Compositional Verification of Nuclear Safety I&C Systems with OCRA

Pakonen, Antti

Published in:
2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)

DOI:
10.1109/ETFA61755.2024.10711009

Published: 13/09/2024

Document Version
Peer reviewed version

Compositional verification of nuclear safety I&C systems with OCRA

Antti Pakonen
VTT Technical Research Centre of Finland Ltd., Espoo, Finland
antti.pakonen@vtt.fi

***Abstract*—Model checking is a powerful formal verification method. However, due to the complexity of instrumentation and control (I&C) system logics—even in critical applications like nuclear power plant safety systems—the challenge of state space explosion means that the analyses cannot always be performed in reasonable time. In compositional verification, this challenge is overcome by reasoning over the system subcomponents separately, and then resolving proof claims for the composite system as a whole.**

In this paper, I present my experiments with OCRA, a tool for the verification of contract requirement. Together with the model checker nuXmv, OCRA can be used for compositional model checking. My industrial case study relates to three legacy I&C systems of the Finnish Olkiluoto nuclear power plant, about to be renewed using (mostly) software-based logic. I concretize the challenges in verifying complex nuclear I&C logics, prove the capabilities of OCRA, and discuss the practical limitations.

***Index Terms*—formal verification, model checking, control engineering, software safety**

I. Introduction

The instrumentation and control (I&C) systems of nuclear power plants (NPPs) are subject to strict requirements [1]. Nuclear regulatory bodies agree that one requirement—for safety systems, in particular—is *simplicity* [2]. However, I&C in even the highest safety class can become complex due to the need to monitor and control plant variables, execute control sequences, and handle operator interaction [3]. Safety I&C can therefore contain, e.g., software-based PID controllers and filters [4], and even the use of binary components like flip-flop memories and delays can also result in complexity limiting the coverage that software testing can achieve [5].

Model checking [6] is a formal verification technique proven to find design issues in systems already subjected to testing and other more conventional techniques. But as the analyzed systems become large and complex, there is a risk of *state space explosion* [6], [7], where the sheer number of reachable model states means that the analysis will not terminate in reasonable time.

In compositional analysis, rather than trying to reason over properties for the complex model as a whole, we reason over each subcomponent separately, based on assumptions over the behaviour of the other subcomponents [7].

In this paper, I present my experiments with OCRA [8], a tool for verification of contract refinement, developed by the Fondazione Bruno Kessler (FBK)[1]. OCRA can resolve proof claims that state that if the subcomponents satisfy their contracts, then the composite model also satisfies its contracts. Combined with the model checker nuXmv [9], OCRA can be used for compositional verification.

My real-world case study relates to three legacy I&C systems of the Olkiluoto NPP units 1 and 2 that are currently being renewed to be based mostly on software-based logic. Our earlier practical model-checking project on those systems—while successful in detecting several design issues [10]—was partially hindered due to the scale and complexity of some of the systems' functional chains.

The contribution of the paper is threefold. First, I concretize the challenges in model-checking complex nuclear I&C logics, based on practical experience. Second, I prove the capabilities of OCRA based on an real-world case study from a Finnish NPP. Third, I discuss the limitations I found in applying OCRA to a realistic industrial verification problem.

I cover the key concepts of model checking and compositional verification in Section II, and discuss nuclear I&C specific challenges in Section III. I summarize related research in Section IV. Section V describes my industrial case study. I then discuss the results in Section VI, and present the conclusions in Section VII.

II. Formal verification techniques

A. Model checking algorithms

Model checking [6] is a computer-assisted method aiming at formal proofs that a system model satisfies stated properties. A model checker is a tool that checks if a desired property holds for all possible system model execution paths. An execution path violating a property is output as a counterexample.

To avoid state space explosion, model checkers employ various techniques. Symbolic model checkers like NuSMV [11] use Decision Diagrams (BDD) [12] for canonical representation of Boolean formulas, to avoid the need to process every model state explicitly. Bounded model checking (BMC) [13] is a technique based on Boolean satisfiability (SAT) solvers, where the length of checked state transition sequences is limited. BMC can detect property violations, but if the search bound is too small, it can also produce a false positive result.

The infinite-state model checker nuXmv [9] extends the capabilites of NuSMV to real and unbounded integer number math, using Satisfiability Modulo Theory (SMT) [14] and a combination of other techniques: IC3 [15], BMC, and simple bounded model checking (SBMC) [16].

[1] https://www.fbk.eu/en/

Other well known model checkers include SPIN [17] for explicit-state analysis, UPPAAL [18] for continuous time analysis, and PRISM [19] for probabilistic model checking.

B. Property specification

The formal properties are specified using different variants of *temporal logic* languages. In Linear Temporal Logic (LTL), *temporal operators* such as "Globally" ($\mathbf{G}\ p : p$ is true at ever state of the path) or "Finally" ($\mathbf{F}\ p : p$ is true at some future state on the path) enable formulation of statements over execution paths. Past LTL operators [20] such as "Once" ($\mathbf{O}\ p : p$ is true at some past (or the current) state on the path.) are also sometimes supported.

Computation Tree Logic (CTL) [6] is based on branching time, and uses *path quantifiers* $\mathbf{A}$ (all execution paths) and $\mathbf{E}$ (some execution path) in conjunction with temporal operators like $\mathbf{G}$ and $\mathbf{F}$. Some CTL properties useful in I&C logic verification (e.g., the "Possibility" [21] pattern $\mathbf{AG\,EF}\,p$) cannot be expressed in LTL.

Property Specification Language (PSL) [22] is an extension of LTL and CTL aiming at better human readability. The Sequential Regular Expressions (SERE) "style" of PSL is convenient in expressing LTL type properties for multi-cycle behavior of I&C systems [23].

C. Compositional verification

If the behaviour of a system subcomponent is influenced by other subcomponents only to a limited degree, reasoning over the global state space can be replaced with reasoning over each subcomponent separately [7]. For concurrent programs, we can associate each subcomponent with an *assumption* on in its input and a *guarantee* on its output, and then devise a compositional proof rule ensuring mutual consistency across the assumptions and guarantees over all the subcomponents (*A-G reasoning*) [7].

OCRA [8] is a tool for automated support for contract-based design [24]. By specifying the assumptions and guarantees as temporal formulas, OCRA supports the verification of contract *refinement*. Informally, for a contract C of component S, and a set of contracts CS for the subcomponents of S, CS is a *refinement* of C, "iff (1) the correct implementations of the sub-contracts form a correct implementation of C; and (2) for each sub-contract C'', the correct implementation of the other sub-contracts and a correct environment of C form a correct environment of C''" [8].

In other words, OCRA can resolve proof claims which say that if the subcomponents of a model satisfy their contracts, then the composite model also satisfies its contracts [25].

The input file specifying the components, their ports and contracts, and their decomposition is called the OCRA System Specification (OSS). The contracts are specified in OTHELLO [26], a language that combines temporal logic with real-time aspects, using natural language expressions. For discrete-time properties, OTHELLO corresponds with LTL [25].

By verifying the contract refinement (written in OSS) with OCRA, and then validating the subcomponent models with nuXmv, the two tools can be used together to perform compositional verification [25].

III. MODEL-CHECKING NUCLEAR I&C LOGICS

A. Industry-specific challenges

VTT[2] has used model checking to verify application logic design in every notable I&C engineering project in the Finnish nuclear industry, since 2008. The method has been used in new-build (Olkiluoto 3, and the cancelled Hanhikivi 1) and I&C renewal (Loviisa 1&2, and Olkiluoto 1&2) projects [10]. By 2024, VTT has identified a total of 94 confirmed design issues—examples can be found in, e.g., [3], [5], [10], [27]. The tools and the work process are described in [3].

(In addition, VTT has successfully applied model checking to verify railway interlocking device logics for the engineering company Mipro [28].)

Complexity in the application logics is a result of several design factors:

- The logics contain delay and memory elements to filter input signals, ensure output commands get carried out, account for feedback from the controlled process, or to execute complex control sequences [23].
- Process monitoring and control algorithms can call for complex function blocks, like PID controller, or first or second order filter [27].
- In dedicated nuclear I&C platforms like TELEPERM XS (TXS) and Spinline, each signal in the block diagram carries both a value and a fault status. This status information about signal *validity* can then be used to, e.g., exclude invalid signals in majority votes [3]. (Signal validity has been an important factor in 7% of the issues VTT has detected. See [3] for an example.)

The scale and complexity of the logics means that the analysis times in model checking can become long. I have only collected and preserved statistics for the 87 models for which a counterexample revealed an actual design issue. With NuSMV, we have a case where disproving a LTL property took 438 seconds, and there are four cases where the analysis time was between two and three minutes per property. With nuXmv's IC3 algorithm, I have recorded analysis times up to 470 seconds. (I recorded the above times on an Intel Core i7-6600U with a clock rate of 2.6 GHz.) Notably, in seven exemplar cases, the analysis time became so long that the analyst eventually had to resort to BMC, which, while often fast, is in general incomplete. On the whole, prohibitively long analysis times for monolithic NuSMV or nuXmv models sometimes occur in VTT's practical projects.

B. Decomposition of nuclear I&C systems

Nuclear power plant safety I&C systems are inherently compositional. To achieve failure tolerance, the systems may have two to four redundant subsystems, so that a failure in

[2]VTT Technical Research Centre of Finland Ltd. is a state owned company providing research and innovation services. https://www.vttresearch.com/.

one subsystem will not prevent the system from carrying out its function.

As an example, I will use the Protection System (PS) of the proposed U.S. European Pressurised Reactor (U.S.EPR)[3]. A simplified view of the four-redundant PS architecture is shown in Fig. 1.

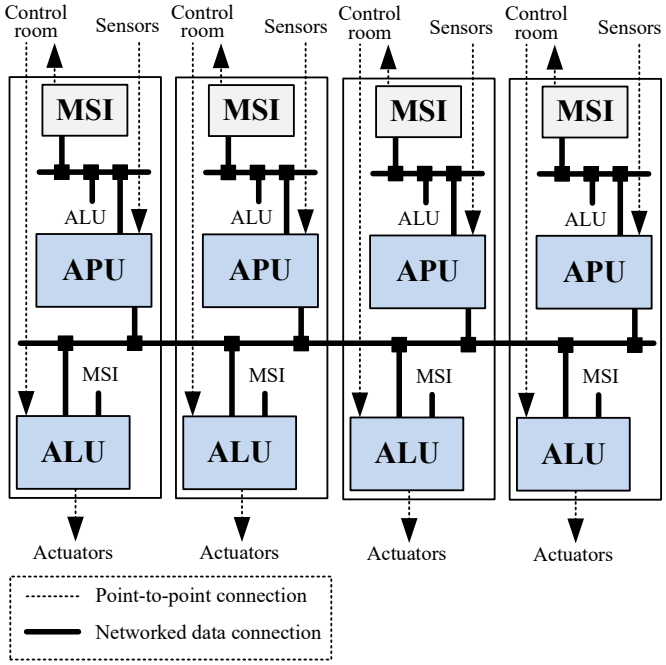

Fig. 1. Simplified view of an exemplar nuclear safety I&C system architecture with four redundant subsystems

The Acquisition and Processing Unit (APU) receives signals from process sensors through a hardwired connection, performs calculations and setpoint comparisons, and sends the results to the Actuation Logic Units (ALUs) of each subsystem via a network. Each ALU then performs a vote over the results, processes commands from the operators, and issues actuation orders. The Monitoring and Services Interface (MSI) units provide information to the operators about the controlled process and the status of the I&C equipment. (Failure tolerance is based on the concept that a failure in or before a single APU will not effect the actuation orders, because the ALUs require two (valid) votes to act. For the ALUs, it typically suffices that two of the four are working as expected.)

Another form of decomposition arises from the fact that each nuclear I&C system is allocated different *functions*. (From [4], we can count at least 72 functions for PS, 60 functions for Safety Automation System (SAS), and 18 functions for Diverse Actuation System (DAS).) Such functions can be self-standing (not depending on signals from other functions), or—as in the case study I present in Section V—form chains[4].

[3]The U.S. Nuclear Regulatory Commission has published the Final Safety Analysis Report (FSAR) for the U.S.EPR online [4].

[4]High-level examples of functions and their logical connections can also be found in [4] under Tier 2, Chapter 7, in the files ML13220A735.pdf and ML13220A737.pdf for PS, and ML13220A774.pdf for SAS.

The functional chains can also spread across different I&C systems, as shown in the running example used in [29].

IV. Related research

In addition to the applications in Finland we discussed above, model checking has also been used in other nuclear industry related I&C projects (e.g., [30]–[32]). Rail traffic applications include, e.g., [33]–[35]. One example of using nuXmv to verify industrial Programmable Logic Controller (PLC) software is [36].

In [34], Limbrée et al. decompose a railway station into symmetrical parts, verify the local properties for the parts with nuXmv, and use OCRA to verify contracts that hold for the station as a whole. The nuXmv models for the parts are directly translated from original application data.

In [35], Haxthausen and Fantechi target a similar application. Due to the topology of railway tracks, and the locality of interlocking systems, they are able to employ an abstraction technique called *model slicing* to a high degree—only the portions of the model that have influence for the track element specific property are included.

COMPASS [37] is a toolset for aerospace domain software engineering, including OCRA for contract refinement, and nuXmv for verification. Several case studies are listed in [37].

In [38], Xie et al. extend SysML with A-G contract modeling, and then present a toolset that generates an OSS file for compositional verification in OCRA. The toolset, which they demonstrate with a spacecraft navigation system case study, also incorporates Fault Tree Analysis.

A framework for compositional verification of modular robot systems is presented in [39]. While the A-G reasoning is based on First-Order Logic and temporal operators, Cardoso et al. make a point of including different techniques for the component verification task, since they cannot apply model checking to every part of their case system.

In [40], Phan et al. use A-G reasoning to prove that a successful runtime assurance for the components of a rover implies that safety properties hold for the global system.

In contrast to the above-mentioned compositional verification studies, our work focuses on process industry I&C systems. In [32], CERN researchers working on model checking tools for PLCs claim that “different compositional verification approaches have been analyzed but no generic method has been found yet to be applied to PLC projects”.

V. Case study

TVO is a power company operating three nuclear reactors in Olkiluoto, on the western coast of Finland. Olkiluoto units 1 and 2 are of the Boiling Water Reactor (BWR) type. Unit 1 was commissioned in 1979, and unit 2 in 1982.

Due to the aging of the analog I&C systems, TVO has started an I&C modernization program. The program includes a project called DIMA, where several Olkiluoto 1 and 2 I&C systems will be renewed using mostly software-based technology, with some hardwired parts.

Before the DIMA project proceeds to detailed I&C system design, the functional architecture of the renewed systems is expressed using technology-neutral and vendor-independent function block diagrams. Since the functional design serves as the input for later design stages, correctness of the diagrams is essential. Accordingly, since 2022, VTT has used model checking to verify the diagrams. VTT has detected issues dealing with spurious actuation, contradictory outputs, and in general, incorrect response to inputs [10].

For my case study, I included functional diagrams from three I&C systems, the Reactor Power Control System (RPCS), the Neutron Flux Measuring System (NFM), and the Turbine (pressure) Control System (TCS).

In the descriptions below, I have renamed the functions and the variables to remove any references to actual identifiers, and partially masked and simplified the actual processing logic.

A. Experimental design

For the case study logics to be representative of typical nuclear safety I&C functions, my aim was to include functions that would have, among them:

1) voting over signals from redundant subsystems (e.g., two-out-of-four Boolean vote, or second-minimum or second-maximum vote over analog signals).
2) design elements often found in logics that have contained design issues (e.g., flip-flop memories, delay blocks, feedback loops) [5].
3) complex calculation over analog data (while avoiding overtly complex and underspecified elements).
4) operator commands (e.g., reset of automatic order, manual actuation command, manual mode selection).

Based on our earlier work on identifying the most common types of properties used in I&C logic verification [23], I also aimed to specify:

- properties dealing with timing and sequences usually written with PSL. OTHELLO supports formulas with nested **then** expressions.
- properties addressing spurious actuation, e.g., $\mathbf{G}(act \rightarrow \mathbf{O}\, criterion)$. OTHELLO provides the **P (in the past)** operator.

From the DIMA systems, I found that RPCS, NFM and TCS contained the right type of logics, suitable decomposition into functions, and sufficient complexity.

From RPCS, I modeled a chain of 19 functions related to calculating the main circulation pump setpoint. Some of the functions and their logical connections are shown in Fig. 2.

From NFM, I modeled a chain of nine functions related to power range monitoring (see Fig. 3), and four functions related to detector driving logic.

For TCS, I modeled a chain of nine functions related to turbine generator control and control mode selection.

My experiments consisted of the following work steps:

1) I modeled the individual functions (the subcomponents of the model), and then the whole monolithic model with nuXvm.

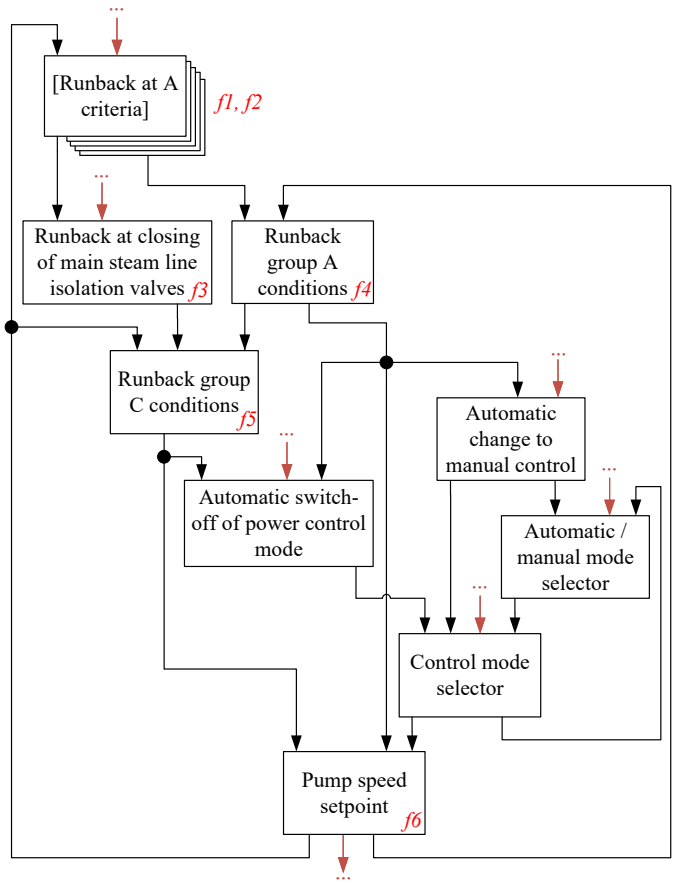

Fig. 2. Partial view of the functional decomposition of the RPCS

2) I formalized LTL properties for the model as a whole, and for each subcomponent.
3) I verified the properties with nuXmv. If I was not able check the LTL properties using the monolithic model, I then checked the properties for individual functions using separate subcomponent-level SMV files.
4) I wrote the system architecture in OSS.
5) I specified the contracts for each subcomponent, and the contract refinement for the system as a whole, based on the LTL properties.
6) I verified the contract refinement with OCRA.

If OCRA could not check the refinement in eight hours, I terminated the process.

For each contract, if needed, I also deliberately introduced an error, to record the time OCRA took to generate the counterexample.

As an example, one requirement for the RPCS is that if the power based on neutron flux exceeds a limit of 108%, RPCS shall initiate a shutdown by driving the main circulation pump setpoint ($desired\_speed$) down according to the "c" ramp value (or faster). The logic is based on two-out-of-four voting over four redundant measurements. Because of the feedback loop between functions $f6$ and $f4$ (see Fig. 2), I need to specify that the neutron flux criterion has held for two consecutive processing cycles, resulting in the contract:

```
COMPONENT dima_rpcs system
...
 CONTRACT c_runback
   assume: true;
   guarantee: always (
        (((count((nf_1>108.0),(nf_2>108.0),
        (nf_3>108.0), (nf_4>108.0))>=2)
        and then
        (count((nf_1>108.0),(nf_2>108.0),
        (nf_3>108.0),(nf_4>108.0))>=2)
        implies then
        (desired_speed<=c_ramp_value))));
```

I then specified the contracts for each individual function (see, again, Fig. 2). Due to the above-mentioned feedback loop, I need to refer to the variable $last\_ramp\_value$ from the previous cycle, and use the **then** operator.

```
COMPONENT f3
...
 CONTRACT r_c_runback
    assume: true;
    guarantee: always ((count((nf_1>108.0),
       (nf2>108.0),(nf_3>108.0),(nf_4>108.0))
       >=2) implies c1_activated);
...
COMPONENT f4
...
 CONTRACT r_c_runback
   assume: true;
   guarantee: always (pump_speed_setpoint
        <=pump_speed_c_setpoint);
...
COMPONENT f5
...
 CONTRACT r_c_runback_1
   assume: true;
   guarantee: always (c1_activated
       implies c_runback_on);
 CONTRACT r_c_runback_2
   assume: true;
   guarantee: always (c1_activated implies
       (c_setpoint=ramp_value));
 CONTRACT c_r_runback_3
   assume: true;
   guarantee: always ((c_setpoint=ramp_value)
       implies then (pump_speed_c_setpoint=
       last_ramp_value));
...
COMPONENT f6
...
 CONTRACT r_c_runback
   assume: true;
   guarantee: always (c_runback_on implies
        (desired_speed=pump_speed_setpoint));
```

I then specified the refinement of contract $c\_runback$ by the contracts of the subcomponents (functions $f3$ to $f6$):

```
REFINEMENT
...
  CONTRACT c_runback REFINEDBY F3.r_c_runback,
          F4.r_c_runback, F5.r_c_runback_1,
          F5.r_c_runback_2, F5.r_c_runback_3,
          F6.r_c_runback;
```

For this contract, OCRA takes just a few seconds to verify the refinement.

I performed the experiments on a 64-bit Windows 11 machine, running on an Intel Core i5-1235U with a clock rate of 1.3 GHz. I used OCRA version 2.1.0, released in July 2021.

Due to the high confidentiality of the design information I was working on, I am unfortunately not able to publish my nuXmv files. Further examples of the contracts I specified, as well as the simplified OSS input files can be found in the online annex[5] to this paper.

B. Experimental results

When constructing the monolithic models, I was able to reach a level of complexity where verification with IC3, BMC or SBMC algorithms of nuXmv took too long to be practically feasible. With certain LTL properties for TCS, I got an "out of memory" error with IC3. With each model, I also needed to rewrite some LTL properties as invariants for getting results fast. While such complexity was not required to experiment with contract refinement, it created a realistic setting for the exercise.

In terms of performance, OCRA for the most part either provided a proof or a counterexample immediately (in fractions of a second). There were two exceptions:

1) For a TCS contract using nested **then** operators, OCRA took 6.2 seconds to verify the refinement. (See contract $high\_p\_ctrl$ in the online annex.)
2) For a certain NFM contract, OCRA did not finish computation in eight hours, so I terminated the process. (When I deliberately added an error to the contract refinement, OCRA produced the counterexample in 75 seconds.)

The contract refinement that OCRA failed to verify is given below. One requirement for NFM is that if the mean value of connected measurements (that are first limited between 0.0 and 100.0) multiplied by amp_factor exceeds 25.0, while the signal $order$ is also on, a certain $action$ must be taken. Simplified to just three measurements, the contract and its refinement read:

```
COMPONENT dima_nfm system
...
 CONTRACT c1
   assume: true;
   guarantee: always ((((amp_factor * (
     (connected_1 ?
       max(0.0,min(100.0,value_1)) : 0.0) +
     (connected_2 ?
       max(0.0,min(100.0,value_2)) : 0.0) +
     (connected_3 ?
       max(0.0,min(100.0,value_3)) : 0.0)
     / count(connected_1,connected_2,
     connected_3))) > 25.0) and order)
     implies action);
```

[5]https://doi.org/10.5281/zenodo.10867637

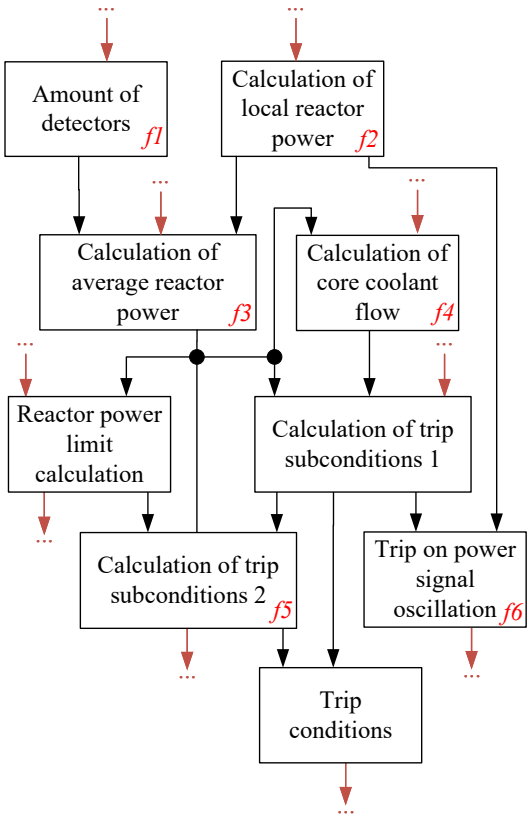

Fig. 3. Partial view of the functional decomposition of the NFM power range monitoring logic

```
REFINEMENT
...
  CONTRACT c1 REFINEDBY F1.r_c1,F2.r_c1_1,
         F2.r_c1_2,F2.r_c1_3,F3.r_c1,F5.r_c1;
...
COMPONENT f1
...
 CONTRACT r_c1
  assume: true;
  guarantee: always (num_connected=count(
     connected_1, connected_2, connected_3));
...
COMPONENT f2
...
 CONTRACT r_c1_1
   assume: true;
   guarantee: always (k1 = (connected_1 ?
     max(0.0,min(100.0,value_1)) : 0.0));
 CONTRACT r_c1_2
   assume: true;
   guarantee: always (k2 = (connected_2 ?
     max(0.0,min(100.0,value_2)) : 0.0));
 CONTRACT r_c1_3
   assume: true;
   guarantee: always (k3 = (connected_3 ?
     max(0.0,min(100.0,value_3)) : 0.0));
```

```
COMPONENT f3
...
 CONTRACT r_c1
   assume: true;
   guarantee: always (average=(amp_factor*
     ((k1+k2+k3)/num_connected)));
...
COMPONENT f5
...
 CONTRACT r_c1
  assume: true;
  guarantee: always (((average > 25.0) and
     order) implies action);
```

When I modified $c1$ to exclude both the division and the max-min operators, OCRA verified the contract refinement in seconds. Including either of the operators causes the analysis time to exceed my limit of eight hours. The issue is therefore likely caused by the combination of nonlinear mathematical functions in the contracts.

Also, when I masked the examples for publication, I accidentally changed the order of the parenthesis in $c1$. Even though the contract was mathematically equivalent, OCRA now produced a counterexample in 97 seconds. The counterexample features a division by zero, and if I add an assumption that prevents the divisor from being zero, the computation time again exceeds my limit. I also noticed that nuXmv fails to prove the property $\mathbf{G}((x * (y/z)) = ((x * y)/z))$ (for *real* variables x, y, z) for a simple test model in reasonable time.

The reader should note that the OSS files in the online annex are simplified and partial versions of the original OSS files I recorder the analysis times with.

As a useful side effect, the case study revealed issues in the DIMA logics not detected earlier: (1) rapid fluctuation of control parameters in three I&C functions, (2) division by zero in one function, and (3) ambiguity of processing order in certain feedback loops spanning several functions.

VI. Discussion

As mentioned above, I aimed to include the most common types of properties from our practical work, to see if they can all be decomposed into contracts. I have collected LTL, CTL and PSL properties written by VTT analysts in practical industrial I&C model checking projects between 2014 and 2022. The data set [41] contains 3923 properties (2984 in LTL, 561 in PSL, and 378 in CTL). The types of properties that occur more than ten times are listed in Table I.

The fourth most common property type—$\mathbf{AG\,EF}\,p$—is a CTL property that cannot be expressed in Othello and decomposed in OCRA. This is not a limitation specific to OCRA, as compositional verification for full CTL (containing existential path quantifiers) is a known hard problem [42]. The property pattern has revealed four real-world design issues, and also serves as a quick check to verify that a certain variable p cannot get stuck (permanently frozen) to its false value.

As PSL's SERE style is just syntactic sugar for LTL, the PSL property types in Table I can be rewritten in Othello with nested **then** operators.

TABLE I
RECURRING TYPES OF PROPERTIES IN VTT'S CUSTOMER PROJECTS

Generalized property type	%	Expressible in OSS
$\mathbf{G}\, p$	52	yes
$\mathbf{G}(p \rightarrow \mathbf{X}\, q)$	11	yes
always {*SERE*} |-> {*SERE*}!	11	yes[a]
$\mathbf{AG}\,\mathbf{EF}\, p$	9.5	no
never {*SERE*}	3.2	yes[a]
$\mathbf{G}(p \rightarrow \mathbf{O}\, q)$	2.7	yes[b]
$\mathbf{G}\, e \rightarrow \mathbf{G}\, p$	1.7	yes[c]
$\mathbf{G}(p \wedge \mathbf{X}\, q \rightarrow \mathbf{X}\,\mathbf{X}\, r)$	1.0	yes
$\mathbf{G}(p \rightarrow \mathbf{F}\, q)$	0.8	yes
$\mathbf{G}(p \wedge \mathbf{X}\, q \rightarrow r)$	0.6	yes
$\mathbf{G}(p \rightarrow \mathbf{Y}\, q)$	0.5	yes
always {*SERE*} |=> {*SERE*}!	0.5	yes[a]
$\mathbf{G}(p \wedge \mathbf{X}\, q \rightarrow r \wedge \mathbf{X}\, s)$	0.3	yes
$\mathbf{G}(p \rightarrow (\neg(\mathbf{Y}\, p \wedge \neg p)\, \mathbf{S}\, q))$	0.3	yes
$i \wedge \mathbf{G}\, e \rightarrow \mathbf{G}\, p$	0.3	yes[c]
$\mathbf{G}\, e \rightarrow \mathbf{G}(p \rightarrow \mathbf{X}\, q)$	0.3	yes[c]
Other	4.7	

[a] using nested **then**
[b] using **in the past**
[c] can be phrased using assumptions

With the patterns $\mathbf{G}\, e \rightarrow \mathbf{G}\, p$ and $\mathbf{G}\, e \rightarrow \mathbf{G}(p \rightarrow \mathbf{X}\, q)$,the analyst has apparently aimed to either filter out irrelevant counterexamples or state some assumptions about the environment. In the OSS contracts, we can state:

```
assume: always e;
```

I give an example for NFM in the online annex.

In general, I aim to avoid assumptions in the contracts, for the same reasons we favor open-loop over closed-loop models [3]. Assumptions about the controlled plant processes or the operators' actions may help in filtering irrelevant execution paths. However, there is then a risk that a realistic execution path relevant to safety is accidentally masked.

The issues OCRA has with nonlinear mathematical formulas could be circumvented by adding monitor variables (with their state defined by said formulas), and referring to their values in the contracts. This approach is not however feasible if the nonlinear computing logic is distributed across subcomponents, rather than mostly contained within one I&C function.

OCRA is under continuous development. Significant improvements were included in the 2021 release version 2.1.0 (support for the timed domain) and the 2020 release version 2.0.0 (e.g., iterator operators like *count*, which is useful in expressing properties for voting logics—see my case study examples).

A practical inconvenience is that OCRA can only be operated from the command line, and it does not support batch interaction, i.e., loading the commands from a separate file (like, e.g., nuXmv does). License is by default only granted for non-commercial use.

VII. CONCLUSION

During a long run of nuclear industry research projects with Aalto University, one of our objectives has always been to experiment with a range of different formal verification tools. VTT's commercial projects in the Finnish nuclear industry have provided us with industrial data and real case studies. The key conclusion of the work presented in this paper is that OCRA is a practically applicable tool that nuclear industry I&C engineering projects would benefit from. In fact, during the case study, I realized that the kind of refinement OCRA can verify is the kind of decomposition of properties that I have sometimes had to manually resort to in practical projects. Having proper tool support for such reasoning would obviously be crucial when working on safety critical applications.

CTL notwithstanding, I was able to decompose all the property types commonly used in our practical projects into contracts, and verify the contract refinement. In one case, though, the nonlinear mathematical formula proved too difficult for OCRA to process, and changing the order of parentheses in the contract (to a mathematically equivalent formula according to the associative law) led to different results.

Nuclear industry I&C systems have certain domain-specific features, but concepts like redundant subsystems and voting over control actions are also used in other domains where fault-tolerance is required (e.g., aviation). Also, the types of functional requirements (and therefore, formal properties) we work with are seldom specific to NPPs, and often apply universally to process industry (safety) systems.

ACKNOWLEDGMENT

This work has been funded by the Finnish National Nuclear Safety and Waste Management Research Programme 2023-2028 (SAFER2028). The case study was provided and co-funded by TVO.

REFERENCES

[1] IAEA, "Technical challenges in the application and licensing of digital instrumentation and control systems in nuclear power plants," International Atomic Energy Agency, Nuclear Energy Series NP-T-1.13, 2015. [Online]. Available: http://www-pub.iaea.org/MTCD/Publications/PDF/P1695_web.pdf

[2] Bel V, BfE, CNSC, CSN, ISTec, KAERI, KINS, NSC, ONR, SSM, STUK, "Licensing of safety critical software for nuclear regulators, common position of international nuclear regulators and authorised technical support organisation," MDEP, Common position Revision 2018, 2018. [Online]. Available: http://www.onr.org.uk/software.pdf

[3] A. Pakonen, I. Buzhinsky, and K. Björkman, "Model checking reveals design issues leading to spurious actuation of nuclear instrumentation and control systems," *Reliab. Eng. Syst.*, vol. 205, p. 107237, 2021.

[4] Areva NP, "U.S. EPR application documents," Areva NP, Final Safety Analysis Report, 2013. [Online]. Available: https://www.nrc.gov/reactors/new-reactors/design-cert/epr/reports.html

[5] A. Pakonen, "Oops! Examples of I&C design issues detected with model checking," in *Proc. ISOFIC*, November 2021. [Online]. Available: https://cris.vtt.fi/files/53941549/Pakonen_ISOFIC_2021_.pdf

[6] E. Clarke, O. Grumber, and D. Peled, *Model checking*, 2nd ed. Cambridge, Massachusetts, US: MIT press, 2001.

[7] D. Giannakopoulou, K. S. Namjoshi, and C. S. Pasareanu, "Compositional reasoning," in *Handbook of Model Checking*, E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds. Cham, Switzerland: Springer, 2018, p. 345–383.

[8] A. Cimatti, M. Dorigatti, and S. Tonetta, "OCRA: A tool for checking the refinement of temporal contracts," in *Proc. ASE*, Nov. 2013, pp. 702–705.

[9] 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*, ser. LNCS, vol. 8559. Springer, 2014, pp. 334–342.

[10] A. Pakonen, "Model-checking I&C logics – practical examples," in *Proc. NPIC & HMIT*, July 2023, pp. 1610–1619.

[11] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV version 2: An opensource tool for symbolic model checking," in *Proc. CAV*, ser. LNCS, vol. 2404. Springer, 2002.

[12] 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.

[13] E. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Form. Methods Syst. Des.*, vol. 19, no. 1, pp. 7–34, Jul 2001.

[14] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Satisfiability*. IOS Press, 2009, pp. 825–885.

[15] A. Cimatti and A. Griggio, "Software model checking via IC3," in *Proc. CAV*, ser. LNCS, vol. 7358. Springer, 2012, pp. 277–293.

[16] A. Biere, K. Heljanko, T. A. Junttila, T. Latvala, and V. Schuppan, "Linear encodings of bounded LTL model checking," *Log. Methods Comput. Sci.*, vol. 2, no. 5, 2006.

[17] G. J. Holzmann, "The model checker SPIN," *IEEE Trans. Softw. Eng.*, vol. 23, no. 5, pp. 279–295, 1997.

[18] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on Uppaal," in *Proc. SFM-RT*, ser. LNCS, vol. 3185. Springer, 2004, pp. 200–236.

[19] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. CAV*, ser. LNCS, vol. 6806. Springer, 2011, pp. 585–591.

[20] M. Benedetti and A. Cimatti, "Bounded model checking for past LTL," in *Proc. TACAS*, ser. LNCS, vol. 2619. Springer, 2003, pp. 18–33.

[21] J. C. Campos, J. Machado, and E. Seabra, "Property patterns for the formal verification of automated production systems," *IFAC Proc. Vol.*, vol. 41, no. 2, p. 5107–5112, 2008.

[22] C. Eisner and D. Fisman, *A Practical Introduction to PSL*. Springer US, 2006.

[23] A. Pakonen, C. Pang, I. Buzhinsky, and V. Vyatkin, "User-friendly formal specification languages – conclusions drawn from industrial experience on model checking," in *Proc. ETFA*, Sept. 2016.

[24] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, and K. G. Larsen, "Contracts for systems design: Theory," INRIA, Tech. Rep. 8759, 2015.

[25] A. Cimatti, M. Dorigatti, and S. Tonetta, "OCRA: Othello Contracts Refinement Analysis, version 2.1.0," Fondazione Bruno Kessler, Tech. Rep., 2021. [Online]. Available: https://ocra.fbk.eu/download/OCRA_Language_User_Guide.pdf

[26] A. Cimatti, M. Roveri, A. Susi, and S. Tonetta, "Verification of requirements for hybrid systems: a formal approach," *ACM Trans. Softw. Eng. Methodol.*, vol. 21, no. 4, pp. 22:1–22:34, 2012.

[27] A. Pakonen, "Model-checking infinite-state nuclear safety I&C systems with nuXmv," in *Proc. INDIN*, July 2021.

[28] A. Pakonen and J. Turunen, "Using model checking for interlocking software verification," *SIGNAL+DRAHT*, vol. 115, no. 9, pp. 41–47, 2023.

[29] I. Buzhinsky and A. Pakonen, "Symmetry breaking in model checking of fault-tolerant nuclear instrumentation and control systems," *IEEE Access*, vol. 8, pp. 197 684–197 694, 2020.

[30] A. Wakankar, A. Kabra, A. Bhattacharjee, and G. Karmakar, "Architectural model driven dependability analysis of computer based safety system in nuclear power plant," *Nucl. Eng. Technol.*, vol. 51, no. 2, pp. 463–478, 2019.

[31] T. Ausberger, K. Kubíček, P. Medvecová, and J. Wolf, "Verification of a safety-related I&C system for nuclear power plant by model checking, test case generation and automatic testing," in *Proc. ETFA*, Sept. 2022.

[32] I. Lopez-Miguel, J.-C. Tournier, and B. Fernandez, "PLCverif: Status of a formal verification tool for programmable logic controller," in *Proc. ICALEPCS 2021*, Oct. 2021.

[33] O. Pavlovic and H.-D. Ehrich, "Model checking PLC software written in function block diagram," in *Proc. ICST*, Paris, France, 2010, p. 439–448.

[34] C. Limbrée, Q. Cappart, C. Pecheur, and S. Tonetta, "Verification of railway interlocking - compositional approach with OCRA," in *Proc. RSSRail*, ser. LNCS, vol. 9707. Springer, 2016, pp. 134–149.

[35] A. E. Haxthausen and A. Fantechi, "Compositional verification of railway interlocking systems," *Systems. Form. Asp. Comput.*, vol. 35, no. 1, pp. 4:1–4:46, 2023.

[36] B. Beckert, M. Ulbrich, B. Vogel-Heuser, and A. Weigl, "Regression verification for programmable logic controller software," in *Proc. ICFEM*, ser. LNCS, vol. 9407. Springer, 2015, pp. 234–251.

[37] M. Bozzano, H. Bruintjes, A. Cimatti, J.-P. Katoen, T. Noll, and S. Tonetta, "COMPASS 3.0," in *Proc. TACAS*, ser. LNCS, vol. 11427. Springer, 2019, pp. 379–385.

[38] J. Xie, W. Tan, Z. Yang, S. Li, L. Xing, and Z. Huang, "SysML-based compositional verification and safety analysis for safety-critical cyber-physical systems," *Connect. Sci.*, vol. 34, no. 1, pp. 911–941, 2021.

[39] R. C. Cardoso, L. A. Dennis, M. Farrel, M. Fisher, and M. Luckcuck, "Towards compositional verification for modular robotic systems," in *Proc. FMAS*, ser. EPCTS, vol. 329. Open Publishing Association, 2020, pp. 15–22.

[40] D. Phan, J. Yang, M. Clark, R. Grosu, J. Schierman, S. Smolka, and S. Stoller, "A component-based simplex architecture for high-assurance cyber-physical systems," in *Proc. ACSD*, June 2017, pp. 50–58.

[41] A. Pakonen, I. Buzhinsky, and V. Vyatkin, "Evaluation of visual property specification languages based on practical model-checking experience," *J. Syst. Softw.*, 2024, in press.

[42] D. E. Long, "Model checking, abstraction, and compositional verification," Ph.D. dissertation, CMU, July 1993.