Efficient Analysis of Cyclic Redundancy Architectures via Boolean Fault Propagation

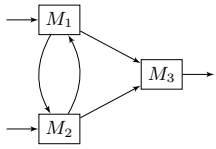
Marco Bozzano, Alessandro Cimatti,
Alberto Griggio, and Martin Jonáš

Fondazione Bruno Kessler, Trento, Italy
{cimatti,bozzano,griggio,mjonas}@fbk.eu

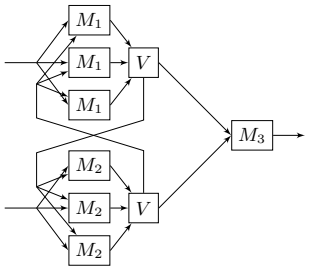
Abstract. Many safety critical systems guarantee fault-tolerance by using several redundant copies of their components. When designing such redundancy architectures, it is crucial to analyze their fault trees, which describe combinations of faults of individual components that may cause malfunction of the system. State-of-the-art techniques for fault tree computation use first-order formulas with uninterpreted functions to model the transformations of signals performed by the redundancy system and an AllSMT query for computation of the fault tree from this encoding. Scalability of the analysis can be further improved by techniques such as predicate abstraction, which reduces the problem to Boolean case.
In this paper, we show that as far as fault trees of redundancy architectures are concerned, signal transformation can be equivalently viewed in a purely Boolean way as fault propagation. This alternative view has important practical consequences. First, it applies also to general redundancy architectures with cyclic dependencies among components, to which the current state-of-the-art methods based on AllSMT are not applicable, and which currently require expensive sequential reasoning. Second, it allows for a simpler encoding of the problem and usage of efficient algorithms for analysis of fault propagation, which can significantly improve the runtime of the analyses. A thorough experimental evaluation demonstrates the superiority of the proposed techniques.

1 Introduction

Fault-tolerance is a fundamental property of safety critical systems that enables their safe operation even in the presence of faults. There are many ways to ensure fault-tolerance, often based on redundancy: spare parts are available for backup and are ready to take over with different degrees of promptness (e.g., hot/warm/cold standby), or with multiple replicas running in parallel. The latter is a common approach to fault-tolerance in computer-based control systems, where the results computed by the independent replicas are combined together by means of voters. The idea dates back to the pioneering space application in Saturn Launch Vehicle [12], and has then been adopted in the Primary Flight Computer [19] of the Boeing 777. The idea is becoming prominent with the advent of modern Integrated Modular Avionics [16], a cost-effective solution for the management of highly intensive software control systems.

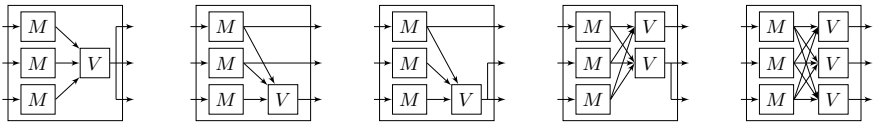


(a) Reference non-redundant system.



(b) TMR redundant system with three replicas of modules M_1, M_2, whose results are combined by a voter.

Fig. 1: Network of computational modules with cyclic dependencies, extended by triple modular redundancy.



(a) V111 (b) V001 (c) V011 (d) V122 (e) V123

Fig. 2: Selected ways of extending a single reference module M with triple modular redundancy (using 1, 2, and 3 voters) [6].

One of the most used instances of the approach to redundancy by using module replicas is the *triple modular redundancy* (TMR) schema, in which the computational modules are replaced by three redundant copies, whose results can be combined by one to three voters. An example of using TMR to add redundancy to a reference non-redundant architecture is shown in Figure 1. Note that there are multiple ways of combining the results of a single triplicated computational module by voters, some of which are shown in Figure 2 [6].

Assessing the actual degree of fault-tolerance of a redundant architecture is directly related to the construction and analysis of the corresponding fault tree [17]. A fault tree describes the combinations of failures of individual components that may cause higher-level malfunction, e.g., bring the system into a dangerous state. Such combinations are traditionally called *cut sets*. Given the set of all cut sets of the system, a fault tree can be reconstructed. Subsequently, from the fault tree expressed as a Binary Decision Diagram, it is possible to compute the reliability of the system from the reliability measures of the components, and to synthesize the analytical form of the reliability function [6].

In this paper, we tackle the problem of automatically analyzing the reliability of redundancy architectures with parallel replicas and voting. We propose a general framework that encompasses also redundancy architectures with cyclic dependencies among components, such as the system from Figure 1, to which

current state-of-the-art approaches [6] are not applicable. The modeling is based on symbolic transition systems over the quantifier-free theory of linear real arithmetic and uninterpreted functions (UFLRA). In particular, real numbers are used to represent the signals of the architecture and multiple instances of the same uninterpreted function symbol are used to represent component replicas. The modeling framework is a strict generalization of the combinational approach proposed in [4,5], that only allows for acyclic architectures.

As the main contribution, we propose an analysis technique based on the reduction to *fault propagation graphs* over Boolean structures [7]. We prove that the reduction is correct: the signal transformation performed by a redundancy architecture can be equivalently viewed in a Boolean way as fault propagation.

We carry out a systematic experimental evaluation on the set of redundancy architectures with cyclic dependencies to evaluate scalability of the proposed solution. Moreover, we perform evaluation on acyclic redundancy architectures to compare the performance against the state-of-the-art approach based on predicate abstraction [5,6], which can be applied only to redundancy architectures without cycles. The proposed approach proves to be very scalable, being able to analyze cyclic architectures with thousands of nodes, and is dramatically more efficient than a direct reduction to model checking of symbolic transition systems over UFLRA. In the restricted set of acyclic benchmarks, the proposed approach provides better performance even over the optimized method proposed in [5] and extended in [6] that adopts a structural form of predicate abstraction to improve over basic AllSMT [14].

The paper is structured as follows. In Section 2, we present logical preliminaries and basic notions of fault propagation graphs. In Section 3, we describe the framework of redundancy architectures with cycles. In Section 4, we present the reduction to fault propagation and prove its correctness. In Section 5, we discuss the related work. The experiments are presented in Section 6. In Section 7, we draw some conclusions and discuss some directions for future work.

2 Preliminaries

2.1 General Background

In this section, we explain the basic mathematical conventions that are used in the paper. We assume that the reader is familiar with standard first-order logic and the basic ideas of Satisfiability Modulo Theories (SMT), as presented e.g. in [1]. A theory in the SMT sense is a pair $(\Sigma, \mathcal{C})$, where Σ is a first-order signature and $\mathcal{C}$ is a class of models over Σ. We use the standard notions of interpretation, assignment, model, satisfiability, validity, and logical consequence. We refer to 0-arity predicates as Boolean variables, and to 0-arity uninterpreted functions as (theory) variables. We denote variables with $x, y, \ldots$, formulas with $\varphi, \psi, \ldots$, and uninterpreted functions with $f, g, \ldots$, possibly with subscripts. We denote vectors with $\bar{\cdot}$ (e.g. $\overline{x}$), and individual components with subscripts (e.g. x_j). We denote the domain of Booleans with $\mathbb{B} = \{\top, \bot\}$. If $x_1, \ldots, x_n$ are variables and

φ is a formula, we write $\varphi(x_1, \ldots, x_n)$ to indicate that all the variables occurring free in φ are in $x_1, \ldots, x_n$. If φ is a formula without uninterpreted functions and μ is a function that maps each free variable of φ to a value of the corresponding sort, $[\![\varphi]\!]_\mu$ denotes the result of the evaluation of φ under this assignment. A Boolean formula is called *positive* if it does not use other logical connectives than conjunctions and disjunctions.

In this paper, we shall use the theory of linear real arithmetic (LRA), in which the numeric constants and the arithmetic and relational operators have their standard meaning, extended with uninterpreted functions (UF), whose interpretation is not fixed in $\mathcal{C}$, and with *voters* (V), which are k-ary functions whose interpretation is the majority function defined as below. For simplicity, we consider only voters with odd arity as even-arity voters are rarely used in practice. However, our approach can be extended to support even-arity voters.

Definition 1. *The k-ary majority function majority*$\colon \mathbb{R}^k \to \mathbb{R}$ *for an odd $k > 0$ is defined by majority$(\overline{x}) = y$ if there is y such that $y = x_j$ for at least $\lceil k/2 \rceil$ distinct j and majority$(\overline{x}) = x_1$ otherwise.*

Given a set of variables $\overline{x}$, we denote with $\overline{x}'$ the set $\{x' \mid x \in \overline{x}\}$. A *symbolic transition system* S is a triple $(\overline{x}, I(\overline{x}), T(\overline{x}, \overline{x}'))$, where $\overline{x}$ is a set of variables, and $I(\overline{x})$, $T(\overline{x}, \overline{x}')$ are formulae over some signature. An assignment to the variables in $\overline{x}$ is a *state* of S. A state s is initial iff it is a model of $I(\overline{x})$, i.e., $s \models I(\overline{x})$. The states s, s' denote a transition iff $s \cup s' \models T(\overline{x}, \overline{x}')$, also written $T(s, s')$. A *trace* is a sequence of states $s_0, s_1, \ldots$ such that s_0 is initial and $T(s_i, s'_{i+1})$ for all i. We denote traces with π, and with π_j the j-th element of π. A state s is *reachable* in S iff there exists a trace π such that $\pi_i = s$ for some i.

2.2 Fault Propagation Graphs

In this section we briefly introduce the necessary notions of fault propagation, and in particular the formalism of *symbolic fault propagation graphs*. Intuitively, fault propagation graphs can be used to describe how failures of some components of a given system can cause the failure of other components of a system. In an explicit (hyper)graph representation, components can be represented by nodes, and dependencies by edges among them, with the meaning that an edge from component c_1 to component c_2 states that the failure of c_1 can cause the failure (propagation) of c_2. In the symbolic representation adopted here, we model components as Boolean variables (where $\bot$ means "not failed" and $\top$ means "failed"), and express the dependencies as Boolean formulae encoding the conditions that can lead to the failure of each component. The basic concepts are formalized in the following definitions. For more information, we refer to [7].

Definition 2 (Fault propagation graph). *A symbolic fault propagation graph (*FPG*) is a pair $(C, canFail)$, where C is a finite set of* system components *and canFail is a function that assigns to each component c a Boolean formula canFail(c) over the set of variables C.*

Definition 3 (Trace of FPG). *Let G be a fault propagation graph $(C, canFail)$. A* state *of G is a function from C to $\mathbb{B}$. A trace of G is a sequence of states $\pi = \pi_0\pi_1 \ldots \in (\mathbb{B}^C)^\omega$ such that all $i > 0$ and $c \in C$ satisfy (i) $\pi_i(c) = \pi_{i-1}(c)$ or (ii) $\pi_{i-1}(c) = \bot$ and $\pi_i(c) = [\![canFail(c)]\!]_{\pi_{i-1}}$.*

Example 1 ([7]). Consider a system with components *control on ground* (G), *hydraulic control* (H), and *electric control* (E) such that G can fail if both H and E have failed, H can fail if E has failed, and E can fail if H has failed. This system can be modeled by a fault propagation graph $(\{\text{G}, \text{E}, \text{H}\}, canFail)$, where $canFail(\text{G}) = \text{H} \wedge \text{E}$, $canFail(\text{H}) = \text{E}$, and $canFail(\text{E}) = \text{H}$.

One of the traces of this system is $\{\text{G} \mapsto \bot, \text{H} \mapsto \top, \text{E} \mapsto \bot\}\{\text{G} \mapsto \bot, \text{H} \mapsto \top, \text{E} \mapsto \top\}\{\text{G} \mapsto \top, \text{H} \mapsto \top, \text{E} \mapsto \top\}^\omega$, where H is failed initially, which causes failure of E in the second step, and the failures of H and E together cause a failure of G in the third step.

Fault propagation graphs are often used to identify sets of initial faults that can lead the system to a dangerous or unwanted state (usually called a *top level event*). Such sets of initial faults are called *cut sets*.

Definition 4 (Cut set). *Let G be a fault propagation graph $G = (C, canFail)$ and φ a positive Boolean formula, called* top level event. *The assignment $cs\colon C \to \mathbb{B}$ is called a* cut set *of G for φ if there is a trace π of G that starts in the state cs and there is some $k \geq 0$ such that $\pi_k \models \varphi$. A cut set cs is called* minimal cut set *if it is minimal with respect to the pointwise ordering of functions $\mathbb{B}^C$, i.e., there is no other cut set cs' such that $\{c \in C \mid cs'(c) = \top\} \subsetneq \{c \in C \mid cs(c) = \top\}$.*

For brevity, when talking about cut sets, we often mention only the components that are set to $\top$ by the cut set.

Example 2 ([7]). The minimal cut sets of the FPG from Example 1 for the top level event $\varphi = \text{G}$ are $\{\text{G}\}$, $\{\text{H}\}$, and $\{\text{E}\}$. These three cut sets are witnessed by the following traces:

1. $\{\text{G} \mapsto \top, \text{H} \mapsto \bot, \text{E} \mapsto \bot\}^\omega$,
2. $\{\text{G} \mapsto \bot, \text{H} \mapsto \top, \text{E} \mapsto \bot\}\{\text{G} \mapsto \bot, \text{H} \mapsto \top, \text{E} \mapsto \top\}\{\text{G} \mapsto \top, \text{H} \mapsto \top, \text{E} \mapsto \top\}^\omega$,
3. $\{\text{G} \mapsto \bot, \text{H} \mapsto \bot, \text{E} \mapsto \top\}\{\text{G} \mapsto \bot, \text{H} \mapsto \top, \text{E} \mapsto \top\}\{\text{G} \mapsto \top, \text{H} \mapsto \top, \text{E} \mapsto \top\}^\omega$.

Note that the FPG has also other cut sets, such as $\{\text{G}, \text{E}\}$, $\{\text{H}, \text{E}\}$, and $\{\text{G}, \text{H}, \text{E}\}$, which are not minimal.

In the following, we work with fault propagation graphs whose all *canFail* formulas are positive. Such fault propagation graphs are called *monotone*. Note that the definition of trace ensures that in each trace, if a component c is set to $\top$ in a state π_i, it is $\top$ in all the subsequent states π_j for $j > i$. This ensures that each trace eventually reaches a fixed point. Moreover, before reaching this fixed point, the trace can contain at most $|C|$ distinct states.

For monotone FPGs, there is an efficient algorithm for minimal cut set enumeration [7]. This approach consists in enumerating of the minimal models of a specific LRA formula, in which theory constraints are used only if the input FPG contains cycles (and which therefore is purely Boolean for acyclic FPGs).

3 Cyclic Redundancy Architectures

In this section, we describe the framework adopted to model redundancy architectures, in form of a restricted class of symbolic transition systems modulo UFLRA. We call this restricted class *transition systems with uninterpreted functions and voters (UF+V TS)*.[1] This modeling framework is more expressive than mere SMT formulas modulo UFLRA, which were used in the previous works on analysis of redundancy architectures [6], as it can express architectures that contain cyclic dependencies among the modules.

Definition 5 (UF+V transition system). *A* transition system with uninterpreted functions and voters *is a tuple* $(V_S, V_{\text{in}}, V_{\text{init}}, T_{\text{next}}, T_{\text{init}})$*, where*

- V_S *is a finite set of real-valued* signal variables*;*
- V_{in} *with* $V_S \cap V_{\text{in}} = \emptyset$ *is a finite set of real-valued* input variables*;*
- V_{init} *is a finite set of real-valued* initial value variables*;*
- $T_{\text{next}} \colon V_S \to$ *Expr is a transition function, where Expr is the set of all expressions of form* $f(x_1, x_2, \ldots, x_k)$ *for* $k \geq 0$*,* $x_i \in (V_S \cup V_{\text{in}})$*, and where* f *is either an uninterpreted function symbol of arity* k *or the function symbol* voter_k *with an odd* $k > 0$*;*
- T_{init} *is an* initial value mapping *that assigns an initial value variable* $T_{\text{init}}(v) \in V_{\text{init}}$ *to each signal* $v \in V_S$ *for which* $T_{\text{next}}(v) = f(\overline{x})$ *for an uninterpreted* f*.*

A UF+V transition system is called *well formed* if it does not contain cyclic dependencies among voters, i.e., there is no sequence $v_1 \ldots v_n$ of signal variables such that $v_1 = v_n$ and each v_i with $i > 0$ satisfies $T_{\text{next}}(v_i) = voter_k(x_1, \ldots, x_k)$ with $x_j = v_{i-1}$ for some $1 \leq j \leq k$. For well formed UF+V TS, we can define *voter depth* $vd \colon V_S \cup V_{\text{in}} \to \mathbb{N}$ as the unique solution to the following set of equations: $vd(in) = 0$ for each $in \in V_{\text{in}}$, $vd(s) = 0$ for each $v \in V_S$ such that $T_{\text{next}}(v) = f(x_1, x_2, \ldots, x_k)$, and $vd(v) = \max\{vd(x_i) \mid 1 \leq i \leq k\} + 1$ for each $v \in V_S$ such that $T_{\text{next}}(v) = voter_k(x_1, x_2, \ldots, x_k)$.

In the rest of the paper, we assume that all UF+V TS are well formed. In the rest of this section, let us fix an arbitrary well formed UF+V transition system $S = (V_S, V_{\text{in}}, V_{\text{init}}, T_{\text{next}}, T_{\text{init}})$.

We now give a formal definition of the behavior of the UF+V system in presence of faults. Intuitively, we are given the set Faults of faulty signal-producing components of the system, which do not have to behave correctly: a faulty component neither has to start in its specified initial value nor respect its transition function.

Definition 6 (Trace of UF+V TS). *A state of a UF+V transition system* S *is an arbitrary assignment of real numbers to signal and input variables* $s \colon (V_S \cup V_{\text{in}}) \to \mathbb{R}$*.*

[1] Note than although UF+V TS and the related concepts can be defined directly in terms of UFLRA symbolic transition systems, we chose to make the definition explicit to simplify the presentation and proofs.

The sequence of states $\pi = \pi_0\pi_1 \ldots \in (\mathbb{R}^{V_S \cup V_{\text{in}}})^\omega$ *is called a* trace of the system S for the fault set $\mathsf{Faults} \subseteq V_S$, input stream $\iota = \iota_0\iota_1 \ldots \in (\mathbb{R}^{V_{\text{in}}})^\omega$, initial value assignment $\text{Init}\colon V_{\text{init}} \to \mathbb{R}$, and interpretation $[\![\_]\!]$, *which to each uninterpreted function symbol of arity* k *assigns a function* $[\![f]\!]\colon \mathbb{R}^k \to \mathbb{R}$, *if:*

- $\pi_i(in) = \iota_i(in)$ *for all* $i \geq 0$ *and* $in \in V_{\text{in}}$.
- *For* $v \in V_S \setminus \mathsf{Faults}$ *such that* $T_{\text{next}}(v) = f(x_1, \ldots, x_k)$ *with an uninterpreted function symbol* f, *it is the case that* $\pi_0(v) = \text{Init}(T_{\text{init}}(v))$ *and all* $i > 0$ *satisfy* $\pi_i(v) = [\![f]\!](\pi_{i-1}(x_1), \ldots, \pi_{i-1}(x_k))$.
- *For all* $i \geq 0$ *and* $v \in V_S \setminus \mathsf{Faults}$ *such that* $T_{\text{next}}(v) = voter_k(x_1, \ldots, x_k)$, *it is the case that* $\pi_i(v) = majority(\pi_i(x_1), \ldots, \pi_i(x_k))$.

Traces for the fault set $\mathsf{Faults} = \emptyset$ *are called* nominal.

Note that each uninterpreted module needs one time step to compute its result, while the results of voters are instantaneous. The time delay for modules allows cyclic dependencies among modules, while no delay for voters gives the expected semantics to architectures where some replicas of a module are guarded by a voter and others are not, such as in schemas from Figures 2b and 2c.

Example 3. Consider the example from Figure 1, where the reference system with 3 modules M_1, M_2, and M_3 is extended with TMR such that the modules M_1 and M_2 are replaced by three replicas whose results are combined by a voter.

We can represent the redundancy version of the system as a UF+V TS as follows. The nominal behavior of the modules M_1, M_2, and M_3 is represented by binary uninterpreted functions f_1, f_2, and f_3, respectively. Further, we represent initial values of M_1, M_2, M_3 by variables $init_{m_1}$, $init_{m_2}$, and $init_{m_3}$ respectively. Finally, we represent the output of i-th replica of each module M_j by a signal variable x_j^i and the output of the voter corresponding to the module M_j by a signal variable x_j^v.

This gives the UF+V transition system $S = (V_S, \{in_1, in_2\}, V_{\text{init}}, T_{\text{next}}, T_{\text{init}})$, with $V_S = \{x_1^1, x_1^2, x_1^3, x_1^v, x_2^1, x_2^2, x_2^3, x_2^v, x_3^1\}$, $V_{\text{init}} = \{init_{m_j} \mid j \in \{1,2,3\}\}$, and

$$T_{\text{next}}(x_1^i) = f_1(in_1, x_2^v) \text{ for } 1 \leq i \leq 3, \qquad T_{\text{init}}(x_1^i) = init_{m_1} \text{ for } 1 \leq i \leq 3,$$
$$T_{\text{next}}(x_2^i) = f_2(in_2, x_1^v) \text{ for } 1 \leq i \leq 3, \qquad T_{\text{init}}(x_2^i) = init_{m_2} \text{ for } 1 \leq i \leq 3,$$
$$T_{\text{next}}(x_3^1) = f_3(x_1^v, x_2^v), \qquad T_{\text{init}}(x_3^1) = init_{m_3},$$
$$T_{\text{next}}(x_j^v) = voter_3(x_j^1, x_j^2, x_j^3) \text{ for } j \in \{1,2\}.$$

We define the class of *redundancy* transition systems, where the only purpose of all voters is to recognize and repair outputs of failed components; more specifically, if all components behave correctly, the voters are not necessary.

Definition 7 (Redundancy UF+V TS). *We call the system* S *a* redundancy UF+V transition system *if in all its nominal traces, all inputs of each voter are always identical. Formally, if* π *is any nominal trace of* S *and if* v *is a variable for which* $T_{\text{next}}(v) = voter_k(\overline{x})$, *then* $\left|\{\pi_i(x_j) \mid 1 \leq j \leq k\}\right| = 1$ *for all* $i \geq 0$.

Similarly to FPGs, a cut set is a set of faults that leads to the undesired behavior of the system. In particular, given a set of signals that are considered as *output signals* (or *outputs*) of the system, a cut set of the given UF+V TS is a set of faults that can cause an incorrect value of at least one output.

Definition 8 ((Minimal) cut set). *A fault set* $\mathsf{Faults} \subseteq V_S$ *is called a* cut set *of* S *for a set of output signals* $V_{\text{out}} \subseteq V_S$ *if there exist an input stream, initial value assignment, and an interpretation such that values of output signals of some trace* π *for the fault set* Faults *differ from the outputs of the nominal trace* π^{nom} *with the same input stream, initial values, and interpretation, i.e., there is* $c \geq 0$ *and* $o \in V_{\text{out}}$ *for which* $\pi_c(o) \neq \pi_c^{nom}(o)$*. A cut set is called minimal (*MCS*) if it is minimal in terms of set inclusion.*

Since the redundancy UF+V TS form a subclass of UFLRA transition systems, there is a straightforward procedure for minimal cut set enumeration. As in the case of combinational systems [6], one can construct a *miter system*, which consists of two copies of the architecture: the first is allowed to fail and the second is constrained to behave nominally. Minimal cut sets can then be obtained by using a technique based on symbolic model checking [3] to enumerate all minimal assignments to fault variables under which it is possible to reach some state in which the outputs of the two copies differ.

4 Reducing Redundancy UF+V TS to Fault Propagation Graphs

In this section, we show the main result of the paper, which is that minimal cut set enumeration of redundancy UF+V transition systems can be reduced to minimal cut set enumeration of Boolean fault propagation graphs, which is more efficient than MCS enumeration based on miter construction and model checking.

4.1 Reduction

We for each UF+V system S define a corresponding FPG S^B. The components of S^B correspond to the signal variables of the original system S. With a slight abuse of notation, we use the same names for the original real-valued signal variables of S and the components of S^B, although they have different types. Intuitively, the reduction ensures that each component v of S^B can fail if and only if there is a trace of S in which the value of the signal variable v deviates from its nominal value.

Definition 9. *Let* $S = (V_S, V_{\text{in}}, V_{\text{init}}, T_{\text{next}}, T_{\text{init}})$ *be a UF+V TS. We define a corresponding* FPG $S^B = (V_S, canFail)$*, where* $canFail(v) = \bigvee_{v' \in \overline{x} \cap V_S} v'$ *if* $T_{\text{next}}(v) = f(\overline{x})$ *and* $canFail(v) = atLeast_{\lceil k/2 \rceil}(\overline{x} \cap V_S)$ *if* $T_{\text{next}}(v) = voter_k(\overline{x})$*, using the definition* $atLeast_m(X) = \bigvee_{\substack{Y \subseteq X \\ |Y|=m}} \bigwedge_{y \in Y} y$.[2]

[2] Note that there are more efficient and compact encodings for the *atLeast* constraint [18]; we use the most simple one for presentation purposes.

Example 4. Consider the transition system S from Example 3. The corresponding fault propagation graph is $S^B = (\{x_1^1, x_1^2, x_1^3, x_1^v, x_2^1, x_2^2, x_2^3, x_2^v, x_3^1\}, \mathit{canFail})$, where

$$\mathit{canFail}(x_1^i) = x_2^v \text{ for all } 1 \le i \le 3, \qquad \mathit{canFail}(x_2^i) = x_1^v \text{ for all } 1 \le i \le 3,$$
$$\mathit{canFail}(x_3^1) = x_1^v \vee x_2^v,$$
$$\mathit{canFail}(x_1^v) = \mathit{atLeast}_2(x_1^1, x_1^2, x_1^3), \qquad \mathit{canFail}(x_2^v) = \mathit{atLeast}_2(x_2^1, x_2^2, x_2^3).$$

4.2 Correctness

We show that the reduction preserves the cut sets. In the rest of the section, let $S = (V_S, V_{\text{in}}, V_{\text{init}}, T_{\text{next}}, T_{\text{init}})$ be an arbitrary redundancy UF+V TS, $\mathsf{Faults} \subseteq V_S$ be an arbitrary fault set, and $V_{\text{out}} \subseteq V_S$ be an arbitrary set of output signals. First, we show that each cut set of S corresponds to a cut set of S^B.

Lemma 1. *If* Faults *is a cut set of* S *for the set of outputs* V_{out}*, then* cs *defined as* $cs(v) = \top$ *iff* $v \in \mathsf{Faults}$ *is a cut set of* S^B *for the top level event* $\bigvee_{o \in V_{\text{out}}} o$*.*

Proof. Let Faults be a cut set of S for some trace π for some ι, Init, and $[\![\_]\!]$. Let π^{nom} be the corresponding nominal trace. Define the trace π^B of S^B as $\pi_0^B = cs$ and for all $i > 0$ define π_i^B by $\pi_i^B(v) = \top$ if $\pi_{i-1}^B(v) = \top$ and $\pi_i^B(v) = [\![\mathit{canFail}(v)]\!]_{\pi_{i-1}^B}$ if $\pi_{i-1}^B(v) = \bot$. In other words, π^B is the unique trace starting in cs in which all the components fail as soon as possible. By monotonicity, the trace π^B has a fixed point, i.e., there is n such that $\pi_n^B = \pi_{n'}^B$ for all $n' > n$.

We show that π^B satisfies $\pi_n^B(o) = \top$ for some $o \in V_{\text{out}}$ and thus cs is a cut set for the top level event $\bigvee_{o \in V_{\text{out}}} o$. To do this, we prove by induction on i and on the voter depth $vd(v)$[3] that for all $v \in V_S$ and $i \ge 0$, $\pi_i(v) \neq \pi_i^{nom}(v)$ implies $\pi_n^B(v) = \top$. We distinguish three cases:

- If $v \in \mathsf{Faults}$, then $\pi_0^B(v) = \top$. From the definition of π^B, this implies that $\pi_l^B(v) = \top$ for all $l \ge 0$. In particular, $\pi_n^B(v) = \top$.
- If $v \notin \mathsf{Faults}$ and $T_{\text{next}}(v) = f(x_1, \ldots, x_k)$, we distinguish two cases:
  - If $i = 0$: since $\pi_0(v) \neq \pi_0^{nom}(v)$, then it must be the case that $\pi_0(v) \neq \text{Init}(T_{\text{init}}(v))$, therefore $v \in \mathsf{Faults}$. This is a contradiction.
  - If $i > 0$: then $\pi_i(v) \neq \pi_i^{nom}(v)$ by definition implies

$$[\![f]\!](\pi_{i-1}(x_1), \ldots, \pi_{i-1}(x_k)) \neq [\![f]\!](\pi_{i-1}^{nom}(x_1), \ldots, \pi_{i-1}^{nom}(x_k))$$

    and hence $\pi_{i-1}(x_j) \neq \pi_{i-1}^{nom}(x_j)$ for some $1 \le j \le k$ because $[\![f]\!]$ is a function. Since $\pi_{i-1}(in) = \pi_{i-1}^{nom}(in)$ holds for all $in \in V_{\text{in}}$, we know that $x_j \in V_S$. Therefore the induction hypothesis implies $\pi_n^B(x_j) = \top$ and thus $\pi_{n+1}^B(v) = \top$ because π_n^B satisfies $\mathit{canFail}(v)$. Since π_n^B was chosen as the fixed point of π^B, this implies $\pi_n^B(v) = \pi_{n+1}^B(v) = \top$.

[3] Induction on the voter depth is employed because UF+V transition systems propagate results of voters instantaneously.

– If $v \notin \mathsf{Faults}$ and $T_{\text{next}}(v) = \mathit{voter}_k(x_1, \ldots, x_k)$, then $\pi_i(v) \neq \pi_i^{nom}(v)$ for any $i \geq 0$ by definition implies

$$\mathit{majority}(\pi_i(x_1), \ldots, \pi_i(x_k)) \neq \mathit{majority}(\pi_i^{nom}(x_1), \ldots, \pi_i^{nom}(x_k)). \quad (1)$$

Since S is a redundancy TS, all $\pi_i^{nom}(x_j)$ are equal and the disequality (1) implies that $\pi_i(x_j) \neq \pi_i^{nom}(x_j)$ for at least $\lceil k/2 \rceil$ of x_j. All these x_j are not in V_{in} and must therefore be in V_S. By definition of voter depth, $vd(x_j) < vd(v)$ for all these x_j. Therefore by the induction hypothesis $\pi_n^B(x_j) = \top$ for at least $\lceil k/2 \rceil$ of x_j and thus $\pi_{n+1}^B(v) = \top$ because π_n^B satisfies $\mathit{canFail}(v)$. This again implies $\pi_n^B(v) = \pi_{n+1}^B(v) = \top$ because π_n^B is the fixed point of π^B.

This finishes the proof: if Faults is a cut set, $\pi_c(o) \neq \pi_c^{nom}(o)$ for some $c \geq 0$ and $o \in V_{\text{out}}$, and thus $\pi_n^B(o) = \top$. Therefore we know that $\pi_n^B \models \bigvee_{o \in V_{\text{out}}} o$ and thus cs is a cut set of S^B. □

For the converse direction, we for each fault set devise a trace of the UF+V TS S that propagates all the possible deviations from the nominal value. We call this trace *maximally fault-propagating*. In this trace, all signal values are from the set $\{0, 1\}$, all nominal signal values are 0 and become 1 only as a result of a fault. Moreover, if there is a trace for the given fault set in which a signal deviates from its nominal value, the value of the corresponding signal in the maximally fault-propagating will be 1.

Definition 10 (Maximally fault-propagating trace). *Let S be a UF+V TS. Define*

– $\iota_i(v_{in}) = 0$ *for all* $i \geq 0$, $v_{in} \in V_{\text{in}}$, *i.e., ι is a stream of constant zero inputs;*
– $\text{Init}(v_{init}) = 0$ *for each* $v_{init} \in V_{\text{init}}$*; and*
– $[\![f]\!](x_1, \ldots, x_k) = 1 - \prod_{1 \leq i \leq k}(1 - x_i)$ *for each uninterpreted f, i.e., the output is 0 if all inputs are 0; it is 1 if at least one input is 1.*

The maximally fault-propagating trace of S *for a fault set* Faults*, denoted as π^{fp}, is the unique trace of S for the above input stream, initial values, interpretation, and the given fault set that for all $i \geq 0$ and v satisfies $\pi_i^{fp}(v) = 1$ whenever $v \in \mathsf{Faults}$.*

Observe that the trace π^{fp} is *monotone*, i.e., once a signal gets set to 1, it stays set to 1 for the rest of the trace. This is formalized by the following lemma, which can be proven by induction on i, $j - i$, and voter depth of v.

Lemma 2. *Let S be a UF+V TS,* Faults *a fault set, and π^{fp} the corresponding maximally fault-propagating trace. Then $\pi_i^{fp}(v) = 1$ for each $i \geq 0$ and $v \in V_S$ implies $\pi_j^{fp}(v) = 1$ for all $j > i$.*

We can now show that if a trace of the FPG version S^B of a UF+V TS S triggers the top level event for some initial fault assignment, there is a trace in the original system S for the corresponding fault set whose output deviates from the nominal one; namely it is the trace π^{fp}.

Lemma 3. *If cs defined as* $cs(v) = \top$ *iff* $v \in \mathsf{Faults}$ *is a cut set of* S^B *for the top level event* $\bigvee_{o \in V_{\text{out}}} o$*, then* Faults *is a cut set of* S *for the set of outputs* V_{out}*.*

Proof. Suppose that the trace π^B of S^B with the initial state cs satisfies $\pi^B_c(o) = \top$ for some $c \geq 0$ and $o \in V_{\text{out}}$. We show that Faults is a cut set of S for the set of output signals V_{out}. Let π^{fp} be the maximally fault-propagating trace of S for Faults and π^{nom} the corresponding nominal trace.

We show that for each $i \geq 0$ and $v \in V_S$, the condition $\pi^B_i(v) = \top$ implies $\pi^{fp}_i(v) \neq \pi^{nom}_i(v)$. We proceed by induction on i:

- For $i = 0$: If $cs = \pi^B_0(v) = \top$, then $v \in \mathsf{Faults}$ and thus $\pi^{fp}_0(v) \neq \pi^{nom}_0(v)$ because $\pi^{fp}_0(v) = 1$ and $\pi^{nom}_0(v) = 0$.
- For $i > 0$: Assume that $\pi^B_i(v) = \top$. We distinguish four cases:
  - If $v \in \mathsf{Faults}$ then $\pi^{fp}_i(v) = 1$ and so $\pi^{fp}_i(v) \neq \pi^{nom}_i(v) = 0$.
  - If $\pi^B_{i-1}(v) = \top$, then we get that $\pi^{fp}_{i-1}(v) \neq \pi^{nom}_{i-1}(v)$ from the induction hypothesis, and thus $\pi^{fp}_i(v) \neq \pi^{nom}_i(v)$ by Lemma 2.
  - If $v \notin \mathsf{Faults}$, $\pi^B_{i-1}(v) = \bot$, and $T_{\text{next}}(v) = f(x_1, \ldots, x_k)$, then $\pi^B_i(v) = \top$ implies that $\pi^B_{i-1}(x_j) = \top$ for at least one $x_j \in V_S$. From the induction hypothesis, we get that $\pi^{fp}_{i-1}(x_j) \neq \pi^{nom}_{i-1}(x_j)$ and since $\pi^{nom}_{i-1}(x_j) = 0$, we know that $\pi^{fp}_{i-1}(x_j) = 1$. By the definition of $[\![f]\!]$ in π^{fp}_i, we know that also $\pi^{fp}_i(v) = 1$, which is not equal to $\pi^{nom}_i(v) = 0$.
  - If $v \notin \mathsf{Faults}$, $\pi^B_{i-1}(v) = \bot$, and $T_{\text{next}}(v) = voter_k(x_1, \ldots, x_k)$, then $\pi^B_i(v) = \top$ implies that at least $\lceil k/2 \rceil$ of $x_j \in V_S$ satisfy $\pi^B_{i-1}(x_j) = \top$. From the induction hypothesis we get that $\pi^{fp}_{i-1}(x_j) \neq \pi^{nom}_{i-1}(x_j)$ for these x_j and since $\pi^{nom}_{i-1}(x_j) = 0$, we know that $\pi^{fp}_{i-1}(x_j) = 1$ for at least $\lceil k/2 \rceil$ of x_j. By the definition of majority function, we know that also $\pi^{fp}_{i-1}(v) = 1$ and thus, by Lemma 2, also $\pi^{fp}_i(v) = 1 \neq 0 = \pi^{nom}_i(v)$.

Therefore $\pi^B_c(o) = \top$ implies $\pi^{fp}_c(o) \neq \pi^{nom}_c(o)$ and Faults is a cut set of S. □

Theorem 1. *For each fault set* Faults*, the following two claims are equivalent:*

1. *The set* Faults *is a cut set of* S *for the set of output signals* V_{out}*.*
2. *The assignment cs defined as* $cs(v) = \top$ *iff* $v \in \mathsf{Faults}$ *is a cut set of* S^B *for the top level event* $\bigvee_{o \in V_{\text{out}}} o$*.*

5 Related Work

Approaches to the analysis of redundant architectures include [6], which addresses the generation of the reliability function for a class of generic architectures including tree- and DAG-like structures. The computation of the reliability is based on predicate abstraction and BDDs. Our work extends and improves the approach of [6] in several directions. First, it supports cyclic architectures, to which predicate abstraction as defined in [6] cannot be applied. Second, it does not require that the redundancy is localized within small blocks (manually

defined by the user or in a library), to which the predicate abstraction can be applied. In contrast, our approach applies the abstraction directly on the level of individual modules and voters. Moreover, the approach of [6] needs to compute the abstracted versions of the specified blocks upfront by quantifier elimination. Finally, our approach outperforms the approach of [6].

Other works on redundant architecture analysis are either based on ad-hoc algorithms [13] which are not fully automated, and require discretization and additional input data from the user, or use simulation techniques such as Monte Carlo analysis [15], which do not examine the system behaviors exhaustively.

A classification of fault tolerant architectures is presented in [10]. The classification is based on three different patterns, namely comparison, voting, and sparing, that can be composed to define generic and possibly cyclic architectures. A follow-up work [11] builds upon these patterns and introduces strategies to evaluate several architectures at once (family-based analysis of redundant architectures) by reduction to Discrete Time Markov Chains. Our techniques are orthogonal, and could be applied on top of the approach proposed in [11].

The concept of *maximally fault-propagating trace* used to prove Lemma 3 is similar to the concept of *maximally diverse interpretations* [8], which can be used to efficiently reduce a formula in the positive fragment of EUF logic to a SAT formula. Both concepts restrict the interpretations of uninterpreted functions to a specific subclass, which exhibits all the relevant behaviors.

6 Experimental Evaluation

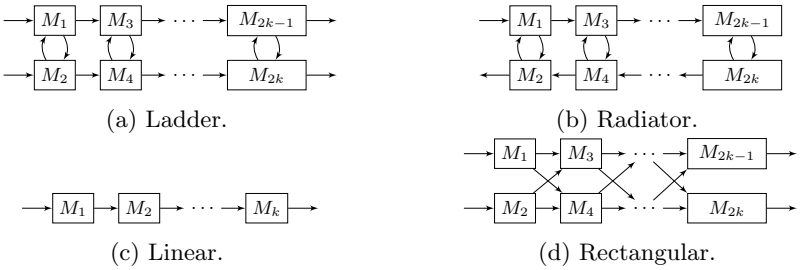
We have performed an experimental evaluation of the proposed approach for minimal cut set enumeration in order to answer the following research questions:

RQ1 How does the new approach scale on redundancy architectures *with cycles*?
RQ2 On redundancy architectures *with cycles*, how do the run-times compare against the approach based on the enumeration of minimal cut sets of the miter system by a model checker?
RQ3 On redundancy architectures *without cycles*, how do the run-times compare against the approach based on predicate abstraction (PA) and BDD-based enumeration [6]?
RQ4 On redundancy architectures *without cycles*, what part of the runtime difference is caused by the different reduction to a Boolean problem (FPG vs PA) and what part is caused by a different solving approach of the resulting Boolean problem (SAT-based vs BDD-based)?

6.1 Benchmarks and Setup

To answer these research questions, we used four sets of redundancy systems:

Scalable cyclic systems This benchmark set contains two kinds of benchmarks. For evaluation on redundancy architectures with a linear number



(a) Ladder. (b) Radiator.

(c) Linear. (d) Rectangular.

Fig. 3: Scalable architectures used in the experimental evaluation.

of cycles, we have generated ladder-shaped (Figure 3a) architectures of all lengths between 1 and 100. For evaluation on redundancy architectures with a large number of cycles, we have generated radiator-shaped (Figure 3b) architectures of all lengths between 1 and 50. For each of the architectures, we have generated its three redundant versions by replacing each module by a TMR block with one to three voters by using schemas from Figures 2b, 2d, and 2e. This yields systems with $2 \cdot \mathit{length} \cdot (3 + \mathit{numVoters})$ signals.

Random cyclic systems We have generated 250 random cyclic redundancy UF+V systems with 1 to 150 modules of arity between 1 and 3, randomly generated 1 to 6 replicas of each module, and 1 to 6 voters of arity 3 or 5, randomly connected to the replicas.

Scalable acyclic systems This benchmark set contains linear-shaped (Figure 3c) and rectangular-shaped (Figure 3d) architectures of all lengths between 1 and 200 that were used for evaluation of predicate abstraction technique [6]. As in the original paper, we have used redundant versions of the systems with the modules replaced by a TMR block with one to three voters.

Random acyclic systems We have used randomly generated acyclic architectures composed of randomly chosen TMR blocks that were also used in [6].

We have evaluated the following approaches for minimal cut set enumeration:

- For the systems *with cycles*, we have generated their FPG version as described in Section 4 and also the UFLRA transition system implementing the miter construction in the SMV format, For enumeration of the minimal cut sets of the fault propagation graphs, we have used the tool SMT-PGFDS [7] (denoted as FPG in the experiments); for enumeration of the minimal cut sets of miter systems, we have used the tool xSAP [2], which internally uses an algorithm based on parametric IC3 [3] (denoted as ParamIC3).
- For the systems *without cycles*, we have generated both their FPG version and the description in the format of the tool OCRA [9] as used in [6]. Although the FPGs could be solved by the tool SMT-PGFDS and the OCRA systems can be solved by predicate abstraction, which is implemented in xSAP, and its BDD-based engine [6], this would not compare only the effect of the reduction to the Boolean case, but also a confounding factor of the underlying

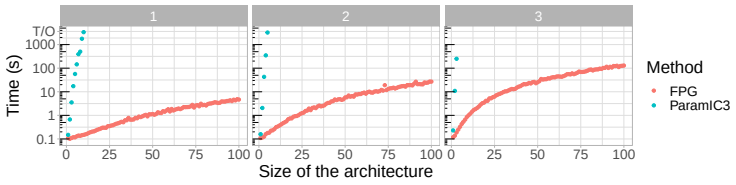


Fig. 4: Solving time on ladder-shaped benchmarks. Divided according to the number of voters per one reference module.

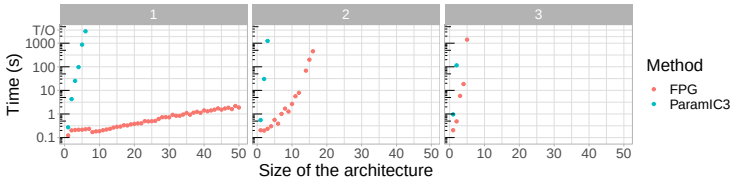


Fig. 5: Solving time on radiator-shaped benchmarks. Divided according to the number of voters per one reference module.

backend (SAT-based in SMT-PGFDS and BDD-based in xSAP). To answer RQ4, we have thus performed more fine-grained analysis as follows.
From each FPG, we generated the corresponding Boolean formula, which is possible since the graph is acyclic [7]. We also generated the Boolean formula obtained by predicate abstraction from each OCRA encoding. We thus obtained two Boolean formulas for each system: one by reduction to fault propagation (FP), and one by reduction by predicate abstraction (PA). We have then used the SAT-based enumeration algorithm of SMT-PGFDS and also BDD-based enumeration algorithm of xSAP on both of these Boolean formulas. This gives 4 combinations: FP-SAT, FP-BDD, PA-SAT, PA-BDD.

All experiments were executed on a cluster of 9 computational nodes, each with Intel Xeon CPU X5650 @ 2.67GHz, 12 CPU and 96 GiB of RAM. We have used time limit 1 hour of wall-clock time and memory limit 16 GB for each benchmark-solver pair. The detailed experimental results can be found at https://es-static.fbk.eu/people/mjonas/papers/tacas22_redarchs/.

6.2 Results for Cyclic Benchmarks

The comparison of running times of FPG-based and of model-checking-based approaches on the scalable cyclic benchmarks is shown in Figures 4 and 5. Figure 4 shows a significant benefit of the technique based on fault propagation on the ladder-shaped benchmarks; not only that it can enumerate cut sets of all the used benchmarks, but its run-times are dramatically better. However, as can be seen on Figure 5, the situation is different on the radiator-shaped benchmarks, which contain a large number of cycles. Although the performance of technique based

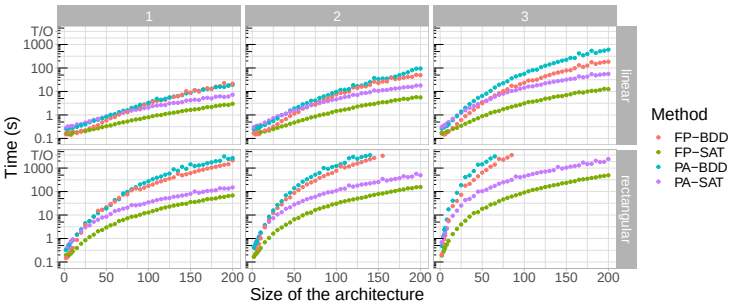


Fig. 7: Solving time on scalable acyclic benchmarks. Divided by the architecture and number of voters per one reference module.

on fault propagation is still superior to the model-checking-based technique, it scales poorly on the systems with 2 and 3 voters per one TMR block. The answer to RQ1 is thus that the proposed approach scales well if the number of cycles in the system is not too large; if the number of cycles is large, the technique scales worse, but nevertheless significantly better than the state-of-the-art technique based on miter construction and model checking [3].

The run-times on random cyclic benchmarks are shown in Figure 6. The figure shows that the performance of the proposed technique is better by several orders of magnitude and can enumerate minimal cut sets of 59 random systems that are out of reach for the technique based on model checking. Note that some of the systems are hard for both of the approaches: both approaches timed out on 66 of the 250 benchmarks. Together with the results for the ladder-shaped and radiator-shaped systems, this answers RQ2: the technique proposed in this paper has significantly better performance than the state-of-the-art technique based on model checking.

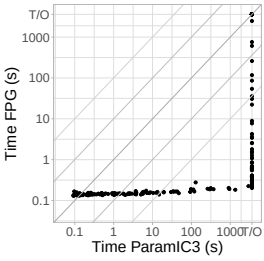


Fig. 6: Solving time on random cyclic benchmarks.

There are two reasons of the observed performance difference. First is the reduction of UFLRA transition system to the Boolean one, which has been also observed to bring significant benefit on acyclic systems in the case of predicate abstraction [6]. Second is the underlying MCS-enumeration technique applied the resulting FPG. This technique reduces the expensive sequential reasoning to an enumeration of minimal models of a single SMT formula, which can significantly improve performance [7].

6.3 Results for Acyclic Benchmarks

The comparison of the performance on acyclic scalable benchmarks is shown in Figure 7. The results are divided according to the method used to reduce the

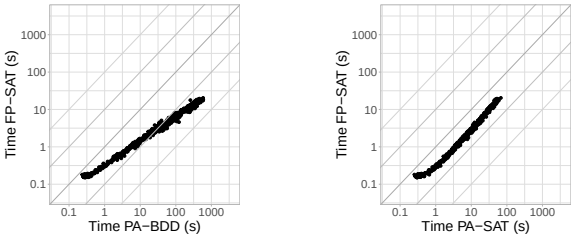


Fig. 8: Solving time on random acyclic benchmarks.

problem to Boolean case (FP vs. PA) and the technique used to enumerate the minimal cut sets of the Boolean system (SAT vs. BDD). Scatter plots of solving times on random acyclic benchmarks can be seen on Figure 8.

The results show that the reduction of the problem to fault propagation and using an off-the-shelf solver for enumeration of minimal cut sets of the resulting Boolean system (i.e., FP-SAT) is clearly superior to the state-of-the-art approach based on predicate abstraction and BDD-based MCS enumeration (i.e., PA-BDD). The difference between these two approaches is even several orders of magnitude on scalable benchmarks and grows with the size of the system and its complexity. The performance is also significantly better on the random benchmarks. This answers RQ3 in favor of the technique proposed in this paper.

As for RQ4, Figures 7 and 8 show that both the different reduction technique (FP vs. PA) and the solving technique (SAT vs. BDD) play a role in this difference. However, the larger part of the runtime difference between the proposed approach (FP-SAT) and the state-of-the-art approach (PA-BDD) [6] is due to better performance of SAT-based enumeration. This insight is additional interesting outcome of our our experiments. Nevertheless, for both of the enumeration approaches, the proposed reduction based on fault propagation provides better performance than the state-of-the-art reduction by predicate abstraction.

7 Conclusions and Future Work

We have presented a framework for modeling redundancy architectures with possible cyclic dependencies among the computational modules and we have developed an efficient approach for enumeration of minimal cut sets of such architectures. The experimental evaluation has shown that this approach dramatically outperforms the state-of-the-art approach based on model checking on cyclic redundancy architectures and has a better performance than the state-of-the-art approach based on predicate abstraction on acyclic architectures.

In the future, we plan to extend the approach to a more general class of voters than majority voters. We also plan to extend the approach to support common cause analysis for different component faults and possibly to synthesize an optimal distribution of the modules of the architecture between the computational nodes of a system such as Integrated Modular Avionics.

References

1. Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009.
2. Benjamin Bittner, Marco Bozzano, Roberto Cavada, Alessandro Cimatti, Marco Gario, Alberto Griggio, Cristian Mattarei, Andrea Micheli, and Gianni Zampedri. The xSAP safety analysis platform. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9636 of *Lecture Notes in Computer Science*, pages 533–539. Springer, 2016.
3. Marco Bozzano, Alessandro Cimatti, Alberto Griggio, and Cristian Mattarei. Efficient anytime techniques for model-based safety analysis. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 603–621. Springer, 2015.
4. Marco Bozzano, Alessandro Cimatti, and Cristian Mattarei. Automated analysis of reliability architectures. In *2013 18th International Conference on Engineering of Complex Computer Systems, Singapore, July 17-19, 2013*, pages 198–207. IEEE Computer Society, 2013.
5. Marco Bozzano, Alessandro Cimatti, and Cristian Mattarei. Efficient analysis of reliability architectures via predicate abstraction. In Valeria Bertacco and Axel Legay, editors, *Hardware and Software: Verification and Testing - 9th International Haifa Verification Conference, HVC 2013, Haifa, Israel, November 5-7, 2013, Proceedings*, volume 8244 of *Lecture Notes in Computer Science*, pages 279–294. Springer, 2013.
6. Marco Bozzano, Alessandro Cimatti, and Cristian Mattarei. Formal reliability analysis of redundancy architectures. *Formal Aspects Comput.*, 31(1):59–94, 2019.
7. Marco Bozzano, Alessandro Cimatti, Anthony Fernandes Pires, Alberto Griggio, Martin Jonáš, and Greg Kimberly. Efficient SMT-Based Analysis of Failure Propagation. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II*, volume 12760 of *Lecture Notes in Computer Science*, pages 209–230. Springer, 2021.
8. Randal E. Bryant, Steven M. German, and Miroslav N. Velev. Exploiting positive equality in a logic of equality with uninterpreted functions. In Nicolas Halbwachs and Doron A. Peled, editors, *Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings*, volume 1633 of *Lecture Notes in Computer Science*, pages 470–482. Springer, 1999.
9. Alessandro Cimatti, Michele Dorigatti, and Stefano Tonetta. OCRA: A tool for checking the refinement of temporal contracts. In Ewen Denney, Tevfik Bultan, and Andreas Zeller, editors, *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*, pages 702–705. IEEE, 2013.
10. Kai Ding, Andrey Morozov, and Klaus Janschek. Classification of hierarchical fault-tolerant design patterns. In *15th IEEE Intl Conf on Dependable, Autonomic*

and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress, DASC/PiCom/DataCom/CyberSciTech 2017, Orlando, FL, USA, November 6-10, 2017, pages 612–619. IEEE Computer Society, 2017.
11. Clemens Dubslaff, Kai Ding, Andrey Morozov, Christel Baier, and Klaus Janschek. Breaking the limits of redundancy systems analysis. *CoRR*, abs/1912.05364, 2019.
12. Walter Haeussermann. Description and performance of the saturn launch vehicle's navigation, guidance, and control system. *IFAC Proceedings Volumes*, 3(1):275–312, 1970. 3rd International IFAC Conference on Automatic Control in Space, Toulouse, France, March 2-6, 1970.
13. Masashi Hamamatsu, Tatsuhiro Tsuchiya, and Tohru Kikuno. On the Reliability of Cascaded TMR Systems. In Yutaka Ishikawa, Dong Tang, and Hiroshi Nakamura, editors, *16th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2010, Tokyo, Japan, December 13-15, 2010*, pages 184–190. IEEE Computer Society, 2010.
14. Shuvendu K. Lahiri, Robert Nieuwenhuis, and Albert Oliveras. SMT techniques for fast predicate abstraction. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 424–437. Springer, 2006.
15. Sungjae Lee, Jae-il Jung, and Inhwan Lee. Voting structures for cascaded triple modular redundant modules. *IEICE Electronic Express*, 4(21):657–664, 2007.
16. Paul J Prisaznuk. Integrated modular avionics. In *Proceedings of the IEEE 1992 National Aerospace and Electronics Conference@ m_NAECON 1992*, pages 39–45. IEEE, 1992.
17. Enno Ruijters and Mariëlle Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Comput. Sci. Rev.*, 15:29–62, 2015.
18. Ed Wynn. A comparison of encodings for cardinality constraints in a SAT solver. *CoRR*, abs/1810.12975, 2018.
19. Y.C. Yeh. Triple-triple redundant 777 primary flight computer. In *1996 IEEE Aerospace Applications Conference. Proceedings*, volume 1, pages 293–307 vol.1, 1996.