Architectural Modeling and Analysis for Safety Engineering

A THESIS TOPIC PROPOSAL
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Danielle Stewart

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Mats P. E. Heimdahl

May, 2019

Acknowledgements

Abstract

Model-based development tools are increasingly being used for system-level development of safety-critical systems. Architectural and behavioral models provide important information that can be leveraged to improve the system safety analysis process. Model-based design artifacts produced in early stage development activities can be used to perform system safety analysis, reducing costs and providing accurate results throughout the system life-cycle.

As critical systems become more dependent on software components, analysis regarding fault propagation through these software components becomes more important. The methods used to perform these analyses require understandability from the side of the analyst, scalability in terms of system size, and mathematical correctness in order to provide sufficient proof that a system is safe. Determination of the events that can cause failures to propagate through a system as well as the effects of these propagations can be a time consuming and error prone process. In this research, we describe a technique for determining these events with the use of Inductive Validity Cores (IVCs) and producing compositionally derived artifacts that encode pertinant system safety information.

Contents

List of Tables

List of Figures

Author Declaration

Some of the material presented within has previously been published in the following papers:

- D. Stewart, J. Liu, M. Heimdahl, M. Whalen, D. Cofer, and M. Peterson. The Safety Annex for Architecture Analysis and Design Language. In *10th Edition European Congress Embedded Real Time Systems*, to appear 2020
- D. Stewart, J. Liu, M. Heimdahl, M. Whalen, D. Cofer, and M. Peterson. Architectural modeling and analysis for safety engineering (AMASE), NASA final report. `https://github.com/loonwerks/AMASE/tree/master/doc/AMASE_Final_Report_2019`, 2019
- D. Stewart, J. Liu, M. Whalen, D. Cofer, and M. Peterson. Safety Annex for Architecture Analysis Design and Analysis Language. Technical Report 18-007, University of Minnesota, March 2018
- D. Stewart, M. Whalen, D. Cofer, and M. P. Heimdahl. Architectural Modeling and Analysis for Safety Engineering. In *IMBSA 2017*, pages 97–111, 2017

All the work contained within represents the original contribution of the author.

Chapter 1

Introduction

System safety analysis is crucial in the development life cycle of critical systems to ensure adequate safety as well as demonstrate compliance with applicable standards. A prerequisite for any safety analysis is a thorough understanding of the system architecture and the behavior of its components; safety engineers use this understanding to explore the system behavior to ensure safe operation, assess the effect of failures on the overall safety objectives, and construct the accompanying safety analysis artifacts. Developing adequate understanding, especially for software components, is a difficult and time consuming endeavor. Given the increase in model-based development in critical systems [8, 26, 29, 33, 37], leveraging the resultant models in the safety analysis process holds great promise in terms of analysis accuracy as well as efficiency.

In this report we describe the *Safety Annex* for the system engineering language AADL (Architecture Analysis and Design Language), a SAE Standard modeling language for Model-Based Systems Engineering (MBSE) [1]. The Safety Annex allows an analyst to model the failure modes of components and then "weave" these failure modes together with the original models developed as part of MBSE. The safety analyst can then leverage the merged behavioral models to propagate errors through the system to investigate their effect on the safety requirements. Determining how errors propagate through software components is currently a costly and time-consuming element of the safety analysis process.

The use of behavioral contracts to capture the error propagation characteristics of software component without the need to add separate propagation specifications (*implicit* error propagation) is a significant benefit for safety analysts. In addition, the annex allows modeling of dependent faults that are not captured through the behavioral models (*explicit* error propagation), for example, the effect of a single electrical failure on multiple software components or the effect hardware failure (e.g., an explosion) on multiple behaviorally unrelated components. Furthermore, the tool enables engineers to investigate the correctness of the nominal system behavior (where no failures have occurred) as well as the system's resilience to component failures.

rework this paragraph In related work in model checking, the *compositional* approach has been shown to provide greater scalability in large scale system models. There are numerous tools that can compute *Minimal Cut Sets*, or the minimal cause of a *top level event* (failure of a safety property), but none are able to do so compositionally. What is the point of this paragraph??

1.1 Objectives and Significance

In this project, we are specifically concerned with collecting proof information from the model checker and leveraging this in order to capture system safety information and artifacts. Previous research has provided a way to find all sets of minimal model elements necessary for the proof of a property and we attempt to use these to show vital information about possible modes of failure of a system and ways that active faults in the system can cause the violation of a safety property. *The objective of this dissertation* is to use the elements required for the proof of a property in order to compositionally generate all minimal cut sets for a safety property. The minimal cut sets can then be used to generate commonly used safety analysis artifacts such as Fault Tree Analysis (FTA) or Failure Modes and Effect Analysis (FMEA) tables, but can also be used to calculate the probability of the violation of the safety property.

While other available tools provide ways to generate minimal cut sets, what we propose is new in the following ways. Due to the implementation of the Safety Annex which utilizes behavioral mechanisms in AGREE (Assume Guarantee Reasoning Environment), the faults can be behaviorally propagated or explicitly propagated. Behavioral propagation allows for discovery of possibly unforeseen effects of failed components. The safety engineer is no longer required to determine how an error will propagate through a complex system, but can instead look at counterexamples and proof traces to see how the system is effected by failures of its components. On the other hand, this does not allow for all possible failure modes. Explicit propagation can describe dependencies between faults such as co-located components or other more specific hardware faults. Given these two possible modes of propagation, the fault model can provide a rich description of the system in question. What is more, other tools calculate minimal cut sets using *monolithic* analysis. This flattens a hierarchical model and utilizes all elements of the model to provide proof of a property. It has been shown that a *compositional* approach to proof is far more scalable due to viewing the model one layer at a time.

The approach we propose takes advantage of these two aspects: the ability of combined behavioral and explicit error propagation within the model and compositional fault analysis.

What we propose is a generic and efficient mechanism for extracting all minimal sets of model elements required for the proof of the safety property and then transforming them into faults that describe the basic events which cause the violation of the property of interest. Once these minimal cut sets are generated, probabilistic computation can proceed across them to provide the likelihood of this property violation.

Compositional generation of minimal cut sets facilitates several useful system and safety engineering tasks. Specifically, it is useful to see how a component failure can effect the overall system behavior and which combinations of failures will violate safety properties. These minimal cut sets provide formal and human-understandable artifacts that can be used in the safety assessment process. Such information is valuable in analyzing safety critical systems and can be used for many purposes in the safety assessment process, such as:

these may need to be changed... just throwing something out there.

Fault Tree Analysis: The traditional safety assessment process at the system level is based on ARP4754A and ARP4761 (cite). After the system is examined and potential functional failures are found and classified, the next step is the Preliminary System Safety Assessment (PSSA) which is updated throughout the system development process. A key element of the PSSA is a system level FTA. The FTA is a deductive failure analysis to determine the causes of a specific undesired event in a top-down fashion. For an FTA, a safety analyst begins with a failure condition and systematically examines the system design to determine all credible faults and failure combinations that could cause the undesired event. By using a model of the system, a fault model of possible failure modes, behavioral and explicit propagation, and automated generation of these failure combinations, the analyst is provided with an efficient means of understanding the system behavior and failures and can easily iterate through this process multiple times throughout system development.

Single Points of Failure: Requirements can often be stated such that no single point of failure can violate the property. In this case, a scalable automated approach can provide insight that may be overlooked by an analyst. By using proof results, the behavior of the system can be examined when this single fault is active and system changes or development can be realized in order to mitigate these failures.

Probability of Violation: The Safety Annex currently provides a way to calculate the combinations of probabilities associated with the faults and test to see if this combination is under a given threshold, but it is also valuable to know what the top level threshold of the system is given the faults and their probabilities. During the fault analysis, this can be calculated while the minimal cut sets are being collected which provides a scalable and efficient means of gathering this information.

1.2 Use in Research and System Development

The traditional safety assessment process at the system level is based on ARP4754A [47] and ARP4761 [46]. It starts with the System level Functional Hazard Assessment (SFHA) examining the functions of the system to identify potential functional failures and classifies the potential hazards associated with them.

The next step is the Preliminary System Safety Assessment (PSSA), updated throughout the system development process. A key element of the PSSA is a system level Fault Tree Analysis (FTA). The FTA is a deductive failure analysis to determine the causes of a specific undesired event in a top-down fashion. For an FTA, a safety analyst begins with a failure condition from the SFHA, and systematically examines the system design (e.g., signal flow diagrams provided by system engineers) to determine all credible faults and failure combinations that could cause the undesired event.

The lack of precise models of the system architecture and its failure modes often forces safety analysts to devote significant effort to gathering architectural details about the system behavior from multiple sources. Furthermore, this investigation typically stops at system level, leaving software function details largely unexplored.

Typically equipped with the domain knowledge about the system, but not detailed knowledge of how the software applications are designed, practicing safety engineers find it a time consuming and involved process to acquire the knowledge about the behavior of the software applications hosted in a system and its impact on the overall system behavior. Industry practitioners have come to realize the benefits of using models in the safety assessment process, and a revision of the ARP4761 to include Model Based Safety Analysis (MBSA) is under way.

We propose a model-based safety assessment process backed by formal methods to help safety engineers with early detection of the design issues. This process uses a single unified model to support both system design and safety analysis. It is based on the following steps:

1. System engineers capture the critical information in a shared AADL/AGREE model: high-level hardware and software architecture, nominal behavior at the component level, and safety requirements at the system level.

2. System engineers use the backend model checker to check that the safety requirements are satisfied by the nominal design model.

3. Safety engineers use the Safety Annex to augment the nominal model with the component failure modes. In addition, safety engineers specify the fault hypothesis for the analysis which corresponds to how many simultaneous faults the system must be able to tolerate.

4. Safety engineers use the backend model checker to analyze if the safety requirements and fault tolerance objectives are satisfied by the design in the presence of faults. If the design does not tolerate the specified number of faults (or probability threshold of fault occurrence), then the tool produces counterexamples leading to safety requirement violation in the presence of faults, as well as all minimal set of fault combinations that can cause the safety requirement to be violated.

5. The safety engineers examine the results to assess the validity of the fault combinations and the fault tolerance level of the system design. If a design change is warranted, the model will be updated with the latest design change and the above process is repeated.

1.3 Intended Contributions

We claim that compositionally generated minimal cut sets have potential system safety engineering uses in several phases of the development cycle. However, efficient and effective generation strategies must be proposed to achieve these benefits. The anticipated contributions of the work are therefore as follows:

- *Behavioral fault propagation through the use of the Safety Annex built on the AGREE model checking capabilities:* This provides a model-based environment that contains the system model in AADL, the behavioral model in AGREE, and the fault model in the Safety Annex. When a fault is activated, a trace of the system is provided through the artifacts generated by a model checker (JKind) and the propagation does not have to be explicitly defined.

- *Explicit fault propagation through the use of the Safety Annex built on the AGREE model checking capabilities:* Allowing for explicit propagation and fault dependencies provides a richer fault model and catches those cases that may be impossible to describe using only behavioral propagation, e.g. co-location of components or hardware failures.

- *Collecting proof information to compositionally generate minimal cut sets:* The thesis will provide a formal way of generating all minimal cut sets through the transformation of all *Minimal Inductive Validity Cores* (MIVCs). This will then be implemented in the Safety Annex tool.

- *Calculating the probabilistic threshold of a safety property given the minimal cut sets:* The thesis will also provide the algorithm used in this calculation and a formal approach to this calculation.

1.4 Evaluation

We plan to evaluate the approach on a large scale aerospace example and evaluate the overhead of the minimal cut set transformation compared to MIVC computation. Evaluation with respect to other similar tools might prove to be difficult since these tools do not perform compositional MinCutSet generation and many only perform explicit *or* behavioral propagation and not both. The baseline system model is also in a different modeling language; whereas we use AADL, related tools do not. One option is to compare EMV2 explicit propagation and FT generation to our approach, but I am not sure why this would be useful, especially since we have been careful to distinguish what we do with EMV2. All of that to say, I am unsure how to write this section.

Therefore, we investigate the following research questions:

- **RQ 1:** Does the mix of behavioral and explicit error propagation provide more information on the state of the system model and the possible modes of failure than just one type of error propagation?

- **RQ 2:** Are the algorithms used to transform MIVCs into MinCutSets scalable and efficient?

- **RQ 3:** What is the time difference between calculating if a fault combination exceeds a given threshold and what the safety property threshold actually is?
- **RQ 4:** Can these MinCutSets provide useful information about the system and its modes of failure that can be used in the certification process?

Upon completion of the proposed research, the transformation algorithms and the probabilsitic computations will be implemented in the Safety Annex. The implementation will be benchmarked and evaluated rigorously. The usefulness of the compositional MinCutSet idea will be shown by utilizing its applications into different projects.

1.5 Chapters and Organization of the Proposal

This proposal is organized in three chapters. Chapter 2 broadly discusses related work, the tools and modeling language used in this project, and some useful formal definitions. Chapter 3 describes the proposed approach and outlines the contributions of this dissertation. Lastly, the conclusion summarizes the approach of the project.

Chapter 2

Background

2.1 Related Work

The related work has two main focuses. The first is in regard to safety analysis tools and research and how the Safety Annex differs from related approaches. The second outlines related work in minimal cut set generation, probabilistic computations over fault trees, and tools that implement this research.

Safety Analysis Tools and Research: A model-based approach for safety analysis was proposed by Joshi et. al in [31–33]. In this approach, a safety analysis system model (SASM) is the central artifact in the safety analysis process, and traditional safety analysis artifacts, such as fault trees, are automatically generated by tools that analyze the SASM.

The contents and structure of the SASM differ significantly across different conceptions of MBSA. We can draw distinctions between approaches along several different axes. The first is whether they propagate faults explicitly through user-defined propagations, which we call *failure logic modeling* (FLM) or through existing behavioral modeling, which we call *failure effect modeling* (FEM). The next is whether models and notations are *purpose-built* for safety analysis vs. those that extend *existing system models* (ESM).

For FEM approaches, there are several additional dimensions. One dimension involves whether *causal* or *non-causal* models are allowed. Non-causal models allow simultaneous (in time) bi-directional error propagations, which allow more natural expression of some failure types (e.g. reverse flow within segments of a pipe), but are more difficult to analyze. A final dimension involves whether analysis is *compositional* across layers of hierarchically-composed systems or *monolithic*. Our approach is an extension of AADL (ESM), causal, compositional, mixed FLM/FEM approach.

Tools such as the AADL Error Model Annex, Version 2 (EMV2) [21] and HiP-HOPS for EAST-ADL [15] are *FLM*-based *ESM* approaches. As previously discussed, given many possible faults, these propagation relationships require substantial user effort and become more complex. In addition, it becomes the analyst's responsibility to determine whether faults can propagate; missing propagations lead to unsound analyses. In our Safety Annex, propagations occur through system behaviors (defined by the nominal contracts) with no additional user effort.

Closely related to our work is the model-based safety assessment toolset called COMPASS (Correctness, Modeling project and Performance of Aerospace Systems) [9]. COMPASS is a mixed *FLM/FEM*-based, *causal compositional* tool suite that uses the SLIM language, which is based on a subset of AADL, for its input models [10, 13]. In SLIM, a nominal system model and the error model are developed separately and then transformed into an extended system model. This extended model is automatically translated into input models for the NuSMV model checker [16, 40], MRMC (Markov Reward Model Checker) [34, 38], and RAT (Requirements Analysis Tool) [43]. The safety analysis tool xSAP [6] can be invoked in order to generate safety analysis artifacts such as fault trees and FMEA tables [7]. COMPASS is an impressive tool suite, but some of the features that make AADL suitable for SW/HW architecture specification: event and event-data ports, threads, and processes, appear to be missing, which means that the SLIM language may not be suitable as a general system design notation (ESM).

SmartIFlow [30] is a *FEM*-based, *purpose-built*, *monolithic non-causal* safety analysis tool that describes components and their interactions using finite state machines and events. Verification is done through an explicit state model checker which returns sets of counterexamples for safety requirements in the presence of failures. SmartIFlow allows *non-causal* models containing simultaneous (in time) bi-directional error propagations. On the other hand, the tools do not yet appear to scale to industrial-sized problems, as mentioned by the authors [30]: "As current experience is based on models with limited size, there is still a long way to go to make this approach ready for application in an industrial context".

The Safety Analysis and Modeling Language (SAML) [26] is a *FEM*-based, *purpose-built*, *monolithic causal* safety analysis language. System models constructed in SAML can be used used for both qualitative and quantitative analyses. It allows for the combination of discrete probability distributions and non-determinism. The SAML model can be automatically imported into several analysis tools like NuSMV [16], PRISM (Probabilistic Symbolic Model Checker) [35], or the MRMC probabilistic model checker [34].

AltaRica [5, 42] is a *FEM*-based, *purpose-built*, *monolithic* safety analysis language with several dialects. There is one dialect of AltaRica which use dataflow (*causal*) semantics, while the most recent language update (AltaRica 3.0) uses non-causal semantics. The dataflow dialect has substantial tool support, including the commercial Cecilia OCAS tool from Dassault [4]. For this dialect the Safety assessment, fault tree generation, and functional verification can be performed with the aid of NuSMV model checking [11]. Failure states are defined throughout the system and flow variables are updated through the use of assertions [3]. AltaRica 3.0 has support for simulation and Markov model generation through the OpenAltaRica (www.openaltarica.fr) tool suite.

Formal verification tools based on model checking have been used to automate the generation of safety artifacts [6, 11, 14]. This approach has limitations in terms of scalability and readability of the fault trees generated. Work has been done towards mitigating these limitations by the scalable generation of readable fault trees [12].

Minimal Cut Set Generation and Probabilistic Evaluations:

2.2 Tools and Modeling Language

2.2.1 Architecture Analysis and Design Language

We are using the Architectural Analysis and Design Language (AADL) to construct system architecture models. AADL is an SAE International standard that defines a language and provides a unifying framework for describing the system architecture for "performance-critical, embedded, real-time systems" [1, 20]. From its conception, AADL has been designed for the design and construction of avionics systems. Rather than being merely descriptive, AADL models can be made specific enough to support system-level code generation. Thus, results from analyses conducted, including the new safety analysis proposed here, correspond to the system that will be built from the model.

An AADL model describes a system in terms of a hierarchy of components and their interconnections, where each component can either represent a logical entity (e.g., application software functions, data) or a physical entity (e.g., buses, processors). An AADL model can be extended with language annexes to provide a richer set of modeling elements for various system design and analysis needs (e.g., performance-related characteristics, configuration settings, dynamic behaviors). The language definition is sufficiently rigorous to support formal analysis tools that allow for early phase error/fault detection.

2.2.2 Assume Guarantee Reasoning Environment

The Assume Guarantee Reasoning Environment (AGREE) is a tool for formal analysis of behaviors in AADL models [17]. It is implemented as an AADL annex and annotates AADL components with formal behavioral contracts. Each component's contracts can include assumptions and guarantees about the component's inputs and outputs respectively, as well as predicates describing how the state of the component evolves over time.

AGREE translates an AADL model and the behavioral contracts into Lustre [27] and then queries a user-selected model checker to conduct the back-end analysis. The analysis can be performed compositionally or monolithically.

Monolithic vs. Compositional Analysis: Compositional analysis of systems was introduced in order to address the scalability of model checking large software systems [17, 28, 41]. Monolithic verification and compositional verification are two ways that mathematical verification of component properties can be performed. In monolithic analysis, the model is flattened and the top level properties are proved using only the leaf level contracts of the components. The analysis can alternatively be performed compositionally following the architecture hierarchy such that analysis at a higher level is based on the components at the next lower level. The idea is to partition the formal analysis of a system architecture into verification tasks that correspond into the decomposition of the architecture. A component contract is an assume-guarantee pair. Intuitively, the meaning of a pair is: if the assumption is true, then the component will ensure that the guarantee is true. The components of a system are organized hierarchically and each layer of the architecture is viewed a system. For any given layer, the proof consists of demonstrating that the system guarantee is provable given the guarantees of its direct subcomponents and the system assumptions. This proof is performed one layer at a time starting from the top level of the system. When compared to monolithic analysis (i.e., analysis of the flattened model composed of all components), the compositional approach allows the analysis to scale to much larger systems [17].

2.2.3 Safety Annex for AADL

The Safety Annex for AADL is a tool that provides the ability to reason about faults and faulty component behaviors in AADL models and has been developed throughout the course of this project [48–51]. In the Safety Annex approach, formal assume-guarantee contracts are used to define the nominal behavior of system components. The nominal model is verified using AGREE. The Safety Annex weaves faults into the nominal model and analyzes the behavior of the system in the presence of faults. The tool supports behavioral specification of faults and their implicit propagation through behavioral relationships in the model as well as provides support to capture binding relationships between hardware and software compönents of the system.

2.3 Inductive Validity Cores and Formal Definitions

2.3.1 Inductive Validity Cores

Given a complex model, it is useful to extract traceability information related to the proof; in other words, which elements of the model were necessary to construct the proof. An algorithm was introduced by Ghassabani, et. al. to provide Inductive Validity Cores (IVCs) as a way to determine which model elements are necessary for the inductive proofs of the safety properties for sequential systems [24]. Given a safety property of the system, a model checker is invoked to construct a proof of the property. The IVC generation algorithm extracts traceability information from the proof process and returns a minimal set of the model elements required in order to prove the property. Later research extended this algorithm in order to produce all minimal IVC elements (all MIVCs) [2, 25].

The MIVC algorithm considers a constraint system consisting of the assumptions and contracts of system components and the negation of the safety property of interest (i.e. the top level event). It then collects all Minimal Unsatisfiable Subsets (MUSs) of this constraint system; these are the minimal explanations of the constraint systems infeasibility in terms of the *negation* of the safety property. Equivalently, these are the minimal model elements necessary to proof the safety property.

2.3.2 Related Definitions

A constraint system is an ordered set of abstract constraints over a set of variables. These constraints restrict the allowed assignments of these variables in some way [36]. In the case of a nominal model augmented with faults, a constraint system is formally defined as follows. Let F be the set of all fault activation literals defined in the model and G be the set of all component contracts (guarantees).

Definition 1. *A constraint system $C = \{C_1, C_2, ..., C_n\}$ where for $i \in \{1, ..., n\}$, C_i has the following constraints for any $f_j \in F$ and $g_k \in G$ with regard to the top level property P:*

$$C_i \in \begin{cases} f_j : & inactive \\ g_k : & true \\ P : & false \end{cases}$$

Given a satisfiable constraint system with regard to a safety property P, it is possible to find the minimal sets of model elements necessary for satisfying P through the use of the all MIVC algorithms [2, 25]. The algorithm collects all minimal unsatisfiable subsets of a given transition system in terms of the negation of the top level property. The minimal unsatisfiable subsets consist of component contracts constrained to *true*. When the constraints on these model elements are removed from the constraint system C, this results in an UNSAT system. This can be seen as the minimal explanation of the constraint systems infeasibility. Thus, this algorithm provides all model elements required for the proof of the safety property.

We utilize this algorithm by providing not only component contracts (constrained to *true*) as model elements, but also fault activation literals constrained to *false*. Thus the resulting MIVCs will contain the required contracts and constrained fault activation literals in order to prove the safety property.

Definitions 2-4 are taken from research by Liffiton et. al. [36].

Definition 2. *: A Minimal Unsatisfiable Subset (MUS) of a constraint system C is a subset of C such that MUS is unsatisfiable and $\forall c \in MUS : MUS \setminus \{c\}$ is satisfiable.*

An MUS is the minimal explaination of the constraint systems infeasability. A closely related set is a *Minimal Correction Set* (MCS). The MCSs describe the minimal set of model elements for which if constraints are removed, the constraint system is satisfied. For constraint system C defined above, this corresponds to which faults are not constrained to inactive (hence active) and violated contracts which lead to the violation of the safety property. In other words, the minimal set of active faults and/or violated properties that lead to the top level event.

Definition 3. *: A Minimal Correction Set (MCS) of a constraint system C is a subset of C such that $C \setminus MCS$ is satisfiable and $\forall S \subset MCS : C \setminus S$ is unsatisfiable.*

A MCS can be seen to "correct" the infeasability of the constraint system. A duality exists between MUSs of a constraint system and MCSs as established by Reiter [44]. This duality is defined in terms of *Minimal Hitting Sets* (MHS). A hitting set of a collection of sets A is a set H such that every set in A is "hit" be H; H contains at least one element from every set in A [36]. Every MUS of a constraint system is a minimal hitting set of the system's MCSs, and likewise every MCS is a minimal hitting set of the system's MUSs [18, 36, 44].

Definition 4. *: Given a collection of sets K, a hitting set for K is a set $H \subseteq \cup_{S \in K} S$ such that $H \cap S \neq \emptyset$ for each $S \in K$. A hitting set for K is minimal if and only if no proper subset of it is a hitting set for K.*

Utilizing this approach, the all MIVC algorithm produces all MUSs [25] and a minimal hitting set algorithm developed by Murakami et. al. takes these MUSs and from them, generates MCSs [23, 39].

Since we are interested in sets of faults that when active cause violation of the safety property, we turn our attention to Minimal Cut Sets. A *Minimal Cut Set* (MinCutSet) is a minimal collection of faults that lead to the violation of the safety property (or in other words, lead to the top level event in the fault tree). Furthermore, any subset of a MinCutSet will not cause this property violation. We define a minimal cut set consistently with much of the research in this field [19, 45]

Chapter 3

Proposed Approach

The contributions of this project can be seen as two main categories of research work. The first set was accomplished in the beginning phase of this project: behavioral and explicit error propagation through the implemenation of the Safety Annex for AADL. The remaining pieces of this research provide the bulk of the contribution and consist of the compositional generation of minimal cut sets through the transformation of inductive validity cores and using the fault tree generated by this transformation to compute the probability of a safety property violation.

3.1 Behavioral and Explicit Error Propagation

The usage of the terms error, failure, and fault are defined in ARP4754A and are described here for ease of understanding [47]. An *error* is a mistake made in implementation, design, or requirements. A *fault* is the manifestation of an error and a *failure* is an event that occurs when the delivered service of a system deviates from correct behavior. If a fault is activated under the right circumstances, that fault can lead to a failure. The term *error propagation* is used to refer to the propagation of the corrupted state caused by an active fault.

The Safety Annex is used to add possible faulty behaviors to a component model. Within the AADL component instance model, an annex is added which contain the fault definitions for the given component. The flexibility of the fault definitions allows the user to define numerous types of fault *nodes* by utilizing the AGREE node syntax. Examples of such faults include valves being stuck open or closed, output of a software component being nondeterministic, or power being cut off.

When a fault is activated by its specified triggering conditions, it modifies the output of the component. This faulty behavior may violate the contracts of other components in the system, including assumptions of downstream components. The impact of a fault is computed by the AGREE model checker when the safety analysis is run on the fault model; the error propagation is not explicitly defined as in some closely related tools [7, 21].

On the other hand, failures in hardware (HW) components can trigger behavioral faults in the system components that depend on them. This makes it beneficial to allow for explicit error propagation and the definition of dependencies in the fault model. For example, a CPU failure may trigger faulty behavior in the threads bound to that CPU. In addition, a failure in one HW component may trigger failure in other HW components located nearby, such as overheating, fire, or explosion in the containment location. The Safety Annex provides the capability to explicitly model the impact of hardware failures on other faults, behavioral or non behavioral.

Users specify dependencies between the HW component faults and faults that are defined in other components, either HW or SW. The hardware fault then acts as a trigger for dependent faults. This allows a simple propagation from the faulty HW component to the SW components that rely on it, affecting the behavior on the outputs of the affected SW components. Within the implementation, this corresponds to a statement linking the active fault to all dependent faults. Thus, if the triggering fault is active, so are all dependencies.

3.1.1 Implementation

The Safety Annex is written in Java as a plug-in for the OSATE AADL toolset, which is built on Eclipse. It is not designed as a stand-alone extension of the language, but works with behavioral contracts specified using the AGREE AADL annex [17]. The architecture of the Safety Annex is shown in Figure 3.1.

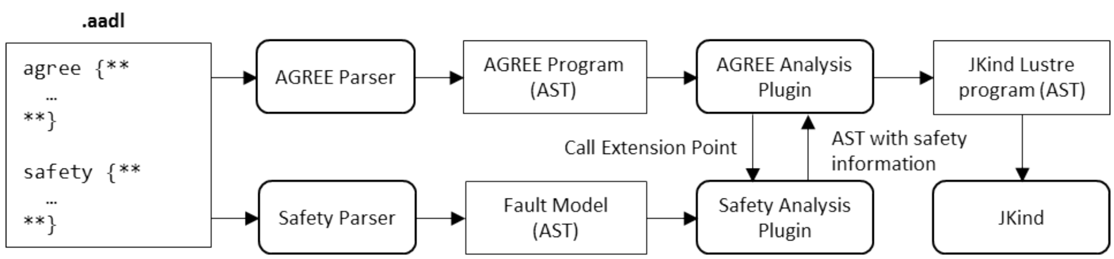

Figure 3.1: Safety Annex Plug-in Architecture

AGREE contracts are used to define the nominal behaviors of system components as *guarantees* that hold when *assumptions* about the values the component's environment are met. When an AADL model is annotated with AGREE contracts and the fault model is created using the Safety Annex, the model is transformed through AGREE into a Lustre model [27] containing the behavioral extensions defined in the AGREE contracts for each system component.

When performing fault analysis, the Safety Annex extends the AGREE contracts to allow faults to modify the behavior of component inputs and outputs. An example of a portion of an initial AGREE node and its extended contract is shown in Figure 3.2. The left column of the figure shows the nominal Lustre pump definition is shown with an AGREE contract on the output; and the right column shows the additional local variables for the fault (boxes 1 and 2), the assertion binding the fault value to the nominal value (boxes 3 and 4), and the fault node definition (box 5). Once augmented with fault information, the AGREE model (translated into the Lustre dataflow language [27]) follows the standard translation path to the model checker JKind [22], an infinite-state model checker for safety properties.

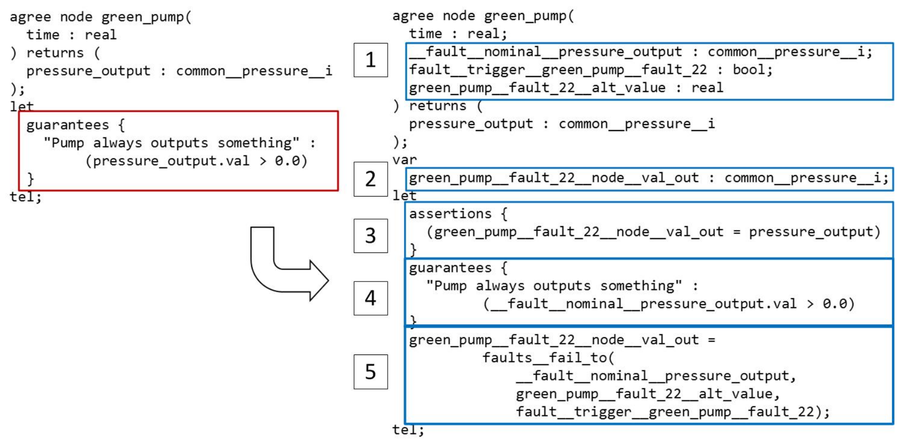

Figure 3.2: Nominal AGREE Node and Extension with Faults

3.2 Generation of Minimal Cut Sets from MIVCs

As was previously explained, the MIVCs are MUSs of a constraint system. The MCSs (Minimal Correction Sets) can be obtained through the use of a hitting set algorithm. The MCSs can be seen as a correction of the infeasability of the constraint system. Recall that the constraint system consists ofsubcomponent contracts, or guarantees, constrained to true, fault activation literals constrained to false, and the *negation* of the safety property. Under the assumption that the nominal system is satisfiable (when no faults are active, the property proves), it is no surprise that the constraint system framed this way is unsatisfiable.

The MCS provides a correction to that infeasability; thus, if the constraints on the elements in the MCS are removed from the constraint system, that system is now satisfiable. If the guarantees are no longer constrained to true and the fault activation literals are not constrained to false, the negation of the safety property *with these unconstrained elements* is now satisfied. These correspond to the minimal set of faults and violated guarantees that cause violation of the safety property.

The basic idea of the proposed algorithm is to collect all MIVCs, transform them into MCSs (Minimal Correction Sets) by the use of a hitting set algorithm, and then transform the MCSs into MinCutSets. It is a requirement of the minimal hitting set algorithm that *all* MUSs are used to find the MCSs [23, 36, 39]. Thus, once all MIVCs have been found and the minimal hitting set algorithm has completed, the MinCutSet generation algorithm can begin.

The MinCutSet generation algorithm begins with a list of $MCSs$ specific to a top level property. Since MIVCs can contain fault activation literals constrained to *false* and and subcomponent contracts constrained to *true*, the $MCSs$ may also contain this mixture. Minimal cut sets only contain faults, and thus the MCSs require additional processing in order to be considered MinCutSets.

Since we assume that the nominal model proves, if any contracts are violated this is due to an active fault in the system. Each contract will then have it's own minimal cut set. Given that MIVCs are collected compositionally, each subcomponent contract has a set of minimal model elements required for its proof. These correspond (on the lowest level) to fault activation literals constrained to false. Once these are collected for the contracts in question, a replacement can be made in the MCS. The violated contract is replaced by the active fault literals that caused its violation. In the end, the MCS is transformed into a MinCutSet.

The formal proofs showing that this transformation is possible will be provided and the algorithms will be implemented in the Safety Annex.

3.3 Probabilistic Computations over the Fault Tree

Chapter 4

Conclusion

System safety analysis is cruicial in the development of critical systems and the generation of accurate and scalable results is invaluable to the assessment process. Having multiple ways to capture complex dependencies between faults and the behavior of the system in the presence of these faults is important throughout the entire process. The artifacts generated from such analyses that are used in the certification process of such systems must be generated in a scalable way and provide accurate and important information. This project has developed and implemented the Safety Annex for AADL which provides a way to capture complex relationships between faults in a model and analyze their effects behaviorally through either compositional or monolithic analysis.

Furthermore, we propose the compositional generation of minimal cut sets to be used in the development of various artifacts used in system certification, such as FTA, FMEA, and single point of failure examinations. This generation is done through the collection of proof elements called MIVCs and their transformation.

Lastly, we propose to use the minimal cut sets and resulting fault trees generated through the transformation algorithms to calculate the probability of the top level event, or violation of the safety property.

References

[1] AS5506C. Architecture Analysis & Design Language (AADL), Jan. 2017.

[2] J. Bendík, E. Ghassabani, M. Whalen, and I. Černá. Online enumeration of all minimal inductive validity cores. In *International Conference on Software Engineering and Formal Methods*, pages 189–204. Springer, 2018.

[3] P. Bieber, C. Bougnol, C. Castel, J. P. Heckmann, C. Kehren, S. Metge, and C. Seguin. Safety Assessment with Altarica - Lessons Learnt Based on Two Aircraft System Studies. In *In 18th IFIP World Computer Congress*, 2004.

[4] P. Bieber, C. Bougnol, C. Castel, J.-P. H. C. Kehren, S. Metge, and C. Seguin. Safety assessment with altarica. In *Building the Information Society*, pages 505–510. Springer, 2004.

[5] P. Bieber, J.-L. Farges, X. Pucel, L.-M. Sèjeau, and C. Seguin. Model - based safety analysis for co-assessment of operation and system safety: application to specific operations of unmanned aircraft. In *ERTS2*, 2018.

[6] B. Bittner, M. Bozzano, R. Cavada, A. Cimatti, M. Gario, A. Griggio, C. Mattarei, A. Micheli, and G. Zampedri. The xSAP Safety Analysis Platform. In *TACAS*, 2016.

[7] M. Bozzano, H. Bruintjes, A. Cimatti, J.-P. Katoen, T. Noll, and S. Tonetta. The compass 3.0 toolset (short paper). In *IMBSA 2017*, 2017.

[8] M. Bozzano, A. Cimatti, A. Griggio, and C. Mattarei. Efficient Anytime Techniques for Model-Based Safety Analysis. In *Computer Aided Verification*, 2015.

[9] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri. The COMPASS Approach: Correctness, Modelling and Performability of Aerospace Systems. In *Computer Safety, Reliability, and Security*. Springer Berlin Heidelberg, 2009.

[10] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Yen Nguyen, T. Noll, and M. Roveri. Model-based codesign of critical embedded systems. 507, 2009.

[11] M. Bozzano, A. Cimatti, O. Lisagor, C. Mattarei, S. Mover, M. Roveri, and S. Tonetta. Symbolic Model Checking and Safety Assessment of Altarica Models. In *Science of Computer Programming*, volume 98, 2011.

[12] M. Bozzano, A. Cimatti, C. Mattarei, and S. Tonetta. Formal safety assessment via contract-based design. In *Automated Technology for Verification and Analysis*, 2014.

[13] M. Bozzano, A. Cimatti, M. Roveri, J. P. Katoen, V. Y. Nguyen, and T. Noll. Codesign of dependable systems: A component-based modeling language. In *2009 7th IEEE/ACM International Conference on Formal Methods and Models for Co-Design*, 2009.

[14] M. Bozzano, A. Cimatti, and F. Tapparo. Symbolic fault tree analysis for reactive systems. In *ATVA*, 2007.

[15] D. Chen, N. Mahmud, M. Walker, L. Feng, H. Lönn, and Y. Papadopoulos. Systems Modeling with EAST-ADL for Fault Tree Analysis through HiP-HOPS*. *IFAC Proceedings Volumes*, 46(22):91 – 96, 2013.

[16] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. Nusmv: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2000.

[17] D. D. Cofer, A. Gacek, S. P. Miller, M. W. Whalen, B. LaValley, and L. Sha. Compositional Verification of Architectural Models. In *NFM 2012*, volume 7226, pages 126–140, April 2012.

[18] J. De Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial intelligence*, 32(1):97–130, 1987.

[19] C. Ericson. Fault tree analysis - a history. In *Proceedings of the 17th International Systems Safety Conference*, 1999.

[20] P. Feiler and D. Gluch. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley Professional, 2012.

[21] P. Feiler, J. Hudak, J. Delange, and D. Gluch. Architecture fault modeling and analysis with the error model annex, version 2. Technical Report CMU/SEI-2016-TR-009, Software Engineering Institute, 06 2016.

[22] A. Gacek, J. Backes, M. Whalen, L. Wagner, and E. Ghassabani. The JKind Model Checker. *CAV 2018*, 10982, 2018.

[23] A. Gainer-Dewar and P. Vera-Licona. The minimal hitting set generation problem: algorithms and computation. *SIAM Journal on Discrete Mathematics*, 31(1):63–100, 2017.

[24] E. Ghassabani, A. Gacek, and M. W. Whalen. Efficient generation of inductive validity cores for safety properties. *CoRR*, abs/1603.04276, 2016.

[25] E. Ghassabani, M. W. Whalen, and A. Gacek. Efficient generation of all minimal inductive validity cores. *2017 Formal Methods in Computer Aided Design (FMCAD)*, pages 31–38, 2017.

[26] M. Gudemann and F. Ortmeier. A framework for qualitative and quantitative formal model-based safety analysis. In *HASE 2010*, 2010.

[27] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The Synchronous Dataflow Programming Language Lustre. In *IEEE*, volume 79(9), pages 1305–1320, 1991.

[28] R. Heckel. Compositional verification of reactive systems specified by graph transformation. In *International Conference on Fundamental Approaches to Software Engineering*, pages 138–153. Springer, 1998.

[29] P. Hönig, R. Lunde, and F. Holzapfel. Model Based Safety Analysis with smartIflow. *Information*, 8(1), 2017.

[30] P. Hönig, R. Lunde, and F. Holzapfel. Model Based Safety Analysis with smartIflow. *Information*, 8(1), 2017.

[31] A. Joshi and M. P. Heimdahl. Model-Based Safety Analysis of Simulink Models Using SCADE Design Verifier. In *SAFECOMP*, volume 3688 of *LNCS*, page 122, 2005.

[32] A. Joshi and M. P. Heimdahl. Behavioral Fault Modeling for Model-based Safety Analysis. In *Proceedings of the 10th IEEE High Assurance Systems Engineering Symposium (HASE)*, 2007.

[33] A. Joshi, S. P. Miller, M. Whalen, and M. P. Heimdahl. A Proposal for Model-Based Safety Analysis. In *In Proceedings of 24th Digital Avionics Systems Conference*, 2005.

[34] J.-P. Katoen, M. Khattri, and I. S. Zapreev. A markov reward model checker. In *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems*, QEST '05. IEEE Computer Society, 2005.

[35] M. Kwiatkowska, G. Norman, and D. Parker. PRiSM 4.0: Verification of Probabilistic Real-time Systems. In *In Proceedings of the 23rd International Conference on Computer Aided Verification (CAV '11)*, volume 6806 of LNCS, 2011.

[36] M. H. Liffiton, A. Previti, A. Malik, and J. Marques-Silva. Fast, flexible mus enumeration. *Constraints*, 21(2):223–250, 2016.

[37] O. Lisagor, T. Kelly, and R. Niu. Model-based safety assessment: Review of the discipline and its challenges. In *The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety*, 2011.

[38] MRMC: Markov Rewards Model Checker. http://wwwhome.cs.utwente.nl/ zapreevis/mrmc/.

[39] K. Murakami and T. Uno. Efficient algorithms for dualizing large-scale hypergraphs. In *2013 Proceedings of the Fifteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 1–13. SIAM, 2013.

[40] NuSMV Model Checker. http://nusmv.itc.it.

[41] A. Pnueli. In transition from global to modular temporal reasoning about programs. In *Logics and models of concurrent systems*, pages 123–144. Springer, 1985.

[42] T. Prosvirnova, M. Batteux, P.-A. Brameret, A. Cherfi, T. Friedlhuber, J.-M. Roussel, and A. Rauzy. The AltaRica 3.0 Project for Model-Based Safety Assessment. *IFAC*, 46(22), 2013.

[43] RAT: Requirements Analysis Tool. http://rat.itc.it.

[44] R. Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987.

[45] E. Ruijters and M. Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer science review*, 15-16:29–62, 5 2015.

[46] SAE ARP 4761. Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, December 1996.

[47] SAE ARP4754A. Guidelines for Development of Civil Aircraft and Systems, December 2010.

[48] D. Stewart, J. Liu, M. Heimdahl, M. Whalen, D. Cofer, and M. Peterson. Architectural modeling and analysis for safety engineering (AMASE), NASA final report. `https://github.com/loonwerks/AMASE/tree/master/doc/AMASE_Final_Report_2019`, 2019.

[49] D. Stewart, J. Liu, M. Heimdahl, M. Whalen, D. Cofer, and M. Peterson. The Safety Annex for Architecture Analysis and Design Language. In *10th Edition European Congress Embedded Real Time Systems*, to appear 2020.

[50] D. Stewart, J. Liu, M. Whalen, D. Cofer, and M. Peterson. Safety Annex for Architecture Analysis Design and Analysis Language. Technical Report 18-007, University of Minnesota, March 2018.

[51] D. Stewart, M. Whalen, D. Cofer, and M. P. Heimdahl. Architectural Modeling and Analysis for Safety Engineering. In *IMBSA 2017*, pages 97–111, 2017.