# Model-checking infinite-state nuclear safety I&C systems with nuXmv

Pakonen, Antti

*Published in:*
2021 IEEE 19th International Conference on Industrial Informatics (INDIN)

*DOI:*
10.1109/INDIN45523.2021.9557445

Published: 23/07/2021

*Document Version*
Peer reviewed version

# Model-checking infinite-state nuclear safety I&C systems with nuXmv

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

***Abstract*—For over a decade, model checking has been successfully used to formally verify the instrumentation and control (I&C) logic design in Finnish nuclear power plant projects. One of the practical challenges is that the model checker NuSMV forces the user to abstract the way analog signals are processed in the model, which causes extra manual work, and could mask actual design issues. In this paper, we experiment with the newer tool nuXmv, which supports infinite-state modelling. Using actual models from practical industrial projects, we show that after changing the analog signal processing to be based on real number math, the analysis times are still manageable. The disadvantage is that certain useful types of formal properties are not supported by the infinite-state algorithms. We also discuss the nuclear industry specific features of I&C programming languages, which cause significant constraints on domain-specific formal verification method and tool development.**

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

## I. Introduction

Modern digital instrumentation and control (I&C) system platforms have made it deceptively easy to design control logics that are very complex. Digital I&C has become the norm, safety critical applications included—even nuclear reactors are kept safe using software based control [1]. In such applications, simplicity is a requirement [2], but even the most critical safety systems can still contain complex analog signal processing [3].

Model checking [4] is a powerful formal verification method. The result is either a logical proof that a stated property holds, or a counterexample trace depicting unwanted system model behaviour. Fondazione Bruno Kessler (FBK) has released a popular, free, open-source model checker called *NuSMV* [5]. In Finland, NuSMV has been used for over a decade to verify I&C application logic design in nuclear power plant (NPP) projects [6].

When using formal verification in safety critical applications, it is important to recognise the limitations of the descriptions, methods, and tools in use [2]. A key limitation of NuSMV is that the model is described in discrete, finite-state terms. In practice, this means that analog signals have to be discretized as integer variables, and only basic math operations are supported. The analyst has to carefully abstract the analog logics [7], increasing the need for time-consuming manual work. Although outright errors in the abstractions are often revealed through “spurious” counterexamples, it is also possible that actual design issues are accidentally masked.

FBK has also released *nuXmv* [8], a model checker enabling analysis of infinite-state models. Among other improvements over NuSMV, nuXmv supports real number variables, and several math operators. But does the improved functionality come at a significant computational cost?

In this paper, we present our experiments on using the infinite-domain algorithms of nuXmv to verify nuclear I&C application logics that contain analog signal processing. The contribution is threefold. First, we concretise the challenges in using finite-state model checkers in our domain. Second, we show that nuXmv is in practice effective in extending the analysis to infinite-state models. Third, we discuss the limitations nuXmv still imposes on the verification process.

In Section II, we cover the basics of model checking. In Section III, we discuss the nuclear industry specific constraints on formal I&C logic verification. Related research is summarised in Section IV. Section V describes our case study. We then discuss our results in Section VI, and present our conclusion in Section VII.

## II. Model checking

### A. Model checking algorithms

Model checking [4] is a computer-assisted method to formally verify whether a system model satisfies stated properties. A tool called a model checker checks if a desired property holds for all the system model execution paths, through exhaustive exploration of all the reachable states. A model execution path that violates the property, if found, is returned to the user as a counterexample.

The key challenge is to avoid *state space explosion* [4], where the number of reachable states to enumerate becomes enormous, and the analysis will not terminate in reasonable time. Symbolic model checkers like NuSMV [5] try to avoid the issue by using Binary Decision Diagrams (BDD) [9] for canonical representation of Boolean formulae, avoiding explicit-state processing which tools like SPIN [10] rely upon. Boolean satisfiability (SAT) solvers are used for bounded model checking (BMC), where the length of checked state transition sequences is limited to a fixed search depth [11]. In NuSMV, the target system is modeled as a finite state machine (FSM), with synchronous processing of model components over discrete transitions.

To enable infinite-state analysis, nuXmv [8] extends the language of NuSMV with real and unbounded integer numbers, and offers new algorithms based on Satisfiability Modulo

Theory (SMT) [12] or abstraction, and a combination of other techniques. IC3 [13], BMC, and simple bounded model checking (SBMC) [14] are extended to the infinite-state case by, e.g., using SMT instead of SAT [8].

In addition to supporting real number variables, nuXmv adds the basic expressions $pi$, $sin$, $cos$, $tan$, $exp$, and $ln$.

Both NuSMV and nuXmv also support real time specifications, and can compute the length of the shortest or longest path from one set of states to another [15]. For continuous time analysis, UPPAAL [16] is a model checker based on timed automata.

### B. Property specification

For specifying the properties, *temporal logic* languages offer a way to formulate statements over execution paths. Linear Temporal Logic (LTL) [4] uses *temporal operators* such as "Globally" ($\mathbf{G}\,p : p$ is true at ever state of the path), "Finally" ($\mathbf{F}\,p : p$ is true at some future state on the path), and "Until" ($p\,\mathbf{U}\,q : q$ is true at some future state, and at every preceding state on the path, $p$ is true). Some tools also support past LTL operators [17] like "Once" ($\mathbf{O}\,p : p$ is true at some past (or the current) state on the path.)

Computation Tree Logic (CTL) [4] uses path quantifiers $\mathbf{A}$ (all execution paths) and $\mathbf{E}$ (some execution path).

Property Specification Language (PSL) [18] is an extension of LTL and CTL. The Sequential Regular Expressions (SERE) style of PSL is convenient for expressing LTL type properties for multi-cycle behaviour [19]. SERE properties are more human readable than the equivalent nested LTL.

## III. Model-checking nuclear I&C logics

### A. Practical projects in Finland

Since 2008, VTT[1] has used model checking to verify I&C application logic design in practical Finnish nuclear industry projects [6], [20]. The method has been successfully used in a new-build project (Olkiluoto 3), an I&C renewal project (at Loviisa 1&2), and to verify early, functional design (Hanhikivi 1).

By 2020, VTT has identified a total of 66 confirmed design issues, in some cases leading to redesign of the application logic. The likelihood of the discovered issues varies, and the estimated safety significance ranges from negligible to severe [21]. Models created by VTT in the practical projects were used in the experiments described in Section V-A.

To facilitate the practical work, VTT and the power company Fortum have developed a graphical front-end for the NuSMV model checker called MODCHK [6].

### B. Industry-specific challenges

A large share of the related research on I&C logic model checking—particularly if the objective is automatic model generation—is built upon the assumption that the design follows the IEC 61131-3 standard. In the nuclear industry, the assumption is not justified, as major safety I&C system suppliers like Framatome and Rolls-Royce use vendor-specific programming languages.

Reasons for eschewing interoperability go beyond the suppliers' need to protect their intellectual property. The overall I&C architecture of a NPP has to follow the design principle of *defence-in-depth* (DiD) [22], and contain successive layers of protection. The different I&C systems on these *DiD layers* need to be both independent from each other, but also *diverse*, i.e., based on different technologies or design principles. Otherwise, a *common-cause failure* (CCF) could lead to simultaneous loss of several levels of protection. Ways to achieve diversity in I&C logics include using different programming languages, program architectures, runtime environments, coding styles, etc.. [2], [23]. In other words, one should *not* follow a universal standard.

For formal verification, the use of non-standard programming languages, along with the fact that the source code for the elementary function blocks is not necessarily available, means that at least part of the model generation has to be done manually.

Another domain-specific, non-standard feature is signal validity. In nuclear I&C platforms like TELEPERM XS (TXS) and Spinline, each signal in the block diagram is associated with both a value and a *status*. If, for example, the input processing logic detects an invalid sensor input, the associated signal is set to "fault" status. The status is then used in the logic to excluded invalid signals in majority voting, making the overall design tolerant to single failures. Validity has been an important factor in 9% of the design issues VTT has detected.

### C. Analog logic modelling

Nuclear regulatory bodies agree that simplicity is required for safety systems [2]. Still, the NPP I&C systems of even the highest safety class can contain quite complex control algorithms.

The U.S. Nuclear Regulatory Commission has published the Final Safety Analysis Report (FSAR) for the proposed U.S. European Pressurised Reactor (U.S.EPR) online [3]. The report contains detailed information about the Protection System (PS), a safety-classified digital I&C system based on the TXS platform. PS is used to prevent radiological release in accidents by tripping the reactor and supplying emergency cooling.

The high-level function block diagrams for the different PS safety functions can also be found in the FSAR[2]. The basic TXS function block types include complex elements like PID Controller and Second Order Filter (See Figure 1). In the diagrams, complex analog logics are used for calculating trip criteria (e.g., high core power level, low departure from nuclear boiling ratio) and actuator control (e.g., main steam relief train).

Another practical example of complex logic in nuclear safety I&C is shown in [6].

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

[2]The function block diagrams for PS can be found in [3] under Tier 2, Chapter 7, in the files ML13220A735.pdf and ML13220A737.pdf. The basic block types are introduced in the file ML13320A726.pfd.

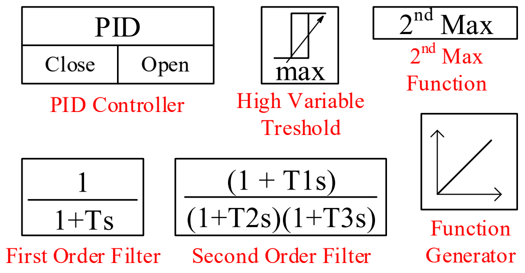

Fig. 1. Examples of non-standard, analog blocks used in the U.S.EPR Protection System [3]

Such systems can only be modelled in discrete-state tools like NuSMV if the analyst carefully abstracts parts of logic [7]. Analog inputs need to be described with integer variables, and the possible values for the variables often need to be limited to avoid state space explosion. Simple math is not a challenge, but a particular problem is the use of analog feedback loops [7]. Storing integer variables to memory seems to significantly increase the analysis time for NuSMV.

## IV. Related Research

As we discussed in Section III-B above, many works on the topic of I&C system logics and model checking rely on the assumption that logic either follows or can be easily transformed to IEC standard 61131-3 or 61499 representations. Examples can be found in, e.g., [24]–[27]. As we pointed out, nuclear I&C logics contain vendor-specific solutions that cannot be easily mapped to those standards.

In addition to the work done in Finland, model checking has also been used in other nuclear I&C applications. In works such as [28] and [29], however, the approach is still based on IEC 61131-3. (In [29], an intermediate IEC 61131-3 model is eventually transformed to a nuXmv model.) In [30], the target application is based on non-standard TXS blocks (including status processing), but after the diagrams were reinterpreted as a Petri net, input constraining was necessary to make the approach feasible. [31] deals with architecture-level analysis of failure rates for I&C controllers using probabilistic model checking.

In [32], nuXmv is used to support regression verification for consecutive versions of a production system's I&C logic design.

In [33], the types of NuSMV models used in our work were extended to timed models in nuXmv, in order to include communication delay between I&C subsystems, and then check real-time specifications. Due to the resulting complexity, IC3 did not compute in reasonable time, and the timed properties could only be checked with BMC. Modelling of I&C hardware failures is discussed in [34].

In our previous work [33], [35], we have found that NuSMV is more applicable for the verification of nuclear I&C logics than, e.g., the explicit-state model checker SPIN, or the real-time model checker Uppaal. Therefore, in this paper, we focus on comparing the performance of NuSMV and nuXmv.

## V. Case Study

### A. Experimental design

To experiment with the scalability of nuXmv's infinite domain algortihms, we have collected eleven NuSMV models constructed by VTT in practical customer projects in the Finnish nuclear industry. These models are taken from three different plant projects, and there are three different companies responsible for the application logic designs. Each model contains analog signal processing that has been significantly abstracted by VTT analysts, in order to verify the logics with NuSMV.

For one of the models—to use it as an example[3]—we have simplified and modified the NuSMV model in order to mask the origin. The masked logic is shown in Figure 2.[4] The original logic consisted of 58 blocks. The function is actuated when the directly measured level—adjusted with a memorized ratio between the measured level and an indirectly calculated value—exceeds the allowed level by 5%. In case the adjusted level differs from the indirectly calculated level by over 1%, the memorized ratio is updated (the CORRECTION signal) after a delay of 10 seconds.

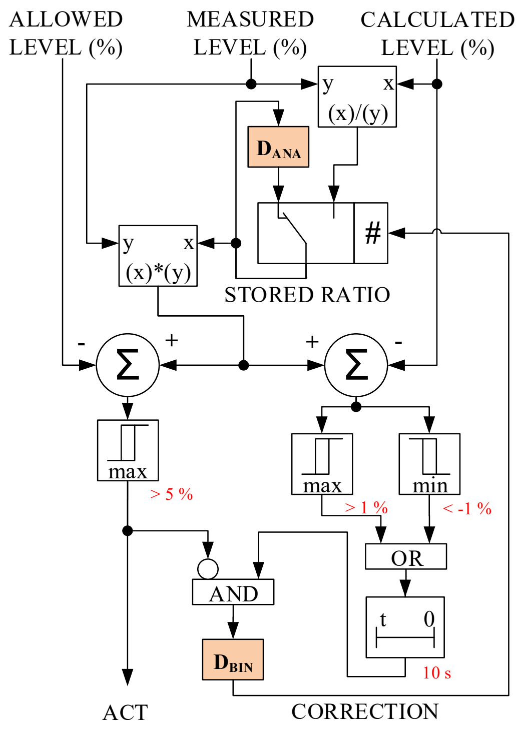

Fig. 2. An exemplar application logic with analog processing

The processing logic of the basic blocks can be found in [3], but we have added the division block, and the $D_{ANA}$ and $D_{BIN}$ cycle delay blocks based on our own invention. For

[3]The finite and infinite-domain input files for our example are available at: http://doi.org/10.5281/zenodo.4316560

[4]In our example, we use the graphical elements of TXS because of the amount of information publicly available in [3]. The reader should not assume that the original logic was therefore based on TXS.

convenience, we explain the logic of the less trivial elements in Figure 3.

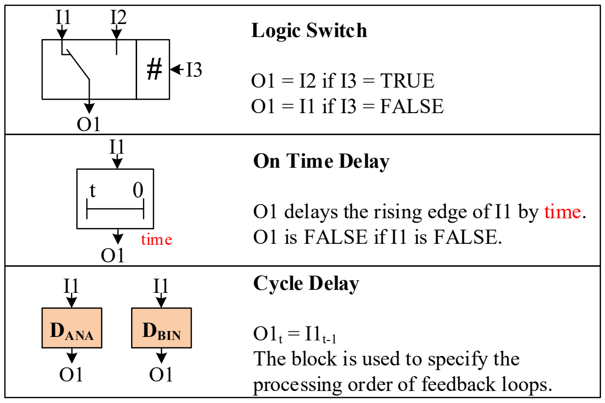
I1
I2
#
I3
O1
Logic Switch
O1 = I2 if I3 = TRUE
O1 = I1 if I3 = FALSE
I1
t
0
time
O1
On Time Delay
O1 delays the rising edge of I1 by time.
O1 is FALSE if I1 is FALSE.
I1
I1
DANA
DBIN
O1
O1
Cycle Delay
O1t = I1t-1
The block is used to specify the processing order of feedback loops.

Fig. 3. Explanation for selected function blocks used in the exemplar logic

(The design issue VTT detected with the original logic has to do with the fact that the $D_{ANA}$ block has a user-configurable initial output value, which had been set to zero. On initialisation, the switch component therefore outputs the value zero for a minimum of ten seconds until the correction can take effect. Assuming that the allowed level and the calculated level are both positive, non-zero numbers, both of the sum blocks will then output a negative value. A negative value cannot be over 5%, so the ACT output is effectively inhibited for the ten seconds after initialisation, no matter how high the measured level is above the allowed level.)

In this example logic, there is only limited benefit on the basic block level from infinite domain modelling—we merely increase the accuracy of the multiplication and division elements. The major benefit has to do with (1) the declaration of the memory variable storing the input value from the last cycle in the $D_{ANA}$ element and (2) the declaration of the analog input variables. The analyst can build the NuSMV input file by declaring:

```
  VAR
       MEASURED_LEVEL: 1..120;
       CALCULATED_LEVEL: 0..120;
...
  MODULE DELAY_ANA(AI1, A1_FAULT, ...)
     VAR
       mem : 0..120;
```

The problem with the above solution is that the analysis times become excessive. The analyst can alleviate the problem by limiting the possible input values, and then—sometimes through trial-and-error—also limit then minimize the possible input values for the *mem* variable, e.g.:

```
  VAR
       MEASURED_LEVEL: 10, 40, 80, 100, 120;
       CALCULATED_LEVEL: 0..120;
...
  MODULE DELAY_ANA(AI1, A1_FAULT, ...)
     VAR
       mem: 0..12;
```

These changes require manual effort, but the effect on analysis times is significant.

When constructing the nuXmv model, the analyst need not worry with any of this, and can simply state:

```
  VAR
       MEASURED_LEVEL: real;
       CALCULATED_LEVEL: real;
...
  MODULE DELAY_ANA(AI1, A1_FAULT, ...)
     VAR
       mem: real;
```

The other ten NuSMV models where similarly modified for nuXmv (although without any of the masking or simplification we did for our public example). The analogue elements found in each logic are listed in Table I. The above example is logic 7 in the table. A masked, simplified version of logic 2 is used as an example in [6].

We rewrote the original PSL properties in LTL for nuXmv. To experiment with the *msat_check_ltlspec_bmc* command, we also rewrote any property using past time temporal operators. (E.g., the property type $\mathbf{G}(\texttt{response} \rightarrow \mathbf{O}\texttt{request})$ we often use in properties addressing spurious actuation [6] was changed to the equivalent [36] $\neg(\neg\texttt{request}\ \mathbf{U}\ (\texttt{response} \wedge \neg\texttt{request}))$.) We omitted the properties originally written in CTL, which are not supported for the infinite domain.

## *B. Experimental results*

For each of the eleven pairs of models, we checked the same temporal properties, and calculated the average model checking time for each property. We focus on the processing time, because it is a tangible measure of computational cost. The number of reachable states, for example, is not a sufficient (or necessary even credible) criterion [6]. All the experiments were done using in an Intel Core i7-6600U CPU with a clock rate of 2.6 GHz. We used NuSMV version 2.6.0 and nuXmv version 2.0.0. Table I shows the results of model checking runs.

For the NuSMV benchmark, we ran BDD-based checks using the commands *check_ltlspec*, and *check_pslspec*.

The nuxmv commands [15] we used where (as named in Table I):

- **IC3** : *check_ltlspec_ic3*
- **BMC** : *msat_check_ltlspec_bmc*
- **SBMC** : *msat_check_ltlspec_sbmc_inc*
- **invar.** : *check_invar_ic3*

Each nuXmv algorithm was performed with bound $k = 30$.

We also experimented with the infinite domain command *msat_check_ltlspec_inc_coi*, but the developers warn that the feature is still experimental, and it failed to prove any property TRUE (although it did successfully detect all the FALSE properties). The algorithm was also the slowest one for nuXmv. For logic 7, in particular, the average analysis time exceeded two hours.

TABLE I
EXPERIMENTAL RESULTS

| | | Average model checking time (s) | | | | |
|---|---|---|---|---|---|---|
| | | | nuXmv | | | |
| # | Analog elements in logic | NuSMV | IC3 | BMC | SBMC | invar. |
| 1 | cycle delay, function[a], limit treshold, multiplication, sum, switch | 31,8 | 0,06 | 3,47 | 0,26 | 0,04 |
| 2 | absolute, cycle delay, deadband, division, limit treshold, memory, multiplication, PI[b], PID[b], sum | 25,9 | 1,32 | 9,58 | 2,96 | 0,13[d] |
| 3 | cycle delay, division, limit treshold, limiter, maximum, memory, minimum, multiplication, switch | 1,79 | **374** | **3080** | **80,0** | 0,07 |
| 4 | limit treshold, 2nd maximum, 2nd minimum | 0,41 | 0,28 | 0,88 | 0,63 | 0,03 |
| 5 | binary adder, cycle delay, linear interpolation, memory, limit treshold, setpoint, switch | 0,39 | 0,10 | 0,04 | 0,05 | 0,05 |
| 6 | cycle delay, limit treshold, minimum, setpoint, sum, switch | 0,20 | 0,15 | **22,9** | 1,41 | 0,03 |
| 7 | cycle delay, division, limit treshold, multiplication, sum, switch | 0,20 | 0,05 | 4,44[c] | 5,91 | 0,03[d] |
| 8 | limit treshold, maximum, sum, 2nd maximum, 2nd minimum | 0,15 | 0,07 | 2,53 | 1,56 | 0,04 |
| 9 | binary adder, division, function[a], minimum, multiplication, sum, switch | 0,13 | 0,04 | 0,34 | 0,22 | 0,02 |
| 10 | function[a], limit treshold, sum | 0,08 | 0,13 | 0,39[c] | 0,18 | 0,04[d] |
| 11 | cycle delay, limit treshold, limiter, linear interpolation, minimum, sum | 0,05 | 0,13 | 8,41 | 0,71 | 0,05[d] |

[a] Custom algorithm for calculating plant-specific process variable(s)
[b] The detailed algorithm was not supplied by the designer. The block was abstracted in the model.
[c] Some properties were modified to replace temporal past operators.
[d] Some properties could not be expressed as invariants.

## VI. DISCUSSION

### A. Performance of nuXmv

Based on our experiments, of the nuXmv infinite domain algorithms, IC3 seems to perform well. A notable exception is logic 3 in Table I, but that particular logic contains several feedback loops, as well as other elements that store real values to memory. In our practical work with NuSMV, we have generally noted that memorisation of analog values, in particular, increases the analysis time. For logic 6, the setpoint function block was modelled far more accurately for nuXmv, contributing significantly to the complexity of the infinite-domain model. Still, IC3 invariant checking performed well for both logics 3 and 6.

IC3 invariant checking was the overall fastest method. A key practical challenge is that nuXmv only supports one temporal operator in invariants, $next$, and does not allow nesting (e.g., $next(next\, p)$). This is a significant limitation—in three of our 11 examples, a real design issue was revealed by a property that could not be expressed as an invariant.

The BMC and SBMC algorithms also performed well, and were is some cases just as fast as IC3 to detect a FALSE property. However, for the TRUE properties, with bound $k = 30$, the analysis times were consistently longer. Limiting the search depth (bound) makes BMC terminate faster, but decreases the reliability.

The nuXmv developers point out that when the domain is infinite, the verification problem is *in general* undecidable, and IC3 may fail in proving a property [15]. However, in our experiments, the IC3 algorithm was always able to determine whether the property was TRUE or FALSE.

### B. Implications for practical projects

Between NuSMV and nuXmv, the verification results did not change. That is, the properties that we verified produced the same outcome in both the finite-domain and the infinite-domain case. On interpretation could be that the conservative assumptions VTT has made in abstracting the math for NuSMV are reasonable.

In any case, it is clear that infinite-domain case makes it easier for the analyst to model the logics that contain analog signal processing, by reducing the need for abstractions.

Furthermore, it is most probable that the BDD-based, finite-domain algorithms of nuXmv are more efficient than the ones used in the older NuSMV.

One drawback of nuXmv's infinite-domain algorithms is that CTL is not supported. Some properties that have proven useful cannot be formulated in LTL. E.g., $\mathbf{AG\,EF}\, p$ has in practice revealed two real design issues where an output was permanently stuck to some value ($\neg p$).

A minor inconvenience is the lack of support for PSL. Although the SERE style of PSL is merely syntactic sugar for LTL, it is particularly convenient in expressing sequences of states. Formal properties for process industry I&C often deal with complex sequences and timing [19].

## VII. CONCLUSION

The nuclear industry has justifiable reasons to use closed, non-standard programming languages for safety classified I&C systems. The downside is the need for manual work in building the input files for model checking tools like NuSMV. Our research shows that the infinite-domain algorithms of nuXmv reduce the need for manual effort, particularly related to the simplification and abstraction of analog signal processing, as required by the heuristics of NuSMV.

In our previous work, we have shown that in the practical verification work, the symbolic verification algorithms of NuSVM consistently outperform the explicit-state algorithms of tools like SPIN, and scale better than the real-time algorithms of tools like UPPAAL. Now, it seems that nuXmv, in turn, can outperform NuSMV, while improving coverage with new functionality.

Currently, infinite-domain models cannot be checked against CTL or PSL properties. Lack of support for PSL is an inconvenience, but certain types of CTL properties—proven useful in practice—cannot be expressed in LTL.

## Acknowledgment

This work has been funded by the Finnish Research Programme on Nuclear Power Plant Safety 2019-2022 (SAFIR 2022). We wish to thank Dr. Igor Buzhinsky—formerly of Aalto University—for valuable discussions and comments.

## 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] 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

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

[5] 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 *International Conference on Computer-Aided Verification (CAV 2002)*, ser. LNCS, vol. 2404. Springer, 2002.

[6] A. Pakonen, I. Buzhinsky, and K. Björkman, "Model checking reveals design issues leading to spurious actuation of nuclear instrumentation and control systems," *Reliability Engineering & System Safety*, vol. 205, p. 107237, 2021.

[7] A. Pakonen, J. Valkonen, S. Matinaho, and M. Hartikainen, "Model checking for licensing support in the Finnish nuclear industry," in *International Symposium on Future I&C for Nuclear Power Plants (ISOFIC)*, 2014.

[8] 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 *26th International Conference on Computer Aided Verification (CAV 2014)*, ser. LNCS, vol. 8559. Springer, 2014, pp. 334–342.

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

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

[11] E. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Formal Methods in System Design*, vol. 19, no. 1, pp. 7–34, Jul 2001.

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

[13] A. Cimatti and A. Griggio, "Software model checking via IC3," in *Computer Aided Verification*, ser. LNCS, vol. 7358. Springer, 2012, pp. 277–293.

[14] A. Biere, K. Heljanko, T. A. Junttila, T. Latvala, and V. Schuppan, "Linear encodings of bounded LTL model checking," *Logical Methods in Computer Science*, vol. 2, no. 5, 2006.

[15] M. Bozzano, R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "nuXmv 2.0.0 user manual," Fondazione Bruno Kessler, Tech. Rep., 2019. [Online]. Available: https://es.fbk.eu/tools/nuxmv/downloads/nuxmv-user-manual.pdf

[16] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on Uppaal," in *Formal Methods for the Design of Real-Time Systems*, ser. LNCS, vol. 3185. Springer, 2004, pp. 200–236.

[17] M. Benedetti and A. Cimatti, "Bounded model checking for past LTL," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003)*, ser. LNCS, vol. 2619. Springer, 2003, pp. 18–33.

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

[19] A. Pakonen, C. Pang, I. Buzhinsky, and V. Vyatkin, "User-friendly formal specification languages – conclusions drawn from industrial experience on model checking," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2016)*. IEEE, 2016, pp. 1–8.

[20] A. Pakonen, "Model-checking of I&C logics – insights from over a decade of projects in Finland," in *12th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies (NPIC & HMIT)*. American Nuclear Society, 2021.

[21] A. Helminen and A. Pakonen, "Potential applications of model checking in probabilistic risk assessments," VTT Technical Research Centre of Finland Ltd., VTT Technical Report VTT-R-00017-20, 2020. [Online]. Available: https://cris.vtt.fi/files/27299692/VTT_R_00017_20.pdf

[22] IAEA, "Approaches for overall instrumentation and control architectures of nuclear power plants," International Atomic Energy Agency, Nuclear Energy Series NP-T-2.1, 2018. [Online]. Available: http://www-pub.iaea.org/MTCD/Publications/PDF/PUB1821_web.pdf

[23] U.S.NRC, ORNL, "Diversity strategies for nuclear power plant instrumentation and control systems," U.S.NRC, NUREG-Series Publications NUREG/CR-7007 ORNL/TM-2009/302, 2008. [Online]. Available: https://www.nrc.gov/docs/ML1005/ML100541256.pdf

[24] T. Ovatman, A. Aral, D. Polat, and A. O. Ünver, "An overview of model checking practices on verification of PLC software," *Software & Systems Modeling*, vol. 15, no. 4, pp. 937–960, Oct 2016.

[25] L. Garcia, S. Mitsch, and A. Platzer, "HyPLC: Hybrid programmable logic controller program translation for verification," in *10th ACM/IEEE International Conference on Cyber-Physical Systems*, ser. ICCPS '19, 2019, p. 47–56.

[26] O. Pavlovic and H. Ehrich, "Model checking PLC software written in Function Block Diagram," in *2010 Third International Conference on Software Testing, Verification and Validation*, 2010, pp. 439–448.

[27] Cheng Pang and V. Vyatkin, "Automatic model generation of IEC 61499 function block using net condition/event systems," in *2008 6th IEEE International Conference on Industrial Informatics*, July 2008, pp. 1133–1138.

[28] J. Yoo, S. Cha, and E. Jee, "Verification of PLC programs written in FBD with VIS," *Empirical Softw. Engineering*, vol. 41, no. 1, p. 79–90, 2009.

[29] B. F. Adiego, D. Darvas, E. B. Viñuela, J. Tournier, S. Bliudze, J. O. Blech, and V. M. G. Suárez, "Applying model checking to industrial-sized PLC programs," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 6, pp. 1400–1410, 2015.

[30] E. Németh and T. Bartha, "Formal verification of safety functions by reinterpretation of functional block based specifications," in *Formal Methods for Industrial Critical Systems*, ser. LNCS, vol. 5596. Springer, 2009, pp. 199–214.

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

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

[33] I. Buzhinsky and A. Pakonen, "Timed model checking of fault-tolerant nuclear I&C systems," in *18th IEEE 17th International Conference on Industrial Informatics (INDIN 2020)*, 2020, pp. 159–164.

[34] 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.

[35] I. Buzhinsky, A. Pakonen, and V. Vyatkin, "Explicit-state and symbolic model checking of nuclear I&C systems: A comparison," in *43rd Annual Conference of the IEEE Industrial Electronics Society (IECON 2017)*, 2017, pp. 5439–5446.

[36] M. Pradella, P. San Pietro, P. Spoletini, and A. Morzenti, "Practical model checking of LTL with past," in *International Workshop on Automated Technology for Verification and Analysis (ATVA03)*, 2003.