

1 Theory

1.1 Definitions

Given a constraint system C where C is an ordered set of abstract constraints over some set of variables, $\{C_1, \dots, C_n\}$. The satisfiability problem is in conjunctive normal form (CNF): $C = \bigwedge_{i=1, \dots, n} C_i$; and each C_i is a disjunction of literals $C_i = l_{i1} \vee \dots \vee l_{ik_i}$ where each literal l_{ij} is either a Boolean variable x or its negation $\neg x$.

Satisfiability (SAT) : A CNF is satisfiable iff there exists an assignment of truth values to its variables such that the formula evaluates to true. If not, it is unsatisfiable (UNSAT).

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

Let F be the set of all faults defined in the model and G be the set of all component contracts (guarantees). $C = \{C_1, C_2, \dots, C_n\}$ where for $i \in \{1, \dots, n\}$, C_i has the following constraints for any $f_j \in F$ and $g_k \in G$ with regard to the top level property P :

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

Given a state space S , a transition system (I, T) consists of the initial state predicate $I : S \rightarrow \{0, 1\}$ and a transition step predicate $T : S \times S \rightarrow \{0, 1\}$. Reachability for (I, T) is defined as the smallest predicate $R : S \rightarrow \{0, 1\}$ which satisfies the following formulas:

$$\begin{aligned} \forall s. I(s) &\Rightarrow R(s) \\ \forall s, s'. R \wedge T(s, s') &\Rightarrow R(s') \end{aligned}$$

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

Given a transition system which satisfies a safety property P , it is possible to find which parts of the system are necessary for satisfying the safety property through the use of Ghassabani's *All Minimal Inductive Validity Cores* algorithm [?, ?]. This algorithm makes use of the collection of all minimal unsatisfiable subsets of a given transition system in terms of the negation of the top level property, i.e. the top level event. This is a minimal subset of the activation literals such that the constraint system C with the formula $\neg P$ is unsatisfiable when these activation literals are held true. Formally:

MUS : A Minimal Unsatisfiable Subset M of a constraint system C is : $\{M \subseteq C \mid M \text{ is UNSAT and } \forall c \in M: M \setminus \{c\} \text{ is SAT}\}$. This is the minimal explanation of the constraint systems infeasability.

A closely related set is a *minimal correction set* (MCS). The MCSs describe the minimal set of model elements for which if constraints are removed, the constraint system is satisfied. For C , this corresponds to which faults are not constrained to inactive (and are hence active) and violated contracts which lead to the violation of the safety property. In other words, the minimal set of active faults and/or violated properties that lead to the top level event.

MCS : A Minimal Correction Set M of a constraint system C is : $\{M \subseteq C \mid C \setminus M \text{ is SAT and } \forall S \subset M : C \setminus S \text{ is UNSAT}\}$. A MCS can be seen to “correct” the infeasability of the constraint system.

A duality exists between MUSs of a constraint system and MCSs as established by Reiter [?]. This duality is defined in terms of *hitting sets*. A hitting set of a collection of sets A is a set H such that every set in A is “hit” by H ; H contains at least one element from every set in A [?].

Hitting Set: Given a collection of sets K , a hitting set for K is a set $H \subseteq \cup_{S \in K} S$ such that $H \cap S \neq \emptyset$ for each $S \in K$. A hitting set for K is minimal if and only if no proper subset of it is a hitting set for K .

Utilizing this approach, we can easily collect the MCSs from the MUSs provided through the all MIVC algorithm and a hitting set algorithm by Murakami et. al. [?, ?].

Cut sets and minimal cut sets provide important information about the vulnerabilities of a system. A *Minimal Cut Set* (MinCutSet) is a minimal collection of faults that lead to the violation of the safety property (or in other words, lead to the top level event). We define MinCutSet in terms of the constraint system in question as follows:

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

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

1.2 Transformation of MCS

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

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

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

(i) $MCS \subseteq \text{MinCutSet}$:

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

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

(ii) $\text{MinCutSet} \subseteq MCS$:

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

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

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

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

Let $MCS = \{f_1, \dots, f_n, g_1, \dots, g_m\}$

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

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

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

1.3 Replacements for Max N Faults Analysis: Algorithm and Theory

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

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

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

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

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

Algorithm 1: Replacement

```

1  $List(MCS) = MCS$  : initialize list of MCSs that contain contracts ;
2  $Cut(g_j)$  : all minimal cut sets with cardinality less than or equal to  $N$  of the
   contract  $g_j$  ;
3  $0 < j \leq \alpha$  ;
4  $0 < i \leq \gamma$  ;
5 for all  $MCS_i \in List(MCS)$  do
6   Remove  $MCS_i$  from  $List(MCS)$ ;
7   for all  $g_j \in MCS_i$  do
8     Remove  $g_j$  from  $MCS_i$  ;
9     for all  $cut_k(g_j) \in Cut(g_j)$  do
10      Add  $cut_k(g_j)$  to  $MCS_i$  ;
11      if  $|MCS_i| \leq N$  then
12        if  $\exists g_j \in MCS_i$  then
13          Add  $MCS_i$  to  $List(MCS)$ 
14        else
15          Add  $MCS_i$  to contract look up table (done)

```

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

$$R = \prod_{j=1}^{\alpha} |Cut(g_j)|, \text{ for } g_j \in MCS_i.$$

Then the total number of replacements for all $MCSs$ is: $R * |MCS|$.

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

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

Theorem 1. *The elimination of any $|MCS_i| \leq N$ will not eliminate any $|MinCutSet| \leq N$.*

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

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

1.4 Replacements for Probabilistic Analysis: Algorithm and Theory

Replacements for Probabilistic Analysis: Algorithm and Theory

In order to complete the probabilistic analysis, we must first calculate allowable combinations of faults. This allows us the option to eliminate unnecessary combinations while performing the algorithm, thus increasing performance and diminishing the problem of combinatorial explosions in the size of minimal cut sets.

The steps are as follows:

- Verify nominal model and generate IVCs
- Compute allowed fault combinations (Algorithm 2)
- Compute minimal cut sets from IVCs and save only allowable combinations (Algorithm 3)
- Incorporate dependent faults and calculate final probabilities (Algorithm 4)
- Display results for the property

Calculating Allowable Fault Combinations In order to calculate allowable combinations, we must take into account probabilities of the combined faults and compare with the given threshold. First we can calculate the possible combinations.

Note that in the following algorithm, we are traversing over Fault objects. This means that the associated probability is a field of said object and easy to access without adding additional mapping from a fault string to its probability (as is done for Algorithm 4).

This being said, each of the allowable fault combinations has a combined probability that we must later access. Thus we store these combinations in a list called \mathcal{F} .

$$\mathcal{F} = \{F_i = MinCutSet \ni P(MinCutSet) > threshold\}$$

The allowable combinations in \mathcal{F} consist of independent faults and dependencies have not yet been incorporated. For ease of calculations and to assist in the elimination

process that takes place in Algorithm 3, we leave the inclusion of dependent faults until after all possible minimal cut sets have been generated. In this way, we eliminate the need to incorporate dependent faults twice: once for allowed combinations and once for minimal cut set generation. To rephrase for clarity, if we first incorporate dependencies into \mathcal{F} , then in order to determine if a *MinCutSet* is an allowable combination (i.e. an element of \mathcal{F}), we would have to incorporate dependent faults before searching \mathcal{F} for inclusion).

Algorithm 2: Calculate Allowable Fault Combinations

```

1 Input:  $\mathcal{Q}$  : faults,  $q_i$ , arranged with probability high to low ;
2 Output:  $\mathcal{F}$ : Allowable combinations without dependencies added ;
3  $\mathcal{F} = \text{emptyMap}$  : allowable fault combinations ;
4  $\mathcal{R} = \mathcal{Q}$ , with  $r \in \mathcal{R}$ ;
5 while  $\mathcal{Q} \neq \{\}$   $\wedge$   $\mathcal{R} \neq \{\}$  do
6    $q = \text{removePriorityElement}(\mathcal{Q})$  ;
7   for  $i = 0 : |\mathcal{R}| \wedge r_i \neq q$  do
8      $prob = p(q) \times p(r_i)$  ;
9     if  $prob < \text{threshold}$  then
10        $\text{removeTail}(\mathcal{R}, j = i : |\mathcal{R}|)$ ;
11     else
12        $\text{add}(\{q, r_i\}, \mathcal{Q})$ ;
13        $\text{add}(\{q, r_i\}, \mathcal{F})$ ;

```

Generate Minimal Cut Sets

At the leaf level, only faults are contained in IVCs (and consequently in MCSs). Thus we store these cut sets in a lookup table for quick access throughout the algorithm. For this algorithm, we assume we are in an intermediate level of the analysis. Given a hypothesis probability threshold, we find only the cut sets that contain allowable combinations of faults in terms of probability of occurrence.

Assume we have used the hitting set to generate MCS from the IVC and we have calculated allowable fault combinations. This is where the algorithm begins.

Compositional Probabilistic Minimal Cut Set Algorithm

$List(MCS)$: the set of all MCSs, ex: $\{\{f_1, f_2, g_1, g_2\}, \dots\}$

I just listed one MCS for the example, but $List(MCS)$ has all of them.

MCS_i : an element of $List(MCS)$.

For our example, $MCS_1 = \{f_1, f_2, g_1, g_2\}$.

$Cut(g_j)$: all minimal cut sets for contract g_j .

Ex: $Cut(g_1) = \{\{f_3, f_4\}\}$ and $Cut(g_2) = \{\{f_5\}, \{f_6\}\}$.

$cut_k(g_j)$: a specific cut set for contract g_j . This is an element of $Cut(g_j)$.

For the example, $cut_1(g_1) = \{f_3, f_4\}$ and $cut_2(g_2) = \{f_6\}$.

We combine probabilities and add dependent faults after this algorithm terminates.

In this algorithm, we have a few options where to eliminate sets that do not correspond with allowed combinations or whether to eliminate at all.

Option 1: Once the inlining is complete and we have a minimal cut set, if $MinCutSet \not\subseteq \mathcal{F}_i$, then we can safely eliminate $MinCutSet$. We avoid the overhead of checking subsets in \mathcal{F} , but we have the complete overhead of inlining.

Option 2: If $cut_k(g_j) \not\subseteq \mathcal{F}_i$ where \mathcal{F} is the set of all possible combinations \mathcal{F}_i , then we can safely eliminate the MCS_i in which g_j is located. This option seems good in terms of early elimination, but would have a detrimental overhead in other areas. We still have to do a search over \mathcal{F} to determine if $cut_k(g_j)$ is a subset of one of the allowed combinations AND we would still have to check final combination (or intermediate combinations) with more than just $cut_k(g_j)$.

Ex: Let $\mathcal{F} = \{\{f_1, f_2, f_5\}, \{f_6\}\}$ be the allowed fault combinations.

Let $MCS = \{f_1, g_1, g_2\}$, $Cut(g_1) = \{\{f_2\}\}$, $Cut(g_2) = \{\{f_5\}, \{f_6\}\}$.

Then for option 2, we would not eliminate anything: $cut_1(g_1) = \{f_2\} \subseteq \mathcal{F}_i$ and likewise for $cut_1(g_2)$ and $cut_2(g_2)$.

The MCSs generated through the inlining process are:

$MCS_1 = \{f_1, f_2, f_5\}$ and $MCS_2 = \{f_1, f_2, f_6\}$ which provides one combination which should be eliminated. Thus we would have to perform another check at the end of the algorithm to eliminate these.

Option 3: If $(\{f_n | f_n \in MCS_i\} \cup cut_k(g_j)) \not\subseteq \mathcal{F}_i$, then we can safely eliminate this MCS_i . This option is better in that we do not have to perform two searches in \mathcal{F} , but has the drawback of needing to find the union of all faults located in MCS_i , ignoring the g_j in MCS_i , and then performing the subset search through \mathcal{F} .

Upside: $List(MCS)$ does not grow as fast IF we can eliminate things along the way.

Downside: Worst case scenario, every $cut_k(g_j)$ has to be combined with faults in MCS_i and a complete search of all subsets of \mathcal{F} completed for every MCS_i and NOTHING is eliminated. In this scenario, it is more efficient to wait until the end and do one search through \mathcal{F} and eliminate accordingly.

Let $|MCS| = n$ ($MCS = \{MCS_1, \dots, MCS_n\}$).

In the worst case scenario, we have a contract in every MCS_i . There are α_i contracts in MCS_i .

Every contract has m_{g_α} minimal cut sets. Then every cut set is combined with faults remaining in MCS_i and searched for in \mathcal{F}_i .

Which option is better? For now in the algorithm, I will just write Option 1, Option 2, Option 3 in their respective locations. This assumes that if option 1 is used, options 2 and 3 are removed from the algorithm. Obviously. The choice of option will also change a lot depending on our data structures for these lists and what algorithms we use to check for subsets within lists. It is also highly dependent on the model that is used in this analysis. Depending on the model itself, different options would be more efficient.

Outline of algorithm

Algorithm 3: Generate Minimal Cut Sets

```

1 Input 1:  $List(MCS)$  : MCSs that contain contracts ;
2 Input 2:  $Cut(g_j)$  : all minimal cut sets of the contract  $g_j$  ;
3 Output: List of allowed minimal cut sets ;
4  $P_{List} = \{\}$  List that will hold the probabilities associated with each allowed
    $MinCutSet$  generated ;
5 for all  $MCS_i \in List(MCS)$  do
6   Remove  $MCS_i$  from  $List(MCS)$ ;
7   for all  $g_j \in MCS_i$  do
8     Remove  $g_j$  from  $MCS_i$  ;
9     for all  $cut_k(g_j) \in Cut(g_j)$  do
10      Option 2 : if  $cut_k(g_j) \subseteq \mathcal{F}_i$  for some  $\mathcal{F}_i \in \mathcal{F}$ , then add  $cut_k(g_j)$  to
         $MCS_i$  else break ;
11      Option 3 : if union of faults in  $MCS_i$  with  $cut_k(g_j)$  is subset of  $\mathcal{F}_i$ 
        for some  $\mathcal{F}_i \in \mathcal{F}$ , then add  $cut_k(g_j)$  to  $MCS_i$  else break ;
12      if  $\exists g \in MCS_i$  then
13        | Add  $MCS_i$  to  $List(MCS)$  ;
14      else
15        | Option 1 and 2 : if  $MCS_i \in \mathcal{F}$ , add dependencies, calculate
          probability, and append  $p(MCS_i)$  to  $P_{List}$  , else break ;
16        | Option 3 : add dependencies, calculate probability, and append
           $p(MCS_i)$  to  $P_{List}$  ;
17 Overall probability =  $sum\{p_i \in P_{List}\}$  ;
```

Incorporate Dependencies and Calculate Probability

Assuming that dependent faults have been collected and mapped appropriately, they are in the form: $\{\{f_1 \rightarrow \{f_3, f_7\}, f_3 \rightarrow \{f_2\}\}$ for example.

We make the assumption that there are no nested dependencies. To clarify this, we cannot have something of the form:

$$f_1 \rightarrow \{f_3, f_5\}$$

$$f_3 \rightarrow \{f_4\}$$

If this is the case, the user must define the dependency as follows:
 $f_1 \rightarrow \{f_3, f_4, f_5\}$. We will probably have to make this assumption clear to users.

Algorithm 4: Incorporate Dependencies and Calculate Probability

```

1 Input:  $F$ : map between allowable combination  $F_i$  and associated probability
  (initially zero) ;
2 Output:  $F$ : map between allowable combinations with dependencies and
  associated probability (nonzero) ;
3  $newMCS$  = empty list ;
4  $p = 1$  ;
5 for all allowable fault combinations  $F_i \in F$  do
6   Remove  $F_i$  from  $F$  ;
7   for all  $f_i \in MCS$  do
8     if  $f$  is key in dependency map then
9        $p = p * prob(f)$  ;
10      append  $f$  to  $newMCS$  ;
11      append dependent faults triggered by  $f$  to  $newMCS$  ;
12      for all depFaults triggered by  $f$  activation do
13        if  $depFault \in MCS$  then
14          remove  $depFault$  from  $MCS$  ;
15   Append  $F_i \rightarrow p$  to  $F$  ;
16 return probability  $p$  ;
17 return  $newMCS$  as the completed MCS ;

```
