IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 25, 2020, accepted October 17, 2020, date of publication October 29, 2020,
date of current version November 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3034799

Symmetry Breaking in Model Checking of Fault-Tolerant Nuclear Instrumentation and Control Systems

IGOR BUZHINSKY[1,2] AND ANTTI PAKONEN[3]
[1]Department of Electrical Engineering and Automation, Aalto University, 02150 Espoo, Finland
[2]Computer Technologies Laboratory, ITMO University, 197101 Saint Petersburg, Russia
[3]VTT Technical Research Centre of Finland Ltd., 02044 Espoo, Finland

Corresponding author: Igor Buzhinsky (igor.buzhinsky@gmail.com)

This work was supported in part by the Finnish Research Programme on Nuclear Power Plant Safety 2019–2022 (SAFIR2022), and in part by the Government of the Russian Federation under Grant 08-08.

ABSTRACT One of the approaches to assure reliability of nuclear instrumentation and control (I&C) systems is model checking, a formal verification technique. Model checking is computationally demanding, but nuclear I&C systems have certain properties that simplify the verification problem. The most notable of these properties are redundancy (duplication of certain system parts in several divisions) and symmetry, which are the means of ensuring failure tolerance. In this work, we extend our previous method of model checking failure tolerance of nuclear I&C systems by proposing an automated symmetry breaking approach that utilizes these properties to simplify the verification problem. As a result, fewer failure combinations need to be checked. We evaluate this approach on a case study that encompasses three safety functions allocated to four I&C systems in the same I&C model.

INDEX TERMS Formal verification, model checking, symmetry breaking, nuclear I&C systems, fault tolerance.

I. INTRODUCTION

Instrumentation and control (I&C) systems of nuclear power plants (NPPs) must be ensured to be correct. This is achieved with approaches that encompass both architectural choices, such as following the defense-in-depth (DiD) [1] principles, and functional verification. In Finland, the latter is performed formally [2]–[5] with the model checking [6], [7] technique.

One of the obstacles of applying model checking in industrial practice is computational complexity. This problem received algorithmic solutions, including symbolic model checking [8], bounded model checking [9], and the IC3 [10] algorithm. However, handling large industrial systems is still a challenge. A complementary approach to reduce computational complexity is utilization of domain-specific knowledge. In this article, we follow this approach to verify fault-tolerant nuclear I&C systems, harnessing their redundancies (the similarity of the intra-system divisions) and symmetries (patterns in inter-system connections).

This work continues our previous work [11]. With respect to [11], its contributions are: (1) we propose a symmetry breaking approach for model checking of nuclear I&C systems, which automates the reasoning based on which certain failure combinations can be omitted from model checking, (2) we improve our failure injection technique to widen the class of formal specifications to which it is applicable, and (3) we enlarge our case study. Specifically, selection of verification configurations to be verified is done by logically proving that during verification certain configurations provide guarantees at least as strong as others.

The remainder of the paper is organized as follows. In Section II, general information about nuclear I&C systems is given. Then, Section III explains how these systems can be formally verified with model checking. In Section IV, our symmetry breaking approach is presented. In Section V, this approach is evaluated on a case study based on a fictitious NPP. Related work is reviewed in Section VI. The paper is concluded in Section VII.

II. NUCLEAR I&C SYSTEMS

The functionality of nuclear I&C systems is usually specified using function block diagrams (FBDs), and the logic is often distributed across multiple processing units in multiple buildings. Formal verification of such systems was previously considered in [2]–[5], [12]–[15]. Among the functions that are most important in terms of formal verification are

The associate editor coordinating the review of this manuscript and approving it for publication was Junjian Qi.

safety functions. Due to the need to assure failure tolerance, they are designed in a redundant, often symmetric way, where identical processing units are placed in different buildings of the NPP. Thus, their verification needs to account both for software (the FBDs) and hardware (failures and communication) [11]. Failure tolerance can also be improved through diversity, i.e., the use of different technology or design principle in a redundant system.

A. DEFENCE-IN-DEPTH

According to the Defence-in-Depth (DiD) principle [1], "a nuclear power plant shall be designed using multiple, successive redundant structures and systems in order to prevent reactor damage and the detrimental effects of radiation." In practice, this requires successive levels of protection (called DiD levels) to be as independent of each other as possible. The I&C systems of the plant must also fulfill the DiD principle. In theory, the I&C architecture could be designed to achieve total independence between the DiD layers. Such a solution, however, is impractical [16]. An optimized architecture avoids these problems using justifiable compromises in the separation of DiD layers. In Section V, our case study will include the following DiD levels (defined according to European guidelines [17]):

- Level 1: prevention of abnormal operation and failures.
- Level 2: control of abnormal operation and failures.
- Level 3: control of accident to limit radiological releases and prevent escalation to core melt conditions.

B. HARDWARE FAILURES

According to the Finnish regulatory guides for nuclear safety [1], the following failure types are defined:

- The *single failure criterion* means that a safety function must be possible to perform even if any single component designed for the function fails.
- A *common cause failure* (CCF) refers to multiple failures across redundant subsystems: a failure of two or more structures, systems and components due to the same single event or cause. Such a failure can manifest as the loss of all of the sensors, computers, communication pathways, and/or actuators used by an I&C system.
- A *consequential failure* refers to a failure caused by a failure of another system, component or structure or by an internal or external event at the facility. An example of such a failure is the simultaneous loss of several I&C systems due to the failure of a shared power supply.

The most safety critical digital I&C systems are those used for reactor protection on DiD level 3. In Finland, such systems belong to safety class (SC) 2, and must fulfill the single failure criterion.[1] At the same time, they also must withstand the CCF of the systems on the lower DiD levels.

Hardware failures in nuclear I&C model checking were previously considered in [18], with detailed failure modes, and in our previous works [11], [19], where the single failure criterion was applied with failures that substitute signal values in failing divisions with nondeterministic values. We follow the latter approach in this article.

III. MODEL CHECKING OF NUCLEAR I&C SYSTEMS

A. FORMAL MODELS

Formal models that we consider represent possible behaviors of the modeled system as sequences of its states (i.e., under the discrete, transition-based model of time). Our formalization is close to the one that is suitable for NuSMV [20] models but is enriched with domain-specific information related to failure tolerance assurance and checking in nuclear I&C systems.

A *module* $(V^{\text{in}}, V^{\text{out}}, V^{\text{int}}, S^{\text{in}}, S^{\text{own}}, S_0, T, \Sigma)$ consists of:

1) Sequences $V^{\text{in}} = (v_1^{\text{in}}, \ldots, v_m^{\text{in}})$ of *input variables*, $V^{\text{out}} = (v_1^{\text{out}}, \ldots, v_n^{\text{out}})$ of *output variables* and $V^{\text{int}} = (v_1^{\text{int}}, \ldots, v_k^{\text{int}})$ of *internal variables* with their sets of possible values $(R_1^{\text{in}}, \ldots, R_m^{\text{in}})$, $(R_1^{\text{out}}, \ldots, R_n^{\text{out}})$ and $(R_1^{\text{int}}, \ldots, R_k^{\text{int}})$ respectively. The value of each variable consists of two parts: the *primary value*, which is either a Boolean or an integer from some finite set, and a binary *fault status* (i.e., a Boolean variable will effectively have four possible values).[2]
2) The sets of *input states* $S^{\text{in}} = R_1^{\text{in}} \times \ldots \times R_m^{\text{in}}$ and *own states* $S^{\text{own}} = R_1^{\text{out}} \times \ldots \times R_n^{\text{out}} \times R_1^{\text{int}} \times \ldots \times R_k^{\text{int}}$.
3) The *initial state relation* $S_0 \subseteq S^{\text{in}} \times S^{\text{own}}$, which specifies the initial states of own variables that are possible given the initial values of input variables. We assume that input variables are not determined by the module, but their values are provided from outside.
4) The *transition relation* $T \subseteq (S^{\text{in}} \times S^{\text{own}}) \times S^{\text{own}}$, which specifies how own states can change in time, given the values of input variables. We assume that T is left-total, i.e., each state always has at least one successor.
5) The set of *input symmetries* $\Sigma \subseteq 2^{V^{\text{in}}}$, where a symmetry $s \in \Sigma$ is a subset of input variables such that any permutation of the values of these variables (selected for the entire module execution) has no effect on the values of output variables. We require all $s \in \Sigma$ to be disjoint. Symmetries must be provided by the user but can be verified automatically [11].

A *formal model* is composed of a sequence $\hat{V}_{\text{in}}$ of *system input variables* (with fault statuses; system input variables are also allowed to be constant, in which case their fault status is always false), modules $M_1, \ldots, M_r$, and a set C of connections between them of the form (p, q, M), where p is a system input variable or an output variable of some module, q is an input variable of a different module, and M is a *connection module*. In the simplest case, a connection module is an *identity module*, which makes the connected variables have the same values. We will also consider *failure modules* that can substitute the input value with an

[1] For SC2, there is an stricter requirement (called N+2) stating that the system has to tolerate a single failure in any component while any other component is simultaneously out of operation due to repair or maintenance.

[2] Fault statuses are typical in nuclear I&C systems and are used to reason about signal validity, e.g., during voting. [5]

arbitrary value within the allowed range of the corresponding variable.[3]

A formal model has the following execution semantics. All the modules execute synchronously. In the first step, the values of system input variables are chosen nondeterministically from their value sets, and the initial states of all the modules are chosen nondeterministically according to their initial state relations. In subsequent steps, system input variables are again chosen nondeterministically, and the modules proceed according to their transition relations.

Additionally, we assume that $M_1, \ldots, M_r$ are deterministic (i.e., their initial state and transition relations are functional) and are internally decomposed into *basic blocks*. This decomposition is similar to the one of the formal model into modules.

One more attribute of a formal model is a set G of *unit groups*, where each group $g \in G$ has a positive number $d(g)$ of *divisions*. Unit groups, in turn, are disjointly composed of *units* and *system inputs*:

1) Units are groups of identical modules, one per each division of the unit group. If $u \in g$ is a unit, we denote its modules as $M(u, i), 1 \leq i \leq d(g)$. These modules are identical, but we distinguish their input and output variables and allow them to be connected with other components of the formal model in a different way.
2) System inputs are groups of system input variables with identical sets of possible values, one per each division of the unit group. If $q \in g$ is a system input, we denote its input variables as $v(q, i), 1 \leq i \leq d(g)$.

Having all these definitions, we can define a formal model as a tuple $F = (\hat{V}_{\text{in}}, \{M_1, \ldots, M_r\}, C, G)$. By $B(F)$ we denote the set of *behavior traces* (or simply *behaviors*) of F, i.e., possible infinite sequences of states, where each state is an assignment of all variables in F. In the definition of $B(F)$, while considering states, we do not distinguish variables that only differ by the division of the unit they belong to: for example, if some Boolean variable w belongs to a unit with four divisions, we treat all states where w is true in exactly three divisions as equivalent. This will allow us to compare behavior sets of models derived from F by adding failures and/or removing certain components.

In Fig. 1, a fictitious formal model is shown that we use as a running example. It has a single unit group g with $d(g) = 2$ divisions and two units called u_{top} and u_{bottom}. Each of these units consists of $d(g) = 2$ identical modules, whose internal decomposition into basic blocks is shown inside the colored rectangles.

A *component* of a formal model is either a module, an output variable of a module, or a system input variable. We view the formal model as a directed graph $\Gamma = (V_\Gamma, E_\Gamma)$ whose vertices are all the components in the formal model and whose arcs are defined as follows: a module is connected to each

[3]Connection modules can also be used to inject communication delays, like in [11]. We do not consider them in this article, although they can be easily accounted for in our approach.

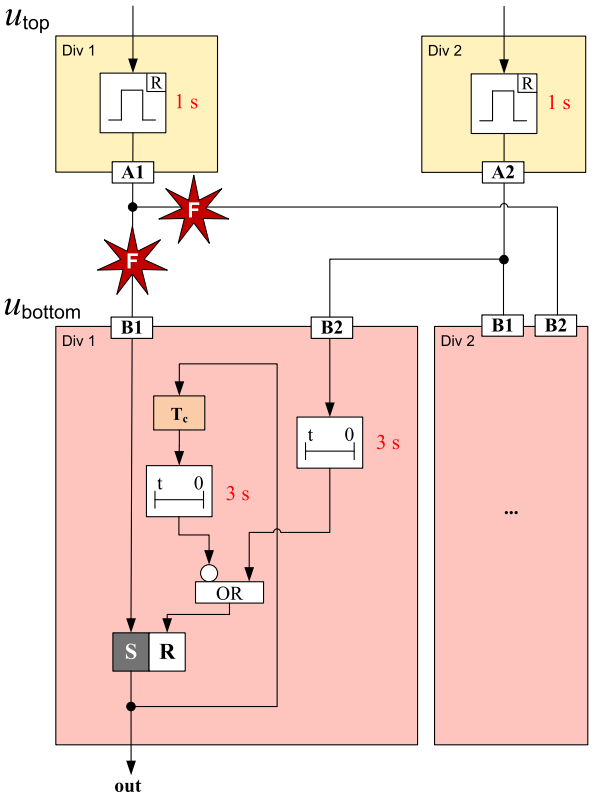


FIGURE 1. Running example of a formal I&C model. The top-most "pulse" elements always output a signal of specified length, starting on the rising edge of the input signal (unless the previous pulse is still active). The "on-delay" element (labeled "t..0") sets its output when its input has been active for over the specified length. The "flip-flop" (labeled S R) latch element is set and reset by the associated inputs, with priority on the set side.

of each output variables, and a beginning of each connection is connected to the module at the end of this connection. Our approach requires Γ to be acyclic (individual modules, however, are allowed to have feedback loops inside them). We say that a component $x \in V_\Gamma$ is *upstream* with respect to a component $y \in V_\Gamma$ if there exists a directed path from x to y in Γ. For example, in Fig. 1, for each $1 \leq i, j \leq 2$, $M(u_{\text{top}}, i)$ is upstream with respect to $M(u_{\text{bottom}}, j)$.

The unit group $g(x)$ of component x is defined as follows:

1) if x is a module or a system input variable, $g(x)$ is the unit group such that $x \in g(x)$;
2) if x is an output variable of a module of unit u, $g(x) = g(u)$.

The division $d(x)$ of component x is defined as follows:

1) if x is a module, $d(x)$ is such that $x = M(u, d(x))$ for some unit u;
2) if x is a system input variable, $d(x)$ is such that $x = v(q, d(x))$ for some system input q;
3) if x is an output variable of a module of unit u, $d(x) = d(u)$.

B. FAILURE MODELING

As in [11], we model failures by placing certain modules on connections inside the system. Such failures also cover

internal failures of computational devices where the modules are executed since these failures can be simulated by replacing their outputs with nondeterministic values (unless the internal contents of modules are queried in the formal specification).

If a unit group g must allow single failures, then a specific division i of g is chosen to be failing and failure modules are placed on all connections that either begin or end (or both) at this division of g. If CCFs are possible in g, then, in the worst case, all connections leading from g to other unit groups are affected. We model this case by placing failures in all divisions of g.

Having a fault-free model, we reason about possible failures that can be added to it with *failure assignments* $\phi : G \to 2^{\mathbb{N}}$. For each unit group g, ϕ gives the indices of failing divisions in g, and thus $\phi(g) \subseteq \{1, \dots, d(g)\}$. For convenience, we extend ϕ so that for a component $x \in V_\Gamma$ it indicates whether x is affected by failures: $\phi(x) \stackrel{\text{def}}{=} \mathbf{1}_{\phi(g(x))}(d(x))$.

By default, we implement failures as replacements of the failing signals with nondeterministic values, as if the system had additional input variables. In Section IV-E, we will show that this treatment of failures must be revised to correctly handle a certain subclass of CTL properties.

C. TEMPORAL LOGICS

Model checking needs formal languages to specify properties to be checked for formal models. Predicates over state variables are not sufficiently flexible since they cannot capture time. *Linear temporal logic* (LTL) is an extension of the Boolean propositional logic that captures time in a particular behavior trace of the formal model with *temporal operators*, such as **G** (''always'') and **F** (''in the future''). For example, if x is an integer state variable, then the formula $\mathbf{F}\,\mathbf{G}(x = 10)$ specifies that x eventually becomes 10 and retains this value forever. An LTL property is said to be satisfied for a formal model if it is satisfied for all its behaviors.

In *computation tree logic* (CTL), the values of temporal formulas are first defined for model states rather than behaviors, and a CTL formula is satisfied for a formal model if it is satisfied in all its initial states. CTL temporal operators are annotated with path quantifiers **A** and **E**, which specify that a property is satisfied for all or for some behaviors starting from the current state—thus, it becomes possible to express reachability.

In our work, CTL properties are limited to the ones of the form $\mathbf{AG}\,\mathbf{EF}\,f$, where f is a Boolean formula. We call them *global possibility properties*: according to this formula, from all reachable states of the model, it is possible to reach a state where f is satisfied. More specifically, if p is a Boolean variable, then checking $\mathbf{AG}\,\mathbf{EF}\,p$ and $\mathbf{AG}\,\mathbf{EF}\,\neg p$ ensures that both values of p are reachable in any reachable state of the model.

D. MODEL CHECKING TOOLS

To work with formal models and properties of the aforementioned classes, we use the following tools:

1) NuSMV [20] and nuXmv [21] model checkers. Formal NuSMV and nuXmv models are specified in their own textual language.
2) MODCHK [22], a graphical front-end to NuSMV. In this tool, modules and formal models can be created visually, from a library of basic blocks written in NuSMV.
3) HW-SW-builder [11], a tool to specify the modular structure of I&C models textually, based on the same basic blocks. HW-SW-builder generates NuSMV models that are similar to the ones produced by MODCHK, but unlike the latter, it supports failure and delay injection into the formal model and allows declaring and checking symmetries that exist in the I&C system. In this work, we further enhance this tool.

IV. PROPOSED APPROACH

A. MOTIVATING EXAMPLE

We return to the example shown in Fig. 1. The connections from division 1 of u_{top} are marked with ''F'' stars, which indicate possible failures. For now, suppose that these failures do not manifest themselves and we need to check an LTL property that specifies the behavior of u_{bottom}, e.g., $f = \mathbf{G}\,\mathbf{F}\,\texttt{out}$. One may notice that this is only sufficient to be done for one division of u_{bottom} since its two modules are identical and receive inputs from identical divisions of u_{top}. In [11], such observations were applied to reduce the number of scenarios to be verified, but reasoning was performed manually and involved larger systems. Can this symmetry breaking reasoning be automated?

When it comes to verifying failure tolerance, failures must also be encompassed in reasoning. If we assume that one of two divisions of the I&C system may have arbitrary failures (by placing failure modules on connections), then it only makes sense to model-check the requirement for the outputs of the other division (otherwise, these outputs would be directly affected by failures). For our example, the cases of verifying division 2 when assuming failures in division 1 and vice versa would be equivalent. Can similar situations be determined automatically, especially for larger systems?

Let us now return to model-checking of f. If arbitrary failures happen in division 1, this adds new behaviors to the model compared to the fault-free case and keeps any previously existing behaviors. Hence, if f is proved to be correct under the presence of failures, checking it for a fault-free system is not needed.

Unfortunately, this reasoning is not applicable to CTL properties due to their non-linear semantics. Now suppose that we need to check a universal reachability property $g = \mathbf{AG}\,\mathbf{EF}\,\neg\texttt{out}$ (''the false value of out is always reachable''). Model-checking g in NuSMV with no failures yields a positive outcome. The same happens if the failures are injected into the outputs of both divisions u_{top} (or, equivalently, if u_{top} is omitted from the system and replaced with nondeterministic inputs to u_{bottom}). However, when a failure is injected into exactly one division of u_{top} (like shown in Fig. 1),

g becomes violated.[4] Is it possible to model-check universal reachability properties while still having a more reliable result for strictly more severe failure assumptions?

B. VERIFICATION CONFIGURATIONS

A *verification configuration* (from now on, also configuration) is a tuple $c = (u, i, \phi)$, where u is a *viewpoint unit*, $1 \le i \le d(u)$ is its *viewpoint division*, and ϕ is a *failure assignment*. Semantically, c corresponds to model-checking a property that involves the variables of module $M(u, i)$ and its upstream components while assuming that the formal model is modified according to ϕ.

Suppose that we need to model-check the LTL property $\mathbf{G}\,\mathtt{out}$ of u_{bottom} in Fig. 1, assuming the single failure criterion. Given that verification with failures in the verified division of u_{bottom} is meaningless, we get the following configurations: $c_{1,2} = (u_{\text{bottom}}, 1, \{(g, \{2\})\})$ and $c_{2,1} = (u_{\text{bottom}}, 2, \{(g, \{1\})\})$.[5] For illustration purposes, we also consider fault-free configurations: $c_{1,\varnothing} = (u_{\text{bottom}}, 1, \{(g, \varnothing)\})$ and $c_{2,\varnothing} = (u_{\text{bottom}}, 2, \{(g, \varnothing)\})$.

C. DOMINATION OF CONFIGURATIONS

Suppose that F is the overall fault-free formal model, i.e., the one with identity modules on connections. Let F_ϕ be the formal model obtained from F by assuming that the failure modules are placed on connections according to ϕ. Configuration $c_1 = (u, i_1, \phi_1)$ *dominates* configuration $c_2 = (u, i_2, \phi_2)$, denoted as $c_1 \ge c_2$, if $B(F_{u,i_1,\phi_1}) \supseteq B(F_{u,i_2,\phi_2})$. Clearly, $\ge$ is a partial order on configurations. We say that configurations c_1 and c_2 are *equivalent*, denoted as $c_1 \equiv c_2$, if $c_1 \ge c_2$ and $c_2 \ge c_1$, and that c_1 *strictly dominates* c_2, denoted as $c_1 > c_2$, if $c_1 \ge c_2$ and $\neg(c_1 \equiv c_2)$. Clearly, $\equiv$ is an equivalence relation on configurations.

How can the domination relation be used to simplify model checking? This can be done by reducing the number of verification configurations to consider. First, suppose that we model-check an LTL property f for all $d(u)$ divisions of unit u, and the failure criterion that must be accounted for during verification corresponds to failure assignments $\Phi = \{\phi_1, \ldots, \phi_r\}$. In this case, all configurations from the set $Q = \{(u, i, \phi) \mid 1 \le i \le d(u), \phi \in \Phi\}$ must be checked. Nonetheless:

1) if for $c_1, c_2 \in Q$ we know that $c_1 > c_2$, then the positive result of model checking c_1 would imply the one of c_2;
2) if we know that some $\hat{Q} \subseteq Q$ is an equivalence class under $\equiv$, it is sufficient to check any $c \in \hat{Q}$.

In Fig. 1, accounting for the identity of the module instances in different divisions of the units and the connections between u_{top} and u_{bottom}, we get: $c_{1,2} \equiv c_{2,1}$, $c_{1,\varnothing} \equiv c_{2,\varnothing}$, and each of $c_{1,2}$ and $c_{2,1}$ strictly dominate $c_{1,\varnothing}$ and $c_{2,\varnothing}$. As a result, it remains to verify either $c_{1,2}$ or $c_{2,1}$. Finally, note that the input variables of $M(u_{\text{bottom}}, 1)$ and $M(u_{\text{bottom}}, 2)$ are not symmetric and had the connections to the input variables of $M(u_{\text{bottom}}, 1)$ or $M(u_{\text{bottom}}, 2)$ been swapped, $c_{1,2}$ and $c_{2,1}$ would become incomparable. How could we obtain such conclusions automatically?

D. COMPUTING THE DOMINATION RELATION

First, we extend the notion of a verification configuration. An *extended configuration* is a tuple (x, ϕ, φ), where:

1) $x \in V_\Gamma$ is a component;
2) ϕ is defined as in the case of a plain configuration;
3) $\varphi \in \{0, 1\}$ is a *failure assumption*: whether a failure directly affects x in addition to failures given by ϕ.

A plain configuration $c = (u, i, \phi)$ can be extended as follows: $E(c) = (M(u, i), \phi, 0)$. We also extend $\ge$ and $\equiv$ to cover extended configurations. For an extended configuration (x, ϕ, φ), a sequence $\kappa(x, \phi, \varphi)$ of sets of *child extended configurations* is defined:

1) if x is a system input variable, $\kappa(x, \phi, \varphi)$ is empty;
2) if x is an output variable of some module y, then $\kappa(x, \phi, \varphi)$ is a singleton sequence consisting of $\{(y, \phi, \varphi)\}$;
3) if x is a module with input variables $v_1^{\text{in}}, \ldots, v_m^{\text{in}}$, then $\cup\kappa(x, \phi, \varphi) = \{(y_i, \phi, \max(\varphi, \phi(x), \phi(y_i))) \mid 1 \le i \le m\}$, where y_i is the beginning of the connection whose end is v_i^{in}, the grouping of elements to the nested sets of $\kappa(x, \phi, \varphi)$ is done according to the input symmetries of x, and these sets are listed in a fixed order for modules of each unit.

We say that components x_1 and x_2 are *comparable* if they are both modules of the same units, system input variables of the same system input, or output variables of modules of the same unit with the same indices.

For given extended configurations $c_1 = (x_1, \phi_1, \varphi_1)$ and $c_2 = (x_2, \phi_2, \varphi_2)$ with $\kappa(x_1, \phi_1, \varphi_1) = (s_{1,1}, \ldots, s_{1,t})$ and $\kappa(x_2, \phi_2, \varphi_2) = (s_{2,1}, \ldots, s_{2,t})$, $c_1 \ge c_2$ can be concluded[6] if and only if x_1 and x_2 are comparable and either the following conditions is satisfied:

1) $\varphi_1 = 1$ (the outputs of x_1 can be directly substituted by failing signals, and x_2 will be unable to have more severe failures);
2) $\varphi_1 = \varphi_2 = 0$ and $\phi(x_1) = 1$ (due to the same reason);
3) $\varphi_1 = \varphi_2 = \phi(x_1) = \phi(x_2) = 0$ and for all $1 \le i \le t$ there are orderings $(c_{1,i,1}, \ldots, c_{1,i,r})$ and $(c_{2,i,1}, \ldots, c_{2,i,r})$ of elements from $s_{1,i}$ and $s_{2,i}$ respectively such that $c_{1,i,j} \ge c_{2,i,j}$ for all $1 \le j \le r$ (the upstream components of x_1 have failures at least as severe as the corresponding upstream components of x_2 have).

These rules can be used to recursively check domination for extended configuration pairs in $\{E(c) \mid c \in Q\}$, where Q

[4] The 3s on-delay element at the end of the non-failing connection (A2 to B2) only receives 1s signal pulses, which means that its output is never set. The other 3s on-delay element, under normal circumstances, resets the SR flip-flop after the setting 1s pulse (A1 to B1) is over, but here, a failure can cause a longer signal pulse, which keeps the set-priority flip-flop on until after the 3s on-delay is over. From that point on, no signal can reset the 3s on-delay, nor therefore the flip-flop. Note that the design is not meant to make sense as a real function, but to prove our point.

[5] Here, we define failure assignments by their graphs.

[6] It might be possible than a configuration dominates another configuration due to properties of formal models other than the ones considered in this article, e.g., due to the implementations of individual modules. Our approach will not detect such cases.

is the set of plain configurations to be model-checked. We implemented this computation in Prolog. Recursive application of rule 3 eventually terminates since Γ is acyclic and system input variables have no child extended configurations. Unfortunately, this rule may need to consider all permutations of elements within each input symmetries in modules. Nonetheless, in our case study, where symmetries are at most of size 4, we are still able to compute the entire matrix of the domination relation in less than one second.

E. SYMMETRY BREAKING WHILE CHECKING GLOBAL POSSIBILITY

For LTL, the reasoning of Sections IV-C and IV-D was applicable due to the following: if F_2 is obtained from F_1 by adding failures on one or more connections, then $B(F_2) \supseteq B(F_1)$ and hence, due to the semantics of LTL, if h is an LTL property satisfied for F_2, h is necessarily satisfied for F_1.

Now suppose that we need to model-check a CTL property. Unfortunately, for CTL, the reasoning of Sections IV-C and IV-D is not applicable since a CTL property is not a predicate that must be satisfied for all behaviors of the formal model. Nonetheless, we will show how to make it applicable for a global possibility property $g = \mathbf{AG}\,\mathbf{EF} f$.

Suppose that g is false for F_1, which means that there is a reachable state σ_1 in F_1 such that for all paths (i.e., in the graph formed of states and transitions of F_1) from σ_1 to some state σ_2' we have $\neg f(\sigma_2')$. We now consider a refined way of adding failures to F_1 so that g is also false for F_2: we augment F_1 and F_2 with a *global failure bit* γ, which is initialized nondeterministically and allowed to change from 1 to 0 on any step but not vice versa. Failure modules in F_1 and F_2 are only allowed to manifest themselves (i.e., substitute signal values) when $\gamma = 1$.

We now show that g is false in F_2. First, it is sufficient to assume that σ_1 has $\gamma = 0$ (otherwise, we may take the corresponding state with $\gamma = 0$, from which f is still unreachable). Second, we consider the same state σ_2 in F_2, also with $\gamma = 0$. Due to failures being disabled, f is again unreachable from this state. Intuitively, in F_2, the failures may drive the checked module M to a potentially larger set of states, but, once γ becomes 0, reachability of f in F_2 and F_1 from the same state becomes equivalent.

In addition, we compare a model with refined failures F_2' with the same model with usual nondeterministic failures F_1'. Again, if g is false in F_2', it will be false in F_1': we take the same state σ_1 in F_1' that witnesses the unreachability of f, then look at the corresponding state σ_2 in F_2 with $\gamma = 0$ (σ_2 can be reached by mimicking the path to it in F_1 with $\gamma = 1$, and setting $\gamma = 0$ on the last transition) and see that f is unreachable from σ_2. Thus, model checking global reachability properties with refined failures not only adheres to symmetry breaking, but also yields more reliable results.

Note that this refinement of the way of adding failures does not affect LTL model checking. At the same time, it increases resource consumption of model checking and thus we do not use this refinement when checking LTL properties.

F. SUPPORTED REQUIREMENT CLASSES

According to the aforementioned assumptions, temporal properties that are compatible with the proposed approach refer to a particular module of interest $M(u, i)$, called the *viewpoint*, while having access also to the variables of all upstream modules of $M(u, i)$. When specifying such properties, i is replaced with a placeholder for the chosen division, which will be substituted with a concrete division should it be chosen for verification.

We consider the following classes of temporal properties:

1) *Common LTL* properties adhere to the aforementioned constraints and are checked under the chosen failure tolerance criteria. They correspond to request-response or absence of spurious actuation requirements.
2) *Isolated LTL* properties are similar to common LTL ones but only involve the variables of $M(u, i)$ and thus are unaffected by the failure tolerance criteria. In addition, they can correspond to invariants over the outputs of this unit.
3) *Global possibility* (Section III-C) properties are checked under the chosen failure tolerance criteria with the failure injection technique presented in Section IV-E.

By contrast, the following property classes are incompatible with the proposed approach:

1) Properties that inquire into the joint behavior of at least two modules that are not upstream/downstream with respect to each other. These properties do not correspond to any viewpoints.
2) Properties that distinguish the divisions of units other than u (e.g., require the values of variables in two particular modules rather than the variables of two arbitrary modules to be true). Configuration domination and equivalence reasoning is inapplicable for such properties.
3) Properties that refer to internal components of modules other than $M(u, i)$ if these modules can be affected by failures according to the chosen failure criterion. This is a technical limitation caused by failures being only injected to connections and can be avoided by wiring the queried variables to extra outputs added to their modules.

In [11], we introduced the class of so-called black-box properties. They do not violate the assumptions above, but some of the assumptions of black-box properties, such as the prohibition of any references to internal variables, can be relaxed.

V. EXPERIMENTAL EVALUATION

A. CASE STUDY

Our case study is based on the U.S. EPR NPP materials [23], [24], our previous case study [11] and our own invention. As in [11], it includes three fault-tolerant subsystems: the 4-redundant protection system (PS), the 2-redundant safety automation system (SAS), and the 4-redundant priority and actuator control system (PACS). These systems implement

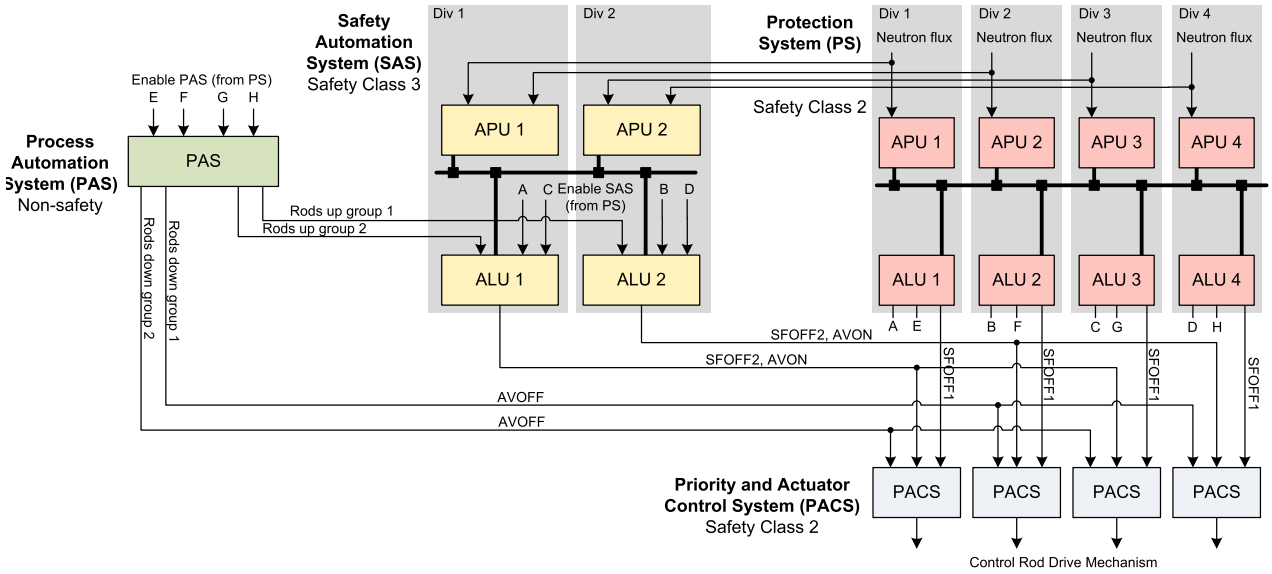


FIGURE 2. Structure of the case study.

two safety functions: preventive protection and reactor protection. Due to the PS and the PACS being jointly responsible for reactor protection, we view them as parts of a single unit group. The PS and the SAS are decomposed into units of two types: acquisition and processing units (APUs) and actuation logic units (ALUs). APUs of each subsystem are connected to the ALUs of the same subsystem in an all-to-all fashion. One more component that we add to this case study in this work is the process automation system (PAS), which is responsible for the normal operation of the NPP, a non-safety function. Accordingly, the PAS has only one division. Note that, to mimic the practical impossibility of following DiD principles perfectly, we have deliberately added many connections across the DiD levels (we do not claim such design choices would be justifiable in real-world systems).

The structure of the case study is shown in Fig. 2. The internal structure of the PAS is shown in Fig. 3. The implementations of some other subsystems can be found in our previous work [11].[7]

B. FUNCTIONAL REQUIREMENTS

According to the Finnish regulatory guides [1] (item 442), the failure criterion is applied to the complete set of systems needed to execute a safety function (associated with a DiD level). A failure in a "lower" DiD level shall not prevent the function in a "higher" DiD level from bringing the plant to controlled/safe state, even if the failure is total (CCF). We therefore subdivide the functional requirements to be checked into several scenarios according to the functions to which they are related:

1) **Level 1 function: normal operation.** The PAS is solely responsible for this function. There is no failure criterion, i.e., the PS, from which PAS receives inputs, is assumed to be fault-free.
2) **Level 2 function: preventive protection.** SAS and PACS shall together satisfy the single failure criterion, i.e., a single failure in either SAS or PACS (but not both) shall not prevent the function from operating. The function shall also tolerate a simultaneous CCF of PAS. During verification, however, this failure criterion can be simplified by assuming a single failure in SAS only: if the viewpoint is in SAS, then it is not affected by the outputs of PACS (see Fig. 2) and if the viewpoint is in PACS, then it is not affected by other PACS units.

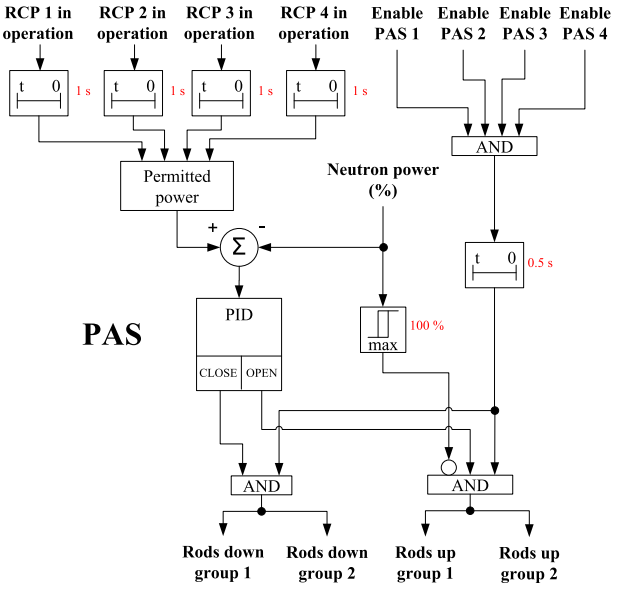


FIGURE 3. Implementation of the PAS as a function block diagram. The "Permitted power" block calculates a limit for the allowed reactor power based on the cooling capacity (number of running reactor coolant pumps). The PID block controls the rods based on the difference between the measured power level and the permitted power.

[7]In the present work, some of the implementations were insignificantly modified to account for the introduction of the PAS.

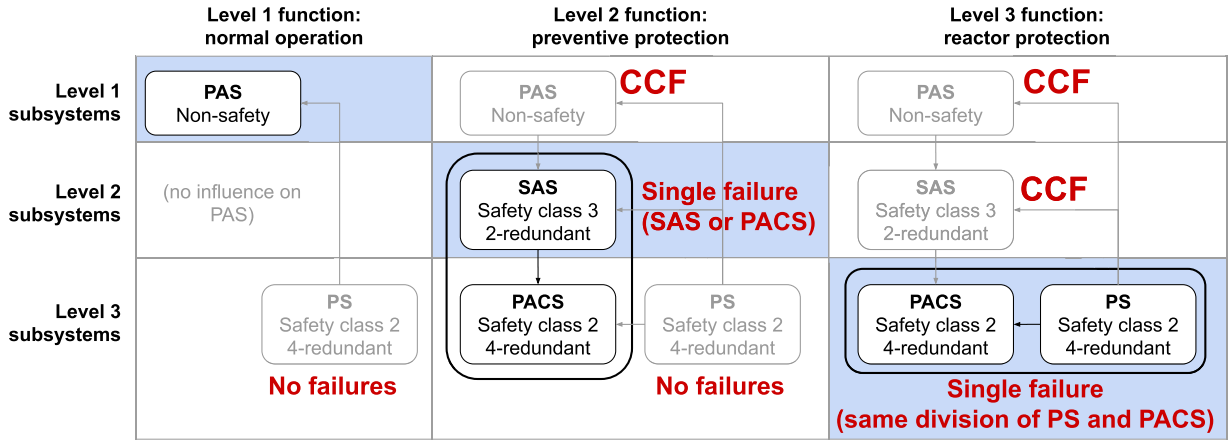


FIGURE 4. Considered verification scenarios (three columns) and the corresponding failure tolerance criteria (shown in bold red).

3) **Level 3 function: reactor protection.** PS and PACS shall together satisfy the single failure criterion, meaning that a single failure is allowed in the same divisions of PS and PACS. A single failure in, e.g., the shared SC2 power supply might cause a simultaneous failure in the same division of both PS and PACS (a consequential failure). The function shall also tolerate a simultaneous CCF in SAS and/or PAS.

The considered verification scenarios and the corresponding failure tolerance criteria are summarized in Fig. 4. In addition, we consider one more scenario:

4) **Artificial scenario**. The single failure criterion is applied to all subsystems independently (with PS and PACS still having failures in the same divisions), not accounting for DiD levels. This scenario is included to compare this work with our previous work [11]. As requirements, we use common LTL and universal reachability properties for SAS and PACS from scenarios 2 and 3 above.

C. EXPERIMENTAL SETUP

The techniques presented in this article were implemented in Java and Prolog as a part of the HW-SW-builder tool [11], which is available online.[8] The models and requirements that we used for our case study can also be found there. Experiments were performed on a single core of 2 GHz Intel Core i7-4510U CPU.

We enhanced HW-SW-builder with support of requirement annotations with viewpoints and allowed numbers of failures in each unit group (in either none, one, or all divisions). Once the tool encounters a new combination of a viewpoint and a failure assignment, it performs symmetry analysis as described in Section IV. Configurations with failures at the viewpoint are excluded from consideration. A separate Prolog query is made for each pair of verification configurations, except for the cases that can be deduced automatically based on transitivity and reflexivity of $\geq$. Then, the property is verified for configurations that were found to be sufficient. We use LTL and CTL model checking based on binary decision diagrams (BDDs) and run nuXmv with command line options "`-dynamic -df -coi`".

[8]https://github.com/igor-buzhinsky/hw-sw-model-builder

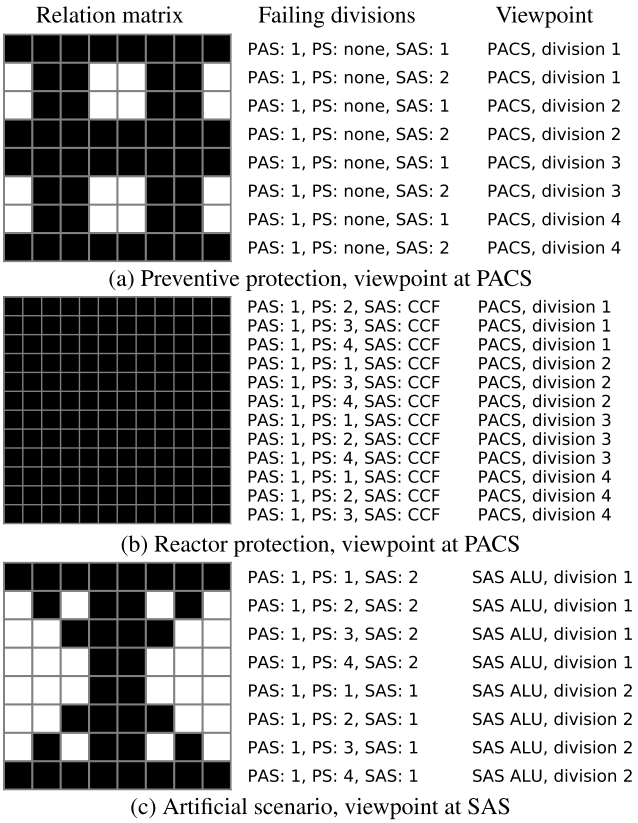
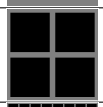
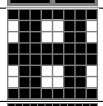
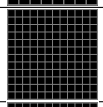
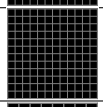
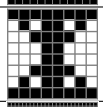
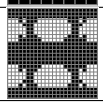


FIGURE 5. Examples of identified domination relations. In each relation matrix, each row and column corresponds to a configuration, which are shown for the rows on the right. Annotations for columns are not shown, but their order is the same as the one of the rows. A cell is black if the configuration of the current row dominates the configuration of the current column and white otherwise. Accordingly, black rows correspond to configurations that dominate all other configurations.

D. RESULTS

We separately examined each failure scenario and each viewpoint, so the symmetry analysis phase is performed once per this combination. In Fig. 5, several examples of domination relation matrices are shown with annotations. The full list of scenario/viewpoint combinations and their relation matrices

TABLE 1. Results of symmetry analysis and model checking. For each scenario-viewpoint pair, the matrix of the domination relation is given. The notation is the same as in Fig. 5.

Failure scenario	Viewpoint unit	Relation matrix	Symmetry analysis time (s)	Property class	Number of properties	Average model checking time (s)
Normal operation	PAS		0.4	Common LTL	6	0.6
				Universal reachability	8	0.8
Preventive protection	SAS ALU		0.4	Common LTL	14	0.6
				Isolated LTL	3	0.3
				Universal reachability	4	3.6
	PACS		0.4	Common LTL*	2	16.6
				Isolated LTL	2	0.1
				Universal reachability	4	3.0
Reactor protection	PS ALU		0.4	Common LTL	22	1.9
				Isolated LTL*	8	0.2
				Universal reachability	4	0.1
	PACS		0.4	Common LTL*	7	13.2
				Isolated LTL	2	0.1
				Universal reachability	4	5.0
Artificial scenario	SAS ALU		0.4	Common LTL	14	0.6
				Universal reachability	4	7.5
	PACS		0.6	Common LTL*	7	9.2
				Universal reachability	4	4.6

* In these experiments, the cone of influence (COI) nuXmv optimization was turned off to avoid a nuXmv bug.

is given in Table 1 together with times spent on symmetry analysis and model checking. In all failure scenarios, there exist configurations that dominate all other configurations. Our tool selects the topmost of such configurations for actual verification.

Although the analysis includes checks over various permutations of symmetric connections, as visible from the table, symmetry analysis times on our case study are negligible. The minimum analysis time is 0.4 s, which is the time spent to create the Prolog model of the system.

Each considered temporal property was model-checked within five minutes. Average model checking times are often at most several seconds, except for three LTL verification cases where we disabled the COI reduction as otherwise we encountered a nuXmv bug. Our tool does remove unused model components (e.g., divisions of the PACS other that the viewpoint) automatically, but since our failure blocks benefit from COI reduction (namely, downstream modules of the failure blocks can be optimized out), this somewhat impacts model checking time.

E. COMPARISON WITH PREVIOUS WORK

The idea of fault tolerance verification of a modular nuclear I&C system was introduced in [11]. The difference of our case study and experimental setup from [11] are:

1) the case study was extended by adding the PAS;
2) failure modules were improved so that they do not take the signal to be altered into account and thus benefit more from COI reduction;
3) in addition, failure modules were altered while checking universal reachability properties as explained in Section IV-E;
4) in this work, we do not report in detail the results of model checking without failures, with communication delays, and with BMC (some comments regarding these cases are nonetheless given below).

A brief comparison of model checking times if possible: the configurations C_{PS}, C_{SAS} and C_{PAC} from [11] roughly correspond to reactor protection verification for PS and the artificial scenario for SAS and PACS, respectively. Notably, now verification of universal reachability (CTL) properties always terminates and is faster (several seconds instead of several minutes for SAS and PACS). This change is due to the enhancement of failure modules. As before, no violations of universal reachability were found, even with the failure module enhancement. Finally, the identified domination relations for the artificial scenario fully comply with the manual reasoning in [11], paragraphs 5–6 of Section IV-B.

To mimic more experiments from [11], we also considered the cases of BMC, fault-free verification and verification with bounded communication delays. We did not find notable discrepancies from our previous results. In particular, verification with delays is still a computational challenge and is often possible only with BMC. However, we were also unable to verify three LTL properties for the PACS with BDD-based model checking in the fault-free, no-delay scenario, but these cases are affected by the disabled COI reduction.

VI. RELATED WORK

Discovery and utilization of symmetries is a rather general idea in formal verification. Partial order reduction [25] is a technique to reduce the state space in verification of distributed systems. Full and partial symmetries in distributed systems were used to reduce formal models in [26]. Symmetry reduction for programs specified in the B language and the CSP process algebra was considered in [27] and [28] respectively. Symmetry breaking techniques for propositional encoding of transitions systems were proposed in [29].

Fault tolerance in redundant safety-critical systems was previously addressed in [18], [30]–[36]. The paper [11] gives a brief overview of some of these works, which, unlike our work, mostly consider detailed fault models. Tolerance to single faults was verified in [30]. CCFs were addressed in [37] from the point of view of probabilistic safety assessment (PSA). Our approach, by contrast, only considers possibility but not probability of fault scenarios. Probabilistic analysis of fault-tolerant redundant systems was also considered in [35], [36]. In particular, work [35] focuses on overcoming the combinatorial explosion caused by multiple system components protected with redundancies. Our work is motivated by a similar idea but applied to the explosion of the number of verification configurations.

Modular nuclear I&C systems can be viewed as a class of computer networks. In verification of computer networks, however, the properties to be verified usually concern delivery of packets rather than the functioning of the algorithm implemented by the network. Symmetries in computer networks were used to simplify formal verification in [38]–[40]. In [40], fault tolerance of computer networks was considered.

VII. CONCLUSION

In this work, we have advanced our previous approach [11] of model-checking nuclear I&C systems under failure tolerance assumptions. Our key contribution is the formal method to automatically determine how the symmetries and redundancies existing in the system under verification can be used to reduce the number of scenarios to be considered during verification. Although we used such reasoning also in [11], only in the present work it is automated. The value of automatic reasoning is emphasized by using a case study that is larger than the one in [11], has a complex structure and is paired with specifications that must be checked under different failure assumptions, which now also include CCFs in addition to single failures. Symmetry analysis for this case study takes less than a second and speeds up model checking in up to 24 times (compared to the naive approach of verifying all possible configurations; this number is reached on the most complex artificial scenario). Finally, we amended the way of checking **AG EF** CTL properties with failure assumptions to make it compatible with the proposed approach and also cover more scenarios.

The proposed approach has several limitations, which might be addressed in future work:

1) We do not consider asynchrony and communication delays. In Section V-E, we shortly comment on following the delay modeling approach from [11]. A more advanced approach [19] has not yet been considered.
2) We require that the modules of the I&C model are described by an acyclic graph (see Section III-A). Although we did not include cycles into our case study, they are possible and may be needed to, e.g., implement periodic tests. To support cycles, domination of configuration can be proven in terms of inclusion of finite behavior sets. If this is proven for all bounds k on behavior lengths, then it is easy to see that this inclusion also holds for infinite behaviors. Reasoning over finite behaviors can be done inductively, with separate proofs for induction base and step. When proving the base, a cycle will vanish since at least one unit on it has not yet communicated its outputs to other modules, and this output will be substituted by some default value. When proving the step, we can use the proof for $k - 1$ in the same places.
3) Temporal properties that can be checked with the proposed approach are constrained as described in Section IV-F. We are aware of properties that violate these constraints, but they are meaningful to check for our case study only in the fault-free scenario.

REFERENCES

[1] STUK. (2019). *YVL B.1 Safety Design of a Nuclear Power Plant*. [Online]. Available: https://www.stuklex.fi/en/ohje/YVLB-1

[2] K. Björkman, J. Frits, J. Valkonen, J. Lahtinen, K. Heljanko, I. Niemelä, and J. J. Hämäläinen, "Verification of safety logic designs by model checking," in *Proc. 6th Amer. Nucl. Soc. Int. Topical Meeting Nucl. Plant Instrum., Control, Hum.-Mach. Interface Technol. (NPIC HMIT)*, 2009, pp. 5–9.

[3] J. Lahtinen, J. Valkonen, K. Björkman, J. Frits, I. Niemelä, and K. Heljanko, "Model checking of safety-critical software in the nuclear engineering domain," *Rel. Eng. Syst. Saf.*, vol. 105, pp. 104–113, Sep. 2012.

[4] A. Pakonen, T. Tahvonen, M. Hartikainen, and M. Pihlanko, "Practical applications of model checking in the Finnish nuclear industry," in *Proc. 10th Int. Topical Meeting Nucl. Plant Instrum., Control Hum. Mach. Interface Technol. (NPIC HMIT)*, La Grange Park, IL, USA: American Nuclear Society, 2017, pp. 1342–1352.

[5] A. Pakonen, I. Buzhinsky, and K. Björkman, "Model checking reveals design issues leading to spurious actuation of nuclear instrumentation and control systems," *Rel. Eng. Syst. Saf.*, vol. 205, Jan. 2021, Art. no. 107237.

[6] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.

[7] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press, 2008.

[8] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L.-J. Hwang, "Symbolic model checking: 10^{20} states and beyond," *Inf. Comput.*, vol. 98, no. 2, pp. 142–170, 1992.

[9] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," *Adv. Comput.*, vol. 58, pp. 117–148, 2003.

[10] A. Cimatti and A. Griggio, "Software model checking via IC3," in *Proc. Int. Conf. Comput. Aided Verification*. Berlin, Germany: Springer, 2012, pp. 277–293.

[11] I. Buzhinsky and A. Pakonen, "Model-checking detailed fault-tolerant nuclear power plant safety functions," *IEEE Access*, vol. 7, pp. 162139–162156, 2019.

[12] J. Yoo, S. Cha, and E. Jee, "A verification framework for FBD based software in nuclear power plants," in *Proc. 15th Asia–Pacific Softw. Eng. Conf.*, Dec. 2008, pp. 385–392.

[13] M. Lin, D. Hou, P. Liu, Z. Yang, and Y. Yang, "Main control system verification and validation of NPP digital I&C system based on engineering simulator," *Nucl. Eng. Des.*, vol. 240, no. 7, pp. 1887–1896, Jul. 2010.
[14] E. Németh and T. Bartha, "Formal verification of safety functions by reinterpretation of functional block based specifications," in *Formal Methods for Industrial Critical Systems*, D. Cofer and A. Fantechi, Eds. Berlin, Germany: Springer, 2009, pp. 199–214.
[15] B. Fernández Adiego, D. Darvas, E. Blanco Viñuela, J.-C. Tournier, S. Bliudze, J. Olaf Blech, and V. Manuel González Suárez, "Applying model checking to industrial-sized PLC programs," *IEEE Trans. Ind. Informat.*, vol. 11, no. 6, pp. 1400–1410, Dec. 2015.
[16] IAEA. (2018). *Approaches for Overall Instrumentation and Control Architectures of Nuclear Power Plants*. International Atomic Energy Agency, Nuclear Energy Series NP-T-2.1. [Online]. Available: http://www-pub.iaea.org/MTCD/Publications/PDF/PUB1821_web.pdf
[17] WENRA Reactor Harmonization Working Group. (2013). *Safety of New NPP Designs*. Western European Nuclear Regulators' Association, Report. [Online]. Available: http://www.wenra.org/media/filer_public/2013/08/23/rhwg_safety_of_new_npp_designs.pdf
[18] J. Lahtinen, "Hardware failure modelling methodology for model checking," VTT, Espoo, Finland, Tech. Rep. VTT-R-00213-14, 2014.
[19] I. Buzhinsky and A. Pakonen, "Timed model checking of fault-tolerant nuclear I&C systems," in *Proc. IEEE Int. Conf. Ind. Informat. (INDIN)*, Jul. 2020, pp. 159–164.
[20] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An opensource tool for symbolic model checking," in *Proc. Int. Conf. Comput. Aided Verification*. Berlin, Germany: Springer, 2002, pp. 359–364.
[21] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The nuXmv symbolic model checker," in *Proc. CAV*, in Lecture Notes in Computer Science, vol. 8559, A. Biere and R. Bloem, Eds. Cham, Switzerland: Springer, 2014, pp. 334–342.
[22] A. Pakonen, T. Matasniemi, J. Lahtinen, and T. Karhela, "A toolset for model checking of PLC software," in *Proc. IEEE 18th Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2013, pp. 1–6.
[23] Areva NP. (2012). *U.S. EPR Protection System, Technical Report ANP-10309NP, Revision 4*. [Online]. Available: https://www.nrc.gov/docs/ML1216/ML121660317.html
[24] reva NP. (2013). *U.S. EPR Final Safety Analysis Report*. [Online]. Available: https://www.nrc.gov/reactors/new-reactors/design-cert/epr/reports.html
[25] D. Peled, "Ten years of partial order reduction," in *Proc. Int. Conf. Comput. Aided Verification*. Berlin, Germany: Springer, 1998, pp. 17–28.
[26] E. A. Emerson and R. J. Trefler, "From asymmetry to full symmetry: New techniques for symmetry reduction in model checking," in *Proc. Adv. Res. Working Conf. Correct Hardware Design Verification Methods*. Berlin, Germany: Springer, 1999, pp. 142–157.
[27] E. Turner, M. Leuschel, C. Spermann, and M. Butler, "Symmetry reduced model checking for B," in *Proc. 1st Joint IEEE/IFIP Symp. Theor. Aspects Softw. Eng. (TASE)*, Jun. 2007, pp. 25–34.
[28] T. Gibson-Robinson and G. Lowe, "Symmetry reduction in CSP model checking," *Int. J. Softw. Tools Technol. Transf.*, vol. 21, no. 5, pp. 567–605, Oct. 2019.
[29] J. Rintanen, "Symmetry reduction for SAT representations of transition systems," in *Proc. Int. Conf. Automated Planning Scheduling (ICAPS)*, 2003, pp. 32–41.
[30] M. Zhang, Z. Liu, C. Morisset, and A. P. Ravn, "Design and verification of fault-tolerant components," in *Methods, Models Tools for Fault Tolerance*, M. Butler, C. Jones, A. Romanovsky, and E. Troubitsyna, Eds. Berlin, Germany: Springer, 2009, pp. 57–84.
[31] A. Joshi and M. P. E. Heimdahl, "Model-based safety analysis of Simulink models using SCADE design verifier," in *Computer Safety, Reliability, and Security*, R. Winther, B. A. Gran, and G. Dahll, Eds. Berlin, Germany: Springer, 2005, pp. 122–135.
[32] M. P. E. Heimdahl, Y. Choi, and M. W. Whalen, "Deviation analysis: A new use of model checking," *Automated Softw. Eng.*, vol. 12, no. 3, pp. 321–347, Jul. 2005.
[33] F. Schneider, S. M. Easterbrook, J. R. Callahan, and G. J. Holzmann, "Validating requirements for fault tolerant systems using model checking," in *Proc. IEEE Int. Symp. Requirements Eng., RE*, Apr. 1998, pp. 4–13.
[34] C. Bernardeschi, A. Fantechi, and S. Gnesi, "Model checking fault tolerant systems," *Softw. Test., Verification Rel.*, vol. 12, no. 4, pp. 251–275, Dec. 2002.
[35] C. Dubslaff, K. Ding, A. Morozov, C. Baier, and K. Janschek, "Breaking the limits of redundancy systems analysis," 2019, *arXiv:1912.05364*. [Online]. Available: http://arxiv.org/abs/1912.05364
[36] C. Dubslaff, A. Morozov, C. Baier, and K. Janschek, "Reduction methods on probabilistic control-flow programs for reliability analysis," 2020, *arXiv:2004.06637*. [Online]. Available: http://arxiv.org/abs/2004.06637
[37] J. K. Vaurio, "Common cause failure probabilities in standby safety system fault tree analysis with testing—Scheme and timing dependencies," *Rel. Eng. Syst. Saf.*, vol. 79, no. 1, pp. 43–57, Jan. 2003.
[38] G. D. Plotkin, N. Bjørner, N. P. Lopes, A. Rybalchenko, and G. Varghese, "Scaling network verification using symmetry and surgery," *ACM SIGPLAN Notices*, vol. 51, no. 1, pp. 69–83, Apr. 2016.
[39] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "Control plane compression," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 476–489.
[40] N. Giannarakis, R. Beckett, R. Mahajan, and D. Walker, "Efficient verification of network fault tolerance via counterexample-guided refinement," in *Proc. Int. Conf. Comput. Aided Verification*. Cham, Switzerland: Springer, 2019, pp. 305–323.

IGOR BUZHINSKY was born in 1992. He received the B.Sc. and M.Sc. degrees in applied mathematics and computer science from ITMO University, Saint Petersburg, Russia, in 2013 and 2015, respectively, the second M.Sc. degree in software engineering and service design from the University of Jyväskylä, Jyväskylä, Finland, in 2015, and the D.Sc.(Tech.) degree from Aalto University, Espoo, Finland, in 2019.

He is currently a Postdoctoral Researcher with Aalto University and a Research Fellow with ITMO University. His research interests include formal verification, synthesis of finite-state models, and reliability of deep learning.

ANTTI PAKONEN was born in 1979. He received the M.Sc.(Tech.) degree in I&C systems from the Helsinki University of Technology, Espoo, Finland, in 2004.

He is currently a Senior Scientist and a Project Manager with VTT Technical Research Centre of Finland Ltd., Espoo, where he has been employed since 2002. His research interests include I&C software engineering, I&C architecture evaluation, practical application of model checking in industrial applications, and knowledge management.