Automated Analysis of Reliability Architectures

Marco Bozzano, Alessandro Cimatti and Cristian Mattarei
Fondazione Bruno Kessler
Trento, Italy
{*bozzano, cimatti, mattarei*}*@fbk.eu*

***Abstract*—The development of complex and critical systems calls for a rigorous and thorough evaluation of reliability aspects. Over the years, several methodologies have been introduced in order to aid the verification and analysis of such systems. Despite this fact, current technologies are still limited to specific architectures, without providing a generic evaluation of redundant system definitions.**

In this paper we present a novel approach able to assess the reliability of an arbitrary combinatorial redundant system. We rely on an expressive modeling language to represent a wide class of architectural solutions to be assessed. On such models, we provide a portfolio of automatic analysis techniques: we can produce a fault tree, that represents the conditions under which the system fails to produce a correct output; based on it, we can provide a function over the components reliability, which represents the failure probability of the system.

At its core, the approach relies on the logical formalism of equality and uninterpreted functions; it relies on automated reasoning techniques, in particular Satisfiability Modulo Theories decision procedures, to achieve efficiency.

We carried out an extensive experimental evaluation of the proposed approach on a wide class of multi-stage redundant systems. On the one hand, we are able to automatically obtain all the results that are manually obtained in [1]; on the other, we provide results for a much wider class of architectures, including the cases of non-uniform probabilities and of two voters per stage.

Keywords*—*safety assessment, reliability architectures, formal verification, fault tree analysis

I. Introduction

Architectures based on redundancy are used pervasively in the design of high-dependability systems. In many cases, basic patterns (such as Triple Module Redundancy) can be composed into stages. An early example of this approach is the Saturn Launch Vehicle Digital Computer described in [2]. Many configurations are possible, that may be more or less preferable, depending on the reliability of the components, and on cost factors of various nature [3].

In order to support the design, is thus important to provide techniques to measure the characteristics of a given selection, or even more importantly, the exploration of various architectural choices in the cost-reliability space.

An example of such analysis is the comprehensive quantification of the space of architectures provided in [1]. A multi-stage TMR is considered, including the ones used in [2], and the optimal solutions are identified, based on the reliability of the voting and computing modules. The results in [1], however, rely on a substantial amount of manual activity, carried out with "paper-and-pencil" techniques, and are limited by substantially simplifying hypotheses (e.g. that all the computing modules have the same failure probability).

In this paper, we propose a novel analysis flow, that allows to assess the reliability of architectures for redundancy, by means of automated techniques for model-based safety assessment (MBSA). MBSA provides for a rich modeling framework, where a comprehensive set of architectural solutions is described in an expressive formal logic of equality and uninterpreted functions (EUF). The framework is supported by automated analysis techniques, that allow for the construction of Fault Trees, and for probabilistic computation. The backend of the tool chain is based on a model checking engine, a particular technique for formal verification. Our approach has two key advantages. First, it is based on an expressive modeling language, where it is possible to describe arbitrary redundancy architectures. Second, the flow is fully automated, and allows both to produce fault trees, and to obtain a closed form representation for the reliability function.

The flow is experimentally evaluated in the same setting as in [1], demonstrating clear advantages. On the one hand, we are able to reproduce all the results in a fully automated manner. On the other hand, we are able to significantly widen the analysis: we analyze several configurations that are disregarded in [1], and we assess the cases of non-uniform probabilities, and of stages with multiple voters.

Related Work: The use of formal methods techniques to analyze redundancy architectures is rather limited. In [4], the formalism of Communicating Sequential Processes (CSP) is used to model and prove the correctness of a single TMR stage. The work is mostly manual, and does not include any quantitative analysis. In [5], a module based on redundancy is designed within the formalism of timed automata, and analyzed using the Uppaal model checker. This work focussed on the specific features of the design, and does not consider multi-staged architectures.

Structure of the paper: This paper is organized as follows. In Section II we provide some background on MBSA and on the underlying tool support. In Section III we propose a modeling framework for a class of architectures for reliability. In Section IV we describe the automated analysis provided by our approach. The experimental results are shown in Section V. In Section VI we draw some conclusions and outline directions for future work.

II. Background

The complexity of safety-critical systems is continuously increasing. Yet, the current state-of-the-practice is largely

characterized by manual approaches, which are error prone, and may ultimately increase the costs of certification. This has motivated, in recent years, a growing interest in techniques for Model-Based Safety Assessment [6]. The perspective of model-based safety assessment is to represent the system by means of a formal model and perform safety analysis (both for preliminary architecture and at system-level) using formal verification techniques. The integration of model-based techniques allows safety analysis to be more tractable in terms of time consumption and costs. Such techniques must be able to verify functional correctness and assess system behavior in presence of faults [7], [8], [9].

At the core of model based safety assessment is on the ability to exhaustively analyze the behaviours of dynamical systems. Traditionally, dynamical systems are modeled as finite state systems: their state can be represented by means of assignments to a specified set of variables [10].

In symbolic model checking, they are represented by means of Boolean logic, where Boolean variables are combined by means of Boolean connectives (e.g. conjunction, disjunction, negation). In this approach, sets of states are represented by the Boolean formula corresponding to the characteristic function of the set. The symbolic analyses of dynamic systems, most notably symbolic model checking techniques (e.g. [11], [12], [13]) rely on efficient ways to represent and manipulate Boolean formulae, in particular Binary Decision Diagrams [14], and, more recently, Boolean satisfiability (SAT) solvers [15].

Techniques for safety assessment, such as the construction Fault Trees and and Failure Mode and Effects Analysis (FMEA) tables, are automated by reduction to symbolic model checking [9], [16], [17], [18], [19], [20], [21].

Boolean logic, however, is a rather limited representation, and fails to represent many important classes of systems with infinite state, including, for example, systems with continuous evolution over time. This limitation has been lifted with the advent of Satisfiability Modulo Theory (SMT) [22], an extension of the SAT decision problem, where the formula is not pure Boolean, but it is expressed in some background theory.

The definition of an SMT problem, as in SAT, is a conjunction of clauses where each clause can be either Boolean or theory formulas. A theory that is commonly used in verification of hybrid systems is the theory of Linear Arithmetic, where logical variables with real values are used to represent. Other theories include the arithmetic over integers and arrays (used in software verification), and the theory of bit vectors (used in hardware verification). On top of SMT solver there are many different verification algorithms that can be used [23], [24], [25], [26].

In the rest of this paper we will focus primarily on the theory of Equality and Uninterpreted Functions ($\mathcal{EUF}$), where variables range over an unspecified domain, and function symbols can be declared, but have no specific property, except for the fact that they are functions, i.e. $(x = y) \rightarrow (f(x) = f(y))$.

From the application point of view, there are several toolsets that are able to carry out verification based on SMT [27], [28]. In this work we rely on *NuSMV3*, that is a complete verification and validation framework for model based analysis. *NuSMV3* is based on an open source verification engine [29], [30], that provides for BDD-based and SAT-based finite state model checking. At its core, *NuSMV3* uses the SMT solver MathSAT [31], [32], that supports several theories like linear arithmetic over reals and integers, difference logic, bit vectors and uninterpreted functions and equalities.

In addition to verification functionalities, *NuSMV3* also provides complex capabilities to perform advanced analyses. Among these, it is able to support Safety Assessment, in particular, Fault Tree Analysis [20] and reliability evaluation.

III. Modeling Architectures for Reliability

In this section we discuss the modeling of reliability architectures using formal models.

When developing safety critical systems, it is important to evaluate the components and the architectures that are required in order to guarantee the safety of the system (i.e. reach a given reliability target). System design has to take into account such analysis in order to minimize the failure probability in relation to the displacement of the components.

Over the years, safety engineering evaluated different architectural patterns; one of the most important and studied [33], [34], [1], [4], [35], [36], is the Triple Modular Redundancy (TMR). The idea of TMR consists in triplicating a module that is considered critical in order to guarantee a correct behavior of the system. As shown in Figure 4a, the input is replicated to each copy of the module M, and the output is provided to a voter V whose role is to propagate the value that is in accordance with the majority of M outputs.

The impact of a Triple Modular Redundancy approach is to increase the reliability when compared with a single (faulty) module. In other words, the main goal is to decrease as much as possible the gap with respect to a perfect (faultless) component. This concept drives the evaluation of redundant architectures. In particular, as shown in Figure 1, the evaluation of a chain of TMR (lower part) consists in comparing it with a sequence of perfect modules (upper part), both receiving the same input, and analyzing the differences on the outputs.

The structure represented in Figure 1 requires the definition of both nominal (a.k.a. perfect) and extended modules. In particular, the latter integrates both nominal and faulty behavior, using a notion of switching between the two possible behaviors, as described in more detail below.

Typically, in this phase of system analysis, the functional behavior of the components is still undefined. Despite this, we need a formalism that allows reliability analysis to be performed independently of the behavioral description. We show that the formalism based on uninterpreted functions covers this need, as it allows for an abstraction of the functional behavior of the system. In particular, it is possible to define the nominal and faulty behaviors with two different functions, and integrate both functions in the behavior of the extended components. The nominal behavior is the same for both the faultless component and extended one, in order to guarantee that they are consistent. Moreover, a faulty component can be described by a behavior that is unconstrained, that is, it

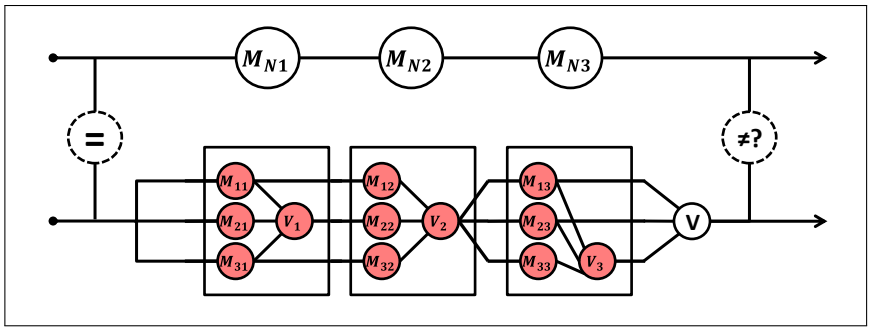


Fig. 1: Comparison TMRs and faultless modules

```
1  MODULE extended_component(nominal_function, fault, input)
2
3  VAR
4    fault_mode : boolean;
5
6  FUN
7    faulty_function : real -> real;
8
9  ASSIGN
10   init(fault_mode) := FALSE;
11   next(fault_mode) := fault;
12
13 DEFINE
14   output: =
15     case
16       (fault_mode = TRUE) : faulty_function(input);
17       TRUE : nominal_function(input);
18     esac;
```

Fig. 2: An example of extended module (SMV language)

can be modeled without putting any constraint over the faulty function.

The formal model that describes the setting shown in Figure 1 is defined using SMV language (the input language of *NuSMV3*) extended with the support for uninterpreted functions. Figure 2 presents the definition of the extended components. More in detail, the extended component receives three parameters: 1) *nominal_function*: the behavior definition in the nominal case; 2) *fault*: an environmental event that specifies whether the fault has occurred; 3) *input*: the input value (of type *real*). Within the definition of the extended component we have: the variable *fault_mode* that keeps track of the current behavior (nominal or faulty), the definition of the *faulty_function* and the multiplexer (line *13* in Figure 2) that implements switching between nominal and faulty behavior.

Figure 3 presents the definition of the extended voter. More in detail, this component receives five parameters: 1) *input_1*, *input_2* and *input_3*: the input values (of type *real*); 2) *fault*: an environmental event that specifies whether the fault has occurred; 3) *default*: the default value provided when the voter is not able to find a majority. In detail, the definition of the extended voter is composed of: the variable *fault_mode* that keeps track of the current behavior (nominal or faulty), the definition of the *voter_function* and the invariant that expresses its behavior, the definition of the *faulty_function* and the multiplexer (line *26* in Figure 3) that implements switching between nominal and faulty behavior.

The modeling capabilities enabled by this encoding into

```
1  MODULE voter_2_3(input_1, input_2, input_3, fault, default)
2
3  VAR
4    fault_mode : boolean;
5
6  FUN
7    voter_function : real * real * real -> real;
8    faulty_function : real * real * real -> real;
9
10 ASSIGN
11   init(fault_mode) := FALSE;
12   next(fault_mode) := fault;
13
14 INVAR
15   case
16     (input_1 = input_2) :
17       voter_function(input_1, input_2, input_3) = input_1);
18     (input_1 = input_3) :
19       voter_function(input_1, input_2, input_3) = input_1);
20     (input_2 = input_3) :
21       voter_function(input_1, input_2, input_3) = input_2);
22     TRUE :
23       voter_function(input_1, input_2, input_3) = default);
24   esac;
25
26 DEFINE
27   output :=
28     case
29       (fault_mode = TRUE) :
30         faulty_function(input_1, input_2, input_3);
31       TRUE : voter_function(input_1, input_2, input_3);
32     esac;
```

Fig. 3: An example of extended voter module (SMV language)

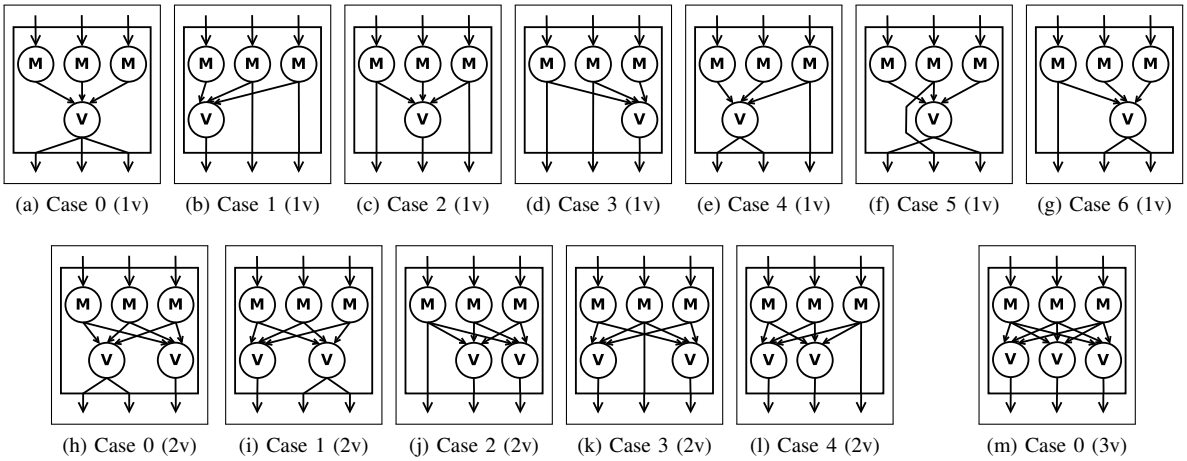
SMV language, extended with the support for $\mathcal{EUF}$ theory, are very broad and powerful, and permit modeling and analysis of different architectural patterns. In this work we concentrate on the definition of TMR structures with both one and two voters. Specifically, we exemplify the capabilities of the techniques we proposed, on the set of configurations shown in Figure 4. The notation that we use to describe a specific configuration is in the format $[t_1, t_2, ..., t_n]$, where each t_i represents a TMR t of case number i (see Figure 4). This notation is contextualized on the number of voters and defines a TMR chain of length n.

IV. Automated Analysis

In this section we analyze in detail the techniques used to carry out safety analysis. Moreover, we present some test cases, that will be used to exemplify and evaluate our approach.

Fault Tree Analysis

Fault Tree Analysis (FTA) is a technique for reliability and safety analysis based on the construction of a Fault Tree



(a) Case 0 (1v) (b) Case 1 (1v) (c) Case 2 (1v) (d) Case 3 (1v) (e) Case 4 (1v) (f) Case 5 (1v) (g) Case 6 (1v)

(h) Case 0 (2v) (i) Case 1 (2v) (j) Case 2 (2v) (k) Case 3 (2v) (l) Case 4 (2v) (m) Case 0 (3v)

Fig. 4: Triple Modular Redundancy (1, 2 and 3 voters per stage)

Diagram [37]. A Fault Tree is a representation of the possible scenarios that allow an undesirable configuration, also called Top Level Event (TLE), to be reached. A Fault Tree, as shown in Figure 5, is characterized by four kinds of nodes:

- basic faults (circles, name starts with "F"): they are the leaves of the tree and represent the faults of basic components e.g. "the generator is broken" or "the switch is stuck at open";
- intermediate events (boxes "S1 fails" and "S2 fails"): they represent an hazardous condition reached by a subsystem;
- top level event (box "TLE"): represents an undesirable configuration that is reachable by the system;
- logic gates (ANDs and ORs gates): they define the relation between the nodes of the tree. Essentially they allow the tree to be represented as a Boolean formula.

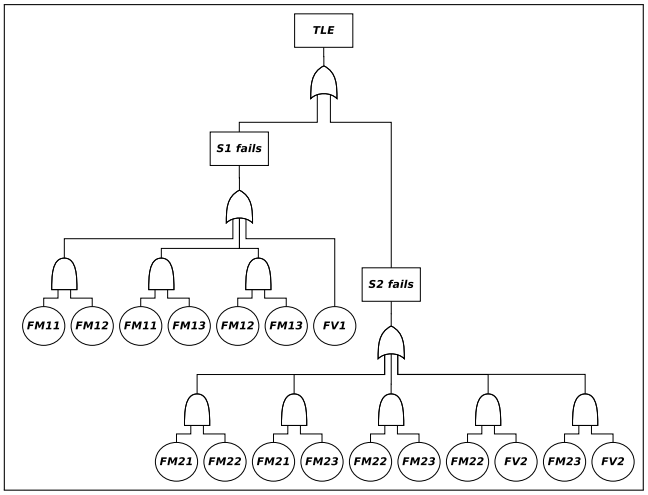


Fig. 5: Fault Tree, TMR 1 voter configuration [0,1]

Figure 5 shows the generated Fault Tree, for a chain [0,1] with 1 voter (i.e, a chain of two TMRs, the first being of type 0, Figure 4a, and the second of type 1, Figure 4b) of TMRs (compare Figures 1 and 4). In this case, the TLE represents the inequality between TMRs and perfect modules. The intermediate event "S1 fails" specifies that at least 2 outputs of stage 1 diverge from the nominal value, whereas "S2 fails" represents, respectively, the same condition for stage 2.

Numerical reliability computation

Considering the set of Minimal Cut Sets (MCSs) of a fault tree, represented in a BDD format, the computation of the reliability can be performed with a breadth first search over the BDD itself, and considering all paths that lead to the $\top$ node. A BDD structure is composed of ITE Boolean nodes, and the reliability can be computed by recursion over the nodes, as shown in 1.

$$P(n) = \begin{cases} P_f * P(n_1) + \\ (1 - P_f) * P(n_2) & n = ITE(f, n_1, n_2) \\ 1 & n = \top \\ 0 & n = \bot \end{cases} \quad (1)$$

Intuitively, the first condition in 1 expresses the ITE concept from the theory of probability, assuming that events are independent, with the positive occurrence of f represented by P_f and, respectively, its negative occurrence with $(1-P_f)$. The remaining two are the base cases. Whenever we reach the evaluation of a $\top$ node, the resulting probability is 1 (regardless of possible variables assignments). Evaluation of a $\bot$ node yields a probability of 0, as the corresponding assignment does not cause the TLE.

Symbolic reliability computation

The techniques we have described for the numerical computation of system reliability can be extended to carry out support symbolic evaluation, i.e. compute the reliability function in analytical form. In particular, each parameter of this function is a symbolic variable representing the failure probability of a single component.

As an example, equation 2 represents the reliability function computed for the configuration [0,1]. This formula has been obtained automatically by using symbolic computation techniques based on equation 1.

$$\begin{aligned} F_{sys}(F_m, F_v) = (F_v) + (2 * F_m * F_v) + (6 * F_m^2) + \\ - (16 * F_m^4 * F_v^2) - (10 * F_v * F_m^2) - (4 * F_m^6 * F_v^2) + \\ - (2 * F_m * F_v^2) + (4 * F_m^2 * F_v^2) + (4 * F_m^3 * F_v^2) + \\ + (14 * F_m^5 * F_v^2) - (4 * F_m^3) - (9 * F_m^4) + (25 * F_v * F_m^4) + \\ + (12 * F_m^5) - (26 * F_v * F_m^5) - (4 * F_m^6) + (8 * F_v * F_m^6) \end{aligned} \quad (2)$$

Computing the symbolic reliability function allows us to compare different architectural configurations independently of the specific values of failure probability. Moreover, the generation of the parametric reliability function allows us to evaluate different modules that implement the same architecture. As an example, let us consider three different modules, M_1, M_2 and M_3, that provide the same capability in terms of functional computation but using different implementations. In this scenario, symbolic computation allows us to express dependencies between failure probability of different modules. For instance, a setting where the probability of failure of M_1 (i.e. $P_f(M_1)$) is equal to F_{M1}, $P_f(M_2) = 7/8 * F_{M1}$ and $P_f(M_3) = 5/8 * F_{M1}$, can be easily expressed in order to evaluate the overall reliability. Equation 3 shows an example of the generated reliability formula, where the failure probability of M_1 is k times the failure of other modules.

$$\begin{aligned} F_{sys}(F_m, F_v, k) = (F_v) + (2 * F_m * F_v) + (2 * F_m^2) + \\ - (4 * F_v * F_m^2) - (4 * F_m^4 * k^2) - (4 * F_m^6 * k^2) + \\ - (2 * F_m * F_v^2) - (2 * F_m^4 * F_v^2) + (2 * F_m^2 * F_v^2) + \\ + (2 * F_m^3 * F_v^2) + (4 * k * F_m^2) + (8 * F_m^5 * k^2) + \\ - (16 * F_v * F_m^5 * k^2) - (10 * k * F_m^4 * F_v^2) - (6 * F_v * k * F_m^2) + \\ - (4 * F_m^4 * F_v^2 * k^2) - (4 * F_m^6 * F_v^2 * k^2) + (2 * k * F_m^2 * F_v^2) + \\ + (2 * k * F_m^3 * F_v^2) + (6 * k * F_m^5 * F_v^2) + (8 * F_v * F_m^4 * k^2) + \\ + (8 * F_v * F_m^6 * k^2) + (8 * F_m^5 * F_v^2 * k^2) - (4 * k * F_m^3) + \\ - (2 * F_v * F_m^3) + (2 * F_v * k * F_m^3) - (F_m^4) - (4 * k * F_m^4) + \\ + (3 * F_v * F_m^4) + (14 * F_v * k * F_m^4) + (4 * k * F_m^5) + \\ - (10 * F_v * k * F_m^5) \end{aligned} \quad (3)$$

Tool chain

In this work, we concentrate on generating artifacts that enable the evaluation of the reliability properties that distinguish different system configurations. Such information is intended to aid the safety engineer to evaluate the best system architecture which is compatible with the design requirements. This evaluation can be carried out by comparing the reliability functions of different architectures, possibly varying the reliability of each single component.

Our approach to generate reliability functions is supported by the tool chain shown in Figure 6. As described in previous sections, our process starts with the definition of the formal model of the system. Then, using the safety assessment capabilities of NuSMV3 we generate the Fault Tree as previously described. The reliability extractor is responsible for computing the reliability functions; it is shown as a separate entity in Figure 6 in order to have a clear view of the process, however it is integrated in the *SA-addon* of *NuSMV3*. Finally, a detailed analytical evaluation can be carried out using specific numerical computation software tools, such as *Octave* or *Matlab*. Note that the use of the reliability extractor makes it possible to construct a library of reliability functions, for specific architectures that are of interest.

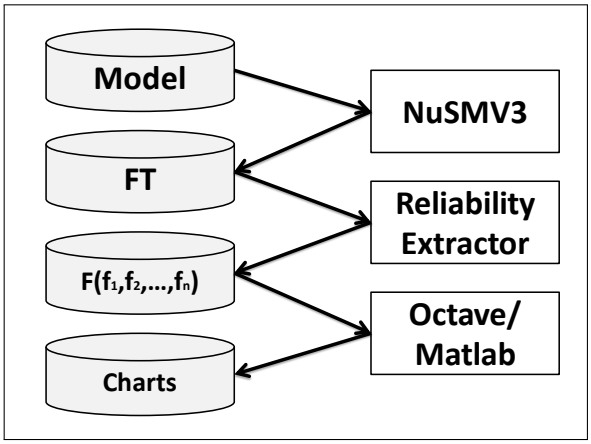


Fig. 6: Tool chain for reliability evaluation

V. Experimental evaluation

The setting for the experimental evaluation consists in generalizing the chain of sequential TMR modules with 1 and 2 voters. The idea is to arbitrarily define an array of TMR configurations that represents the patterns that have to be cyclically applied. For each of these patterns we generate the reliability function parameterized by $F_m = (1 - R_m)$ and $F_v = (1 - R_v)$, which represent the failure probability for modules and voters. Moreover, the reliability functions are generated in *Matlab* format and stored together in order to provide a reliability function library of known architectural patterns.

This setting allows us to easily compare the reliability of architectures. For instance, considering the patterns described in Table II, we can compare them together and generate the chart shown in Figure 9a. This view highlights, for each pair of values for F_m and F_v, the best configuration. Moreover, this approach allows for the generation of very informative artifacts. In particular, with our approach it is possible to provide a 3-dimensional view of the comparison between chains of TMR with 1 voter. This view is shown in Figure 7

and it allows for a clear interpretation of system reliability when varying the probability of failure of each component.

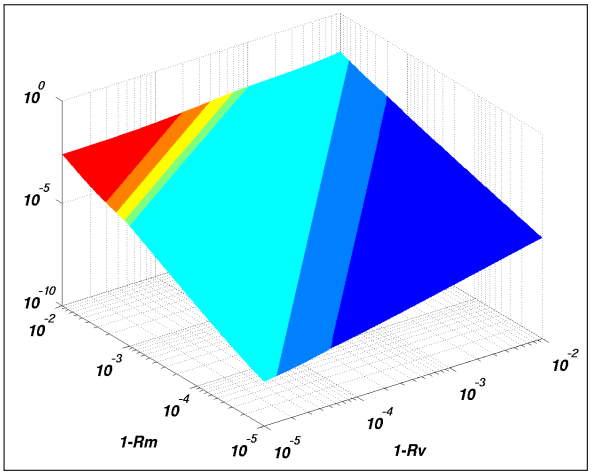


Fig. 7: 3D view for 1 voter comparison

Currently, state-of-the-art approaches for reliability evaluation do not allow for a completely automated analysis. In particular, current techniques are dedicated to specific architectural patterns. Our approach differs from previous techniques by proposing a completely automated process for the evaluation of system reliability. In view of this fact, the scalability of our techniques is not directly comparable to previous works. However, considering the time spent during the whole process, our approach is significantly more effective than standard manual analysis.

Nevertheless, the performance on the generation of the Fault Trees and the reliability functions for a chain of length 8 with 1 voter takes on average 90 seconds (with an Intel Xeon E3-1270 at 3.40GHz). Moreover, these artifacts can be stored without any loss of information, and then possibly re-used to evaluate new architectural configurations. For what concerns the evaluation and plotting of the results, using numerical tools such as *Octave* or *Matlab*, the performances are dependent on the range of values and the number of instances that have to be generated. For instance, generating Figure 9a, with range $10^{-5} \sim 10^{-2}$ for x and y axes and 400 instances, takes 115 seconds (with an Intel Core2Duo T7700 at 2.40GHz).

In the rest of this section we evaluate our approach on the following design space: one voter vs two voters per stage, and uniform vs non-uniform probability distribution. The analyses are intended to provide an overview of the approach capability both in terms of expressivity and usability.

One voter per stage, uniform distribution

The analysis of the TMR with 1 voter consists in evaluating chains of length 8 with patterns of length 4. Moreover, we explicitly added the configurations studied in [1] in order to have a direct comparison with the previous results. The outcome of this analysis is presented in Figure 9a where each (colored) area expresses that a specific configuration is better than the others in terms of system reliability. The configurations in Figure 9a, explained in Table II, confirm the results presented in previous work, and highlight the power of our approach.

By analyzing the results, we see from Figure 9a that the configurations that consider multiple outputs from the voter (e.g. the configurations in Figure 4e, 4f and 4g) are not more reliable than the others, for the considered reliability values.

One voter per stage, non-uniform distribution

As we described in Section IV, it is possible to relax the assumption that all modules have the same failure probability. In this way, it is possible to accommodate the trade-off between cost and reliability (module with higher reliability may come at the price of higher cost). In this scenario, we are able to provide the evaluation of redundant systems characterized by non-uniform failure probability for each module. Similarly to the analysis for uniform probability, in Figure 9b and Table III, we report the comparison between TMR with $(7/8) * F_m$ for M_1, where M_1 is the left-side module for each configuration in Figure 4. The results of this analysis show that, when the module 1 has higher reliability with respect to the others, the best configurations are the ones shown in Figures 4a, 4d and 4c. This result can be explained by the fact that M_2 and M_3 are less reliable than M_1, and in this case the voter is more effective on the modules that have lower reliability.

Two voters per stage, uniform distribution

Similarly to the analysis for 1 voter, we performed a comparison between configurations that consider TMR with 2 voters. The results are reported in Figure 9c, with details in Table IV. The results of the analysis is similar to the case with 1 voter. In particular, when the reliability of the voter increases the configurations switch gradually from the one in Figure 4l (moderate use of voters) to the one in Figure 4i (intensive use of voters).

Two voters per stage, non-uniform distribution

The analysis on the reliability of TMR chains with 2 voters and non-uniform probability considers the case when one voter has higher reliability with respect to the other. In detail, we analyze the case of $(1/2) * F_v$ for V_1, where V_1 is the left-side voter for each configuration in Figure 4. The higher reliability of the left-side voter imposes the use of configurations that concentrate the computation on this part (left-side) of the TMR, (in particular we are referring to the one shown in Figure 4h). When R_v decreases, the best configurations are the ones that minimize the use of voters, and in particular the ones shown in Figures 4j, 4k and 4l.

One voter vs. two voters per stage

An interesting view about the chains of TMR is the comparison between 1 and 2 voters per stage. In particular, we use the standard evaluation in the area of $10^{-5} \sim 10^{-2}$ for x and y axes, as for previous analysis. The results of the evaluation are presented in Figure 10a, and they are explained in Table VI. In this case, we highlight the difference in the order of magnitude of reliability between the two approaches. In particular, Figure 10b shows in red the area where 1 voter is better, and in blue the area where it is worse. The z axis

Identification	Description	Array of configurations
(a)	standard 1 voter	[0, 0, 0, 0, 0, 0] (1v)
(b)	standard 3 voters	[0, 0, 0, 0, 0, 0] (3v)
(c)	1 voter with 1 fanout	[1, 2, 3, 1, 2, 3] (1v)
(d)	1 voter with 1 fanout	[1, 2, 3, 3, 2, 1] (1v)
(e)	2 voters with 1 fanout	[3, 4, 2, 3, 4, 2] (2v)
(f)	no redundancy	

TABLE I: Configurations for system reliability: proportional evaluation

of this plot represents the value of the difference between two sets of configurations. Analyzing such view, we can see that the approach with 2 voters is clearly better when the reliability of the module is reasonably lower than the reliability of the voter. Differently, when the two reliabilities are comparable, the difference between the approaches is negligible.

System unreliability, proportional evaluation

This analysis evaluates system reliability when varying the ratio between R_v and R_m, with R_v fixed to 10^{-5}. In this work we propose the same evaluation introduced in [3] in order to compare our automated approach with previous results. The configurations with 1, 2 and 3 voters are described in detail in Table I. The results of this analysis are reported in Figure 8, where it is shown that the configuration with 3 voters performs better than the others. Moreover, it can be noticed that the standard 1 voter setting is an interesting choice only if the reliability of the voter is not less than $10^2 * R_m$.

System reliability, varying non-uniform probabilities

The evaluation of system reliability is clearly influenced by the probability distribution of failures that characterizes each single component. In view of this fact, we propose an evaluation of system reliability by varying non-uniform distributions for two specific settings. In detail, we analyze the standard TMR chain with 1 voter, described in Table III configuration (h), and one chain with 2 voters explained by configuration (c) of Table V.

In the first case, we consider the probability of failure for M_1 as $k * F_m$, with k varying from $1/2$ to 2. Figure 11a shows the results of this analysis. It is possible to notice that such TMR configurations have an impact on system reliability only when F_m is significantly bigger than F_v. In particular, the probability of system failure is influenced only when $F_m > 10^2 * F_v$.

Figure 11b shows the evaluation on the configuration with 2 voters. In this case, we consider the probability of failure for V_1 as $k * F_v$, with k varying from $1/4$ to 4. Differently from previous analysis, the impact on the reliability of the system is significant only when $F_m < 10 * F_v$. This result can be explained by the fact that, in this area, the reliability of the voter is close to the reliability of the module.

VI. Conclusions and Future Work

In this paper we presented a novel flow for the automated analysis of architectures for reliability. The approach is based on the use of uninterpreted functions, and allows to model different architectural solutions without specific commitment

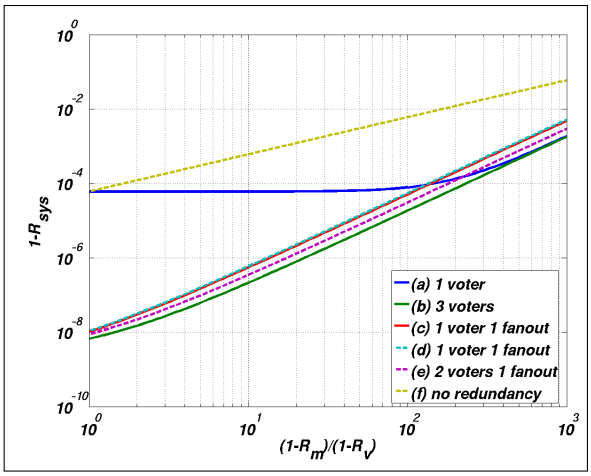


Fig. 8: System reliability: proportional evaluation

to the nature of the blocks being combined. The flow is supported by a tool that provides for the automated generation of the fault trees for the case where the redundancy fails, and computes a closed form of the reliability function. We carried out an extensive experimental evaluation of the approach: we are able to automatically obtain all the results that are manually obtained in [1], and we extend them to the cases of non-uniform probabilities and of two voters per stage.

In the future, we plan to work along the following directions. First, we will increase the scalability of the approach, by investigating various forms of symmetry breaking, and compositional reasoning techniques based on predicate abstraction [38]. Second, we will consider the analysis of sequential systems, that may require the use of probabilistic model checking of Markov decision processes [39]. Finally, we will work to integrate within the framework a search procedure for advanced synthesis of optimal configurations.

References

[1] M. Hamamatsu, T. Tsuchiya, and T. Kikuno, "On the reliability of cascaded tmr systems," in *Dependable Computing (PRDC), 2010 IEEE 16th Pacific Rim International Symposium on*. IEEE, 2010, pp. 184–190.

[2] I. B. M. Corporation, "Saturn v – launch vehicle digital computer: Simplex models," NASA, Technical Note NASA Part No. 50M35010, Nov. 1964. [Online]. Available: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19730063841_1973063841.pdf

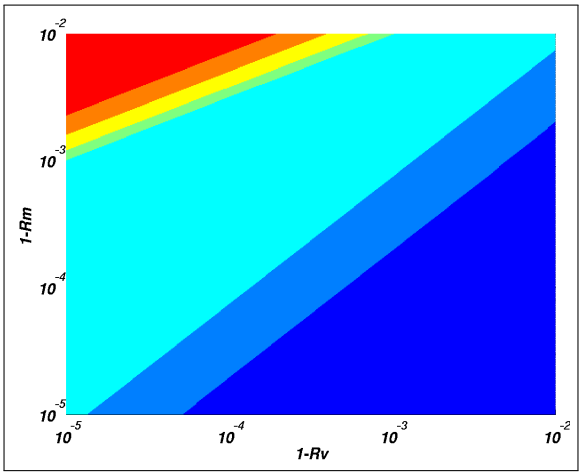
[3] S. Lee, J. il Jung, and I. Lee, "Voting structures for cascaded triple modular redundant modules," *Ieice Electronic Express*, vol. 4, no. 21, pp. 657–664, 2007.

[4] T. Lanfang, T. Qingping, and L. Jianli, "Specification and verification of the triple-modular redundancy fault tolerant system using csp," in *DEPEND 2011, The Fourth International Conference on Dependability*, 2011, pp. 14–17.

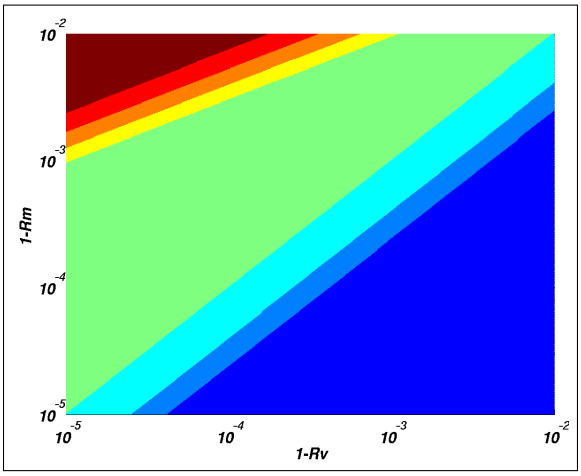
[5] M. Zhang, Z. Liu, C. Morisset, and A. Ravn, "Design and verification of fault-tolerant components," *Methods, Models and Tools for Fault Tolerance*, pp. 57–84, 2009.

[6] A. Joshi, M. Whalen, and M. P. Heimdahl, "Modelbased safety analysis: Final report," Tech. Rep., 2005.

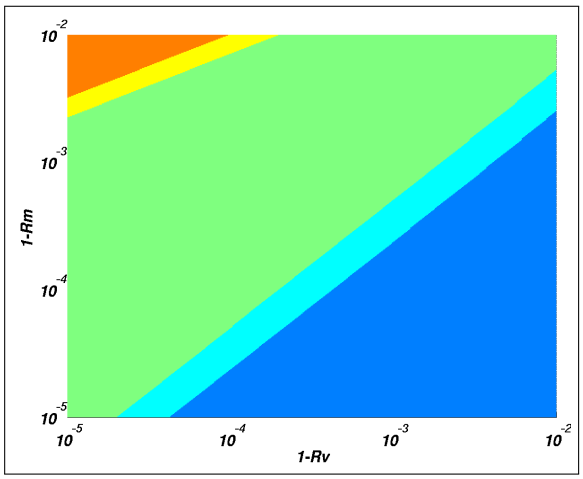
[7] M. Bozzano, A. Villafiorita, O. Åkerlund, P. Bieber, C. Bougnol, E. Böde, M. Bretschneider, A. Cavallo, C. Castel, M. Cifaldi *et al.*, "Esacs: an integrated methodology for design and safety analysis of complex systems," *Proc. ESREL 2003*, pp. 237–245, 2003.



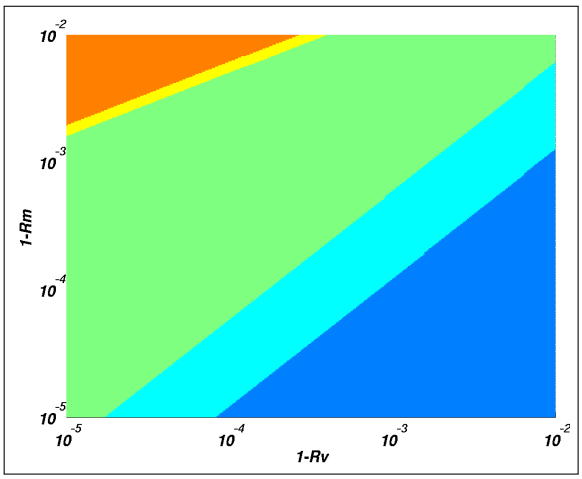
(a) 1 voter: uniform probability



(b) 1 voter: non-uniform probability



(c) 2 voters: uniform probability



(d) 2 voters: non-uniform probability

Fig. 9: Find best for 1 and 2 voters, uniform and non-uniform probability

Architecture color	Array of configurations
(a) *blue*	[1, 1, 1, 1, 1, 1, 1, 1]
(b) *blue/lightblue*	[1, 1, 1, 1, 2, 3, 1, 2]
(c) *lightblue*	[1, 2, 3, 1, 2, 3, 1, 2]
(d) *green*	[3, 2, 1, 0, 3, 2, 1, 0]
(e) *yellow*	[3, 0, 3, 0, 3, 0, 3, 0]
(f) *orange*	[0, 0, 3, 0, 0, 0, 3, 0]
(g) *red*	[0, 0, 0, 0, 0, 0, 0, 0]

TABLE II: Configurations for 1 voters uniform probability

Architecture color	Array of configurations
(a) *blue*	[3, 3, 3, 3, 3, 3, 3, 3]
(b) *blue/lightblue*	[3, 3, 3, 2, 3, 3, 3, 2]
(c) *lightblue*	[3, 2, 3, 1, 3, 2, 3, 1]
(d) *green*	[3, 1, 3, 2, 3, 1, 3, 2]
(e) *yellow*	[3, 2, 1, 0, 3, 2, 1, 0]
(f) *orange*	[3, 0, 3, 0, 3, 0, 3, 0]
(g) *red*	[0, 0, 3, 0, 0, 0, 3, 0]
(h) *darkred*	[0, 0, 0, 0, 0, 0, 0, 0]

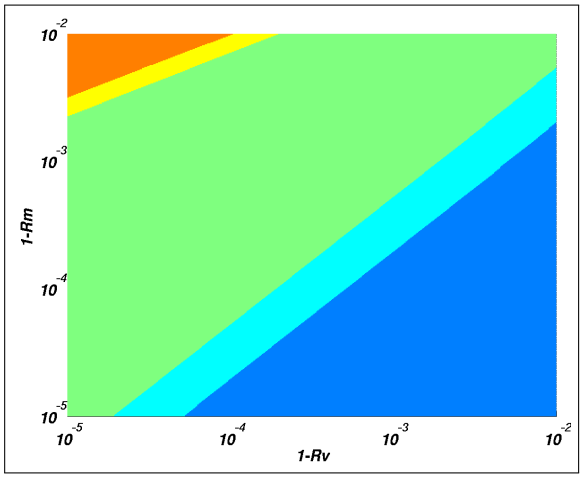
TABLE III: Configurations for 1 voter non-uniform probability

Architecture color	Array of configurations
(a) *blue*	[4, 4, 4, 4, 4, 4, 4, 4]
(b) *lightblue*	[4, 4, 3, 2, 4, 4, 3, 2]
(c) *green*	[4, 3, 2, 3, 4, 3, 2, 3]
(d) *yellow*	[1, 1, 1, 4, 1, 1, 1, 4]
(e) *orange*	[1, 1, 1, 1, 1, 1, 1, 1]

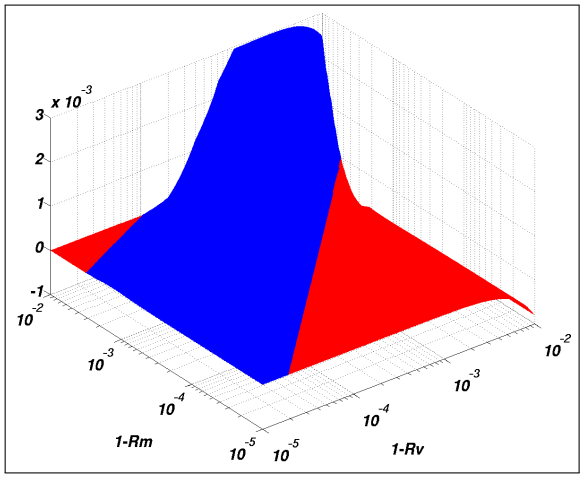
TABLE IV: Configurations for 2 voters uniform probability

Architecture color	Array of configurations
(a) *blue*	[4, 4, 4, 4, 4, 4, 4, 4]
(b) *lightblue*	[3, 3, 2, 3, 3, 3, 2, 3]
(c) *green*	[2, 3, 4, 2, 3, 4, 2, 3]
(d) *yellow*	[0, 4, 0, 0, 4, 0, 0, 4]
(e) *orange*	[0, 0, 0, 0, 0, 0, 0, 0]

TABLE V: Configurations for 2 voter non-uniform probability

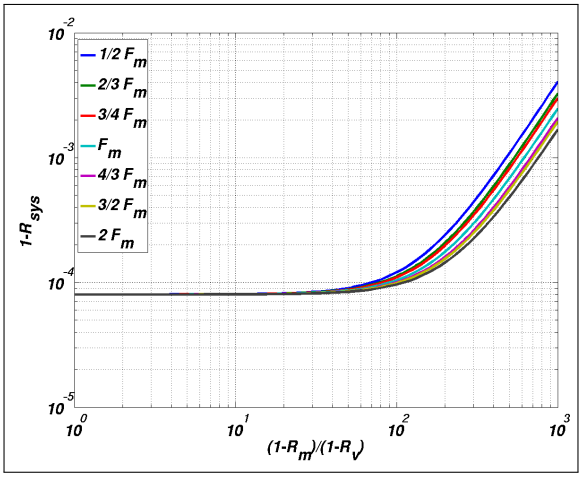


(a) 1 and 2 voters comparison

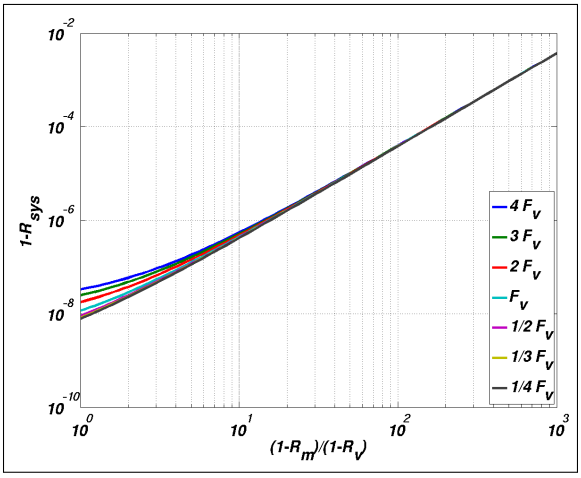


(b) 1 voter vs 2 voters (blue means 2v is better)

Fig. 10: 1 voter vs 2 voters



(a) Varying F_m for M_1



(b) Varying F_v for V_1

Fig. 11: System reliability when varying non-uniform probability

Architecture color	Array of configurations
(a) *blue*	[1, 1, 1, 1, 1, 1, 1, 1] (1v)
(b) *lightblue*	[1, 1, 1, 1, 2, 3, 1, 2] (1v)
(c) *green*	[4, 3, 2, 3, 4, 3, 2, 3] (2v)
(d) *yellow*	[1, 1, 1, 4, 1, 1, 1, 4] (2v)
(e) *orange*	[0, 0, 0, 0, 0, 0, 0, 0] (1v)

TABLE VI: Configurations for 1 voter vs. 2 voters

[8] O. Akerlund, P. Bieber, E. Boede, M. Bozzano, M. Bretschneider, C. Castel, A. Cavallo, M. Cifaldi, J. Gauthier, A. Griffault *et al.*, "Isaac, a framework for integrated safety analysis of functional, geometrical and human aspects," *Proc. ERTS*, vol. 2006, 2006.

[9] M. Bozzano and A. Villafiorita, *Design and Safety Assessment of Critical Systems*. CRC Press (Taylor and Francis), an Auerbach Book, 2010.

[10] G. J. Holzmann, "The model checker spin," *Software Engineering, IEEE Transactions on*, vol. 23, no. 5, pp. 279–295, 1997.

[11] K. McMillan, *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[12] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," *Advances in Computers*, vol. 58, pp. 117–148, 2003.

[13] K. L. McMillan, "Interpolation and sat-based model checking," in *CAV*, ser. Lecture Notes in Computer Science, W. A. H. Jr. and F. Somenzi, Eds., vol. 2725. Springer, 2003, pp. 1–13.

[14] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[15] J. P. M. Silva, I. Lynce, and S. Malik, "Conflict-driven clause learning sat solvers," in *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, 2009, vol. 185, pp. 131–153.

[16] The FSAP/NuSMV-SA platform. `http://es.fbk.eu/tools/FSAP`.

[17] M. Bozzano and A. Villafiorita, "The FSAP/NuSMV-SA Safety Analysis Platform," *Software Tools for Technology Transfer*, vol. 9, no. 1, pp. 5–24, 2007.

[18] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri, "Safety, dependability, and performance analysis of extended AADL models," *The Computer Journal*, vol. doi: 10.1093/com, March 2010.

[19] M. Bozzano and A. Villafiorita, "The fsap/nusmv-sa safety analysis platform," *STTT*, vol. 9, no. 1, pp. 5–24, 2007.

[20] M. Bozzano, A. Cimatti, and F. Tapparo, "Symbolic fault tree analysis for reactive systems," in *ATVA*, ser. Lecture Notes in Computer Science, K. S. Namjoshi, T. Yoneda, T. Higashino, and Y. Okamura, Eds., vol. 4762. Springer, 2007, pp. 162–176.

[21] M. Bozzano, A. Cimatti, O. Lisagor, C. Mattarei, S. Mover, M. Roveri, and S. Tonetta, "Symbolic model checking and safety assessment of altarica models," *ECEASST*, vol. 46, 2011.

[22] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, 2009, vol. 185, pp. 825–885.

[23] L. M. de Moura, S. Owre, H. Rueß, J. M. Rushby, N. Shankar, M. Sorea, and A. Tiwari, "Sal 2," in *CAV*, ser. Lecture Notes in Computer Science, R. Alur and D. Peled, Eds., vol. 3114. Springer, 2004, pp. 496–500.

[24] S. Tonetta, "Abstract model checking without computing the abstraction," in *FM*, ser. Lecture Notes in Computer Science, A. Cavalcanti and D. Dams, Eds., vol. 5850. Springer, 2009, pp. 89–105.

[25] A. Cimatti, S. Mover, and S. Tonetta, "Smt-based verification of hybrid systems," in *AAAI*, J. Hoffmann and B. Selman, Eds. AAAI Press, 2012.

[26] ——, "Smt-based scenario verification for hybrid systems," *Formal Methods in System Design*, vol. 42, no. 1, pp. 46–66, 2013.

[27] S. Bensalem, V. Ganesh, Y. Lakhnech, C. Munoz, S. Owre, H. Rueß, J. Rushby, V. Rusu, H. Saıdi, N. Shankar *et al.*, "An overview of sal," in *Proceedings of the 5th NASA Langley Formal Methods Workshop*, 2000.

[28] M. Fränzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert, "Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 1, no. 3-4, pp. 209–236, 2007.

[29] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NuSMV : a new symbolic model checker," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 2, no. 4, pp. 410–425, Mar. 2000.

[30] The NuSMV model checker. `http://nusmv.fbk.eu`.

[31] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani, "Mathsat: Tight Integration of SAT and Mathematical Decision Procedures," *Journal of Automated Reasoning*, vol. 35, pp. 265–293, 2005.

[32] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani, "The mathsat5 smt solver," in *TACAS*, ser. Lecture Notes in Computer Science, N. Piterman and S. A. Smolka, Eds., vol. 7795. Springer, 2013, pp. 93–107.

[33] J. A. Abraham and D. P. Siewiorek, "An algorithm for the accurate reliability evaluation of triple modular redundancy networks," *Computers, IEEE Transactions on*, vol. 100, no. 7, pp. 682–692, 1974.

[34] D. D. Thaker, R. Amirtharajah, F. Impens, I. Chuang, and F. T. Chong, "Recursive tmr: Scaling fault tolerance in the nanoscale era," *Design & Test of Computers, IEEE*, vol. 22, no. 4, pp. 298–305, 2005.

[35] J. M. Johnson and M. J. Wirthlin, "Voter insertion algorithms for fpga designs using triple modular redundancy," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2010, pp. 249–258.

[36] T. Anderson and P. A. Lee, *Fault tolerance, principles and practice*. Prentice/Hall International, 1981.

[37] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl, *Fault Tree Handbook*. Nuclear Regulatory Commission, 1981, no. NUREG-0492.

[38] A. Cimatti, J. Dubrovin, T. A. Junttila, and M. Roveri, "Structure-aware computation of predicate abstraction," in *FMCAD*. IEEE, 2009, pp. 9–16.

[39] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri, "Safety, dependability and performance analysis of extended aadl models," *Comput. J.*, vol. 54, no. 5, pp. 754–775, 2011.

[40] A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds., *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, 2009.