Using Inductive Validity Cores to Produce Minimal Cut Sets

Danielle Stewart*, Jing (Janet) Liu†, Michael W. Whalen*, Darren Cofer†, Mats Heimdahl*
*University of Minnesota
Department of Computer Science and Engineering
Email: {dkstewar, whalen, heimdahl}@umn.edu
†Collins Aerospace
Trusted Systems - Engineering and Technology
Email: {Jing.Liu, Darren.Cofer}@collins.com

***Abstract*—Risk and fault analysis are important activities that help to ensure that critical systems operate in an expected way. 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 affects of these propagations can be a time consuming and error prone process. In this paper, 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. We describe the technique of generating these safety artifacts, prove that our approach is sound, and describe its implementation in the Osate tool suite for AADL. We then present experiments in which we benchmark our approach in terms of scalability and the artifacts produced.**

I. Introduction

Risk and safety analyses are an important activity used to ensure that critical systems operate in an expected way. From nuclear power plants and airplanes to heart monitors and automobiles, critical systems are vitally important in our society. The systems are required to not only operate safely under nominal (normal) conditions, but also under conditions when faults are present in the system. Guaranteeing these properties when in the presence of faults is an active area of research and various safety artifacts are often required during the development process of critical systems. In these systems, software is an important component to assess in terms of safety. An area of interest in safety-critical system engineering and development is how failures in the system are propagated through software designs. It is a difficult problem to reason about failure propagations through software in a large scale critical system and automating the process can provide much needed insight into the system and potential problems that could be costly in terms of resources or even life.

For complex systems, safety analysis techniques can produce thousands of combinations of events that can cause system failure. Many of these events are propagated through software components. Determination of these events can be a very time consuming and error prone process. SAE Standard Aerospace Recommended Practice (ARP) 4761 provides general guidance on evaluating the safety aspects of a design and from the design phase through to the detailed design phase, safety artifacts regarding fault propagation and effects are an important part of the assessment process [1].

In this paper, we describe a new technique for determining these events utilizing model checking techniques on a model of the system written in an architecture design language, AADL [2]. The model is annotated with assume-guarantee contracts for component level requirements on both hardware and software components using AGREE and extended into a fault model using the Safety Annex [3]–[5].

The AADL system model is annotated with behavioral contracts using AGREE and then extended with faults using the Safety Annex. During the analysis of the extended system model, a customized Lustre program is generated for each layer of the architecture and this is fed to JKind model checker [6]. The SMT solver JKind has the ability to produce all Inductive Validity Cores (IVCs) for a constraint system consisting of model elements with a safety property [7], [8]. IVCs trace a safety property to a minimal set of model elements necessary for proof. This research shows how the generation of all IVCs can be used to generate all Minimal Cut Sets (MinCutSets) with regard to a top level event (i.e. the negation of the safety property). This allows for compositional fault analysis and MinCutSet generation.

II. Background

Following the definitions provided by Liffiton, et. al. [9], we consider a constraint system C as an ordered set of abstract constraints $\{C_1, C_2, \ldots, C_n\}$ over some set of variables. Each constraint C_i restricts the assignment of those variables in some way. For every subset $S \subseteq C$, we have a method of determining whether S is *satisfiable (SAT), meaning there exists an assignment that is allowed by all* $C_i \in S$ *or* unsatisfiable, meaning no such assignment exists.

It is of interest to identify certain subsets of a constraint system that have particular properties.

A minimal unsatisfiable subset (MUS) M of a constraint system C is a subset $M \subseteq C$ such that M is unsatisfiable and

$\forall c \in M$, $Mn\{c\}$ is satisfiable. This is the minimal explanation of a constraint systems infeasability.

A related subset of a constraint system is the minimal correction set (MCS) M of a constraint system C is a subset $M \subseteq C$ such that $C \setminus M$ is satisfiable and $\forall S \subset M$, CnS is unsatisfiable. In other words, the removal of the constraints in M from C "corrects" the infeasibility of C and these are the minimal of such sets in C.

A duality exists between these two sets and it is defined in terms of *hitting sets*. A hitting set of a collection of sets A is a set H such that every set in A is hit by H. That means that H contains at least one element from every set in A. A *minimal hitting set* H_{min} is a hitting set that is subset minimal, i.e. no element can be removed from H_{min} without removing its hitting set property and missing some set in A. Every MUS of a constraint system is a minimal hitting set of the systems MCSs and every MCS is a minimal hitting set of its MUSs [9]–[11].

A. 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 [7]. 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 can extract traceability information from that proof process and return a minimal set of the model elements required in order to prove the property. A short time later, this algorithm was refined in order to produce all such minimal sets of IVC elements [8].

This algorithm is of interest to us for the following reason. All IVCs are MUSs of a constraint system that consists of the assumptions and contracts of system components and the negation of the safety property of interest. The set of all IVCs then contains the minimal explanation of the constraint systems infeasibility in terms of the negation of the safety property; hence, these are the minimal model elements necessary to proof the safety property.

In section XX, 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.

B. 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" [2], [12]. 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.

C. 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 [13].

D. Assume Guarantee Reasoning Environment

The Assume Guarantee Reasoning Environment (AGREE) is a tool for formal analysis of behaviors in AADL models [13]. 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 [14] and then queries a user-selected model checker to conduct the back-end analysis. The analysis can be performed compositionally or monolithically.

E. 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 [3], [4]. 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.

F. Fault Tree Analysis

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 occurance 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 1 shows a simple example of a fault tree based on SAE ARP4761 [1]. 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 1, the AND gate has three inputs and one output. The leaves of the tree represent the basic events of the system and in the case of this fault tree, these three events are also the Minimum Cut Set (MCS) for this top level event. An MCS is the minimum set of basic events that must occur together in order to cause the TLE to occur. Finding these sets is important 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 [15], [16].

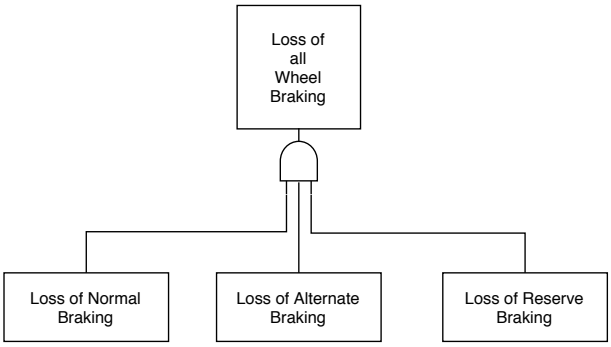

Fig. 1. A simple fault tree

There are two main types of FTA that we differentiate here as *qualitative* analysis and *quantitative* analysis. In qualitative analysis, the structure of the fault tree is considered and the cut sets 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 occurance of the basic events.

The formal definition of a fault tree is provided in section III and more details regarding quantitative and qualitative FTA is explained there as well.

III. Methodology

In our description leading up to the generation of compositional fault trees, we must first define important aspects of this procedure.

A. Preliminaries

Intuitively a constraint system contains the contracts that constrain component behavior and faults that are defined over these components. In the case of the nominal model, a constraint system is defined as follows:

Let F be the set of all faults defined in the model and G be the set of all component contracts (guarantees). $C = \{C_1, C_2, ..., C_n\}$ 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:

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

Given a state space S, a transition system (I, T) consists of the initial state predicate $I : S \to \{0, 1\}$ and a transition step predicate $T : S \times S \to \{0, 1\}$. Reachability for (I, T) is defined as the smallest predicate $R : S \to \{0, 1\}$ which 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 $\mathcal{P} : S \to \{0, 1\}$ 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 which satisfies a safety property P, it is possible to find which parts of the system are necessary for satisfying the safety property through the use of Ghassabani's *All Minimal Inductive Validity Cores* algorithm [**?**], [**?**]. This algorithm makes use of the collection of all minimal unsatisfiable subsets of a given transition system in terms of the negation of the top level property, i.e. the top level event. This is a minimal subset of the activation literals such that the constraint system C with the formula $\neg P$ is unsatisfiable when these activation literals are held true. Formally:

Definition 1. *: A Minimal Unsatisfiable Subset M of a constraint system C is : $\{M \subseteq C | M$ is UNSAT and $\forall c \in M$: $M \setminus \{c\}$ 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 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 2. *: A Minimal Correction Set M of a constraint system C is : $\{M \subseteq C | C \setminus M$ is SAT and $\forall S \subset M : C \setminus M$ 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 [10]. This duality is defined in terms of *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 [9].

Definition 3. *: 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, we can easily collect the MCSs from the MUSs provided through the all MIVC algorithm and a hitting set algorithm by Murakami et. al. [**?**], [**?**].

Cut sets and minimal cut sets provide important information about the vulnerabilities of a system. 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). We define MinCutSet in terms of the constraint system in question as follows:

Definition 4. *: Minimal Cut Set of a constraint system C with all faults in the system denoted as the set F is : $\{cut \subseteq F | C \setminus cut$ is SAT and $\forall c \subset cut : C \setminus c$ is UNSAT $\}$.*

When the MCS contains only faults, the MCS is equivalent to the MinCutSet as shown in the first part of the proof. When contracts exist in the MCS, a replacement can be made which transforms the MCS into the MinCutSet.

B. Qualitative Analysis

Transformation of MCS : When the MCS contains only faults, the MCS is equivalent to the MinCutSet as shown in the first part of the proof. When contracts exist in the MCS, a replacement can be made which transforms the MCS into the MinCutSet.

Theorem 1. *The MinCutSet can be generated by transformation of the MCS.*

Proof. For faults in the model F and subcomponent contracts G:
$MCS \cap F \neq \{\} \vee MCS \cap G \neq \{\}$.

Part 1: $MCS \cap G = \{\}$ (Leaf level of system)

(i) MCS $\subseteq$ MinCutSet:

Let $M \in MCS$. Then $C \setminus M$ is SAT. Since $!\exists g \in C$ for $g \in G$, thus $M \subseteq F$ and is a cut set.

By minimality of the MCS, M is a minimal cut set for $\neg P$.

(ii) $MinCutSet \subseteq$ MCS:

Let $M \in MinCutSet$. Then all faults in M cause $\neg P$ to occur by definition. Thus, $C \setminus M$ is SAT.

By minimality of $MinCutSet$, M is also minimal and thus is a minimal correction set.

Part 2: $MCS \cap G \neq \{\}$ (Intermediate level of system)

Assume $\overline{C} = \{F, G, P\}$ with the constraints that all $f \in F$ are inactive and all $g \in G$ are valid with regard to top level property P, i.e. the nominal model proves.

Let $MCS = \{f_1, ..., f_n, g_1, ..., g_m\}$
For $g_1 \in MCS$, $\exists F_1 \subseteq F$ where F_1 is a minimal set of active faults that cause the violation of g_1. Replace g_1 in MCS with F_1. Then $MCS = \{f_1, ..., f_n, F_1, g_2, ..., g_m\}$.

Perform this replacement for all $g_i \in MCS$ until we reach $MCS = \{f_1, ..., f_n, F_1, ..., F_m\}$.

Since F_i is minimal, the MCS retains its minimality. Furthermore $MCS \subseteq F$ and $C \setminus MCS$ is SAT. Therefore the MCS is transformed into MinCutSet.

□

Replacement of a Contract with its MinCutSets for Max N Faults Analysis : At the leaf level, only faults are contained in IVCs (and consequently in MCSs). Thus we store these cut sets in a lookup table for quick access throughout the algorithm. For this algorithm, we assume we are in an intermediate level of the analysis. Given a fault hypothesis of max N faults, we disregard any cut sets over cardinality N and collect the rest.

Assuming we have used the hitting set to generate MCS from the IVC, this is where the algorithm begins.

Let $MCS = F \cup G$ for faults $f \in F$ and contracts $g \in G$. If $|G| = 0$, then add MCS to contract lookup table (it is already a MinCutSet).

Assume $|G| = \alpha > 0$.

Let $Cut(g_j) = cut_1(g_j), ..., cut_\beta(g_j)$ be all minimal cut sets for a contract $g_j \in G$ where $|Cut(g_j)| = \beta_j$.

Algorithm 1: Replacement

```
List(MCS) = MCS : initialize list of MCSs that
  contain contracts ;
Cut(g_j) : all minimal cut sets with cardinality less
  than or equal to N of the contract g_j ;
0 < j ≤ α ;
0 < i ≤ γ ;
for all MCS_i ∈ List(MCS) do
    Remove MCS_i from List(MCS);
    for all g_j ∈ MCS_i do
        Remove g_j from MCS_i ;
        for all cut_k(g_j) ∈ Cut(g_j) do
            Add cut_k(g_j) to MCS_i ;
            if |MCS_i| ≤ N then
                if ∃g_j ∈ MCS_i then
                    Add MCS_i to List(MCS)
                else
                    Add MCS_i to contract look up
                      table (done)
```

The number of replacements R that are made in this algorithm are given as the combination of minimal cut sets of the contracts within the MCS. The validity of this statement follows directly from the general multiplicative combinatorial principle. Therefore, the number of replacements R is given by:

$$R = \prod_{j=1}^{\alpha} |Cut(g_j)|$$

This is equivalent to the number of minimal cut sets generated which follows easily from the combinatorial calculation. It is also important to note that the cardinality of $List(MCS)$ is bounded. Every new MCS that is generated that still contains contracts is added to $List(MCS)$. Thus every contract up until the penultimate contract contributes to

the cardinality of $List(MCS)$. The bound is given by:

$$|List(MCS)| \leq \prod_{j=1}^{\alpha-1} |Cut(g_j)|$$

Theorem 2. *The elimination of any* $|MCS_i| \leq N$ *will not eliminate any* $|MinCutSet| \leq N$.

Proof. If $!\exists g_j \in MCS_i$, then $MCS_i = MinCutSet$ by proof previously given. Thus, this cut set is not required for consideration in the analysis since $|MinCutSet| > N$.

If $\exists g_j \in MCS_i$, then further replacement will need to be made. Thus in the next phase of the algorithm, we will have MCS_{i+1}. In this case, the size can only get larger and $|MCS_i| \leq |MCS_{i+1}|$ and we have not eliminated a cut set that must be considered for the analysis.

□

C. Quantitative analysis

IV. Case Studies

To demonstrate the effectiveness of this approach and compare to state of the art existing tools, we describe two case studies.

A. Wheel Brake System

The Wheel Brake System (WBS) described in AIR6110 is a well-known example that has been used as a case study for safety analysis, formal verification, and contract based design [17]–[21]. The preliminary work for the safety annex used a simplified model of the WBS [3]. In order to demonstrate scalability of our tools and compare results with other studies, we constructed a functionally and structurally equivalent AADL version of one of the most complex WBS NuSMV/xSAP models (arch4wbs) described in [17], [22]. We refer readers to this publication for diagrams of the full arch4wbs model [17]. A simplified diagram taken from ARP4761 is shown in Figure 2.

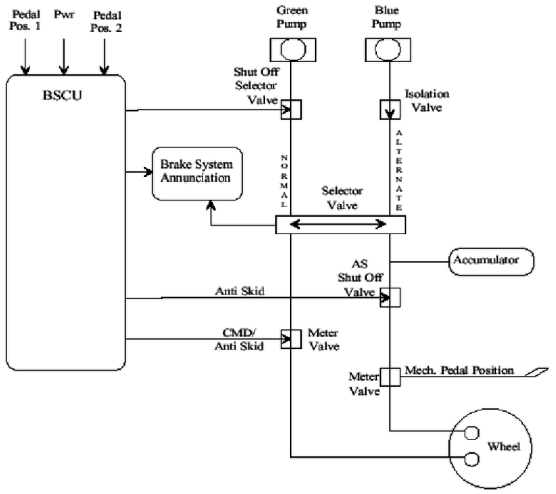

Fig. 2. Simplified model of the WBS from ARP4761

1) WBS architecture description: The WBS is composed of two main parts: the control system and the physical system. The control system electronically controls the physical system and contains a redundant Braking System Control Unit (BSCU) in case of failure. The physical system consists of the hydraulic circuits running from hydraulic pumps to wheel brakes. This is what provides braking force to each of the 8 wheels of the aircraft.

There are three operating modes in the WBS model. In *normal* mode, the system uses the *green* hydraulic circuit. The normal system is composed of the green hydraulic pump and one meter valve per each of the 8 wheels (shown as one wheel in Figure 2). Each of the meter valves are controlled through electronic commands coming from the BSCU. These signals provide brake commands as well as antiskid commands for each of the wheels. The braking command is determined through a sensor on the pilot pedal position. The antiskid command is calculated based on information regarding ground speed, wheel rolling status, and braking commands.

In *alternate* mode, the system uses the *blue* hydraulic circuit. The wheels are all mechanically braked in pairs (one pair per landing gear). The alternate system is composed of the blue hydraulic pump, four meter valves, and four antiskid shutoff valves. The meter valves are mechanically commanded through the pilot pedal corresponding to each landing gear. If the system detects lack of pressure in the green circuit, the selector valve switches to the blue circuit. This can occur if there is a lack of pressure from the green hydraulic pump, if the green hydraulic pump circuit fails, or if pressure is cut off by a shutoff valve. If the BSCU channel becomes invalid, the shutoff valve is closed.

The last mode of operation of the WBS is the *emergency* mode. This is supported by the blue circuit but operates if the blue hydraulic pump fails. The accumulator pump has a reserve of pressurized hydraulic fluid and will supply this to the blue circuit in emergency mode.

The model size and statistics are described in Table I.

Architecture	Total component types	30
	Leaf component types	20
	Total component instances	169
	Leaf component instances	143
	Max depth	5
Nominal Contracts	No. of assumptions	17
	No. of guarantees	113
	No. of System level properties	29
Fault Model	Total fault nodes	33
	No. of faults in instance	141

TABLE I
WBS Model Statistics

2) System Safety Properties: The AIR6110 document contains a set of safety requirements on the expected probability of an unwanted occurance of event, e.g. ''the loss of all wheel braking shall be extremely remote." The case study performed using xSAP/NuSMV focused on these safety requirements that are outlined in AIR6110 [17], [22]. The detailed tech report of the xSAP/NuSMV model of the WBS is also available [23].

S18-WBS-R-0321 *Never loss of all wheel braking.*

S18-WBS-R-0322-left *Never asymmetrical loss of wheel braking (left side).*

S18-WBS-R-0322-right *Never asymmetrical loss of wheel braking (right side).*

S18-WBS-0323 *Never inadvertent braking with all wheels locked during takeoff roll before V1.*

S18-WBS-R-0324 *Never inadvertent braking of all wheels during takeoff roll after V1.*

S18-WBS-R-0325-w1...8 *Never inadvertent braking of one wheel without locking (wheels 1-8).*

Braking implies cmd w1...8 *If system is braking, then pilot has commanded braking (wheels 1-8).*

Cmd implies braking w1...8 *If pilot commands braking, then system is braking (wheels 1-8).*

These requirements violations are the top level events (TLE) for the fault tree computations.

3) Qualitative analysis comparison: In qualitative analysis, we are interested in the Minimal Cut Sets (MCS), their cardinalities, and the time of computation. Previous work produced hierarchical fault trees for all top level properties without bounds on the cardinality of cut sets or fault restrictions. They performed the hierarchical fault tree generation and implemented two procedures to generate the fault trees from the model. the algorithms were either based on existing Binary Decision Diagram (BDD) procedures or a combination of BDD with Bounded Model Checking (BMC). The BMC and BDD approach first computes the MCSs up to some specific depth using BMC and then performs a BDD algorithm to generate the rest of the MCSs [18], [22].

4) Quantitative analysis comparison: For quantitative analysis, the main calculation of interest is: what is the probability of a TLE occuring? Given a set of MCSs and probabilities for leaf level faults (BEs), the computation of this TLE probability is possible using the techniques described in section III. Previous work has used this technique of TLE probabilistic computation when the MCSs were able to be computed. The TLE probabilities were approximated in the events of timeouts.

V. Related Work

Formal verification tools based on model checking have been used to automate the generation of safety artifacts [24]–[26]. The techniques and tools developed early on suffered from a limitation brought on by difficulty in generating MCSs and readable fault trees or other safety artifacts.

Work has been done towards mitigating these limitations by the scalable generation of readable fault trees. Othello Contracts Refinement Analysis (OCRA) is a tool that checks refinement of contracts specified in linear temporal logic [27]. This tool was extended in order to provide hierarchically organized and automatically generated fault trees [18]. This process entails extending the model to include failure ports on both the input (failure of environment) and output (failure of component). The fault tree is generated starting at the TLE and linking it to the intermediate events that could cause this event to occur. The tree is defined recursively in this fashion.

Given a set of MCSs and probabilities for leaf level faults (BEs), the computation of this TLE probability is possible using BDD techniques. The difficulty in this calculation is that often with large systems it is not easy (or possible) to get MCSs. This problem was addressed in previous work using an ''anytime" approach in which the TLE probability is approximated while utilizing model checking to compute MCSs of an xSAP model [19], [22]. This provides a way to estimate the probability using lower and upper bounds and even if the model checker cannot complete its task of finding all MCSs, the probability range is known.

Closely related to our work is the model-based safety assessment toolset called COMPASS (Correctness, Modeling project and Performance of Aerospace Systems) [28]. COMPASS is a tool suite that uses the SLIM language, which is based on a subset of AADL, for its input models [29], [30]. 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 [31], [32], MRMC (Markov Reward Model Checker) [33], [34], and RAT (Requirements Analysis Tool) [35]. The safety analysis tool xSAP [26] can be invoked in order to generate safety analysis artifacts such as fault trees and FMEA tables [36]. The fault tree generation provided utilizes the "anytime' techniques for probabilistic analysis and the contract based approach as well as the xSAP approach for the safety artifact generation' [28].

VI. Conclusion

We have developed a way to leverage recent research in model checking techniques in order to generate minimal cut sets in a compositional fashion. Using the idea of Inductive Validity Cores (IVCs), which are the minimal model elements necessary for a proof of a safety property, we are able to restate the safety property as a top level event and provide the model elements as faults of components and their contracts. The JKind model checker then uses the faults and contracts as model elements and returns all minimal IVC elements. These are used to generate minimal cut sets using the Hitting Set algorithm. Future work includes... For more details on the tool, models, and approach, see the technical report [4]. To access the tool plugin, users manual, or models, see the repository [37].

Acknowledgments. This research was funded by NASA contract NNL16AB07T and the University of Minnesota College of Science and Engineering Graduate Fellowship.

References

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

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

[3] D. Stewart, M. Whalen, D. Cofer, and M. P. Heimdahl, "Architectural Modeling and Analysis for Safety Engineering," in *IMBSA 2017*, 2017, pp. 97–111.

[4] D. Stewart, J. Liu, M. Whalen, D. Cofer, and M. Peterson, "The Safety Annex for Architecture Analysis Design and Analysis Language," University of Minnesota, Tech. Rep. 18-007, March 2018. [Online]. Available: https://www.cs.umn.edu/research/technical_reports/view/18-007

[5] J. Backes, D. Cofer, S. Miller, and M. W. Whalen, "Requirements Analysis of a Quad-Redundant Flight Control System," in *NFM*, ser. LNCS, vol. 9058, 2015, pp. 82–96.

[6] A. Gacek, J. Backes, M. Whalen, L. Wagner, and E. Ghassabani, "The JKind Model Checker," *ArXiv e-prints*, Dec. 2017.

[7] E. Ghassabani, A. Gacek, and M. W. Whalen, "Efficient generation of inductive validity cores for safety properties," *CoRR*, vol. abs/1603.04276, 2016. [Online]. Available: http://arxiv.org/abs/1603.04276

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

[9] M. H. Liffiton, A. Previti, A. Malik, and J. Marques-Silva, "Fast, flexible mus enumeration," *Constraints*, vol. 21, no. 2, pp. 223–250, 2016.

[10] R. Reiter, "A theory of diagnosis from first principles," *Artificial intelligence*, vol. 32, no. 1, pp. 57–95, 1987.

[11] J. De Kleer and B. C. Williams, "Diagnosing multiple faults," *Artificial intelligence*, vol. 32, no. 1, pp. 97–130, 1987.

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

[13] D. D. Cofer, A. Gacek, S. P. Miller, M. W. Whalen, B. LaValley, and L. Sha, "Compositional Verification of Architectural Models," in *NFM 2012*, vol. 7226, April 2012, pp. 126–140.

[14] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The Synchronous Dataflow Programming Language Lustre," in *IEEE*, vol. 79(9), 1991, pp. 1305–1320.

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

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

[17] M. Bozzano, A. Cimatti, A. F. Pires, D. Jones, G. Kimberly, T. Petri, R. Robinson, and S. Tonetta, "Formal Design and Safety Analysis of AIR6110 Wheel Brake System," in *CAV 2015, Proceedings, Part I*, 2015, pp. 518–535.

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

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

[20] A. Joshi and M. P. Heimdahl, "Model-Based Safety Analysis of Simulink Models Using SCADE Design Verifier," in *SAFECOMP*, ser. LNCS, vol. 3688, 2005, p. 122.

[21] AIR 6110, "Contiguous Aircraft/System Development Process Example," Dec. 2011.

[22] Mattarei, "Scalable safety and reliability analysis via symbolic model checking: theory and applications," Theses, Universita Degli Studi di Trento, Feb. 2016. [Online]. Available: https://profiles.stanford.edu/cristian-mattarei

[23] M. Bozzano, A. Cimatti, A. F. Pires, D. H. Jones, G. Kimberly, T. J. Petri, R. V. Robinson, and S. Tonetta, "Contract-based design and safety analysis of an aircraft wheel brake system: Revisiting AIR6110 with formal methods," Foundazione Bruno Kessler, Tech. Rep., Feb 2015. [Online]. Available: https://es-static.fbk.eu/projects/air6110/download/case_study_report.pdf

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

[24] 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*, vol. 98, 2011.

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

[26] 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.

[27] A. Cimatti, M. Dorigatti, and S. Tonetta, "Ocra: A tool for checking the refinement of temporal contracts," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Nov 2013, pp. 702–705.

[28] 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.

[29] 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.

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

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

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

[33] 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*, ser. QEST '05. IEEE Computer Society, 2005.

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

[36] 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.

[37] D. Stewart, J. Liu, M. Whalen, D. Cofer, and M. Peterson, "Safety annex for aadl repository," https://github.com/loonwerks/AMASE, 2017.