Composition of Fault Forests

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

[1] University of Minnesota, Minneapolis, MN, USA,
{dkstewar, mwwhalen, heimdahl}@umn.edu
[2] Collins Aerospace – Applied Research & Technology, Cedar Rapids, IA, USA,
{jing.liu, darren.cofer}@collins.com

Abstract. Safety analysis is used to ensure that critical systems operate within some level of safety when failures are present. As critical systems become more dependent on software components, it becomes more challenging for safety analysts to comprehensively enumerate all possible failure causation paths. Any automated analyses should be sound to sufficiently prove that the system operates within the designated level of safety. This paper presents a compositional approach to the generation of fault forests (sets of fault trees) and minimal cut sets. We use a behavioral fault model to explore how errors may lead to a failure condition. The analysis is performed per layer of the architecture and the results are automatically composed. A complete formalization is given. We implement this by leveraging minimal inductive validity cores produced by an infinite state model checker. This research provides a sound alternative to a monolithic framework. This enables safety analysts to get a comprehensive enumeration of all applicable fault combinations using a compositional approach while generating artifacts required for certification.

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 ubiquitous in our society. These systems are required to operate safely under nominal and faulty conditions. Proving that the system operates within some level of safety when failures are present 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 used during development and certification of critical systems [1]. Examples include *minimal cut sets* – each set represents the minimal set of faults that must all occur in order to violate a safety property and *fault trees* – the evaluation that determines all credible failure combinations which could cause an undesired top level hazard event. The fault tree can be transformed to an equivalent Boolean formula whose literals appear in the minimal cut sets. Since the introduction of minimal cut sets in the field of safety analysis, much research has been performed to address the generation of these sets and associated formulae [2–4]. As critical systems get larger, more minimal cut sets are possible with increasing cardinality. In recent years, symbolic model checking has been used to address scaling the analysis of systems with millions of minimal cut sets [5–7].

The state space explosion is a challenge when performing formal verification on industrial sized systems. This problem can arise from combining parallel processes together and attempting to reason monolithically over them. Compositional reasoning takes advantage of the hierarchical organizaton of a system model. A compositional approach verifies each component of the system in isolation and allows global properties to be inferred about the entire system [8]. The *assume-guarantee* paradigm is commonly used in compositional reasoning where the assumed behavior of the environment implies the guaranteed behavior of the component [9].

Using an assume-guarantee reasoning framework, we extend the definition of the nomimal transition system to allow for unconstrained guarantees. We use this idea to reason about all possible violations of a safety property per layer of analysis and then compose the results.

After we provide the formalization, we describe the implementation in the OSATE tool for the Architecture Analysis and Design Lanugage (AADL) [10]. AADL has two annexes that are of interest to us: the Assume-Guarantee Reasoning Environment (AGREE) [9] and the safety annex [11]. AGREE provides the assume-guarantee reasoning required for the transition system extension, and the safety annex allows us to define faults on component outputs. To implement the formalization, we look to recent work in formal verification. 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`) [12,13]. Inductive validity cores produce the minimal sets of model elements necessary to prove a property. Each set contains the behavioral contracts – the requirement specifications of components – used in a proof. We collect all MIVCs per layer to generate the minimal cut sets and thus the fault trees to be composed.

This paper presents a compositional approach to generating *fault forests* (sets of fault trees) and associated minimal cut sets, allowing us to reason uniformly about faults in various types of system components and their impact on system properties. The main contributions of this research include the formalization of the composition of fault forests and its implementation, enabling safety analysts to get a comprehensive enumeration of all applicable fault combinations. The resulting fault trees correspond with the system architecture and reflect the interface specifications developed in the system under consideration. Our objective is to provide safety engineers with verification tools so that they do not lose sight of the fault forest for the trees.

The organization of the paper is as follows. Section 2 describes a running example, Section 3 outlines the formalization of this approach. The implementation is discussed in Section 4 and related work follows in Section 5. The paper ends with a conclusion and discussion of future work.

2 Running Example

In a typical Pressurized Water Reactor (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 in turn 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 simplified version of a subsystem within a PWR that monitors these factors. Each subsystem contain three sensors that monitor pressure, temperature, and radiation. If any of these conditions are too high, a shut down command is sent from the sensors to the parent components. The temperature, pressure, and radiation sensor subsystems each contain three associated sensors for redundancy. Each sensor reports the associated environmental condition to a majority voter component. If the majority of the sensors reports high, a shut down command is sent to the subsystem. If any subsystem reports a shut down command, the top level system will shut down. Pressure, radiation, and temperature all have associated thresholds for high values which we refer to as T_p, T_r, and T_t respectively. The safety properties P_i of interest in this system is: *if an environmental threshold is surpassed, then we shut down the system.* The specifications of these properties are shown at the top of Figure 1.

For reference throughout this paper, we provide Figure 1 which shows the guarantees and faults of interest for this running example. We do not show all guarantees and assumptions that are in the model, but only the ones of interest for the illustration.

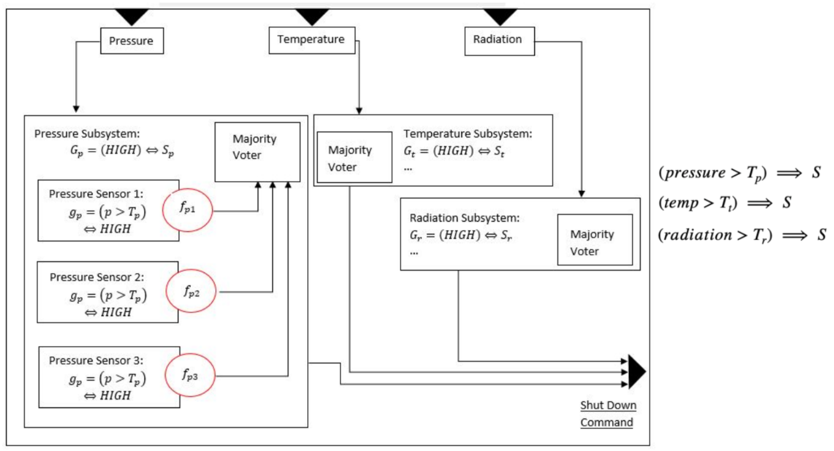

Pressure
Temperature
Radiation
Pressure Subsystem: $G_p = (HIGH) \Leftrightarrow S_p$
Majority Voter
Pressure Sensor 1: $g_p = (p > T_p) \Leftrightarrow HIGH$
f_{p1}
Pressure Sensor 2: $g_p = (p > T_p) \Leftrightarrow HIGH$
f_{p2}
Pressure Sensor 3: $g_p = (p > T_p) \Leftrightarrow HIGH$
f_{p3}
Majority Voter
Temperature Subsystem: $G_t = (HIGH) \Leftrightarrow S_t$ ...
Radiation Subsystem: $G_r = (HIGH) \Leftrightarrow S_r$ ...
Majority Voter
Shut Down Command
$(pressure > T_p) \implies S$
$(temp > T_t) \implies S$
$(radiation > T_r) \implies S$

Fig. 1. Sensor System Nominal and Fault Model Details

3 Formalization

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:

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

A safety property $P : U \to bool$ 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 $(I, T) \vdash P$. We assume the transition relation has the structure of a top level conjunction. Given $T(u, u') = T_1(u, u') \wedge \cdots \wedge T_n(u, u')$ we will write $T = \wedge_{i=1..n} T_i$ for short. By further abuse of notation, T is identified with the set of its top-level conjuncts $\wedge_{i=1..n} T_i$. Thus, $T_i \in T$ means that T_i is a top-level conjunct of T, and $S \subseteq T$ means all top level conjuncts of S are top-level conjuncts of T.

The set of all nominal guarantees of the system G consists of conjunctive constraints $g \in G$. Given no faults (i.e., nominal system) and a transition relation T consisting of conjunctive constraints T_i, each g is one of the transition constraints T_i where:

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

We consider an arbitrary layer of analysis of the architecture and assume the property holds of the nominal relation $(I, T) \vdash P$. Let the set of all faults in the system be denoted as F. A fault $f \in F$ is a modification of the nominal 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. We extend the transition system so that we can view the system behavior in the presence of faults—or equivalently the absence of nominal constraints. 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 $gf_i \in GF$.

$$gf_i = \textit{if } af_i \textit{ then } f_i \textit{ else } g_i$$

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 $(I,T) \vdash P$ for T defined in equation 1, then $(I,T') \vdash P$ for T' defined in equation 2.*

Proof. By the mapping of each constrained activation literal $\neg af_i$ to the associated guarantee g_i and the constraint of the activation literals to be false, 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 an SMT solver for satisfiability during the model checking engine procedures.

If the $af_i \in AF$ defined in T' are unconstrained, this allows more behaviors to the transition system and could cause a violation of P. If so, a counterexample may be produced. For each counterexample, we can partition AF into two sets that we call *non-faulty variables (NFV)* and *faulty variables (FV)*. The set NFV consists of a set of activation literals that are constrained to be false throughout the counterexample, and FV contains those that can be non-deterministically assigned any valuation at some point in the trace. By mapping some of the variables in AF to false, we know that their associated guarantees in GF are non-faulty for all considered executions. We define $T'(NFV)$ as a relaxation of T' (2):

$$T'(NFV) = gf_1 \wedge gf_2 \wedge \cdots \wedge gf_n \wedge \bigwedge \{\neg af_i | af_i \in NFV\}$$

The activation literals constrained to be false in $T'(NFV)$ indicate that their associated guarantees to be valid. In the remainder of this section, we assume that all $af_i \in AF$ are unconstrained and when given a true valuation will lead to a violation of the associated guarantee. This violation causes the output that the guarantee constrains to become non-deterministic. The Boolean variables in FV correspond to Boolean variables in the fault tree.

Definition 1. *A fault tree FT is a pair $(r, \mathcal{L})$ where:*

r: the root r is a negated desirable property,
$\mathcal{L}$: a Boolean equation whose literals are faulty variables.

All literals af of the Boolean equation $\mathcal{L}$ are elements of the set FV. A fault tree may correspond to a single layer of the system architecture where the root r is a violated guarantee or a violated safety property depending on the parent component under analysis. The tree may also describe the relationship between faults and multiple layers of the system architecture. The root r still corresponds to a violated guarantee or property, but the structure of the Boolean formula $\mathcal{L}$ will reflect the layers of the system architecture. If r is a violated safety property, then $r \in P$. If r is a violated guarantee for some lower level parent component, then $r \in \pi$, where π is the set of parent component guarantees.

Definition 2. *A fault tree $FT = (r, \mathcal{L})$ is valid if and only if a true valuation for r and for all $af \in \mathcal{L}$ is satisfiable given the respective transition system constraints.*

The hierarchy of the fault tree is dependent on the associated Boolean formula. A more intuitive structure is that of *disjunctive normal form* (DNF) as seen in both fault trees depicted in Figure 2, but DNF is not required under our definition of a fault tree.

Traditionally, a safety property is a property of the system and in the assume-guarantee reasoning environment is a top level guarantee. In the following formalism, each layer of analysis is viewed as distinct from the system hierarchy as the proof is being constructed, and the properties we wish to prove are guarantees of a component. We use the notation P to refer to the set of all parent properties at a given layer of analysis. If the analysis is being performed at the top level, these are all safety properties of the system. If the analysis is being performed at an intermediate level, these are all guarantees of the parent component.

A goal of compositional safety analysis is to reflect failures of leaf and intermediate components at the top level. Not all guarantees must be valid to prove a parent level guarantee. To this end, we wish to make a distinction between all guarantees of a component and those that are required to prove parent guarantees. The subset π of P are the guarantees that must be valid to prove the guarantees of a parent component. These are the critical guarantees of a component. Given that there may be multiple safety properties and multiple intermediate level guarantees, we do not compose single fault trees per layer, but rather forests of trees.

Definition 3. *A fault forest FF is a set of fault trees.*

Definition 4. *A fault forest FF is valid if and only if for all $FT \in FF$, the fault tree FT is valid as per Definition 2.*

The goal of this formalization is to show that the composition of fault forests results in a valid fault forest. First, we assume we can derive all minimal counterexamples to the proof of a property (or guarantee) at any layer of compositional assume-guarantee analysis. Then we prove that after composition, the tree we obtain is a fault tree describing the system in the presence of faults. In Section 4, we discharge the assumption and show how we derive a valid fault forest for each layer of analysis. Since a fault forest is only valid with respect to the transition system from whence it came, we will now iteratively extend the model with each composition step.

Components and Their Composition: To prove each parent component guarantee $\pi_i \in \pi$, a certain subset of child guarantees are required to be non-faulty, i.e., the associated activation literals are given a false valuation. We use the set NFV to denote the non-faulty variables of the children components that are required to prove parent guarantees π. These non-faulty variables are used in the relaxation of T' (Equation 2). This can be stated as $(I, T'(NFV)) \vdash \pi$.

The violation of certain child guarantees may lead to the violation of a parent guarantee π_i. The activation literals of the child are given a true valuation and are denoted as FV: faulty variables. A set of faulty variables of the children components contain the activation literals that correspond to leaves of a fault tree $\mathcal{L}$ with the root $r = \neg\pi_i$ for parent guarantee π_i. In other words, the fault tree $FT_i \in FF$ is associated with a property π_i. The non-faulty variables NFV contain the valid child guarantees that are required to prove π_i, and the fault tree FT_i reflects the child guarantee violations that may lead to the violation of π_i.

Definition 5. *A component is the tuple $Comp(M, FF, NFV, \pi)$ where:*

- *M: the model consisting of the set of all children properties P_c extended with non-deterministic faults: $gf_i \in P_c$ where $gf_i =$ if af_i then f_i else g_i,*

- *FF: the ordered set of fault trees for this component,*
- *NFV: the set of non-faulty variables, $NFV \subseteq P_c$,*
- *π: the ordered set of properties $\pi \subseteq P$ such that $(I, T'(NFV)) \vdash \pi$, i.e., all properties π hold if the variables in NFV are given a true valuation.*

and $FT_i \in FF$ corresponds to $\pi_i \in \pi$ for each of the i properties: the root of FT_i is $\neg\pi_i$.

Given the definition of a component, we now discuss what it means to compose components. Each layer of composition moves iteratively closer to a monolithic model by the enlargement of each set described in a component. To begin this iterative process, we define the composition of fault forests. To show that the composition of fault trees results in a valid fault tree, let ϕ be a function $\phi : B \times B \to B$ for Boolean equations B. We use this mapping to define the composition of parent component fault tree FT_p and child component fault tree FT_c, where $FT_c = (r_c, \mathcal{L}_c)$ and $FT_p = (r_p, \mathcal{L}_p)$.

$$FT_c \circ FT_p = \phi(FT_c, FT_p) = \begin{cases} (r_p, \mathcal{L}_p(r_c, \mathcal{L}_c)) & r_c \in \mathcal{L}_p \\ (r_p, \mathcal{L}_p) & r_c \notin \mathcal{L}_p \end{cases} \tag{3}$$

where $\mathcal{L}_p(r_c, \mathcal{L}_c)$ is the replacement of af_{r_c} in $\mathcal{L}_p$ with $(r_c, \mathcal{L}_c)$. Intuitively, each of the violated guarantees has an associated activation literal. If an activation literal is found in the parent leaf equation $\mathcal{L}_p$, replace that activation literal (af_{r_c}) with the associated violated child guarantee (r_c).

Let n be the number of properties for some parent component p and let m be the number of properties for some child component c. Then the parent fault forest FF_p is a mapping $FF_p : S_1 \to B$ for $S_1 = \{1, 2, \ldots, m\}$ and the set of Boolean equations B and $FF_c : S_2 \to B$ for $S_2 = \{1, 2, \ldots n\}$. And let ϕ_F be a function $\phi_F : seq(B) \times seq(B) \to seq(B)$ for finite sequences of Boolean equations $seq(B)$. We use this function to define the composition of parent and child component fault forests $FF_p = \{(r_{p1}, \mathcal{L}_{p1}), \ldots, (r_{pm}, \mathcal{L}_{pm})\}$ and $FF_c = \{(r_{c1}, \mathcal{L}_{c1}), \ldots, (r_{cn}, \mathcal{L}_{cn})\}$. ϕ_F is a mapping such that for all $i \in S_1$ and for all $j \in S_2$:

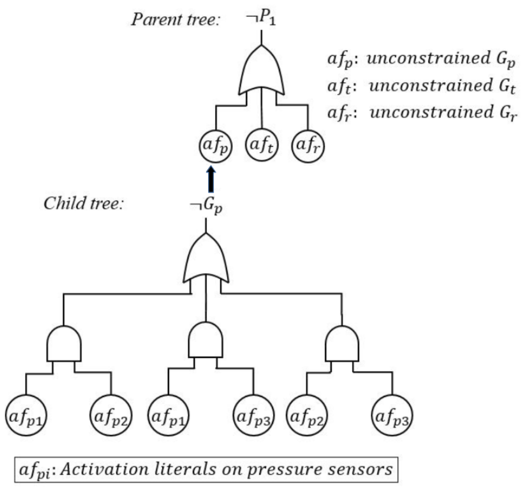
$$FF_c \circ FF_p = \phi_F(FF_c, FF_p) = \begin{cases} (r_{pi}, \mathcal{L}_{pi}(r_{cj}, \mathcal{L}_{cj})) & r_{cj} \in \mathcal{L}_{pi} \\ (r_{pi}, \mathcal{L}_{pi}) & r_{cj} \notin \mathcal{L}_{pi} \end{cases} \tag{4}$$

where $\mathcal{L}_{pi}(r_{cj}, \mathcal{L}_{cj})$ is the replacement of $af_{r_{cj}}$ in $\mathcal{L}_{pi}$ with $(r_{cj}, \mathcal{L}_{cj})$.

Each literal in the formula $\mathcal{L}_p$ is a fault activation literal af_i. If af_i has its associated guarantee gf_i in the set of child roots r_c, then the mapping ϕ_F will extend af_i in $\mathcal{L}_p$ with the leaf formula of the child root gf_i. The resulting fault forest is a sequence of fault trees $FF = \{(r_{pk}, \mathcal{L}_k) : k = 1, \ldots, m\}$. The roots of the resulting forest are the same roots as the parent forest while the leaf formulae may change based on replacement.

We return to the sensor system example to illustrate this mapping. Graphically, this is represented in Figure 2. The top level (parent) component is defined as: $Comp_p(M_p, FF_p, NFV_p, \pi_p)$ and $FF_p = \{(\neg P, af_p \vee af_t \vee af_r)\}$ where each activation literal is associated with the unconstrained guarantees G_p, G_t, and G_r. The child layer has a fault forest consisting of three fault trees, one for each subsystem. The pressure subsystem fault tree is $FT_p = (\neg G_p, (af_{p1} \wedge af_{p2}) \vee (af_{p1} \wedge af_{p3}) \vee (af_{p2} \wedge af_{p3})$. The leaf formulae for each subsystem tree corresponds to pairwise combinations of active sensor faults. We now show the composition of the pressure subsystem child and top level parent fault trees.

The mapping ϕ_F iterates through each tree in the parent forest – in this case, we have only one. Then for each parent tree it iterates through the Boolean literals in $\mathcal{L}$. If there is a match between



Parent tree:
$\neg P_1$
af_p: unconstrained G_p
af_t: unconstrained G_t
af_r: unconstrained G_r
af_p
af_t
af_r
Child tree:
$\neg G_p$
af_{p1}
af_{p2}
af_{p1}
af_{p3}
af_{p2}
af_{p3}
af_{pi}: Activation literals on pressure sensors

Fig. 2. Sensor System Composition of Fault Trees

a child root and a parent leaf, the replacement is made. We represent the unconstrained (violated) guarantee as $\neg G_p$ and it is associated with the fault activation literal af_p. Thus, af_p will be extended with $\{\neg G_p, (af_{p1} \wedge af_{p2}) \vee (af_{p1} \wedge af_{p3}) \vee (af_{p2} \wedge af_{p3})\}$. This extension is done for each leaf formula in $\mathcal{L}_p$ from the parent fault forest. The end result of the replacement is easy to see in Figure 2.

We have provided the foundational definitions necessary to discuss what it means to compose components. The composition of child component $Comp_c$ and parent component $Comp_p$ is defined as:

Definition 6. *$Comp_c(M_c, FF_c, NFV_c, \pi_c) \circ Comp_p(M_p, FT_p, NFV_p, \pi_p)$ $= Comp_\circ(M', FF', NFV', \pi')$ where:*

- *$M' = M_c \cup M_p$ is the iterative enlargement of the model by combining children guarantees with parent guarantees,*
- *$FF_c \circ FF_p$ is the composed fault forest,*
- *$NFV' = NFV_c \cup NFV_p$ is the set of non-faulty variables,*
- *$\pi' = \pi_c \cup \pi_p$ are valid properties such that $(I, T'(NFV')) \vdash \pi'$.*

The enlargement of the model, M', iteratively flattens the composed layers by taking the union of children guarantees and parent guarantees. The fault forests are composed into a set of fault trees describing the enlarged model. The non-faulty variables from child and parent are combined into a set NFV' such that $(I, T'(NFV')) \vdash \pi'$. Given that in child and parent components, the properties π can be derived from the non-faulty variables, we show that this relationship holds after composition. To state $(I, T'(NFV)) \vdash \pi$, we use the shorthand $NFV \vdash \pi$.

Theorem 2. *If $NFV_c \vdash \pi_c$ and $NFV_p \vdash \pi_p$, then $NFV' \vdash \pi'$*

Proof. Assume antecedent. Let $p' \in \pi'$. If $p' \in \pi_c$ then $NFV_c \vdash p'$ and likewise if $p' \in \pi_p$, then $NFV_p \vdash p'$. In either case, $NFV_c \cup NFV_p = NFV' \vdash \pi'$. $\square$

Composition of Fault Trees and Forests: We work under the *monotonicity assumption*, commonly adopted in safety analysis, that an additional fault cannot cancel the effect of existing faults. Without this assumption, we cannot show that the resulting fault tree is valid. Given Definition 2, we show that the composition of two fault trees results in a valid fault tree. We will then extend this to show that the composition of two fault forests results in a valid fault forest.

Lemma 1. *If FT_c and FT_p are valid fault trees, then their composition $\phi(FT_c, FT_p)$ is also a valid fault tree.*

Proof. Assume the antecedent. Then $(r_c, \mathcal{L}_c)$ is satisfiable with regard to the child component transition system and all $af \in \mathcal{L}_c$ and r_c are given true valuations.

Case 1: If the child root $\neg g_i$ does not have an associated $af_i \in \mathcal{L}_p$, then $\phi(FT_c, FT_p) = FT_p$ and the inclusion of the additional constraints from the child transition system in M_c does not negate the effects of the faults in FT_p. Thus, it is a valid fault tree.

Case 2: If the child root $\neg g_i$ has an associated $af_i \in \mathcal{L}_p$, then af_i has a true valuation. Given the mapping defined between guarantees and activation literals, replacement of $af_i \in \mathcal{L}_p$ with $\neg g_i$ preserves satisfiability. Furthermore, by the monotonicity assumption, the addition of more constraints ($af \in \mathcal{L}_c$) to the Boolean formula does not change satisfiability in the extended transition system.

In all cases, $\phi(FT_c, FT_p)$ is a valid fault tree. □

Lemma 2. *If FF_c and FF_p are valid fault forests, then their composition $\phi(FF_c, FF_p)$ is also a valid fault forest.*

Proof. Assume the antecedent. Then for all $FT_j \in FF_p$ and $FT_i \in FF_c$, FT_i and FT_j are valid fault trees as per Definition 4. For each iteration defined in the mapping ϕ_F, apply Lemma 1 and the monotonicity assumption.

□

We have shown that a single layer of composition produces valid fault forests. To perform this analysis across n layers of architecture we use induction to show that the resulting fault forest is valid. The notation ϕ_F^n indicates the iterated function ϕ_F which is a successive application of ϕ_F with itself n times. Assume the fault forest FF_0 is obtained at the leaf level of the architecture.

Theorem 3. *If $\phi_F^n(FF_{n-1}, FF_n)$ is a valid fault forest, then $\phi^{n+1}(FF_n, FF_{n+1})$ is a valid fault forest.*

Proof. Base case: Each fault forest per layer is valid by construction. By Lemma 2, $\phi_F(FF_0, FF_1)$ is a valid fault forest.

Inductive assumption: Assume $\phi_F^n(FF_{n-1}, FF_n)$ is a valid fault forest.

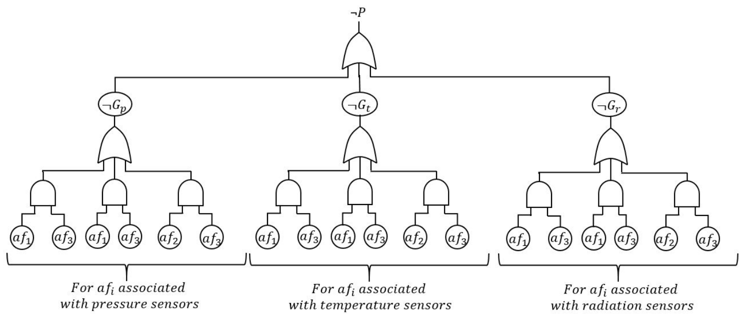
$$\begin{aligned} \phi_F^{n+1}(FF_n, FF_{n+1}) &= ((FF_0 \circ FF_1) \circ FF_2) \circ \cdots \circ FF_n) \circ FF_{n+1})) \\ &= \phi_F^n(FF_{n-1}, FF_n) \circ FF_{n+1} \end{aligned}$$

By inductive assumption and Lemma 2, $\phi_F^{n+1}(FF_n, FF_{n+1})$ is a valid fault forest.

□

After applying these techniques to the pressurized water reactor example, the resulting fault forest consists of one tree associated with the single top level safety hazard as shown in Figure 3.

In this section, we have formalized the idea that fault trees (and forests) can be composed without losing the validity of each composed tree. We proved that this can be performed iteratively across an arbitrary number of layers.



$\neg P$
$\neg G_p$
$\neg G_t$
$\neg G_r$
af_1
af_3
af_1
af_3
af_2
af_3
For af_i associated with pressure sensors
For af_i associated with temperature sensors
For af_i associated with radiation sensors

Fig. 3. Sensor System Fault Forest

4 Implementation

To implement the formalism described in Section 3, we must compute minimal cut sets per layer of analysis, transform them into their related Boolean formula, and compose them. As previously described, Ghassabani et al. developed the *all minimal inductive validity core* algorithm (`All_MIVCs`) [12, 13]. The `All_MIVCs` algorithm 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 for the property. Thus, if we violate at least one contract from every MIVC set, we have in essence "broken" every proof. The idea is that the hitting sets of all MIVCs produces the minimal cut sets.

4.1 Formal Background

JKind is an open-source industrial infinite-state inductive model checker for safety properties [14]. Models and properties in JKind are specified in Lustre [15], 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.

Each step of induction is sent to an SMT (Satisfiabilty Modulo Theory)-solver to check for *satisfiability*, i.e. there exists a total truth assignment to a given formula that evaluates to true. If there does not exist such an assignment, the formula is considered *unsatisfiable*. 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. Expression of the base and induction steps of a temporal induction proof as SAT problems is straightforward and is shown below for step k:

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

When proving correctness it is shown that the formulas are *unsatisfiable*, i.e., the property P is provable. The idea behind finding an *inductive validity core* (IVC) for a given property P is based on inductive proof methods used in SMT-based model checking, such as k-induction and IC3/PDR [16]. Generally, an IVC computation technique aims to determine, for any subset $S \subseteq T$, whether P is provable by S. A minimal subset that satisfies P is seen as a minimal proof explanation and called a minimal inductive validity core.

Definition 7. *Inductive Validity Core (IVC) [12]: $S \subseteq T$ for $(I, T) \vdash P$ is an Inductive Validity Core, denoted by $IVC(P, S)$, iff $(I, S) \vdash P$.*

Definition 8. *Minimal Inductive Validity Core (MIVC) [13]: $S \subseteq T$ is a minimal Inductive Validity Core, denoted by $MIVC(P, S)$, iff $IVC(P, S) \wedge \forall T_i \in S.\ (I, S \setminus \{T_i\}) \not\vdash P$.*

The *constraint system* consists of the constrained formulas of the transition system and the negation of the property. The `All_MIVCs` algorithm collects all *minimal unsatisfiable subsets* (MUSs) of a constraint system generated from a transition system at each induction step [13, 17].

Definition 9. *A Minimal Unsatisfiable Subset (MUS) 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 (the disjunction of these guarantees) is provable.

In the case of an UNSAT system, we may ask: what will correct this unsatisfiability? A related set answers this question:

Definition 10. *A Minimal Correction Set (MCS) M of a constraint system C is a subset $M \subseteq C$ such that $C \setminus M$ is satisfiable and $\forall M' \subset M : C \setminus M'$ is unsatisfiable.*

An MCS can be seen to "correct" the infeasability of the constraint system by the removal from C the constraints found in an MCS. Returning to the PWR example, the MCSs of the constraint system C are $MCS_1 = \{G_t\}$, $MCS_2 = \{G_p\}$, $MCS_3 = \{G_r\}$. If any single guarantee is violated, a shut down from that subsystem may not get sent when it should and the safety property P will be violated. This corresponds exactly to the definition of a minimal cut set.

For the following definitions, we remind readers of the extended transition system defined in Equation 2 of Section 3 and that the elements of T' are the set $GF \cup AF$ for potentially faulty guarantees GF and activation literals AF. We use the notation $af \rightarrow \{true, false\}$ to indicate a constraint on the literal af.

Definition 11. *Given a constraint system C, a cut set S of a top level event $\neg P$ is a set $S \subseteq AF \subseteq C$ such that $\forall af \in S$, $af \rightarrow \{true\}$ and $S \cup \{\neg P\}$ is satisfiable in C.*

Intuitively, a cut set is a true valuation for some subset of fault activation literals within a constraint system containing such that the constraint system is satisfiable given those true valuations and the violation of a safety property.

Definition 12. *A cut set S is minimal if and only if $\forall af \in S$, $S \setminus \{af\} \cup \{\neg P\}$ is unsatisfiable.*

Our approach in computing minimal cut sets through the use of inductive validity cores is to supply activation literals constrained to be false to the algorithm. The resulting MCSs consist of elements $\neg af_i$. The removal of this constraint from the constraint system results in nondeterministically true activation literals. By the definition of an MCS, we know that $C \setminus MCS$ is satisifiable. This removal of constraints from C removes the *false* constraint from each element in the MCS. Liffiton et. al showed that any subset of a satisfiable set is also satisfiable [18], so we know that for set S consisting of elements of MCS with constraints removed, $S \cup \{\neg P\}$ is also satisfiable. This is the definition of a cut set. Minimality comes directly from the definition of a minimal correction set.

A duality exists between the MUSs of a constraint system and the MCSs as established by Reiter [19]. This duality is defined in terms of *Minimal Hitting Sets* (*MHS*).

Definition 13. *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. This is noted in previous work [18, 20] and the proof of such is given by Reiter (Theorem 4.4 and Corollary 4.5) [19].

4.2 Algorithm Implementation

The algorithms in this paper are implemented in the Safety Annex [11] for the Architecture Analysis and Design Language (AADL) [21] and require the Assume-Guarantee Reasoning Environment (AGREE) [9] to annotate the AADL model in order to perform verification using the back-end model checker `JKind` [14]. For more information on the application of the safety annex in practice, see previous work [11, 22, 23].

In the formalism, any guarantee in the model had an associated fault activation literal and could be unconstrained. In the implementation, we rely on the fault model created in the safety annex to dictate which output constraints are modified (i.e., which guarantees can be violated) and how they are modified. A user may define multiple, single, or no faults on a single output. Each explicit fault defined in the safety annex is added to the Lustre program as are assocated fault activation literals [11, 23]. This corresponds to the f_i and af_i described in Section 3.

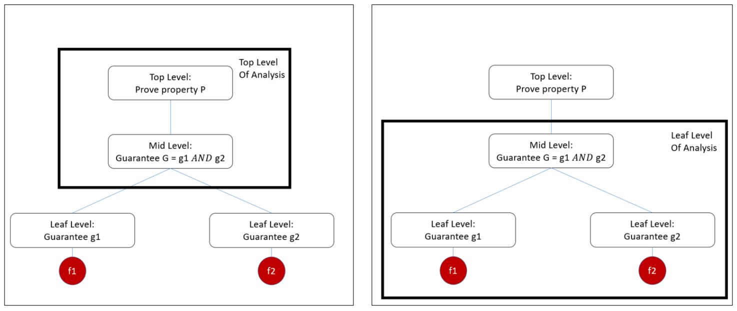


Fig. 4. Illustration of Two Layers of Analysis

The `All_MIVCs` algorithm requires specific equations in the Lustre model to be flagged for consideration in the analysis; these we call *IVC algorithm elements*. All equations in the model can be used as IVC algorithm elements or one can specify directly the equations to consider. In this implementation, the IVC algorithm elements are added differently depending on the layer. In the leaf architectural level, fault activation literals are added to the IVC algorithm elements and are constrained to *false*. In middle or top layers, supporting guarantees are added. This is shown in Figure 4. The figure shows an arbitrary architecture with two analysis layers: top and leaf. The top layer analysis adds G as IVC algorithm element; the leaf layer analysis adds f_1 and f_2.

A requirement of the hitting set algorithm is that to find all MCSs, all MUSs must be known. Ghassabani et al. [13] showed that finding all MIVCs is as hard as model checking. Once the MIVC analysis is complete for a property at a given layer, a hitting set algorithm is used to generate the related MCSs [24]. Depending on the layer of analysis, the MCSs contain either guarantees (mid layer) or fault activation literals (leaf layer).

The composition of these results is performed top down and shown in Algorithm 1. For each guarantee found in an MCS, a replacement is made with the guarantee's own MCSs. This is done

Algorithm 1: Compose Results

```
1  $R \leftarrow \texttt{All\_MCSs}(P) = \vee_{i=1}^{n} MCS_i$
2  where $MCS_i = \wedge_{j=1}^{m} gf_j$
3  Function resolve ($R$):
4      for $\forall$ OR-node in $R$ do
5          for $\forall gf_j$ in OR-node do
6              if $\exists MCS(gf_j)$ then
7                  $R \leftarrow$ replace $gf_j$ in $R$ with $\texttt{All\_MCSs}(gf_j)$;
8                  resolve ($\texttt{All\_MCSs}(gf_j)$);
9              else
10                 $R \leftarrow$ replace $gf_j$ in $R$ with $af_j$;
11     convert $R$ to DNF
```

recursively until all replacements have been made (line 7, 8 of Algorithm 1). If on the other hand there are no MCSs for a given guarantee, that guarantee is replaced by its associated fault activation literal (line 10). At the leaf level of analysis, no guarantees have associated MCSs (there are no children properties) and thus reaches the end of recursion. At that time, the formula is converted back into disjunctive normal form of fault activation literals to finish the translation into the traditional fault tree (line 11). The fault tree that is produced has a depth associated with the architecture of the system model. The gates supported in this tool include *And* and *Or* gates.

Theorem 4. *Algorithm 1 terminates*

Proof. No infinite sets are generated by the `All_MIVCs` or minimal hitting set algorithms [13, 25]; therefore, for all g_i in the model, $\texttt{All\_MCSs}(g_i)$ is a finite set and $MCS(g_i)$ is a finite set. Each call to `Resolve` processes a guarantee that was not previously resolved, and for all g_i at the leaf layer of analysis, $\texttt{All\_MCSs}((g_i) = \emptyset$. Given that there are finite layers in a model, the algorithm terminates. □

5 Related Work

Minimal cut sets generated by monolithic analysis look at explicitly defined faults throughout the architecture and attempt through various techniques to find the minimal violating set for a particular property. We now outline some of the common monolithic approaches to minimal cut set generation.

The representation of Boolean formulae as Binary Decision Diagrams (BDDs) was first formalized in the mid 1980s [26] and was extended to the representation of fault trees not many years later [27]. 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 minimal cut sets. Unfortunately, due to the structure of BDDs, the worst case is exponential in size in terms of the number of variables. In industrial sized systems, this is not realistically useful.

SAT based computation was introduced to address scalability problems in the BDD approach; initially it was used as a preprocessing step to simplify the decision diagram [28], but later was extended to allow for all minimal cut set processing and generation without the use of BDDs [29]. Since then, much research has focused on leveraging the power of model checking in the problems of safety assessment, e.g., [5, 6, 11]. 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 [29]. These algorithms are implemented in xSAP [30] and COMPASS [31]. Another related work is contract based safety analysis performed using the OCRA tool [32]. By contrast, this research performs the minimal cut set computations in a purely compositional fashion.

The model based safety assessment tool AltaRica 3.0 [33] performs a series of processing to transform the model into a reachability graph and then compile to Boolean formula in order to compute the minimal cut sets. Other tools such as HiP-HOPS [34] 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) [35] 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 minimal cut set generation.

To our knowledge, a fully compositional approach to generating fault forests or minimal cut sets has not been introduced.

6 Conclusion and Future Work

We presented a formalism that defines the composition of fault forests by extending the transition system to allow for fault activation literals. This formalism is implemented by leveraging recent research in model checking techniques. Using the idea of minimal inductive validity cores (MIVCs), which are the minimal model elements necessary for a proof of a safety property, we are able to provide fault activation literals as model elements to the `All_MIVCs` algorithm which provides all the MIVCs 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 graphical hierarchical fault trees as well as perform scalability studies that compare this approach with other non-compositional approaches to minimal cut set generation.

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

References

1. SAE ARP4754A, "Guidelines for Development of Civil Aircraft and Systems," December 2010.
2. W. Vesely, F. Goldberg, N. Roberts, and D. Haasl, "Fault tree handbook," tech. rep., Technical report, US Nuclear Regulatory Commission, 1981.
3. 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.
4. C. Ericson, "Fault tree analysis - a history," in *Proceedings of the 17th International Systems Safety Conference*, 1999.
5. 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.
6. A. Schäfer, "Combining real-time model-checking and fault tree analysis," in *International Symposium of Formal Methods Europe*, pp. 522–541, Springer, 2003.
7. M. Bozzano, A. Cimatti, and F. Tapparo, "Symbolic fault tree analysis for reactive systems," in *ATVA*, 2007.
8. S. Berezin, S. Campos, and E. M. Clarke, "Compositional reasoning in model checking," in *International Symposium on Compositionality*, pp. 81–102, Springer, 1997.
9. D. D. Cofer, A. Gacek, S. P. Miller, M. W., B. LaValley, and L. Sha, "Compositional Verification of Architectural Models," in *NFM 2012*, vol. 7226, pp. 126–140, April 2012.

10. P. Feiler and D. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley Professional, 2012.
11. 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.
12. E. Ghassabani, A. Gacek, and M. W. Whalen, "Efficient generation of inductive validity cores for safety properties," *CoRR*, vol. abs/1603.04276, 2016.
13. 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.
14. A. Gacek, J. Backes, M. Whalen, L. Wagner, and E. Ghassabani, "The JKind Model Checker," *CAV 2018*, vol. 10982, 2018.
15. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The Synchronous Dataflow Programming Language Lustre," in *IEEE*, vol. 79(9), pp. 1305–1320, 1991.
16. T. Kahsai, P.-L. Garoche, C. Tinelli, and M. Whalen, "Incremental verification with mode variable invariants in state machines," in *NASA Formal Methods Symposium*, pp. 388–402, Springer, 2012.
17. 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.
18. M. H. Liffiton, A. Previti, A. Malik, and J. Marques-Silva, "Fast, flexible MUS enumeration," *Constraints*, vol. 21, no. 2, pp. 223–250, 2016.
19. R. Reiter, "A theory of diagnosis from first principles," *Artificial intelligence*, vol. 32, no. 1, pp. 57–95, 1987.
20. J. De Kleer and B. C. Williams, "Diagnosing multiple faults," *Artificial intelligence*, vol. 32, no. 1, pp. 97–130, 1987.
21. AS5506C, "Architecture Analysis & Design Language (AADL)," Jan. 2017.
22. D. Stewart, J. J. Liu, D. Cofer, M. Heimdahl, M. W. Whalen, and M. Peterson, "Aadl-based safety analysis using formal methods applied to aircraft digital systems," *Reliability Engineering & System Safety*, vol. 213, p. 107649, 2021.
23. D. Stewart, M. Whalen, D. Cofer, and M. P. Heimdahl, "Architectural modeling and analysis for safety engineering," in *IMBSA 2017*, pp. 97–111, 2017.
24. 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.
25. 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.
26. R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.
27. A. Rauzy, "New algorithms for fault trees analysis," *Reliability Engineering & System Safety*, vol. 40, no. 3, pp. 203–211, 1993.
28. 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, 2015.
29. M. Bozzano, A. Cimatti, A. Griggio, and C. Mattarei, "Efficient anytime techniques for model-based safety analysis," in *International Conference on Computer Aided Verification*, pp. 603–621, Springer, 2015.
30. 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.
31. M. Bozzano, H. Bruintjes, A. Cimatti, J.-P. Katoen, T. Noll, and S. Tonetta, "The COMPASS 3.0 toolset," in *IMBSA 2017*, 2017.
32. M. Bozzano, A. Cimatti, C. Mattarei, and S. Tonetta, "Formal safety assessment via contract-based design," in *Automated Technology for Verification and Analysis*, 2014.
33. T. Prosvirnova, *AltaRica 3.0: a Model-Based approach for Safety Analyses*. Theses, Ecole Polytechnique, Nov. 2014.
34. D. Chen, N. Mahmud, M. Walker, L. Feng, H. Lönn, and Y. Papadopoulos, "Systems Modeling with EAST-ADL for Fault Tree Analysis through HiP-HOPS*," *IFAC Proceedings Volumes*, vol. 46, no. 22, 2013.
35. M. Gudemann and F. Ortmeier, "A framework for qualitative and quantitative formal model-based safety analysis," in *HASE 2010*, 2010.