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:

-

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 [**?**, **?**, **?**, **?**, **?**], 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) [**?**]. 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 [**?**] and ARP4761 [**?**]. 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. Chapter 3 describes the proposed approach. In Chapter 3, first we mention some formal notations and background. Then, we provide a formalization of the MIVC to MinCutSet transformation. We also provide the algorithm used to calculate system safety probability. The notion of minimal IVCs and minimal all IVCs are formalized there. Then, we show how these fault model artifacts can be used within the safety assessment process.

Chapter 2

Background

A model-based approach for safety analysis was proposed by Joshi et. al in [**?**, **?**, **?**]. 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) [**?**] and HiP-HOPS for EAST-ADL [**?**] 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) [**?**]. 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 [**?**, **?**]. 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 [**?**, **?**], MRMC (Markov Reward Model Checker) [**?**, **?**], and RAT (Requirements Analysis Tool) [**?**]. The safety analysis tool xSAP [**?**] can be invoked in order to generate safety analysis artifacts such as fault trees and FMEA tables [**?**]. 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 [**?**] 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 [**?**]: "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) [**?**] 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 [**?**], PRISM (Probabilistic Symbolic Model Checker) [**?**], or the MRMC probabilistic model checker [**?**].

AltaRica [**?**, **?**] 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 [**?**]. For this dialect the Safety assessment, fault tree generation, and functional verification can be performed with the aid of NuSMV model checking [**?**]. Failure states are defined throughout the system and flow variables are updated through the use of assertions [**?**]. 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 [**?**, **?**, **?**]. 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 [**?**].

Still need to add in research done in the realm of min ut set generation and probabilistic computations regarding top level event.

Chapter 3

Proposed Approach

3.1 Fault Tree Analysis

The use of fault trees are common in many safety assessment processes and the ability to generate the cut sets needed for the construction of the fault tree is a useful part of any safety analysis tool. The fault tree is a safety artifact commonly referenced in requirement protocol documents such as ARP4761, ARP4754, and AIR6110 [**?**, **?**, **?**].

A Fault Tree (FT) is a directed acyclic graph whose leaves model component failures and whose gates model failure propagation [**?**]. The system failure under examination is the root of the tree and is called the Top Level Event (TLE). The node types in a fault tree are *events* and *gates*. An event is an occurrence within the system, typically the failure of a subsystem down to an individual component. Events can be grouped into Basic Events (BEs), which occur independently, and *intermediate events* which occur dependently and are caused by one or more other events [**?**]. These events model the failure of the system (or subsystem) under consideration. The gates represent how failures propagate through the system and how failures in subsystems can cause system wide failures. The two main logic symbols used are the Boolean logic AND-gates and OR-gates. An AND-gate is used when the undesired top level event can only occur when all the lower conditions are true. The OR-gate is used when the undesired event can occur if any one or more of the next lower conditions is true. This is not a comprehensive list of gate types, but we focus our attention on these two common gate types.

Figure 3.1 shows a simple example of a fault tree based on SAE ARP4761 [**?**]. In this example, the top level event corresponds to an aircraft losing all wheel braking. In order for this event to occur, all of the basic events must occur. This is seen through the use of the AND gate below the top level event. The gates in the fault tree describe how failures propagate through the system. Each gate has one output and one or more inputs. In Figure 3.1, the AND gate has three inputs and one output. The leaves of the tree represent the basic events of the system. In the case of this fault tree, these three events are also the Minimal Cut Sets (MinCutSets) for this top level event. A MinCutSet is the minimal set of basic events that must occur together in order to cause the TLE to occur. Generating and analyzing these MinCutSets is important

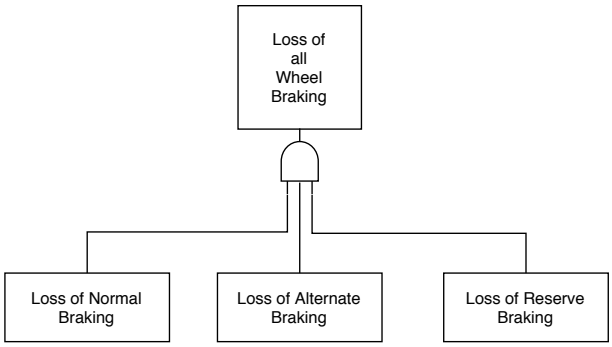

Figure 3.1: A simple fault tree

to FTA and has been an active area of interest in the research community since fault trees were first described in Bell Labs in 1961 [**?**, **?**].

There are two main types of fault tree analysis that we differentiate here as *qualitative* analysis and *quantitative* analysis. In qualitative analysis, the structure of the fault tree is considered and the MinCutSets are a way to indicate which combinations of component failures will cause the system to fail. On the other hand, in quantitative analysis the probability of the TLE is calculated given the probability of occurrence of the basic events. By being able to generate MinCutSets based on both cardinality and probability, this allows for either form of FTA to be created.

3.2 Inductive Validity Cores

Given a complex model, it is often useful to extract traceability information related to the proof, in other words, which portions 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 [**?**]. Given a safety property of the system, a model checker can be invoked in order 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 IVC elements [**?**].

The IVC 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 what are called 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.

In section **??**, we show the formal definitions of IVCs in detail.

Our approach utilizes a few tools in order to generate the artifacts of interest and a brief background will be helpful. and hence the rest of the background section consists of a brief description of AADL, AGREE, and the SOTERIA tools and languages.

3.3 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" [**?**, **?**]. 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.

3.4 Compositional Analysis

Compositional analysis of systems was introduced in order to address the scalability of model checking large software systems. 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. On the other hand, the analysis can 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 [**?**].

3.4.1 Assume Guarantee Reasoning Environment

The Assume Guarantee Reasoning Environment (AGREE) is a tool for formal analysis of behaviors in AADL models [**?**]. 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 [**?**] and then queries a user-selected model checker to conduct the back-end analysis. The analysis can be performed compositionally or monolithically.

3.5 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 [**?**,**?**]. add other citations 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.

3.6 Definitions

Intuitively a constraint system contains the contracts that constrain component behavior and faults that are defined over these components. In the case of a nominal model augmented with faults, a constraint system is 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 state space S, a transition system (I, T) consists of the initial state predicate $I : S \rightarrow \{0, 1\}$ and a transition step predicate $T : S \times S \rightarrow \{0, 1\}$. Reachability for (I, T) is defined as the smallest predicate $R : S \rightarrow \{0, 1\}$ that satisfies the following formulas:

$$\forall s.I(s) \Rightarrow R(s)$$
$$\forall s, s'.R \wedge T(s, s') \Rightarrow R(s')$$

A safety property $\mathcal{P} : S \rightarrow \{0, 1\}$ is a state predicate. A safety property $\mathcal{P}$ holds on a transition system (I, T) if it holds on all reachable states. More formally, $\forall s.R(s) \Rightarrow \mathcal{P}(s)$. When this is the case, we write $(I, T) \vdash \mathcal{P}$ [**?**].

Given a transition system that satisfies a safety property P, it is possible to find which model elements are necessary for satisfying the safety property through the use of the *All Minimal Inductive Validity Cores All-MIVCs* algorithms [**?**, **?**]. This 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. Recall that this constraint system is in terms of the *negation* of the safety property. 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. This information is used throughout this section to provide the underlying theory behind the generation of minimal cut sets from all MIVCs.

Definitions 1-3 are taken from research by Liffiton et. al. [**?**].

Definition 2. *: A Minimal Unsatisfiable Subset (MUS) of a constraint system C is :* $\{MUS \subseteq C \mid MUS$ *is UNSAT and* $\forall c \in MUS$*:* $MUS \setminus \{c\}$ *is SAT*$\}$*. This 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, this corresponds to which faults are not constrained to inactive (and are 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 :* $\{MCS \subseteq C \mid C \setminus MCS$ *is SAT and* $\forall S \subset MCS : C \setminus S$ *is UNSAT*$\}$*. 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 [**?**]. This duality is defined in terms of *minimal hitting sets*. 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 [**?**]. The MCSs can be generated from the MUSs if all MUSs are known. Thus, the use of the All-MIVCs algorithm is required.

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 MCSs are generated from the MUSs that are provided by the All-MIVCs algorithm [**?**] and a minimal hitting set algorithm developed by Murakami et. al. [**?**, **?**].

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). We define a minimal cut set consistently with much of the research in this field [**?**, **?**]

Definition 5. *: A Minimal Cut Set can be defined as the set of faults in a system that cause the violation of the safety property. Furthermore, any strict subset of these faults will not cause violation of the safety property.*

The generation of MIVCs traverses the program in a top down fashion. The transformation of MIVCs to MinCutSets traverses this tree in a bottom up fashion if and only if All-MIVCs have been generated. It is a requirement of the minimal hitting set algorithm that *all* MUSs are used to find the MCSs [**?**, **?**, **?**]. Thus, once All-MIVCs have been found and the minimal hitting set algorithm has completed, the MinCutSets Generation algorithm can begin.

The MinCutSets Generation Algorithm begins with a list of $MCSs$ specific to a top level property. These $MCSs$ may contain a mixture of fault activation literals constrained to *false* and and subcomponent contracts constrained to *true*. We remove all constraints from each MCS and call the resulting sets I, for *Intermediate* set. Replacement of subcomponent contracts with their respective minimal cut sets can then proceed. For each of those contracts in I, we check to see if we have previously obtained a $MinCutSet$ for that contract. If so, replacement is performed. If not, we recursively call this algorithm to obtain the list of all MinCutSets associated with this subcomponent contract. At a certain point, there will be no more contracts in the set I in which case we have a minimal cut set for the current property. When this set is obtained, we store it in a lookup table keyed by the given property that this I is associated with.

Notes regarding the following algorithm: at the onset, the current property P is a top level property. Each of the properties has a list of associated $MCSs$. When the algorithm states that constraints on these elements are removed, more specifically the fault activation literals in MCS are constrained to *true* and the component contracts are constrained to *false*. $List(I)$ is the collection of all $MCSs$ with all constraints removed. Assuming All-MIVCs have been found and the minimal hitting set algorithm has terminated, giving us a list of $MCSs$ for each property in the system.