Efficient Anytime Techniques for Model-Based Safety Analysis

Marco Bozzano, Alessandro Cimatti, Alberto Griggio, and Cristian Mattarei

Fondazione Bruno Kessler, Trento, Italy

Abstract. Safety analysis investigates system behavior under faulty conditions. It is a fundamental step in the design of complex systems, that is often mandated by certification procedures. Safety analysis includes two key steps: the construction of all minimal cut sets (MCSs) for a given property (i.e. the sets of basic faults that may cause a failure), and the computation of the corresponding probability (given probabilities for the basic faults).
Model-based Safety Analysis relies on formal verification to carry out these tasks. However, the available techniques suffer from scalability problems, and are unable to provide useful results if the computation does not complete.
In this paper, we investigate and evaluate a family of IC3-based algorithms for MCSs computation. We work under the monotonicity assumption of safety analysis (i.e. an additional fault can not prevent the violation of the property). We specialize IC3-based routines for parameter synthesis by optimizing the counterexample generalization, by ordering the exploration of MCSs based on increasing cardinality, and by exploiting the inductive invariants built by IC3 to accelerate convergence.
Other enhancements yield an "anytime" algorithm, able to produce an increasingly precise probability estimate as the discovery of MCSs proceeds, even when the computation does not terminate.
A thorough experimental evaluation clearly demonstrates the substantial advances resulting from the proposed methods.

Keywords: Formal methods, Safety Analysis, Fault Tree, IC3, Parameter Synthesis

1 Introduction

Safety analysis [1,2,3] is an essential step for the design of critical systems. Safety analysis activities aim at demonstrating that a given system meets the conditions that are required for its deployment and use in the presence of faults. In many application domains, such activities are mandatory to obtain system certification. Safety analysis includes two key steps: (i) the construction of all *minimal cut sets* (MCSs), i.e. (minimal) sets of faults that lead to a *top level event* (TLE), such as the loss of a desirable functionality; and (ii) the computation of the corresponding *fault probability* (i.e. the probability of reaching the TLE), given probabilities for the basic faults.

In recent years, there has been a growing interest in model-based safety analysis (MBSA) [4,5,6,7,8,3,9]. Its purpose is to automate the most tedious and error-prone

activities that today are carried out manually. This is done by analyzing models where selected variables represent the occurrence of faults. Cut sets are assignments to such variables that lead to the violation of the property. Formal verification tools, notably those based on model checking [8,10] have been extended to automate traditional safety analysis activities, such as the generation of minimal cut sets, and to perform probabilistic evaluation.

The practical application of MBSA in an industrial setting poses two key problems. The first one is scalability. In addition to the sheer size of the models, a specific factor is the possibly huge number of relevant MCSs, corresponding to different fault combinations. The second problem is to support the state of the practice. In manual safety analysis, the exploration often proceeds according to the importance and likelihood of fault configurations: MCSs of lower cardinality, that are typically associated with higher probability, are explored before the ones with higher cardinality. When the analysis is considered to be sufficiently thorough, over-approximation techniques are used to assess the weight of the unexplored MCSs.

In this paper, we investigate and evaluate a family of efficient algorithms for safety analysis. We work under the *monotonicity assumption*, commonly adopted in safety analysis, that an additional fault can not prevent the violation of the property. We specialize IC3-based routines for parameter synthesis by optimizing the generalization of counterexamples, and by ordering the exploration of MCSs based on increasing cardinality. We also propose a way to accelerate convergence by exploiting the inductive invariants built by IC3.

The practical applicability of our approach is enhanced by proposing a method to precisely compute the under- and over-approximated probability of failure. This technique produces an increasingly precise estimation as the discovery of MCSs proceeds, with the advantage of providing an "anytime" algorithm.

The described approach was implemented within the XSAP platform for safety analysis [11,12], extending and integrating the model checking routines of the underlying NUXMV model checker [13]. We carried out a thorough experimental evaluation on a number of benchmarks from various sources. The results clearly demonstrate the substantial advances resulting from the proposed methods. First, we can complete the computation of all MCSs more efficiently, and for much larger problems than previously possible. Second, even when the computation fails to terminate due to the number of MCSs, the algorithms produce intermediate approximations of increasing precision at the growth of the available computation resources. Furthermore, although here we concentrate on invariant properties of finite-state systems, our techniques can be easily extended to consider also arbitrary LTL properties and infinite-state models (where faults are still expressed with propositional variables).

Related Work. The field of MBSA is receiving increasing attention [14]. Many works cover aspects of modeling (see for example [15,16,10,11]), and propose dedicated mechanisms for the description of faults, also in probabilistic settings. Here, we work within assumptions derived from practical industrial experience. In particular, we assume that the faults are specified as discrete variables in a qualitative transition system, and that probabilities are attached to the basic faults after MCSs have been computed.

The ESACS project [16] pioneered the idea of model-based safety assessment by means of model checking techniques. The work in [17] proposes an algorithmic approach to the automatic construction of fault trees. The approach relies on the structure of the system, and does not apply model checking techniques.

In this paper, we focus on the fully automated construction of MCSs for a given transition system. There are relatively few works addressing the problem [18,8,19]. They share two key differences with respect to the work presented here. First, they do not rely on recent IC3 [20] techniques; second, none of them tackles the problem of anytime techniques. Specifically, the approach in [18] proposes the idea of layering of the exploration in terms of cardinality of MCSs. The approach is SAT-based, using bounded model checking; it does not directly discuss the problem of reaching convergence, likely adopting an induction-based approach. [16] investigates the generation of orders between faulty events, using a BDD-based approach. Automated fault tree analysis in probabilistic settings is covered in [21]. In [8], an approach based on BDDs and dynamic cone of influence is proposed. The approach does not scale well for models containing many variables. In [19], techniques based on SAT-based bounded model checking are combined with BDD-based techniques in order to achieve completeness. The approach is shown to substantially outperform the engines used in a proprietary industrial tool.

The work on IC3-based parameter synthesis [22] can in principle address the problem tackled in this paper. Here we propose several enhancements based on the specific features of the problem, with dramatic improvements in terms of scalability.

Structure of the paper. The rest of this paper is structured as follows. In Section 2 we overview SA, and in Section 3 we formally characterize the problem of MBSA. In Section 4 we discuss the available baseline, and in Section 5 we present our new algorithms for MCS computation. In Section 6 we discuss the anytime approximation. In Section 7 we experimentally evaluate the approach, and in Section 8 we draw some conclusions and present directions for future work.

2 Safety Analysis

Traditional techniques for safety analysis include Fault Tree Analysis (FTA) and Failure Mode and Effects Analysis (FMEA) [23,24]. FTA is a deductive technique, whereby an undesired state (the so called *top level event* – TLE) is specified, and the system is analyzed for the possible fault configurations (sets of faults, a.k.a. basic events) that may cause the top event to occur. Fault configurations are arranged in a tree, which makes use of logical gates to depict the logical interrelationships linking such events with the TLE, and which can be evaluated quantitatively, to determine the probability of the TLE. Of particular importance in safety analysis is the list of *minimal* fault configurations, i.e. the *Minimal Cut Sets (MCSs)*.

FMEA works in a bottom-up fashion, and aims at producing a tabular representation (called *FMEA table*) that represents the causality relationships between (sets of) faults and a list of properties (representing undesired states, as in the case of FTs). Although FMEA is different in spirit from FTA, generation of MCSs can also be used as a building

block for computing FMEA tables, in particular under the assumption of monotonicity (i.e., any super-set of a MCS will still cause the TLE) [3,25].

More specifically, a cut set is a set of faults that represents a necessary, but not sufficient, condition that may cause a system to reach an unwanted state/behaviour. For instance, the cut set {battery1_failure, battery2_failure} may cause the safety hazard "fuel pump malfunctioning" in a 2-redundant electrical system. Moreover, minimality implies that every proper super-set of it cannot prevent the possibility to have the malfunction. When the safety hazard is reachable without triggering of any fault, the FT collapses to true, representing the empty cut set (which is evidently minimal).

An important aspect of safety assessment is the quantitative evaluation of FTs, i.e. the association of FT nodes with probabilities. In particular, the determination of the probability of the TLE is used to estimate the likelihood of the safety hazard it represents. Such computation can be carried out by evaluating the probability of the logical formula given by the disjunction of the MCSs (each MCS, in turn, being the conjunction of its constituent faults). It is standard practice, in particular for complex systems, to consider only cut sets up to a maximum cardinality – in order to simplify the computation. This approach is justified by the fact that, in practical cases, cut sets with high cardinality have low probabilities, and may be "safely" ignored. However, it is essential to have criteria to estimate the error which is inherent in such approximation, since under-approximating the probability of a hazard would not be acceptable.

3 Model-Based Safety Analysis

3.1 Minimal Cut Set Computation

We represent a plant using a transition system, as follows. A transition system is a tuple $S = \langle V, F, I, T\rangle$, where V is the set of state variables, $F \subseteq V$ is a set of parameters, the *failure mode variables*; I is the initial formula over V; T is the transition formula over V and V' (V' being the next version of the state variables). A state s (resp. s') is an assignment to the state variables V (V'). A trace of S is a sequence $\pi = s_0, s_1, \ldots, s_n$ of states such that s_0 satisfies I and for each k, $1 \leq k \leq n$, $\langle s_{k-1}, s_k\rangle$ satisfies T.

A cut set is formally defined as follows [8].

Definition 1 (Cut set). *Let $S = \langle V, F, I, T\rangle$ be a transition system, $FC \subseteq F$ a fault configuration, and TLE a formula over V (the top level event). We say that FC is a cut set of TLE, written $cs(FC, TLE)$ if there exists a trace $s_0, s_1, \ldots, s_k$ for S such that: $i)$ $s_k \models TLE$; $ii)$ $\forall f \in F \;\; f \in FC \iff \exists i \in \{0, \ldots, k\}\; (s_i \models f)$.*

Intuitively, a cut set is a fault configuration whose faults are active at some point along a trace witnessing the occurrence of the top level event. In safety analysis, it is important to identify the fault configurations that are minimal in terms of failure mode variables – as they represent simpler explanations for the top level event, and they have higher probability, under the assumption of independent faults. Minimal configurations, called *minimal cut sets*, are defined as follows.

Definition 2 (Minimal Cut Sets). *Let $S = \langle V, F, I, T\rangle$ be a transition system and $FConf = 2^F$ be the set of all fault configurations, and TLE a top level event. We*

define the set of cut sets and minimal cut sets of TLE as follows:

$$\begin{aligned} \mathit{CS}(\mathit{TLE}) &= \{\mathit{FC} \in \mathit{FConf} \mid cs(\mathit{FC}, \mathit{TLE})\} \\ \mathit{MCS}(\mathit{TLE}) &= \{cs \in \mathit{CS}(\mathit{TLE}) \mid \forall cs' \in \mathit{CS}(\mathit{TLE})\ (cs' \subseteq cs \Rightarrow cs' = cs)\} \end{aligned}$$

The previous definition of MCS is based on the assumption that fault configurations are *monotonic*, i.e. activating additional faults cannot prevent triggering the top level event. This is an assumption that is commonly applied in practice, considering that it leads to a conservative over-approximation of the unreliability (probability of TLE). In cases where this is not desirable, the notion of MCS can be generalized to the more general one of *prime implicant* [26] i.e., with no monotonicity assumption. However, this is not considered here.

3.2 Computing faults probability

Algorithm 1: Probability computation.

Input: BDD (n), Probability map ($\mathcal{P}$), Hashtable ($cache$)
Result: Probability

```
1  if n in cache then
2      return cache[n];
3  if n = ⊤ then
4      return 1.0;
5  if n = ⊥ then
6      return 0.0;
7  pthen = Probability_computation(get_then_node(n), P, cache);
8  pelse = Probability_computation(get_else_node(n), P, cache);
9  pcur = P(get_var(n));
10 cache[n] = pcur · pthen + (1.0 − pcur) · pelse;
11 return cache[n];
```

Given a set of MCSs and a mapping $\mathcal{P}$ giving the probability for the basic faults, it is possible to compute the probability of the occurrence of the top-level event. Under the assumption that basic faults are independent[1], the probability of a single MCS σ is given by the product of the probabilities of its basic faults:

$$\mathcal{P}(\sigma) = \prod_{f_i \in \sigma} \mathcal{P}(f_i).$$

For a set of MCSs S, the probability can be computed using the above and the following recursive formula:

$$\mathcal{P}(S_1 \cup S_2) = \mathcal{P}(S_1) + \mathcal{P}(S_2) - \mathcal{P}(S_1 \cap S_2).$$

Interpreting the set of MCSs as a disjuction of partial assignments to the fault variables, then it is possible to represented such formula using a Binary Decision Diagram, a simple and efficient way of computing its probability is shown in Algorithm 1. The algorithm exploits the following facts:

[1] Specific techniques for the case of common cause analysis are out of the scope of this paper.

(i) the probability of two disjoint sets is simply the sum of the two probabilities; and
(ii) the two children t and e of a BDD node with variable v correspond to the two disjoint sets of assignments for the formulae $v \wedge t$ and $\neg v \wedge e$ respectively;
(iii) if the variable v does not occur in the formula f, then f is independent from v, and so $\mathcal{P}(v \wedge f) = \mathcal{P}(v) \cdot \mathcal{P}(f)$;
(iv) $\mathcal{P}(\neg v) = 1 - \mathcal{P}(v)$ by definition.

4 Basic algorithms for MCS computation

BDD-based algorithms. The work in [8] presents a series of symbolic algorithms for the computation of MCSs using BDDs. The algorithms are based on a reachability analysis on the symbolic transition system extended with history variables for fault events. Intuitively, each state is decorated with the faults that have occurred in its history; at the end of the reachability, each state is thus associated with the set of cut sets that are required to reach it. MCSs are extracted by projecting the reachable states over the history variables and then minimizing the result, using standard routines provided by BDD packages.

Exploiting BMC. An improved version of the BDD-based routines is presented in [19], by exploiting Bounded Model Checking (BMC) as a preprocessing step. Essentially, the idea is to run BMC up to a maximum (user-defined) depth k to check the invariant property stating that the top level event can never be reached. Whenever a counterexample trace is found, a cut set cs (not necessarily minimal) is extracted from it, and the model is strengthened with constraints excluding all the supersets of cs. When no more counterexamples of length at most k are found, a BDD-based algorithm is invoked on the strengthened model, in order to discover the remaining cut sets not yet covered.

The approach can be generalized to completely avoid the use of BDDs. The idea is to use the BMC engine incrementally to enumerate cut sets, and combine it with a generic "black box" procedure for checking invariant properties, invoked periodically (e.g. before increasing the BMC bound k) to check whether all the MCSs have been enumerated.

MCS via parameter synthesis. The work in [22] presents an efficient extension of the IC3 algorithm (called ParamIC3) that allows to compute, given a model M depending on some parameters P, the set of all values of P such that the model satisfies a given invariant property. The algorithm works by complement, constructing the set of "good" parameters by incrementally blocking "bad" assignments extracted from counterexample traces generated by IC3.

The technique can be immediately exploited also for MCS computation as follows. First, the model is extended with history variables for fault events, as in [8]. The parameter synthesis algorithm is then invoked on the extended model, considering the history variables as parameters, and checking the property that the top level event is never triggered. Each "bad" assignment blocked by ParamIC3 (see [22]) corresponds to a fault configuration reaching the top level event. When the algorithm terminates, the MCS set can be extracted by simply dropping the subsumed bad assignments.

5 Efficient algorithms for MCS computation

In practice, the BDD-based routines of [8] show rather poor scalability, and are typically not applicable to problems of realistic size. Using BMC as a preprocessing step helps significantly [19], but ultimately also this technique is limited by the scalability problems of BDD-based approaches. The technique of [22], being based on the very-efficient IC3 algorithm, is much more promising. However, in the basic formulation given in the previous section, its performance is extremely poor when the number of possible fault configurations leading to the top level event is large. In this Section, we show how the situation can be dramatically improved by exploiting the monotonicity assumption on faults under which we are operating.

5.1 Monotonic parameter synthesis

The first (trivial) improvement exploits the definition of monotonicity to generalize the set of "bad" parameters to be blocked whenever IC3 generates a counterexample trace. This idea is similar to the *dynamic pruning* optimization of [8] for BDD-based computation. The monotonicity assumption ensures that if a set of faults F is sufficient to generate the top level event, so does any set $S \supseteq F$. Therefore, any assignment to the (parameters corresponding to the) fault variables $\gamma = \{f_j, \ldots, f_k\} \cup \{\neg f_i, \ldots, \neg f_h\}$ extracted from an IC3 counterexample trace can be immediately generalized to $\gamma' = \{f_j, \ldots, f_k\}$, by dropping all the variables assigned to false.

The above optimization prevents the algorithm from explicitly considering all cut sets that are subsumed by the one just found, i.e. $F = \{f_j, \ldots, f_k\}$. However, F itself might not be minimal. In this case, IC3 would later have to find another configuration $G \subset F$, and the effort spent in blocking F would have been wasted.

We address this by modifying the branching heuristic of the SAT solver used by IC3. In the modified heuristic, (SAT variables corresponding to) faults are initially assigned to false, and they have higher priority than the other variables, so that no other variable is assigned by a SAT decision before all the fault variables are assigned. This ensures that fault variables are assigned to true only when necessary to satisfy the constraints.

The above idea is very simple to implement and integrate in the IC3-based algorithm (in total, it requires about 20 lines of code), and it provides a significant performance boost (as we will show in Section 7). However, it is still not sufficient to ensure that no redundant cut sets are generated. The reason is that, by the nature of IC3, ParamIC3 enumerates counterexample traces in an increasing order of length k, so that it only considers traces of length $k+1$ when all the traces of length $\leq k$ have already been excluded.[2] This means that, if the shortest trace that leads to the top level event from a set F of faults is k, but there exists another set of faults $S \supset F$ that leads to the top level event in $h < k$ steps, then S will necessarily be blocked by ParamIC3 before F. In some extreme cases, this might make the heuristic completely ineffective.

[2] For readers familiar with IC3, strictly speaking this is not fully accurate: if the IC3 implementation uses a priority queue for managing counterexamples to induction [20], some counterexamples of length $h > k$ may be generated before all those of length $\leq k$ are blocked. However, the argument still holds in this case, so the issue can be ignored for simplicity.

5.2 Enumerating only MCS

Algorithm 2: Basic MCS enumeration with ParamIC3

```
   Input: Model (M = ⟨I, T⟩), Top level event (TLE), Faults (F)
   Result: MCS
1  bound = 1;
2  MCS = ⊥;
3  while True do
4      c = make_atmost(F, bound);
5      region = ParamIC3(I ∧ ¬ MCS, T ∧ ¬ MCS, (¬TLE ∨ ¬c), F);
6      MCS = MCS ∨ ¬ region;
7      done = IC3(I ∧ ¬ MCS, T ∧ ¬ MCS, ¬TLE);
8      if done then
9          return MCS
10     else
11         bound = bound + 1;
```

We address the problem by incorporating in our algorithm a solution originally proposed in [18]. The idea is to force the algorithm to proceed by *layering*, by forcing the search to compute the cuts sets of increasing cardinality, instead of analyzing traces of increasing length. The pseudo-code for the basic version is shown in Algorithm 2. At each iteration of the main loop, the algorithm uses an "atmost" constraint c to limit the cardinality of the cut sets generated, by relaxing the invariant property to check from ¬TLE to $(\neg\text{TLE} \vee \neg c)$. The termination check is performed by invoking the "regular" version of IC3 on the model strengthened to exclude the already-computed cut sets, to check whether there are other fault configurations that can reach the top level event. It is easy to see that Algorithm 2 enumerates only the MCSs, and thus it avoids the exponential blow-up suffered from ParamIC3 on the model of Example 1. However, it does so at a significant price, since it needs two IC3 calls per iteration. On less pathological examples, the overhead introduced might largely outweigh the potential benefits.

Algorithm 2 can be improved by exploiting the capability of IC3 (and so also of ParamIC3) of generating a proof for verified properties in the form of an *inductive invariant* entailing the property P. In our specific case, the inductive invariant ψ produced by ParamIC3 on line 5 of Algorithm 2 would satisfy the following: (i) $I \wedge \neg\text{MCS} \wedge \text{region} \models \psi$; (ii) $\psi \wedge T \wedge \neg\text{MCS} \wedge \text{region} \models \psi'$; and (iii) $\psi \wedge \neg\text{MCS} \wedge \text{region} \models (\neg\text{TLE} \vee \neg c)$. The first improvement is based on the observation that the inductive invariant can be fed back to ParamIC3 at the next iteration of the main loop, thus avoiding the need of restarting the search from scratch. The second improvement, instead, exploits the computed invariant to check whether all the MCSs have been enumerated, thus avoiding the second invocation of IC3 of line 7. This is done by checking with a SAT solver whether the current invariant ψ is strong enough to prove that the top level event cannot be reached by any fault configuration not covered by the already-computed cut sets. Note that this does not affect completeness, since in the worst case the atmost constraints simplifies to true after $|\boldsymbol{F}|$ iterations of the loop. However, the hope is that

Algorithm 3: Enhanced MCS enumeration with ParamIC3

Input: Model ($\mathcal{M} = \langle I, T \rangle$), Top level event (TLE), Faults ($\boldsymbol{F}$)
Result: MCS

bound = 1;
MCS = $\bot$;
invar = $\top$;
while *True* **do**
    c = make_atmost($\boldsymbol{F}$, bound);
    region, invar = ParamIC3($I \wedge \neg$ MCS $\wedge$ invar, $T \wedge \neg$ MCS $\wedge$ invar, ($\neg$TLE $\vee \neg$c), $\boldsymbol{F}$);
    MCS = MCS $\vee \neg$ region;
    done = check_unsat($\neg$ MCS $\wedge$ invar $\wedge$ TLE);
    if *done* **then**
        return *MCS*
    else
        bound = bound + 1;

in practice the inductive invariant will allow to exit the loop much earlier. The enhanced algorithm is shown in Algorithm 3, where the improvements are displayed in red.

Example 1. Consider the following example, using the syntax of NUXMV [13].

```
MODULE main
IVAR
 fault_1 : boolean;
 ...
 fault_N : boolean;

DEFINE fault_count := fault_1 + ... + fault_N;

VAR counter : 1 .. N;
    status : boolean;

ASSIGN
  init(counter) := 1;
  next(counter) := counter = 10 ? 1 : counter + 1;

TRANS (fault_count = 0) | (fault_count > (N - counter));

ASSIGN
  init(status) := TRUE;
  next(status) := (fault_count = 0);
```

There are N fault variables, and suppose the top level event occurs when the `status` variable becomes false, i.e., whenever at least one fault occurs. Therefore, the MCSs for this model are the N singleton sets containing one fault variable each. However, the **TRANS** constraint forces an inverse dependency between the number of steps to reach the top level event and the cardinality of the smallest cut sets needed: for k steps, the smallest cut sets have cardinality $N - k$, and there are $\binom{N}{k}$ of them. Therefore, even with the branching heuristic described above, ParamIC3 will generate an exponential number of counterexamples (since $\sum_{k=1}^{N} \binom{N}{k} = 2^N - 1$) before finding the MCSs. $\diamond$

6 Anytime approximation

An additional benefit of Algorithm 3 compared to the other algorithms of Sections 4 and 5 is that it provides an "anytime" approximation behaviour on the set of MCSs, in the sense that at any point during its execution, the candidate solution is a subset of all the MCSs. As pointed out in Section 2, however, such underapproximation is useful only if it is possible to estimate its error in terms of failure probability. Here, we show a simple but effective procedure for estimating the approximation error on the fly, during the execution of Algorithm 3. This allows to consider a bound on the error as an alternative stopping criterion for the algorithm, which might be useful in cases when the full computation of all the MCSs would be too expensive.

The idea is to keep two running bounds for the probability x of reaching the top-level event, such that at any point in the execution of the algorithm $P_L(\text{TLE}) \leq x \leq P_U(\text{TLE})$. Initially, we set $P_L(\text{TLE}) = 0$ and $P_U(\text{TLE}) = 1$. When a minimal cut set m_1 is found, $P_L(TLE)$ is incremented by computing the probability of the fault configurations represented by m_1 that are not covered by the already-computed MCSs. This can be done by constructing the BDD for the formula $m_1 \wedge \neg\text{MCS}$, and then computing its probability with Algorithm 1.[3]

For updating the upper bound $P_U(\text{TLE})$, instead, we exploit fact that Algorithm 3 proceeds by layers of increasing cardinality. More precisely, when ParamIC3 returns at line 7, we know that all the fault configurations of cardinality smaller or equal to the current bound that are not included in MCS will definitely not cause the top-level event. The probability P_{excluded} of these configurations can be computed with Algorithm 1 by constructing the BDD for the formula $\neg\text{make\_atmost}(\boldsymbol{F}, \text{bound}) \wedge \neg\text{MCS}$. With this, the new value of $P_U(\text{TLE})$ is given by $1 - P_{\text{excluded}}$. An illustration of this idea is shown in Figure 1. The red area represents the minimal cut sets found within a specific cardinality, and the blue one shows all the supersets of those cut sets. The white area denotes the configurations that cannot cause the TLE, whereas the gray one represents the unknown part. Figure 2 shows instead an example of the evolution of the error bounds during the execution of Algorithm 3 for one instance of our benchmark set: $P_L(\text{TLE})$ becomes non-zero after the first cut set found, and then grows continuously at every cut set, whereas $P_U(\text{TLE})$ decreases in steps, whenever an individual cardinality has been fully explored.

7 Experimental Evaluation

We have implemented the algorithms described in the previous sections in the XSAP [11,12] platform for model-based safety analysis. In this Section, we experimentally evaluate their performance and effectiveness. The benchmarks and executables for reproducing the results are available at `https://es-static.fbk.eu/people/mattarei/dist/FTA2015/`.

[3] For performance reasons, it might make sense to perform this computation for clusters of cut sets rather than for individual ones, trading granularity for efficiency.

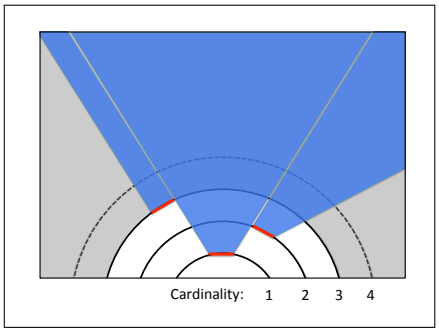

Fig. 1. Illustration of the probability error estimation in Algorithm 3.

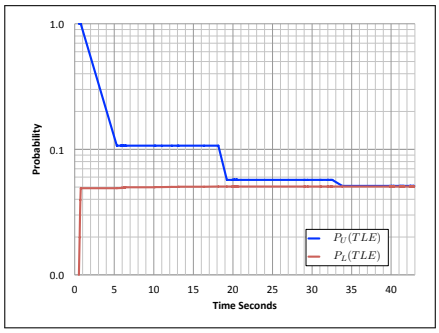

Fig. 2. Example of evolution of probability error bounds.

7.1 Benchmarks

The benchmarks used for the evaluation come from a set of real-world test cases from the avionics domain, where safety assessment and Fault Tree Analysis are parts of the formal analysis of the models.

Aircraft Electrical System. The first set of benchmarks describes the architecture of an aircraft-oriented electrical system. These problems were developed as part of the MISSA project [27], and previously analyzed using OCAS, a proprietary model-based safety assessment platform, as well as the FSAP [28] toolset. This comparison is described in [19]. This family of benchmarks is composed of four different models, where each of them is a refinement of the previous one. The properties that are taken into account describe the situation when the system that manages the alternate/continuous current is malfunctioning. Each model has two properties, for a total of 8 benchmark instances. The size of the models varies from 35 to 297 state variables and from 437 to 14030 AND gates (in an And-Inverter-Graph representation [29] of the transition relation), whereas the number of faults is between 9 and 105.

Next-gen collision avoidance. The second set of instances comes from the analysis of a novel, "next generation" air traffic control system that is being studied at NASA. Part of the activities involves the evaluation of different technological approaches in order to discover the safer and most efficient one. This process is supported by different analysis techniques, and one of those is based on formal model-based safety assessment. The formal model is composed of an on-ground Air Traffic Control System (ATC), a set of aircraft that rely on ground-based separation systems like the ATC (GSEP), and a set of aircraft that have self-separating capabilities (SSEP) as support of the standard ground-based approach.

The benchmark instances encode different architectural solutions for the Next-gen collision avoidance system. The system is composed of various numbers of GSEP and SSEP aircraft, and one ATC. The models contain 47 basic faults, and the objective is to compute the MCSs for the violation of the property "Two Aircraft shall not have a Loss

of Separation", meaning that the distance between two aircraft is below a certain safety limit. The models are scaled by varying the number of aircraft of each kind (GSEP and SSEP, from 0 to 3 each) and the number communication rounds between each aircraft and the ATC (from 1 to 10). The size of the models varies from 162 to 330 state variables and from 1700 to 5110 AND gates.

Wheel Braking System. The third family of benchmarks models an aircraft-based wheel braking system (WBS), described in the Aerospace Information Report, version 6110 [30]. The model was developed in a joint project between FBK and The Boeing Company [31], and it is representative of an industrial system of significant size. The WBS describes a redundant architecture that takes as input the pedal information (the brake signal coming from the pilot), computes the braking force that has to be applied to the 8 wheels, and drives the hydraulic system in order to physically operate the right braking force. This system is characterized by three redundant sub units:

(i) normal brake system, receiving the pedal information and driving the hydraulic system. This unit is composed of two sub components that work in parallel in order to prevent that a single failure can cause the complete malfunctioning;
(ii) alternate brake system, receiving the pedal information and the output from the normal brake system: when the latter one is not operating as expected, it operates as backup by driving the hydraulic system;
(iii) emergency brake system, behaving similarly to the alternate one: it receives pedal information and both outputs from the normal and alternate sub systems, and operates as a backup of the alternate one.

The benchmark set consists of 4 different variants of the WBS architecture, expressing various kinds of faulty behaviour. The models contain 261 fault variables and 1482 state variables, whereas the number of AND gates varies between 35182 and 35975.

7.2 Performance evaluation

In the first part of our analysis, we evaluate the performance of different techniques for the computation of the set of MCSs. We consider the following algorithms:

BDD is the procedure of [8];
BMC+BDD is the enhancement of [19] that uses BMC as a preprocessor. The BMC implementation uses the branching heuristic described in §5 for reducing the number of fault configurations to enumerate;
BMC+IC3 is the variant of the previous technique outlined in §4, using IC3 as a "black box" invariant checking procedure. (The branching heuristic of §5 for fault variables is used also in this case);
ParamIC3 is a basic version of ParamIC3, exploiting monotonicity for generalizing parameter regions to block;
ParamIC3+faultbranch is the enhanced version of ParamIC3 that uses the branching heuristic for fault variables of §5;
MCS-ParamIC3-simple is the basic MCS procedure described in of Algorithm 2. We use m-cardinality networks [32] for encoding the cardinality constraints;

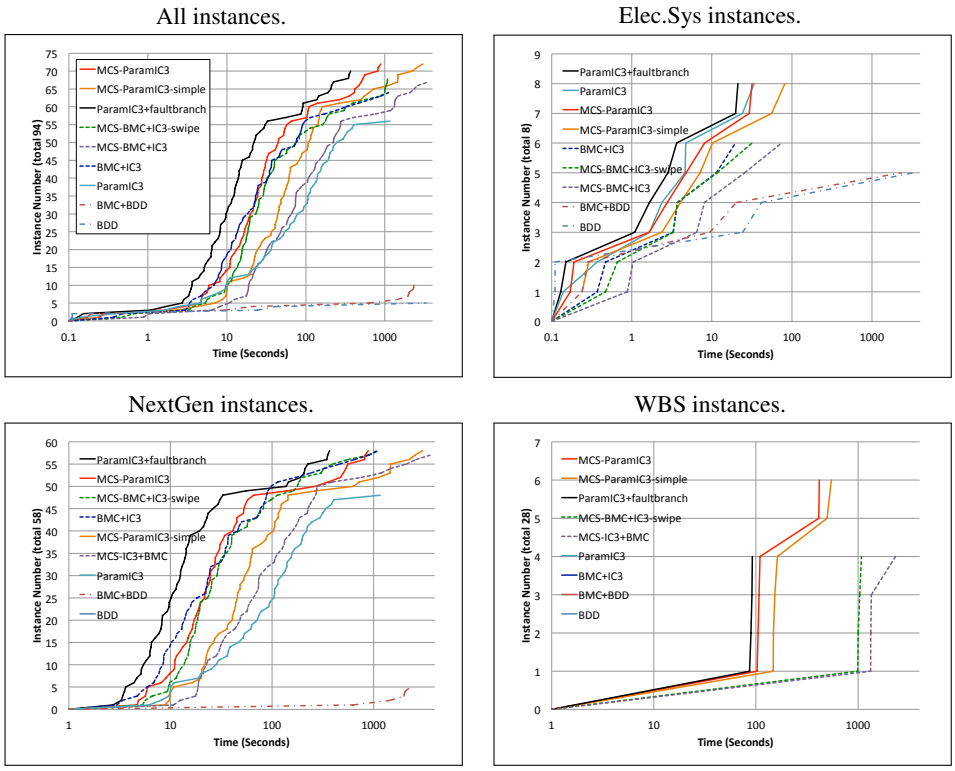

Fig. 3. Results of performance evaluation.

MCS-ParamIC3 is the enhanced MCS procedure of Algorithm 3;

MCS-BMC+IC3 is an anytime variant of BMC+IC3, in which the BMC solver is forced to enumerate only MCSs, using cardinality constraints: whenever IC3 finds that a given cardinality has been fully enumerated, the bound of the atmost constraint is increased, and BMC is restarted;

MCS-BMC+IC3-swipe is a variant of the above, in which IC3 is invoked less frequently and BMC is limited to a maximum counterexample length k, instead of fully enumerating a given cardinality. This is expected to improve performance, at the price of losing the "anytime" feature.

We have run our experiments on a cluster of Linux machines with 2.5GHz Intel Xeon E5420 CPUs, using a timeout of 1 hour and a memory limit of 4Gb. The results are shown in Figure 3. The plots show the number of solved instances (y-axis) in the given timeout (x-axis) for each of the algorithms considered. More information is provided in Table 1, where for each configuration we show the number of solved instances and the total execution time (excluding timeouts).

From the results, we can clearly see the benefits of the techniques discussed in Section 5. Using the specialized branching heuristic, ParamIC3+faultbranch performs very well in general, especially on the Elec.Sys and NextGen families. However, for

Table 1. Summary of scalability evaluation.

Algorithm	# solved				Total Time (sec)
	All	Elec.Sys	NextGen	WBS	
MCS-ParamIC3	**72**	8	58	6	7837
MCS-ParamIC3-simple	**72**	8	58	6	19326
ParamIC3+faultbranch	**70**	8	58	4	3222
MCS-BMC+IC3-swipe	**68**	6	58	4	9896
MCS-BMC+IC3	**67**	6	57	4	23210
BMC+IC3	**64**	6	58	0	5477
ParamIC3	**56**	8	48	0	6787
BMC+BDD	**10**	5	5	0	10753
BDD	**5**	5	0	0	3377

the harder WBS instances, the heuristic is not enough. On the contrary, the cardinality-based enumeration introduces an overhead for easier problems, but it pays off for harder ones, making MCS-ParamIC3 the best performing overall. Moreover, even for simpler problems the integrated approach of Algorithm 3 is not very far from the performance of ParamIC3+faultbranch. More importantly, the anytime behaviour of MCS-ParamIC3 is extremely useful in all cases in which none of the algorithms terminates, i.e. in the majority of the WBS instances. Its usefulness is evaluated in Section 7.3.

7.3 Error estimation

In order to assess the usefulness of the anytime behaviour, we evaluate the effectiveness of our technique for estimating error bounds on the probability of faults. For this, we consider the instances of the WBS benchmark set that could not be completed within the timeout, and for each of them we study the evolution of the probability bounds during the execution of MCS-ParamIC3. The results are summarized in Table 2, where we show the number of MCSs found of each cardinality, as well as the evolution of the probability bounds during the execution for a representative subset of the WBS instances (we could not include all instances for lack of space).

From the table, we can see how for most instances error bounds converge quickly towards the actual fault probability, and then continue improving very slowly, confirming the intuition of safety engineers that it is often enough to consider only MCSs of small cardinality in practice. There is only one case where the bounds are very loose, namely the M1-S18-WBS-R-323 instance. However, in this case the fault probability is several order of magnitudes smaller than for the other properties.

We remark that the probabilities for the basic faults are not artificially generated; on the contrary, they have been estimated by domain experts, and the error bounds that we have obtained matched their expectations. The table shows that, for these problems, the error estimation provided by our technique is precise enough to make our results useful in practice even when the computation of the set of MCSs does not terminate.

8 Conclusions and Future Work

In this paper we presented a family of algorithms for model-based safety analysis, based on IC3. The algorithms tightly integrate the generation and minimization of cut sets, and

Table 2. Evolution of probability error bounds on hard WBS instances.

Instance	**card**	**# MCS**	**Time**	$P_L(\mathbf{TLE})$	$P_U(\mathbf{TLE})$
M1-S18-WBS-R-0321	2	6	3.686	4.4999799997e-10	4.7856862743e-09
	3	627	27.937	4.5052040749e-10	4.5368234398e-10
	4	629	96.760	4.5052047798e-10	4.5052230781e-10
	5*	38950	3549.163	4.5052047798e-10	4.5052230781e-10
M1-S18-WBS-R-0322-left	1	2	1.809	9.9999750001e-06	1.4392898712e-05
	2	2	3.827	1.0000324995e-05	1.0004616980e-05
	3	203	23.106	1.0000325102e-05	1.0000328223e-05
	4*	46287	3271.215	1.0000325102e-05	1.0000328223e-05
M1-S18-WBS-R-0323	6	13689	480.034	1.0696143952e-28	3.5789505917e-22
	7*	52035	3596.097	1.0701599223e-28	3.5789505917e-22
M1-S18-WBS-R-0324	2	1	3.603	2.5000000000e-11	4.3619410877e-09
	4	2	9.273	2.5000000001e-11	2.5001833724e-11
	5	8729	360.012	2.5000000003e-11	2.5000000881e-11
	6*	23995	2905.057	2.5000000003e-11	2.5000000881e-11
M1-cmd_implies_braking_w1	1	13	4.508	1.1299483157e-04	1.1708790375e-04
	2	30	12.944	1.1299924596e-04	1.1300309322e-04
	3	7428	265.771	1.1299925205e-04	1.1299925473e-04
	4	3815	865.818	1.1299925205e-04	1.1299925205e-04
	5	1768	1956.225	1.1299925205e-04	1.1299925205e-04
	6	168	3465.792	1.1299925205e-04	1.1299925205e-04

*: cardinalities for which not all the MCSs could be computed within the timeout

enable the computation of the hazard probability, both numerically and symbolically. Moreover, we introduced a method to provide an estimate for the remaining computation, when the generation does not terminate, and to safely approximate the final result. This makes the approach anytime, and makes it possible to deal with cases where the number of cut sets may explode.

There are several directions of ongoing and future work. First, we are extending our implementation to handle arbitrary LTL properties and infinite-state systems. Second, concerning the routines for MCS generation, we want to investigate the role of parallelization, based on partitioning/cofactoring the space of parameters. Another line of research which is orthogonal with respect to the generation of MCSs is their presentation in a more structured way, namely as a multi-level Fault Tree (rather than as DNF). Ongoing work includes generation of hierarchical FTs using contract-based design [25].

An important open challenge we wish to explore is the relaxation of the monotonicity assumption on faults. Traditionally, in the avionics and aerospace domain (from which our benchmarks are taken) non-monotonic analysis is rarely considered, as it does not provide significant benefits – most systems are indeed monotonic and, whenever they are not, monotonic analysis already provides an accurate over-approximation. However, in other domains this is known not to be the case: for example, in circuits two subsequent inversions may prevent the occurrence of a top level event. Given the hardness of the non-monotonic analysis, it may be also worth to compute a monotonic over-approximation and find other means to tighten the measure (or to compute the tightness of the approximation). Finally, we want to study strategies to detect non-monotonicity, as in some cases it may be unclear whether it holds or not.

References

1. Leveson, N.G.: Safeware: System Safety and Computers. Addison-Wesley (1995)
2. Storey, N.: Safety Critical Computer Systems. Addison-Wesley (1996)
3. Bozzano, M., Villafiorita, A.: Design and Safety Assessment of Critical Systems. CRC Press (Taylor and Francis), an Auerbach Book (2010)
4. Bozzano, M., Villafiorita, A., et al.: ESACS: An Integrated Methodology for Design and Safety Analysis of Complex Systems. Proc. ESREL 2003 (2003) 237–245
5. Bieber, P., Bougnol, C., Castel, C., Christophe Kehren, J.P., Metge, S., Seguin, C.: Safety assessment with AltaRica. In: Building the Information Society. Volume 156 of IFIP International Federation for Information Processing. Springer (2004) 505–510
6. Bozzano, M., Cavallo, A., Cifaldi, M., Valacca, L., Villafiorita, A.: Improving Safety Assessment of Complex Systems: An Industrial Case Study. International Symposium of Formal Methods Europe (FME 2003), Pisa, Italy, LNCS **2805** (September 2003) 208–222
7. Joshi, A., Miller, S., Whalen, M., Heimdahl, M.: A Proposal for Model-Based Safety Analysis. In: Proc. DASC, IEEE Computer Society (2005)
8. Bozzano, M., Cimatti, A., Tapparo, F.: Symbolic Fault Tree Analysis for Reactive Systems. In: Proc. ATVA. Volume 4762 of LNCS., Springer (2007) 162–176
9. Bozzano, M., Cimatti, A., Katoen, J.P., Nguyen, V., Noll, T., Roveri, M.: Safety, Dependability and Performance Analysis of Extended AADL Models. Comput. Journal **54**(5) (2011) 754–775
10. Bozzano, M., Cimatti, A., Katoen, J.P., Nguyen, V.Y., Noll, T., Roveri, M.: The compass approach: Correctness, modelling and performability of aerospace systems. In Buth, B., Rabe, G., Seyfarth, T., eds.: SAFECOMP. Volume 5775 of Lecture Notes in Computer Science., Springer (2009) 173–186
11. xSAP: The xSAP safety analysis platform. `http://xsap.fbk.eu`
12. Bittner, B., Bozzano, M., Cavada, R., Cimatti, A., Gario, M., Griggio, A., Mattarei, C., Micheli, A., Zampedri, G.: The xSAP Safety Analysis Platform. (2015) Submitted to CAV'15.
13. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv Symbolic Model Checker. In: Proc. CAV. (2014) 334–342
14. Ortmeier, F., Rauzy, A., eds.: Model-Based Safety and Assessment - 4th International Symposium, IMBSA 2014, Munich, Germany, October 27-29, 2014. Proceedings. Volume 8822 of Lecture Notes in Computer Science., Springer (2014)
15. Batteux, M., Prosvirnova, T., Rauzy, A., Kloul, L.: The altarica 3.0 project for model-based safety assessment. In: 11th IEEE International Conference on Industrial Informatics, INDIN 2013, Bochum, Germany, July 29-31, 2013, IEEE (2013) 741–746
16. Bozzano, M., Villafiorita, A.: Integrating Fault Tree Analysis with Event Ordering Information. Proc. ESREL 2003 (2003) 247–254
17. Majdara, A., Wakabayashi, T.: Component-based modeling of systems for automated fault tree generation. (2009) 1076–1086
18. Abdulla, P.A., Deneux, J., Stålmarck, G., Ågren, H., Åkerlund, O.: Designing Safe, Reliable Systems Using Scade. In Margaria, T., Steffen, B., eds.: Leveraging Applications of Formal Methods, First International Symposium, ISoLA 2004, Paphos, Cyprus, October 30 - November 2, 2004, Revised Selected Papers. Volume 4313 of Lecture Notes in Computer Science., Springer (2004) 115–129
19. Bozzano, M., Cimatti, A., Lisagor, O., Mattarei, C., Mover, S., Roveri, M., Tonetta, S.: Safety Assessment of AltaRica Models via Symbolic Model Checking. Science of Computer Programming **98**(4) (2015) 464483

20. Bradley, A.R.: SAT-Based Model Checking without Unrolling. In Jhala, R., Schmidt, D.A., eds.: VMCAI. Volume 6538 of LNCS., Springer (2011) 70–87
21. Böde, E., Peikenkamp, T., Rakow, J., Wischmeyer, S.: Model based importance analysis for minimal cut sets. In Cha, S.D., Choi, J., Kim, M., Lee, I., Viswanathan, M., eds.: Automated Technology for Verification and Analysis, 6th International Symposium, ATVA 2008, Seoul, Korea, October 20-23, 2008. Proceedings. Volume 5311 of Lecture Notes in Computer Science., Springer (2008) 303–317
22. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Parameter synthesis with IC3. In: Proceedings of FMCAD, IEEE (2013) 165–168
23. SAE: ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment (December 1996)
24. Vesely, W., Stamatelatos, M., Dugan, J., Fragola, J., Minarick III, J., Railsback, J.: Fault Tree Handbook with Aerospace Applications (2002)
25. Bozzano, M., Cimatti, A., Mattarei, C., Tonetta, S.: Formal Safety Assessment via Contract-Based Design. In: Proc. ATVA. Number 8837 in LNCS. Springer (2014) 81–97
26. Coudert, O., Madre, J.: Fault Tree Analysis: 10^{20} Prime Implicants and Beyond. In: Proc. RAMS. (1993)
27. MISSA: The MISSA Project (Last retrieved on January 28, 2015) `http://www.missa-fp7.eu`.
28. Bozzano, M., Villafiorita, A.: The FSAP/NuSMV-SA Safety Analysis Platform. STTT **9**(1) (2007) 5–24
29. Biere, A., Heljanko, K., Wieringa, S.: AIGER. (2011) `http://fmv.jku.at/aiger/`.
30. SAE: AIR 6110, Contiguous Aircraft/ System Development Process Example (December 2011)
31. Bozzano, M., Cimatti, A., Pires, A.F., Jones, D., Kimberly, G., Petri, T., Robinson, R., Tonetta, S.: A formal account of the AIR6110 Wheel Brake System. (2015) Submitted to CAV'15.
32. Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: A Parametric Approach for Smaller and Better Encodings of Cardinality Constraints. In: Proceedings of CP. (2013)