 1 - e^{-\lambda t} \tag{4.1}$$

This distribution is often used for failure modes in continuous time models e.g. by Grunske et al. [GCW07] and Bozzano et al. [BCK+10b]. Continuous probability distributions are not directly expressible in a discrete time context as SAML. Nevertheless, using the temporal resolution Δt (see Def. (25)) it can be approximated using the geometric distribution as shown in Eq. (4.2). Here $P(X \leq k)$ computes the probability that the failure mode occurs within the first k time-steps of length Δt, i.e. within the time $t = k \cdot \Delta t$.

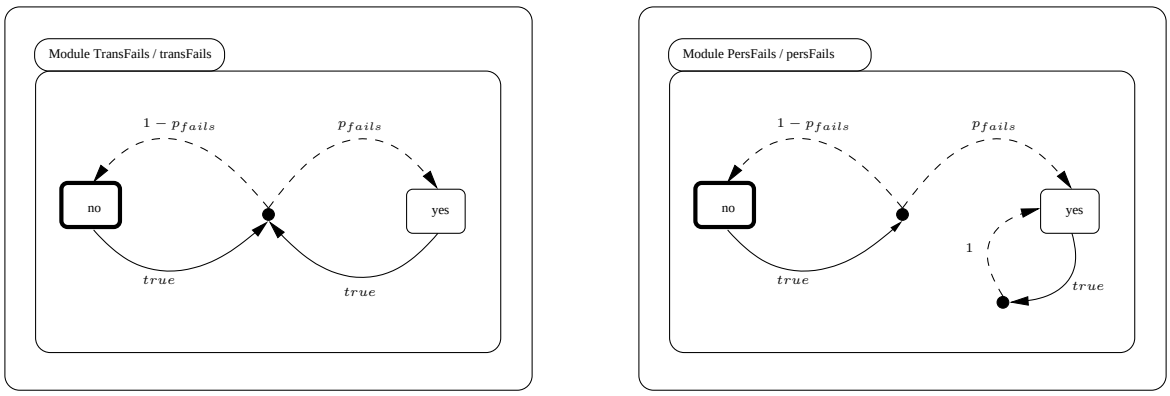
$$P(X \leq k) = 1 - P(X > k) = 1 - (1 - p)^k \tag{4.2}$$

Using the identity $e^x = lim_{n\to\infty}(1 + \frac{x}{n})^n$ the continuous exponential distribution can be approximated with the discrete geometric distribution as shown in Eq. (4.3). For longer time intervals, k approximates n and the smaller the basic time unit Δt is, the better this approximation becomes [GO10b].

$$1-e^{-\lambda t} = 1-lim_{n\to\infty}\left(1+\frac{-\lambda t}{n}\right)^n = 1-lim_{n\to\infty}\left(1-\frac{\lambda k \Delta t}{n}\right)^n \approx 1-(1-\lambda\Delta t)^k \tag{4.3}$$

[4]But there exist of course other failure modes, which are transient and should be modeled as per-demand probabilities resp. persistent failure modes, which should be modeled with per-time failure rates.

In SAML per-time failure mode occurrence patterns are modeled as failure modules as shown in Fig. 4.6 for transient (Fig. 4.6(a)) and persistent occurrence patterns (Fig. 4.6(b)). These are similar to the SAML variants of the purely qualitative failure modules. The main difference is that the non-deterministic choices in the failure modules in Fig. 4.4 and Fig. 4.5 are replaced with probabilistic transitions and the failure state *yes* is reached with the probability p_{fails} and the failure module stays in state *no* with probability $1 - p_{fails}$.



(a) Transient Per-Time Failure Module (b) Persistent Per-Time Failure module

Figure 4.6.: Quantitative Per-Time Failure Mode Modeling

Combining the result of Eq. (4.3) and Eq. (4.2), $\lambda\Delta t$ is an approximation for the probability p_{fails} in a per-time failure module and is called the per-step failure probability.

Definition 26. Per-Step Failure Probability
For a per-time failure mode with a failure rate λ and the temporal resolution Δt(both with compatible time units) the per-step failure probability p_{step} *is defined as:*

$$p_{step} = \lambda\Delta t$$

Failure modes which are not exponentially distributed can often be modeled via an approximation using a combination of multiple exponential distributions. Nevertheless, single exponential distributions are often used for per-time failure mode modeling in other safety analysis approaches as developed by Grunske et al. [GCW07], Bozzano et al. [BCK+10b] and also by Raiteri et al. [CR05, BPRW08a].

One important case of non-exponentially distributed failure mode occurrence are per-demand failure modes. In SAML it is possible to directly model per-demand failure occurrence patterns in addition to per-time, which is currently not supported in these other approaches.

Per-Demand Failure Mode Modeling

Occurrence patterns for a per-demand failure mode are more difficult to model than for per-time failure modes. It must be assured, that the failure effect can only appear at the moment of a *request* to the safety-critical component. The basic idea is to define a predicate *demand* for each possible per-demand failure mode. For each request, the activation condition of the failure module for the per-demand failure mode is then this *demand* predicate. This assures that the failure can occur only when the safety-critical system is requested.

One problem with the direct application of this approach is that the occurrence of the per-demand failure mode can only be observed one time-step after the demand. The problem with this is that the *effect* modeling is dependent on the state of the failure module. The observation of this state would then only be possible one time-step too late.

Nevertheless, this problem can easily be solved as shown in Fig. 4.7 (for transient failure modes). The idea is it to use the result of a *previous* probabilistic transition of the failure module to signal whether the current demand can be met and to make the current probabilistic transition in order to signal the outcome for the *next* demand. So if t_i specifies the time of the j^{th} demand, a transition of the failure module at time t_k with $k < i$ will decide the outcome at t_i. The demand at time t_i in turn will then decide the outcome at some time t_l where $l > i$.

To be able to cope with the very first demand, a new initial state is introduced to the failure module, the state *init*[5]. The activation condition of the transition leaving *init* is set to *true* which guarantees that this state is left in the first time-step. It reaches state *no* with probability $1 - p_{fails}$ and state *yes* with probability p_{fails}. The state that is reached after this first transition decides the outcome for the first demand to the possibly failing component. This general per-demand modeling for per-demand failure modes is shown in Fig. 4.7 for a transient failure module and in Fig. 4.8 for a persistent one.

Strictly speaking, this solution delays the occurrence possibility of a per-demand failure mode for one time-step after the beginning of a system run, as in the initial state of the per-demand failure module there is no information whether the first demand will succeed or not. If this is a problem, an additional *start* state for each module can be added. If the transition leaving this start state has activation condition *true* and is purely non-deterministic, i.e. has the probability 1 to reach any of the *real* initial states, there is no change in the overall computed probability.

Another possibility is to use the per-demand failure mode integration as described in [GO10b] and [GO10c]. This modeling introduces additional *undecided* states and requires rather complex changes of the module in which the failure effect is modeled. This potentially increases the state space and is much more difficult to conduct. In most cases, the per-demand integration is feasible the way described here which is simpler

[5]Most often modeled with the numeric value -1.

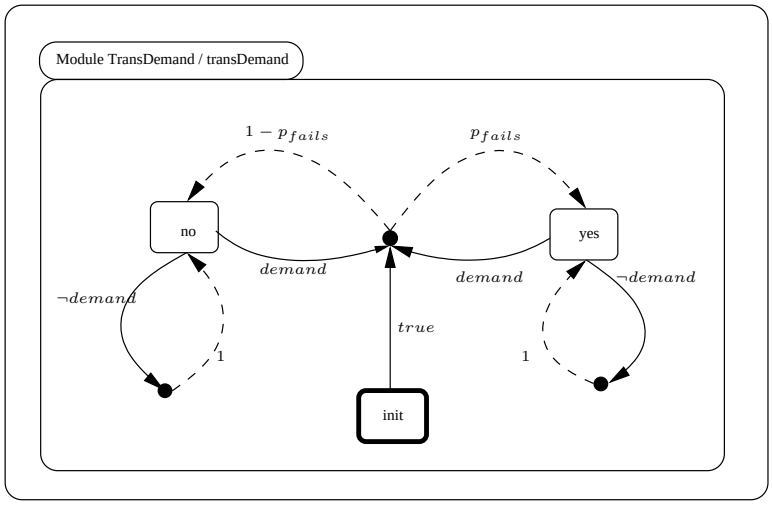


Figure 4.7.: Transient Per-Demand Failure Automaton

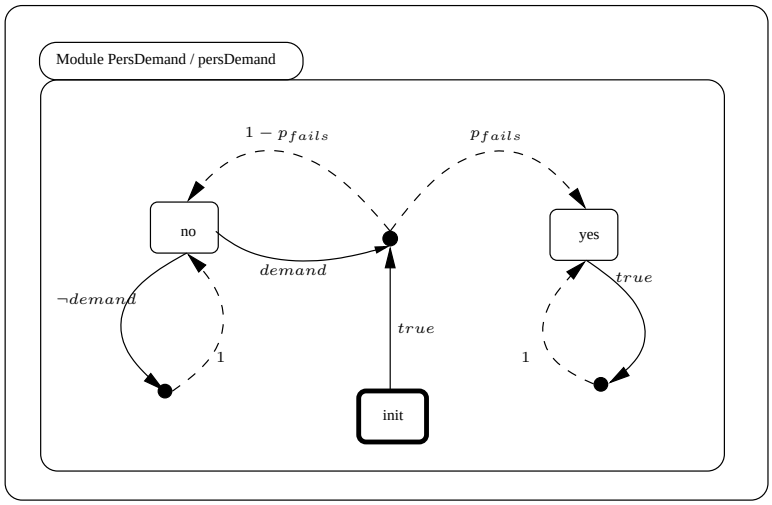


Figure 4.8.: Persistent Per-Demand Failure Automaton

than the outlined alternatives. Note, that this is a similar design choice to make for the per-time failure mode modeling, where the failure modules are generally modeled with *no* as initial state.

Case Study Model

These two different kinds of failure occurrence patterns are used for the modeling of the case study. A variety of failures modes is possible, it is assumed that a list of the relevant failure modes exists. The ones considered in the modeling and analysis are the following:

The sensors can omit a signal (*S1FailsSig*, *S2FailsSig*), making it impossible for one of the arithmetic units to process the data from the sensors correctly. The arithmetic units themselves can omit producing output data (*A1FailsSig*, *A2FailsSig*). The monitor can fail to detect that situation (*MonitorFails*), either switching if not necessary or not switching if necessary. The activation of the A2 unit may fail (*A2FailsActivate*) although the monitor sent the activation signal to A2. The failure occurrence patterns for these six failure modes are integrated into the SAML model of the case study for its safety analysis.

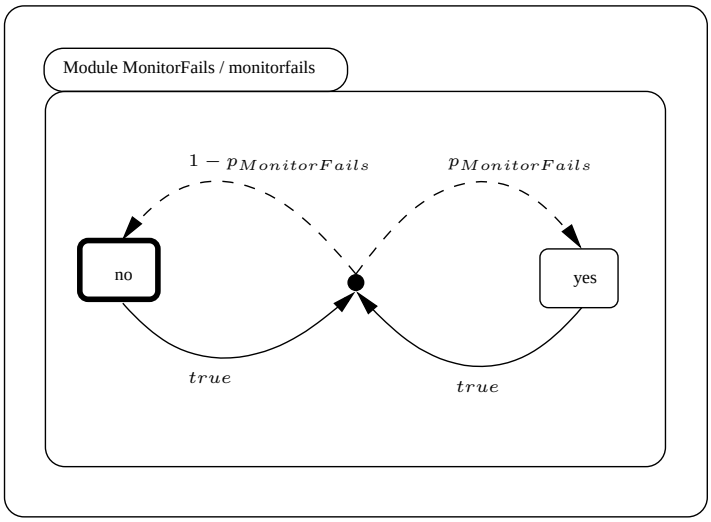


Figure 4.9.: *MonitorFails* Failure Occurrence Pattern

Example of a Per-Time Failure Mode As an example of a *per-time* failure mode, the modeling of the transient failure mode *MonitorFails* is presented. The per-time failure module is shown in Fig. 4.9. At any system step, the failure may appear or disappear depending on its per-step probability $p_{MonitorFails}$.

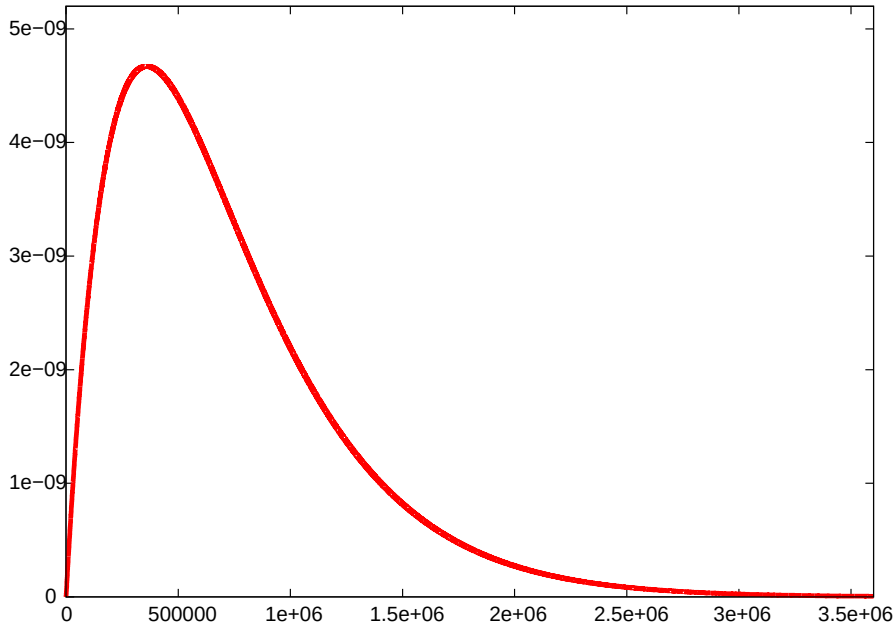
The failure rate for the *MonitorFails* is assumed to be $\lambda_{MonitorFails} = 1 \cdot 10^{-6} \frac{1}{h}$ which translates to a per-step probability of $p_{MonitorFails} = 2.7778 \cdot 10^{-12}$ for $\Delta t = 10ms$. The approximation errors for this per-time failure mode are shown in Fig. 4.10. in two time-vs-error (time in ms) plots. The absolute approximation error

$$\epsilon_{abs}(t) := |(1 - e^{-\lambda t}) - (1 - (1 - \lambda \Delta t)^k)|$$

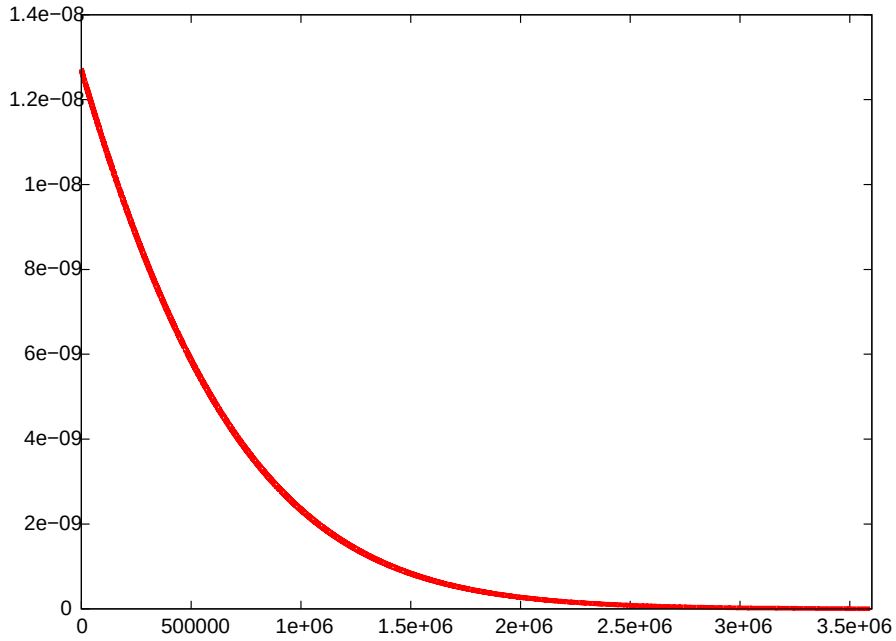
in Fig. 4.10(a), the relative approximation error

$$\epsilon_{rel}(t) := |\frac{\epsilon_{abs}(t)}{1 - e^{-\lambda t}}|$$

is shown in Fig. 4.10(b). Both decrease for longer run-times as expected from Eq. (4.3).



(a) Absolute Approximation Error



(b) Relative Approximation Error

Figure 4.10.: Approximation Error for Per-Time Failure Modeling

Example of a Per-Demand Failure Mode The failure mode *A2FailsActivate*, the failure to activate the A2 unit if required, can only appear if there is a request from the monitor and is therefore modeled as a per-demand failure mode. The corresponding failure module can be seen in Fig. 4.11. The predicate *demand* is defined as $arith2 = idle \wedge activate$, i.e. the backup arithmetic unit is in its *idle* state and should be activated because A1 does not deliver any more signal (predicate *activate* holds). The per-demand occurrence probability of this failure mode is modeled as $1 \cdot 10^{-7}$.

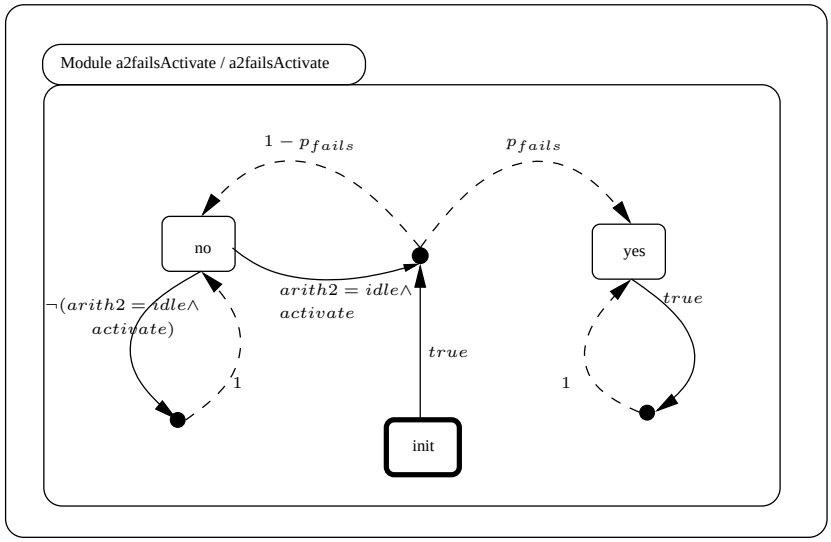


Figure 4.11.: *A2FailsActivate* Failure Occurrence Pattern

Additional Failure Modes For the analysis of the complete case study, the failure modules for the four remaining failure modes are also modeled. *S2FailsSig* is modeled as a per-time failure mode analogous to *S1FailsSig*, both with a failure rate of $\lambda_{S1FailsSig} = 1 \cdot 10^{-2} \frac{1}{h}$. The failure modes *A2FailsSig*, *A1FailsSig* are modeled as per-time failure modes. The failure rate for *A2FailsSig* is also $1 \cdot 10^{-2} \frac{1}{h}$, the failure rate for *A1FailsSig* is $1 \cdot 10^{-6} \frac{1}{h}$, as it is assumed that the primary unit A1 is of better quality than the secondary unit. With a temporal resolution $\Delta t = 10ms$, this translates to a per-step failure probability of $2.7778 \cdot 10^{-8}$ for the failure rates $1 \cdot 10^{-2} \frac{1}{h}$ and $2.7778 \cdot 10^{-12}$ for the failure rates $1 \cdot 10^{-6} \frac{1}{h}$.

The modeling presented here only defines the various occurrence patterns and therefore *when* a certain failure mode can occur and how probable that is. This forms the foundation of the modeling of the effects of the occurrence of the failure modes, i.e. *what* happens if the failure mode occurs.

4.5.3. Failure Effect Modeling

The failure modules specify the occurrence pattern of a failure mode but not its effect. Effects are not modeled using additional parallel SAML modules, but by the direct integration into the modules of the functional system. This integration is done in a *conservative* way, which is characterized by the following:

1. The failure effect cannot occur if the failure mode is not present.
2. The original behavior[6] of the SAML module is preserved, if the failure mode does not occur.

Conservative Integration

To achieve these aspects of a conservative integration in a formal system model, the effects of the failure modes have to be integrated in a special way. For purely qualitative failure mode modeling, rules to achieve this have been introduced in [OGR07]. These rules specify how the failure effect integration can be kept conservative for purely qualitative statechart modeling. They are adapted to quantitative modeling in SAML as follows, where $fails_i$ refers to the state variable of a failure module for the failure mode to integrate:

1. Start with the functional SAML system model.
2. Model the failure modules accordingly to their type as parallel SAML modules as described in Section 4.5.2.
3. To model the direct effects of a failure mode
   a) new values for the state variables (and update rules for successor states from these) may be introduced
   b) additional update rules may only be introduced if their activation condition is of the form $\phi \wedge fails_i = yes$ and the activation condition of the "original" update rules then must be changed to $\phi \wedge fails_i \neq yes$
4. The rest of the functional SAML system model is not changed, in particular not the initial states of the state variables

A very important aspect of the modeling of the failure occurrence patterns, which had not explicitly been mentioned in [OGR07] is the possibility that the failure does not occur at all. For a SAML failure module this means that the probability that the state *no* is not left will always be non-zero. If this holds and the above rules are followed, the failure mode integration is conservative. This is formulated in Lemma 3. Note that instead of the symbolic values for the state variable $fails_i$, the integral representations -1 for *init*, 0 for *no* and 1 for *yes* are used.

[6] The paths of the model of the system

Lemma 3. Conservative Integration

Let M be a SAML model with the set of state variables Var and M' be the SAML model in which the effect of a failure mode γ_i is integrated into M adhering to the rules above. Let F_i be the corresponding failure module, $fails_i$ the state variable that indicates that the failure mode is absent ($fails_i \leq 0$) or present ($fails_i > 0$). Let τ_{MDP} be the MDP corresponding to M and τ'_{MDP} be the MDP corresponding to $M'||F_i$. Then:

$$\forall \pi \in Paths(\kappa(\tau_{MDP})) : \exists \pi' \in Paths(\kappa(\tau'_{MDP})) : \pi \equiv \pi'|_{Var} \tag{4.4}$$

where $\pi'|_{Var}$ is the projection of π' onto the state variables $v \in Var$

proof see p. 168

It is important to note that this notion of conservative integration applies only to the presence of the same paths and traces. Typically the probabilities will change by the introduction of the failure mode modeling, as this modeling is probabilistic. In general this is not a problem, as the relation of the probabilities in the original model and the equivalent paths in the extended system model is not changed.

If more than one failure mode is integrated into the model, there exists the possibility that the modeling of effects of one failure mode γ_i is masked by the modeling of the effects of another failure mode γ_j. This situation can be resolved at modeling time by specifying a priority on the failure modes, e.g. always first considering γ_i before γ_i. Another possibility to cope with this situation is to use non-deterministic modeling in such a way that the adversary resolves this situation at the time of the analysis. Which approach is better depends on the nature of the modeled system and on the amount of information on the failure effect that is available.

Lemma 3 also applies to the successive integration of all failure modes. This means that each successive integration is conservative. Therefore the original behavior of the system model is still possible after each of the failure mode occurrence patterns and failure mode effects modeling is integrated. The resulting SAML model in which all failure modes are integrated conservatively, is called the *extended system model*. If non-deterministic modeling is used or the priority of failure effects modeling is always the same, then all possible orderings of conservative failure effect integration as described above produce isomorphic extended system models.

Definition 27. Extended System Model

Let M be a functional SAML model, Δ the set of failure modes γ_i with corresponding failure module F_i and $(M')_i$ the conservative integration of the failure mode γ_i into the model M' adhering to the construction rules. Then $((M)_1 \ldots)_n||F_1|| \cdots ||F_n$ is called the extended system model M^+.

Case Study Model

The failure effects of the failure modes of the case study can now be integrated conservatively. For illustration purpose, the changes that are necessary for the integration of the per-time failure mode *MonitorFails* are presented here. A more detailed modeling of the case study and the integration of the remaining failure modes is given in Section 7.2. The original SAML model of the monitoring unit is shown in Fig. 4.12.

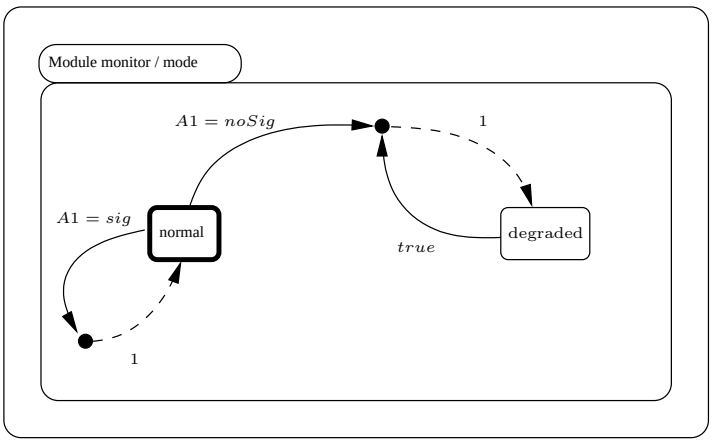


Figure 4.12.: Functional Model of Monitoring Unit

The monitoring unit stays in state *normal* as long as A1 is in state *sig* and produces an output signal. If A1 does not produce any more output signal and switches to state *noSig*, the monitoring unit enters state *degraded*. This signals the off-switching of A1 and the activation of A2. The state *degraded* cannot be left any more, a reset of the system would require an external trigger and is not part of the modeling presented here.

The integration of the effect of *MonitorFails* is shown in Fig. 4.13. The effect of this failure mode can be either the switch to *degraded* without being necessary or no switch although A1 still delivers a signal. These effects are modeled by introducing new transitions. If the failure mode occurs, the state *normal* can be left although A1 is in *sig* or *normal* is not left although A1 is in *noSig*.

As described in the rules for failure mode integration, the original transitions are now labeled with the conjunction of their original activation condition and $monitorfails = no$. This means the original behavior is preserved as long as the failure mode is absent. If the failure mode is present, the monitoring unit makes a non-deterministic choice between staying in state *normal* or entering state *degraded*, independent of the actual state of A1. If the monitoring unit is already in state *degraded*, the failure mode occurrence has no effect. Note that the failure probability is modeled in the occurrence pattern, i.e. the corresponding failure module.

The integration of a per-demand failure mode is illustrated with the *A2FailsActivate* failure mode. The effect of this failure mode is that the intended activation of the second

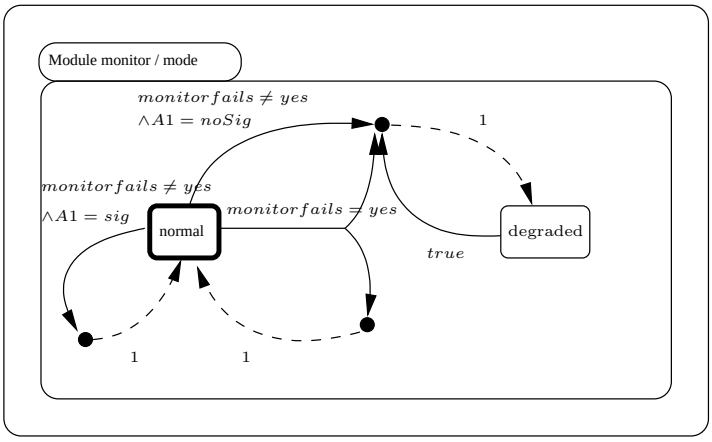


Figure 4.13.: *MonitorFails* Failure Effect Modeling

arithmetic unit does not succeed. The original modeling of the secondary arithmetic unit is shown in Fig. 4.14. It stays in state *idle* as long as the predicate *activate* does not hold (the monitoring unit is in state *normal*). Once the activation signal is received, it enters the *sig* state and stays in that state as long as there is a signal. If there is no more signal it enters the state *noSig*.

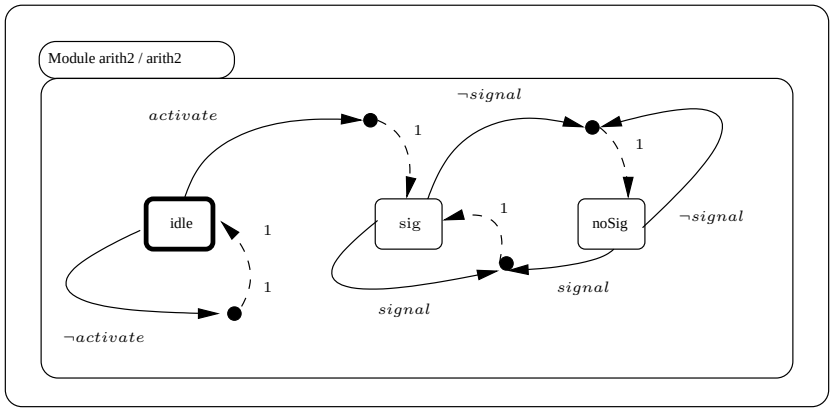


Figure 4.14.: Functional Model of A2

The integration of *A2FailsActivate* into this model can be seen in Fig. 4.15. The modeling of the failure effect is straightforward: whenever the unit A2 should be activated, it will enter state *sig* only if the failure mode did not appear, i.e. the predicate $actFailure \neq yes$ holds. If the predicate *activate* holds, but the failure mode occurred ($actFailure = yes$ holds), the *idle* state is not left and as result the activation of A2 fails. This models the same behavior as if *activate* did not hold at all. As explained in the last section, the demand in this case is that A2 is in state *idle* and *activate* holds.

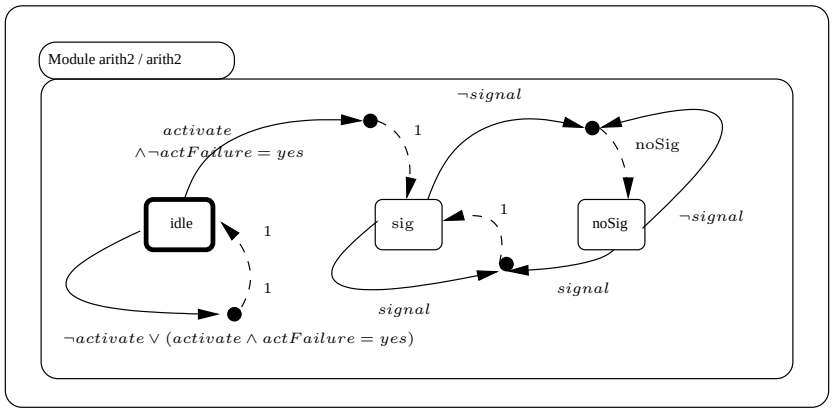


Figure 4.15.: *A2FailsActivate* Failure Effect Modeling

A possible state sequence of the original module of *arith2* is shown in Table 4.1. Here the second arithmetic unit is in its idle state until the *activate* signal is sent at $t = 3$ by the monitoring unit. After then it enters its *sig* state where it delivers an output signal.

time-step	1	2	3	4	5
arith2	idle	idle	idle	sig	sig
activate	false	false	true	false	false

Table 4.1.: State Sequence of the Functional System Model

Table 4.2 shows a state sequence of the *arith2* module and the failure module for *A2FailsActivate* of the extended system model. It shows that if there is no failure mode occurrence, the state sequence of the *arith2* module stays exactly the same as before. The failure module enters the state *no*, i.e. absence of the failure mode at the first demand after its initial state, therefore there is no failure mode effect at time-step 3 when the *demand* ($arith2 = idle \wedge activate$) occurs.

time-step	1	2	3	4	5
A2FailsActivate	*init*	*no*	***no***	*yes*	*yes*
arith2	idle	idle	idle	**sig**	sig
activate	false	false	**true**	false	false
demand	false	false	**true**	false	false

Table 4.2.: Trace of Extended System Model without Failure Mode Occurrence

The effect on the state sequence of the *arith2* module of the occurrence of the *A2FailsActivate* failure mode is shown in Table 4.3. Here the failure module enters its *yes* state immediately after the initial state which results in the occurrence of the

failure mode when the *demand* occurs at time-step 3. The effect of the occurrence of the failure mode is that *arith2* stays in the state *idle* and therefore fails to deliver a signal.

time-step	1	2	3	4	5
A2FailsActivate	*init*	*yes*	***yes***	*no*	*no*
arith2	idle	idle	idle	**idle**	idle
activate	false	false	**true**	false	false
demand	false	false	**true**	false	false

Table 4.3.: Trace of Extended System Model with Failure Mode Occurrence

For the analysis of the complete case study, the remaining failure modes were integrated analogously. The effect of the occurrence of *S1FailsSig* or *S2FailsSig* is to switch either sensor S1 or S2 from state *sig* to state *noSig*. The effect of the failure modes *A1FailsSig*, *A2FailsSig* is to switch either A1 or A2 to state *noSig* from their *idle* or *sig* states, independent of the presence of a signal from one of the input sensors, for the description of the rest of the modeling see Section 7.2.

This forms the extended system model of the example case study as a SAML model. All the necessary information for a safety analysis is contained in it. Methods for accurate formal qualitative and quantitative model-based safety analysis of such a model are introduced in the next chapter.

4.6. Related Work

The presented modeling is a direct extension of the modeling presented in the works of Ortmeier et al. on DCCA [ORS05, ORS06] and the formal failure mode modeling presented in [OGR07, Ort06]. Previously only qualitative modeling was used and therefore the presented quantitative modeling greatly extends the accuracy of the models with respect to physical environment modeling and especially failure mode occurrence pattern, in particular the distinction between per-time and per-demand failure modes.

Other failure injection based approaches with similar failure effects modeling in which the effect of a failure is integrated a system model were developed in the ISAAC [rBB+06] project for SCADE by Abdulla et al. [ADS+04] and in the ESAC [B+03] and ISAAC projects for the FSAP / NuSMV-SA tool by Bozzano et al. [BV03b]. The integration into SCADE models was in an ad-hoc fashion, for the FSAP/NuSMV-SA tool, a library of possible failure modes was created. This library presently includes different types of failure modes for single bits, e.g. bit-inversion, stuck at 0 or 1 or random non-deterministic value assignment. A very interesting aspect is the automatic integration of the failure mode effects into the formal system model. Probabilistic modeling is not supported and therefore only qualitative analyses are possible.

Quantitative failure mode modeling was studied by Grunske et al. in [GCW07] and within the AVACS [AVA03] project by Böde et al. [BPRW08a]. Both employ continuous

failure mode modeling with failure rates which does not allow for per-demand failure mode modeling. The integration of the failure effects is similar to the methods presented here and the conservative integration principle could also be extended to these approaches for failure mode modeling. Nevertheless in these approaches there is no explicit failure occurrence pattern modeling. On the one hand, this reduces state space as there is no need for additional state variables modeling the occurrence. But on the other hand, this reduces the expressiveness of the modeling or at least increases the complexity of the models, as more has to be integrated into the system model itself.

A very recent approach to formal safety analysis was developed in the COMPASS project [BCK+09a] where a subset of the architecture analysis and design language (AADL) [SA04] and its error annex [SA06] is used. It allows for the specification of failure modes and is formalized in the SLIM language described by Bozzano et al. in [BCK+09b, BCK+10b]. This is a further development based on the FSAP/NuSMV-SA tool. It allows for the specification of hybrid systems with constant continuously changing variables and failure modes with a continuous time model. Analogously to the FSAP/NuSMV-SA tool, the integration of the failure modes into the formal system can be done automatically. As only continuous modeling is provided, no per-demand failure mode modeling is supported. Per-demand failure mode modeling is one part of the AADL error annex not supported by SLIM. The usage of the established modeling language AADL is very interesting. A formal semantics of AADL and its error annex which allows for the integration of per-time and per-demand failure mode modeling would be a promising application of using SAML as an intermediate language.

Summary

This chapter introduced a modeling guideline to construct extended system models in SAML for formal model-based safety analysis. Extended system models consist of hardware, software and environment modeling. The environment model specifies the physical behavior of the system's surroundings and the failure modes that are possible in the system. The existing formal failure modeling was extended with probabilistic failure mode modeling. The modeling of different types of probabilistic failure modes, namely per-time and per-demand was introduced and the conservative integration of the failure modes was shown which provably retains the original behavior of the system after the integration of the failure effects. The modeling approach was illustrated using a case study taken from the safety analysis literature.

5. Formal Safety Analysis

There are two ways to write error-free programs; only the third one works.

(Alan J. Perlis)

This chapter introduces methods which allow for formal model-based safety analysis of extended system models specified in SAML. Qualitative analyses allow for the computation of critical combinations of failure modes and deduction of temporal ordering sequences of these failure modes. Extensions of existing qualitative safety analysis methods are developed which allow for analysis of self-healing systems. New methods can deduce dynamic fault trees (DFT) information directly from a SAML extended system model.

This chapter also introduces new, quantitative safety analysis methods which allow for computation of the occurrence probability of a system hazard with much higher accuracy than previously possible. The methods use current state-of-the art probabilistic model-checking tools to compute hazard probabilities directly from SAML extended system models.

All analyses are formulated as temporal logic proof obligations which can be verified automatically. The proof obligations are constructed in such a way that the resulting safety analysis is guaranteed to be sound and complete. The application of the analysis methods is illustrated on the case study introduced in the previous chapter.

Section 5.1 gives a motivation for the formal safety analysis techniques. Section 5.2 presents qualitative safety-analysis techniques which extend the existing qualitative safety analysis methods. Section 5.3 presents a new quantitative model-based safety analysis method. Related work is discussed in Section 5.4.

5.1. Motivation

From the safety engineer's point of view the most important questions in the development of a safety critical system are:

- "What can cause a hazard?"
- "What is the probability that a hazard occurs?"

The model-based safety analysis methods presented in this chapter serve to answer these questions correctly. If a formal system model is available, formalizing them as proof obligations can be very convenient, as model checking tools can compute the answers fully automatically.

Nevertheless, the correct formalizing of these properties is not trivial. The main considerations for the formalization are *correctness* and *completeness* of the analysis. For safety analysis, completeness refers to the fact that no cause of the hazard is forgotten. This means that each combination of failure modes is found which *can* cause the hazard. Correctness on the other hand means that for each combination of failure modes that can cause the hazard, *there actually exists* a run of the system on which it causes the hazard.

A safety analysis technique should be both complete and correct. If it is correct, it is never too optimistic and will never label an unsafe system as safe, which might render the system more dangerous than presumed. If the analysis is complete, then it is not too pessimistic and will not label a potentially safe system as unsafe, which might prevent its realization. So proving completeness and correctness of an analysis technique is important for the actual applications, especially if a system is to be certified. If the applied analysis is not considered correct, a certification may not be granted.

In addition to find failure mode combinations that can cause a hazard, it is also very interesting to analyze temporal dependencies of the failure modes. This means that two failure modes may only cause the hazard if they occur in a given sequence. If for example two failure modes γ_1, γ_2 can only cause a hazard if γ_1 appears *before* γ_2, the system can be made safer by the introduction of a warning or entering a *safe mode* if γ_1 occurs.

Even more beneficial is the extension to quantitative safety analysis methods which compute the occurrence probability of a hazard. In traditional analysis techniques, this is often conducted a-posteriori by a quantitative estimation based on the qualitative analysis *results*. New developments in the field of probabilistic model checking and the increase of available computing resources now allow for the direct quantitative analysis of the extended system model itself. Such an approach gives much more precise and reliable results, because all dependencies inherent in the system are *automatically* considered.

Overall, the qualitative analysis methods give a "yes or no" answer whereas quantitative analysis methods provide a more differentiating and precise answer. Nevertheless, both kinds of analysis methods are necessary. Firstly because qualitative analysis are

much more efficient and should therefore be used were adequate. Secondly, for some aspects the naïve reasoning that probability zero means impossible is wrong.

In general for a model-based safety analysis one will (i) built an extended system model (as described in the previous chapter) (ii) analyze the qualitative properties of the system (potentially using the qualitative results to improve the system) and (iii) analyze the quantitative properties of the system.

5.2. Qualitative Model-Based Safety Analysis

Qualitative safety analysis techniques are of *possibilistic* nature. They compute whether and how failure modes can cause a hazard in the worst case. These analyses are typically the first which are conducted on an extended system model. One example is the deductive cause-consequence analysis (DCCA). In this dissertation it forms the basis for further qualitative analysis and is also the starting point of the quantitative safety analysis methods presented in this chapter.

5.2.1. Deductive Cause Consequence Analysis

DCCA is an approach to deduce information about critical failure mode combinations from an extended system model. It forms the basis for the formal model-based safety analysis of SAML models. It computes whether a subset of the set of all failure modes is critical with respect to a hazard. Critical here means that the combined occurrence of the failure modes *can* actually cause the hazard. The result of a DCCA is a set of all critical combinations of failure modes. This information can be used to implement risk-reducing measures. If from every such critical combination at least one failure mode can be prevented, then the occurrence of the hazard is impossible[1]. DCCA has been introduced by Ortmeier et al. in [ORS05] and has been successfully applied to several case studies [ORS06, ORS05, GOR08]. In the following, the notion of a SAML model M or extended system model M^+ is used in the formal specifications, instead of τ_{MDP}, e.g. $\kappa(M^+)$ actually refers to the embedded Kripke structure of the corresponding MDP of M^+. From the context the meaning is always clear. The definitions (28)-(29) and the following description are adapted from [ORS05].

Definition 28. DCCA / Minimal Critical Set

For an extended system model M^+ and a set of failure modes Δ, a subset of component failures $\Gamma \subseteq \Delta$ is called critical *wrt. a system hazard H if*

$$\kappa(M^+), s_0 \models \mathbf{E}\,[\overline{\Gamma}\,\mathbf{U}\,H], \text{ where } \overline{\Gamma} := \bigwedge_{\delta_i \in \Delta \setminus \Gamma} \neg\delta_i$$

Γ *is called* minimal critical *if is has on proper critical subset.*

[1] This is often called the "minimal cut set theorem" whose name is derived from FTA.

Informally this formula states that a set of failure modes Γ is critical, if there exists a path of the system on which no failure mode which is not in Γ occurs ($\overline{\Gamma}$), until finally the hazard H occurs. The predicates δ_i are defined to be *true* if the failure module corresponding to the failure mode γ_i is in state *yes* and $false$ otherwise. If modeled as described in the previous chapter, in this situation only the failure modes in Γ and their effects could have been the cause for H.

DCCA generalizes functional correctness, failure modes and effect analysis (FMEA) and minimal cut sets of fault tree analysis (FTA). Functional correctness corresponds to the analysis whether the empty set $\emptyset$ is critical. If that is true, the system hazard can occur without any failure mode as its cause and therefore the system is considered functionally incorrect. The main focus of FMEA is the analysis whether single failure modes can cause the hazard. This can be expressed in DCCA via the analysis of failure sets with a single failure mode.

Compared to FTA, which is widely using in industrial practice (see Section 2.2.1), DCCA has several advantages. It is proven to be both *correct* and *complete* [ORS06][2], therefore it is guaranteed never to produce worse results than a (formal) FTA. Fault tree analysis can be complete but it is very difficult to conduct correctly and therefore can be too pessimistic. This is because fault trees are in general not compositional as shown by Ortmeier and Schellhorn [OS06]. DCCA and FTA can be combined, by using results from an (informal) FTA as the starting point of the analysis of minimal critical sets, i.e. using the results of an informal FTA as first candidates for minimal critical sets. This typically reduces the number of necessary proof obligations for a complete DCCA significantly.

Although the number of possible combinations of failure modes for a DCCA is exponential (2^n for n failure modes), the actual number in a concrete example is most often much lower. This results from the fact that criticality is monotone. Therefore if a set Γ_i is critical, then any superset $\Gamma_i \subset \Gamma_j$ is also critical and does not have to be analyzed, see Ortmeier's dissertation [Ort06].

Definition 29. Complete DCCA

For an extended system model M^+ and a set of failure modes Δ, a DCCA is called complete *with respect to H if each minimal critical set has been found. The result is the set of minimal critical sets of the system:*

$$MCSS(M^+, \Delta, H) := \{\Gamma | \Gamma \textit{ is critical} \wedge \nexists \Gamma' \subset \Gamma : \Gamma' \textit{ is critical} \}$$

While DCCA computes all combinations of failure modes that can cause a hazard, it does not discern between different sequences of these failure modes. For this, a new method which analyzes the necessary orderings of failure modes in a minimal critical sets in order to potentially cause the hazard has been developed in this dissertation.

[2]Complete here means that no combination of failure modes that is a possible causes for the hazard is missed. Correct means that for each critical combination there actually exists a system run on which the hazard is caused.

5.2.2. Ordered Minimal Critical Sets

Failure modes that are causes for hazards can occur at different times on a system trace. If failure modes are in the same minimal critical set, then their appearance on a trace is considered critical, regardless of the sequence in which they occur. The results of a DCCA can be augmented by analyzing whether there exits a necessary temporal ordering for the failure modes to be critical.

Failure Mode Ordering Relation

If a critical combination of failure modes for a system hazard is established, i.e. a minimal critical set is found by conducting a DCCA, then a partial order relation on the failure modes can be defined. This relation captures the temporal ordering between two failure modes in a minimal critical set[3].

Definition 30. Before Order / Strict Before Order
For an extended system model M^+, a set of failure modes Δ, a predicate logic hazard H, and a minimal critical set $\Gamma \subseteq \Delta$, two failures modes $\gamma_1, \gamma_2 \in \Gamma$ are in

- **before order:** $\gamma_1 \preceq_\Gamma \gamma_2$*, if on all paths π of $\kappa(M^+)$ on which the failures in Γ were the cause for the hazard H, the first occurrence of the failure mode γ_1 is not after the first occurrence of the failure mode γ_2*
- **strict before order:** $\gamma_1 \prec_\Gamma \gamma_2$*, if on all paths π of $\kappa(M^+)$ on which the failures in Γ were the cause for the hazard H, the failure mode γ_2 does appear after the failure mode γ_1*

If such an ordering exists, i.e. the relations are not empty, concentrating the risk-reducing measures on γ_1 can be very beneficial. Alternatively, watchdogs can be used to detect the occurrence of γ_1 and if possible the system can enter a degraded "safe mode", in which no more safety-critical functions are executed. Based on these two relations an additional relation for temporal ordering can be defined which describes the simultaneous occurrence of two failure modes:

Definition 31. Simultaneous Order
For an extended system model M^+, a set of failure modes Δ, a predicate logic hazard H and a minimal critical set $\Gamma \subseteq \Delta$, two failure modes $\gamma_1, \gamma_2 \in \Gamma$ are in **simultaneous order** $\gamma_1 \sim_\Gamma \gamma_2$*, if on all paths π of $\kappa(M^+)$ on which the failures in Γ were the cause for the hazard H, the first occurrence of the failure mode γ_1 is at the same time as the first occurrence of the failure mode γ_2*

$$\sim_\Gamma := \{(\gamma_i, \gamma_j) | \gamma_i \preceq_\Gamma \gamma_j \wedge \gamma_j \preceq_\Gamma \gamma_i\} \tag{5.1}$$

[3]This is not dependent on DCCA, but can also be employed to minimal cut sets of a fault tree analysis.

Note that the temporal relations are not total, because for two failure modes γ_1 and γ_2 from the same minimal critical set, different traces may exist on which they are cause the hazard, but their ordering is different. This is often the case with transient failures, as they can appear and disappear totally at random.

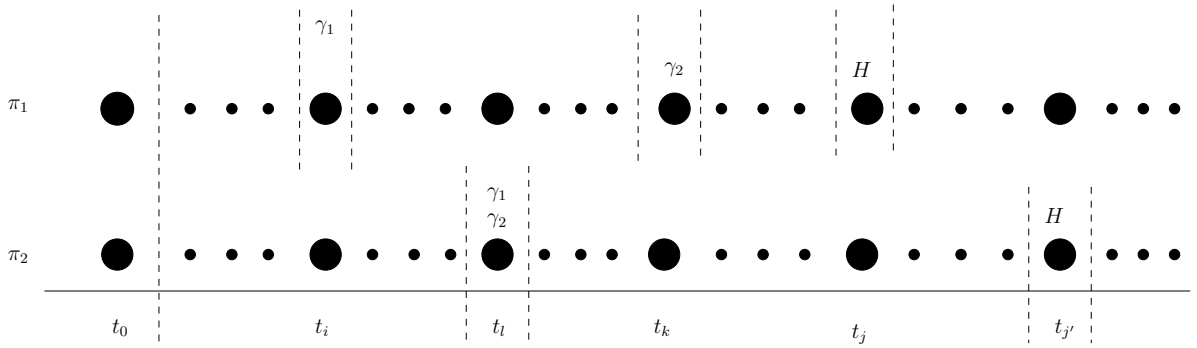


Figure 5.1.: Example Traces with Before Ordering between γ_1 and γ_2

In Fig. 5.1 two different possible traces of a system and the first occurrences of the failure modes and the hazard are shown. In both traces the hazard H finally occurs, either at time-step t_j or at $t_{j'}$ and the minimal critical set Γ consists of the two failure modes γ_1 and γ_2. In the upper trace, γ_1 occurs at time t_i and γ_2 at some time t_k afterwards. In the lower trace, both failure modes occur at the same time-step t_l for the first time. From these traces, $\gamma_1 \not\prec_\Gamma \gamma_2$, $\gamma_1 \not\sim_\Gamma \gamma_2$ and $\gamma_2 \not\preceq_\Gamma \gamma_1$ can be deduced. $\gamma_1 \not\prec_\Gamma \gamma_2$ because in the lower trace the first occurrence of both failure modes is at the same time, $\gamma_1 \not\sim_\Gamma \gamma_2$ and $\gamma_2 \not\preceq_\Gamma \gamma_1$ hold because in the upper trace the first occurrence of γ_2 is after γ_1. If no other trace exists on which γ_1 and γ_2 cause the hazard, $\gamma_1 \preceq_\Gamma \gamma_2$ would hold.

All these partial temporal ordering relations can be deduced using temporal logic formulas, which allow for the automatic computation of the relations from an extended system model expressed in SAML. This deductive computation of ordered minimal critical sets was introduced in [GOR08].

Deductive Failure Order Analysis

To deduce the temporal ordering of the failure modes directly from an extended system model, it is necessary to analyze all the paths on which the combination of the failures modes in Γ is the cause of the hazard. For each minimal critical set Γ, these are all paths on which the non-critical failure modes do not appear before the occurrence of the hazard. This can concisely be expressed in linear time logic (LTL). CTL is not suited for this, as we neither want to express something about a single trace of the system nor about all traces of the system, but about the subset of traces on which the failure modes of a selected minimal critical set caused the hazard. For each of the ordering relations, a

LTL formula is specified which computes whether the specified temporal ordering exists for two failure modes of a minimal critical set.

Lemma 4. Deductive Before Ordering / Strict Before Ordering
For an extended system model M^+, a set of failure modes Δ, a predicate logic hazard H, a minimal critical set Γ and two failure modes $\gamma_1, \gamma_2 \in \Gamma$:

$$\begin{aligned} &\gamma_1 \preceq_\Gamma \gamma_2 \Leftrightarrow \\ &\kappa(M^+), \pi \models \quad (\overline{\Gamma}\mathbf{U}H) \rightarrow [(\neg H \wedge \neg\gamma_1 \wedge \neg\gamma_2) \\ &\qquad\qquad \mathbf{U}(\gamma_1 \wedge ((\neg H \wedge \neg\gamma_2) \\ &\qquad\qquad\quad \mathbf{U}((\gamma_2 \wedge \neg H) \wedge \mathbf{F}\, H)))] \end{aligned} \tag{5.2}$$

$$\begin{aligned} &\gamma_1 \prec_\Gamma \gamma_2 \Leftrightarrow \\ &\kappa(M^+), \pi \models \quad (\overline{\Gamma}\mathbf{U}H) \rightarrow [(\neg H \wedge \neg\gamma_1 \wedge \neg\gamma_2) \\ &\qquad\qquad \mathbf{U}((\gamma_1 \wedge \neg\gamma_2 \wedge \neg H) \wedge \\ &\qquad\qquad\quad \mathbf{X}((\neg H \wedge \neg\gamma_2) \\ &\qquad\qquad\qquad \mathbf{U}((\gamma_2 \wedge \neg H) \wedge \mathbf{F}\, H)))] \end{aligned} \tag{5.3}$$

proof see p. 170

The left part of the implications $(\overline{\Gamma}\mathbf{U}H)$ describes those paths of the extended system model on which the failure modes in Γ are the cause of the hazard (i.e. no failure mode in $\Delta \setminus \Gamma$ appears before H). The right part of the implication describes the relation of the failure modes on the path. For the *before* ordering it is assured that the first occurrence of γ_2 is *not after* the first occurrence of γ_1 (Eq. (5.2)). For the *strict before* relation, the temporal *next* ($\mathbf{X}$) operator describes the situation that at least one time step passed after the occurrence of γ_1 before the first occurrence of γ_2 (Eq. (5.3)).

The relation $\sim_\Gamma$ can be checked either if for two failure modes $\gamma_1 \preceq_\Gamma \gamma_2 \wedge \gamma_2 \preceq_\Gamma \gamma_1$ holds or by checking the proof obligation in Eq. (5.4). In this proof obligation it is assured that the very first occurrence of both failure modes under consideration happens at the same time and before the first occurrence of the hazard.

Lemma 5. Deductive Simultaneous Order
For an extended system model M^+, a set of failure modes Δ, a predicate logic hazard H,a minimal critical set Γ and two failure modes $\gamma_1, \gamma_2 \in \Gamma$

$$\begin{aligned} &\gamma_1 \sim_\Gamma \gamma_2 \Leftrightarrow \\ &\kappa(M^+), \pi \models \quad (\overline{\Gamma}\mathbf{U}H) \rightarrow [(\neg H \wedge \neg\gamma_1 \wedge \neg\gamma_2) \\ &\qquad\qquad \mathbf{U}((\gamma_1 \wedge \gamma_2 \wedge \neg H) \wedge \mathbf{F}\, H)] \end{aligned} \tag{5.4}$$

proof see p. 171

All the described proof obligations are formulated for two failure modes. This means the temporal relation between two failure modes can be computed at a time. If more than two failures are included in a minimal critical set, then the relations must be computed for all pairs of failures, i.e. $\binom{n}{2}$ combinations for n failures. All these combinations can easily be generated automatically from a given minimal critical set of failures modes.

Application of DCCA and Ordering Analysis

To illustrate a qualitative analysis using DCCA and the deductive ordering analysis, the case study presented in the previous chapter is used. The six possible failure modes in this case study are sensor signal omitting (*S1FailsSig*, *S2FailsSig*), malfunction of the arithmetic units (*A1FailsSig*, *A2FailsSig*), failure of the watchdog monitor (*MonitorFails*) which results in either unnecessary switching of arithmetic processing units or missing a necessary switch and the failure of the activation of the backup arithmetic unit (*A2FailsActivate*). Using DCCA to analyze the case study presented in Section 4.2 with the hazard "no output produced" results in the following minimal critical sets:

- $\Gamma_1 := \{A1FailsSig\}$
- $\Gamma_2 := \{A2FailsSig, MonitorFails\}$
- $\Gamma_3 := \{MonitorFails, A2ActivateFails\}$
- $\Gamma_4 := \{MonitorFails, S2FailsSig\}$
- $\Gamma_5 := \{S1FailsSig, S2FailsSig\}$

The deductive ordering analysis of these five minimal critical sets shows that *MonitorFails* $\prec_{\Gamma_3}$ *A2ActivateFails* holds. Therefore the failure modes in the set {*MonitorFails*, *A2ActivateFails*} can only cause the system hazard if the monitor fails *before* the activation of A2 fails. The critical situation arises when the monitor misdetects a malfunction of A1, sends the activation signal but A2 does not get properly activated.

The set Γ_2 is critical when the monitor misdetects a malfunction of A1, the activation of A2 succeeds, but then A2 fails. A similar situation occurs if S2 fails after the activation of A2 (Γ_4). Here A2 does not receive an input signal from its only signal source S2. The set Γ_5 is critical because if both input sensors fail, then no more output signal can be produced. These three sets do not exhibit any temporal ordering.

The surprising result is the fact that Γ_1 contains only *A1FailsSig* as a single point of failure. As the system is built with A2 as a hot spare unit, this should not happen. Luckily, verification tools provide counterexamples for violated proof obligations which can be further analyzed to identify the flaw. A closer analysis of this situation reveals

that when A1 fails, the system needs a short amount of time to activate A2. In this time, no output signal is produced and therefore *A1FailsSig* is considered a single point of failure.

Nevertheless omitting the signal in this situation is only temporary and the system gets operational again. This can be seen in the state sequence shown in Table 5.1. Here the A1 unit fails to deliver a signal at causing the temporary hazard. This is observed by the monitoring unit which activates A2. When A2 starts delivering a signal, the temporary malfunctioning is resolved.

time-step	1	2	3	4	5	6
A1FailsSig	*no*	***yes***	***no***	*no*	*no*	*no*
arith1	sig	sig	**noSig**	noSig	noSig	noSig
arith2	idle	idle	idle	idle	**sig**	sig
Monitor	normal	normal	normal	**degraded**	**off**	off
H	false	false	**true**	true	**false**	false

Table 5.1.: Temporary Hazard Occurrence

Such an effect is not directly detectable with DCCA as the hazard is a Boolean predicate. One solution to this problem is to introduce an observer module which signals the hazard after a certain time passes. In this way some time-dependent hazards can be expressed as Boolean predicates. This approach will be used for the quantitative analysis in Section 5.3.

But for qualitative safety analysis, on the other hand, a much more general and elegant approach to cope with this problem is also possible. It can cope with more general self-healing systems which can recover from a temporary hazardous state. In order to analyze such systems correctly, a new variant of DCCA was developed in this dissertation that can cope directly with such situations.

5.2.3. Adaptive DCCA

To address the increasing complexity of modern systems, new paradigms of the development of system are developed which allow the system to decide certain aspects autonomously. Prominent examples of these paradigms are IBM's Autonomic Computing Initiative proposed by Kephart and Chess [KC03] and Organic Computing proposed by Müller-Schloer et al. [MS04, MSvW04].

A very important subclass of these systems which could be very beneficial for safety-critical system are *self-healing* systems. These can recover autonomously from a temporary system hazard. This will most often be realized by the introduction of a certain amount of redundancy in the system. The drawback of the autonomy is that it is much harder to give guarantees of the behavior of self-healing systems. Nevertheless, this is

essential for using such systems in safety-critical applications. This dissertation presents a new method for safety analysis and verification of such systems.

As shown in the example case study, DCCA cannot directly be applied to a self-healing system, as the hazard occurrence may only be temporary. If this is not reflected, a safety analysis would be too pessimistic, as a recovery after the occurrence of the hazard is possible. As long as enough redundancy is available, a self-healing system can get back to its functional mode again or at least enter a degraded "safe mode". Only a permanent occurrence of the system hazard is considered critical. This fact is reflected in *adaptive DCCA* (aDCCA) which was first introduced in [GOR06b] from where the definition (32), the following description and Lemma 6 are adapted.

Definition 32. aDCCA / Minimal Adaptive-Critical Set for Adaptive DCCA

For a system M^+ and a set of failure modes Δ a subset of component failures $\Gamma \subseteq \Delta$ is called adaptive-critical for a system hazard for a hazard H if

$$\kappa(M^+), s_0 \models \mathbf{E}\,[\overline{\Gamma}\ \mathbf{U}\ (\mathbf{E}\,\mathbf{G}\,(\ H \wedge \overline{\Gamma}))] \ \textit{where}\ \overline{\Gamma} := \bigwedge_{\delta \in (\Delta \setminus \Gamma)} \neg\, \delta$$

Γ a minimal adaptive-critical set if Γ is adaptive-critical and no proper subset of Γ is adaptive-critical.

The aDCCA proof obligation in Def. 32 states that a set of failure modes Γ is adaptive-critical if there exists a trace such that only these failure modes can lead to permanent system failure. The permanent occurrence of the hazard is expressed as $\mathbf{E}\,\mathbf{G}\,(H \wedge \overline{\Gamma})$ which means that a continuation exists on which H holds in all states. The last part in the formula ($\wedge\overline{\Gamma}$) is necessary to assure the failures modes of Γ are the reason for the *permanent* system hazard and no other failure mode occurs on the path continuation.

Just like with DCCA, the *adaptive-criticality* property of a set of failures is also monotonic for the notion of *adaptive-critical sets*. This means: $\forall\ \Gamma_1, \Gamma_2 \subseteq \Delta : \Gamma_1 \subseteq \Gamma_2 \Rightarrow$ (Γ_1 is adaptive-critical set $\Rightarrow$ Γ_2 is adaptive-critical set). A complete aDCCA and the resulting set of minimal adaptive-critical sets $aMCSS(M^+, \Delta, H)$ are defined analogously to Def. 29.

Lemma 6. aDCCA Completeness

For a complete aDCCA of an extended system model M^+, a set of failure modes Δ and a hazard H the following formula holds:

$$\kappa(M^+), s_0 \models \mathbf{A}\,((\bigwedge_{\Gamma \in aMCSS(M^+,\Delta,H)} \neg \bigwedge_{\gamma_j \in \Gamma} \mathbf{F}\,\gamma_j) \rightarrow \neg\mathbf{F}\,\mathbf{G}\,H)$$

proof see p. 172

For the formulation of the completeness lemma, CTL* logic is used. Fortunately the CTL subset – for which model-checking is much more efficient – is sufficient to compute the minimal adaptive-critical sets. The formula of Lemma (6) states, that on all (**A**) those traces on which from each minimal adaptive-critical set the occurrence of at least one failure mode can be prevented ($\neg \bigwedge_{\gamma_j \in \Gamma} \mathbf{F}\, \gamma_j$), the hazard can always be repaired by the system again ($\neg \mathbf{F}\, \mathbf{G}\, H$). In other words: the hazard can only become permanent if all failure modes of at least one minimal adaptive-critical set occur. This means that aDCCA is complete and no cause for the hazard is omitted. The correctness of aDCCA can also be guaranteed. For each minimal adaptive-critical set, the proof obligation in Def. 6 can compute a *witness run* of the system on which the combination of failure modes causes the hazard[4].

The relation of aDCCA to DCCA is shown in Lemma (7). It states that every set of failures that is considered adaptive-critical with aDCCA is also considered critical with DCCA. The other direction of the implication does not hold. As a counterexample consider a system where the hazard H can occur but is always repaired after $k > 0$ time-steps and is then absent for at least one time-step. Such a system never fails permanently and therefore is not considered adaptive-critical. Note that this also means that there is no necessary subset inclusion relation between the minimal critical sets and the minimal adaptive-critical sets. In the example above the occurrence of the temporary hazard would be considered the cause of a minimal critical set, but there would be no minimal adaptive-critical superset of it.

Lemma 7. DCCA Implication for aDCCA

Let M^+ be an extended system model, Δ be a finite set of failure modes, $\Gamma \subseteq \Delta$, and M be the embedded Kripke structure of $\kappa(M^+)$ then

$$\kappa(M^+), s_0 \models \mathbf{E}\,[\overline{\Gamma}\mathbf{U}\, \mathbf{E}\,\mathbf{G}\,(H \wedge \overline{\Gamma})] \Rightarrow \kappa(M^+), s_0 \models \mathbf{E}\,[\overline{\Gamma}\mathbf{U}\, H]$$

proof see p. 173

Just as in DCCA, the adaptive variant computes the critical combinations of failure modes without considering any temporal ordering. Analogous to the deductive ordering analysis of traditional systems, it is possible to specify proof obligations to compute the ordering relations for a self-healing system.

Adaptive Deductive Ordering Analysis

The ordering relation of minimal adaptive-critical sets of a self-healing system can be deduced from a SAML model analogously to non-self-healing systems. For non-self-healing systems, all traces on which only the failure modes of a minimal adaptive-critical set appear before the hazard are of interest. For self-healing systems, the traces

[4]Most often this will be computed by specifying the negation of the proof obligation and using a model-checker to compute a counterexample. A counterexample to the negation is then equivalent to a witness of the original proof obligation.

to consider are those on which the non-critical failure modes do *not appear at all and* the hazard finally becomes *permanent*.

This situation is formalized as $(\overline{\Gamma}\mathbf{U}(\mathbf{G}\,(H \wedge \overline{\Gamma}))$. This is the precondition for each of the proof obligations in the Lemmas (8) and (9). For a minimal adaptive-critical set Γ, this precondition describes all those traces on which the hazard H eventually becomes permanent and all failure modes that are not in Γ do not occur at all.

Lemma 8. Adaptive Deductive Before Ordering / Strict Before Ordering
For an extended system model M^+, a set of failure modes Δ, a predicate logic hazard H, a minimal adaptive-critical set Γ and two failure modes $\gamma_1, \gamma_2 \in \Gamma$,

$$\begin{aligned} &\gamma_1 \preceq_\Gamma \gamma_2 \Leftrightarrow \\ &\kappa(M^+), \pi \models \quad (\overline{\Gamma}\mathbf{U}(\mathbf{G}\,(H \wedge \overline{\Gamma})) \rightarrow [(\neg\gamma_1 \wedge \neg\gamma_2) \\ &\qquad\qquad \mathbf{U}(\gamma_1 \wedge ((\neg\gamma_2) \\ &\qquad\qquad\quad \mathbf{U}(\gamma_2 \wedge (\mathbf{F}\,\mathbf{G}\,H)))] \end{aligned} \tag{5.5}$$

$$\begin{aligned} &\gamma_1 \prec_\Gamma \gamma_2 \Leftrightarrow \\ &\kappa(M^+), \pi \models \quad (\overline{\Gamma}\mathbf{U}(\mathbf{G}\,(H \wedge \overline{\Gamma}))) \rightarrow [(\neg\gamma_1 \wedge \neg\gamma_2) \\ &\qquad\qquad \mathbf{U}((\gamma_1 \wedge \neg\gamma_2) \wedge \\ &\qquad\qquad\quad \mathbf{X}((\neg\gamma_2) \\ &\qquad\qquad\qquad \mathbf{U}(\gamma_2 \wedge (\mathbf{F}\,\mathbf{G}\,H))))] \end{aligned} \tag{5.6}$$

proof see p. 173

Analogously to the deductive ordering analysis for non-self-healing systems, the proof obligations assure that the first occurrence of γ_1 is not after the first occurrence of γ_2 for the before ordering, respectively the first occurrence of γ_2 is at least one time step after the first occurrence of γ_1. The difference here is the omission of $\neg H$, as the eventual permanent occurrence of the hazard $(\mathbf{F}\,\mathbf{G}\,H)$ is of interest.

Lemma 9. Adaptive Deductive Simultaneous Order
For an extended system model M^+, a set of failure modes Δ, a predicate logic hazard H, a minimal adaptive-critical set Γ and two failure modes $\gamma_1, \gamma_2 \in \Gamma$

$$\begin{aligned} &\gamma_1 \sim_\Gamma \gamma_2 \Leftrightarrow \\ &\kappa(M^+), \pi \models \quad (\overline{\Gamma}\mathbf{U}(\mathbf{G}\,(H \wedge \overline{\Gamma})) \rightarrow [(\neg\gamma_1 \wedge \neg\gamma_2) \\ &\qquad\qquad \mathbf{U}((\gamma_1 \wedge \gamma_2) \wedge (\mathbf{F}\,\mathbf{G}\,H))] \end{aligned} \tag{5.7}$$

proof see p. 174

The combination of aDCCA and adaptive deductive ordering analysis can easily be applied to the example case study, by checking the respective proof obligations.

Application of adaptive DCCA and Deductive Ordering Relation Analysis

In contrast to the first analysis with DCCA, where the recovery from the temporary hazard after the failure of A1 was not anticipated, no more single point of failure is identified with this more accurate analysis. For the analysis of combinations of two failure modes, 15 combinations had to be checked with aDCCA. This lead to the following 8 minimal adaptive-critical sets:

- $\Gamma_1 := \{A1FailsSig, A2FailsSig\}$
- $\Gamma_2 := \{A1FailsSig, MonitorFails\}$
- $\Gamma_3 := \{A1FailsSig, A2FailsActivate\}$
- $\Gamma_4 := \{A1FailsSig, S2FailsSig\}$
- $\Gamma_5 := \{A2FailsSig, MonitorFails\}$
- $\Gamma_6 := \{MonitorFails, A2FailsActivate\}$
- $\Gamma_7 := \{MonitorFails, S2FailsSig\}$
- $\Gamma_8 := \{S1FailsSig, S2FailsSig\}$

The minimal adaptive-critical sets $\Gamma_5, \Gamma_6, \Gamma_7, \Gamma_8$ are equivalent to the results of DCCA. For Γ_6, the temporal relation $MonitorFails \prec_{\Gamma_6} A2FailsActivate$ holds. The differences of DCCA and aDCCA are the first four minimal adaptive-critical sets which all include the failure of A1, which was considered a single point of failure in the prior analysis.

The first set Γ_1 describes the situation that both arithmetic units fail. If this happens, no more output signal can be produced. The critical combination in set Γ_4 results from the situation where A1 fails, A2 gets activated, but S2 fails to deliver a signal. Neither of these two minimal adaptive-critical sets exhibits a temporal ordering.

The sets Γ_2 and Γ_3 describe two situations where A1 fails, but the system does not correctly enter its degraded mode. Either because the monitor fails to detect that A1 failed (Γ_2) or because the activation of A2 fails (Γ_3). The temporal ordering analysis of these two sets reveals that both $MonitorFails \preceq_{\Gamma_2} A1FailsSig$ and $A1FailsSig \prec_{\Gamma_3} A2FailsActivate$ hold. This means that the minimal adaptive-critical set Γ_2 is only critical if the monitor fails before or at the same time as A1 fails. For Γ_3 to be critical, the failure mode *A1FailsSig* must occur strictly before *A2FailsActivate*.

In the case of Γ_3 and Γ_6 the failure mode which must appear after the first failure mode is the per-demand failure mode *A2FailsActivate*. This is not surprising, as the *demand* is triggered by the watchdog monitor which tries to activate A2. This *demand* can only be triggered if either the monitor detects a malfunction of A1 or if the monitor itself fails with a misdetection. Therefore per-demand failure modes will often be in such

a temporal relation. Nevertheless it is not mandatory, e.g. if not a single failure mode occurrence is critical, but a sequence.

Summarizing, a precise qualitative analysis of extended SAML system models is possible with the new qualitative analysis methods aDCCA for minimal adaptive-critical sets and the deductive ordering analyses. These extend the range of systems which are analyzable and allow for a further refinement of the analysis. This results in a more precise qualitative analysis of extended SAML system models than possible before. For each critical combination of failure modes, witness paths can be computed and analyzed that show how the combination of the failure modes can cause the hazard. This information can be used to augment the systems by the integration of risk-reducing measures. Once this has been completed, quantitative analysis can be used to compute the overall probability of the occurrence of a hazard.

5.3. Quantitative Model-Based Safety Analysis

After the possibilistic qualitative safety analysis and the potential introduction of risk-reducing measures, the next step is to apply a *probabilistic*, quantitative safety analysis. In many traditional safety analysis approaches, the occurrence probability of a system hazard is computed a-posteriori on the results of a qualitative analysis. Starting from the occurrence probabilities of the single failure modes, the overall hazard probability is estimated. This is most often based on the "fault tree formula" shown in Eq. (5.8) which is used in FTA to estimate the probability for the minimal cut sets to cause the top-level event. But is also used for the minimal critical sets of DCCA or in the FSAP/NuSMV-SA tool by Bozzano and Villafiorita [BV03b].

$$P(H) \leq \sum_{\Gamma \in MCSS(M^+,\Delta,H)} \prod_{\gamma \in \Gamma} P(\gamma) \tag{5.8}$$

This estimation can be very coarse, it cannot even be guaranteed that the result of the sum on the right hand part is below 1! The reason for this is that only the *results* of a qualitative analysis of the formal system model are used. Therefore the estimation must be very conservative. It is also assumed that all failure modes are stochastically independent, which is very often not the case. For example, if there exists a temporal ordering relation for some minimal critical sets. In addition, it is completely unclear how per-time failure rates and per-demand failure probabilities can be combined in such an approach.

5.3.1. Probabilistic DCCA

To overcome this problem, a new method was developed in this dissertation which exploits the probabilistic information in an extended system model expressed in SAML. It computes the overall occurrence probability of a hazard directly on the formal system

model. Analogous to the usage of CTL for DCCA and aDCCA, the probabilistic logic PCTL (see Section 3.4.2) is used for probabilistic DCCA (pDCCA) which was introduced in [GO10b].

Definition 33. Probabilistic DCCA (pDCCA)
For an extended system model M^+ and a hazard H, if

$$M^+, s_0 \models P_{max}[true\mathbf{U}\, H]_{>0} \tag{5.9}$$

holds, then the computation of the probability

$$P(H) := P_{max=?}[true\mathbf{U}\, H] \tag{5.10}$$

is called the probabilistic DCCA.

By using pDCCA, the problems outlined above are eliminated. The coarse estimation is replaced with an accurate computation of the occurrence probability. Assumptions about the stochastic independence are not necessary, as existing dependencies are automatically considered. This of course also includes the effect of possible temporal ordering relations. All these influences on the overall hazard probability are implicitly and automatically considered in the pDCCA formula and can be checked directly on a SAML extended system model [GO10a].

It is important to note that this analysis computes the overall hazard probability of all minimal critical sets together. The probability that a single minimal critical set causes the hazard cannot easily be calculated. The reason for this is that the PCTL analogon of the DCCA proof obligation, $P_{max}[\overline{\Gamma}\mathbf{U}\, H]_{>0}$, does *not* formalize "maximal probability of traces on which only failure modes in Γ may cause H". Traces can exist where the failure modes of Γ cause the hazard, but other failure modes also appear, but without having an effect. Any such trace would not be considered in this formula which therefore does not formalize this property. A correct formalization of this property would require the ability to reference the caused effect of a failure mode and should be formalized as: "maximal probability of traces on which only the failure effects of failure modes in Γ may cause H". Unfortunately it is not yet clear whether it is possible to express this without elaborate changes to the SAML system model.

pDCCA in the form as shown in Def. 33 is well suited for non-reactive systems which do not continue to run infinitely. A reactive system potentially runs infinitely long (at least in an abstract view). This means that the system hazard will occur eventually. The reason for this is that semantics of PCTL is defined over infinite traces of a system. If a reactive system were analyzed with pDCCA as in Eq. (5.10), this would result in hazard probability of 1, as in general every combination of failure modes will eventually occur on an infinite trace and therefore the hazard is inevitable. This does not offer more information than pure qualitative analysis, and is therefore not adequate for the quantitative analysis of reactive systems.

5.3.2. Probabilistic DCCA for Reactive Systems

To overcome this problem and to make pDCCA feasible for the analysis of reactive systems, the computation of the occurrence probability of the hazard is restricted to finite traces. In particular a *mission time* of the safety-critical system is specified which represents the considered time interval for the quantitative analysis.

This mission time is defined as a multiple of the temporal resolution of the system. Then the occurrence probability of the hazard can be computed using the *bounded until* PCTL operator (see Eq. (3.5)). If $P_{max}[\phi \mathbf{U}^{\leq k} \psi]_{>0}$ holds, then there exists, with non-zero probability, a bound $j \leq k$, so that ψ becomes *true* after no more that j steps and ϕ is *true* for all time steps $i < j$. Using this, the overall probability of the occurrence of the hazard of a reactive system can be computed with the bounded variant of pDCCA for a given mission time $t = k \cdot \Delta t$.

Definition 34. Bounded pDCCA

For an extended system model M^+ and a hazard H, if

$$M^+, s_0 \models P_{max}[true \mathbf{U}^{\leq k} H]_{>0} \tag{5.11}$$

holds, the computation of the probability

$$P(H) := P_{max=?}[true \mathbf{U}^{\leq k} H] \tag{5.12}$$

is called the bounded pDCCA with mission time $t = k \cdot \Delta t$.

If the analyzed system is not a reactive system, the mission time is basically system-inherent and does not influence the outcome of the analysis[5]. In this case the constant k of the equation (5.12) can be set to ∞ and the bounded until operator effectively becomes unbounded and it corresponds to a general pDCCA. As the probability of a bounded until operator is computed iteratively and the unbounded operator can be computed directly in one single step, it is advisable to use pDCCA if possible, to minimize the time needed for the analysis.

Specifying a "correct" mission time for a bounded pDCCA is not trivial. It is obvious that a single step or too few steps in general may result in a zero hazard probability, which will often be unrealistic. The specification of too long mission times can render the analysis infeasible because of the required running time. The goal is to specify a realistic mission time that covers a representative time interval for the system. At the end of the next section first approaches to address these challenges are presented.

5.3.3. Adaptive pDCCA

Unfortunately, using pDCCA directly for adaptive systems is not as straightforward as it is for qualitative analysis. The PCTL analogon of the aDCCA formula would be

[5] After a given time, "nothing interesting" will happen in such systems, in particular no hazard occurrence.

$P_{max=?}[true\mathbf{U}\, P_{max}[\mathbf{G}\, H]_{>0}]$. This cannot be used to compute the probability that the hazard eventually becomes permanent. The reason for this is that the CTL formula $\mathbf{E}\,\mathbf{G}\,\phi$ is not equal to $P_{max}[\mathbf{G}\,\phi]_{>0}$. An example for this has been shown in Fig. 3.7. The same holds for the bounded variant of the above formula $P_{max=?}[true\mathbf{U}^{\leq k} P_{max}[\mathbf{G}\, H]_{>0}]$.

A solution which can be used to handle this problem *if* the maximal time of the temporary hazard is known, is to introduce an additional *observer* module. The states of such a module then signal whether the hazard has occurred permanently or just temporary and the permanent hazard occurrence can therefore be reduced to a Boolean predicate. Most often this will be realized by counting the number of time-steps the hazard is present. If it is possible to decide whether the occurrence is permanent (e.g. the hazard was present longer than the maximal time to be only temporary), then a Boolean predicate ("Observer is in state X") can be used to signal the permanent hazard occurrence. pDCCA (or bounded pDCCA for reactive systems) can then be used for quantitative analysis of such a system.

This additional observer module must of course not be allowed to change the probability of the resulting traces. This means that the observer must only contain deterministic transitions and no non-deterministic or probabilistic choices[6]. Then the parallel composition with the other SAML modules does not change the probability of the resulting traces.

Application of pDCCA

An observer module was added for the quantitative analysis of the example case study. The observer module was modeled as shown in Fig. 5.2. Its initial state *init* is left if the system is started. The successor states of *init* is state *ok* if either A1 or A2 produces a signal, or state "error1" if no valid signal is produced from either arithmetic unit. This situation is similar for the waiting states *error1* and *error2*, there is a return to state *ok* if a signal is produced or a state change to the next state if no such signal is available. If this happens in state *error2*, the successor is the state *hazard*. If the observer enters this state, the system did not produce a signal for three time steps in a row which is considered as a permanent hazard.

The case study was analyzed for a running time of $10h$. It is a reactive system, therefore bounded pDCCA was used. As described in the failure mode modeling in the previous chapter, the temporal resolution was $\Delta t = 10ms$. This translates to an analysis time of $k = 3.600.000$ steps for a mission time of $10h$. The overall maximal probability that the system fails to deliver an output signal within $10h$ of operational time (using the failure rates and probabilities as described in Section 4.5.2 is therefore:

$$P_{max=?}[true\mathbf{U}^{\leq 3.600.000}(observerSig = hazard)] = 9.1490 \cdot 10^{-7} \tag{5.13}$$

[6]This means all probabilistic choice must have probability 1

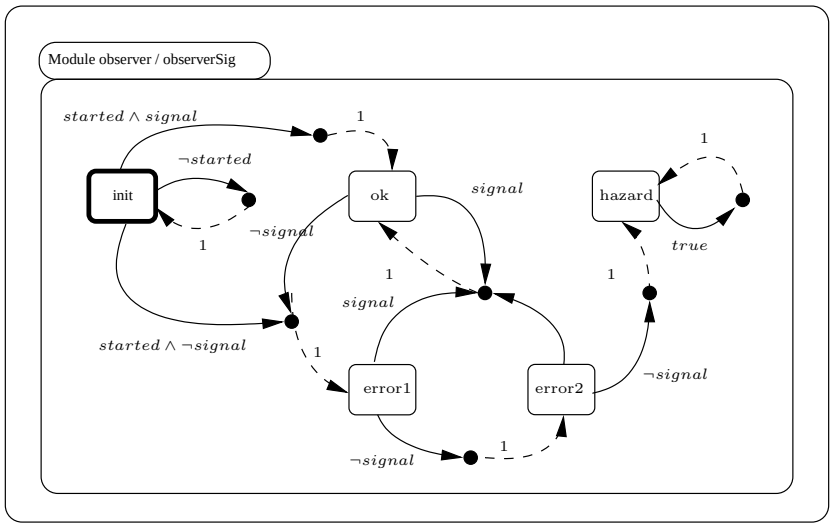


Figure 5.2.: Observer Automaton

Extrapolation of Results and Effects of Different Temporal Resolutions

The probability that the hazard occurs in the case study was computed for the mission time of $10h$. The time needed for a bounded pDCCA is directly dependent on the number of time-steps, as the algorithms used for the computation carry out k matrix-vector multiplications for k time-steps. This can lead to very long or even infeasible running times for large case studies. Reducing the number of necessary multiplications for the algorithm itself, e.g. by using a square and multiply approach, is not possible, as this generally destroys the sparsity of the underlying models and significantly increases the memory requirements.

Because of this, methods to reduce the analysis time without reducing the accuracy (or only slightly reducing the accuracy) would be very beneficial. The most important factors that determine the number of steps k are the temporal resolution and the mission time. This raises the question how different temporal resolutions affect the results and whether the pDCCA results for a mission time t can be extrapolated to a longer time $t' > t$ without explicitly analyzing this mission time.

Whether and how this is possible in general is still an open question. But the effect of different temporal resolutions on the overall analysis result and the possibility to extrapolate from a shorter to a longer analysis interval can be analyzed experimentally.

Accuracy of the Failure Approximation The accuracy of the discrete approximation of the per-time failure rates with the geometric distribution has been discussed in Section 4.5.2. To analyze the *effect* of this accuracy on the overall hazard probability, two different temporal resolutions, $\Delta t_1 = 10ms$ and $\Delta t_2 = 1s$ were considered. For the failure rate $10^{-2}\frac{1}{h}$ of the failure mode *S1FailsSig*, the maximal and cumulative relative

error for $t = 10h$ is shown in table 5.2.

	$\Delta t_1 = 10ms$	$\Delta t_2 = 1s$
maximal relative error	$1.2710 \cdot 10^{-8}$	$4.4577 \cdot 10^{-4}$
cumulative relative error	$1.3889 \cdot 10^{-6}$	0.048764

Table 5.2.: Relative Approximation Error for Failure Rates

This table shows that the order of magnitude of the approximation error is roughly the same as the ratio of the lower to the higher temporal resolution, and that the higher resolution approximates slightly better than explainable only by the ratios. The two different temporal resolutions were then used to analyze the system with pDCCA (after all other model parameters were adjusted accordingly). The result for Δt_1 is shown in Eq. (5.14), the result for Δt_2 is shown in Eq. (5.15).

$$P_{max=?}[true\mathbf{U}^{\leq 3.600.000}H] \approx 9.1490 \cdot 10^{-7} \quad (5.14)$$

$$P_{max=?}[true\mathbf{U}^{\leq 36.000}H] \approx 9.2707 \cdot 10^{-7} \quad (5.15)$$

From these results, one can conclude that the orders of magnitude of the approximation errors for the failure rates does not necessarily influence the overall result in a comparable way. For a coarser first estimate, a lower temporal resolution can be adequate. This could be used to compare different variants of a system, possibly identifying the most promising variants and analyze those later with more accuracy.

Extrapolation of pDCCA Results For this experiment, the computed pDCCA of the case study were used to determine the accuracy of an extrapolation of a mission time of $10h$ (with $\Delta t = 10ms$) from shorter mission times. The computed hazard probabilities up to $10h$ are shown in Fig. 5.3. The results were computed in $10s$ steps.

Three different methods were used to extrapolate a longer mission time from the computed results for $t = 10s, 20s, 30s, 40s$. A simple, rather obvious form of extrapolation is a directly proportional one. This means to use the quotient $\frac{t'}{t}$ (the quotient of the longer time interval to the analyzed interval) as a scaling factor to extrapolate the probabilities. This results in a linear approximation.

As second approach a cubic approximation was chosen which is based on the numerical derivations of the computed probabilities. The function graph of the computed probabilities for every continuous cumulative probability distribution is monotone and changes its curvature. Therefore a cubic extrapolation was used because it is the simplest approach that can also change the curvature.

The third approach was to use an exponential function of the form $1 - b \cdot e^{-ct}$ and to use optimization techniques to find the parameters b and c such that the approximation is best for the computed values wrt. the sum of squared deviances.

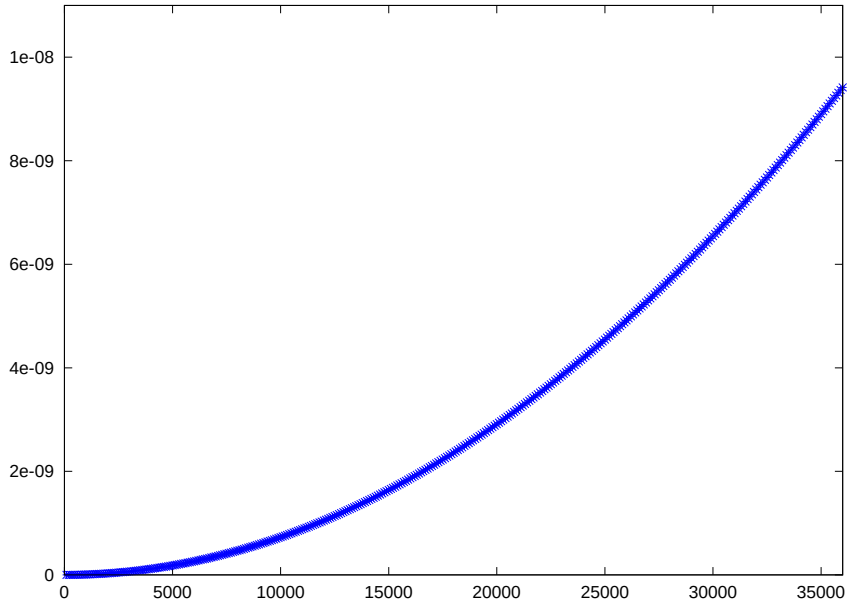


Figure 5.3.: Computed Hazard Probabilities

	value for $t = 10h$	cumulative deviance	max. rel. error
linear	$1.3120 \cdot 10^{-9}$	0.0011048	0.99857
cubic	$9.1421 \cdot 10^{-7}$	$4.9125 \cdot 10^{-7}$	$7.5364 \cdot 10^{-4}$
exponential	$4.2785 \cdot 10^{-10}$	0.0010835	53.982

Table 5.3.: Results of the Extrapolation Experiments

The results are shown in table 5.3. These first experiments show that simple linear extrapolation is likely not sufficient and will not yield good results. Surprisingly, the exponential extrapolation approach is not much better considering the cumulative deviance and even worse considering the extrapolation for $t = 10h$. In contrast, the cubic approximation led to good results in the case study and seems a promising candidate for further study.

From the experiments and the properties of the case study, some important factors that influence the quality of the extrapolation approach were identified:

- **Persistent changes in the system under consideration:** Persistent changes in a system lead to stochastic dependencies which are clearly not representable in a simple linear model.
- **Representative extrapolation interval:** The computed results which form the basis for the extrapolation must reflect the representative details of the behavior of the system.

Whether there are persistent changes in a system can be verified by checking whether a representative state after the starting phase of a system can always be reached again. If this is possible, then there is always a way to "reset" the system and an extrapolation of the probabilities is reasonable. Some ongoing research in this area is done by Seguin et al. [TIS11] for the new EPOCH probabilistic model-checker. Here strongly connected components in the MDP are exploited to increase the efficiency of probabilistic model-checking. Such components could correspond to "non-resettable" system states.

The choice of a representative interval can be reduced to find a good coverage of states. This means that the mission time must be chosen in such a way that within its corresponding time steps, as many states as possible are reachable. One approach to check for a representative interval could be to use bounded model checking described by Clarke et al. [CBRZ01] to check whether all states are reachable within k steps. The influence of probabilistic choices on the overall result is reduced with the length of the considered path. As pDCCA is a reachability analysis, such an extrapolation could be suited to identify promising system variants. Such approaches will be studied further in the ProMoSA project (see the outlook in Section 8.2).

5.4. Related Work

The failure injection methods as developed by Abdulla et al. for SCADE [ADS+04] and by Bozzano and Villafiorita for FSAP / NuSMV-SA [BV03b] also employ a qualitative analysis for critical failure modes. In general, first a safety property is specified and then a combination of failure modes is integrated into the system. If the verification of the safety property fails, the failure mode combination is considered critical. Using this method, a fault tree consisting of three layers can be deduced from the formal system. The lowest layer is the conjunction of the critical combinations of failure modes, the second layer is the disjunction of each critical failure mode set and the highest layer is the system hazard. This will always lead to a complete FTA, but in contrast to DCCA and aDCCA, there is no proof of correctness of the analysis. The chosen safety properties do not require the failure modes to be the actual cause of the hazard, this is only assumed implicitly.

Dynamic Fault Trees Analysis (DFT) [vM02] extends qualitative fault tree analysis with several addition fault tree gates that also capture some dynamics of the system. Two of these are *priority-AND* (PAND) and *simultaneous-AND* (SAND) which correspond to the temporal ordering of two basic events. A very similar notion of ordering is employed in an algebraic technique based on Papadopoulos' and McDermid's Hip-HOPS notation [PM91], developed by Walker et al. [WP07, WBP07]. In this approach besides PAND, a priority-OR (POR) relation is defined. The correlation of the different temporal ordering relations, temporal fault tree gates and the Hip-HOPS notation is shown in Table 5.4.

The computation of temporal ordering of failure modes in minimal critical sets can

Failure Order Relation	Temporal Fault Tree Gate	Hip-HOPS
$\phi_1 \preceq_\Gamma \phi_2$	$(\phi_1 SAND \phi_2) OR (\phi_1 PAND \phi_2)$	$\phi_1 < \phi_2 + \phi_1 \& \phi_2$
$\phi_1 \prec_\Gamma \phi_2$	$\phi_1 PAND \phi_2$	$\phi_1 < \phi_2$
$\phi_1 \sim_\Gamma \phi_2$	$\phi_1 SAND \phi_2$	$\phi_1 \& \phi_2$

Table 5.4.: Temporal Ordering Notations

automatically construct dynamic fault trees with the temporal ordering gates PAND and SAND. For minimal (adaptive-)critical sets, failure modes in POR relation are considered as single point of failures. Therefore no automatic deduction of a priority OR relation is specified. A technique to deduce dynamic fault tree gates via a temporal failure mode ordering from a formal system model is presented by Bozzano and Villafiorita in [BV03a] and integrated into the FSAP / NuSMV-SA tool by Bozzano et al. [BV03b]. In contrast to deductive ordering analysis, an additional state variable is introduced into the system which signals the failure mode ordering. Therefore formally, each time a different system is analyzed. This is in contrast to deductive temporal ordering analysis, where the same extended system model is analyzed, but with different proof obligations. Nevertheless, for most systems the analysis results will be similar for both methods.

The quantitative assessment of hazard probabilities of failure modes with temporal ordering dependencies has been approached in different ways. Many approaches derive state space based systems from a dynamic fault tree which represent the temporal ordering of the failure modes. The underlying models for analysis can be stochastic Petri Nets which are used for example by Raiteri [CR05] or Bernardi and Donatelli [BD04]. Another possibility to analyze dynamic fault trees is to use Markov Chain variants as described by Boudali et al. [BCS07b, BCS07a]. These models are then analyzed as continuous time Markov Chains (CTMC). Very often these approaches also support additional dynamic fault tree gates like functional dependency (FDEP), sequence (SEQ) or warm spare (WSP). As with all CTMC analysis, the assumption is made that all probabilities are exponentially distributed.

An interesting approach was developed by Merle et al. [MRLB10, MRLV10] and is described in detail by Merle in his dissertation thesis [Mer10]. It uses algebraic techniques to deduce probabilities from a dynamic fault tree. This allows for using arbitrary probability distributions and is therefore more flexible than approaches based on Markov models. In SAML as in most other safety analysis approaches based on probabilistic model-checking, arbitrary probability distributions can only be approximated with several exponential (respectively geometric) distributions. This will very likely increase the state space of the underlying MDP and is therefore less practical for non-exponentially distributed failure mode probabilities.

Quantitative analysis of system models similar to the ones presented here is often conducted on the basis of CTMC models. Bozzano et al. developed the SLIM language [BCK+09b, BCK+10b], which formalizes a subset of the AADL error annex. SLIM

models can include a failure model which is analyzed with the MRMC model checker after the non-probabilistic parts of the system are abstracted away with bisimulation techniques. This approach is described by Katoen et al. in [KKZJ07] and uses the SigRef tool developed by Wimmer et al. [WHH+06]. This "Performability Analysis" restricts the analyzable models at the moment to non-hybrid models which do not use continuous variables.

Grunske et al. introduced probabilistic failure modes and effects analysis (pFMEA) in [GCW07]. This technique allows to deduce FMEA tables with probabilities and overall system hazard probabilities from a formal system description which also expresses functional properties. This approach is very similar to failure injection and is not only a state space representation of an existing dynamic fault tree. It is extended by Aljazaar et al. [AFG+09, AKLFL10] to also extract counterexamples from a quantitative system description, namely those with the highest probability of occurrence. Compared to qualitative SAML analysis, this is more computationally expensive. But if feasible, it delivers more information how the dependability of a system can be increased. An integration of the generation of probabilistic counterexamples would also be very beneficial for safety analysis based on SAML.

Within the AVACS project [AVA03], an approach to find the failure mode combination that accounts for the biggest part of the overall hazard probability is presented by Böde et al. [BPRW08a]. The approach is called "Importance Analysis of Minimal Cut Sets". Here, failure modes are integrated into a functional system model analogous to failure injection. For the analysis, a *quotient* system is constructed using bisimulation techniques. In the quotient system all traces are eliminated on which failure modes appear which are not currently analyzed. This is accomplished using the SigRef tool developed by Wimmer et al. [WHH+06]. It is then used to rank the critical combinations of failure modes relative to another. The problem here is that the analyzed system is different for each analysis, and therefore strict propositions about the probabilities are not possible. A relative ranking of minimal critical sets would also be possible in SAML, by using the PCTL analogon of the DCCA formula [GO10b].

Summary

This chapter introduced new qualitative and quantitative model-based analysis methods. New qualitative methods are the deductive temporal ordering analysis, adaptive DCCA and adaptive deductive temporal ordering analysis. The ordering analyses allow for a more precise analysis of the temporal relations of critical failure modes. In contrast to other approaches these analysis rely only on different temporal logic proof obligations and do not require changes to the system model itself. Adaptive DCCA allows for provably complete and correct qualitative safety analysis of self-healing systems. Safety analysis of these systems was not possible before the development of aDCCA.

This chapter also presented the quantitative analysis method pDCCA. It allows a

much more precise computation of the occurrence probability than previous a-posteriori methods which rely on the quantitative analysis of the results of a prior qualitative analysis. In contrast to other model-based safety analysis methods, the combination of per-time and per-demand failure modes is possible. First experiments on the effects of different temporal resolutions and the possibility to extrapolate pDCCA results to longer mission times were very promising. pDCCA and approaches based on these extrapolations will further be used to quickly identify promising system variants in the DFG-funded ProMoSA project [OG10] were the probabilistic analysis results will be used as objective function for system optimization.

6. Transformation and Analysis of SAML Models

A man's got to know his limitations.

(Harry Callahan)

The previous chapters introduced SAML, its semantics and several model-based safety analysis methods. SAML is designed to be a tool-independent intermediate language. For concrete analysis, an extended system model of a safety-critical system must be transformed into the input specification language of an appropriate analysis tool. This chapter presents exemplary transformations to translate SAML models for state of the art probabilistic and symbolic model-checkers.

Section 6.1 motivates the usage of model-checking tools for the analysis and explains how SAML benefits from the design as tool-independent modeling and intermediate language. Section 6.2 very briefly describes some details of the implementation of the transformations. The transformations necessary for the analysis of quantitative aspects of a SAML model with a probabilistic analysis tool are described in 6.3. The required transformation for a qualitative analysis tool is described in Section 6.4. Related work is discussed in Section 6.5.

6.1. Motivation

Several efficient model-checking tools have been developed for the analysis of formal system specifications. For a long time, most research was focused on the analysis of qualitative properties of systems, specified in different qualitative logics. The current state of the art is the compact representation of Kripke structures as (ordered) binary decision diagrams (BDD) and their analysis based on *symbolic model-checking* developed by Burch et al. [BCM+90].

For the analysis of labeled Markov chains, probabilistic model-checking has become practical in recent years. It is driven primarily by the massive increase of computing resources, the development of efficient numerical model-checking algorithms and the development of space-efficient representations of Markov models. While representations based on sparse matrices are very time-efficient, new approaches based on multi-terminal binary decision diagrams (MTBDD) as described by de Alfaro et al. [dAKN+00] allow for a much more space-efficient representation of Markov models. This makes even very large state spaces accessible for analysis, analogously to the BDD representation of symbolic model-checkers. In both cases giving a concise number of states for an analyzable system is not possible in general, as this depends on many different system-specific factors.

The development of both qualitative and quantitative model-checkers is still an active research area. A lot of current research for probabilistic model-checking focuses on exploiting parallelism. Bernadat et al. focus on multi-core architectures [BBC+08], Bosnacki et al. [BES09] use the GPUs of modern graphic hardware to accelerate the necessary numerical computations. Maisonneuve presents in her Master's thesis [Mai09] approaches to optimize the MTBDD representation with heuristic variable reordering which increases the range of analyzable models. Other methods are based on bisimulation techniques for more efficient analyses [KKZJ07, HWP+06, WHH+06]. To benefit from this ongoing research and the resulting improvements, in particular from the increase in analysis efficiency, SAML is designed as a *tool-independent* language that can be transformed into the input specification of different analysis frameworks.

This can be achieved by using model transformations to transform SAML models into model descriptions suitable for formal analysis with verification tools. An important aspect for this was to keep SAML as simple as possible, but at the same time being expressive enough for accurate modeling of extended system models for safety analysis. Using semantically founded transformations, it can be assured that the analyzed models are equivalent[1]. Therefore the analysis results of different verification tools are "compatible" with each other. This allows to choose the most appropriate analysis tool depending on the character of the property but also on the efficiency of the available verification tool. Obvious examples are the model-based safety analyses presented in the last chapter. DCCA / aDCCA and deductive ordering analysis would be analyzed with qualitative verification tools and pDCCA with quantitative ones.

[1]For example the equivalency between the paths of the MDP and its embedded Kripke structure.

6.2. Implementation of Transformations

A prototypical implementation of the transformations described in this chapter has been developed. Technically, the implementation is based on Parr's ANTLR parser generator [Par07] to read SAML specifications. The SAML specifications are then converted into a tree representation. This tree representation is transformed by a Common Lisp program[2] into the input specification of the different verification tools. Common Lisp was chosen because it offers functional programming capabilities which are well-suited for model transformations. It is also very appropriate for rapid-prototyping of algorithms. The foundations of these model transformations have been introduced in [GO10a].

Fig. 6.1 shows the currently implemented transformations for SAML models. For qualitative analysis (for example with DCCA / aDCCA and deductive ordering analysis), the embedded Kripke structures can be transformed into the input specification language of either NuSMV or Cadence SMV. NuSMV is the open source reimplementation of the original symbolic model verifier (SMV) [McM90] developed at the Fondazione Bruno Kessler, Cadence SMV is a proprietary implementation available from Cadence Berkeley Labs.

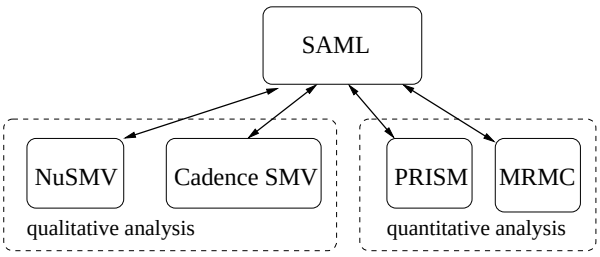


Figure 6.1.: Implemented SAML Transformations

For the analysis of quantitative properties like pDCCA, the most widely used probabilistic model-checker is PRISM which is developed by Kwiatkowska et al. [KNP02b, KNP02a] at the University of Oxford. It allows for the analysis of MDPs and uses an internal representation based on MTBDD. If the SAML model does not contain any non-deterministic choices, but is purely probabilistic, then it can be analyzed with the Markov Reward Model Checker (MRMC) of Katoen et al. [KKZ05] developed at the RWTH Aachen. Its advantage over PRISM is its increased speed. Its main disadvantage – besides the fact that only DTMCs are analyzable – is the representation as sparse matrices which is much less space efficient than symbolic MTBDD representation and is not feasible for larger models.

[2]Steel Bank Common Lisp

6.3. Transformation for Quantitative Analysis

The transformation of a SAML model into the input language of the probabilistic model checker PRISM for quantitative analysis is rather straightforward. The PRISM semantics is described in [NPK10]. SAML was designed in dependence on the PRISM language and is basically an extended subset thereof. Therefore PRISM models are structured in a very similar fashion in parallel modules and further in update rules for state variables with potentially parallel assignments. The PRISM language itself was not chosen, as it allows modeling which is not in accordance with the guidelines presented in Chapter 4.

The two main differences between SAML and the PRISM language are the synchronization of update rules and the specification of non-deterministic choices. In general, PRISM modules are completely asynchronous. This means that at each time-step only one module assigns new values to its state variables and the state variables of all other modules stay the same. As already mentioned in Chapter 4, this interleaved semantics is less appropriate for safety analysis, especially in a discrete time model. In order to synchronize update rules, synchronization labels can be specified in PRISM. Active update rules in different modules, but with the same synchronization label, are executed in parallel.

Non-deterministic choice is specified in the PRISM language only implicitly. If the activation conditions of two different update rules are semantically equivalent, then these update rules specify an intrinsic non-deterministic choice. Therefore it is also possible to have partially "overlapping" activation conditions which results in normalization of the associated probabilities. This means that the probabilities in the underlying MDP are not the same as those specified in the model. This often leads to undesired effects which are not anticipated. Because of this all non-determinism is specified explicitly in SAML and overlapping activation conditions are considered modeling errors.

6.3.1. Example Transformation

To illustrate the transformation of a SAML model into the specification language of PRISM, the small example model (introduced in Section 3.2) shown in Fig. 6.2 is used. Its first module a contains the following update rule:

```
v_a = 0 & v_b1 = 0 & v_b2 = 0 ->
        choice:(p_a : (v_a' = 0) + (1 - p_a) : (v_a' = 1));
```

It is a purely probabilistic update rule, because there is only a single probability distribution which assigns the value 0 to `v_a` with probability p_a and the value 1 with a probability $1 - p_a$. As mentioned above, the synchronization of the probabilistic update rules must be assured. The PRISM language allows for the usage of synchronization labels which are used analogously to synchronization in process algebras (e.g. see Bergstra and Klop [BK86]). Only update rules that have the same label are executed in parallel. Therefore to achieve full synchronicity of the resulting PRISM models, the same label `t`

```
constant double p_a := 0.1;
constant double p_b1 := 0.9;
constant double p_b2 := 0.09;
constant double p_b3 := 0.01;

formula case3 := v_a = 0 & !(v_b1 = 0 & v_b2 = 0 | v_b1 = 1 & v_b2 = 1);

module a
       v_a : [0..2] init 0;

       v_a = 0 & v_b1 = 0 & v_b2 = 0 ->
               choice:(p_a : (v_a' = 0) + (1 - p_a) : (v_a' = 1));
       v_a = 0 & v_b1 = 1 & v_b2 = 1 ->
               choice:(1 : (v_a' = 2));
       case3 -> choice:(1 : (v_a' = 1));
       v_a = 1 -> choice:(1 : (v_a' = 1));
       v_a = 2 -> choice:(1 : (v_a' = 2));
endmodule

module b
       v_b1 : [0..1] init 0;
       v_b2 : [0..1] init 0;

       true -> choice:(p_b1 : (v_b1' = 0) & (v_b2' = 0) +
                       p_b2 : (v_b1' = 1) & (v_b2' = 0) +
                       p_b3 : (v_b1' = 1) & (v_b2' = 1))
             + choice:(1 : (v_b1' = 1) & (v_b2' = 1));
endmodule
```

Figure 6.2.: Example SAML Model

(for "tick") is used for every update rule of the model. For the update rule above this would be written as:

```
     [t] v_a = 0 & v_b1 = 0 & v_b2 = 0 ->
               (p_a : (v_a' = 0) + (1 - p_a) : (v_a' = 1));
```

The other probabilistic update rules of the example model are transformed analogously. The parallel composition mechanism of PRISM then composes all the transformed modules equivalently to the synchronous parallel composition adapted for SAML modules as described in Section 3.3.1. This assures that the active update rules in the parallel modules are executed synchronously.

The transformation for update rules with non-deterministic choices is slightly more complex, as the representation of non-determinism in PRISM and SAML differs. The single update rule of the module *b* contains a non-deterministic choice between two different probability distributions:

```
1  convert-updates-SAML-to-PRISM (updateSetSAML)
2  updatesPRISM := ∅
3  for (update u : updateSetSAML)
4    (φ, variableSet, pDistList) := u
5    for (i = 0; i < |pDistList|; i++)
6      updatesPRISM := updatesPRISM ∪ {(φ, variableSet, pDistList[i])}
7  for (update u : updatePRISM)
8    addSynchronizationTick(u)
9  return (updatesPRISM)
```

Listing 6.1: Pseudocode to Convert a SAML Module to a PRISM Module

```
true -> choice:(p_b1 : (v_b1' = 0) & (v_b2' = 0) +
                p_b2 : (v_b1' = 1) & (v_b2' = 0) +
                p_b3 : (v_b1' = 1) & (v_b2' = 1))
      + choice:(1 : (v_b1' = 1) & (v_b2' = 1));
```

This non-determinism is denoted by the `choice` keyword. In a PRISM specification, non-determinism is given implicitly, by defining update rules with semantically equivalent activation conditions. The single SAML update rule is therefore transformed into two separate update rules, synchronized with the system tick and having an equivalent activation condition:

```
[t] true ->  p_b1 : (v_b1' = 0) & (v_b2' = 0) +
             p_b2 : (v_b1' = 1) & (v_b2' = 0) +
             p_b3 : (v_b1' = 1) & (v_b2' = 1);
[t] true ->     1 : (v_b1' = 1) & (v_b2' = 1);
```

This approach is used to transform all update rules of all modules of a SAML model to the input language of the PRISM model-checker.

6.3.2. Transformation to PRISM

The transformation of a SAML model to a PRISM model is conducted for each module. For each module all update rules are transformed in the way described above. The pseudocode in listing 6.1 shows the transformation of a set of SAML update rules to a set of PRISM update rules. Firstly, all non-deterministic updates rules are split into a set of update rules with the same activation condition. Secondly, all resulting update rules are labeled with the "tick" synchronization label.

When the transformation is applied to the example SAML model in Fig. 6.2 it results in a PRISM model with the same state space, the same non-deterministic choices and the same probability distributions as the original SAML model. The transformed PRISM model of the example is shown in Fig. 6.3. In addition to the transformation of the

```
mdp

const double p_a = 0.1;
const double p_b1 = 0.9;
const double p_b2 = 0.09;
const double p_b3 = 0.01;

formula case3 = v_a = 0 & !(v_b1 = 0 & v_b2 = 0 | v_b1 = 1 & v_b2 = 1);

module a
       v_a : [0..2] init 0;

       [t] v_a = 0 & v_b1 = 0 & v_b2 = 0 -> p_a : (v_a' = 0) + (1 - p_a) : (v_a' = 1);
       [t] v_a = 0 & v_b1 = 1 & v_b2 = 1 -> 1 : (v_a' = 2);
       [t] case3 -> 1 : (v_a' = 1);
       [t] v_a = 1 -> 1 : (v_a' = 1);
       [t] v_a = 2 -> 1 : (v_a' = 2);
endmodule

module b
       v_b1 : [0..1] init 0;
       v_b2 : [0..1] init 0;

       [t] true -> p_b1 : (v_b1' = 0) & (v_b2' = 0) +
                       p_b2 : (v_b1' = 1) & (v_b2' = 0) +
                       p_b3 : (v_b1' = 1) & (v_b2' = 1);
       [t] true -> 1 : (v_b1' = 1) & (v_b2' = 1) ;
endmodule
```

Figure 6.3.: Transformed PRISM Model

update rules, the constant definitions and the Boolean variables are also adjusted to the PRISM syntax.

An additional detail of PRISM models is the explicit specification whether it is a MDP or a DTMC. This is noted in the first line with the keywords `dtmc` or `mdp`. Although any DTMC is just a special case of MDP, the analysis of DTMCs can be much more time efficient. In addition, if a SAML model can be expressed as a DTMC, it can also be analyzed with the MRMC model-checker. Its current version cannot analyze MDPs but experiments have shown that it is much more time efficient than PRISM for the analysis of feasible models. Its input language is very simple and requires the explicit sparse transition matrix and the labeling of each state. The easiest way to create this is to use the export function of PRISM which constructs the state space of the DTMC and then exports the state space and the labeling function in the format appropriate for MRMC.

6.4. Transformation for Qualitative Analysis

For qualitative safety analysis, a SAML model is converted into a description of its embedded Kripke structure. One possible approach would be to convert the MDP of a SAML model directly into a Kripke structure on the semantic level. This means that its transition relation would be encoded directly and not using a notion of parallel modules with state variables. Another possible approach is the transformation on the syntactical level, i.e. transforming SAML modules into similar parallel module representation of the embedded Kripke structure.

The disadvantage of a transformation on the semantic level compared to a syntactic transformation is that it would require the construction of the whole state space. This is very inefficient as this has to be done for both the transformation and also for the later analysis. An important advantage of the transformation on the syntactical level is that it allows for a much better traceability of the model artifacts. This is very helpful in safety analysis, in particular for the interpretation of counterexamples. Therefore the choice was made to transform SAML models on the syntactical level for qualitative safety analysis.

Perhaps the most prominent example of a model-checker for Kripke structures is the symbolic model verifier (SMV) [McM90, BCM+90] which can verify both CTL and LTL property specifications. Several variants of SMV exist, the most current and accessible one is the open-source reimplementation NuSMV by Cimatti et al. [CCG+02]. It is available for many platforms in binary and source code form and is under ongoing development. Because of its availability and good performance it is used here as analysis tool for qualitative properties of SAML models. The described transformation for other SMV variants like Cadence SMV differs only in minor details and is not explained in detail here[3].

Analogously to SAML, NuSMV models are also structured as parallel modules which can contain one or more state variables. An obvious difference of NuSMV to SAML is of course the absence of probabilistic update rules as NuSMV models can only be non-deterministic. A minor difference is that a complete model is constructed from the parallel composition of a set of single modules which must explicitly be imported in a distinguished *main* module. This means that every NuSMV model has to include a main module and that a SAML module named *main* must be renamed for the transformation.

A more subtle difference of SAML and the NuSMV specification language makes the transformation challenging. SAML allows for parallel assignments of new values to variables in a single update rule. This can be very convenient in a probabilistic environment, where not only the fact is of interest *that* a parallel assignment is possible, but also *how probable* a specific one is. As to my current knowledge, such parallel assignments are not directly expressible in the specification language of NuSMV (or of

[3]The transformation to Cadence SMV is also implemented, the syntax of the input language is slightly different, the method of the transformation and the structure of the transformed models is the same.

other SMV variants). NuSMV only allows the assignment of a new value to a *single* state variable `v` via the `next(v)` construct.

The problem of the lack of parallel assignments is solved by introducing an additional state variable for each update rule with parallel assignments. Basically, an index number is assigned to each possible parallel assignment and the new state variable ($choose_i$[4] for update rule u_i) has a domain of these indices. The possible values for such a chooser variable are called the *admissible* values. At each time-step, the assigned new value for this chooser variable is selected non-deterministically and indicates which of the parallel assignments is to be carried out for the other state variables.

In order to transform a SAML module into a NuSMV module, first all probabilistic distributions are transformed into a set of non-deterministic choices. Next for each parallel assignment, a new chooser variable is introduced which indicates the possible new values. This chooser variable is then referenced in the activation conditions of the update rules of the original state variables. The basic principles of this approach are best described using an example.

6.4.1. Example Transformation

For illustration purposes, the transformation is explained for the SAML model shown in Fig. 6.2. The first step of the transformation of this model into a NuSMV model representation is to transform it into its embedded Kripke structure. This means that all probability distributions are replaced with non-deterministic choices.

In module a there is no non-deterministic or parallel assignment, therefore only the probabilistic choices have to be eliminated. The following update rule for the state variable `v_a` assigns the new value of 0 with a probability of p_a and a new value of 1 with probability $1 - p_a$:

```
v_a = 0 & v_b1 = 0 & v_b2 = 0 ->
        choice:(p_a : (v_a' = 0) + (1 - p_a) : (v_a' = 1));
```

The transformation of this probabilistic distribution into a purely non-deterministic choice is achieved by splitting it into two distinct probabilistic distributions, each with probability 1. This results in one non-deterministic choice for each element of the probabilistic distribution:

```
v_a = 0 & v_b1 = 0 & v_b2 = 0 ->
                  choice:(1 : (v_a' = 0));
                + choice (1 : (v_a' = 1));
```

Module b contains two state variables and therefore updates rules with parallel assignments. The transformation for these is more difficult as parallel assignments are not directly possible in NuSMV. The single update rule of module b is:

[4]The name of this chooser variable must of course be distinct from all other state variables in the model.

```
true -> choice:(p_b1 : (v_b1' = 0) & (v_b2' = 0) +
                p_b2 : (v_b1' = 1) & (v_b2' = 0) +
                p_b3 : (v_b1' = 1) & (v_b2' = 1))
      + choice:(1 : (v_b1' = 1) & (v_b2' = 1)) ;
```

As outlined above, firstly the two probabilistic distributions are transformed into a set of non-deterministic choices:

```
true -> choice:(1 : (v_b1' = 0) & (v_b2' = 0))         // index 1
      + choice:(1 : (v_b1' = 1) & (v_b2' = 0))         // index 2
      + choice:(1 : (v_b1' = 1) & (v_b2' = 1))         // index 3
      + choice:(1 : (v_b1' = 1) & (v_b2' = 1));        // index 4
```

Each of these four non-deterministic choices[5] is labeled with an index from 1 to 4 and the chooser variable `choose_b_1` (for module `b` update rule number 1) with the set of admissible values $\{1, 2, 3, 4\}$ is defined. The original update rule in the module b is then split into two updates rules, one for the state variable `v_b1` and one for `v_b2`. In addition, the single module containing these two state variables is split into two separate modules, each containing one of the state variables. The activation condition of these update rules becomes the conjunction of the original activation condition (`true`) and the value of the `choose_b_1` variable:

```
module split_b_1
       v_b1 : [0..1] init 0;

       true & choose_b_1 = 1 -> choice:(1 : (v_b1' = 0));
       true & choose_b_1 = 2 -> choice:(1 : (v_b1' = 1));
       true & choose_b_1 = 3 -> choice:(1 : (v_b1' = 1));
       true & choose_b_1 = 4 -> choice:(1 : (v_b1' = 1));
endmodule

module split_b_2
       v_b2 : [0..1] init 0;

       true & choose_b_1 = 1 -> choice:(1 : (v_b2' = 0));
       true & choose_b_1 = 2 -> choice:(1 : (v_b2' = 0));
       true & choose_b_1 = 3 -> choice:(1 : (v_b2' = 1));
       true & choose_b_1 = 4 -> choice:(1 : (v_b2' = 1));
endmodule
```

In this way, for each possible value of `choose_b_1`, there is exactly one active update rule for the state variable `v_b1` and one for `v_b2`. Together, these correspond to exactly one parallel assignment in the original module. The remaining task is the definition of the update rule of the `choose_b_1` state variable. This is simply a non-deterministic choice between any of its possible valuations[6]:

[5] The fourth choice is equivalent to the third and could be eliminated in an optimization.

[6] Note that this does not change any probabilities, as the behavior is purely non-deterministic.

```
      true -> choice:(1 : (choose_b_1' = 1))
            + choice:(1 : (choose_b_1' = 2))
            + choice:(1 : (choose_b_1' = 3))
            + choice:(1 : (choose_b_1' = 4));
```

At each time step, the new value for `choose_b_1` is selected non-deterministically from the set $\{1, 2, 3, 4\}$ of admissible values. The only problem arises at the very first time step, because MDPs (and therefore SAML modules) allow only for a single initial state and therefore the specification of non-deterministic assignments of initial values is not possible. But in order to retain the behavior of the original model, any of the new activation conditions must be satisfiable at the very first time-step. This problem is solved by defining one MDP for each of the possible initial values of the `choose_b_1` chooser variable.

One of the 4 possibilities for the definition of the SAML module for `choose_b_1` is shown below. The other 3 would be completely analogous, but with a different value for the initial state.

```
module choose_1
      choose_b_1 : [1..4] init 1;

      true -> choice:(1 : (choose_b_1' = 1))
            + choice:(1 : (choose_b_1' = 2))
            + choice:(1 : (choose_b_1' = 3))
            + choice:(1 : (choose_b_1' = 4));
endmodule
```

This syntactical transformation has the following effect on the semantic level: the elimination of parallel assignments in a SAML module leads to a set Λ of MDPs. Each element $\tau_i \in \Lambda$

of this set differs from the other elements in its initial state. This effect of the outlined parallel assignment elimination does not directly pose a problem, as the analysis is conducted on the embedded Kripke structures. These allow for the specification of a *set of initial states* (see Def. 14), so that the resulting set of MDPs which differ only by their initial state can be mapped to one single Kripke structure, whose paths then consist of the paths in $\bigcup_{\tau_i \in \Lambda} Paths(\kappa(\tau_i))$.

Nevertheless this can be problematic, as the definition of the semantics of temporal logic properties is based on a single initial state (see Section 3.4.1). Both NuSMV and Cadence SMV verification tools accept specifications of Kripke structures with multiple initial states, but the semantics of the temporal logic formulas is then defined in the following way:

$$\tau_{Kripke}, S_0 \models \phi \Leftrightarrow \forall s_0 \in S_0 : \tau_{Kripke}, s_0 \models \phi$$

This means for example that an existentially quantified CTL* formula $\exists\phi$ which would be valid for one initial state s_0, does not hold for the whole Kripke structure if it is invalid

for *one* of the other initial states. On the other hand, its negation $A\neg\phi$ would also be false, if ϕ holds for one initial state s_0.

This behavior can be rather counter-intuitive and may lead to undesired results. Especially for existentially quantified temporal properties like the DCCA or aDCCA proof obligations. One possible solution to this problem is to verify each formula for each of the generated Kripke structures. An existentially quantified formula then holds if it holds for one of the Kripke structures, and an always quantified formula holds if it holds for each of the generated Kripke structures.

Although valid, such an approach would be rather inefficient and impractical for all but very simple models, as a potentially very large number different Kripke structures were to be analyzed. A much more practical approach is to add an explicit new initial starting state which unifies the set of MDPs into a single one. In the example this would work as follows: add 0 as new initial state of the `choose_b_1` variable and then add a non-deterministic step from this initial state to any of its other admissible states.

The addition of the new update rule with activation condition `choose_b_1 = 0` and the conjunction of all other activation conditions with `!(choose_b_1 = 0)` is then also added to the other SAML modules. If `choose_b_1 = 0` holds, the state variables stay in their initial state. In this way there is one additional initial state for each state variable in the resulting model. Therefore the complete transformation of the example model results in the following:

```
module choose_1
       choose_b_1 : [0..4] init 0;

       true -> choice:(1 : (choose_b_1' = 1))
             + choice:(1 : (choose_b_1' = 2))
             + choice:(1 : (choose_b_1' = 3))
             + choice:(1 : (choose_b_1' = 4));
endmodule

module split_a
 v_a : [0..2] init 0;

       choose_b_1 = 0 -> choice:(1 : (v_a' = 0))
       !(choose_b_1 = 0) & v_a = 0 & v_b1 = 0 & v_b2 = 0 ->
                         choice:(p_a : (v_a' = 0) + (1 - p_a) : (v_a' = 1));
       !(choose_b_1 = 0) & v_a = 0 & v_b1 = 1 & v_b2 = 1 ->
                         choice:(1 : (v_a' = 2));
       !(choose_b_1 = 0) & case3 -> choice:(1 : (v_a' = 1));
       !(choose_b_1 = 0) & v_a = 1 -> choice:(1 : (v_a' = 1));
       !(choose_b_1 = 0) & v_a = 2 -> choice:(1 : (v_a' = 2));
endmodule

module split_b_1
       v_b1 : [0..1] init 0;
```

```
       choose_b_1 = 0 -> choice:(1 : (v_b1' = 0));
       !(choose_b_1 = 0) & true & choose_b_1 = 1 -> choice:(1 : (v_b1' = 0));
       !(choose_b_1 = 0) & true & choose_b_1 = 2 -> choice:(1 : (v_b1' = 1));
       !(choose_b_1 = 0) & true & choose_b_1 = 3 -> choice:(1 : (v_b1' = 1));
       !(choose_b_1 = 0) & true & choose_b_1 = 4 -> choice:(1 : (v_b1' = 1));
endmodule

module split_b_2
       v_b2 : [0..1] init 0;

       choose_b_1 = 0 -> choice:(1 : (v_b2' = 0));
       !(choose_b_1 = 0) & true & choose_b_1 = 1 -> choice:(1 : (v_b2' = 0));
       !(choose_b_1 = 0) & true & choose_b_1 = 2 -> choice:(1 : (v_b2' = 0));
       !(choose_b_1 = 0) & true & choose_b_1 = 3 -> choice:(1 : (v_b2' = 1));
       !(choose_b_1 = 0) & true & choose_b_1 = 4 -> choice:(1 : (v_b2' = 1));
endmodule
```

As all of the safety analyses presented in Chapter 5 are reachability analyses, this is a valid approach for SAML extended system models. The soundness of the outlined transformation is shown in the next section.

6.4.2. Formal Transformation

The general transformation of a SAML model for qualitative analysis works as follows:

1. Eliminate the probabilistic distributions
2. Eliminate the parallel assignments
3. Assemble the resulting set of models to a single one

To show the soundness of the outlined transformation of a SAML model some additional definitions are necessary. To formalize the mapping of a SAML update rule to a set of update rules of solely non-deterministic choices without parallel assignments, an *update tuple* is defined as in Def. 35.

Definition 35. Update Tuple
An update tuple (ϕ, V, Θ) *consists of*

- *a Boolean activation condition* ϕ
- *a set of variables* V
- *a set* Θ *of probabilistic choices* θ *where each* θ *is a set of pairs of the form* $(p, Expr)$ *where* p *is a probability and* $Expr$ *is a set of pairs of state variables* $v \in V$ *and expressions that define their new values.*

The update tuple is a triple of an activation condition ϕ, a set V of variables for which the update rule defines new values and the set of probability distributions. The set of variables is comprised of all the state variables of the SAML module in which the update rule is specified. A probability distribution is defined as a set of pairs $(p, Expr)$ where p is a probability and $Expr$ defines an expression over the state variables and constants for each state variable.

The following update rule of module a has a corresponding update tuple as shown in Eq. (6.1):

```
v_a = 0 & v_b1 = 0 & v_b2 = 0 ->
        choice:(p_a : (v_a' = 0) + (1 - p_a) : (v_a' = 1));
```

$$\left(v_a = 0 \wedge v_{b1} = 0 \wedge v_{b2} = 0, \{v_a\}, \left\{ \left\{ \begin{array}{c} (p_a, \{(v_a, 0)\}), \\ (1 - p_a, \{(v_a, 1)\}) \end{array} \right\}, \right\}\right) \tag{6.1}$$

As this module only holds a single state variable, there is no parallel assignment elimination necessary.

Eq. 6.2 shows the *update tuple* representation of the following update from module b of the example:

```
true -> choice:(p_b1 : (v_b1' = 0) & (v_b2' = 0) +
                p_b2 : (v_b1' = 1) & (v_b2' = 0) +
                p_b3 : (v_b1' = 1) & (v_b2' = 1))
      + choice:(1 : (v_b1' = 1) & (v_b2' = 1)) ;
```

$$\left(true, \{v_{b1}, v_{b2}\}, \left\{ \left\{ \begin{array}{c} (p_{b1}, \{(v_{b1}, 0), (v_{b2}, 0)\}), \\ (p_{b2}, \{(v_{b1}, 1), (v_{b2}, 0)\}), \\ (p_{b3}, \{(v_{b1}, 1), (v_{b2}, 1)\}) \end{array} \right\}, \{(1, \{(v_{b1}, 1), (v_{b2}, 1)\})\} \right\}\right) \tag{6.2}$$

The activation condition ϕ is simply $true$, the set of variables that get a new value assigned in the update rule is $V := \{v_{b1}, v_{b2}\}$ and there are two different probabilistic distributions in Θ. The first, θ_1 has 3 elements $(p_{b1}, \{(v_{b1}, 0), (v_{b2}, 0)\})$, $(p_{b2}, \{(v_{b1}, 1), (v_{b2}, 0)\})$ and $(p_{b3}, \{(v_{b1}, 1), (v_{b2}, 1)\})$. The second, θ_2 has only one element $(1, \{(v_{b1}, 1), (v_{b2}, 1)\})$. Each element of the probabilistic distributions consists of a pair of a probability p and a set of pairs of variables and expressions that define their respective new value in the update.

Parallel Assignment Elimination

To formalize the indexing of the different possibilities of parallel assignments, a function is required that maps each element of a set to a unique index. This is used to specify the index to mark parallel assignments.

Definition 36. Set Element Index

Let η_S be a mapping of each element of a set $s \in S$ to a distinct element of the interval $[1; |S|] \subseteq \mathbb{N}$.

Using this mapping, the elimination of parallel assignments from a parallel update tuple can formally be defined as shown in Def. 37. Here η_S is used to assign an index to the different possible parallel assignments.

Definition 37. Parallel Assignment Elimination

Let (ϕ, V, Θ) be an update tuple u_m, and $\nu(Expr, v)$ be a mapping from the set of expression Expr of new values for the state variables to the new values for the state variable v, the parallel assignment elimination mapping μ is defined as

$$\mu\left((\phi, V, \Theta)\right) := \bigcup_{v \in V} \{(\phi \wedge choose_m = i, \{v\}, (1, \nu(Expr, v)))|(i, Expr) \in \Pi\} \tag{6.3}$$

where

$$\Pi := \{(\eta_T(Expr), Expr)|Expr \in T\}$$
$$T := \{Expr|(p, Expr) \in \theta, \theta \in \Theta\}$$

and $choose_m$ is a new identifier chosen as name of the chooser variable.

This definition can now be applied to the parallel assignment of the example module b. The original update tuple is presented in Eq. 6.2. The first step is to construct the auxiliary sets T and Π.

$$\mathrm{T} = \{\{(v_{b1}, 0), (v_{b2}, 0)\}, \{(v_{b1}, 1), (v_{b2}, 0)\}, \{(v_{b1}, 1), (v_{b2}, 1)\}, \{(v_{b1}, 1), (v_{b2}, 1)\}\}$$

$$\Pi = \left\{ \begin{array}{l} (1, \{(v_{b1}, 0), (v_{b2}, 0)\}), \\ (2, \{(v_{b1}, 1), (v_{b2}, 0)\}), \\ (3, \{(v_{b1}, 1), (v_{b2}, 1)\}), \\ (4, \{(v_{b1}, 1), (v_{b2}, 1)\}) \end{array} \right\}$$

This eliminates the probability distributions and replaces them with non-deterministic choices. The result consists of a set of update tuples where each modifies exactly one state variable and each one has exactly one probability distribution with a single element and probability 1. The non-determinism is expressed in the update tuple that modifies the *choose_b_1* variable which is shown in Eq. (6.6). Each of the sets of update tuples of Eq. (6.4)-(6.6) modifies exactly one state variable. In the transformed model, each module contains exactly one state variable and all the update tuples of one of these sets.

$$\left\{ \begin{array}{l} (true \wedge choose\_b\_1 = 1, \{v_{b1}\}, \{\{(1, \{(v_{b1}, 0)\})\}\}), \\ (true \wedge choose\_b\_1 = 2, \{v_{b1}\}, \{\{(1, \{(v_{b1}, 1)\})\}\}), \\ (true \wedge choose\_b\_1 = 3, \{v_{b1}\}, \{\{(1, \{(v_{b1}, 1)\})\}\}), \\ (true \wedge choose\_b\_1 = 4, \{v_{b1}\}, \{\{(1, \{(v_{b1}, 1)\})\}\}) \end{array} \right\} \tag{6.4}$$

$$\left\{ \begin{array}{l} (true \wedge choose\_b\_1 = 1, \{v_{b2}\}, \{\{(1, \{(v_{b2}, 0)\})\}\}), \\ (true \wedge choose\_b\_1 = 2, \{v_{b2}\}, \{\{(1, \{(v_{b2}, 0)\})\}\}), \\ (true \wedge choose\_b\_1 = 4, \{v_{b2}\}, \{\{(1, \{(v_{b2}, 1)\})\}\}), \\ (true \wedge choose\_b\_1 = 3, \{v_{b2}\}, \{\{(1, \{(v_{b2}, 1)\})\}\}) \end{array} \right\} \tag{6.5}$$

$$\left\{ \left(true, \{choose\_b\_1\}, \left\{ \begin{array}{l} \{(1, \{(choose\_b\_1, 1)\})\} \\ \{(1, \{(choose\_b\_1, 2)\})\} \\ \{(1, \{(choose\_b\_1, 3)\})\} \\ \{(1, \{(choose\_b\_1, 4)\})\} \end{array} \right\} \right) \right\} \tag{6.6}$$

In summary, the parallel assignment elimination of a SAML system model for qualitative analysis works as follows: each update rule with parallel assignments

1. is transformed using the parallel assignment elimination mapping μ (Def. 37)
2. gets an associated new chooser variable which models parallel assignments
3. defines new SAML modules, each containing exactly one of the original state variables

This results in a partitioning of the original model into single modules with only a single state variable and one single module for each newly introduced chooser variable. The parallel composition of this leads to a set of MDPs, each with a different initial state, due to differing initial valuations of the chooser variables.

The behavior of the resulting MDPs is identical to the behavior of the original model with respect to the non-deterministic behavior, represented by the embedded Kripke structures. Lemma (10) states that the paths of the original and the transformed set of MDPs in Λ are equivalent in the following sense: for each path π of the embedded Kripke structure of the original model, there exists a path π' in one of the embedded Kripke structures of the transformed models such that for the projection of π' onto the original state variables var: $\pi'|_{\{var\}} \equiv \pi$ holds. The same holds the other way round for each embedded Kripke structure of each of the MDPs in the set Λ.

Lemma 10. Transformation Equivalence

Let M be a SAML model with state variables $v_1, \ldots, v_k$, and update rules $u_1 \ldots u_m$ and the corresponding MDP τ_{MDP}. Let $U := \bigcup_{i=1}^{m} \{\mu(u_i)\}$ be the set of update tuples of M with eliminated parallel assignments.

Let M' be the parallel composition $M_1 || \cdots || M_k || M_{choose_1} || \cdots || M_{choose_m}$ where M_i is a SAML module with the single state variable v_i and all the updates of the set U of the form (ϕ, v_i, Θ) and M_{choose_i} is a SAML module with the chooser variable $choose_i$ and the update rules for $choose_i$ from U of the form

$$(true, \{choose_i\}, \{\{(1, \{(choose_i, [1, j] \subseteq \mathbb{N}\}\})$$

```
1  eliminiate_parallel_assign (updateSetSAML)
2   newStateVars := samlStateVars
3   updatesNoParallel := ∅
4   for (update u : updateSetSAML)
5     if hasParallelAssigns (u) then
6       chooseVar := getFreshVariable ()
7       newStateVars := newStateVars ∪ {chooseVar}
8       (φ, variableSet, pDistList) := u
9       for (variableName v : variableSAML)
10          for (i = 0; i < |pDistList|; i++)
11            updatesNoParallel := updatesNoParallel ∪
12              {(φ ∧ chooseVar = i, {v}, [getExpr(v, pDistList[i])])}
13        updatesNoParallel := updatesNoParallel ∪
14              {(true, {chooseVar}, {0..|pDistList|})}
15    else
16      updatesNoParallel := updatesNoParallel ∪ {u}
17  return (updatesNoParallel)
```

Listing 6.2: Pseudocode to Eliminate Parallel Assignments from a Set of Update Tuples

Let $\Lambda := \{\tau'_{MDP}\}$ *be the set of MDPs corresponding to the set of possible* M' *with a fixed initial value for each* $choose_i$ *variable, then*

$$\forall \pi : (\pi \in Paths(\kappa(\tau_{MDP})) \Leftrightarrow \exists \tau'_{MDP} \in \Lambda : \pi \equiv \pi'|_{\{v_1,\ldots,v_k\}} \wedge \pi' \in Paths(\kappa(\tau'_{MDP})))$$

proof see p. 174

Listing 6.2 shows the pseudocode to implement the above parallel assignment elimination. Its input is a set of SAML update tuples ($updateSetSAML$). Its output is a set of update tuples with eliminated parallel assignments and the necessary update tuples of newly introduced state variables. The resulting set of update tuples is then used to define new SAML modules, each holding a single state variable.

Single State Extension

As described in the last section, the simple combination of these embedded Kripke structures of the MDPs to a single Kripke structure with multiple initial states can be problematic. Although it is well defined, it can lead to problems with the semantics of temporal logic specification on these Kripke structures. As mentioned before, the intended semantics of the transformations is unfortunately not equivalent to the semantics implemented in the verification tools. Therefore an additional initial state which combines the set of MDPs into a single one is introduced as follows:

```
1  single_initial_state (saml-model)
2   if model_has_parallel_assigns (saml-model) then
3      global_chooser := select_one_chooser_var (saml-model)
4      trans_chooser := add_new_initial_state (global-chooser)
5      chooser_name := get_name (trans_chooser)
6      chooser_init := get_initial_val (trans_chooser)
7      for (module m : get_modules (saml-model))
8        if is_chooser_var (get_state_var (m)) then
9          chooser = get_state_var (u);
10         set_initial_state(chooser,select_one(init_val(chooser)))
11       else
12         for (update-rule u : get_update_rules (module))
13           (φ,varSet,pDistList) := u
14           u := (φ∧ chooser_name ≠ chooser_init,varSet,pDistList)
15         v := get_state_var (module)
16         u := (chooser_name = chooser_init,{v},{{(1,init_val (v))}})
17         add_update_rule (u, m)
18     return (saml-model)
19   else
20     return (saml-model)
```

Listing 6.3: Pseudocode for Single State Extension

- select one of the chooser variables as *global_chooser*
- add a new state 0 to the *global_chooser*
- define 0 as initial state of *global_chooser*
- for each of the other chooser variables use any of their admissible values as single initial state (generally the value 1 as the smallest one)
- for each state variable add a new update tuple of the form

$$(global\_chooser = 0, \{v\}, \{(1, \{v, init(v)\})\})$$

- for each other update tuple of the state variables, set the activation condition to the conjunction of the original one and $\neg global\_chooser = 0$

This means that the global chooser variable is in its *new* initial state 0 for exactly one time-step and then enters one of its admissible states. All other chooser variables have a single well-defined value in the first time-step and all non-chooser state variables stay in their respective initial states for one additional time-step. This means that now, there exists a single initial state in the transformed model. Listing 6.3 shows the pseudocode to add an additional unique single initial state to the SAML model.

This construction leads to a new single initial state s_{-1} of the embedded Kripke structure from which any of the original initial states of the set S_0 can be reached in one time-step via a non-deterministic choice. In particular this means for a SAML model M and a SAML model M' which is the result from the transformation as described above, that for each path of the embedded Kripke structure of M', the postfix of this path has an equivalent path in the embedded Kripke structure of M and vice versa. This is shown in Lemma (11).

Lemma 11. Single Initial State Extension
Let M be a SAML model with state variables $v_1 \dots v_k$ and M' the SAML model which results from parallel assignment elimination and single state extension of M, then

$$\begin{aligned} & \forall \pi : (\pi = s_0 s_1 \dots \in Paths(\kappa(M)) \\ \Leftrightarrow \quad & \exists \pi' = s'_{-1} s'_0 \dots \in Paths(\kappa(M')) : s_0 \equiv s'_{-1}|_{v_1,\dots,v_k} \wedge \forall i \geq 0 : s_i \equiv s'_i|_{v_1,\dots,v_k}) \end{aligned}$$

proof see p. 175

In general this additional initial state must be reflected in the temporal logic proof obligations. But as the transformation is done in such a way that the values of all non-chooser state variables of the original model in s_{-1} are exactly the same as in state s_0, all the qualitative safety analysis methods presented in Section 5.2 are equivalent on both models. The reason for this is that they consist of reachability analyses and the construction does not change the reachable states (for $v_1 \dots v_k$), but only delays everything for one time-step. The safety analysis proof obligations do also not contain Boolean expression over the values of the chooser variables.

Nevertheless this effect must be accounted for if other temporal logic properties should be checked. Then the one time-step delay must be considered by using the next operator to skip the new initial state. For an LTL formula ϕ, this will generally simply be the formula $\mathbf{X}(\phi)$. For CTL formulas, it is a bit trickier. The intended semantics of an existentially quantified ψ_E formula is that "there exists an initial state such that ψ_E holds". This can be accounted for with $\mathbf{E}\,\mathbf{X}(\psi_E)$, i.e. there exists a state s_0 reachable in one time-step from s_{-1} such that ψ_E holds. For an always quantified formula ψ_A, the intended semantics is "ψ_A holds for all possible initial states". This can analogously be expressed in the transformed model as $\mathbf{A}\,\mathbf{X}(\psi_A)$. Note, that parsing and automatic transformation of temporal logic specifications is not implemented at the moment, but such an implementation would be rather straightforward.

For practical purposes, the addition of the new initial state seems to be a more convenient solution than to check every temporal logic property on the multiple Kripke structures resulting from the parallel assignment elimination. Yet it should be noted that other possible semantically sound transformations of SAML models for qualitative safety analysis are conceivable.

For the concrete verification of qualitative properties on the transformed model, it now only has to be expressed in an appropriate format for a verification tool like NuSMV.

6.4.3. Transformation to NuSMV

The specification of the transformed SAML example model as a NuSMV model is shown in Fig. 6.4. Analogous to SAML models, it is also structured in separate modules, here labeled with the `MODULE` keyword. In the declaration of the modules, the state variables of all other modules must explicitly be passed as input variables.

Inside the modules, the declaration of the domain of the state variables is located in the variable declaration part (keyword `VAR`). In the assignment (keyword `ASSIGN`) part, the initial value is defined via the `init` keyword. The update rules for the new values are defined via `next`. The different activation conditions are treated as separate cases, the new values for the state variable are given after the colon. All cases are enclosed between the `case` and `esac` keywords. The parallel composition of the modules in NuSMV is synchronously parallel, analogous to SAML. The semantics of the assignment with `next(v)` is equivalent to the assignment to `v'` in SAML. This subset of the full specification language of NuSMV is sufficient to express the transformed model, a detailed overview on the syntax of the input language of NuSMV can be found in the manual by Cavada et al. [CCOR11].

Non-deterministic assignments are expressed as the assignment of a *set* of possible new values. This can be seen in the very first update rule of the module `A`. In the original model this was a probabilistic choice to assign `V_A` the new value 0 with probability p_a and the value 1 with probability $1 - p_a$. This is transformed into the non-deterministic assignment of the set $\{0, 1\}$ to `next(V_A)`.

```
next (V_A) := case
((!(G1332 = 0)) & (((V_A = 0) & (V_B1 = 0)) & (V_B2 = 0))) : {1,0};
...
esac;
```

The names for the introduced chooser variables are selected randomly from a pool of unused variable names. The update rules for the chooser variable `G1332` of the module `SPLIT_G1332` consist of non-deterministic assignments of any of its admissible values. Its initial value is 0 as it is used as the global chooser variable for the single state extension and indicates the very first time-step of the system.

The new module `main` holds the instances of all the other modules and corresponds to the *main* function of many programming languages which marks the entry point for program execution. This module is responsible to provide an instance of each module and to pass the state variables explicitly as parameters and is required in all NuSMV models.

```
MODULE SPLIT_V_B2 (V_A,G1332,V_B1)

VAR
V_B2 : 0..1;

ASSIGN
init (V_B2) := 0;

next (V_B2) := case
((!(G1332 = 0)) & ((G1332 = 4) & TRUE)) : 0;
((!(G1332 = 0)) & ((G1332 = 3) & TRUE)) : 0;
((!(G1332 = 0)) & ((G1332 = 2) & TRUE)) : 1;
((!(G1332 = 0)) & ((G1332 = 1) & TRUE)) : 1;
(G1332 = 0) : 0;
esac;

MODULE SPLIT_V_B1 (V_A,G1332,V_B2)

VAR
V_B1 : 0..1;

ASSIGN
init (V_B1) := 0;

next (V_B1) := case
((!(G1332 = 0)) & ((G1332 = 4) & TRUE)) : 0;
((!(G1332 = 0)) & ((G1332 = 3) & TRUE)) : 1;
((!(G1332 = 0)) & ((G1332 = 2) & TRUE)) : 1;
((!(G1332 = 0)) & ((G1332 = 1) & TRUE)) : 1;
(G1332 = 0) : 0;
esac;
MODULE SPLIT_G1332 (V_A,V_B1,V_B2)

VAR
G1332 : 0..4;

ASSIGN
init (G1332) := 0;

next (G1332) := case
TRUE : {1,2,3,4};
esac;

MODULE A (G1332,V_B1,V_B2)

VAR
V_A : 0..2;
```

```
DEFINE
CASE3 := (((V_A = 0) & (!(((V_B1 = 0) & (V_B2 = 0))
                    | ((V_B1 = 1) & (V_B2 = 1))))));

ASSIGN
init (V_A) := 0;

next (V_A) := case
((!(G1332 = 0)) & (((V_A = 0) & (V_B1 = 0)) & (V_B2 = 0))) : {1,0};
((!(G1332 = 0)) & (((V_A = 0) & (V_B1 = 1)) & (V_B2 = 1))) : 2;
((!(G1332 = 0)) & CASE3) : 1;
((!(G1332 = 0)) & (V_A = 1)) : 1;
((!(G1332 = 0)) & (V_A = 2)) : 2;
(G1332 = 0) : 0;
esac;


MODULE main
VAR
G1333 : SPLIT_V_B2(G1336.V_A,G1335.G1332,G1334.V_B1);
G1334 : SPLIT_V_B1(G1336.V_A,G1335.G1332,G1333.V_B2);
G1335 : SPLIT_G1332(G1336.V_A,G1334.V_B1,G1333.V_B2);
G1336 : A(G1335.G1332,G1334.V_B1,G1333.V_B2);
```

Figure 6.4.: NuSMV Specification of the SAML Model in Fig. 6.2

This model specification can then be analyzed with the NuSMV model checker. For each state variable of the original SAML model, there exists exactly one corresponding state variable in the transformed model. Therefore counterexamples computed by NuSMV can easily be traced back into the SAML model (ignoring the additional initial state).

This syntactical transformation is sound, as only the parallel composition and the assignment of new values to state variables is NuSMV specific. The module composition is also synchronously parallel and the assignment for the next values is equivalent to the semantics of the assignment in SAML. Therefore the SAML model with the eliminated parallel assignments and the single initial state is semantically equivalently expressible as a NuSMV model in this way.

6.5. Related Work

In the Topcased project [VPF+06], Farail et al. [FGP+08] use the formal language FIACRE as an intermediate language for formal analysis. Different models (mainly in AADL) which are constructed in the Eclipse Modeling Framework (EMF) are transformed into FIACRE, the FIACRE models are then transformed into the verification

tools of the CADP toolbox developed by Fernandez et al. [FGK+96] or the TINA petri net analyzer developed by Berthomieu et al. [BaFV04]. Due to the amount of modeling artifacts that are supported in FICARE, only structural properties of systems can be proven, the models are too complex for a successful verification of behavioral aspects [FGP+08]. SAML has deliberately been designed to be as simple as possible for formal analysis while retaining the expressiveness needed for extended system modeling.

The SLIM formalization by Bozzano et al. [BCK+09b] of an AADL subset and the AADL error annex [SA06] developed in the COMPASS project [BCK+10b] can also transformed into different analysis tools for either qualitative or quantitative analysis. SLIM separates the specification in nominal and failure parts. The nominal part, which can also include continuous variables with change rate in addition to discrete state variables, is transformed into a variant of the NuSMV model checker which also supports satisfiability modulo theory (SMT) solving techniques that can solve such hybrid system behavior. This is achieved by the integration of the MathSAT SAT solver developed by Bruttomesso et al. [BCF+08] into the COMPASS toolchain.

The probabilistic failure modeling uses continuous time models which are analyzed with the MRMC model-checker. This is called performability analysis in COMPASS. The transformation consists of two steps: firstly, the state space of the SLIM model is generated. The semantics allows instantaneous transitions which are not probabilistic and do not consume time and therefore are of no effect on the quantitative model. In a second step, all states resulting from such transitions are collapsed together using the "lumping" mechanism. This is conducted using bisimulation minimization of the labeled transition system with the SigRef tool developed by Wimmer et al. [WHH+06] in the AVACS project. Nevertheless, this performability analysis is only possible on models that do not express such hybrid behavior [Ngu10]. The necessary transformations for SLIM are realized on the semantic level and are directly integrated into the verifications tools (NuSMV and MRMC are developed by groups participating in COMPASS). Therefore it is not easily possible to use other verification tools for the analyses (e.g. like PRISM for quantitative analysis of larger models) as with the tool-independent approach based on SAML.

Summary

This chapter presented model transformations which allow for safety analysis of extended SAML models with different verification tools. At the same time, they guarantee that the transformed models are equivalent in some sense, rendering the approach tool-independent. Therefore it benefits directly from improved analysis tools and it is also possible to add newly developed analysis tools. The currently implemented transformations allow to use PRISM and MRMC for quantitative analysis and NuSMV and Cadence SMV for qualitative analysis. The transformations are on the syntactical level, which facilitates the integration of additional verification tools.

The transformations of SAML models into the input language of PRISM requires the explicit synchronization of the update rules and the transformation of non-deterministic choices into update rules with equivalent activation conditions.

The transformation for qualitative analysis is more complex. Many analysis tools do not allow for parallel assignments for multiple state variables which is possible in SAML. This problem was solved by the introduction of one additional state variable per parallel assignments that signals which possibility was chosen. The result of this transformation is a set of SAML models, for which path inclusion equivalency holds. It is possible to combine this set of models into a single equivalent model of a Kripke structure, on which all of the qualitative model-based safety analyses can be conducted. This can be analyzed using qualitative verification tools like the different SMV variants.

7. Case Studies

Programming is usually taught by examples.

(Niklaus Wirth)

This chapter presents three case studies which are analyzed using the formal model-based analysis methods presented in the previous chapter. It describes how systems from different application domains can be modeled and analyzed. The chapter discusses advantages as well as limitations of the modeling and model-based safety analysis methods presented in this dissertation. The results of the analysis of each case study are presented, as well as information about the computational effort that is required to conduct the analyses.

A larger industrial case study from the railway domain is described in Section 7.1. Section 7.2 presents the full modeling of the case study which was used in Chapter 4 and Chapter 5. Section 7.3 discusses a case study of a self-healing system from production automation. Section 7.4 concludes the chapter with some discussion about related work on the selected case studies.

7.1. Radio-Based Railroad Control

This case study is currently the largest and most realistic one that has been analyzed using the methods described in the last chapters. It clearly shows that the developed techniques are applicable in a real-world context.

7.1.1. Description

The following case study of a radio-based railroad control was used as a reference case study in the priority research program 1064 "Integrating software specifications techniques for engineering applications" of the German Research foundation (DFG). It was supplied by the German railway organization, Deutsche Bahn and addresses a proposed novel technique for controlling railroad crossings. This technique aims at medium speed routes with a maximum speed of $160\frac{km}{h}$. It is described Reif et al. in [RST00] where a first FTA of the case study was presented. Its purely qualitative modeling and safety analysis using DCCA was presented by Ortmeier et al. in [ORS05] and its quantitative modeling and safety analysis with pDCCA in [GO10c].

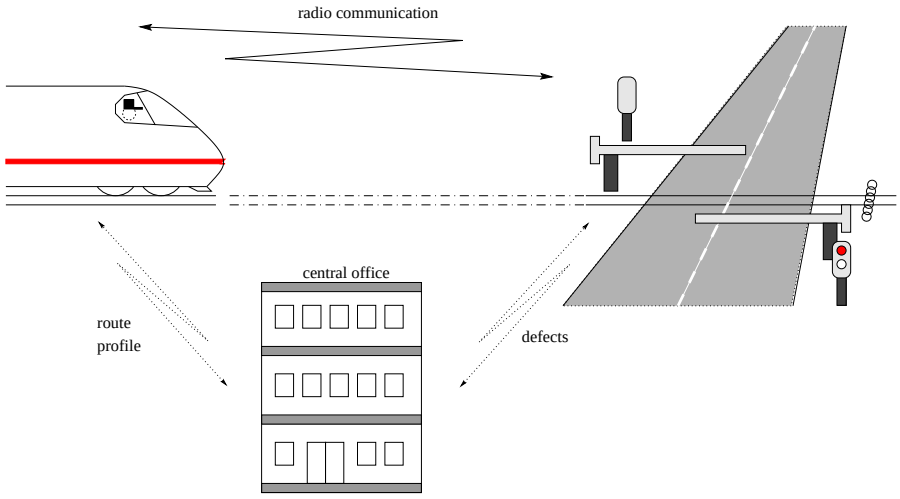


Figure 7.1.: Radio-based Railroad Crossing [RST00]

The following description of the case study is taken directly from Reif et al. [RST00]:

> "The main difference between this technology and the traditional control of level crossings is, that signals and sensors on the route are replaced by radio communication and software computations in the train and in the level crossing. This offers cheaper and more flexible solutions, but also shifts safety-critical functionality from hardware to software.

> Instead of detecting an approaching train by a sensor, the train computes the position where it has to send a signal to secure the level crossing. Therefore the train has to know the position of the level crossing, the time needed to secure the level crossing, and its current speed and position, which is measured by an odometer.
>
> When the level crossing receives this command, it switches on the traffic lights, first the *yellow* light, then the *red* light, and finally closes the barriers. When they are closed, the level crossing is *safe* for a certain period of time. The stop signal, indicating an insecure crossing, is also substituted by computation and communication. The train requests the status of the level crossing. Depending on the answer the train will brake or pass the crossing. The level crossing periodically performs self-diagnosis and automatically informs the central office about defects and problems. The central office is responsible for repair and provides route descriptions for trains. These descriptions indicate the positions of level crossings and maximum speed on the route."

For the formal modeling in SAML and model-based safety analysis, the case study is adapted as follows: The train continuously monitors its position and its speed via the odometer. When it approaches the crossing, it initiates the communication by sending a *secure* request. When the crossing receives this, it starts closing the barrier. Shortly before the train reaches the *latest braking point* – the smallest distance to the barrier where it is possible for the train to stop in front of the crossing with its current speed – it requests the status of the crossing. The crossing acknowledges this signal if the sensor at the barrier detects that the barrier is closed. If there is no *secured* signal, the train brakes immediately. The passing of the train is detected by a sensor located after the crossing, allowing the crossing to reopen the barrier. The central office was not modeled for the analysis.

7.1.2. Modeling

The structure of the modeling of the case study is similar to the original modeling for Statecharts presented by Ortmeier et al. [ORS05]. But of course, here the quantitative information is directly integrated into the extended system model. In return, a more accurate modeling of the case study, in particular of the probabilistic failure mode modeling and of the physical environment, and a more accurate quantitative analysis with pDCCA is possible.

Modeling of Hardware and Software

The hardware and software modeling in this case study is the crossing itself which is modeled by its states, the crossing control which is responsible to control the state changes of the crossing and the control of the train.

Crossing The control of the crossing is triggered by signals from the communication with the approaching train. The module representing the crossing is shown in Fig. 7.2. The state variable `crossing` has three possible states: an *opened* state which is also the initial state, a *closing* state and a fully *closed* state. The state *opened* is left when the activation condition *close* holds, which means that the train has sent its closing request to the crossing and the request has been received. This also starts an (external) timer and the crossing is in state *closing* as long as the timer counts ($timer_1 > 0$). The closing of the barrier of the crossing needs some time, 30s in the model of the case study. When this has passed, the crossing enters state *closed*. This starts another timer and the crossing gets opened again (enters state *opened*) when either a predefined amount of time elapses ($timer_2 = 0$)[1] or the $trainPassed$ signal arrives which announces that the train has passed the crossing.

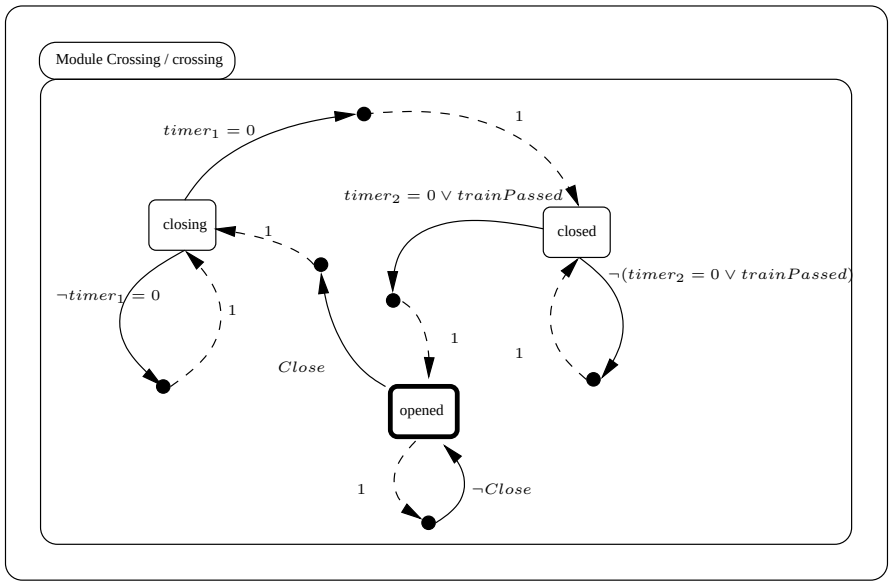


Figure 7.2.: Original Crossing Model

Train Control The behavior of the train is defined by the train control. It models the current state of the train. The train control continuously monitors the speed of the train as well as its position and initiates the communication with the crossing in such way that it can still stop before the dangerous position if the crossing is not closed. To compute the positions from where to initiate the communication and when to brake if there is no answer received, the train uses the reading from the odometer of the train called *virtual_speed*.

[1]This is a requirement from the Deutsche Bahn to assure that people who become impatient after a long waiting and decide to cross the track anyway do so at least fast.

The corresponding module is shown in Fig. 7.3. It consists of the state variable `trainControl`. Its initial state is *idle* which means that the train approaches the crossing. When the train determines that the position to initiate the closing of the crossing is reached (*posCloseReached* holds), it enters the state *waitClose* where it stays until the status reporting position has been reached at which it requests the state of the barrier of the crossing. It then enters the state *waitSig* where it stays until either the crossing signals that it is closed (*ackReceived* holds) or the last breaking position has been reached (*posBrakeReached* holds). If the crossing signals its closed state, the train enters the *ok* state and passes the crossing without braking.

If the last breaking position has been reached without a signal from the crossing, the train brakes to prevent entering the crossing and the state of the train control is set to *brake*. All the positions are not fixed, but are computed by the train control dependent on the *virtual_speed* reading of the odometer.

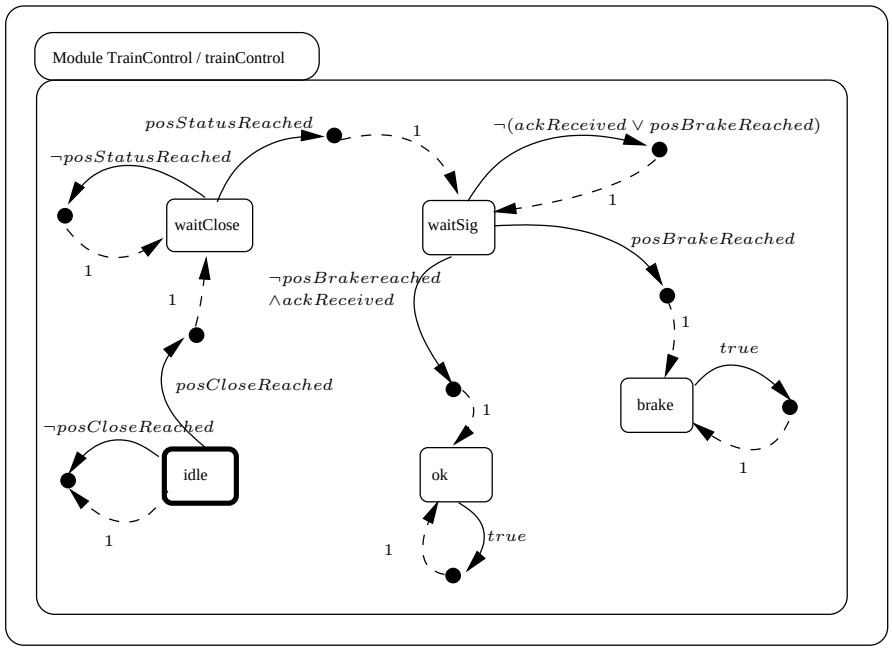


Figure 7.3.: Train Control

Modeling of Physical Behavior

The modeling of the physical behavior of the train covers mainly its movement. In order to be realistic, not only the current position and velocity must be considered, but also the modeling of acceleration and deceleration (braking). This is separated into three

parts: the position, the speed and the acceleration[2].

Train Position The movement of the train is modeled as the change of this position. The train is therefore modeled as a module with a state variable that tracks the current train position, as shown in Fig. 7.4. The update rules define the value of the change of the corresponding state variable. The railroad track that is of interest is $10km$ long and the state variable representing the train position has to cover this distance. Its domain is the interval from 0 to 1000, one spatial unit therefore corresponds to $10m$. The danger spot, i.e. the position at which the train must come to a halt if the crossing is not closed, is at the $9km$ position (`pos_gp = 900`). The position of the sensor, that detects whether the train has passed, is located at $9.5km$ (`pos_sensor_passed = 950`).

The first update rule changes the the position of the train by adding the current speed to the current position. This works, as the speed is defined as rate of change per time-unit. The activation condition of the second update rule is active once the train has completely passed by the crossing and continues on the track outside the observed $10km$. Therefore the position of the train will not get larger than 1000.

```
module TrainPos
  trainPos : [0..1000] init 0;

  trainPos + trainSpeed < 1000  -> choice (1 : (trainPos' = trainPos + trainSpeed));
  !(trainPos + trainSpeed < 1000) -> choice (1 : (trainPos' = 1000));
endmodule
```

Figure 7.4.: Train Position Module

Train Speed The speed of the modeled train on such a track is assumed to be $v = 115\frac{km}{h}$ or $32\frac{m}{s}$. For a temporal resolution of $\Delta t = 5s$, i.e. for every discrete system time step 5 seconds pass, this means that the position of the train increases by $\frac{(\Delta t \cdot v)}{10m}$ units. The current *real* speed of the train is modeled by a second module holding the `trainSpeed` state variable. The corresponding module is shown in Fig. 7.5. The initial value is 16 which corresponds to $32\frac{m}{s}$ and is also the maximal speed. The minimal speed is 0.

Train Acceleration The value of `trainSpeed` on the other hand depends on the value of the acceleration of the train. The current acceleration is modeled in a third module as shown in Fig. 7.6. It contains the `trainAcc` state variable with specifies the current acceleration of the train and has the domain $\{-1, 0, 1\}$. The meaning of an acceleration

[2]This is an example of the general approach to model physical units via their rate of change, as mentioned in Sect. 4.4

```
module TrainSpeed
  trainSpeed : [0..16] init 16;

  (trainSpeed + trainAcc < 0) -> choice (1 : (trainSpeed' = 0));
  (trainSpeed + trainAcc > 16) -> choice (1 : (trainSpeed' = 16));
  !(trainSpeed + trainAcc < 0 | trainSpeed + trainAcc > 16) ->
     choice (1 : (trainSpeed' = trainSpeed + trainAcc));
endmodule
```

Figure 7.5.: Train Speed Module

of value 1 is that `trainSpeed` changes 1 unit in one time-step. As one `trainSpeed` unit corresponds to $2\frac{m}{s}$ and $\Delta t = 5s$ this means that if `trainAcc` has the value 1, the acceleration is $\frac{2}{5}\frac{m}{s^2}$ (and $-\frac{2}{5}\frac{m}{s^2}$ for the deceleration with `trainAcc` = -1).

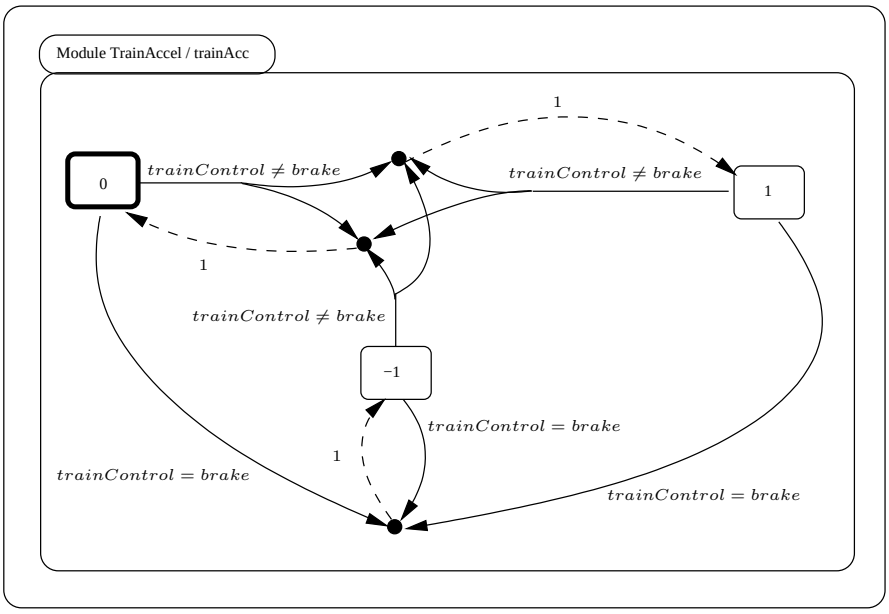


Figure 7.6.: Train Acceleration Module

Modeling of Failure Modes

In this case study, different failure modes are possible. The relevant ones for the safety analysis are the following: failure of the brakes (*error_brake*) which models that the train does not decelerate as intended, failure of the communication between the train and the crossing (*error_comm*), failure of the crossing closed sensor (*error_closed*) which results in wrongly signaling a closed barrier, failure of the actuator of the barrier (*error_actuator*) which means that the barrier does not close as intended, failure of the sensor that detects

the passage of the train (*error_passed*) and finally a deviation of the odometer of the train (*error_odo*) which means that the train is faster or slower than reported by its odometer, i.e. the *virtual_speed* – which is used by the train to compute the points to communicate with the crossing and the last braking point – is not equal to the value of `trainSpeed`.

Error_passed This failure mode models the malfunction of the sensor that detects whether the train has passed the crossing. A malfunction can either be a signal that a train has passed although there was none or omitting a signal although a train has passed. This failure is integrated into the original model of the crossing (see Fig. 7.2).

The misdetection of the *trainPassed* signal is integrated as shown in Fig. 7.7. The crossing can now also leave the state *closed* if the failure module corresponding to the *error_passed* failure mode is in state *yes*, i.e. the sensor misdetects a passed train and the crossing gets opened as result.

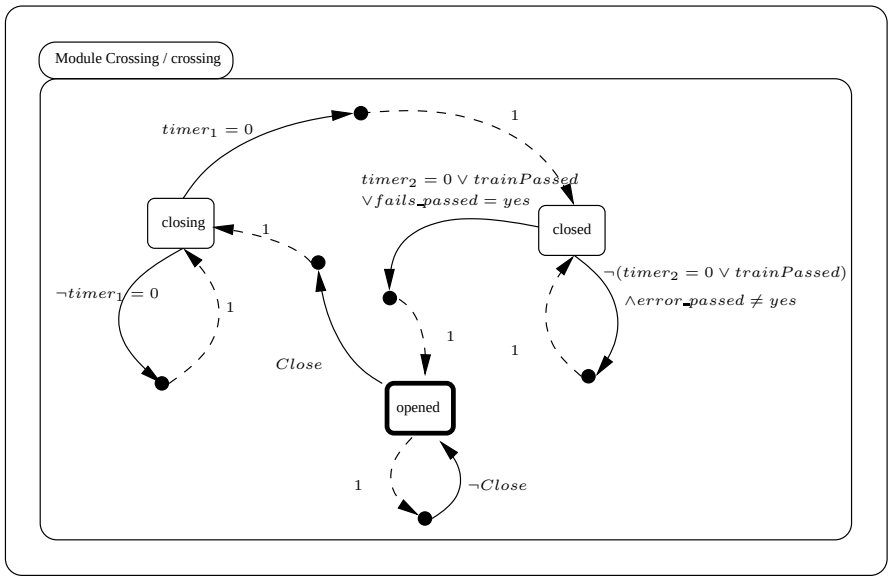


Figure 7.7.: Failure Effect Modeling of *error_passed*

The occurrence pattern of the failure mode is modeled as follows: The failure rate λ_{error_passed} is assumed to be $7 \cdot 10^{-9} \frac{1}{s}$. This means that the expected value of failures per time-step with $\Delta t = 5s$ of this sensor is $5s \times \lambda_{error\_passed} = 3.5 \cdot 10^{-8}$ which is used as the per-step failure probability for the modeling of a transient per-time failure module. The failure occurrence pattern is modeled transiently, as the intended failure mode is a sensor misdetection which is only a temporary disturbance.

Error_comm This failure mode models the malfunction of the radio communication between the arriving train and the railroad crossing such that the requested *close* com-

mand is not received. In Fig. 7.8 the module representing the crossing with the failure effect modeling of both the *error_passed* as well as the *error_comm* failure mode is shown together with the parallel failure modules for the failure modes.

The integration of *error_comm* is achieved by splitting the activation condition *Close* of the original module into $Close \wedge (error\_comm \neq yes)$[3] for the original behavior and $Close \wedge (error\_comm = yes)$ for the failure effect. Whenever $Close \wedge (error\_comm = yes)$ holds, the state *opened* is not left although the train tries to signal the *Close* command and therefore the demand to close the crossing fails.

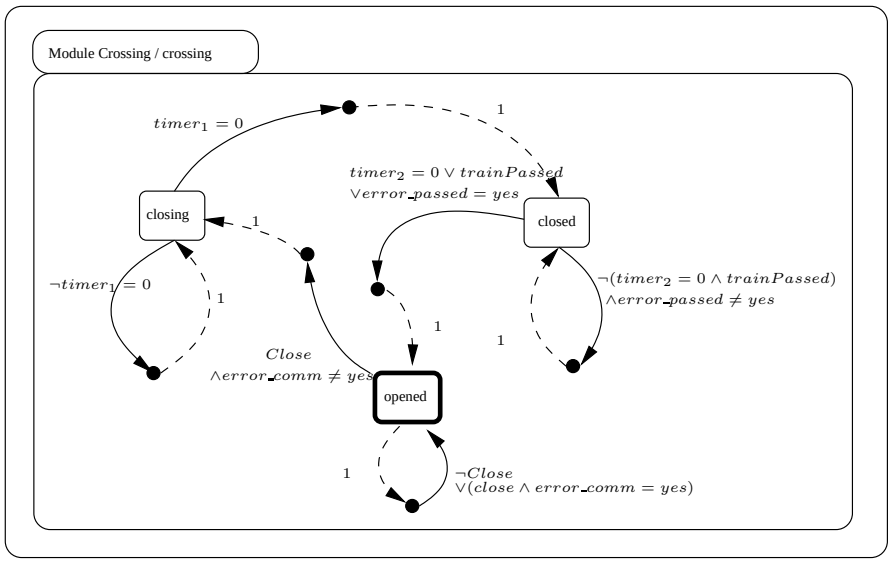


Figure 7.8.: Integration of *error_passed* and *error_comm*

The failure mode is integrated as a per-demand failure mode with a per-demand failure probability of $3.5 \cdot 10^{-5}$. In the case study, there are generally three demands to the communication function. The first one is the initial command to close the barrier sent from the train to the crossing. In this case the *demand* is that the crossing is in state *opened* and that the *Close* command is sent by the arriving train. This modeling is shown in the per-demand failure module in Fig. 7.9 modeling the occurrence pattern of *error_comm*. The second demand is when the train requests the state of the barrier and the third demand is when the crossing acknowledges a correctly secured barrier.

Error_closed This failure mode models the effect that the secured signal is sent from the crossing to the train, although the barrier of the crossing is not closed. This means that the sensor at the crossing misdetects the closing of the barrier. The failure mode is modeled in the following way: the secured signal of the crossing is sent to the train

[3]The value of *error_comm* would be either 0 or −1 in the model to signal the absence of the failure mode.

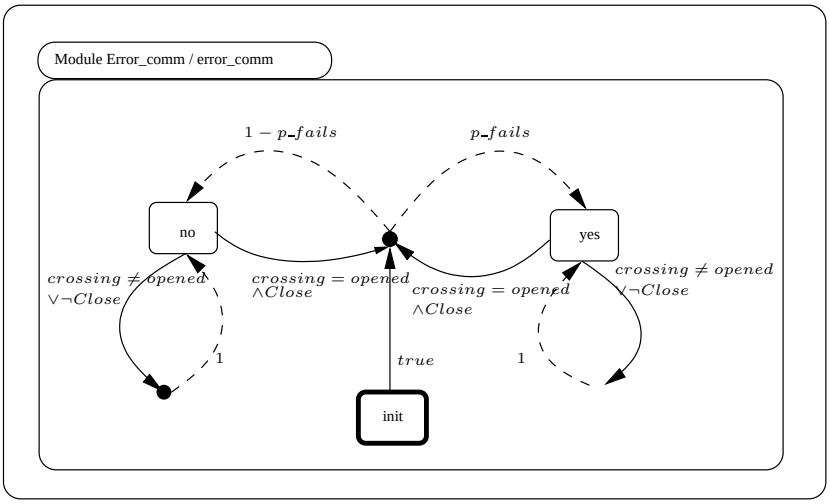


Figure 7.9.: Per-demand Failure Module for *error_comm*

although the crossing is not closed and the train mode therefore changes from *waitSig* to *ok* and passes the crossing without braking. This can be seen in Fig. 7.10, the activation condition of the transition from state *waitSig* to *ok* can now also become true if *error_closed=yes* holds. The activation conditions of the transitions to state *brake* or *waitSig* can only be active if the failure mode does not occur.

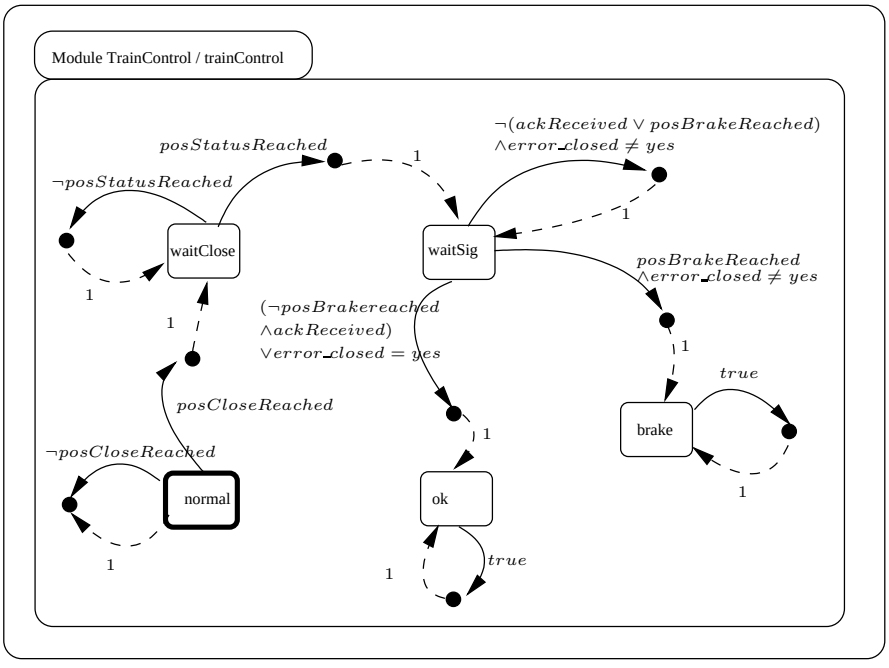


Figure 7.10.: Modeling of *error_closed*

Error_actuator This failure mode models the fact that the actuator of the barrier does not work correctly and therefore the crossing gets stuck and does not close properly.

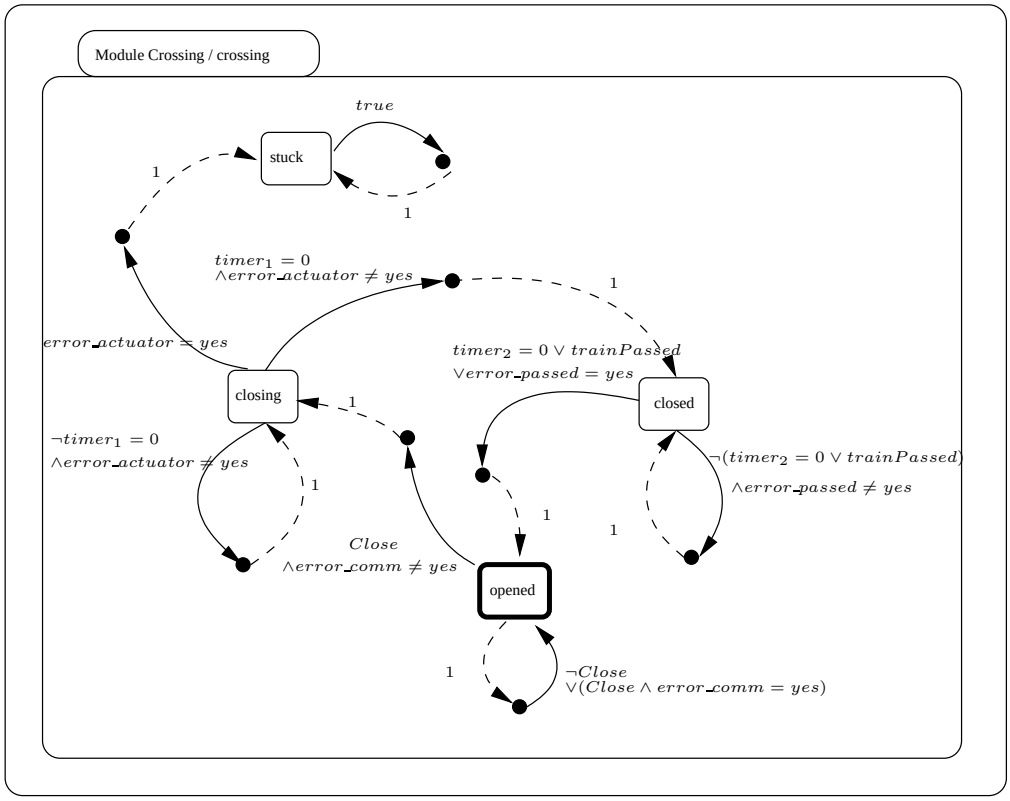


Figure 7.11.: Integration of *error_passed*, *error_comm* and *error_actuator*

The effects modeling is shown in Fig. 7.11. The integration of the failure effect is modeled with an additional state. The model of the crossing is extended with the additional state *stuck*. This state is reached when the *error_actuator* failure mode occurs while the crossing is in state *closing*. Once this state is entered, it cannot be left again, i.e. the barrier is stuck and the crossing stays opened.

This failure mode is integrated as a persistent per-time failure mode with a failure rate of $7 \cdot 10^{-9} \frac{1}{s}$. The per-time failure modeling was chosen, because the effect of the failure mode is that the barrier can get stuck at any time while closing and not only directly at the start of the movement.

Error_brake This failure modes models the malfunctioning of the brakes of the train. Its effect is therefore integrated into the train acceleration model. If the train does not brake, it can choose non-deterministically between keeping constant speed ($trainAcc = 0$) or accelerating ($trainAcc = 1$). If the train brakes, then the train decelerates

($trainAcc = -1$). The malfunctioning of the brakes is modeled that the train does not decelerate although $trainControl = brake$ holds.

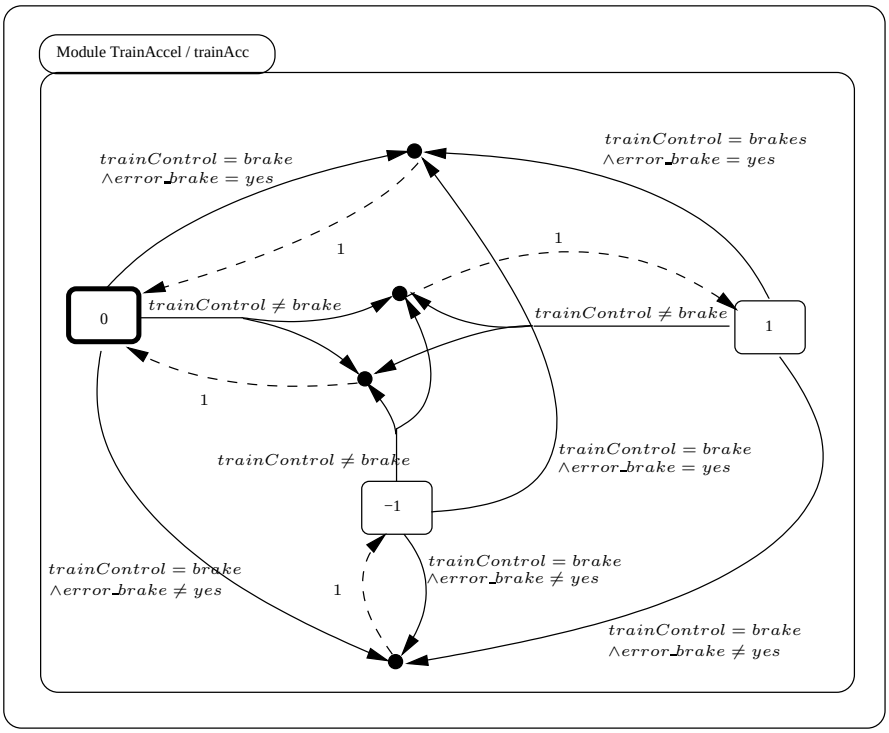


Figure 7.12.: Integration of *error_brake*

The failure is integrated into the `TrainAcc` module as shown in Fig. 7.12. All original updates rules with activation condition $trainControl = brake$ are split up into two orthogonal update rule with the activation conditions $trainControl = brake \wedge error\_brake \neq yes$ and $trainControl = brake \wedge error\_brake = yes$. When $error\_brake = yes$ holds, the only possibility is an acceleration of 0, i.e. continuing at constant speed.

The failure mode is modeled as per-time failure mode with a failure rate of $7e^{-9}\frac{1}{s}$. Although for this failure mode, one could also argue for a per-demand failure modeling. The demand in this case would be the deceleration of the train, the failure effect modeling would not change, only the failure module would have to be adjusted accordingly. But as with the failure mode *error_actuator*, the failure of the brakes can appear in a longer time interval and not only at the moment of activation of the brakes.

Error_odo This failure mode models the deviation of the measured speed of the train from the real speed. As mentioned before, the train uses the reading of the odometer as the *virtual_speed* to compute the position from where to initiate the communication and to compute the last braking point.

Whenever the failure mode occurs, the actual reading of the odometer is chosen from a discrete approximation of a normally distributed deviation with mean $\mu = 0\frac{m}{s}$ and a standard deviation of $\sigma = 4\frac{m}{s}$ and a maximal deviation of $\pm 6\frac{m}{s}$. The deviation is stored in the state variable `errorOdoSkew`, the corresponding module is shown in Fig. 7.13. The deviation distribution is reflected in the probabilities of the update rules of the `errorOdoSkew` state variable.

```
module ErrorOdoSkew
errorOdoSkew : [-3..3] init 0;

 error_Odo = 1 -> choice (
     0.0059770  : (errorOdoSkew' = -3) +
     0.060598   : (errorOdoSkew' = -2) +
     0.24173    : (errorOdoSkew' = -1) +
     0.38339    : (errorOdoSkew' =  0) +
     0.24173    : (errorOdoSkew' =  1) +
     0.060598   : (errorOdoSkew' =  2) +
     0.0059770  : (errorOdoSkew' =  3));
 error_Odo = 0 -> choice (1 : (errorOdoSkew' = 0));
endmodule
```

Figure 7.13.: Deviation of Odometer for *error_odo*

The effect of the occurrence of this failure mode is as follows: the value of the state variable `errorOdoSkew` is used to compute the *virtual_speed* as the deviation of the real speed of the train, i.e. $virtual\_speed = trainSpeed + errorOdoSkew$. The occurrence pattern of the *error_odo* failure mode is modeled as a transient per-time failure mode with a failure rate of $6 \cdot 10^{-5}\frac{1}{s}$.

7.1.3. Results

DCCA

The safety analysis results were computed on a 8 core 2.66 Ghz Xeon computer with 16G RAM. The hazard for which the safety analysis was conducted is "the train is on the crossing but the crossing is not closed". This is formulated as:

$$crossing \neq closed \wedge trainPos < pos\_gp \wedge trainPos + trainSpeed \geq pos\_gp$$

This means that the train passes the danger point (*pos_gp*) at a time when the crossing not closed. In the formal model this means that the train position is before the danger point at the current time-step, but will reach or cross the danger point in the next time-step because of its current speed. This is a potential cause for an accident and is therefore a hazard. The embedded Kripke structure of the extended system model

was analyzed with DCCA using the NuSMV model-checker version 2.5. The analysis required 14.518s and lead to the following minimal critical sets:

- $\Gamma_1 := \{error\_passed\}$
- $\Gamma_2 := \{error\_odo\}$
- $\Gamma_3 := \{error\_comm,\ error\_closed\}$
- $\Gamma_4 := \{error\_comm,\ error\_brake\}$
- $\Gamma_5 := \{error\_closed,\ error\_actuator\}$
- $\Gamma_6 := \{error\_actuator,\ error\_brake\}$

The failure mode *error_passed* is a single point of failure. It is critical when the sensor misdetects that the train has passed and therefore the crossing opens again. The *error_odo* failure mode is only critical as a single point of failure if the deviation is large enough (in this case up to $6\frac{m}{s}$). Then the deviation of the real from the virtual speed is large enough and the train brakes too late. The set Γ_3 describes the situation that the sensor at the barrier fails and therefore the signal to proceed is sent to the train, but the barrier did not receive the close signal and therefore stays in state *opened*. The set Γ_4 describes the situation that the train cannot signal the close signal to the crossing, but can also not brake and therefore enters the crossing while the barrier is opened. Γ_5 describes the situation that the crossing actuator malfunctions, i.e. the crossing does not close, but the sensor of the crossing misdetects it as closed. Finally, set Γ_6 describes the situation that the crossing does not close correctly and the train is not able to brake in time because of the malfunctioning of its brakes. Note that this failure mode is *not* a single point of failure (a minimal critical set of size 1) as it was conjectured in earlier analyses based on FTA [RST00, TO03].

Deductive Ordering Analysis

For the minimal critical sets with two elements, deductive ordering analysis was conducted. Using the NuSMV model-checker version 2.5, this required 30m and 47.6s and revealed that there are no temporal dependencies between the failure modes.

pDCCA

This case study is a good example for a non-reactive system. The train passes the crossing exactly once and only this hazard probability is of interest. Therefore pDCCA can directly be applied for quantitative safety analysis without specifying a mission time.

For pDCCA, the resulting model consisted of 21.491.073 states, 803.961.338 probabilistic transitions and 22.268.090 non-deterministic choices. The maximal occurrence probability of the hazard was computed to be:

$$P_{max}(H) = 2.7331 \cdot 10^{-6}$$

Using the PRISM model-checker version 3.3, the computation required 3.7G RAM and an analysis time of $12m$ and $9.4s$ in sparse matrix representation and 36M RAM and an analysis time of $32m$ and $2.1s$ for MTBDD representation. Note, that both methods compute the same hazard probability only up to a certain precision, due to different numerical methods. In this case the difference is smaller than 10^{-13}. If this is of concern, the threshold values for the algorithms can be adjusted accordingly in the configuration of the verification tools.

Advantage over existing methods For this case study, the major advantage of the presented model-based safety analysis method is the more accurate determination of the occurrence probability of the hazard. The existing more traditional analysis methods are based on an a-posteriori estimation of the hazard probability on the resulting critical combinations of failures (e.g. see [ORS05]). The following equations show the result using the standard approach based on the quantitative fault tree analysis (FTA) formula as shown in Eq. (7.1).

$$P_{FTA}(H) \leq \sum_{\Delta \in \text{minimal cut sets}} \prod_{\delta \in \Delta} P(\delta) \tag{7.1}$$

This estimation relies on the stochastic independence of the occurrence of the failure modes. Therefore the occurrence probabilities of failure modes in the same minimal cut (or critical) set are multiplied and the estimated occurrence probabilities of all minimal cut sets then summed together.

In the case study there are two single point of failures, *error_passed* and *error_odo*. Both are per-time failure modes, so the time interval to consider in the analysis must be defined. For this (see [GO10c]), the probabilities for *error_passed* and *error_odo* were estimated for *one* train passing the crossing which in the example is roughly[4] 312s. As the occurrence probabilities of the other failure modes in sets of size two are of similar value but are multiplied, their effect on the overall hazard probability is basically neglectable, as it is in the order of $O(1 \cdot 10^{-14})$, see Eq. (7.2).

$$P_{FTA}(H) \leq P(error\_passed) + P(error\_odo) + P(\text{cut sets of size } \geq 2) \tag{7.2}$$
$$P_{FTA}(H) \leq 2.8 \cdot 10^{-6} + 2.5 \cdot 10^{-2} + O(1 \cdot 10^{-14}) \approx 2.5 \cdot 10^{-2} \tag{7.3}$$

The previous approach to refine coarse results like in Eq. (7.3) was to use constraint probabilities. For example, one could come to the conclusion that only deviations of at least $4\frac{m}{s}$ are dangerous. This would then lower the probability for *error_odo* by the order

[4]Derived from the formal model of the railroad crossing, i.e. the length of 10.000m of the observed track and the average speed of the train of the train.

of 2, but would still be very imprecise. Another drawback of the previous approach is that if the system is not carefully analyzed for dependencies, the estimation can even be too optimistic. One example for this is the decision whether a deviation of the odometer of $6\frac{m}{s}$ could be justified or not.

With quantitative model-based safety analysis, the need for these coarse estimations is enormously reduced. Per-Time and per-demand failure modes can be combined directly in the extended system model. The distribution of the deviation of the odometer can also directly be specified in the extended system model, with higher deviations having a lower occurrence probability approximating a normal distribution. The limit for the accuracy is the model size and verification effort and not an a-priori specification of constraint-probabilities or manual dependency analysis.

But of course the biggest advantage of the quantitative model-based safety analysis is that all the logical or stochastic dependencies[5] which often are inherent in such a system are *automatically* considered in the quantitative analysis of the system model.

7.2. Hot Spare Backup System

7.2.1. Description

This case study consists of two redundant input sensors (S1 and S2) measuring an input signal (I). This signal is then processed in an arithmetic unit to generate the desired output signal. Two arithmetic units process the signal, a primary unit (A1) and its backup unit (A2). The primary unit gets an input signal from both input sensors, the backup unit only from one of the two sensors. If the primary unit (A1) produces no output signal, then a monitoring unit (M) switches to the backup unit (A2) for the generation of the output signal. The backup unit will only produce an output signal if it has been triggered by the monitor. If the secondary unit is triggered, the primary unit will get switched off. A schematic view of the case study is depicted in Fig. 7.14. This case study is adapted from Walker et al. and is described in [WBP07]. A first qualitative analysis has been described in [GOR08] and a first quantitative analysis in [GO10b].

This system is functionally correct if it delivers a correct output signal. If it is used in a safety-critical environment, a malfunctioning (omission of values) can be very dangerous and the system can become safety-critical.

[5] One rather surprising result is for example that the faster the train, the more likely is the situation that the sensor is passed without sensor failure. A slower train speed translates to a higher probability of a *error_passed* failure mode, as the more time passes the more the probability shrinks that the failure does not occur.

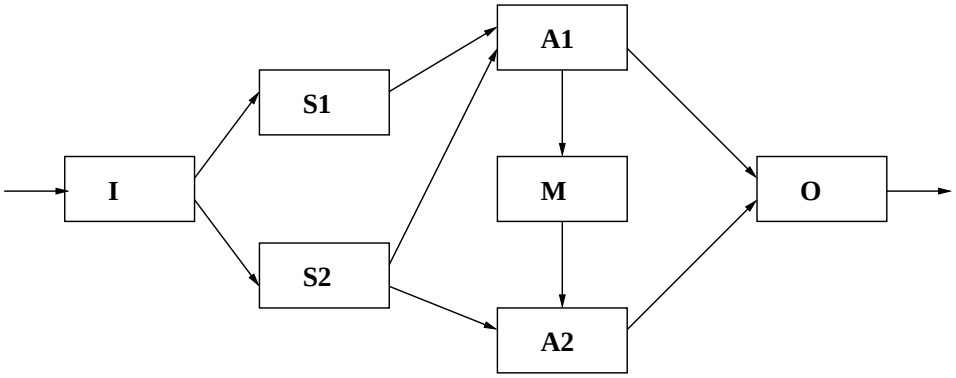


Figure 7.14.: Schematic View of Example Case Study [GOR08]

7.2.2. Modeling

Many aspects of the formal SAML model for this case study have already been presented in Chapter 4. Therefore this section describes in particular the integration of the remaining failure modes.

Modeling of Failure Modes

The failure modes that are considered in the case study and their failure rates or failure probabilities are the following: The per-time failure modes *S1FailSig* and *S2FailsSig* model an omission of the signal from either S1 or S2. Both are modeled with a failure rate of $\lambda = 1 \cdot 10^{-2} \frac{1}{h}$. Analogously, the per-time failure modes *A1FailsSig* and *A2FailsSig* describe the omission of the processed signal from either A1 or A2. The failure rate of *A2FailsSig* is $1 \cdot 10^{-2} \frac{1}{h}$, the failure rate of *A1FailsSig* is $1 \cdot 10^{-6} \frac{1}{h}$, as the primary unit A1 is of better quality than its backup unit A2. The per-time failure mode *MonitorFails* describes the situation that the monitoring unit fails to detect the signal omission from A1. Its failure rate is $\lambda_{MonitorFails} = 1 \cdot 10^{-6} \frac{1}{h}$. The per-demand failure mode *A2FailsActivate* describes the failure to activate the secondary backup unit A2 if the primary unit fails. Its occurrence probability is $1 \cdot 10^{-7}$.

S1FailsSig and S2FailsSig Fig. 7.15 shows the modeling of the S1 sensor and the failure effect of the per-time *S1FailsSig* failure mode. The initial state of the sensor S1 is the state *sig* – which is also its initial state – and produces an output signal as long as the corresponding failure module is in state *no*. When the failure mode occurs, the sensor S1 changes to state *noSig* where it does not deliver a signal any more. The occurrence pattern is modeled transiently, therefore S1 becomes operational again when the corresponding failure module reenters the state *no*. The second sensor S2 is modeled analogously, as well as the failure module for the *S2FailsSig* failure mode.

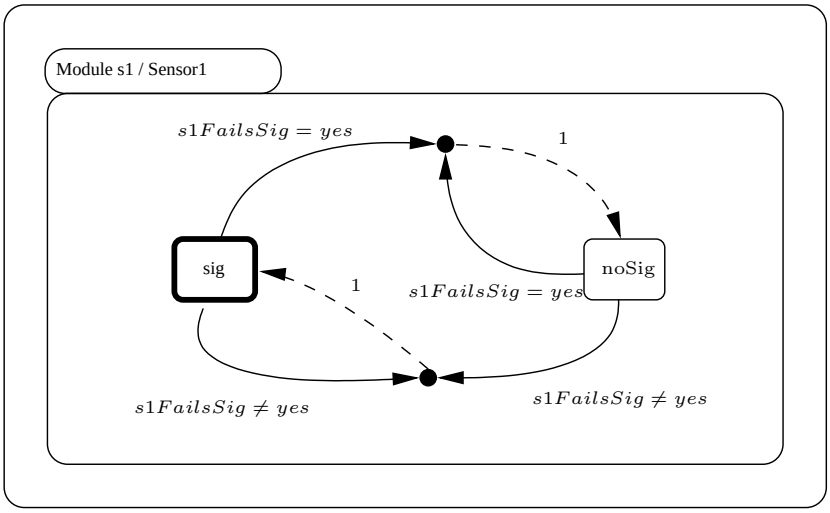


Figure 7.15.: *S1FailsSig* Failure Effect Modeling

A1FailsSig The modeling of the first arithmetic unit A1 and the failure effects of the per-time failure mode *A1FailsSig* is shown in Fig. 7.16. A1 is in the initial state *start* when the system is switched on. Afterwards it enters state *noSig* if the failure mode *A1FailsSig* occurs or the state *Sig* if no failure mode occurs. From any of these states it can be switched off by the monitoring unit. This is normally done if the monitor detects a signal omission of A1. Once A1 is switched off, it is not possible to become operational again.

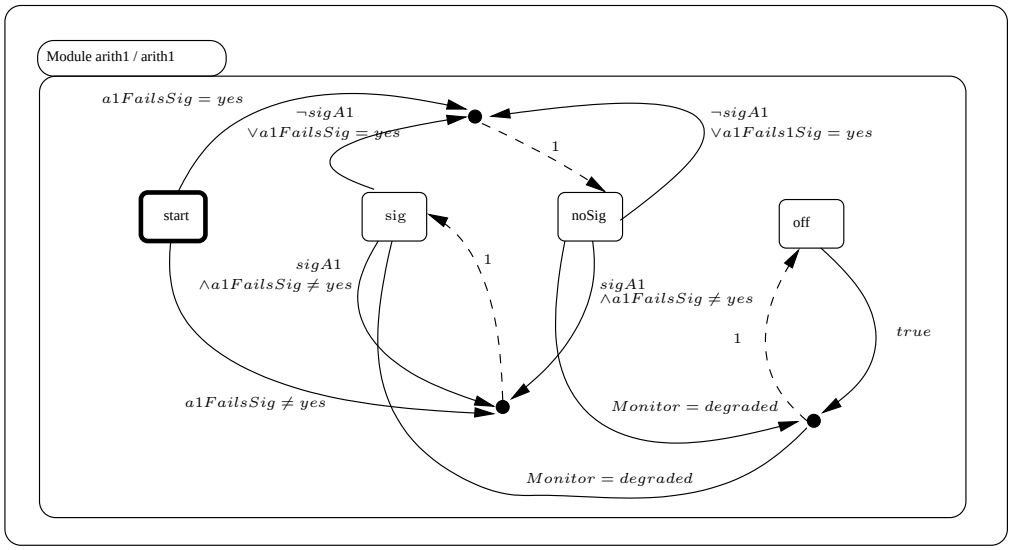


Figure 7.16.: A1 with *A1FailsSig* Failure Effect Modeling

A2FailsSig Fig. 7.17 shows the modeling of the second arithmetic unit A2 and the failure effect modeling for the per-time failure mode *A2FailsSig*, as well as the per-demand *A2FailsActivate* failure mode. The integration of *A2FailsActivate* has already e explained in Section 4.5.3.

The initial state of A2 is *idle*, as it is used as hot-spare backup for A1. If it is activated by the monitoring unit (*activate* holds) and *A2FailsActivate* does not occur, it enters either state *sig* if *A2FailsSig* does not occur or state *noSig* if *A2FailsSig* occurs. If *A2FailsActivate* occurs when the activation should take place, A2 stays in the state *idle* and never processes a signal from the sensors.

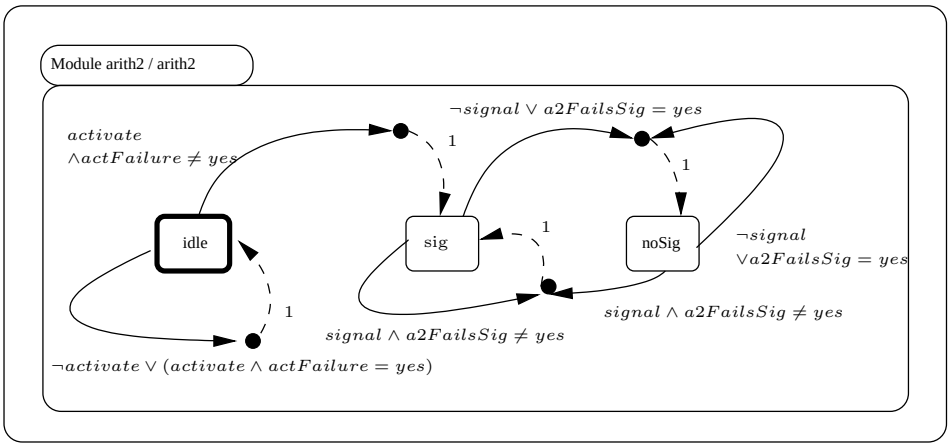


Figure 7.17.: A2 with *A2FailsActivate* and *A2FailsSig* Failure Effect Modeling

MonitorFails In Fig. 7.18 the monitoring unit and the failure effect modeling of the per-time failure mode *MonitorFails* is shown. The monitor is initially in state *normal* where it observes the primary arithmetic unit A1. If the failure mode *MonitorFails* occurs, it either stays in *normal* or enters the state *degraded* non-deterministically, independent of the state of A1. In state *degraded* it switches off A1 and tries to activate A2. If *MonitorFails* does not occur, it stays in state *normal* as long as A1 works correctly. It switches to state *degraded* and therefore activates A2 once A1 ceases to work. Once it activated A2, the monitor stays in the state *degraded*.

7.2.3. Results

aDCCA

Using DCCA to analyze the case study gives a too pessimistic view of the system, as *A1FailsSig* is considered to be a single point of failure because the degraded mode in which the A2 unit delivers the output signal is not considered (see Section 5.3.2). There-

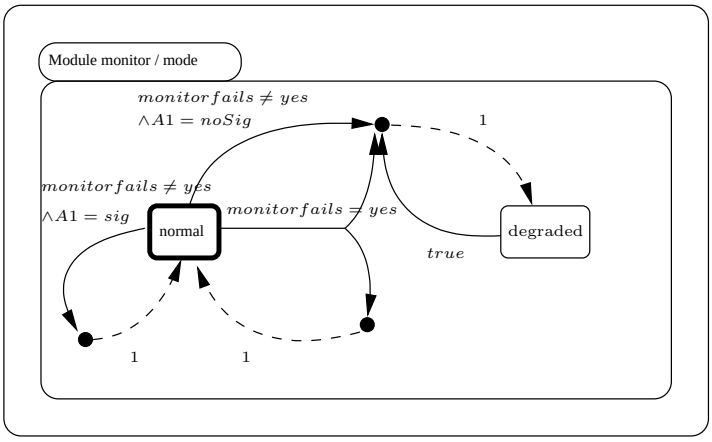


Figure 7.18.: *MonitorFails* Failure Effect Modeling

fore the case study was analyzed using aDCCA and adaptive ordering analysis which consider a failure mode combination as critical if the hazard can become permanent.

The resulting minimal adaptive-critical sets are:

- $\Gamma_1 := \{A1FailsSig, A2FailsSig\}$
- $\Gamma_2 := \{A1FailsSig, MonitorFails\}$
- $\Gamma_3 := \{A1FailsSig, A2FailsActivate\}$
- $\Gamma_4 := \{A1FailsSig, S2FailsSig\}$
- $\Gamma_5 := \{A2FailsSig, MonitorFails\}$
- $\Gamma_6 := \{MonitorFails, A2FailsActivate\}$
- $\Gamma_7 := \{MonitorFails, S2FailsSig\}$
- $\Gamma_8 := \{S1FailsSig, S2FailsSig\}$

The first set describes the situation, that both arithmetic units fail, and therefore no processed signal can be produced at all. A similar situation occurs for the set Γ_8 where both sensors fail and there is no input signal for any of the arithmetic units. The second and third sets are critical, as in both the primary arithmetic unit fails but the backup unit does not get activated. In set Γ_2 the reason is the failure of the monitoring unit, in set Γ_3 the reason is that the activation of A2 itself does not succeed. Γ_4 is critical because the combined failure of A1 and S2 mean that there is no input signal for A2 which can be processed. The fifth, sixth and seventh sets are critical because the monitoring unit wrongly switches off A1. In Γ_5 this does not work, as A2 itself fails, for Γ_6 the activation of A2 does not succeed and in Γ_7 the S2 sensor fails which is the only input for A2.

Deductive Ordering Analysis

Using these eight minimal adaptive-critical sets, adaptive deductive temporal ordering analysis proves the following adaptive temporal ordering relations:

- $MonitorFails \preceq_{\Gamma_2} A1FailsSig$
- $A1FailsSig \prec_{\Gamma_3} A2FailsActivate$
- $MonitorFails \prec_{\Gamma_6} A2FailsActivate$

If the monitoring unit fails before or at the same time as the first arithmetic unit fails, it cannot switch to A2 and therefore the hazard occurs. If the monitoring fails after A1, it can detect the failure of A1 and activate the backup unit. The other two minimal adaptive-critical sets show that *A2FailsActivate* is only critical if it occurs after either *A1FailsSig* or *MonitorFails*. The reason for this is that it is a per-demand failure mode which is triggered by the monitor, either by a detected failure of A1 or by a malfunctioning of the monitor. So for Γ_3 the critical situation arises just when the monitor detected the malfunctioning of A1, but the activation of A2 does not succeed. For Γ_6 the critical situation arises when the monitor wrongly switches off A1 and then the activation of A2 does not succeed.

The complete analysis time to compute all the adaptive minimal critical sets and the adaptive temporal ordering relations required $10.114s$ on a 8 core 2.66 Ghz Xeon computer with 16G RAM with NuSMV model-checker version 2.5.

pDCCA

As the case study is a reactive system, bounded pDCCA was conducted using the PRISM model-checker version 3.3. Using a sparse matrix representation and a mission time of $10h$ with a temporal resolution of $\Delta t = 10ms$, the full analysis required $3m$ and $36.7s$. The computed maximal probability that the systems fails to deliver an output signal within $10h$ is

$$P_{max=?}[true\mathbf{U}^{\leq 3.600.000}(observerSig = hazard)] = 9.1490 \cdot 10^{-7}$$

Advantages over existing methods If a quantitative method based on the a-posteriori analysis of the qualitative results like FTA is used, the estimation of the actual hazard probability is either very pessimistic (if the dependency of the ordering is not considered) or it can get very complex. To increase the accuracy of the analysis, the system model must be analyzed further and the dependencies of the effects of the various failure modes must be explored (the model is based on finite state machines and time passes if a state changes which can then be detected in the next time step).

The adaptive deductive ordering analysis clearly showed that the failure modes of this system are not stochastically independent. Therefore as many of the orderings of the

occurrence patterns of the failure modes as possible would have be examined to enhance the result. The probabilistic model-based analysis pDCCA does this automatically and will in general be more accurate and much less error prone than an a-posteriori analysis of the model.

7.3. Self-Adaptive Production Cell

This case study is a self-healing system from the production automation domain. In contrast to the simple adaptation to a failing component as in the case study shown in Section 7.2, a much more elaborate approach is taken here. It allows for much more autonomous behavior, while at the same time it can be formalized to allow for specification and verification of behavioral guarantees of the system.

7.3.1. Restore Invariant Approach

The idea of the restore invariant approach (RIA) was first presented in [GOR06a]. It allows for formal specification and verification of a class of autonomous systems. Its main idea is to specify a *corridor* of acceptable behavior for a system and verify the desired properties under the assumption that the system stays inside that corridor. How the system achieves staying inside the corridor is not specified directly, but with an invariant which is monitored continuously.

Using this technique, any algorithm that can restore this invariant is guaranteed to preserve the verified properties, making the approach very modular. Only the correctness of an algorithm with respect to the restoration of the invariant has to be proven, not its correctness with respect to the system itself. This is clearly an advantage, as proving correctness to the complete system would generally be more complex.

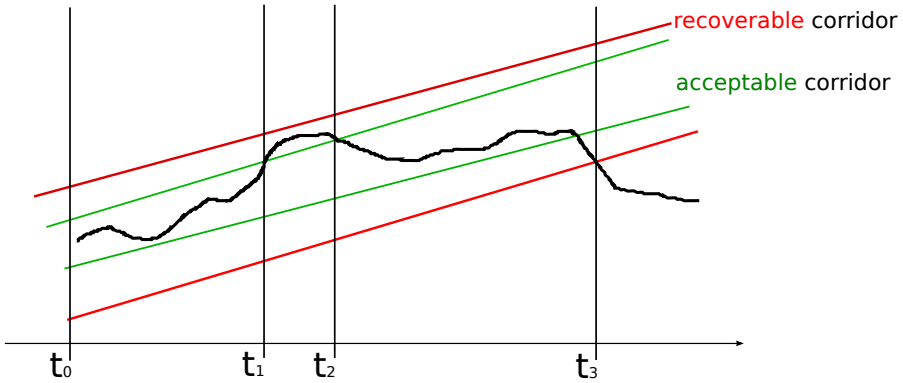


Figure 7.19.: Schematics of Restore Invariant Approach

The basic idea of RIA is illustrated in Fig. 7.19. The curved black line shows a hypothetical trajectory of a self-healing system, the colored lines mark the corridors.

This is of course not a spatial view of the system, but an abstract view of the states of the system states and its behavioral corridor.

From the start at t_0 to time t_1 the system stays in the *acceptable* corridor (green), the invariant holds and the system fulfills its desired properties. At t_1 the invariant is violated, but the system stays in its *recoverable* corridor (red) and enters its *reconfiguration* state. The adaptation algorithm restores the invariant at t_2 and the system stays in the acceptable corridor until t_3 when the system leaves its recoverable corridor and a return to acceptable operation is no more possible.

The concept of RIA modeling is rather general: The crucial point is that the adaptation mechanism must be able to decide whether an invariants holds and to restore it if not. Such a mechanism can be organized in different ways. For example it can be global or local and centralized or distributed. Which realization is adequate depends on the type of model that is considered and also on the available techniques for analysis and verification.

The formal RIA modeling presented here considers a system consisting of agents, each with a set of *capabilities*. The goal to be fulfilled requires a certain sequence of application of these capabilities. The invariant holds if all required capabilities are present and can be used in the correct order. In this situation, all the agents are configured to use one of their capabilities to reach the specific goal. If any agent loses its assigned capability, the invariant may be violated. As long as each required capability is present, the invariant can be restored by changing the assignments of the agents in order to achieve a valid configuration. This describes the *recoverable* corridor described above. Such an adaptation mechanism corresponds to the definition of *self-healing* as formalized by Seebach et al. in [SOR07]:

> "[a system is called] **self-healing** for a given set $\mathcal{C}$ of capabilities and a goal G, if after failure/loss of any capability $c \in \mathcal{C}$, then it will eventually come to [...] [an assignment of capabilities] in which G will be achieved again (as long as this is theoretically possible)."

For modeling in SAML and for formal safety analysis, the possible forms of detection of invariant violation and its restoration have to be restricted. The invariant is described as a Boolean predicate and the restoration mechanism is specified via temporal logic assertions. This will be described in detail in Section 7.3.3.

7.3.2. Description of the Case Study

The modeling approach is illustrated by applying it to a case study from production automation. It consists of three robots, which are connected with autonomous transportation units. The production cell is self-healing in case of failures of the tools of the robots. This is the reference case study of the SAVE-ORCA project within the priority research program 1183 "Organic Computing" of the German Research Foundation (DFG) and was introduced in [GOR06a] from where the description is adapted:

Every robot has three capabilities: drill a hole in a workpiece (D), insert a screw into a drilled hole (I) and tighten an inserted screw (T). These capabilities are executed with exactly one of the three different tools available for each robot. Every workpiece must be processed by all three tools in the given order (Drill, Insert, Tighten = DIT). Workpieces are transported from and to the robots by autonomous carts. Changing the tool of a robot is assumed to require some time, therefore the standard configuration of the system is to distribute the capabilities between the three robots and to have the carts transfer workpieces accordingly.

Generally, a valid configuration could be the assignment of multiple capabilities to one robot. This is not considered here for the reason described above. Therefore a configuration of the cell consists of assigning one capability to each robot and assigning autonomous transportation units the task to bring processed workpieces from one robot to the next one in correct processing order, as shown in Fig. 7.20.

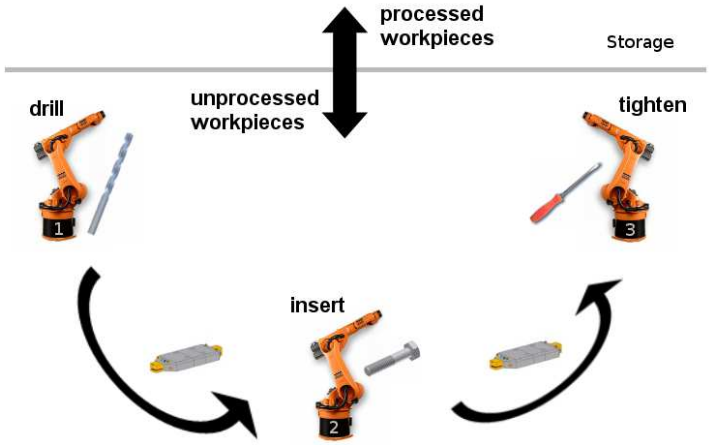
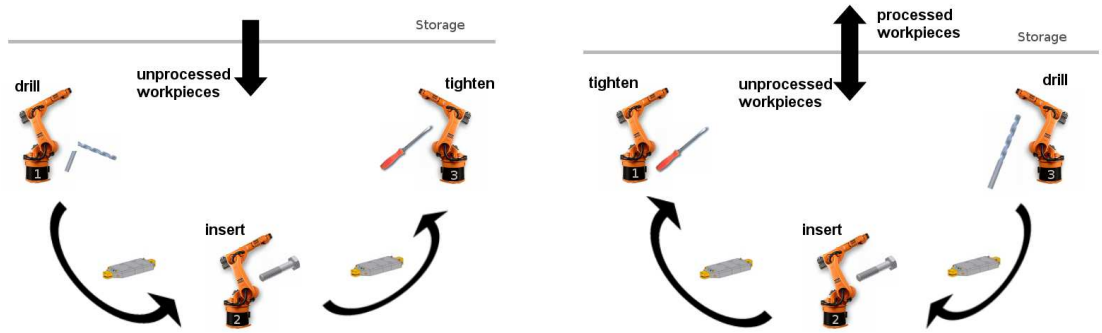


Figure 7.20.: Configuration of Robot Cell [SOR07]

If one or more tools break, the current configuration allows no more correct DIT processing of the incoming workpieces. Such a disturbance is shown in Fig. 7.21(a) in which the drill of one robot broke and DIT processing is not possible any more, as no other robot is configured to drill. A more traditional production cell would probably stop working and wait for maintenance.

However in this situation, it is obvious that the robots would still be able to achieve the overall goal, as in theory all three tools are available. The robot with the broken tool just has to switch to another tool. So it should be possible for a self-healing system to detect this situation and reconfigure itself. This works of course only if in addition to the robot with the broken tool, at least one of the other robots also switches its capability.



(a) Temporary Hazard due to Broken Drill

(b) Reconfigured Robot Cell

Figure 7.21.: Invariant Violation and Restoration [SOR07]

One possible reconfiguration outcome is shown in Fig. 7.21(b). Now the left robot drills, the right robot tightens the screws and the middle robot is left unchanged. For this error resolution, not only the assignment of the tasks to the robots must be changed, but also the routes of the carts and the routing of the incoming and outgoing workpieces. If only the tools were switched, the processing of all tasks would be possible, but not in the correct order.

This example only shows one reconfiguration for one error. Additional reconfigurations can take place if other tools break. Of course there is an end to the possibility of adaption when the necessary tools are no more available, i.e. when all drills break or when one robot has no more intact tools at all.

In summary, this self-healing system tries to preserve its functionality as long as possible even if some of the tools break. This is achieved by reconfiguration and works as long as there is enough internal redundancy in the system.

7.3.3. Modeling

The purpose of the modeling of this case study was to validate the feasibility of a formal safety-analysis of a self-healing system and to illustrate the application of the restore-invariant approach. Currently it is not possible to use the restore invariant approach in a probabilistic model, therefore the modeling presented here is purely qualitative. The exact reason for this will be explained in more detail in the last part of this section. This means that the different possible successor states are chosen in a purely non-deterministic way. The basis for the formal modeling presented here was introduced in [GOR06a], its safety analysis was introduced in [GOR06b].

Modeling of Hardware

The hardware model of the case study consists of the models for the robots and for the transportation units. The transportation units are used to fetch (partially) processed workpieces from a robot and to deliver it to the next one in correct order. The robots are responsible to process. the workpieces that are delivered by a transportation units. At any time, every robot is configured to use exactly one of its capabilities, i.e. either Drill, Insert or Tighten.

Robot Model The model of one of the robots is shown in Fig. 7.22. Initially the robot is in state `reconf` until it gets a role assigned which processing step the robot should execute, i.e. the predicate *configured* holds in conjunction with a predicate signaling which capability to use.

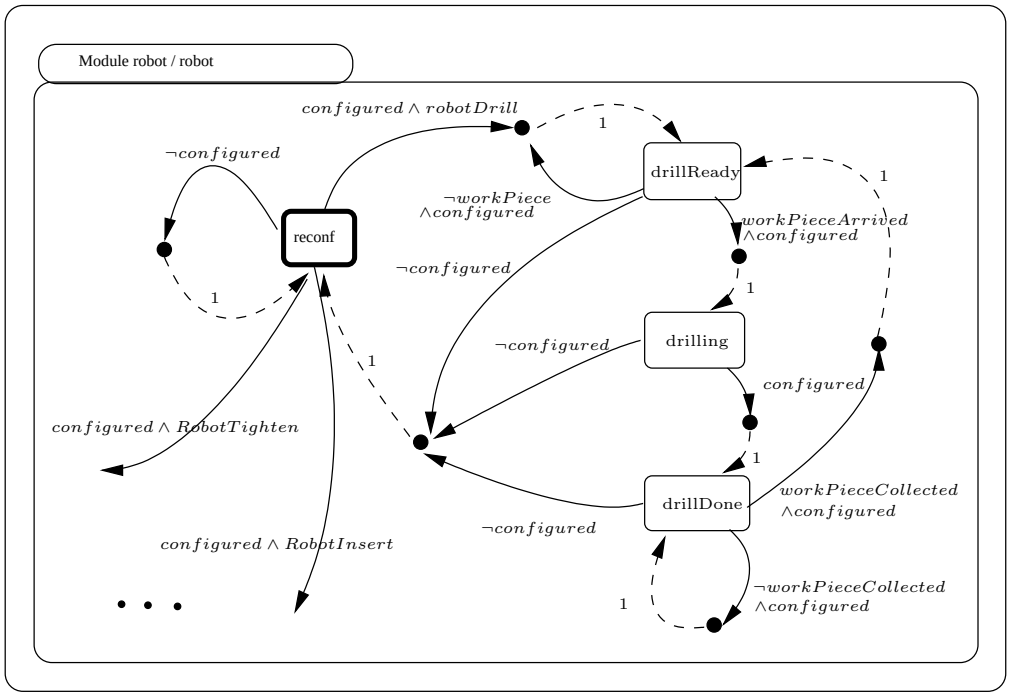


Figure 7.22.: Robot Behavior Model

Once one of the three possible capabilities is assigned to the robot, it enters the respective *ready* state. In the figure only the states for the *Drill* capability are shown, *Insert* and *Tighten* are completely analogous but are omitted to keep the graphical representation clearer. The robot waits in state *drillReady* until a transportation unit delivers a workpiece to process (*workPiceArrived* holds). When this happens, it enters state `drilling` and stays there as long as the processing takes[6]. After the processing is

[6]This can be defined by a constant for each of the capability, but is defined here uniformly as one

finished, it enters the *drillDone* state in which it waits until the processed workpiece is fetched by a transportation unit and then re-enters the *drillReady* state. Whenever the $configured$ does not hold, i.e. the production cell is not configured in a correct way, the robot enters its *reconf* state and waits for a new capability assignment.

Transportation Unit Model The model of the two transportation unit consists of three modules for each. The first one is the configuration of the transportation unit which decides from which robot the partially processed workpieces are fetched and to which the workpieces are delivered next. This defines the capability assignment of a transportation unit. The corresponding module is shown in Fig. 7.23.

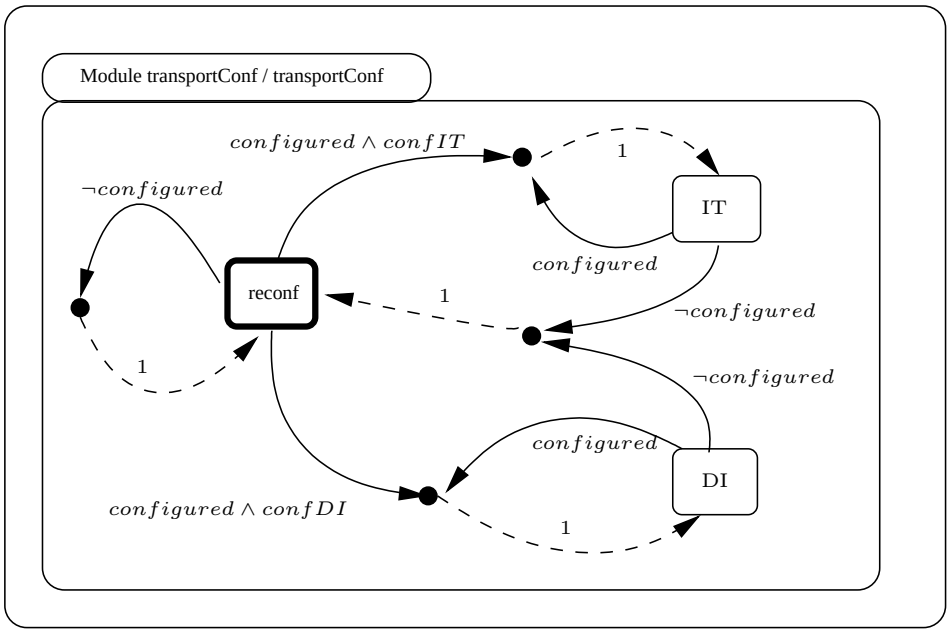


Figure 7.23.: Transportation Unit Configuration

The initial state is *reconf* in which the transportation unit waits until a new configuration is assigned (analogous to the behavior of the robots). If that happens, it enters either state *DI* indicating that it transports workpieces from the drilling to the inserting robot. Or it enters state *IT* indicating that workpieces are transported from the inserting robot to the one that tightens the screws. If a reconfiguration of the cell is triggered, it re-enters the *reconf* state until a new capability is assigned.

The second module of the model of the transportation units represents its current state. It indicates whether it is currently idle (waiting or unconfigured), whether it is currently loaded with a partially processed workpiece for delivery or if it is heading back

time-step.

to collect the next workpiece. The graphical representation of this module is shown in Fig. 7.24.

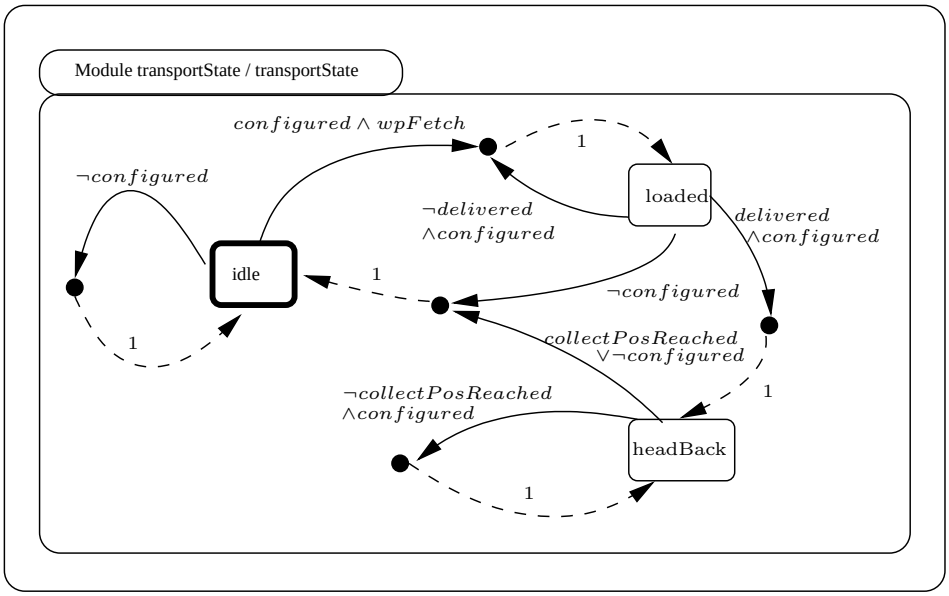


Figure 7.24.: Transportation Unit State

If the transportation unit is either unconfigured or is currently waiting for a partially processed workpiece *behind* a robot, it is in the state *idle*. It is also in this state when it waits *in front* of the robot that fetches the transported workpiece. On its way to deliver a workpiece it is in state *loaded* and on the way back to collect the next workpiece, it is in state *headBack*.

For both transportation units, there exists a third module that tracks its current position in the production cell. The behavior of that module is shown in Fig. 7.25.

The initial state is *undefined*, i.e. there is no assigned position for the transportation unit. Once a reconfiguration has assigned a valid configuration to the transportation unit, the position changes to *behind* the respective robot. If, for example, the configuration is *DI* then the transportation unit moves to *behindDrill*. It waits there until a partially processed workpiece is available. The workpiece is then collected and brought to the next processing step. On this way it is in state *loaded* and passes the position *between* two robots and finally delivers the workpiece *in front* of the robot that is next in processing order. At that position it waits until the workpiece is fetched and then returns back to the configured *behind* position. On this way it is in state *headBack*. Whenever there is a reconfiguration, the position becomes *undefined* again. The modeling is equivalent for the *IT* configuration of a transportation unit.

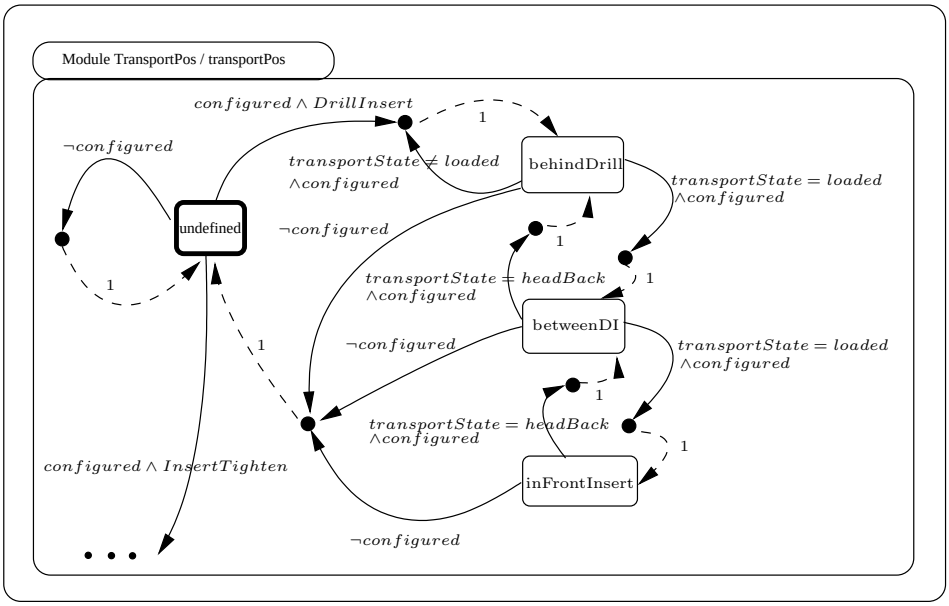


Figure 7.25.: Transportation Unit Position

Modeling of Physical Behavior

The physical behavior model in the case study contains primarily the processing of the workpieces. Each of the workpieces consists of two modules: one tracks the current position of the workpiece, the other tracks its current processing state.

Workpiece Model The position of the workpiece is tracked throughout the production cell. At the beginning it is located `outside` of the cell, in the storage. For each robot and tool, it then passes from the *in front* position to the *in* position, onto the *after* position. This can be seen in Fig. 7.26 for the Drill capability.

The workpiece stays *in front* of one of the robots until it is ready. When this happens, its position changes to *in*, meaning it is currently processed. This position is changed as soon as the robot signals the completion of the task, causing the position of the workpiece to change to *after*, where it stays until one of the transportation units collects it for delivery to the next robot. If both transportation units arrive to collect the workpiece, the production would come to a standstill. This should of course prevented by a correct capability assignment.

When the workpiece is located on one of the transportation units, it gets delivered to either the drilling or inserting robot, where its position changes analogously as explained above for the Drill capability. After the last processing step (Tighten capability), its position changes to *behindCell* (not shown in the figure), indicating the completion of the processing.

The processing state of the workpiece is described via three state variables: one to

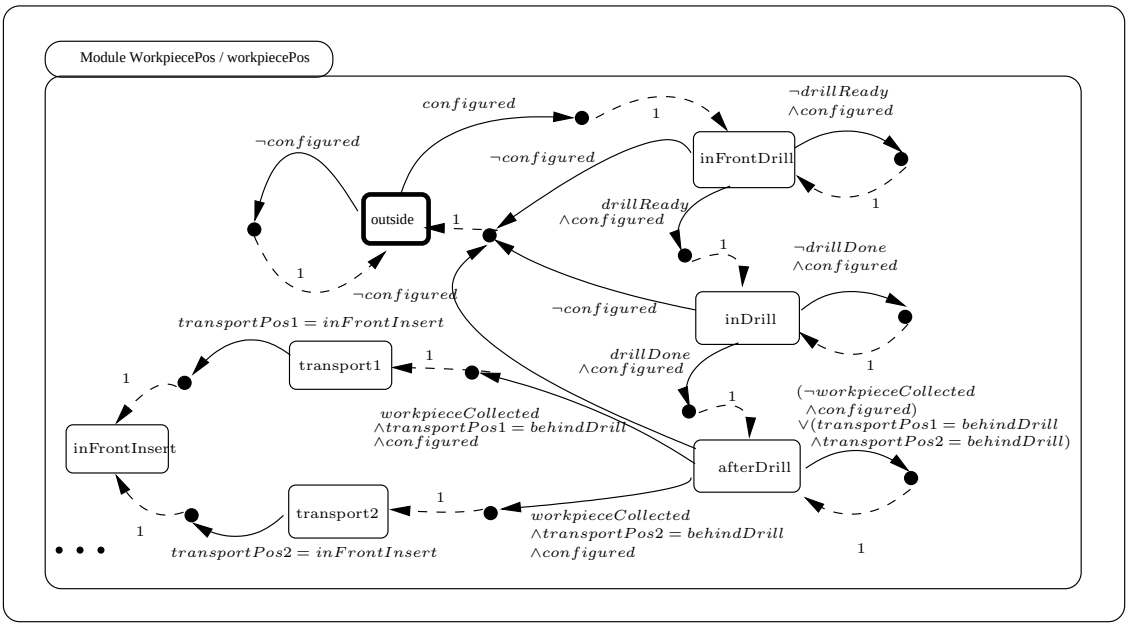


Figure 7.26.: Workpiece Position

indicate every processing step that is executed by one of the tools. In an unprocessed workpiece, all state variables are set to their initial value 0. This module is shown in Fig. 7.27 in textual SAML notation.

```
module WorkpieceState
 drilled : [0..1] init 0;
 inserted : [0..1] init 0;
 tightened : [0..1] init 0;

 workpiecePos = afterDrill -> choice : (1 : (drilled' = 1) &
                                (inserted' = inserted) & (tightened' = tightened));
 workpiecePos = afterInsert -> choice : (1 : (inserted' = 1) &
                                (drilled' = drilled) & (tightened' = tightened));
 workpiecePos = afterTighten -> choice : (1 : (tightened' = 1) &
                                (drilled' = drilled) & (inserted' = inserted));
 workpiecePos = outside ->
            choice : (1 : (drilled' = 0) & (inserted' = 0) & (tightened' = 0));
endmodule
```

Figure 7.27.: Workpiece State

Whenever the position of the workpiece is directly *after* one of the robots, the corresponding process step has been finished. This is indicated by setting respective state variable from 0 to 1. When the workpiece is completely processed and leaves the pro-

duction cell (its position becomes *behindCell*), all processing steps are deleted again and it is reintroduced into the production cell (the position becomes *outside* again). The reason for this is that there is only a finite amount of modules can be specified in a SAML model. With this modeling, the production can continue arbitrary long, as there will always eventually be unprocessed workpieces available.

Based on the described modules, the production cell can fulfill the goal to process workpieces in DIT order, if a fixed and correct initial assignment of the capabilities is specified. In such a situation, there would be a continuous processing of workpieces which after leaving the cell will be "reset" and reintroduced in the storage. The continuous and correct processing can be verified as shown in [GOR06a].

Modeling of Failure Modes

But of course from the point of safety analysis, more interesting than functional correctness of one specific configuration is its behavior under the occurrence of failures. The failure modes that are modeled in the case study are the loss of capabilities of the robots (i.e. the corresponding tool breaks). To model broken tools in the production cell, persistent failure modules are used for all tools of all robots. The considered failure modes are the following:

- *R1FailsD*: broken drill of robot 1
- *R1FailsI*: broken insertion mechanism of robot 1
- *R1FailsT*: broken tightening mechanism of robot 1
- *R2FailsD*: broken drill of robot 2
- *R2FailsI*: broken insertion mechanism of robot 2
- *R2FailsT*: broken tightening mechanism of robot 1
- *R3FailsD*: broken drill of robot 3
- *R3FailsI*: broken insertion mechanism of robot 3
- *R3FailsT*: broken tightening mechanism of robot 1

The most appropriate failure mode modeling in this case study would be a per-demand modeling, as the abstraction is that the tools are used for exactly one time-step where they can get broken. But as the model of the case study is not probabilistic, the discrimination between per-time and per-demand is not necessary here and the simpler and less complex per-time failure mode integration was chosen.

The integration of the failure modes into the extended system model works as follows: whenever the tool breaks which belongs to the currently assigned capability, then the

respective robot enters its *reconf* state and processes no more workpieces. This means that the robot locally detects the occurrence of the failure mode, but not necessarily the global invariant violation. The robot waits in that state until it gets a new capability assignment from the invariant restoration mechanism. The integration is realized by introducing new transitions from the working states of a robot to its reconfiguration state.

Modeling of Software for RIA

After the integration of the failure modes, the mechanism of detection and restoration of the invariant can be specified in the form of the restore invariant approach. The RIA modeling presented here was first introduced in [GOR06a] and uses Boolean predicates to monitor the invariant and temporal logic assertions to restore it. Its formalization is presented in detail here for reasons of clarity, necessary for safety analysis is merely the fact that it exists and works.

For a more compact expression, predicates are introduced which describe the state of the robots. The predicates R_i^a with $i \in \{1,2,3\}$ and $a \in \{d,i,t\}$ hold if robot i has been assigned to used the capability a (at most one single capability can be assigned to a robot at a time, i.e. $R_i^a \rightarrow \bigwedge_{b \in \{d,i,t\} \setminus a} \neg R_i^b$). This means for example that R_2^d holds if the module corresponding to the robot 2 is either in state *drillReady*, *drilling* or *DrillDone*.

The next predicates describe the state of the transportation units. C_j^x with $j \in \{1,2\}$ and $x \in \{it, di\}$ are modeled analogously to the robot predicates and refer to the capability assignment in the model of the carts. For example, C_1^{di} holds if the cart 1 has been assigned to fetch partially processed workpieces from robot with the Drill capability for further processing of the robot with the Insert capability (analogous to R_i^a at most one of the possible predicates can hold for each cart).

These predicates form the basis for the formalization of RIA. Using them, a configuration of the robots (i.e. each robot has an assigned capability) is defined as shown in Eq. (7.4). A configuration of the carts is defined as shown in Eq. (7.5). This means that $robotConf$ and $carConf$ hold if neither one of the robot nor one of the cart awaits a reconfiguration.

$$robotConf := (R_1^d \vee R_1^i \vee R_1^t) \wedge (R_2^d \vee R_2^i \vee R_2^t) \wedge (R_3^d \vee R_3^i \vee R_3^t) \tag{7.4}$$

$$cartConf := (C_1^{di} \vee C_1^{it}) \wedge (C_2^{di} \vee C_2^{it}) \tag{7.5}$$

This does not necessarily specify a configuration in such a way that all required capabilities are available for correct processing. This fact is described with $ditCapable$ in Eq. (7.6) for a valid robot configuration (not considering any failures) and with

cartCapable in Eq. (7.7) for a valid cart configuration.

$$ditCapable := \bigwedge_{a\in\{d,i,t\}} \left(\bigvee_{j\in\{1,2,3\}} R_j^a \wedge \left(\bigwedge_{k\in\{1,2,3\}\setminus j} \neg R_k^a \right) \right) \tag{7.6}$$

$$cartCapable := \bigwedge_{x\in\{di,it\}} \left(\bigvee_{j\in\{1,2\}} C_j^x \wedge \left(\bigwedge_{k\in\{1,2\}\setminus j} \neg C_k^x \right) \right) \tag{7.7}$$

For the definition of the invariant, it is necessary to also consider the occurrence of the various failure modes. To specify this, the predicates $fails_i^a$ with $i \in \{1,2,3\}$ and $a \in \{d,i,t\}$ are used, where $fails_i^a$ holds if the tool a of robot i is broken. For example, if the failure mode *R1FailsD* occurs and the corresponding failure module is in state *yes*, then $fails_1^d$ holds. Using these, the failure of a single robot j is signaled by $ditFailure_j$ (see Eq. (7.8)) which states that a robot has been assigned a capability that it can no longer use because its required tool is broken. Eq. (7.9) defines $ditFailure$ as the disjunction of the failures of all robots in the cell.

$$ditFailure_j := \bigvee_{a\in\{d,i,t\}} (R_j^a \wedge fails_j^a) \tag{7.8}$$

$$ditFailure := \bigvee_{i\in\{1,2,3\}} ditFailure_j \tag{7.9}$$

Therefore the invariant to monitor is the predicate $\neg ditFailure$ which holds if the system is in its *acceptable* corridor. It indicates that there is no robot which cannot use its currently assigned capability. Now, whenever the invariant restoration mechanism detects that $ditFailure$ holds, it must trigger a reconfiguration and restore the invariant.

For proving functional properties and the specification of a correct reconfiguration mechanism, it must be possible to identify whether the system is still in its *recoverable* corridor so that it is theoretically possible to restore the invariant. This is expressed with the predicate *ditPossible*. In this modeling, it is defined as the disjunction of the enumeration of all possible valid configurations. The reason for this is the fact that SAML currently does not support the usage of quantifiers[7].

The abstract invariant restoration can then be defined using the previous definitions. It consists of two temporal logic specifications: $confDIT$ Eq. (7.10) and $confCorrect$ Eq. (7.11).

$$confDIT := \mathbf{G}\left((robotConf \wedge cartConf) \rightarrow ditCapable \wedge cartCapable\right) \tag{7.10}$$

[7] A possibility for a more general modeling based on the formal RIA approach introduced here, but using the Alloy constraint solver is described by Güdemann et al. in [GNO+08]

Here $confDIT$ specifies that every reconfiguration results in a valid configuration of the production cell according to the specification. Whenever ($\mathbf{G}$) the robots and the carts are configured ($robotConf \wedge cartConf$), then all required capabilities are available, i.e. all required tools for the robots ($ditCapable$) are available and the routes for the carts ($cartCapable$) are set correctly.

$$confCorrect := \mathbf{G}\,((endReconf) \rightarrow \mathbf{X}(ditPossible \rightarrow \neg ditFailure)) \tag{7.11}$$

The $confCorrect$ property specifies that whenever a reconfiguration has just been finished ($endReconf$), then in the next time-step (i.e. the first processing step after the reconfiguration) only available capabilities have been assigned (the invariant $\neg ditFailure$ holds) as long as the system is in the recoverable corridor ($ditPossible$). This means that a reconfiguration is always successful as long as a correct reconfiguration is still possible despite potential occurrences of single failures. Note, that no proposition about the carts appears in $confCorrect$ as there is no failure mode modeled for them.

These LTL formulas are used as *assumptions* to prove properties of the system. They are assumed to be true – a correct invariant restoration mechanism exists – and other properties (in particular the aDCCA proof obligations) are proven under this assumption. This approach forms the basis of the invariant restoration mechanism. To my current knowledge the only tool that supports this directly[8] is the Cadence SMV model checker. So currently an extended system model of a self-healing system specified using RIA can only be analyzed with this verification tool. Either by direct specification in its input language or by transforming a system model into it.

Now everything required for a detection of a violation of the invariant and the restoration thereof has been defined, with the exception of a model of the actual capability assignment. This is conducted by the additional module *Control* which consists of the state variable `control`. Its graphical representation of the module is shown in Fig. 7.28.

Its initial value is *reconf*. From there it enters the state *initialize*, where it stays until all three robots and both transportation units are also in their After then, it non-deterministically enters one of the states *r1D*, *r1I* and *r1T*. Each signals which capability is assigned to the first robot. This is repeated for the second and third robot (not shown in the figure). After the assignment for the third robot, the control non-deterministically enters the states *t1DI* or *t1IT* for the assignment of the route for the first transportation unit and after then *t2DI* or *t2IT* which assigns the route to the second transportation unit. Finally it enters the state *endReconf*, signaling the end of the reconfiguration. Then it enters state *idle* and stays in this state until the predicate $ditFailure$ holds, i.e. a new reconfiguration must be done.

The *assumption* mechanism of the Cadence SMV model checker is used to assume $confDIT$ and $confCorrect$ for verification of other properties. In effect, these LTL formulas specify that whenever the *Control* module has just completed an assignment

[8] via the construct `using confDIT, confCorrect prove ...`

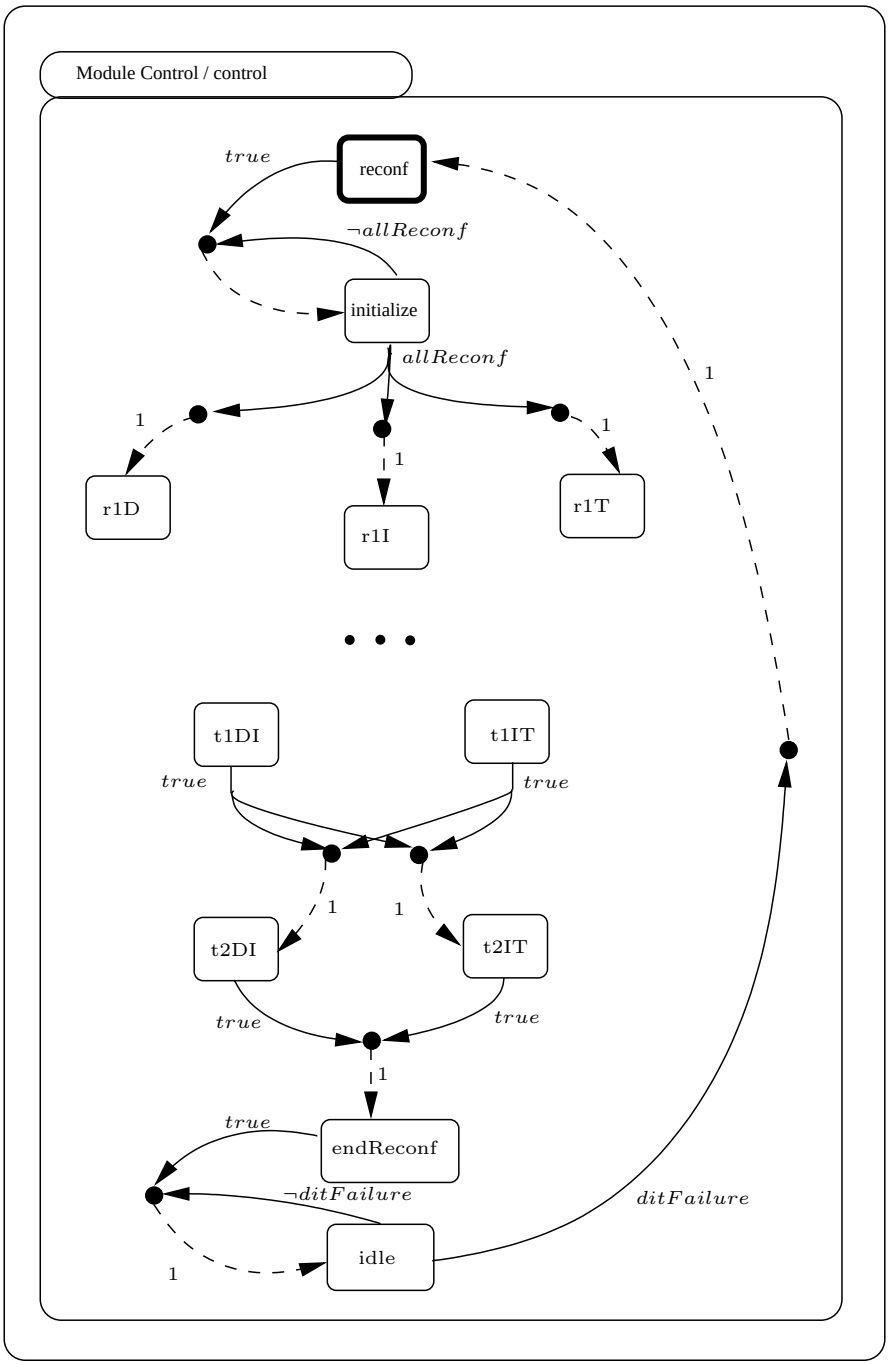


Figure 7.28.: Capability Assignment for Robots and Carts

to each robot and cart and is in its *endReconf* state, then the invariant holds in the next time-step, as long as this is still possible (see Eq. (7.11)).

This models the *restoration mechanism* of the RIA approach in the case study. When-

ever a reconfiguration is triggered, the configuration by the *control* module will be non-deterministic, but will result in a valid one. Practically this limits the verification of temporal logic properties to only those traces where valid configurations are reached.

Now it also becomes clear why this is difficult for quantitative analysis. One problem is that the currently available probabilistic model checkers do not support the *assumption* mechanism. This problem could perhaps be solved by transforming the LTL assumptions to an automaton. Nevertheless, the effect on the resulting probability space is unclear which makes the interpretation of the analysis results very difficult. In effect the RIA approach limits the traces on which the properties are verified to those where the restoration works correctly. This means that the computed probabilities would always have to be considered in relation to the probability that the restoration does not succeed. Nevertheless, currently it is not possible to compute this, as there are many different (possibly infinitely many) adversaries possible that do not restore the invariant correctly, but the verification allows only the computation of the maximizing or minimizing adversary.

A possible solution would be the specification of one explicit reconfiguration mechanism. But this would bypass RIA itself and be much less flexible. Quantitative model-based safety analysis of self-healing systems with such a degree of autonomy must therefore still be considered an open research question.

7.3.4. Results

aDCCA

The analysis to compute all adaptive minimal critical sets on a 2.6Ghz Xeon CPU with 16G RAM required 262.76s. In total 262 proof obligations had to be checked for complete aDCCA. This resulted in the following adaptive minimal critical sets:

- $\Gamma_1 := \{R1FailsD, R1FailsI, R1FailsT\}$
- $\Gamma_2 := \{R2FailsD, R2FailsI, R2FailsT\}$
- $\Gamma_3 := \{R3FailsD, R3FailsI, R3FailsT\}$
- $\Gamma_4 := \{R1FailsD, R2FailsD, R3FailsD\}$
- $\Gamma_5 := \{R1FailsI, R2FailsI, R3FailsI\}$
- $\Gamma_6 := \{R1FailsT, R2FailsT, R3FailsT\}$
- $\Gamma_7 := \{R1FailsD, R1FailsI, R2FailsD, R2FailsI\}$
- $\Gamma_8 := \{R1FailsD, R1FailsI, R3FailsD, R3FailsI\}$
- $\Gamma_9 := \{R3FailsD, R3FailsI, R2FailsD, R2FailsI\}$

- $\Gamma_{10} := \{R1FailsD, R1FailsT, R2FailsD, R2FailsT\}$
- $\Gamma_{11} := \{R1FailsD, R1FailsT, R3FailsD, R3FailsT\}$
- $\Gamma_{12} := \{R3FailsD, R3FailsT, R2FailsD, R2FailsT\}$
- $\Gamma_{13} := \{R1FailsI, R1FailsT, R2FailsI, R2FailsT\}$
- $\Gamma_{14} := \{R1FailsI, R1FailsT, R3FailsI, R3FailsT\}$
- $\Gamma_{15} := \{R3FailsI, R3FailsT, R2FailsI, R2FailsT\}$

The result of the analysis are 15 minimal adaptive-critical sets, 6 of size 3 and 9 of size 4. The minimal adaptive-critical sets Γ_1, Γ_2 and Γ_3 describe the situation that one of the robots loses all its capabilities. This makes the production impossible, as in the variant of the case study presented here the robots cannot be configured to fulfill two capabilities at the same time. Nevertheless, this could be realized as a *degraded mode* in a more complex version of the case study.

The minimal adaptive-critical sets Γ_4, Γ_5 and Γ_6 describe the situation, that one of the capabilities is completely unavailable. As every capability is required, this situation is critical for the production cell.

The remaining 9 minimal adaptive-critical sets of size 4 (Γ_7 to Γ_{15}) describe a different situation: each robot still has a capability left and overall all capabilities are available. Still it is no longer possible to configure the production cell correctly, because two of the robots have only the same single capability left and therefore the third one would have to be configured to use more than one capability at a time.

Ordering Analysis

An adaptive ordering on the adaptive-critical sets could not be found. The reason for this is that both the initial capability allocation as well as the resolving of the temporary hazard is done autonomously by the invariant restoration. This means that there is no fixed sequence of allocation of capabilities and therefore any sequence of failure mode occurrence can be problematic.

Advantages over existing methods This case study clearly shows the advantage of using aDCCA for an adaptive system. In general. it is difficult to know beforehand how long the reconfiguration needs and how many broken tools can be tolerated. Using the restore invariant approach is a convenient abstraction which allows for the verification and safety analysis of a self-healing system, independent of the concrete implementation of the reconfiguration algorithm. Any algorithm that manages to restore the invariant as long as possible, retains the properties that are verified with RIA.

7.4. Related Work

The first case study in Section 7.2 has been introduced by Walker et al. [WBP07], where it is analyzed using an algebraic approach. A comparison of the analysis method used by Walker et al. in [WBP07] and Walker and Papadopoulos in [WP07] is discussed in Section 5.4.

The autonomous production cell and the restore invariant approach were introduced in [GOR06a]. The approach has been extended in [GNO+08] to cover more general resource-flow systems modeled in the Organic Design Pattern (ODP) developed by Seebach et al. [SOR07]. In the ODP, an UML model is extended with OCL constraints which are used to describe the invariant that must be restored. A method to check whether this invariant is violated using constraint solvers like Alloy was developed by Güdemann et al. in [GNO+08] and based on KodKod by Nafz et al. in [NOS+09]. These approaches are still limited to a fixed number of robots, workpieces and autonomous transportation units. The newest extension of the RIA formalization approach described by Nafz et al. in [NSS+10] uses interactive theorem proving and modularization which allows an unlimited number of agents in such a system. The functional properties of the system are formulated in interval temporal logic (ITL) (e.g. see Cau [CMZ02]) which is formalized in KIV as described by Balser et al. [BBR08]. This is an interesting approach which significantly extends the basic RIA modeling presented in this chapter. On the other hand, this approach is not well suited for safety analysis, as this is inherently state-based quickly becomes problematic for interactive theorem proving. In addition, aDCCA (and of course pDCCA) is not directly expressible in ITL. It seems much more suited to verify functional properties of larger self-healing systems that can be modeled in the ODP framework than for safety analysis.

The radio based railway crossing was analyzed by Reif et al. using formal fault tree analysis and interactive theorem proving in [RST00] and Thums and Schellhorn in [TS02]. The first use of failure injection, model-based safety analysis and model-checking for the analysis of the case study was presented by Ortmeier et al. in [ORS05]. Both these analyses where purely qualitative and did not allow for the computation of hazard probabilities. For the estimation of hazard probabilities, the fault-tree formula was used which is very coarse and assumes stochastic independence. A distinction between per-time and per-demand failure modes is not possible in any of these approaches. Ortmeier et al. introduced safety optimization in [OSR04]. In this approach an additional safety margin and the allowed speed for the train were used as parameters of an additional model of stochastic behavior based on distribution functions. This model consisted of the opposing goal to minimize the time-delay at the crossing and to minimize the hazard probability. These goals are weighted according to their projected costs and the resulting scalar function can be optimized using mathematical optimization techniques. A more detailed description can be found in Ortmeier's dissertation [Ort06]. This basic approach can be extended using the quantitative model-based analysis pDCCA as objective function for optimization as described in [GO11a] which gives more

accurate results than the a-posteriori estimations for the hazard probabilities used before. Using multi-objective optimization also allows for computing non-convex Pareto sets (optimal compromises). A serious limitation of the previous scalarization approach was that only convex Pareto sets could be found.

Summary

This chapter presented modeling and model-based safety analysis of three different case studies. The case studies were selected to show how different types of systems can be modeled and how different concrete analyses work. It also showed the current limits of the approach. Quantitative analysis of a more autonomous case study is currently not possible. On the other hand, the safety analyses of each case study were beyond the current state of the art and more accurate than previously possible. The major achievement is the successful quantitative model-based safety analysis of the larger case study of the railroad crossing. It clearly shows that the full approach – in particular the resource-intensive probabilistic analysis – is definitely applicable to real-world systems.

8. Conclusion And Outlook

The true logic of this world is in the calculus of probabilities.

(James Clerk Maxwell)

This dissertation introduced new formal, model-based qualitative and quantitative safety analysis methods which advance the current state of the art. The foundation for these is the new formal modeling framework SAML which allows convenient specification of formal models of safety-critical systems for accurate qualitative and quantitative formal safety analysis. Modeling guidelines showed how accurate models of different failure modes and realistic physical behavior models can be expressed in SAML. Semantically well-founded model transformations allow for using the most appropriate and efficient verification tools. Several case studies showed the benefit and efficiency of the approach and illustrated the application of formal modeling and safety analysis.

8.1. Summary and Conclusion

8.1.1. Summary

SAML

The developed SAML modeling framework allows for convenient expression of formal models for safety analysis. With SAML, it is possible to combine non-deterministic, qualitative and probabilistic, quantitative behavior in a single system model. This allows for accurate specification of different kinds of failure modes and of the behavior of a system's physical environment. SAML is designed as a tool-independent language. Semantically well-founded model transformations convert SAML models fora specific analyses with appropriate verification tools . The model transformations are proven to be semantically sound and preserve the model semantics. This tool-independent approach makes any advancement of the verification tools directly usable for safety analysis.

Note that one example of this effect is the support for stochastic hypothesis testing which allows computation of confidence intervals even in the case of models that are too large for full model checking [You05]. This feature is introduced in version 4.0 of the PRISM model checker, but was not available in version 3.3 that was available when pDCCA was originally developed. Now stochastic hypothesis testing can be used for the analysis of SAML models without any change to the model transformations.

Modeling and Safety Analysis Methods

Different new analysis techniques have been developed for accurate formal safety analysis of SAML models. These extend the analyzable class of systems, augment the qualitative safety analysis results and allow for a combined qualitative and quantitative, model-based safety analysis of extended SAML system models.

The basis for accurate safety analysis are the modeling methods for different types of failure modes. They allow for a sound combination of per-time and per-demand failure mode modeling in the same model which was not possible in previous approaches. Probabilistic modeling also allows for a more accurate modeling of the behavior of the physical environment of a system resulting in more precise safety analysis results.

Critical combinations of failure modes, e.g. computed with the previous analysis method DCCA, can be further analyzed using deductive ordering analysis. It allows for a more precise *qualitative* safety analysis than before. Its results can be used to augment the effect of the implementation of risk-reducing measures by identifying the failure modes where a reduction of risk is most beneficial. Using deductive ordering analysis also allows for an automatic deduction of several dynamic fault tree gates directly from a system model.

Probabilistic DCCA enhances the previous quantitative safety analysis results which were based on a-posteriori analysis of critical failure mode combinations. Quantitative safety analysis with pDCCA allows for accurate computation of hazard occurrence

probabilities directly on the system model. It does not rely on the assumption of stochastic independence and allows for a combined analysis of per-time and per-demand failure modes in a single model. This was not considered in any existing failure-injection, model-based safety analysis approach. It is much more accurate than previous a-posteriori estimations, in particular by automatically considering all system inherent logical and stochastic dependencies which were very hard or even impossible to consider in previous approaches.

Adaptive DCCA and adaptive deductive ordering analysis are an extension of the qualitative safety analysis methods to self-healing systems with a certain degree of autonomy. The restore invariant approach was developed as a very general way of formally specifying self-healing systems. It allows a very flexible way to define self-healing behavior. Based on RIA, qualitative properties of self-healing systems can be verified with aDCCA, which was not so generally possible before. Quantitative safety analysis of self-healing systems is possible if a modeling approach based on an observer is used instead which seems to be possible for a large class of systems.

The applicability of the presented modeling and safety analysis techniques to practical examples has been demonstrated with several case studies with sizes of more than 10^7 states. It has been shown that these methods allow a much more accurate assessment of the safety of a system than was possible before. One key aspect is that both qualitative and quantitative analyses are conducted on an equivalent model. This raises the confidence in the computed results, as it is guaranteed that they are "compatible" to each other. The effort needed for modeling and analysis has also proven to be acceptable. Due to the tool-independence, the analysis effort will be reduced further by the continuing advancement and increasing efficiency of verification tools.

8.1.2. Conclusion

In conclusion, the greatest benefit of the results of this dissertation is the possibility of conducting probabilistic safety analysis directly on extended system models while providing means of combining per-time and per-demand failure modes in the analysis. This was not possible before and marks an important step forward in the state of the art. The second main benefit is bridging the gap between qualitative and quantitative safety analysis achieved by the definition of SAML and the semantically well-founded transformations of SAML models into the input specification of different verification tools. This increases confidence in the results of the safety analyses by assuring the compatibility of these different views on the properties of a safety-critical system.

These techniques could be a great help for the safety engineer in assessing the safety of a system in the early design phases. Different system variants can be compared, critical failure combinations can be identified and risk-reducing measures can be applied. The effects can be analyzed by inspecting the computed counterexamples which serve as "witnesses" for properties. They can provide very useful insights into the functioning of the system, as Edmund M. Clarke puts it in the *Birth of Model-Checking* [Cla08]:

> *"It is impossible to overestimate the importance of the counterexample feature."*

New developments even allow the use of probabilistic analysis to compute the most probable counterexamples. Approaches for this are described by Han et al. [HKB09] and Aljazzar et al. [AKLFL10]. This means that risk-reducing measures can be applied where their effect will be most significant. Due to the tool-independence of SAML, the described safety analysis approach benefits directly from such new developments without additional effort.

Nevertheless, it is important to note that the results of the safety analyses should be interpreted by an expert in the field. Formal model-based safety analysis is primarily an extension of existing safety analysis. The subtitle of this dissertation – *"Push the safety button"* – is meant in the following way: using the safety analysis methods presented in this dissertation, it is possible to support the safety engineer by allowing safety analysis in early design phases of the development process and facilitating accurate comparison of different design variants. In the future, this could be realized as an additional function of a design tool used by a safety engineer.

A complete automated safety analysis, where the results are used without further interpretation is not favored and was not the goal of this work. The developed methods are powerful and objective tools for safety analysis – but like any other tool they must be applied correctly (in this case by a safety engineer). Galloway et al. [GMMP02] and Lisagor et al. [LSK10] argue that, in particular, producing traditional looking analysis artifacts like fault trees or FMEA tables with novel, automatic techniques can lead to problems if not correctly interpreted. This risk also exists with quantitative analysis results. As noted by Alexander and Kelly [AK09], these are easily misinterpreted and sometimes even misused by stakeholders not directly involved in the safety analysis process itself.

Nevertheless, the developed automatic analysis techniques can provide highly valuable support in the development process of safety-critical systems, both by pointing out potential problems and also by calculating probabilities for the comparison of different system design variants.

8.2. Outlook

Based on the results of this dissertation, several new possibilities for further research were identified. A first extension to the presented approach would be to extend SAML with an explicit failure model. This would allow automatic combination of the nominal model and the failure model to form the extended system model. This step is currently conducted manually and a change in the nominal behavior may therefore require a lot of effort if the integration of the failure mode behavior has to be redone. Such an approach has been taken by Bozzano et al. in their work on SLIM [BCK+10b]. However, the

possibility to consider both per-time and per-demand failure modes is a major advantage to the approach based on SLIM. Nevertheless it also makes automatic failure injection much harder, as correct integration of per-demand failure effects is more complicated. Such an extension would be a clear advance in the convenience of modeling safety-critical systems in SAML and is currently ongoing work.

An orthogonal extension would be to use SAML as an intermediate language. Models from different higher-level specification languages could then be transformed into SAML and further be analyzed with all supported verification tools. An outline of this approach is shown in Fig. 8.1. In this figure, the approach described in this dissertation is shown in the *verification* and *intermediate* layers (see also Fig. 6.1). On top of this, an *engineering* layer is added in which models in existing specification frameworks, such as SCADE or Matlab / Simulink, are extended to include failure mode models and probabilistic modeling. Such models could then be transformed into SAML in a semantically well-founded way. Once this has been achieved, the existing model transformations from SAML into the verification layer can be used for formal analysis. An additional challenge in such an approach would, of course, be the back-transformation of the analysis results into the engineering layer. If successful, this approach would allow the inclusion of qualitative and quantitative model-based safety analysis directly into widely used development frameworks. A first overview of the planned approach is described in [GO11b].

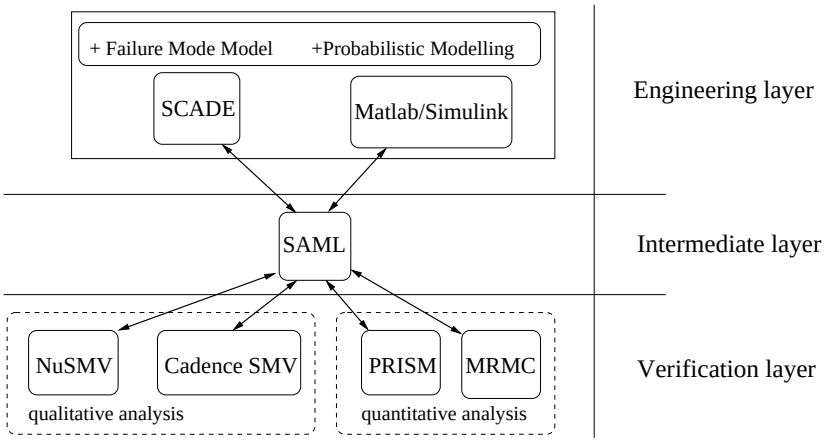


Figure 8.1.: Outline of Model-Driven Safety Analysis [GO11b]

A completely different research topic is based on exploiting quantitative analysis results to optimize systems. Although many safety-critical systems are developed according to the *as low as reasonably possible* (ALARP) principle, the definition of a *reasonable* low risk is unclear. It depends not only on hazard probabilities, but also on other goals, which are very often even antagonistic to safety. The most obvious examples are, of course, economic factors, such as direct cost of a system or the costs of using a system in a safer way, e.g. low permissible speed for a train. For the implementation of a

concrete system, it will therefore very often be an important challenge to find a good compromise between acceptable safety and the antagonistic goals. In order to find such compromises, quantitative safety analysis can be used as an objective function for multi-objective optimization of system designs and very promising first results have already been achieved [GO11a].

Use of quantitative model-based safety analysis results based on pDCCA and SAML forms the basis for the DFG-funded Probabilistic Models in Safety Analysis (ProMoSA) project [OG10], which aims both at integrating qualitative and quantitative model-based safety analysis into industrial engineering tools and at the automatic optimization of safety-critical systems under different antagonistic goal functions.

A. Proofs

Lemma 1. Embedded Kripke Structure

Let $\tau_{Kripke} = \kappa(\tau_{MDP})$ be the tuple $\tau_{Kripke} = (S, \{s_0\}, T, L, AP)$ as defined in Def. 17, then τ_{Kripke} is a Kripke structure.

Proof. By definition the set of states, initial state, labeling and atomic propositions is the same in τ_{MDP} as in $\tau_{Kripke} = \kappa(\tau_{MDP})$.

In order for τ_{Kripke} to be a proper Kripke structure, the transition relation T must be left-total, i.e. $\forall s \in S : \exists t \in S : (s,t) \in T$. From the definition of T and the definition of a MDP (Def. 3) we know that for each state s, $\mathbf{Steps}(s)$ is a set of indices and corresponding discrete probability distribution of the form (j,p). For each such p we know that $\sum_{t \in S} p(s,t) = 1$. This means that for each state s there exists a state t such that $p(s,t) > 0$ and (s,t) is in T (by definition). So for each reachable state s there exists a successor state t and therefore the transition relation is total. □

Lemma 2. Embedded Kripke Structure Path Equivalence

Let ρ be the projection of a path of the MDP of the form $\omega = s_0(j_0, p_0)s_1 \ldots$ to a state sequence π of the form $\pi = s_0 s_1 \ldots$.

Then the diagram in Fig. A.1 is commutative, i.e. $Paths(\kappa(\tau_{MDP})) = \rho(Paths(\tau_{MDP}))$.

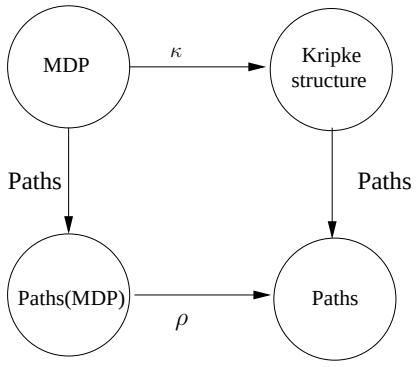


Figure A.1.: Mapping of MDP Traces to Kripke Structure Traces

Proof. To show:

$$\forall \pi : (\pi \in Paths(\kappa(\tau_{MDP})) \Leftrightarrow \exists \omega \in Paths(\tau_{MDP}) : \rho(\omega) = \pi)$$

"$\Rightarrow$":

Let π be an arbitrary element of $Paths(\kappa(\tau_{MDP}))$. As κ maps a MDP τ_{MDP} to a Kripke structure τ_{Kripke} with the same states, for each pair of states (s,t) in the transition relation T of τ_{Kripke} there exists an index, probability distribution pair (j,p) in τ_{MDP} such that $p(s,t) > 0$ holds (see Lemma (1)). Therefore for $\pi = s_0, s_1 \ldots$, for every pair $(s_i, s_{i+1}) \in T$ there exists a pair (j,p) such that $p(s_i, s_{i+1}) > 0$ in the MDP and $\omega = s_0(j_0, p_0(s_0, s_1))s_1 \ldots$ is a path of τ_{MDP} (Def. (4)) and by its definition, ρ projects ω onto π (as the sequence of states is preserved)

"$\Leftarrow$":

By contradiction: Assume $\exists \omega \in Paths(\tau_{MDP}) : \rho(\omega) = \pi \wedge \pi \notin Paths(\kappa(\tau_{MDP}))$. This means that for $\omega = s_0(j_0, p_0)s_1 \ldots s_i(j_i, p_i)s_{i+1} \ldots$ the probability $p_i(s_i, s_{i+1}) = 0$ must hold (else π would be an element of $Paths(\kappa(\tau_{MDP}))$, see Lemma (1)). But if such a pair (j_i, p_i) exists, then ω is not a path of a MDP (see Def. (4)), as the probability to reach the next state must always be non-zero. □

Corollary 1. Embedded Kripke Structure Trace Equivalence

A MDP τ_{MDP} and its embedded Kripke structure $\tau_{Kripke} = \kappa(\tau_{MDP})$ are isomorphic wrt. their set of traces.

Proof. From Lemma 2, for each Path $\pi = s'_0 s'_1 \ldots$ of the embedded Kripke structure $\kappa(\tau_{MDP})$ there exists a Path $\omega = s_0(j_0, p_0)s_1 \ldots$ of the MDP τ_{MDP} such that the projection of ω is equal to π. The same holds vice versa.

From the construction of the embedded Kripke structure (see Def. 17) the labeling function for each state $L(s_0)$ is equal to $L(s'_0)$, as the labeling function is the same. □

Lemma 3. Conservative Integration

Let M be a SAML model with the set of state variables Var and M' be the SAML model in which the effect of a failure mode γ_i is integrated into M adhering to the rules for conservative integration. Let F_i be the corresponding failure module, $fails_i$ the state variable that indicates that the failure mode is absent ($fails_i \leq 0$) or present ($fails_i > 0$). Let τ_{MDP} be the MDP corresponding to M and τ'_{MDP} be the MDP corresponding to $M'||F_i$. Then:

$$\forall \pi \in Paths(\kappa(\tau_{MDP})) : \exists \pi' \in Paths(\kappa(\tau'_{MDP})) : \pi \equiv \pi'|_{Var} \tag{A.1}$$

where $\pi'|_{Var}$ is the projection of π' onto the state variables $v \in Var$

Proof. From the construction rules for failure effect modeling, it is assured that for all traces for all states in which the failure mode is absent (the failure module is not in state "yes"), there exists a successor state with non-zero probability on which the failure mode is also absent[1].

[1]This means that the occurrence pattern is modeled in a sensible way, so that the failure mode may not appear at all and therefore the occurrence of the failure mode is not forced.

Therefore assume that

$$\kappa(\tau'_{MDP}), s_0 \models \mathbf{A}\,\mathbf{G}\,(fails_i \leq 0 \rightarrow \mathbf{E}\,\mathbf{X}(fails_i \leq 0)) \tag{A.2}$$

holds. This means that for all states on all traces if the failure mode is absent, then there is also a successor state in which the failure mode is absent.

Then for each $\pi \in Paths(\kappa(\tau_{MDP}))$ a corresponding $\pi' \in Paths(\kappa(\tau'_{MDP}))$ can be constructed in the following way:

Let s_k be the $k-th$ state of a path $\pi = s_0, s_1 \ldots$ of $\kappa(\tau_{MDP})$. In the complete SAML model, the activation condition that held in s_k is of the form $\bigwedge_{j=1}^{m} \phi_j$ (the conjunction of all active activation conditions of all parallel SAML modules).

Now let ϕ_t be the activation condition of the SAML module that was changed in M' to integrate the failure effect. Assume there exists a path π' of $\kappa(\tau'_{MDP})$ on which for the $k-th$ state s'_k $fails_i \leq 0$ holds and $s_k \equiv s'_k|_{Var}$, i.e. ϕ_t also holds in s'_k.

Let s_{k+1} be a successor state of s_k on π. As $fails_i \leq 0$ holds in s'_k, any possible successor state s'_{k+1} on π' cannot be a state introduced by the failure mode effects modeling (rule 3b). As $\phi_t \wedge fails_i \leq 0$ holds in s'_k there also exists a successor state s'_{k+1} such that $fails_i \leq 0$ (because of (A.2)) and for which $s_{k+1} \equiv s'_{k+1}|_{Var}$ holds. The reason is that all active update rules of the complete model of the original model M have a corresponding active update rule in M' if $fails_i \leq 0$ holds (and by construction one which is active if $fails_i > 0$ holds).

So for each path π of $\kappa(\tau_{MDP})$ for each state s_k, if there exists an equivalent state s'_k on a path π' of $\kappa(\tau'_{MDP})$ on which $fails_i \leq 0$ holds, there exists an equivalent continuation of π and π' such that $\pi \equiv \pi'|_{Var}$.

As the initial state does not change (rule 4), $s_0 \equiv s'_0|_{Var}$ and $fails_i \leq 0 \in L(s'_0)$ holds (the absence of the failure mode is the initial state of the failure module in the described failure mode modeling) and the assertion follows by induction. □

Lemma 4. Deductive Before Ordering / Strict Before Ordering

For an extended system model M^+, a set of failure modes Δ, a predicate logic hazard H,a minimal critical set Γ and two failure modes $\gamma_1, \gamma_2 \in \Gamma$,

$$\begin{aligned} &\gamma_1 \preceq_\Gamma \gamma_2 \Leftrightarrow \\ &\kappa(M^+), \pi \models \quad (\overline{\Gamma}\mathbf{U}H) \rightarrow [(\neg H \wedge \neg\gamma_1 \wedge \neg\gamma_2) \\ &\qquad\qquad \mathbf{U}(\gamma_1 \wedge ((\neg H \wedge \neg\gamma_2) \\ &\qquad\qquad\ \mathbf{U}((\gamma_2 \wedge \neg H) \wedge \mathbf{F}\,H)))] \end{aligned} \tag{A.3}$$

$$
\begin{aligned}
&\gamma_1 \prec_\Gamma \gamma_2 \Leftrightarrow \\
&\kappa(M^+), \pi \models \quad (\overline{\Gamma}\mathbf{U}H) \rightarrow [(\neg H \wedge \neg\gamma_1 \wedge \neg\gamma_2) \\
&\qquad\qquad \mathbf{U}((\gamma_1 \wedge \neg\gamma_2 \wedge \neg H) \wedge \\
&\qquad\qquad\quad \mathbf{X}((\neg H \wedge \neg\gamma_2) \\
&\qquad\qquad\qquad \mathbf{U}((\gamma_2 \wedge \neg H) \wedge \mathbf{F}\, H)))] \qquad \text{(A.4)}
\end{aligned}
$$

Proof. Before Ordering:

At first the expression of equation (A.3) is expanded:

$$
\begin{aligned}
&\kappa(M^+), \pi \models (\overline{\Gamma}\mathbf{U}H) \rightarrow [(\neg H \wedge \neg\gamma_1 \wedge \neg\gamma_2) \\
&\qquad \mathbf{U}(\gamma_1 \wedge ((\neg H \wedge \neg\gamma_2) \\
&\qquad\quad \mathbf{U}((\gamma_2 \wedge \neg H) \wedge \mathbf{F}\, H))] \\
\Leftrightarrow\ & \neg(\overline{\Gamma}\mathbf{U}H) \vee \qquad \text{(A.5)} \\
&\exists k \geq 0 : \kappa(M^+), s_k \models \gamma_1 \wedge \qquad \text{(A.6)} \\
&\qquad (\exists m \geq k : \kappa(M^+), s_m \models (\neg H \wedge \gamma_2) \wedge \qquad \text{(A.7)} \\
&\qquad\quad (\exists l \geq m : \kappa(M^+), s_l \models H) \wedge \qquad \text{(A.8)} \\
&\qquad\quad \forall j : k \leq j < m : \kappa(M^+), s_j \models \neg H \wedge \neg\gamma_2) \wedge \\
&\qquad \forall i : 0 \leq i < k : \kappa(M^+), s_i \models \neg H \wedge \neg\gamma_1 \wedge \neg\gamma_2 \qquad \text{(A.9)}
\end{aligned}
$$

The equation (A.5) is used to select the traces of the system where Γ actually was the cause for the hazard. If it is true, then the failures did not appear before the hazard and the failures were not the cause. Therefore we are only interested in the cases where (A.5) is false and the rest is true.

If this is true, then there exist 3 special time steps on the path π. Let k_0 be the smallest k at which γ_1 is true and m_0 the smallest m at which γ_2 is true, from equation (A.7) and (A.6) we know that these exist and that $m_0 \geq k_0$.

From (A.8) we know that there is a smallest l_0 at which H is true and that $l_0 \geq m_0$. As (A.7) implies that H is false for m_0 we have the following temporal relation for the mentioned time steps: $l_0 > m_0 \geq k_0$. This implies that both failures appear before the hazard and that γ_1 does not appear after γ_2, therefore $\gamma_1 \preceq_\Gamma \gamma_2$. □

Strict Before Ordering:

$$
\begin{aligned}
& \kappa(M^+), \pi \models (\overline{\Gamma}\mathbf{U}H) \rightarrow [(\neg H \wedge \neg\gamma_1 \wedge \neg\gamma_2) \\
& \quad \mathbf{U}((\gamma_1 \wedge \neg\gamma_2 \wedge \neg H) \wedge \\
& \quad\quad \mathbf{X}((\neg H \wedge \neg\gamma_2) \\
& \quad\quad\quad \mathbf{U}((\gamma_2 \wedge \neg H) \wedge \mathbf{F}\, H)))] \\
\Leftrightarrow \quad & \neg(\overline{\Gamma}\mathbf{U}H) \vee && \text{(A.10)} \\
& \exists k \geq 0 : (\kappa(M^+), s_k \models \gamma_1 \wedge \neg\gamma_2 \wedge \neg H) \wedge \\
& \quad (\exists m \geq k+1 : \kappa(M^+), s_m \models (\gamma_2 \wedge \neg H)) \wedge && \text{(A.11)} \\
& \quad\quad (\exists l \geq m : \kappa(M^+), s_m \models H) \wedge && \text{(A.12)} \\
& \quad\quad\quad (\forall j : k \leq j < m : \kappa(M^+), s_j \models \neg\gamma_2 \wedge \neg H) \wedge \\
& \quad\quad (\forall i : 0 \leq i < k : \kappa(M^+), s_i \models \neg\gamma_1 \wedge \neg\gamma_2 \wedge \neg H) && \text{(A.13)}
\end{aligned}
$$

Assume (A.10) to be false and the rest to be true for the reason mentioned before.

Let k_0 be the smallest k where $\gamma_1 \wedge \neg\gamma_2 \wedge \neg H$ is true, let m_0 be the smallest m where $\gamma_2 \wedge \neg H$ is true and l_0 the smallest l where H is true. From (A.11) we know that $m_0 \geq k_0 + 1$, i.e. $m_0 > k_0$ and from (A.12) and (A.11) we know that $l_0 > m_0$. Therefore we know that if H appears on a trace, both failures appear before the hazard and γ_1 appears before γ_2, i.e. $\gamma_1 \prec_\Gamma \gamma_2$. □

Lemma 5. Deductive Simultaneous Order

For an extended system model M^+, a set of failure modes Δ, a predicate logic hazard H,a minimal critical set Γ and two failure modes $\gamma_1, \gamma_2 \in \Gamma$

$$
\begin{aligned}
& \gamma_1 \sim_\Gamma \gamma_2 \Leftrightarrow \\
& \kappa(M^+), \pi \models \quad (\overline{\Gamma}\mathbf{U}H) \rightarrow [(\neg H \wedge \neg\gamma_1 \wedge \neg\gamma_2) \\
& \qquad\qquad\qquad \mathbf{U}((\gamma_1 \wedge \gamma_2 \wedge \neg H) \wedge \mathbf{F}\, H)] && \text{(A.14)}
\end{aligned}
$$

Proof. At first the expression of equation (A.14) is expanded:

$$
\begin{aligned}
& \kappa(M^+), \pi \models (\overline{\Gamma}\mathbf{U}H) \rightarrow [(\neg H \wedge \neg\gamma_1 \wedge \neg\gamma_2) \\
& \quad \mathbf{U}((\gamma_1 \wedge \gamma_2 \wedge \neg H) \wedge \mathbf{F}\, H))] \\
\Leftrightarrow \quad & \neg(\overline{\Gamma}\mathbf{U}H) \vee && \text{(A.15)} \\
& \exists k \geq 0 : \kappa(M^+), s_k \models (\gamma_1 \wedge \gamma_2 \wedge \neg H) && \text{(A.16)} \\
& \quad \wedge(\exists m \geq k : \kappa(M^+), s_m \models H) \wedge && \text{(A.17)} \\
& \quad \forall i : 0 \leq i < k : \kappa(M^+), s_i \models \neg\gamma_1 \wedge \neg\gamma_2 \wedge \neg H && \text{(A.18)}
\end{aligned}
$$

Again, only the traces where Γ is the actual cause of H are of interest. Therefore we assume (A.15) to be false and the second part of the whole equation to be true.

Let k_0 be the smallest k where $\gamma_1 \wedge \gamma_2 \wedge \neg H$ is true and m_0 the smallest m where H is true, then we know that $m_0 > k_0$ because H is false at k_0 (A.16). On all time steps before k_0 both γ_1 and γ_2 are false (A.18). Therefore H appears after the failures and the first time the failures appear, they appear together, i.e. $\gamma_1 \sim_\Gamma \gamma_2$. □

Lemma 6. aDCCA Completeness Lemma

For a complete aDCCA for an extended system model M^+, a set of failure modes Δ and a hazard H the following formula holds:

$$\kappa(M^+), s_0 \models \mathbf{A}\,((\bigwedge_{\Gamma \in aMCSS(M^+,\Delta,H)} \neg \bigwedge_{\gamma_j \in \Gamma} \mathbf{F}\,\gamma_j) \rightarrow \neg \mathbf{F}\,\mathbf{G}\,H)$$

Proof. By contradiction:

$$\begin{aligned}
&\text{Assume:}\\
&\kappa(M^+), s_0 \not\models \mathbf{A}\,((\textstyle\bigwedge_{\Gamma \in aMCSS(M^+,\Delta,H)} \neg \bigwedge_{\gamma \in \Gamma} \mathbf{F}\,\gamma) \rightarrow \neg \mathbf{F}\,\mathbf{G}\,H) && \text{(A.19)}\\
\Leftrightarrow\quad &\text{there exists } \pi = s_0, s_1, \ldots \in Paths(\kappa(M^+)) : && \text{(A.20)}\\
&\kappa(M^+), \pi \models (\textstyle\bigwedge_{\Gamma \in aMCSS(M^+,\Delta,H)} \neg \bigwedge_{\gamma \in \Gamma} \mathbf{F}\,\gamma) && \text{(A.21)}\\
&\text{and } \kappa(M^+), \pi \models \mathbf{F}\,\mathbf{G}\,H && \text{(A.22)}
\end{aligned}$$

Since the set Δ is finite, there is a state s_i on π such that all component failures $\gamma \in \Delta$ that will ever happen have occurred at least once. This means all failure modes that have not appeared on π before s_i will not appear at all.

$$\begin{aligned}
&\forall \gamma \in \Delta :\ (\forall j < i :\ \kappa(M^+), s_j \models \neg\gamma) && \text{(A.23)}\\
&\qquad \rightarrow \kappa(M^+), s_i \models \mathbf{G}\,\neg\gamma\\
\text{Let}\quad &\Gamma := \{\gamma \in \Delta \mid \exists\, j < i\ : \kappa(M^+), s_j \models \gamma\} && \text{(A.24)}\\
\overset{\text{(A.22),(A.24)}}{\Rightarrow}\quad &\kappa(M^+), \pi \models (\overline{\Gamma}\mathbf{U}\ \mathbf{E}\,\mathbf{G}\,(H \wedge \overline{\Gamma}))\\
\Leftrightarrow\quad &\Gamma \text{ is adaptive-critical set (Def. 32)}\\
\Rightarrow\quad &\exists\, \Gamma_0 \subseteq \Gamma :\ \Gamma_0 \text{ is minimal adaptive-critical set}\\
&\text{therefore } \Gamma_0 \in aMCSS(M^+, \Delta, H)\\
\overset{\text{(A.24)}}{\Rightarrow}\quad &\kappa(M^+), \pi \models \textstyle\bigwedge_{\gamma \in \Gamma} \mathbf{F}\,\gamma && \text{(A.25)}\\
\Rightarrow\quad &\kappa(M^+), \pi \models \textstyle\bigwedge_{\gamma \in \Gamma_0} \mathbf{F}\,\gamma, \quad \text{as } \Gamma_0 \subseteq \Gamma\\
\Rightarrow\quad &\kappa(M^+), \pi \not\models (\textstyle\bigwedge_{\Gamma \in aMCSS(M^+,\Delta,H)} \neg \bigwedge_{\gamma \in \Gamma} \mathbf{F}\,\gamma)\\
\Rightarrow\quad &\text{contradiction to (A.20)}
\end{aligned}$$

□

Lemma 7. DCCA Implication for aDCCA

Let M^+ be an extended SAML model, Δ be a finite set of failure modes, $\Gamma \subseteq \Delta$, and M be the embedded Kripke structure of $\kappa(M^+)$ then

$$\kappa(M^+), s_0 \models \mathbf{E}\,[\overline{\Gamma}\mathbf{U}\,\mathbf{E}\,\mathbf{G}\,(H \wedge \overline{\Gamma})] \Rightarrow \kappa(M^+), s_0 \models \mathbf{E}\,[\overline{\Gamma}\mathbf{U}\,H]$$

Proof. Let π be a path of $\kappa(M^+)$ such that $\pi = s_0 s_1 \ldots s_n \ldots$ where n is the smallest index such that $\forall j \geq n : H \in L(s_j)$, i.e. H holds globally after s_n.

Such a trace exists because $\mathbf{E}\,[\overline{\Gamma}\mathbf{U}\,\mathbf{E}\,\mathbf{G}\,(H \wedge \overline{\Gamma})]$ holds. Therefore $\forall i < n : \overline{\Gamma} \in L(s_i)$ and $H \in L(s_n)$ which together imply $\kappa(M^+), s_0 \models \mathbf{E}\,[\overline{\Gamma}\mathbf{U}\,H]$. □

Lemma 8. Adaptive Deductive Before Ordering / Strict Before Ordering

For an extended system model M^+, a set of failure modes Δ, a predicate logic hazard H, a minimal adaptive-critical set Γ and two failure modes $\gamma_1, \gamma_2 \in \Gamma$,

$$\begin{aligned}
&\gamma_1 \preceq_\Gamma \gamma_2 \Leftrightarrow \\
&\kappa(M^+), \pi \models \quad (\overline{\Gamma}\mathbf{U}(\mathbf{G}\,(H \wedge \overline{\Gamma})) \rightarrow [(\neg\gamma_1 \wedge \neg\gamma_2) \\
&\qquad\qquad \mathbf{U}(\gamma_1 \wedge ((\neg\gamma_2) \\
&\qquad\qquad\quad \mathbf{U}(\gamma_2 \wedge (\mathbf{F}\,\mathbf{G}\,H))))]
\end{aligned} \tag{A.26}$$

$$\begin{aligned}
&\gamma_1 \prec_\Gamma \gamma_2 \Leftrightarrow \\
&\kappa(M^+), \pi \models \quad (\overline{\Gamma}\mathbf{U}(\mathbf{G}\,(H \wedge \overline{\Gamma}))) \rightarrow [(\neg\gamma_1 \wedge \neg\gamma_2) \\
&\qquad\qquad \mathbf{U}((\gamma_1 \wedge \neg\gamma_2) \wedge \\
&\qquad\qquad\quad \mathbf{X}((\neg\gamma_2) \\
&\qquad\qquad\qquad \mathbf{U}(\gamma_2 \wedge (\mathbf{F}\,\mathbf{G}\,H))))]
\end{aligned} \tag{A.27}$$

Proof. First the expressions in Eq. (A.26) and Eq. (A.27) are expanded, the following reasoning is analogous to the reasoning of Lemma (4).

□

Lemma 9. Adaptive Deductive Simultaneous Order

For an extended system model M^+, a set of failure modes Δ, a predicate logic hazard H, a minimal adaptive-critical set Γ and two failure modes $\gamma_1, \gamma_2 \in \Gamma$

$$\begin{aligned}
&\gamma_1 \sim_\Gamma \gamma_2 \Leftrightarrow \\
&\kappa(M^+), \pi \models \quad (\overline{\Gamma}\mathbf{U}(\mathbf{G}\,(H \wedge \overline{\Gamma})) \rightarrow [(\neg\gamma_1 \wedge \neg\gamma_2) \\
&\qquad\qquad \mathbf{U}((\gamma_1 \wedge \gamma_2) \wedge (\mathbf{F}\,\mathbf{G}\,H))]
\end{aligned} \tag{A.28}$$

Proof. The reasoning is analogous to Lemma (5).

□

Lemma 10. Transformation Equivalence

Let M be s SAML model with state variables $v_1, \ldots, v_k$, and update rules $u_1 \ldots u_m$ and the corresponding MDP τ_{MDP}. Let $U := \bigcup_{i=1}^{m} \{\mu(u_i)\}$ be the set of update tuples of M with eliminated parallel assignments.

Let M' be the parallel composition $M_1 || \cdots || M_k || M_{choose_1} || \cdots || M_{choose_m}$ where M_i is a SAML module with the single state variable v_i and all the updates of the set U of the form (ϕ, v_i, Θ) and M_{choose_i} is a SAML module with the chooser variable $choose_i$ and the update rules for $choose_i$ from U of the form

$$(true, \{choose_i\}, \{\{(1, \{(choose_i, [1, j] \subseteq \mathbb{N}\}\}\})$$

Let $\Lambda := \{\tau'_{MDP}\}$ be the set of MDPs corresponding to the set of possible M' with a fixed initial value for each $choose_i$ variable, then

$$\forall \pi : (\pi \in Paths(\kappa(\tau_{MDP})) \Leftrightarrow \exists \tau'_{MDP} \in \Lambda : \pi \equiv \pi'|_{\{v_1,\ldots,v_k\}} \wedge \pi' \in Paths(\kappa(\tau'_{MDP})))$$

Proof. $\Rightarrow$:

Let $\pi \in Paths(\kappa(\tau_{MDP}))$, $\pi = s_0 \ldots s_n s_{n+1} \ldots$ be a path of $\kappa(\tau_{MDP})$. Then each element (s_n, s_{n+1}) in the transition relation T of $\kappa(\tau_{MDP})$ results from the existence of a probability distribution p in τ_{MDP} where $p(s_n, s_{n+1}) > 0$ (from the definition of the embedded Kripke structure)

To the MDP τ_{MDP} and $p(s_n, s_{n+1})$, there exits an activation condition $\bigwedge_{j=1}^{m} \phi_j$ which holds in s_n in the complete SAML model of M (see definition of parallel composition) where each ϕ_j is the activation condition of one update rule u_j of the parallel SAML modules of M.

Let $choose_j$ be the chooser variable introduced by the parallel assignment elimination μ of update rule u_j and t_j be one of the admissible indices of $choose_j$.

Assume that there exists a state s'_n on a path $\pi' \in Paths(\kappa(\tau'_{MDP}))$ of one $\tau_{MDP} \in \Lambda$ that for each for each admissible value of $chooser_j$ of each update rule u_j for which the activation condition ϕ_j holds in s_n the state s'_n is of the form $s'_n|_{var_j, choose_j} = \begin{pmatrix} s_n \\ t_j \end{pmatrix}$ where (s_i) denotes a vector of the values of all state variables of the state s_i and var_j is the set of variables of u_j. Then the activation conditions of each u_j also hold in s'_n (in the form $\phi_j \wedge chooser_j = t_j$) and therefore there also exists a successor state s'_{n+1} of s'_n for which $s'_{n+1}|_{var_j} = s_{n+1}$ holds, because the state variables of each update rule u_j get new values assigned from a parallel assignment in τ_{MDP} which gets the index t_j assigned by the mapping μ and whenever $choose_j$ has the value t_j (as in s'_n), the next values of the variables in var_j are the same as in the original parallel assignment of the complete SAML model of M. By construction of the update rules of the chooser variables, such a successor state s'_{n+1} exists for all possible valuations of all chooser variables.

By the definition of the chooser variables, there exists a MDP τ'_{MDP} in Λ such that $s'_0|_{v_1,\ldots v_k} = s_0$ for the initial state s'_0 of τ'_{MDP} and $s'_0|_{v_1,\ldots,v_k,choose_i} = \begin{pmatrix} s_0 \\ t'_i \end{pmatrix}$ for each possible chooser variable $choose_i$ and admissible index t'_i of $choose_i$. Therefore, a corresponding successor states can be found for each path starting in the initial state s'_0 of $\kappa(\tau_{MDP})$ and the assertion follows by induction.

$\Leftarrow$:

By contradiction: Assume $\exists\pi : \exists\tau'_{MDP} \in \Lambda : \pi \equiv \pi'|_{\{v_1,\ldots,v_k\}} \wedge \pi' \in Paths(\kappa(\tau'_{MDP}))$ but $\pi \notin Paths(\kappa(\tau_{MDP}))$. For each successor states s_n, s_{n+1} on π there existed states s'_n, s'_{n+1} with $s'_n|_{v_1,\ldots,v_k} = s_n$, $s'_{n+1}|_{v_1,\ldots,v_k} = s_{n+1}$ and a non-deterministic choice of a probability distribution in τ'_{MDP} such that $p(s'_n, s'_{n+1}) > 0$ (else s'_n, s'_{n+1} would not be successor states on π' and therefore s_n, s_{n+1} not on π).

From the construction of τ'_{MDP} via the parallel assignment elimination with μ, any such probabilistic distribution is of the form $p(s'_n, s'_{n+1}) = 1$ and results from the existence of a probability distribution $\overline{p}$ of τ_{MDP} such that $\overline{p}(s'_n|_{\{v_1,\ldots,v_k\}}, s'_{n+1}|_{\{v_1,\ldots,v_k\}}) > 0$ and $(s'_n|_{\{v_1,\ldots,v_k\}}, s'_{n+1}|_{\{v_1,\ldots,v_k\}})$ is in the transition relation of the embedded Kripke structure of τ_{MDP}.

Therefore for each pair of successor states (s_n, s_{n+1}) of π, for the transition relation T of $\kappa(\tau_{MDP})$: $(s'_n|_{\{v_1,\ldots,v_k\}}, s'_{n+1}|_{\{v_1,\ldots,v_k\}}) = (s_n, s_{n+1}) \in T$ holds. This contradicts the assumption that no such π exists in $\kappa(\tau_{MDP})$. □

Lemma 11. Single Initial State Extension

Let M be a SAML model with state variables $v_1 \ldots v_k$ and M' the SAML model which results from parallel assignment elimination and single state extension of M, then

$$\begin{aligned} & \forall\pi : (\pi = s_0 s_1 \ldots \in Paths(\kappa(M)) \\ \Leftrightarrow\ & \exists\pi' = s'_{-1}s'_0 \ldots \in Paths(\kappa(M')) : s_0 \equiv s'_{-1}|_{v_1,\ldots,v_k} \wedge \forall i \geq 0 : s_i \equiv s'_i|_{v_1,\ldots,v_k}) \end{aligned}$$

Proof. "$\Rightarrow$:"

From Lemma (10) it is known, that for each $\pi = s_0 s_1 \ldots \in Paths(\kappa(M))$, there exists a path $\pi' = s'_0 s'_1 \ldots$ in one of the MDPs resulting from the parallel assignment elimination of M for which $\forall i \geq 0 : s_i \equiv s'_i|_{v_1,\ldots,v_k}$. Therefore it is sufficient to show that every initial state of one of the MDPs of Λ is a successor state of s'_{-1}.

From the construction, it is known that $s'_{-1}|_{v_1,\ldots,v_k} \equiv s'_0|_{v_1,\ldots,v_k}$, i.e. all the state variables do not change their value in the first transition. At the same time

$$s_{-1}|_{choose_1,\ldots,choose_j,\ldots,choose_n} = \begin{pmatrix} 1 \\ \vdots \\ 0 \\ \vdots \\ 1 \end{pmatrix}$$

holds, i.e. $choose_j$ is the global chooser variable with new initial state 0 and all other chooser variables have the initial value 1 (or any other of their admissible values, but always a unique fixed one for s_{-1}).

By construction of the single state extension, the values for all chooser variables therefore are any of their possible admissible indices in the successor states of s_{-1} and for every state $s'_n, n \geq 0$, $choose_j \neq 0$. This means, in every successor state s'_0 of s'_{-1} the state variables $v_1 \dots v_k$ have their respective initial values and there exists one such successor state for every possible admissible valuation of the chooser variables. This means the set of successor states of s'_{-1} is exactly the set of initial states of the MDPs in Λ.

"$\Leftarrow$:"

By Contradiction: Assume

$$\exists \pi' = s'_{-1}s'_0 \dots \in Paths(\kappa(M')) : s_0 \equiv s'_{-1}|_{v_1,\dots,v_k} \wedge \forall i \geq 0 : s_i \equiv s'_i|_{v_1,\dots,v_k}$$

but $\pi = s_0 s_1 \dots \notin Paths(\kappa(M))$. From Lemma (10) we can conclude that in this situation s'_{-1} must have a successor state s'_0 which is not the initial state of one of the MDPs resulting from the parallel assignment elimination (else $\pi \in Paths(\kappa(M))$ would hold).

From the definition of the single state extension, all state variables do not change their values in the transition from s'_{-1} to s'_0, the global chooser gets one of its admissible values assigned after the first step and all other chooser variable also get one of their respective admissible values assigned. Therefore every successor state of s'_{-1} is the initial state of one of the MDPs resulting from the parallel state assignment which is a contradiction to the assumption. □

B. Peer-Reviewed Publications

- 2011
  - *Model-Based Multi-Objective Safety Optimization* Matthias Güdemann, Frank Ortmeier, Proceedings of the 30^{th} International Conference on Computer Safety, Reliability and Security (SAFECOMP 2011), Springer LNCS (to appear 19.9.2011)
  - *Towards Model-driven Safety Analysis* Matthias Güdemann, Frank Ortmeier, Proceedings of the 3^{rd} International Workshop On Dependable Control Of Discrete Systems (DCDS 2011), IEEE
- 2010
  - *A Framework for Qualitative and Quantitative Model-Based Safety Analysis* Matthias Güdemann, Frank Ortmeier, Proceedings of the 12^{th} High Assurance System Engineering Symposium (HASE 2010)
  - *Quantitative Model-Based Safety Analyis: A Case Study* Matthias Güdemann, Frank Ortmeier, Proceedings of the 5^{th} conference for Sicherheit, Schutz und Zuverlaessigkeit (SICHERHEIT 2010), Lecture Notes in Informatics (LNI)
  - *SysML in Digital Engineering* Matthias Güdemann, Stefan Kegel, Frank Ortmeier, Olaf Poenicke, Klaus Richter, Proceedings of the 1^{st} International Workshop on Digital Engineering (IWDE 2010), ACM Digital Library
  - *Probabilistic Model-Based Safety Analysis* Matthias Güdemann, Frank Ortmeier, Proceedings of the 8^{th} Workshop of Quantitative Aspects of Programming Languages (QAPL 2010) at ETAPS 2010, EPTCS
  - *ProMoSA - Probabilistic Models for Safety Analysis* Frank Ortmeier, Matthias Güdemann Modellbasierte Entwicklung eingebetteter Systeme (MBEES 2010), 03.02.2010 - 05.02.2010, Schloss Dagstuhl
- 2008
  - *Computing Ordered Minimal Critical Sets* Matthias Güdemann, Frank Ortmeier, Wolfgang Reif Proceedings of Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS / FORMAT 2008) (eds. G. Tarnai & E. Schnieder)

  - *A specification and construction paradigm for Organic Computing systems* M. Güdemann, F.Nafz, F.Ortmeier, H.Seebach and W.Reif, Proceedings of the 2^{nd} IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008), IEEE Computer Society Press
  - *Developing Safety-Critical Mechatronical Systems* Matthias Güdemann, Frank Ortmeier, Wolfgang Reif 7. Internationales Heinz Nixdorf Symposium: Selbstoptimierende mechatronische Systeme, HNI Schriftenreihe

- 2007
  - *Using Deductive Cause-Consequenc Analysis (DCCA) with SCADE* Matthias Güdemann, Frank Ortmeier, Wolfgang Reif Proceedings of 27^{th} SAFECOMP 2007, LNCS 4680, Springer, LNCS
  - *Formal Failure Models* Frank Ortmeier, Matthias Güdemann, Wolfgang Reif Proceedings of the 1^{st} IFAC Workshop on Dependable Control of Discrete Systems (DCDS 2007), Elsevier
  - *Modeling of self-adaptive systems with SCADE* Matthias Güdemann, Andreas Angerer, Frank Ortmeier, Wolfgang Reif Proceedings of the 40^{th} IEEE International Symposium on Circuits and Systems (ISCAS 2007), IEEE

- 2006
  - *Safety and Dependability Analysis of Self-Adaptive Systems* M. Güdemann, F. Ortmeier, W. Reif Proceedings of the 2^{nd} International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2006), IEEE Computer Society Press
  - *Towards Safe and Secure Organic Computing Applications* Matthias Güdemann, Florian Nafz, Wolfgang Reif and Hella Seebach C. Hochberger and R. Liskowsky, editors, INFORMATIK 2006 – Informatik für Menschen, volume P-93 of GI-Edition – Lecture Notes in Informatics, Köllen Verlag
  - *Formal Modeling and Verification of Systems with Self-x Properties* Matthias Güdemann, Frank Ortmeier and Wolfgang Reif Proceedings of the 3^{rd} International Conference on Autonomic and Trusted Computing (ATC 2006), Springer

Bibliography

[ADS+04] Parosh Aziz Abdulla, Johann Deneux, Gunnar Stalmarck, Herman Agren, and Ove Åkerlund. Designing safe, reliable systems using SCADE. In *Proceedings of the* 1^{st} *International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 04)*. Springer, 2004.

[AFG+09] Husain Aljazzar, Manuel Fischer, Lars Grunske, Matthias Kuntz, Florian Leitner-Fischer, and Stefan Leue. Safety analysis of an airbag system using probabilistic fmea and probabilistic counterexamples. In *Proceedings of the 2009 Sixth International Conference on the Quantitative Evaluation of Systems (QEST 2009)*, pages 299–308, Washington, DC, USA, 2009. IEEE Computer Society.

[AK09] Robert Alexander and Tim Kelly. Escaping the non-quantitative trap. In *Proceedings of the* 27^{th} *International System Safety Conference (ISSC 2009)*, 2009.

[AKLFL10] Husain Aljazzar, Matthias Kuntz, Florian Leitner-Fischer, and Stefan Leue. Directed and heuristic counterexample generation for probabilistic model checking: a comparative evaluation. In *Proceedings of the 2010 ICSE Workshop on Quantitative Stochastic Models in the Verification and Design of Software Systems (QUOVADIS 2010)*, pages 25–32, New York, NY, USA, 2010. ACM.

[APGR99] André Arnold, Gérald Point, Alain Griffault, and Antoine Rauzy. The altarica formalism for describing concurrent systems. *Fundam. Inf.*, 40:109–124, November 1999.

[AVA03] AVACS. Automatic verification and analysis of complex systems. Technical report, University of Oldenburg, 2003.

[B+03] M. Bozzano et al. ESACS: An integrated methodology for design and safety analysis of complex systems. In *Proceedings of European Safety and Reliability Conference (ESREL 2003)*, pages 237–245, Maastricht, The Netherlands, 2003. Balkema publisher.

[BaFV04] B. Berthomieu and P.-O. Ribet an F. Vernadat. The tool tina - construction of abstract state spaces for petri nets and time petri nets. *International Journal of Production Research*, 42, 2004.

[BB03] Marc Bouissou and Jean-Louis Bon. A new formalism that combines advantages of fault-trees and markov models: Boolean logic driven markov processes. *Reliability Engineering & System Safety*, 82(2):149 – 163, 2003.

[BBC+04] Pierre Bieber, Christian Bougnol, Charles Castel, Jean pierre Heckmann, Christophe Kehren, Sylvain Metge, Christel Seguin, P. Bieber, C. Bougnol, C. Castel, J. p. Heckmann, C. Kehren, and C. Seguin. Safety assessment with altarica - lessons learnt based on two aircraft system studies. In *Proceedings of the* 18^{th} *IFIP World Computer Congress, Topical Day on New Methods for Avionics Certification*, page 26, 2004.

[BBC+08] Jiri Barnat, Lubos Brim, Ivana Cerná, Milan Ceska, and Jana Tumova. ProbDiVinE-MC: Multi-core LTL Model Checker for Probabilistic Systems. In *Proceedings of the* 5^{th} *International Conference on Quantitative Evaluation of Systems (QEST 2008)*, pages 77–78, 2008.

[BBR08] M. Balser, S. Bäumler, and W. Reif. An interval temporal logic with compositional interleaving. Technical Report 2008-20, University of Augsburg, 2008.

[BCF+08] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. The mathsat 4 smt solver. In Aarti Gupta and Sharad Malik, editors, *Proceedings of the International Conference on Computer Aided Verification (CAV 2008)*, volume 5123 of *Lecture Notes in Computer Science*, pages 299–303. Springer Berlin / Heidelberg, 2008.

[BCK+09a] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. COMPASS project webpage. http://compass.informatik.rwth-aachen.de/, 2009.

[BCK+09b] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. Model-based codesign of critical embedded systems. In *Proceedings of the* 2^{nd} *International Workshop on Model Based Architecting and Construction of Embedded Systems (ACES-MB 2009)*, volume 507, pages 87–91. CEUR Workshop Proceedings, 2009.

[BCK+10a] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Nguyen, Thomas Noll, Marco Roveri, and Ralf Wimmer. A model checker for AADL. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Proceedings of the Computer Aided Verification Conference (CAV 2010)*, volume 6174 of *Lecture Notes in Computer Science*, pages 562–565. Springer Berlin / Heidelberg, 2010.

[BCK+10b] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. Safety, dependability, and performance analysis of extended AADL models. *The Computer Journal*, 2010.

[BCM+90] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedingsof the 5^{th} Annual IEEE Symposium on Logic in Computer Science (LICS 1990)*, pages 428–439, Philadelphia, Pennsylvania, 1990. IEEE Computer Society Press.

[BCR+09] Marco Bozzano, Alessandro Cimatti, Marco Roveri, Joost-Pieter Katoen, Viet Yen Nguyen, and Thomas Noll. Verification and performance evaluation of AADL models. In *Proceedings of the the 7^{th} joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (ESEC/FSE 2009)*, ESEC/FSE '09, pages 285–286, New York, NY, USA, 2009. ACM.

[BCS02] P. Bieber, C. Castel, and C. Seguin. Combination of fault tree analysis and model checking for safety assessment of complex systems. In *Proceedings of the 4^{th} European Dependable Computing Conference (EDCC 2002)*, volume 2485 of *LNCS*, pages 19–31, Toulouse, France, 2002. Springer-Verlag.

[BCS07a] Hichem Boudali, Pepijn Crouzen, and Mariëlle Stoelinga. A compositional semantics for dynamic fault trees in terms of interactive markov chains. In *Proceedings of the 5th international conference on Automated technology for verification and analysis (ATVA 2007)*, pages 441–456, Berlin, Heidelberg, 2007. Springer-Verlag.

[BCS07b] Hichem Boudali, Pepijn Crouzen, and Marielle Stoelinga. Dynamic fault tree analysis using input/output interactive markov chains. In *Proceedings of the 37^{th} Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2007)*, pages 708–717, Washington, DC, USA, 2007. IEEE Computer Society.

[BD04] Simona Bernardi and Susanna Donatelli. Stochastic petri nets and inheritance for dependability modelling. In *Proceedings of the 10^th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2004)*, pages 363 – 372, march 2004.

[BDW00] T. Bienmöller, W. Damm, and H. Wittke. The STATEMATE verification environment – making it real. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12^{th} international Conference on Computer Aided Verification (CAV 2000)*, number 1855 in LNCS, pages 561–567, Chicago, IL, USA, 2000. Springer.

[BES09] D. Bosnacki, S. Edelkamp, and D. Sulewski. Efficient probabilistic model checking on general purpose graphics processors. In C. Pasareanu, editor, *Proceedings of the* 16^{th} *International SPIN Workshop (SPIN 2009)*, volume 5578 of *LNCS*, pages 32–49. Springer, 2009.

[BF93] Ricky W. Butler and George B. Finelli. The infeasibility of quantifying the reliability of life-critical real-time software. *IEEE Transactions on Software Engineering*, 19:3–12, 1993.

[Bit01] Friedemann Bitsch. Safety patterns - the key to formal specification of safety requirements. In Udo Voges, editor, *Proceedings of the* 20^{th} *nternational Conference on Computer Safety, Reliability and Security (SAFECOMP 2001)*, volume 2187 of *LNCS*, pages 176–189. Springer Berlin / Heidelberg, 2001.

[BK86] J.A. Bergstra and J.W. Klop. Algebra of communicating processes. In *Proceedings CWI Symposium Mathematical and Computer Science*, CWI Monographs I series, 1986.

[Bou07] Marc Bouissou. A generalization of dynamic fault trees through boolean logic driven markov processes (BDMP) ®. In *Proceedings of the European Safety and Reliability Conference (ESREL 2007)*. Springer-Verlag, 2007.

[Boz11] Marco Bozzano. Automated Generation of Compact FMEA Tables in the COMPASS Toolset. Talk at CISEC Model Based Safety Assessment Workshop (MBSAW 2011), 2011. accessed April 2011.

[BPRW08a] Eckard Böde, Thomas Peikenkamp, Jan Rakow, and Samuel Wischmeyer. Model based importance analysis for minimal cut sets. Reports of SFB/TR 14 AVACS 29, SFB/TR 14 AVACS, Apr 2008. ISSN: 1860-9821.

[BPRW08b] Eckard Böde, Thomas Peikenkamp, Jan Rakow, and Samuel Wischmeyer. Model based importance analysis for minimal cut sets. In Sung Deok Cha, Jin-Young Choi, Moonzoo Kim, Insup Lee, and Mahesh Viswanathan, editors, *Proceedings of the* 6^{th} *International Symbposium on Automated Technology for Verification and Analysis (ATVA 2008)*, volume 5311 of *LNCS*, pages 303–317, 2008.

[BV03a] M. Bozzano and A. Villafiorita. Integrating fault tree analysis with event ordering information. In *Proceedings of Proceedings of the European Safety & Reliability Conference (ESREL 2003)*, pages 247–254. Balkema, 2003.

[BV03b] M. Bozzano and Adolfo Villafiorita. Improving system reliability via model checking: the FSAP/NuSMV-SA safety analysis platform. In *Proceedings*

of the 22nd International Conference on Computer Safety, Reliability and Security (SAFECOMP 2003), pages 49–62. Springer, 2003.

[CAB+01] W. Chan, R. J. Anderson, P. Beame, D. H. Jones, D. Notkin, and W. E. Warner. Optimizing symoblic model checking for statecharts. *IEEE Transactions on Software Engineering*, 27(2):170 – 190, 2001.

[CBRZ01] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. In *Formal Methods in System Design*, page 2001, 2001.

[CCG+02] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV 2002)*, volume 2404 of *LNCS*. Springer, 2002.

[CCOR11] Roberto Cavada, Alessandro Cimatti, Emanuele Olivetti, and Marco Pistoreand Marco Roveri. NuSMV v2.5 User Manual, accessed: January 2011.

[CdAFL09] Krishnendu Chatterjee, Luca de Alfaro, Marco Faella, and Axel Legay. Qualitative logics and equivalences for probabilistic systems. *Logical Methods in Computer Science*, 5(2), 2009.

[CG04] Frank Ciesinski and Marcus Größer. On probabilistic computation tree logic. In Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, Joost-Pieter Katoen, and Markus Siegle, editors, *Validation of Stochastic Systems*, volume 2925 of *LNCS*, pages 147–188. Springer, 2004.

[CGP00] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[CH00] E. M. Clarke and W. Heinle. Modular translation of statecharts to smv. Technical report, School of Computer Science, Carnegie Mellon University, 2000.

[Cla08] Edmund Clarke. The birth of model checking. In Orna Grumberg and Helmut Veith, editors, *25 Years of Model Checking*, volume 5000 of *LNCS*, pages 1–26. Springer, 2008.

[CMZ02] A. Cau, B. Moszkowski, and H. Zedan. *ITL – Interval Temporal Logic*. Software Technology Research Laboratory, SERCentre, De Montfort University, The Gateway, Leicester LE1 9BH, UK, 2002. www.cms.dmu.ac.uk/∼cau/itlhomepage.

[CR05] Daniele 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)*, pages 45 – 60. ENTCS, 2005.

[dAFH+05] Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. Model checking discounted temporal properties. *Theoretical Computer Science*, 345:139–170, 2005.

[dAKN+00] L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of probabilistic processes using MTBDDs and the Kronecker representation. In S. Graf and M. Schwartzbach, editors, *Proceedings of the 6^{th} International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000)*, volume 1785 of *LNCS*, pages 395–410. Springer, 2000.

[Dow97] Mark Dowson. The ariane 5 software failure. *SIGSOFT Softw. Eng. Notes*, 22:84–, March 1997.

[ea03] M. Bozzano et al. ESACS: An integrated methodology for design and safety analysis of complex systems. In *Proceedings of the European Conference on Safety and Reliability (ESREL 2003)*, 2003.

[Est11] Esterel Technologies, http://www.esterel-technologies.com/?id=13281, accessed Feb. 28. 2011. SCADE Suite, 2011.

[FGK+96] Jean-Claude Fernandez, Hubert Garavel, Alain Kerbrat, Laurent Mounier, Radu Mateescu, and Mihaela Sighireanu. CADP - a protocol validation and verification toolbox, 1996.

[FGP+08] Patrick Farail, Pierre Gaufillet, Florent Peres, Jean-Paul Bodeveix, Mamoun Filali, B. Berthomieu, Saad Rodrigo, F. Vernadat, Hubert Garavel, and Frédéric Lang. FIACRE: an intermediate language for model verification in the TOPCASED environment. In *European Congress on Embedded Real-Time Software (ERTS 2008)*, http://www.see.asso.fr, janvier 2008. SEE.

[FM92] Peter Fenelon and John A. McDermid. New directions in software safety: Causal modelling as an aid to integration. Technical report, High Integrity Systems Engineering Group, Department of Computer Science, University of York, 1992.

[FM93] Peter Fenelon and John A McDermid. An integrated tool set for software safety analysis. *Journal of Systems and Software*, 21(3):279 – 290,

1993. Applying Specification, Verification, and Validation Techniques to Industrial Software Systems.

[GCW07] L. Grunske, R. Colvin, and K. Winter. Probabilistic model-checking support for FMEA. In *Proceedings of the* 4^{th} *International Conference on Quantitative Evaluation of Systems (QEST 2007)*. IEEE, 2007.

[GH08] L. Grunske and Jun Han. A comparative study into architecture-based safety evaluation methodologies using aadl's error annex and failure propagation models. In *Proceedings of the* 11^{th} *High Assurance Systems Engineering Symposium (HASE 2008)*, pages 283 –292, 2008.

[GLYW05] Lars Grunske, Peter Lindsay, Nisansala Yatapanage, and Kirsten Winter. An automated failure mode and effect analysis based on high-level design specification with behavior trees. In Judi Romijn, Graeme Smith, and Jaco van de Pol, editors, *Integrated Formal Methods*, volume 3771 of *Lecture Notes in Computer Science*, pages 129–149. Springer, 2005.

[GMMP02] A. Galloway, J.A. McDermid, J. Murdoch, and D.J. Pumfrey. Automation of system safety analysis: Possibilities and pitfalls. In *Porceedings of the* 20^{th} *International System Safety Conference (ISSC 2002)*, 2002.

[GNO+08] M. Güdemann, F. Nafz, F. Ortmeier, H. Seebach, and W. Reif. A specification and construction paradigm for Organic Computing systems. In Sven Brueckner, Paul Robertson, and Umesh Bellur, editors, *Proceedings of the* 2^{nd} *IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008)*, pages 233–242. IEEE Computer Society Press (2008), 2008.

[GO10a] Matthias Güdemann and Frank Ortmeier. A framework for qualitative and quantitative model-based safety analysis. In *Proceedings of the* 12^{th} *High Assurance System Engineering Symposium (HASE 2010)*, 2010.

[GO10b] Matthias Güdemann and Frank Ortmeier. Probabilistic model-based safety analysis. In *Proceedings of the* 8^{th} *Workshop on Quantitative Aspects of Programming Languages (QAPL 2010)*. EPTCS, 2010.

[GO10c] Matthias Güdemann and Frank Ortmeier. Quantitative model-based safety analysis: A case study. In *Proceedings of 5th conference for Sicherheit, Schutz und Zuverlaessigkeit (SICHERHEIT 2010)*. Lecture Notes in Informatics (LNI), 2010.

[GO11a] Matthias Güdemann and Frank Ortmeier. Model-Based Multi-Objective Safety Optimization. In *Proceedings of the* 30^{th} *International Conference*

on Computer Safety, Reliability and Security (SAFECOMP 2011). Springer LNCS, 2011. to appear: 19.9.2011.

[GO11b] Matthias Güdemann and Frank Ortmeier. Towards Model-driven Safety Analysis. In *Proceedings of the 3rd international Workshop on Dependable Control of Discrete Systems (DCDS 2011)*. IEEE, 2011.

[GOR06a] M. Güdemann, F. Ortmeier, and W. Reif. Formal modeling and verification of systems with self-x properties. In Laurence T. Yang, Hai Jin, Jianhua Ma, and Theo Ungerer, editors, *Proceedings of the Third International Conference on Autonomic and Trusted Computing (ATC 2006)*, LNCS, pages 38–47. Springer, September 2006.

[GOR06b] M. Güdemann, F. Ortmeier, and W. Reif. Safety and dependability analysis of self-adaptive systems. In *Proceedings of the* 2^{nd} *International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2006)*. IEEE CS Press, 2006.

[GOR07] Matthias Güdemann, Frank Ortmeier, and Wolfgang Reif. Using deductive cause consequence analysis (DCCA) with SCADE. In *Proceedings of the* 26^{th} *International Conference on Computer Safety, Reliability and Security (SAFECOMP 2007)*. Springer LNCS 4680, 2007.

[GOR08] M. Güdemann, F. Ortmeier, and W. Reif. Computing ordered minimal critical sets. In E. Schnieder and G. Tarnai, editors, *Proceedings of the* 7^{th} *Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 2008)*, 2008.

[GPM09] Xiaocheng Ge, Richard F. Paige, and John A. McDermid. Probabilistic failure propagation and transformation analysis. In Bettina Buth, Gerd Rabe, and Till Seyfarth, editors, *Proceedings of the* 28^{th} *International Conference on Computer Safety, Reliability and Security (SAFECOMP 2009)*, LNCS, pages 215–228. Springer, 2009.

[Gro06] OMG Object Modeling Group. The unified modeling language, 2006.

[Gru08] Lars Grunske. Specification patterns for probabilistic quality properties. In *Proceedings of the 30th international conference on Software engineering (ICSE 2008)*, pages 31–40, New York, NY, USA, 2008. ACM.

[GV04] Alain Griffault and Aymeric Vincent. The Mec 5 model-checker. In *Proceedings of the International Conference on Computer Aided Verification (CAV 2004)*, volume 3114 of *LNCS*, pages 488–491. Springer, July 2004.

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data-flow programming language Lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[HJ94] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:102–111, 1994.

[HK98] Michael Huth and Martha Z. Kwiatkowska. Comparing ctl and pctl on labeled markov chains. In *Proceedings of the IFIP TC2/WG2.2,2.3 International Conference on Programming Concepts and Methods (PROCOMET 1998)*, pages 244–262, London, UK, UK, 1998. Chapman & Hall, Ltd.

[HKB09] Tingting Han, Joost-Pieter Katoen, and Damman Berteun. Counterexample generation in probabilistic model checking. *IEEE Trans. Softw. Eng.*, 35(2):241–257, 2009.

[HKMKS00] Holger Hermanns, Joost-Pieter Katoen, Joachim Meyer-Kayser, and Markus Siegle. A markov chain model checker. In *Proceedings of the* 6^{th} *International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 2000)*, pages 347–362. Springer, 2000.

[HN96] D. Harel and A. Naamad. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, October 1996.

[HWP+06] Marc Herbstritt, Ralf Wimmer, Thomas Peikenkamp, Eckard Böde, Michael Adelaide, Sven Johr, Holger Hermanns, and Bernd Becker. Analysis of Large Safety-Critical Systems: A quantitative Approach. Reports of SFB/TR 14 AVACS 8, SFB/TR 14 AVACS, Feb 2006. ISSN: 1860-9821, http://www.avacs.org.

[ILT+10] Takuto Ishimatsu, Nancy Leveson, John Thomas, Masa Katahira, Yuko Miyamoto, and Haruka Nakao. Modeling and hazard analysis using stpa. In *Proceedings of the Conference of the International Association for the Advancement of Space Safety*, 2010.

[Int98] International Electrotechnical Commission. Functional safety of electricsl/electronic/programmable electronic safety-related systems, 1998.

[Int06] International Electrotechnical Commission. IEC 61025: Fault Tree Analysis. Edition 2.0, 2006, 2006.

[Int09] International Organisation for Standardization. ISO 26262: Road vehicles-functional safety, Draft International Standard (DIS), 2009.

[JH05] Anjali Joshi and Mats P.E. Heimdahl. Model-Based Safety Analysis of Simulink Models Using SCADE Design Verifier. In *Proceedings of the* 24^{th} *International Conference on Computer Safety, Reliability and Security (SAFECOMP 2005)*, volume 3688 of *LNCS*, page 122, 2005.

[JH07] Anjali Joshi and Mats P. E. Heimdahl. Behavioral fault modeling for model-based safety analysis. In *Proceedings of the* 10^{th} *IEEE High Assurance Systems Engineering Symposium (HASE 2007)*, pages 199–208, Washington, DC, USA, 2007. IEEE Computer Society.

[JMWH05] Anjali Joshi, Steven P. Miller, Michael Whalen, and Mats P.E. Heimdahl. A Proposal for Model-Based Safety Analysis. In *Proceedings of the* 24^{th} *Digital Avionics Systems Conference (DASC 2005)*, Oct 2005.

[JWH05] Anjali Joshi, Mike Whalen, and Mats P.E. Heimdahl. Modelbased safety analysis: Final report. Technical Report CR-2006-213953, NASA Contractor Report, 2005.

[KC03] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[KKZ05] J.-P. Katoen, M. Khattri, and I. Zapreev. A markov reward model checker. In *Proceedings of the* 2^{nd} *International Conference on Quantitative Evaluation of Systems (QEST 2005)*, pages 243–245. IEEE Computer Society, 2005.

[KKZJ07] Joost-Pieter Katoen, Tim Kemna, Ivan S. Zapreev, and David N. Jansen. Bisimulation minimisation mostly speeds up probabilistic model checking. In *Proceedings of the* 13^{th} *International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 2007)*, pages 87–101, 2007.

[Kle86] T. A. Kletz. Hazop and HAZAN notes on the identification and assessment of hazards. Technical report, Inst. of Chemical Engineers, Rugby, England, 1986.

[KNP02a] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. In *Proceedings of the* 8^{th} *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002)*, volume 2280 of *LNCS*. Springer, 2002.

[KNP02b] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM: Probabilistic symbolic model checker. In T. Field, P. Harrison, J. Bradley, and U. Harder, editors, *Proceedings of the* 12^{th} *International Conference*

on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS 2002), volume 2324 of *LNCS*, pages 200–204. Springer, 2002.

[KZH+10] Joost-Pieter Katoen, Ivan S. Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation*, In Press, Corrected Proof:167–176, 2010.

[Lad99] Peter B. Ladkin. A quick introduction to why-because analysis, 1999. accessed: January 2011.

[Lad00] Peter Ladkin. Causal reasoning about aircraft accidents. In Floor Koornneef and Meine van der Meulen, editors, *Computer Safety, Reliability and Security*, volume 1943 of *LNCS*, pages 344–360. Springer, 2000.

[Lad01] Peter B. Ladkin. *Causal System Analysis*. Springer, 2001.

[Lad08] Peter B. Ladkin. An overview of IEC 61508 on E/E/PE functional safety, 2008.

[Lam77] L. Lamport. Proving the correctness of multiprocess programs. *Software Engineering, IEEE Transactions on*, SE-3(2):125 – 143, march 1977.

[Lap95] J. C. Laprie. Dependable computing: Concepts, limits, challenges. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing (FTCS 1995)*, 1995.

[Lev92] Nancy G. Leveson. High-pressure steam engines and computer software. In *Proceedings of the* 14^{th} *international conference on Software engineering (ICSE 1992)*, pages 2–14, New York, NY, USA, 1992. ACM.

[Lev03] Nancy G. Leveson. A new approach to hazard analysis for complex systems. In *Proceedings of the International Conference of the System Safety Society*, 2003.

[Lev04a] Nancy Leveson. A new accident model for engineering safer systems. *Safety Science*, 42(4):237 – 270, 2004.

[Lev04b] N.G. Leveson. A systems-theoretic approach to safety in software-intensive systems. *IEEE Transactions on Dependable and Secure Computing*, 1(1):66 – 86, 2004.

[Lev11] Nancy G. Leveson. *Engineering a Safer World*. MIT Press, 2011. to appear, draft available from http://sunnyday.mit.edu/safer-world/index.html (accessed May, 2011).

[Lew73] David Lewis. Causation. *Journal of Philosophy*, 70(17):556–567, 1973.

[Lio96] J. L. Lions. Ariane 5 – flight 501 failure. Technical Report 33, ESA, 1996. available at http://www.esa.int/export/esaCP/Pr_33_1996_p_EN.html.

[LL96] Gerard Le Lann. The Ariane 5 Flight 501 Failure - A Case Study in System Engineering for Computing Systems. Research Report RR-3079, INRIA, 1996. Projet REFLECS.

[LM06] O. Lisagor and J. A. McDermid. Towards a practicable process for automated safety analysis. In *Porceedings of the* 24^{th} *International System Safety Conference (ISSC 2006)*, 2006.

[LSK10] Oleg Lisagor, Linling Sun, and Tim Kelly. The illusion of method: Challenges of model-based safety assessment. In *Proceedings of the* 28^{th} *International System Safety Conference (ISSC 2010)*, 2010.

[LT93] N. G. Leveson and C. S. Turner. An investigation of the therac-25 accidents. *IEEE Computer*, 26(7), 1993.

[Mai00] Monika Maidl. The common fragment of ctl and ltl. In *In IEEE Symposium on Foundations of Computer Science (FOCS 2000)*, pages 643–652, 2000.

[Mai09] V. Maisonneuve. Automatic heuristic-based generation of MTBDD variable orderings for PRISM models. Internship report, ENS Cachan & Oxford University Computing Laboratory, 2009.

[McM90] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1990.

[Mer10] Guilliaume Merle. *Algebraic modelling of Dynamic Fault Trees, contribution to qualitative and quantitative analysis*. PhD thesis, LURPA, Ecole Normale Supérieure (ENS) de Cachan, 2010.

[MIL] MIL-STD 1629. *Failure mode and effects analysis*. U. S. Department of Defense.

[MMB96] Robin E. McDermott, Raymond J. Mikulak, and Michael R. Beauregard. *The Basics of FMEA*. Quality Resources, 1996.

[Moy05] Matthieu Moy. *Techniques and Tools for the Verification of Systems-on-a-Chip at the Transaction Level*. PhD thesis, INPG, Grenoble, France, December 2005.

[MRLB10] G. Merle, J.-M. Roussel, J.-J. Lesage, and A. Bobbio. Probabilistic algebraic analysis of fault trees with priority dynamic gates and repeated events. *IEEE Transactions on Reliability*, 59(1):250 –261, mar. 2010.

[MRLV10] G. Merle, J.-M. Roussel, J.-J. Lesage, and N. Vayatis. Analytical calculation of failure probabilities in dynamic fault trees including spare gates. In *Proceedings of the European Safety & Reliability Conference (ESREL 2010)*, 2010.

[MS04] C. Müller-Schloer. Organic computing: on the feasibility of controlled emergence. In *CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 2–5, New York, NY, USA, 2004. ACM.

[MSvW04] Christian Müller-Schloer, Christoph von der Malsburg, and Rolf P. Würtz. Organic computing. *Informatik Spektrum*, 27(4):332–336, August 2004.

[Nel08] Paul S. Nelson. A stamp analysis of the lex comair 5191 accident. Master's thesis, Lund University Sweden, 2008.

[Ngu10] Viet Yen Nguyen. Personal Communication, June 2010.

[NOS+09] F. Nafz, F. Ortmeier, H. Seebach, J.-P. Steghöfer, and W. Reif. A universal self-organization mechanism for role-based Organic Computing systems. In *Proceedings of the Sixth International Conference on Autonomic and Trusted Computing (ATC 2009)*, 2009.

[NPK10] Gethin Norman, Dave Parker, and Marta Kwiatkowska. The PRISM language - semantics, accessed: June 2010.

[NSS+10] Florian Nafz, Hella Seebach, Jan-Philipp Steghöfer, Simon Bäumler, and Wolfgang Reif. A formal framework for compositional verification of organic computing systems. In Bing Xie, Juergen Branke, S. Sadjadi, Daqing Zhang, and Xingshe Zhou, editors, *Proceedings of the 7^{th} Conference on Autonomic and Trusted Computing (ATC 2010)*, volume 6407 of *LNCS*, pages 17–31. Springer, 2010.

[OG10] Frank Ortmeier and Matthias Güdemann. Promosa - probabilistic models for safety analysis. In *Proceedings of 6. Dagstuhl-Workshop on Model-Based Development of Embedded Systems (MBEES 2010)*, 2010.

[OGR07] Frank Ortmeier, Matthias Güdemann, and Wolfgang Reif. Formal failure models. In *Proceedings of the 1^{st} IFAC Workshop on Dependable Control of Discrete Systems (DCDS 2007)*. Elsevier, 2007.

[OR04] F. Ortmeier and W. Reif. Safety optimization: A combination of fault tree analysis and optimization techniques. In *Proceedings of the Conference on Dependable Systems and Networks (DSN 2004)*, Florence, 2004. IEEE Computer Society.

[ORS05] F. Ortmeier, W. Reif, and G. Schellhorn. Formal safety analysis of a radio-based railroad crossing using deductive cause-consequence analysis (DCCA). In *Proceedings of 5th European Dependable Computing Conference (EDCC 2005)*, volume 3463 of *LNCS*. Springer, 2005.

[ORS06] F. Ortmeier, W. Reif, and G. Schellhorn. Deductive cause-consequence analysis (DCCA). In *Proceedings of the* 17^{th} *IFAC World Congress*. Elsevier, 2006.

[Ort06] Frank Ortmeier. *Formale Sicherheitsanalyse*. Logos Verlag Berlin, 2006.

[OS06] Frank Ortmeier and Gerhard Schellhorn. Formal Fault Tree Analysis - Practical Experiences. In *Proceedings of* 6^{th} *International Workshop On Automated Verification of Critical Systems (AVoCS 2006)*, 2006.

[OSR04] F. Ortmeier, G. Schellhorn, and W. Reif. Safety optimization of a radio-based railroad crossing. In E. Schnieder and G. Tarnai, editors, *Proceedings of Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS / FORMAT 2004)*, 2004.

[Par07] Terence Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Programmers. Pragmatic Bookshelf, 2007.

[PLH06] Steven J. Pereira, Grady Lee, and Jeffrey Howard. A system-theoretic hazard analysis methodology for a non-advocate safety assessment of the ballistic missile defense system. In *Proceedings of the AIAA Missile Sciences Conference*, 2006.

[PM91] Y. Papadopoulos and J. A. McDermid. Hierarchically performed hazard origin and propagation studies. In *Proceedings of the* 10^{th} *International Conference on Computer Safety, Reliability,and Security (SAFECOMP 1991)*, 1991.

[Pnu77] Amir Pnueli. The temporal logic of programs. *Foundations of Computer Science, Annual IEEE Symposium on*, pages 46–57, 1977.

[PP07] David Parker and Yannis Papadopoulos. Effective multi-criteria redundancy allocation via model-based safety analysis. In *Proceedings of the IFAC Workshop on Intelligent Manufacturing Systems*. Elsevier, 2007.

[PPM99] Alberto Pasquini, Yiannis Papadopoulos, and John McDermid. Hierarchically performed hazard origin and propagation studies. In Karama Kanoun, editor, *Computer Safety, Reliability and Security*, volume 1698 of *LNCS*, pages 688–688. Springer, 1999.

[PPRd10] B. Perrot, T. Prosvirnova, A. Rauzy, and J.-P. Sahut d'Izarn. Introduction au nouveau langage pour la sûreté de fonctionnement : Altarica nouvelle génération. In *Actes du congrès LambdaMu'17*, 2010.

[PS05] Thilo Paul-Stüve. Performing a why-because analysis. Technical report, University of Bielefeld, 2005.

[PWP+10] Yiannis Papadopoulos, Martin Walker, David Parker, Erich Rüde, Rainer Hamann, Andreas Uhlig, Uwe Grätz, and Rune Lie. Engineering failure analysis and design optimisation with HiP-HOPS. *Engineering Failure Analysis*, 2010.

[rBB+06] O. Åkerlund, P. Bieber, E. Boede, M. Bozzano, M. Bretschneider, C. Castel, A. Cavallo, M. Cifaldi, J. Gauthier, A. Griffault, O. Lisagor, A. Luedtke, S. Metge, C. Papadopoulos, T. Peikenkamp, L. Sagaspe, C. Seguin, H. Trivedi, and L. Valacca. ISAAC, a framework for integrated safety analysis of functional, geometrical and human aspects. In *Proceedings of 2nd European Congress on Embedded Real Time Software (ERTS 2006)*, 2006.

[RST00] W. Reif, G. Schellhorn, and A. Thums. Safety analysis of a radio-based crossing control system using formal methods. In E. Schnieder and U. Becker, editors, *Proceedings of the 9^{th} IFAC Symposium Control in Transportation Systems (CTS 2000)*, June 2000.

[RTC92] RTCA. DO-178B: Software considerations in airborne systems and equipment certification, December, 1st 1992.

[SA97] R. M. Sinnamon and J. D. Andrews. Improved efficiency in qualitative fault tree analysis. *Quality and Reliability Engineering Internationa*, 13:293–298, 1997.

[SA04] SAE-AS5506. Architecture analysis and design language (AADL), 2004.

[SA06] SAE-AS5506/1. Architecture analysis and design language (AADL) Annex E: Error Model Annex, 2006.

[SOR07] H. Seebach, F. Ortmeier, and W. Reif. Design and Construction of Organic Computing Systems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007)*. IEEE Computer Society Press, 2007.

[STR02] G. Schellhorn, A. Thums, and W. Reif. Formal fault tree semantics. In *Proceedings of The Sixth World Conference on Integrated Design & Process Technology (IDPT 2002)*, Pasadena, CA, 2002.

[Sys08] SysML 1.1 specification, 2008.

[Te95] Tucker S. Taft and Robert A. Duff (eds.). Ada 95 reference manual, language and standard libraries, international standard iso/iec 8652:1995(e), 1995.

[TIS11] Florent Teichteil, Guillaume Infantes, and Christel Seguin. Epoch probabilistic model-checking. Talk at CISEC Model Based Safety Assessment Workshop (MBSAW 2011), 2011. accessed April 2011.

[TO03] A. Thums and F. Ortmeier. Formal safety analysis in transportation control. In E. Schnieder, editor, *International Workshop on Software Specification of Safety Relevant Transportation Control Tasks*, volume 12 (no. 535) of *VDI Fortschritt-Berichte*. VDI Verlag GmbH, 2003.

[TOWS04] A. Thums, F. Ortmeier, W.Reif, and G. Schellhorn. Interactive verification of statecharts. In H. Ehrig, editor, *Integration of Software Specification Techniques for Applications in Engineering (INT 2004)*, pages 355 – 373. Springer LNCS 3147, 2004.

[TS02] A. Thums and G. Schellhorn. Formal safety analysis in transportation control. In R. Slovak and E. Schnieder, editors, *Proceedings of the Workshop on Software Specification of Safety Relevant Transportation Control Tasks*, Braunschweig, Germany, 2002. VDI-Verlag.

[VDF+02] Dr. W. Vesley, Dr. Joanne Dugan, J. Fragole, J. Minarik II, and J. Railsback. *Fault Tree Handbook with Aerospace Applications*. NASA Office of Safety and Mission Assurance, August 2002.

[vM02] M. Čepin and B. Mavko. A dynamic fault tree. *Reliability Engineering & System Safety*, 75, 2002.

[VPF+06] François Vernadat, Christian Percebois, Patrick Farail, Robertus Vingerhoeds, Alain Rossignol, Jean P. Talpin, and David Chemouil. The TOPCASED project - a toolkit in open-source for critical applications and system development. In *Data Systems In Aerospace (DASIA)*. European Space Agency (ESA Publications), 2006.

[Wal05] Malcolm Wallace. Modular architectural representation and analysis of fault propagation and transformation. *Electronic Notes in Theoretical Computer Science*, 141(3):53 – 71, 2005. Proceedings of the Second International Workshop on Formal Foundations of Embedded Software and Component-based Software Architectures (FESCA 2005).

[WBP07] Martin Walker, Leonardo Bottaci, and Yiannis Papadopoulos. Compositional Temporal Fault Tree Analysis. In *Proceedings of the* 26^{th} *International Conference on Computer Safety, Reliability and Security (SAFECOMP 2007)*, pages 106–119, 2007.

[WHH+06] Ralf Wimmer, Marc Herbstritt, Holger Hermanns, Kelley Strampp, and Bernd Becker. Sigref – A Symbolic Bisimulation Tool Box. In Susanne Graf and Wenhui Zhang, editors, *Proceedings of the* 4^{th} *International Symposium on Automated Technology for Verification and Analysis (ATVA 2006)*, volume 4218 of *LNCS*, pages 477–492. Springer, 2006.

[WP07] M. Walker and Y. Papadopoulos. Pandora 2: The time of priority-OR gates. In *Proceedings of the* 1^{st} *IFAC workshop on Dependable Control of Discrete Systems (DCDS 2007)*. Elsevier, 2007.

[WWGP10] Ian Wolforth, Martin Walker, Lars Grunske, and Yiannis Papadopoulos. Generalizable safety annotations for specification of failure patterns. *Software: Practice and Experience*, 40:453–483, 2010.

[You05] Håkan L. S. Younes. S.: Ymer: A statistical model checker. In *Proceedings of the International Conference on Computer Aided Verification (CAV 2005)*. Springer, 2005.

[ZSR+10] Lijun Zhang, Zhikun She, Stefan Ratschan, Holger Hermanns, and Ernst Hahn. Safety verification for probabilistic hybrid systems. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Proceedings of the International Conference on Computer Aided Verification (CAV 2010)*, volume 6174 of *LNCS*, pages 196–211. Springer, 2010.