Using Minimal Inductive Validity Cores to Generate Minimal Cut Sets

Danielle Stewart[1], Mats Heimdahl[1], Michael Whalen[1], Jing (Janet) Liu[2], and Darren Cofer[2]

[1] University of Minnesota, Minneapolis, MN, USA,
{dkstewar, heimdahl, mwwhalen}@umn.edu
[2] Collins Aerospace - Trusted Systems, Cedar Rapids, IO, USA,
{jing.liu, darren.cofer}@collins.com

Abstract. Risk and fault analysis are activities that help to ensure that critical systems operate in an expected way, even in the presence of component failures. As critical systems become more dependent on software components, analyses of error propagation through these software components becomes crucial. These analyses should be understandable to the analyst, scalable, and sound, in order to provide sufficient guarantees that the system is safe. A commonly used safety artifact is the set of all *minimal cut sets*, minimal sets of faults that may lead to a violation of a safety property. In this research, we define how minimal cut sets can be derived from certain results of model checking, the Minimal Inductive Validity Cores (MIVCs). Using a compositional model checking approach, we can incorporate both hardware and software failures and auto-generate safety artifacts. This research describes a technique for determining the Minimal Cut Sets by the use of MIVCs and producing compositionally derived artifacts that encode pertinent system safety information. We describe our technique, prove that it is sound, and demonstrate it in an implementation in the OSATE tool suite for AADL.

1 Introduction

Risk and safety analyses are important activities 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. These systems are required to not only operate safely under nominal (normal) conditions, but also under conditions when faults are present in the system. Guaranteeing that system safety properties hold in the presence of faults is an important aspect of critical systems development and falls under the discipline of safety analysis. Safety analysis produces various safety related artifacts that are often used during the development process of critical systems [1,2]. Many of these safety artifacts require the generation of *Minimal Cut Sets*, the minimal sets of faults that cause the violation of a system safety property. Since the introduction of minimal cut sets in the field of safety analysis [3], much research has been performed to address the generation of these sets [4–8]. One of the challenges with minimal cut set generation is is scaling to industrial-sized systems. As the system gets larger, more minimal cut sets are possible with increasing cardinality. In recent years, the capabilities of model checking have been leveraged to address this problem. [4, 9–13].

Scaling model checking of complex hardware and software is challenging; one way to address this problem is to take advantage of the architecture of the system model through a *compositional* approach [14–16]. Compositional model checking reduces the verification of a large system into multiple smaller verification problems that can be solved independently and which together guarantee correctness of the original problem. One way to structure compositional verification is hierarchically: layers of the system architecture are analyzed independently and their composition demonstrates a system property of interest.

Recently, Ghassabani et al. developed an algorithm that traces a safety property to a minimal set of model elements necessary for proof; this is called the *all minimal inductive validity core* algorithm (`All_MIVCs`) [17–19]. Inductive validity cores produce the minimal set of model elements necessary to prove a property. Each set contains the *behavioral contracts* – the requirement specifications for components – of the model used in a proof. When the `All_MIVCs` algorithm is run, this gives the minimal set of contracts required for proof of a safety property. If all of these sets are obtained, we have insight into every proof path for the property. Thus, if we violate at least one contract from every MIVC set, we have in essence "broken" every proof path. This is the information that is used to perform fault analysis using MIVCs.

Safety analysts are often concerned with faults in the system, i.e., when components or subsystems deviate from nominal behavior, and the propagation of errors through the system. To this end, the model elements included in the reasoning process of the `All_MIVCs` algorithm are not only the contracts of the system, but faults as well. This will provide additional insight into how an active fault may violate contracts that directly support the proof of a safety property.

This paper proposes a new method of minimal cut set generation using compositional model checking, allowing us to reason uniformly about faults in hardware and software and their impact (propagation) to system properties. The main contributions of this research are summarized as follows: (1) We propose a novel method for minimal cut set generation using Minimal Inductive Validity Cores (MIVCs) generated during model checking. (2) We provide proof of the soundness of this method. (3) We discuss the implementation of the algorithm for compositional cut set generation.

The organization of the paper is as follows. Section 2 provides a running example, Section 3 provides the preliminaries for Section 4 which outlines the formalisation of this approach. The implementation of the algorithms is discussed in Section 5 and related work follows in Section 6. The paper ends with a conclusion and discussion of related work.

2 Running Example

We present a running example of a simplified sensor system in a Pressurized Water Reactor (PWR). In a typical PWR, the core inside of the reactor vessel produces heat. Pressurized water in the primary coolant loop carries the heat to the steam generator. Within the steam generator, heat from the primary coolant loop vaporizes the water in a secondary loop, producing steam. The steamline directs the steam to the main turbine, causing it to turn the turbine generator, which produces electricity. There are a few important factors that must be considered during safety assessment and system design. An unsafe climb in temperature can cause high pressure and hence pipe rupture, and high levels of radiation could indicate a leak of primary coolant.

The following sensor system can be thought of as a subsystem within a PWR that monitors these factors. A diagram of the model is shown in Figure 1 and represents a highly simplified version of a safety critical system. The temperature subsystem details are shown at the bottom of Figure 1; each of the subsystems have a similar architecture.

The subsystems each contain three sensors that monitor pressure, temperature, and radiation. Environmental inputs are fed into each sensor in the model and the redundant sensors monitor temperature, pressure, or radiation respectively. If temperature, pressure, or radiation is too high, a shut down command is sent from the sensors to the parent components.

2.1 PWR Nominal Model

The temperature, pressure, and radiation sensor subsystems use a majority voting mechanism on the sensor values and will send a shut down command based on this output. The safety property of

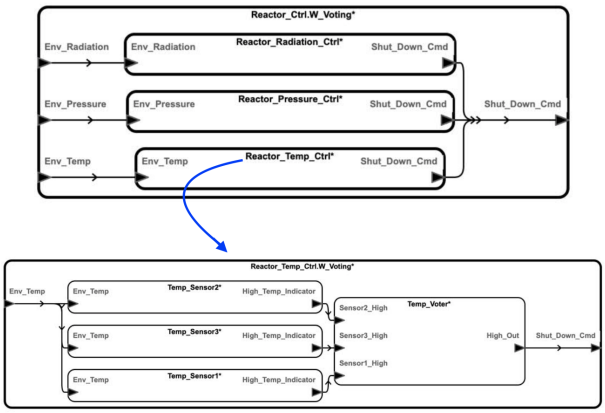

Fig. 1. PWR Sensor System

interest in this system is: *shut down when and only when we should*; the AGREE guarantee stating this property is shown in Figure 2.

```
guarantee "Shut down when and only when we should":
    Shut_Down_Cmd =
        ((Env_Temp> HIGH_TEMPERATURE_THRESHOLD)  or
        (Env_Pressure > HIGH_PRESSURE_THRESHOLD) or
        (Env_Radiation > HIGH_RADIATION_THRESHOLD));
```

Fig. 2. Sensor System Safety Property

The safety of the system requires a shut down to take place if the temperature, pressure, or radiation levels climb beyond safe levels; thus, a threshold for each subsystem is introduced. If any sensor subsystem reports passing that threshold, a shutdown command is sent. Supporting guarantees are located in each sensor subsystem and correspond to temperature, pressure, and radiation sending a shut down command if sensed inputs are above a given threshold. Each sensor has a similar guarantee. For reference throughout this paper, we provide Figure 3 which shows the guarantees and faults of interest for this running example.

Note: the thresholds vary for pressure, temperature, and radiation. These are given as constants T_p, T_t, and T_r respectively. The overall (or "top level") shutdown command is defined notationally as S; each sensor subsystem provides their own shutdown command, S_p for example. The faults are shown as "fail low" which correspond to the temp (or pressure or radiation) being high, but the sensor reports safe ranges. We also do not list all guarantees and assumptions that are in the model, but only the ones of interest for the illustration.

2.2 PWR Fault Model

The faults that are of interest in this example system are any one of the sensors failing high or low. If sensors report high and a shut down command is sent, we shut down when we should not. On the other hand, if sensors report low when it should be high, a shut down command is not sent and we

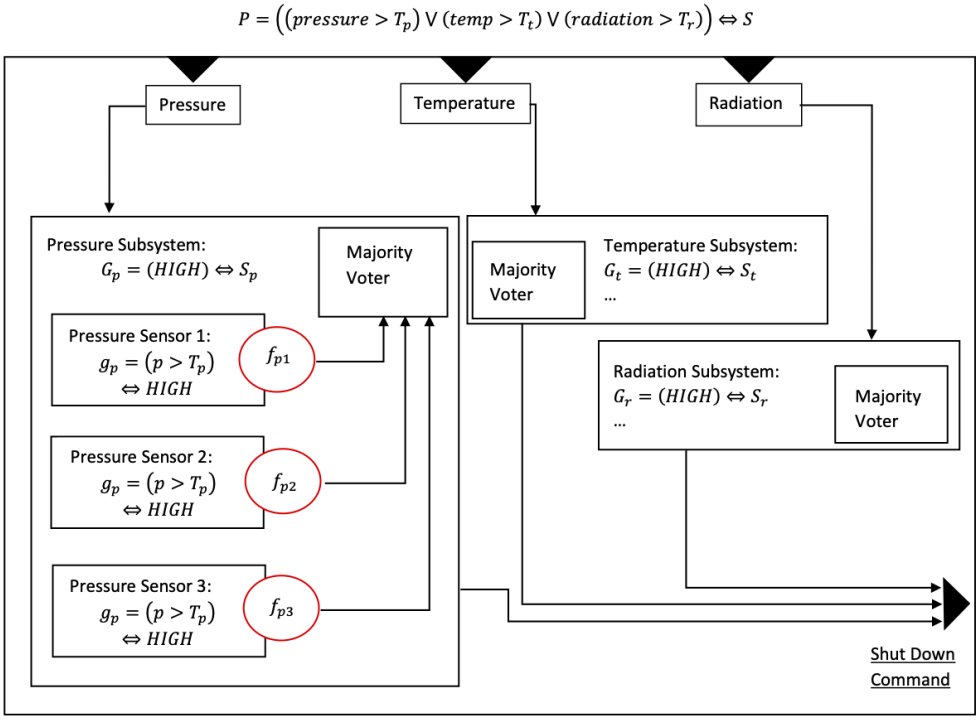

Fig. 3. Sensor System Nominal and Fault Model Details

do not shut down when we should. For simplification, we focus on the failures when sensors report low when they should not.

A fault is defined for each sensor in the system using the safety annex. An example of a temperature sensor fault stuck at high is shown in Figure 4.

```
annex safety {**
    fault temp_sensor_stuck_at_high "temp sensor stuck at high": Common_Faults.stuck_true {
        inputs: val_in <- High_Temp_Indicator;
        outputs: High_Temp_Indicator <- val_out;
        probability: 1.0E-5 ;
        duration: permanent;
    }
**};
```

Fig. 4. Fault on Temperature Sensor Defined in the Safety Annex for AADL

The Safety Annex provides a way to weave the faults into the nominal model by use of the *inputs* and *outputs* keywords. This allows users to define a fault and attach it to the output of a component. If the fault is active, the error can possibly violate the guarantees of this component or the assumptions

of downstream components [20]. The activation of a fault is not up to the user, but instead left up to the model checker, JKind, to determine if the activation of this fault will contribute to a violation of higher level guarantees. If so, it can be activated during the analysis.

3 Preliminaries

The algorithms in this paper are implemented in the Safety Annex for the Architecture Analysis and Design Language (AADL) and require the Assume-Guarantee Reasoning Environment (AGREE) [21] to annotate the AADL model in order to perform verification using the back-end model checker `JKind` [22].

Architecture Analysis and Design Language We are using the Architectural Analysis and Design Language (AADL) to construct system architecture models of performance-critical, embedded, real-time systems [23, 24]. Language annexes to AADL provide a richer set of modeling elements for various system design and analysis needs, and the language definition is sufficiently rigorous to support formal analysis tools that allow for early phase error/fault detection.

Compositional Analysis One way to structure compositional verification is hierarchically: layers of the system architecture are analyzed independently and their composition demonstrates a system property of interest. Compositional verification partitions the formal analysis of a system architecture into verification tasks that correspond into the decomposition of the architecture [15]. A proof consists of demonstrating that the system property is provable given the contracts of its direct subcomponents and the system assumptions [15, 25]. 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 [21, 25, 26].

Assume Guarantee Reasoning Environment The Assume Guarantee Reasoning Environment (AGREE) is a tool for formal analysis of behaviors in AADL models and supports compositional verification [21]. It is implemented as an AADL annex and is used to annotate AADL components with formal behavioral contracts. Each component's contracts includes assumptions and guarantees about the component's inputs and outputs respectively. AGREE translates an AADL model and the behavioral contracts into Lustre [27] and then queries the `JKind` model checker to conduct the back-end analysis [22].

JKind JKind is an open-source industrial infinite-state inductive model checker for safety properties [22]. Models and properties in JKind are specified in Lustre [27], a synchronous dataflow language, using the theories of linear real and integer arithmetic. JKind uses SMT-solvers to prove and falsify multiple properties in parallel.

Safety Annex for AADL The Safety Annex for AADL provides the ability to reason about faults and faulty component behaviors in AADL models [20, 28]. In the Safety Annex approach, AGREE is used to define the nominal behavior of system components, faults are introduced into the nominal model, and JKind is used to analyze the behavior of the system in the presence of faults. Faults describe deviations from the nominal behavior and are attached to the outputs of components in the system.

3.1 Formal Background

Given a state space U, a transition system (I, T) consists of an initial state predicate $I : U \rightarrow bool$ and a transition step predicate $T : U \times U \rightarrow bool$. We define the notion of reachability for (I, T) as

the smallest predicate $R : U \rightarrow bool$ which satisfies the following formulas:

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

A safety property $P : U \rightarrow bool$ is a state predicate. A safety property P holds on a transition system (I, T) if it holds on all reachable states, i.e., $\forall u.\ R(u) \Rightarrow P(u)$, written as $R \Rightarrow P$ for short. When this is the case, we write $(I, T) \vdash P$.

Completely cut inductive and SAT descriptions - this paragraph is taken from FMCAD paper. The 2016 paper has more info in induction; I can extend this with that in mind if necessary. Just let me know.

The idea behind finding an IVC for a given property P [17] is based on inductive proof methods used in SMT-based model checking, such as k-induction and IC3/PDR [29–31]. Generally, an IVC computation technique aims to determine, for any subset $S \subseteq T$, whether P is provable by S. Then, a minimal subset that satisfies P is seen as a minimal proof explanation called a minimal Inductive Validity Core. Ghassabani et al. demonstrate that the minimization process is as hard as model checking [18], so finding a minimal inductive validity core may not be possible for some model checking problems.

Definition 1. *Inductive Validity Core (IVC) [17]: $S \subseteq T$ for $(I, T) \vdash P$ is an Inductive Validity Core, denoted by $IVC(P, S)$, iff $(I, S) \vdash P$.*

Definition 2. *Minimal Inductive Validity Core (MIVC) [18]: $S \subseteq T$ is a minimal Inductive Validity Core, denoted by $MIVC(P, S)$, iff $IVC(P, S) \wedge \forall T_i \in S.\ (I, S \setminus \{T_i\}) \nvdash P$.*

Okay, it seems to me that I *have* to bridge the gap somehow between transition systems and IVCs and constraint systems with their MUSs/MCSs/hitting sets. Without some discussion on how this can occur, it will make little sense to a reader that isn't familiar. So, I placed this conversation in prelim and made them into definitions for later reference. A k-induction model checker utilizes parallel SMT-solving engines at each induction step to glean information about the proof of a safety property. The transition formula is translated into clauses such that satisfiability is preserved [32]. The translated system, consisting of the constrained formulas of the transition system and the negation of the property, is often called a *constraint system*. The `All_MIVCs` algorithm collects all *minimal unsatisfiable subsets* (MUSs) of a constraint system generated from a transition system at each induction step. [18, 19].

Definition 3. *A Minimal Unsatisfiable Subset (MUS) [33].M of a constraint system C is a set $M \subseteq C$ such that M is unsatisfiable and $\forall c \in M : M \setminus \{c\}$ is satisfiable.*

The MUSs are the minimal explanation of the infeasibility of this constraint system; equivalently, these are the minimal sets of model elements necessary for proof of the safety property.

Returning to our running example, this can be illustrated by the following. Given the constraint system $C = \{G_p, G_t, G_r, \neg P\}$, a minimal explanation of the infeasability of this system is the set $\{G_p, G_t, G_r, \}$. If all three guarantees hold, then P is provable.

A related set is a *minimal correction set*:

Definition 4. *A Minimal Correction Set (MCS) [33] M of a constraint system C is a subset $M \subseteq C$ such that $C \setminus M$ is satisfiable and $\forall S \subset M : C \setminus S$ is unsatisfiable.*

A MCS can be seen to "correct" the infeasability of the constraint system by the removal from C the constraints found in an MCS. In the case of an UNSAT system, we may ask: what will correct this unsatisfiability? Returning to the PWR example, we can find the MCSs of the constraint system C: $MCS_1 = \{G_t\}$, $MCS_2 = \{G_p\}$, $MCS_3 = \{G_r\}$. If any single guarantee is violated, a shut down from that subsystem will not get sent when it should and the safety property P will be violated.

A duality exists between the MUSs of a constraint system and the MCSs as established by Reiter [33]. This duality is defined in terms of *Minimal Hitting Sets* (*MHS*).

Definition 5. *A hitting set of a collection of sets A is a set H such that every set in A is "hit" by H; H contains at least one element from every set in A.*

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 [33–35]. For the PWR top level constraint system, it can be seen that each of the MCSs intersected with the MUS is nonempty. This gives the minimal set of guarantees for which, if violated, will cause P to be violated.

4 Formalization

Given an initial state I and a transition relation T consisting of conjunctive constraints as defined in section 3. The nominal guarantees of the system, G, consist of conjunctive constraints $g \in G$. Given no faults (i.e., nominal system), each g is one of the transition constraints T_i where:

$$T_n = g_1 \wedge g_2 \wedge \cdots \wedge g_n \tag{1}$$

We consider an arbitrary layer of analysis of the architecture and assume the property holds of the nominal relation $(I, T_n) \vdash P$. Given that our focus is on safety analysis in the presence of faults, let the set of all faults in the system be denoted as F. A fault $f \in F$ is a deviation from the normal constraint imposed by a guarantee. Without loss of generality, we associate a single fault and an associated fault probability with a guarantee. Each fault f_i is associated with an *activation literal*, af_i, that determines whether the fault is active or inactive.

To consider the system under the presence of faults, consider a set GF of modified guarantees in the presence of faults and let a mapping be defined from activation literals $af_i \in AF$ to these modified guarantees $gf_i \in GF$.

$$\sigma : AF \rightarrow GF$$
$$gf_i = \sigma(af_i) = \textit{if } af_i \textit{ then } f_i \textit{ else } g_i$$

The transition system is composed of the set of modified guarantees GF and a set of conjunctions assigning each of the activation literals $af_i \in AF$ to false:

$$T = gf_1 \wedge gf_2 \wedge \cdots \wedge gf_n \wedge \neg af_1 \wedge \neg af_2 \wedge \cdots \wedge \neg af_n \tag{2}$$

Theorem 1. *If $(I, T_n) \vdash P$ for T_n defined in equation 1, then $(I, T) \vdash P$ for T defined in equation 2.*

Proof. By application of successive evaluations of σ on each constrained activation literal $\neg af_i$ and the weakening of the antecedent by introduction of the activation literals, the result is immediate. □

Consider the elements of T as a set $GF \cup AF$, where GF are the potentially faulty guarantees and AF consists of the activation literals that determine whether a guarantee is faulty. This is a set that is considered by an SMT-solver for satisfiability during the k-induction procedures. The posited problem is thus: $GF \wedge AF \wedge \neg P$ for the safety property in question. Recall, if this is an *unsatisfiable* constraint system, then P is provable given these constraints. On the other hand, if it is *satisfiable*, then we know that given the constraints in GF and AF, P is not provable. These satisfiability constraints contain the information we wish to find.

Let us view this in terms of the PWR system example and focus on the temperature sensor subsystem. The safety property to be proved is G_t, the supporting guarantees are found in each of the three temperature sensors, g_{ti}. Faults f_{ti} are defined for each sensor. The transition system is:

$$T = gf_{t1} \wedge gf_{t2} \wedge gf_{t3} \wedge \neg af_{t1} \wedge \neg af_{t2} \wedge \neg af_{t3}$$

The MIVCs for this subsystem layer correspond to all pairwise combinations of constrained activation literals. Intuitively, if any two sensor faults do *not* occur, then two of the three sensor guarantees are not violated and the system responds appropriately to high temperature; therefore, G_t is provable.

The MCSs for this subsystem layer happen to also correspond to all pairwise combinations of constrained activation literals. If any two sensor faults *do* occur, then two of the three sensor guarantees will be violated and the system does not respond to high temperature as required. This would result in the inability to prove G_t. (Note: it is not always the case that the MCSs are the same as the MIVCs – in this case it is due to majority voting on three sensors.)

4.1 Transforming MCS into Minimal Cut Set

The MCSs contain the information needed to find minimal cut sets, but their elements consist of constrained activation literals and/or guarantees. The link between the activation literals, faults, and guarantees is defined through σ mapping (equation 4); σ must be applied to each element in an MCS to map back to the associated fault. Without loss of generality, let $MCS = \{af_1, \cdots, af_m\}$. Let $\sigma(MCS) = \{\sigma(\neg af_1), \cdots, \sigma(af_m)\}$ be a mapping where MCS is a minimal correction set with regard to some property G and $MCS \subseteq AF$.

Lemma 1. *$\sigma(MCS)$ is a cut set of G.*

Proof. Assume towards contradiction that $\sigma(MCS)$ is not a cut set of G. Then $gf_1 \wedge \cdots \wedge gf_n \wedge af_1 \cdots \wedge af_m \wedge \neg af_{k+1} \wedge \neg af_n \wedge \neg G$ is unsatisfiable. Thus, the *true* activation literals do not affect the provability of G. This contradicts $C \setminus MCS$ is satisfiable. □

Lemma 2. *$\sigma(MCS)$ is minimal.*

Proof. Assume toward contradiction that $\sigma(MCS)$ is not minimal with regard to G. Then there exists $S \subset MCS$ such that $\sigma(S)$ is a minimal cut set of G. This implies that the corresponding constraint system $C \setminus S$ is satisfiable. This contradicts the minimality of MCS. □

Minimal cut sets generated by monolithic analysis look only at explicitly defined faults throughout the architecture and attempt through various techniques to find the minimal violating set for a particular property. In this approach, explicit faults are analyzed as well as supporting guarantees. We view violated guarantees as deviations from nominal behavior and refer to them as "faulty". Thus, this approach provides an overapproximation of the minimal cut sets compared to a monolithic approach. Let $MonoCuts$ be the set of all minimal cut sets using a monolithic approach and let $CompCuts$ be the set of minimal cut sets using the above approach.

Theorem 2. $MonoCuts \subseteq CompCuts$.

Proof. Let $M \in MonoCuts$ where M is a mimimal cut set for safety property P. Then all $f_i \in M$, if active simultaneously, violate the property P. A direct translation of this system to a σ form constraint system gives: $g_1 \wedge \cdots \wedge g_n \wedge \neg af_1 \wedge \cdots \wedge \neg af_n \wedge \neg P$.

Without loss of generality, let $M = \{f_1, \ldots, f_k\}$. Then we know that $g_1 \wedge \cdots \wedge g_n \wedge af_1 \wedge \cdots \wedge af_k \wedge \neg af_{k+1} \wedge \cdots \wedge \neg af_n \wedge \neg P$ is satisfiable. Then $\{af_1, \ldots, af_k\}$ is a correction set for the system and can be mapped by σ to a minimal cut set by Lemmas 1 - 2. □

This isn't quite right... Monolithic takes *all* guarantees in the model in the proof where as compositional takes only that layer. I do not make the distinction in this proof. Will consider this and rewrite.

5 Implementation

In the formalism section, we viewed the problem from the perspective of an arbitary layer of the architectural analysis. This resulted in explicit faults at the leaf level and violated guarantees ("non-deterministic faults") at the middle/top layers. Each MCS generated at each level gives varied insight into the system. In this implementation section, we describe the mapping of the results from an arbitrary layer into the top/mid or leaf layers of analysis. Minimal cut sets traditionally contain explicitly defined faults as elements; to this end, we implemented a compositional mapping from explicit faults to the guarantees they violate. The end result are the minimal cut sets that contribute to a violation of the top level safety property.

The `All_MIVCs` algorithm requires *IVC elements* to be explicitly added to the Lustre program; these elements are those of consideration for the MIVC analysis. In this implementation, the IVC elements are added differently depending on the layer. In the leaf architectural level, only explicitly defined faults are added to IVC elements. In middle or top layers, supporting guarantees are added. This is shown in Figure 5.

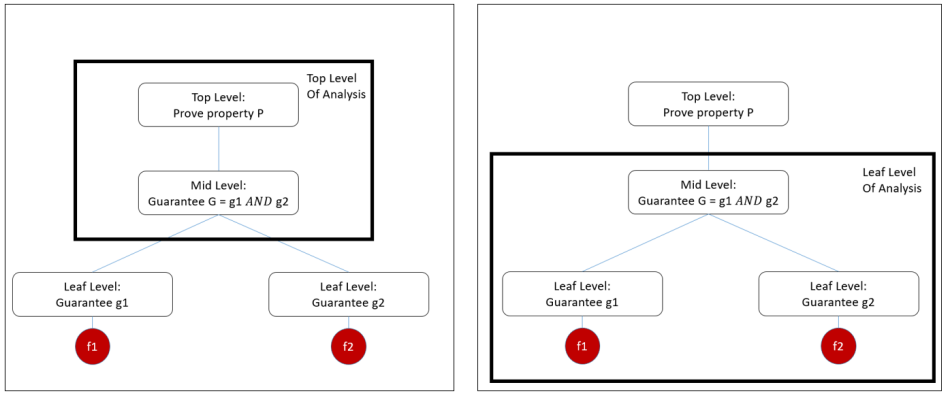

Fig. 5. Illustration of Two Layers of Analysis

The figure shows an arbitrary architecture with two analysis layers: top and leaf. The top layer analysis adds G as IVC element; the leaf layer analysis adds f_1 and f_2 as IVC elements. The first layer of analysis shows that G supports the proof of P, thus is an MIVC. The second layer of analysis shows that *given the model*, if both f_1 and f_2 are constrained to false, a proof is found for G.

Each explicit fault defined in the safety annex is added to the Lustre program as described in safety annex implementation [20, 28] and additionally the constrained faults are added as IVC elements for leaf layer analysis. A requirement of the hitting set algorithm is that to find *all* MCSs, *all* MUSs must be known. Ghassabani et al. [18] showed that finding all MIVCs is as hard as model checking; and thus cannot always be found. It is a requirement of this analysis that all MIVCs are computed. Once this analysis is complete at each layer, a hitting set algorithm is used to generate the related MCSs [36]. Depending on the layer of analysis, the MCSs contain either faulty (or violated) guarantees or explicitly defined faults.

Since minimal cut sets traditionally contain only explicit faults, we aimed to provide minimal cut sets in this format. A mapping is performed from the bottom up which replaces a guarantee with its corresponding cut set. At the leaf level, only constrained faults are in the MCS and thus can be saved as a mapping from the property under proof consideration to the set of *unconstrained* faults that contribute to its violation. In Figure 5, the mapping would be from G to $\{f_1, f_2\}$ showing that the minimal cut set for G is $\{f_1, f_2\}$.

We continue in this fashion until all MCSs in that layer are processed. Then we move up; if a guarantee is in the MCS, we check for a mapping from that contract to its minimal cut set. If it exists, we perform iterative replacements (a guarantee can have multiple cut sets). This mapping is shown in Algorithm 1.

Algorithm 1: Transform MCS into Minimal Cut Sets

```
1 Function Init(List(MCS), P):
2     for all MCS ∈ List(MCS) do
3         MCS̄ ← remove constraints from MCS
4         List(MCS̄) ← append MCS̄
5     map ← map(P → ∅)                    // from P to list of min cut sets
6     Transform(List(MCS̄), P)
1 Function Transform(List(MCS̄),P, map):
2     for all MCS̄ ∈ List(MCS̄) do
3         if ¬∃g ∈ MCS̄ then
4             map(P → append MCS̄)
5         else
6             for all cut(g) do
7                 newMCS̄ ← replace g with cut(g)
8                 List(MCS̄) ← append newMCS̄
```

The number of replacements R that are made in this algorithm are constrained by the number of minimal cut sets there are for all α contracts within the initial MCS. We call the set of all minimal cut sets for a contract g: $Cut(g)$. The following formula defines an upper bound on the number of replacements. The validity of this statement follows directly from the general multiplicative combinatorial principle.

Lemma 3. *The number of replacements R is bounded by the following formula:*

$$R \leq \sum_{i=1}^{\alpha}(\prod_{j=1}^{i} |Cut(g_j)|) \tag{3}$$

Proof. Assume there exists a $g_i \in \overline{MCS}$. The number of replacements made between g_i and its minimal cut sets is at most $|Cut(g_i)|$. We iteratively perform this replacement for all α contracts in $\overline{MCS}$ and make, in the worst case, $|Cut(g_1)| \times |Cut(g_2)| \times \cdots \times |Cut(g_\alpha)|$ replacements. □

It is also important to note that the cardinality of $List(\overline{MCS})$ is bounded, i.e. the algorithm terminates. Every new set that is generated through some replacement of a contract with its minimal cut set is added to $List(\overline{MCS})$ in order to continue the replacement process for all contracts in $\overline{MCS}$.

Theorem 3. *Algorithm 1 terminates*

Proof. No infinite sets are generated by the `All_MIVCs` or minimal hitting set algorithms [18, 37]; therefore, every MCS produced is finite. Thus, every minimal cut set of every contract is finite. Furthermore, a bound exists on the number of additional sets that are added to $List(I)$:
$|List(\overline{MCS})| \leq R$ by lemma 3. □

Given that the worst case size in terms of cut set cardinality and number of replacements can grow quickly, we implemented strategies to prune the size of the cut sets and hence the growth of these intermediate sets.

5.1 Pruning to Address Scalability

The safety annex provides the capability to specify a type of verification in what is called a *fault hypothesis statement*. These come in two forms: maximum number of faults or probabilistic analysis. Algorithm 1 is the general approach, but the implementation changes slightly depending on which form of analysis is being performed. This pruning improves performance and diminishes the problem of combinatorial explosions in the size of minimal cut sets for larger models.

Max N Analysis Pruning This statement restricts the number of faults that can be independently active simultaneously and verification is run with this restriction present. For example, if a max 2 fault hypothesis is specified, two or fewer faults may be active at once. In terms of minimal cut sets, this statement restricts the cardinality of minimal cut sets generated to N.

If the number of faults in an intermediate set $List(\overline{MCS})$ exceeds the threshold N, any further replacement of remaining contracts in that intermediate set can never decrease the total number of faults in $List(\overline{MCS})$; therefore, this intermediate set is eliminated from consideration.

Probabilistic Analysis Pruning The second type of hypothesis statement restricts the cut sets by use of a probabilistic threshold. Any cut sets with combined probability higher than the given probabilistic threshold are removed from consideration. The allowable combinations of faults are calculated before the transformation algorithm begins; this allows for a pruning of intermediate sets during the transformation. If the faults within an intermediate set are not a subset of any allowable combination, that intermediate set is pruned from consideration and no further replacements are made.

6 Related Work

The representation of Boolean formulae as Binary Decision Diagrams (BDDs) was first formalized in the mid 1980s [38] and were extended to the representation of fault trees not many years later [5]. After this formalization, the BDD approach to FTA provided a new approach to safety analysis. The model is constructed using a BDD, then a second BDD - usually slightly restructured - is used to encode MinCutSets [39]. Unfortunately, due to the structure of BDDs, the worst case is exponential in size in terms of the number of variables [5, 38, 39]. In industrial sized systems, this is not realistically useful.

SAT based computation was then introduced to address scalability problems in the BDD approach; initially it was used as a preprocessing step to simplify the decision diagram [40], but later extended to allow for all MinCutSet processing and generation without the use of BDDs [41]. Since then, numerous safety related research groups have focused on leveraging the power of model checking in the problems of safety assessment [9, 10, 20, 41–45].

Bozzano et al. formulated a Bounded Model Checking (BMC) approach to the problem by successively approximating the cut set generation and computations to allow for an "anytime approximation" in cases when the cut sets were simply too large and numerous to find [41, 46]. These algorithms are implemented in xSAP [47] and COMPASS [48].

The model based safety assessment tool AltaRica 3.0 [49] performs a series of processing to transform the model into a reachability graph and then compile to Boolean formula in order to compute the MinCutSets [50]. Other tools such as HiP-HOPS [51] have implemented algorithms that follow the failure propagations in the model and collect information about safety related dependencies and hazards. The Safety Analysis Modeling Language (SAML) [52] provides a safety specific modeling language that can be translated into a number of input languages for model checkers in order to provide model checking support for MinCutSet generation.

To our knowledge, a fully compositional approach to calculating minimal cut sets has not been introduced.

7 Conclusion and Future Work

Should there be a discussion on other things in future work? 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 faults of components and their contracts as model elements to the `All_MIVCs` algorithm which provides all minimal IVCs that pertain to this property. These are used to generate minimal cut sets. Future work includes leveraging the system information embedded in this approach to generate hierarchical fault trees as well as perform scalability studies that compare this approach with other non-compositional approaches to MinCutSet generation. To access the algorithm implementation, Safety Annex users manual, or example models, see the repository [53].

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. SAE ARP4754A, "Guidelines for Development of Civil Aircraft and Systems," December 2010.
3. W. Vesely, F. Goldberg, N. Roberts, and D. Haasl, "Fault tree handbook," Technical report, US Nuclear Regulatory Commission, Tech. Rep., 1981.
4. 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.
5. A. Rauzy, "New algorithms for fault trees analysis," *Reliability Engineering & System Safety*, vol. 40, no. 3, pp. 203–211, 1993.
6. C. Ericson, "Fault tree analysis - a history," in *Proceedings of the 17th International Systems Safety Conference*, 1999.
7. M. Bozzano and A. Villafiorita, *Design and Safety Assessment of Critical Systems*, 1st ed. Boston, MA, USA: Auerbach Publications, 2010.
8. M. Rausand and A. Høyland, *System reliability theory: models, statistical methods, and applications*. John Wiley & Sons, 2003, vol. 396.
9. P. Bieber, C. Castel, and C. Seguin, "Combination of fault tree analysis and model checking for safety assessment of complex system," in *European Dependable Computing Conference*. Springer, 2002, pp. 19–31.
10. A. Schäfer, "Combining real-time model-checking and fault tree analysis," in *International Symposium of Formal Methods Europe*. Springer, 2003, pp. 522–541.
11. M. Bozzano, A. Cimatti, C. Mattarei, and S. Tonetta, "Formal safety assessment via contract-based design," in *Automated Technology for Verification and Analysis*, 2014.
12. M. Bozzano, A. Cimatti, and F. Tapparo, "Symbolic fault tree analysis for reactive systems," in *ATVA*, 2007.
13. 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.
14. R. J. Anderson, P. Beame, S. Burns, W. Chan, F. Modugno, D. Notkin, and J. D. Reese, "Model checking large software specifications," *ACM SIGSOFT Software Engineering Notes*, vol. 21, no. 6, pp. 156–166, 1996.
15. E. M. Clarke, D. E. Long, and K. L. McMillan, "Compositional model checking," in *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*. IEEE, 1989, pp. 353–362.
16. K. L. McMillan, "Verification of infinite state systems by compositional model checking," in *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*. Springer, 1999, pp. 219–237.
17. 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
18. 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.
19. 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*. Springer, 2018, pp. 189–204.
20. 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*, Jan 2020.
21. 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.
22. A. Gacek, J. Backes, M. Whalen, L. Wagner, and E. Ghassabani, "The JKind Model Checker," *CAV 2018*, vol. 10982, 2018.
23. AS5506C, "Architecture Analysis & Design Language (AADL)," Jan. 2017.
24. P. Feiler and D. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley Professional, 2012.
25. D. Cofer, A. Gacek, S. Miller, M. W. Whalen, B. LaValley, and L. Sha, "Compositional verification of architectural models," in *NASA Formal Methods Symposium*. Springer, 2012, pp. 126–140.

26. R. Heckel, "Compositional verification of reactive systems specified by graph transformation," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 1998, pp. 138–153.
27. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The Synchronous Dataflow Programming Language Lustre," in *IEEE*, vol. 79(9), 1991, pp. 1305–1320.
28. D. Stewart, M. Whalen, D. Cofer, and M. P. Heimdahl, "Architectural Modeling and Analysis for Safety Engineering," in *IMBSA 2017*, 2017, pp. 97–111.
29. N. Een, A. Mishchenko, and R. Brayton, "Efficient implementation of property directed reachability," in *2011 Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, 2011, pp. 125–134.
30. T. Kahsai, P.-L. Garoche, C. Tinelli, and M. Whalen, "Incremental verification with mode variable invariants in state machines," in *NASA Formal Methods Symposium*. Springer, 2012, pp. 388–402.
31. S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*, 1971, pp. 151–158.
32. N. Eén and N. Sörensson, "Temporal induction by incremental sat solving," *Electronic Notes in Theoretical Computer Science*, vol. 89, no. 4, pp. 543–560, 2003.
33. R. Reiter, "A theory of diagnosis from first principles," *Artificial intelligence*, vol. 32, no. 1, pp. 57–95, 1987.
34. M. H. Liffiton, A. Previti, A. Malik, and J. Marques-Silva, "Fast, flexible MUS enumeration," *Constraints*, vol. 21, no. 2, pp. 223–250, 2016.
35. J. De Kleer and B. C. Williams, "Diagnosing multiple faults," *Artificial intelligence*, vol. 32, no. 1, pp. 97–130, 1987.
36. A. Gainer-Dewar and P. Vera-Licona, "The minimal hitting set generation problem: algorithms and computation," *SIAM Journal on Discrete Mathematics*, vol. 31, no. 1, pp. 63–100, 2017.
37. 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)*. SIAM, 2013, pp. 1–13.
38. R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.
39. A. Rauzy, "Binary decision diagrams for reliability studies," in *Handbook of performability engineering*. Springer, 2008, pp. 381–396.
40. M. Bozzano, A. Cimatti, O. Lisagor, C. Mattarei, S. Mover, M. Roveri, and S. Tonetta, "Safety assessment of altarica models via symbolic model checking," *Science of Computer Programming*, vol. 98, pp. 464–483, 2015.
41. M. Bozzano, A. Cimatti, A. Griggio, and C. Mattarei, "Efficient anytime techniques for model-based safety analysis," in *International Conference on Computer Aided Verification*. Springer, 2015, pp. 603–621.
42. M. Bozzano, A. Cimatti, and F. Tapparo, "Symbolic fault tree analysis for reactive systems," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2007, pp. 162–176.
43. M. Bozzano and A. Villafiorita, "Improving system reliability via model checking: The fsap/nusmv-sa safety analysis platform," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2003, pp. 49–62.
44. M. Volk, S. Junges, and J.-P. Katoen, "Fast dynamic fault tree analysis by model checking techniques," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 370–379, 2017.
45. 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.
46. C. Mattarei, "Scalable safety and reliability analysis via symbolic model checking: Theory and applications," Ph.D. dissertation, Ph. D. thesis, University of Trento, Trento, Italy, p 2, 2016.
47. 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.
48. 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.
49. T. Prosvirnova, "AltaRica 3.0: a Model-Based approach for Safety Analyses," Theses, Ecole Polytechnique, Nov. 2014. [Online]. Available: https://pastel.archives-ouvertes.fr/tel-01119730
50. T. Prosvirnova and A. Rauzy, "Automated generation of minimal cut sets from altarica 3.0 models," *International Journal of Critical Computer-Based Systems*, vol. 6, no. 1, pp. 50–80, 2015.

51. Y. Papadopoulos and M. Maruhn, "Model-based synthesis of fault trees from matlab-simulink models," in *2001 International Conference on Dependable Systems and Networks*. IEEE, 2001, pp. 77–82.
52. M. Gudemann and F. Ortmeier, "A framework for qualitative and quantitative formal model-based safety analysis," in *HASE 2010*, 2010.
53. D. Stewart, J. Liu, M. Whalen, D. Cofer, and M. Peterson, "Safety annex for AADL repository," https://github.com/loonwerks/AMASE, 2018.