1 Theory

1.1 Definitions

Given a constraint system C where C is an ordered set of abstract constraints over some set of variables, $\{C_1, ..., C_n\}$. The satisfiability problem is in conjuntive normal form (CNF): $C = \wedge_{i=1,...,n} C_i$; and each C_i is a disjunction of literals $C_i = l_{i1} \vee ... \vee l_{ik_i}$ 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). $C = \{C_1, C_2, ..., C_n\}$ where for $i \in \{1, ..., n\}$, C_i has the following constraints for any $f_j \in F$ and $g_k \in G$ with regard to the top level property P:

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

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

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

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

Given a transition system which satisfies a safety property P, it is possible to find which parts of the system are necessary for satisfying the safety property through the use of Ghassabani's *All Minimal Inductive Validity Cores* algorithm [2, 3]. 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 : $\{M \subseteq C | M$ is UNSAT and $\forall c \in M$: $M \setminus \{c\}$ is SAT$\}$. This is the minimal explaination of the constraint systems infeasability.

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

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

A duality exists between MUSs of a constraint system and MCSs as established by Reiter [6]. This duality is defined in terms of *hitting sets*. A hitting set of a collection of sets A is a set H such that every set in A is "hit" be H; H contains at least one element from every set in A [4].

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. [1, 5].

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 : $\{cut \subseteq F | C \setminus cut$ is SAT and $\forall c \subset cut : C \setminus c$ is UNSAT $\}$.

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

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 $\subseteq$ MinCutSet:

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

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

(ii) $MinCutSet \subseteq$ MCS:

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

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

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

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

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

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

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

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

Algorithm 1: Replacement

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

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

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

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

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

Theorem 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

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 occurrance.

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

Rough draft for review

- We cannot assume independence of all faults since we have the possibility of dependent fault definitions (HW faults). This is addressed in the algorithm in a way that eliminates the need for complete independence assumption.
- Need a way to associate faults in MCS with their associated probability. For now, I am just going to assume they are readily available in some data type like an ordered pair. Since the MCS faults are just strings, we will have to implement something that maps the strings to the fault and accesses the probability from there. I am not sure what the best way of doing this is yet. So assume they come together at the start of this algorithm description.

$List(MCS)$: the set of all MCSs, ex: $\{\{(f_1, p_1), (f_2, p_2), g_1, g_2\}, ...\}$
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, it would be $\{(f_1, p_1), (f_2, p_2), g_1, g_2\}$.

$Cut(g_j)$: all minimal cut sets for contract g_j.
Ex: $Cut(g_1) = \{\{(f_3, p_3), (f_4, p_4)\}\}$ and $Cut(g_2) = \{\{(f_5, p_5)\}, \{(f_6, p_6)\}\}$.

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

For the example, $cut_1(g_1) = \{(f_3, p_3), (f_4, p_4)\}$ and $cut_2(g_2) = \{(f_6, p_6)\}$.

- We also need a way to associate overall probabilities over sets with a given set. For example, all sets in $List(MCS)$ has an associated probability that will need to be calculated over the whole $List(MCS)$ set at the end of the algorithm. It is the same for MCS_i and $Cut(g_j)$. I will assume that the data types supporting this algorithm have an easy mapping.

Each individual fault is associated with a probability. These stay associated (similar to ordered pair). Thus the associated probabilities of the faults in $cut_k(g_j)$ are stored with their faults.

We combine probabilities when a complete MinCutSet is generated (no longer an MCS). Thus, all replacements are complete in MCS and only faults remain. At that point, we check for dependent faults and adjust accordingly. Then a particular MCS_i element of $List(MCS)$ has gone through replacement and has an associated probability. We save this in a list of probabilities for all $MCSs$ and that list is called P_{List}. At the end of the algorithm, since these elements are ORed together (sets of MinCutSets), the probabilities are added together (dropping the error term).

So, p_{MCS_i} corresponds to the probability of the MinCutSet generated from MCS_i and is stored in a list of probabilities called P_{List}.

Outline of algorithm

Algorithm 2: Probabilistic

```
1  List(MCS) = MCS : initialize list of MCSs that contain contracts ;
2  Cut(g_j) : all minimal cut sets of the contract g_j ;
3  for all MCS_i ∈ List(MCS) do
4      Remove MCS_i from List(MCS);
5      for all g_j ∈ MCS_i do
6          Remove g_j from MCS_i ;
7          for all cut_k(g_j) ∈ Cut(g_j) do
8              Add cut_k(g_j) to MCS_i (all individual faults are associated with
                their probabilities already);
9              if ∃g ∈ MCS_i then
10                 Add MCS_i to List(MCS) ;
11             else
12                 (p_MCS_i, newMCS_i) = calculateProbability(MCS_i) ;
13                 if p_MCS_i ≥ threshold then
14                     Append p_MCS_i to P_List ;
15                     Save newMCS_i for display/FT ;
16 Overall probability = sum{p_i ∈ P_List} ;
```

Assuming that the MCS replacements have been completely performed, the MCS is in the following form:

$MCS = \{(f_1, p_1), ..., (f_n, p_n)\}$, where f_i is the fault with associated probability p_i.

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 3: calculateProbability

```
1  newMCS = empty list ;
2  p = 1 ;
3  for all f_i ∈ MCS do
4      if f is key in dependency map then
5          p = p * prob(f) ;
6          append f to newMCS ;
7          append dependent faults triggered by f to newMCS ;
8          for all depFaults triggered by f activation do
9              if depFault ∈ MCS then
10                 remove depFault from MCS ;
11 return probability p ;
12 return newMCS as the completed MCS ;
```

References

1. A. Gainer-Dewar and P. Vera-Licona. The minimal hitting set generation problem: algorithms and computation. *SIAM Journal on Discrete Mathematics*, 31(1):63–100, 2017.
2. E. Ghassabani, A. Gacek, and M. W. Whalen. Efficient generation of inductive validity cores for safety properties. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 314–325. ACM, 2016.
3. E. Ghassabani, M. Whalen, and A. Gacek. Efficient generation of all minimal inductive validity cores. In *Proceedings of the 17th Conference on Formal Methods in Computer-Aided Design*, pages 31–38. FMCAD Inc, 2017.

4. M. H. Liffiton, A. Previti, A. Malik, and J. Marques-Silva. Fast, flexible mus enumeration. *Constraints*, 21(2):223–250, 2016.
5. 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)*, pages 1–13. SIAM, 2013.
6. R. Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987.