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 violate a safety property. In this research, we define how minimal cut sets can be derived from certain results of model checking, which are called 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 IVCs 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. 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 that system safety properties hold in the presence of faults is an important aspect of critical systems development and falls under the realm 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* (MinCutSets), the minimal sets of faults that cause the violation of a system safety property. Since the introduction of MinCutSets 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].

Model checking of complex hardware and software models can be challenging in terms of scalability; 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; a model checker then performs the verification per layer of the system hierarchy.

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 not only what is necessary for the verification of the property, but we can also find what combination of contracts, if *violated*, will provide a state of the system which makes the safety property unprovable.

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 on how an active fault may violate contracts that directly support the proof of a safety property.

This paper proposes a new method of MinCutSet generation in a compositional fashion. The main contributions of this research are summarized as follows: 1. We propose a novel method of MinCutSet generation by leveraging Minimal Inductive Validity Cores (MIVCs). 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

To illustrate the generation of minimal cut sets through the use of IVCs, we present a running example of a 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 pipe rupture, likewise a climb in pressure. A very low pressure can indicate a leak somewhere in the lines, 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 AADL model is shown in Figure 1. May add cartoon figure demonstrating this model/process later - depending on space. Also can adjust figure placement after rewriting, adding, and cutting is done.

2.1 PWR Nominal Model

The "top-level" system is an abstraction of the PWR and contains sensor subsystems. The subsystems contain 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.

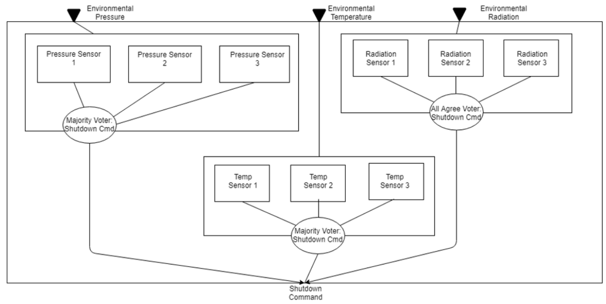

Fig. 1. Pressurized Water Reactor Sensor System

The temperature, pressure, and radiation sensor subsystems use a voting mechanism on the redundant sensor values and will send a shut down command based on this output. The safety property of 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 become unsafe; thus, a threshold is introduced and if any sensor subsystem reports passing that threshold, a shutdown command is sent. But on the other hand, we do not want to shut down the system if it is not necessary. If a sensor reports high temperature erroneously and a shut down occurs, this costs time and money. This is the reason that the contract is stated as such.

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.

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 shouldn't. On the other hand, if sensors report low when it should be high, a shut down command is not sent and we do not shut down when we should. For the remainder of this example, we focus on the failures when sensors report low when they shouldn't.

Two faults are defined with the safety annex for each sensor in the system. An example of a temperature sensor fault stuck at high is shown in Figure 3.

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 then in essence violate the guarantees of this component and possibly the assumptions of downstream components [20]. The activation of a fault is not up to the

```
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. 3. Fault on Temperature Sensor Defined in the Safety Annex for AADL

user, but instead left up to the backend model checker, JKind, to determine if the activation of this fault will cause violation of higher level guarantees. If so, it can be activated during the analysis.

For simplicity, throughout this paper we refer only to faults that fail high. This is to keep the example and results described concise. For ease of reference, a table is provided giving model elements of interest in the sensor example. We refer to these throughout this section. Note: the thresholds vary for pressure, temperature, and radiation. These are given as constants T_p, T_t, and T_r respectively. The shutdown command is defined notationally as S. 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 this analysis. Still messing around with how to display this in a way that it isn't messy, doesn't take up a ton of space, and am not currently happy with this approach. But I really hated the tables. Too much info for what is actually needed. Will keep working on this.

PWR System: $P = ((temp > T_t) \vee (pressure > T_p) \vee (radiation > T_r)) \iff S$

Temp Subsystem : $G_t = temp > T_t \iff S$

Pressure Subsystem: $G_p = pressure > T_p \iff S$

Radiation Subsystem: $G_r = radiation > T_r \iff S$

Temp Sensors (3): $g_p = pressure > T_p \iff S$, Fault f_{ti}: fails low for $i = 1, 2, 3$.

Pressure Sensors (3): $g_r = radiation > T_r \iff S$, Fault f_{pi}: fails low for $i = 1, 2, 3$.

Radiation Sensors (3): $g_r = radiation > T_r \iff S$, Fault f_{ri}: fails low for $i = 1, 2, 3$

3 Preliminaries

In this paper we consider *safety properties* over infinite-state machines. The states are vectors of boolean variables that define the values of state variables. We assume there are a set of legal *initial states* and the safety property is specified as a propositional formula over state variables. A *reachable state space* means that all states are reachable from the initial state.

Given a state space U, a transition system (I, T) consists of an initial state predicate $I : U \to bool$ and a transition step predicate $T : U \times U \to bool$. We define the notion of reachability for (I, T) as the smallest predicate $R : U \to bool$ 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 $P : U \to \mathit{bool}$ 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 $(I, T) \vdash P$.

3.1 Induction

This section is likely far longer than what it needs to be for this paper. We really just need a brief intro on induction before hitting sec 3.2. Will do a shortening - just wanted to write it all out for my own sake (and thesis sake). But I think we still need a paragraph linking transition systems to the SAT problem. Easy to do without all this verbage. For an arbitrary transition system (I, T), computing reachability can be very expensive or even impossible. Thus, we need a more effective way of checking if a safety property P is satisfied by the system. The key idea is to over-approximate reachability. If we can find an over-approximation that implies the property, then the property must hold. Otherwise, the approximation needs to be refined.

A good first approximation for reachability is the property itself. That is, we can check if the following formulas hold:

$$\forall s.\ I(s) \Rightarrow P(s) \tag{1}$$
$$\forall s, s'.\ P(s) \wedge T(s, s') \Rightarrow P(s') \tag{2}$$

If both formulas hold then P is *inductive* and holds over the system. If (1) fails to hold, then P is violated by an initial state of the system. If (2) fails to hold, then P is too much of an over-approximation and needs to be refined.

The JKind model checker used in this research uses k*-induction* which unrolls the property over k steps of the transition system. For example, 1-induction consists of formulas (1) and (2) above, whereas 2-induction consists of the following formulas:

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

That is, there are two base step checks and one inductive step check. In general, for an arbitrary k, k-induction consists of k base step checks and one inductive step check as shown in Figure 4 (the universal quantifiers on s_i have been elided for space). We say that a property is k-inductive if it satisfies the k-induction constraints for the given value of k. The hope is that the additional formulas in the antecedent of the inductive step make it provable.

$$\begin{gathered} I(s_0) \Rightarrow P(s_0) \\ \vdots \\ I(s_0) \wedge T(s_0, s_1) \wedge \cdots \wedge T(s_{k-2}, s_{k-1}) \Rightarrow P(s_{k-1}) \\ P(s_0) \wedge T(s_0, s_1) \wedge \cdots \wedge P(s_{k-1}) \wedge T(s_{k-1}, s_k) \Rightarrow P(s_k) \end{gathered}$$

Fig. 4. k-induction formulas: k base cases and one inductive step

In practice, inductive model checkers often use a combination of the above techniques. Thus, a typical conclusion is of the form "P with lemmas $L_1, \ldots, L_n$ is k-inductive".

3.2 The SAT Problem

Boolean Satisfiability (SAT) solvers attempt to determine if there exists a total truth assignment to a given propositional formula, that evaluates to TRUE. Generally, a propositional formula is any combination of the disjunction and conjunction of literals (as an example, a and $\neg a$ are literals). For a given unsatisfiable problem, solvers try to generate a proof of unsatisfiability; this is generally more useful than a proof of satisfiability. Such a proof is dependent on identifying a subset of clauses that make the problem unsatisfiable (UNSAT).

SAT solvers in model checking work over a constraint system to determine satisfiability. A *constraint system* C is an ordered set of n abstract constraints $\{C_1, C_2, ..., C_n\}$ over a set of variables. The constraint C_i restricts the allowed assignments of these variables in some way [21]. Given a constraint system, we require some method of determining, for any subset $S \subseteq C$, whether S is *satisfiable* (SAT) or *unsatisfiable* (UNSAT). When a subset S is SAT, this means that there exists an assignment allowed by all $C_i \in S$; when no such assignment exists, S is considered UNSAT.

There are several ways of translating a propositional formula into clauses such that satisfiability is preserved [22]. By performing this translation, k-inductive model checkers are able to utilize parallel SAT-solving engines to glean information about the proof of a safety property at each inductive step. Expression of the base and induction steps of a temporal induction proof as SAT problems is straightforward. As an example, we look at an arbitrary base case from Figure 4.

$$I(s_0) \land T(s_0, s_1) \land \cdots \land T(s_{k-2}, s_{k-1}) \land \neg P(s_{k-1})$$

When proving correctness it is shown that the formulas are *unsatisfiable*. If an n^{th} inductive-step is unsatisfiable, that means following an n-step trace where the property holds, there exists no next state where it fails, i.e., the property P is provable.

3.3 Background Information on Toolsuite

Ugh, still not happy with the placement of this section. But its here for now until we think of a better flow. I think each paragraph can also be shortened. The toolsuite used to perform these analyses are described in this section. 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) [23] to annotate the AADL model in order to perform verification using the back-end model checker `JKind` [24].

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" [25, 26]. Language annexes to AADL provide a richer set of modeling elements for various system design and analysis needs.The language definition is sufficiently rigorous to support formal analysis tools that allow for early phase error/fault detection.

Compositional Analysis Complex systems are usually composed from libraries of components. The specification of these systems are decomposed into properties of each individual component [15]. Compositional verification partitions 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. 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 [27]. 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 [23, 27, 28].

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 [23]. It is implemented as an AADL annex and is used to annotate AADL components with formal behavioral contracts. Each component's contracts can include assumptions and guarantees about the component's inputs and outputs respectively. A guarantee defines the nominal behavior of components and an assumption defines what the component expects from its environment. AGREE translates an AADL model and the behavioral contracts into Lustre [29] and then queries the `JKind` model checker to conduct the back-end analysis [24].

JKind JKind is an open-source industrial infinite-state inductive model checker for safety properties [24]. Models and properties in JKind are specified in Lustre [29], 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, 30]. In the Safety Annex approach, AGREE is used to define the nominal behavior of system components; faults are then woven 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.

4 Formalization

Section desperately needs figures of some kind to break up the text. Will try to make a few small ones of PWR example results. Compositional analysis proceeds from the top layer of the architecture down through the system model; faults are defined on leaf level components and guarantees are defined on all components. Due to the difference in analysis per layer, this section focuses on the formalism per layer type we are in.

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, each g is one of the transition constraints T_i where:

$$T_n = g_1 \wedge g_2 \wedge \cdots \wedge g_n \tag{3}$$

We 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. Any "faults" in a mid-layer are simply violated guarantees, or deviations from normal behavior.

4.1 Top Layer of Compositional Analysis

Since faults are defined at leaf layers of the architecture, the top (and middle) layers only contain guarantees in the analysis. The `All_MIVCs` algorithm collects all *minimal unsatisfiable subsets* (MUSs) of a given transition system in terms of the *negation* of the top level property [18, 19]. Formally, an MUS 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* (MCS); a MCS M of a constraint system C is a subset $M \subseteq C$ such that $C \setminus M$ is satisfiable and $\forall S \subset M : C \setminus S$ 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 top level constraint system: $MCS_1 = \{G_t\}$, $MCS_2 = \{G_p\}$, $MCS_3 = \{G_r\}$. 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 [31]. This duality is defined in terms of *Minimal Hitting Sets* (*MHS*). A hitting set of a collection of sets A is a set H such that every set in A is "hit" 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 [21, 31, 32].

For the PWR top level constraint system, it can be seen that each of the MCSs intersected with the MUS is nonempty. And now we have the minimal set of guarantees for which, if violated, will cause P to be unprovable.

4.2 Leaf Layer of Compositional Analysis

The faults in the safety annex are defined on leaf level components. Thus, for the lowest analysis layer, we must take into consideration faults and the guarantees their activation may violate. A fault $f \in F$ is a deviation from the normal constraint imposed by a guarantee. For the purposes of this paper, each guarantee at the leaf layer of analysis has an associated fault. 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 $gf_i \in GF$.

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

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:

$$T = gf_1 \wedge gf_2 \wedge \cdots \wedge gf_n \wedge \neg af_1 \wedge \neg af_2 \wedge \cdots \wedge \neg af_n \tag{4}$$

Lemma 1. *If $(I, T_n) \vdash P$ for T_n defined in equation 3, then $(I, T) \vdash P$ for T defined in equation 4.*

Proof. By application of successive evaluations of σ on each constrained activation literal $\neg af_i$, the result is immediate. $\square$

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 a SAT-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 $(I, T) \vdash P$. On the other hand, if it is *satisfiable*, then we know that given the constraints in GF and AF, P is not provable. These are the exact constraints 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.)

4.3 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 4.2). At the leaf layer, only activation literals are found in MCSs and 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 $MCS \subseteq AF$. Question: Does minimality need its own proof?

Lemma 2. *$\sigma(MCS)$ is a minimal 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. Minimality follows directly from the definition of MCS. □

In terms of the PWR example, the minimal cut sets for the temperature subsystem property G_t consist of all pairwise faults on the temperature sensors; if any two faults occur on the sensors at the same time, we violate the temperature subsystem guarantee.

Once these lower level minimal cut sets are generated, it is a matter of simple set replacement to find the higher level minimal cut sets. This can be easily seen in our example. An MCS at the top level has the element G_t. We systematically replace the contract with the faults that cause their violation. This results in three distinct minimal cut sets for P from the temperature subsystem: $\{f_{t1}, f_{t2}\}, \{f_{t1}, f_{t3}, \{f_{t2}, f_{t3}$. All minimal cut sets for P are given as similar pairwise combinations from each subsystem and total 9 for the entire system.

Seems I need a theorem to round it out: that replacement will give min cut sets of safety property. Will think about how to formulate this.

5 Implementation of the Algorithms

I have some ideas on this section - I want to change it to link back to the example a bit more. Also can simplify the algorithm somewhat. Wait to read this - it will be changing. Focus on the first 4 sections. The transformation of MIVCs to MinCutSets can only be performed 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 [21, 33, 34]. Thus, once all MIVCs have been found and the minimal hitting set algorithm has completed, the MinCutSet generation can begin.

The MinCutSet generation algorithm begins with a list of MCSs specific to a property. These MCSs may contain a mixture of fault activation literals constrained to *false* and subcomponent contracts constrained to *true*. We remove all constraints from each MCS and call the resulting sets I, for *Intermediate* set. 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. The reason is because at the lowest levels of the system, the only model elements used in the constraint system analyzed by the `All_MIVCs` algorithm are faults. Thus when the contracts at the lowest level are the safety properties for the `All_MIVCs` algorithm, the MUSs contain only faults (likewise the MCSs). When this cut set is obtained for the lowest level properties, it is stored in a lookup table keyed by the given property. Algorithm 1 describes this process.

Algorithm 1: MinCutSets Generation Algorithm

Function `replace`(P)**:**
- $List(I)$:= $List(MCS)$ for P with all constraints removed ;
- **for** *all* $I \in List(I)$ **do**
  - **if** *there exists contracts* $g \in I$ **then**
    - **for** *all constrained contracts* $g \in I$ **do**
      - **if** *there exists* $MinCutSets$ *for* g *in lookup table* **then**
        - **for** *all* $minCut(g)$ **do**
          - $I_{repl} = I$;
          - I_{repl} := replace g with $minCut(g)$;
          - add I_{repl} to $List(I)$;
      - **else**
        - replace(g) ;
  - **else**
    - add I as $minCut(g)$ for P ;

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

$$R \leq \sum_{i=1}^{\alpha} (\prod_{j=1}^{i} |Cut(g_j)|) \tag{5}$$

It is also important to note that the cardinality of $List(I)$ is bounded, i.e. the algorithm terminates. Every new I that is generated through some replacement of a contract with its minimal cut set is added to $List(I)$ in order to continue the replacement process for all contracts in I. Adding to this set requires proof regarding termination.

Theorem 1. *Algorithm 1 terminates*

Proof. No infinite sets are generated by the `All_MIVCs` or minimal hitting set algorithms [18, 34]; therefore, every MCS produced is finite. Thus, every $MinCutSet$ of every contract g is finite. Furthermore, a bound exists on the number of additional intermediate sets I that are added to $List(I)$: $|List(I)| \leq R$ (Equation 5). □

The reason for this upper bound is that for a contract g_1 in MCS, we make $|Cut(g_1)|$ replacements and add the resulting lists to $List(I)$. Then we move to the next contract g_2 in I. We must additionally make $|Cut(g_1)| \times |Cut(g_2)|$ replacements and add all of these resulting lists to $List(I)$, and so on throughout all contracts. Through the use of basic combinatorial principles, we end with the above formula for the upper bound on the number of additional intermediate sets.

Pruning to Address Scalability The MinCutSets are filtered during this process based on a fault hypothesis given before analysis begins. 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.

If the number of faults in an intermediate set I exceeds the threshold N, any further replacement of remaining contracts in that intermediate set can never decrease the total number of faults in I; 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 [35] 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 [36]. Unfortunately, due to the structure of BDDs, the worst case is exponential in size in terms of the number of variables [5, 35, 36]. 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 [37], but later extended to allow for all MinCutSet processing and generation without the use of BDDs [38]. 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, 38–42].

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 [38, 43]. These algorithms are implemented in xSAP [44] and COMPASS [45].

The model based safety assessment tool AltaRica 3.0 [46] 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 [47]. Other tools such as HiP-HOPS [48] 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) [49] 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

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 [50].

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. M. H. Liffiton, A. Previti, A. Malik, and J. Marques-Silva, "Fast, flexible MUS enumeration," *Constraints*, vol. 21, no. 2, pp. 223–250, 2016.
22. 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.
23. 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.
24. A. Gacek, J. Backes, M. Whalen, L. Wagner, and E. Ghassabani, "The JKind Model Checker," *CAV 2018*, vol. 10982, 2018.
25. AS5506C, "Architecture Analysis & Design Language (AADL)," Jan. 2017.

26. P. Feiler and D. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley Professional, 2012.
27. 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.
28. 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.
29. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, “The Synchronous Dataflow Programming Language Lustre,” in *IEEE*, vol. 79(9), 1991, pp. 1305–1320.
30. D. Stewart, M. Whalen, D. Cofer, and M. P. Heimdahl, “Architectural Modeling and Analysis for Safety Engineering,” in *IMBSA 2017*, 2017, pp. 97–111.
31. R. Reiter, “A theory of diagnosis from first principles,” *Artificial intelligence*, vol. 32, no. 1, pp. 57–95, 1987.
32. J. De Kleer and B. C. Williams, “Diagnosing multiple faults,” *Artificial intelligence*, vol. 32, no. 1, pp. 97–130, 1987.
33. 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.
34. 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.
35. R. E. Bryant, “Graph-based algorithms for boolean function manipulation,” *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.
36. A. Rauzy, “Binary decision diagrams for reliability studies,” in *Handbook of performability engineering*. Springer, 2008, pp. 381–396.
37. 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.
38. 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.
39. 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.
40. 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.
41. 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.
42. 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.
43. 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.
44. 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.
45. 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.
46. 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
47. 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.
48. 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.
49. M. Gudemann and F. Ortmeier, “A framework for qualitative and quantitative formal model-based safety analysis,” in *HASE 2010*, 2010.
50. D. Stewart, J. Liu, M. Whalen, D. Cofer, and M. Peterson, “Safety annex for AADL repository,” https://github.com/loonwerks/AMASE, 2018.