Model-checking I&C logics — practical examples

Antti Pakonen

Citation:

Pakonen, A. Model-checking I&C logics — practical examples. 13th Nuclear Plant Instrumentation, Control and Human-Machine Interface Technologies (NPIC&HMIT 2023), Knoxville, TN, USA, pp. 1610-1619. ANS, 2023.

DOI: 10.13182/NPICHMIT23-41122

© 2023 The American Nuclear Society, La Grange Park, Illinois.

Model-Checking I&C Logics — Practical Examples

Antti Pakonen*

VTT Technical Research Centre of Finland Ltd., Espoo, Finland

d*oi.org/10.13182/NPICHMIT23-41122*

ABSTRACT

A spurious actuation of an instrumentation and control (I&C) system function is an illustrative example of a "negative" requirement being violated. Verifying such requirements with testing is very hard. Model checking is a formal verification method, aimed at mathematical proof that a (system) model fulfills stated formal properties. Due to the exhaustive coverage, design issues are found in I&C systems already subjected to, e.g., testing. The formal properties can also address the absence of unwanted functionality—spurious signals, contradictory commands, frozen outputs, etc.

In this paper, we discuss the use of model checking the Finnish nuclear industry, where the method has been applied in different plant life-cycle phases. In the Olkiluoto 3 newbuild and Loviisa 1&2 renewal projects, the focus was on detailed logic design. In the Hanhikivi 1 newbuild and Olkiluoto 1&2 I&C renewal projects, we instead verified functional diagrams, developed early in the projects as input for the later detailed design stages.

Through two practical examples of design issues identified in these projects, we demonstrate how easy it is to disprove "negative" requirements having to do with contradictory signals. We also demonstrate how to filter out irrelevant counterexamples, to find out other types of problematic scenarios, even if the first one returned by the model checker can otherwise be ruled out.

Keywords: model checking, formal verification, I&C software

1. INTRODUCTION

Verification methods like testing, simulation, or manual review have inherent limitations—for a system of sufficient complexity, it is not possible to consider every conceivable scenario, and achieve 100% coverage. Techniques like test automation do help, but perfect coverage is just not possible. The risk for hidden errors remains.

Formal verification methods, on the other hand, aim at mathematical proof. If we express functional requirements as formal specifications, we can use computer tools like *model checkers* [1] to prove the correctness of a given implementation with respect to those specifications. Formal specifications also promote unambiguity, and reveal contradictions in the requirements themselves [2].

In the Finnish nuclear industry, VTT has used model checking to verify instrumentation and control (I&C) logic design in new-build and I&C renewal projects since 2008. To date, we have identified a total of 75 confirmed design issues. The probability and the safety significance of the issues varies, but the results have in many cases led to the evaluated systems' redesign. We have also successfully applied model checking in the railway industry, on commission from the Finnish engineering company Mipro.

In this paper, we demonstrate the potential of model checking through two examples, both of them based on real-world design issues we have detected in practical nuclear industry projects. The examples show

*antti.pakonen@vtt.fi

how easy and fast it is to verify the absence of unwanted behavior, specifically, by disproving a "negative" requirement on contradictory signals. We also demonstrate how the analyst can filter out irrelevant execution paths, to come up with more likely counterexamples. As the design errors only come apparent in very unlikely and quite counter-intuitive scenarios, it is easy to understand (1) how they could have been missed in, e.g., testing, and (2) what the real benefits of formal verification are.

Throughout the paper, we discuss the verification of I&C "logics", since we have not only verified (1) I&C application software, but also (2) Field-Programmable Gate Array (FPGA) based designs, and (3) functional architecture diagrams.

2. I&C LOGIC MODEL CHECKING

2.1. Model Checking

In model checking [1], a software tool called a model checker is used to analyze whether a formal model satisfies stated formal properties. In our case, the model is based on the I&C logic, and the formal properties are specified based on functional requirements for the I&C. The result is either a proof that the desired property holds for all the reachable states of the model, or a counterexample describing an execution path that violates the property.

If the analyst makes an outright error in either modeling the system or specifying the formal property, the error will almost certainly * result in a counterexample, allowing the analyst to fix their mistake.

The inherent challenge is to avoid *state space explosion*, a situation where the number of states becomes too enormous for the model checker to enumerate through [1]. Symbolic model checkers (like NuSMV [4] and nuXmv [5]) avoid explicit state enumeration by employing Binary Decision Diagrams (BDD), which allow for a canonical representation of Boolean formulae. Another technique to make the analysis faster (for computationally expensive models) is to limit the length of checked state transition sequences, by using Boolean satisfiability (SAT) solvers to perform *bounded model checking* (BMC).

Of the different model checkers available (discrete or continuous time, finite or infinite-domain, deterministic or probabilistic, etc.) we have found NuSMV and nuXmv most useful for the verification of I&C logics.

2.2. Formal Property Specification

The verified properties are specified using temporal logic languages, most commonly Linear Temporal Logic (LTL) and Computational Tree Logic (CTL) [1]. Temporal logic allows the analyst to formulate statementes over *execution paths*. The properties can usually be categorized as either *liveness properties* (something "good" shall eventually happen) or *safety properties* (something "bad" shall never happen) [6].

In addition to common Boolean operators, LTL and CTL use temporal operators [7], as listed in Table 1.

In some contexts, graphical symbols are also used ($\circ$ for $\mathbf{X}$, $\Box$ for $\mathbf{G}$, and $\diamond$ for $\mathbf{F}$).

CTL adds the *path quantifiers* $\mathbf{A}$ ("for all execution paths") and $\mathbf{E}$ ("for some execution path"). (E.g., $\mathbf{EF}\, p$: "there exists some execution path where p is true at some future state").

Examples on how the properties can address unwanted I&C system behavior include:

1. The absence of spurious actuation (act without $request$) can be proven by verifying the LTL property $\mathbf{G}(act \rightarrow request)$, or, e.g., $\mathbf{G}(act \rightarrow \mathbf{O}\, request)$, if there is an indeterminate delay between $request$ and act [3].
2. The absence of contradictory commands (e.g., simultaneous $start$ and $stop$) can be proven by verifying the LTL property $\mathbf{G}\, \neg(start \wedge stop)$. (See also our examples in Section 4.)
3. The absence of scenarios where the logic is permanently frozen to some state p can be proven by verifying the CTL property $\mathbf{AG\, EF}\, \neg p$.

*It is possible to make an error in both the model *and* the property so that both errors are masked—the erroneous model satisfies the erroneous property. A property can also be accidentally specified so that it always holds (e.g., $\mathbf{G}((p \wedge \neg p) \rightarrow q)$).

Table 1: Temporal logic operators.

Op.	Semantics
$\mathbf{X}\,p$	"neXt": p is true in the next state on the path.
$\mathbf{G}\,p$	"Globally": p is true at every state on the path.
$\mathbf{F}\,p$	"Future": p is at some future state on the path.
$p\,\mathbf{U}\,q$	"Until": q is true at some future state, and at every preceding state on the path p is true.
$\mathbf{Y}\,p$	"Yesterday": p holds in the previous state on the path. ($\mathbf{Y}\,p$ is false in the initial state.)
$\mathbf{Z}\,p$	Otherwise equivalent to $\mathbf{Y}\,p$, but true on in the initial state.
$\mathbf{H}\,p$	"Historically": p is true at every preceding (and current) state on the path.
$\mathbf{O}\,p$	"Once": p is true at at some past (and current) state on the path.
$p\,\mathbf{S}\,q$	"Since": q is true at at some past state, and for every state that has followed, p has been true.

2.3. Related Research

CERN has applied model checking in verification of I&C software, and has also developed and released an open source verification platform PLCverif [8]. PLCverif currently only supports programming languages specific to Siemens products.

Elsewhere, we have not found evidence of established practical use of model checking in the context of nuclear I&C. Below, we mention some recent nuclear domain research efforts.

In [9], the authors describe a verification process (that includes model checking) for function block diagrams, and apply the process to verify functions of a Post Accident Monitoring System.

In [10], the authors used model checkers to prove properties for a FPGA platform designed for nuclear applications, first on the architecture level, and then on the code level.

In [11], the authors used probabilistic model checking to analyze architectural properties (e.g., availability) for a Reactor Trip System and a Engineered Safety Feature Actutation System of a PWR.

In [12], the authors used timed automata to verify safety properties for a digital feed water control system.

3. PRACTICAL INDUSTRY PROJECTS IN FINLAND

There are five nuclear reactors in Finland, in two power plants. The utility Fortum has operated two VVER-440 units in Loviisa for over four decades. The utility TVO has operated two BWR units in Olkiluoto similarly for over four decades, and since 2023, also operates an EPR.

In terms of I&C, there have been four major activities in the last two decades: (1) the Olkiluoto 3 EPR new-build, (2) the Hanhikivi-1 AES-2006 new-build (cancelled in 2022), (3) the I&C renewal projects LARA and ELSA at Loviisa 1 and 2, and (4) the I&C renewal project DIMA at Olkiluoto 1 and 2.

Model checking has been successfully used in all of these projects.

The work was carried out using MODCHK [3], a graphical front-end for NuSMV (see Fig. 1). The work process and our modeling approach are described in [3].

The scope of the projects is summarized in Table 2. (The Finnish Safety Class 2 (SC2) corresponds with Class 1 / Category A, and SC3 with Class 2 / Category B in IEC 61226 [13].)

For the Hanhikivi 1 new-build, we verified several revisions of the functional architecture, where the vendor-independent logic of the safety functions was specified using block diagrams. We detected issues dealing

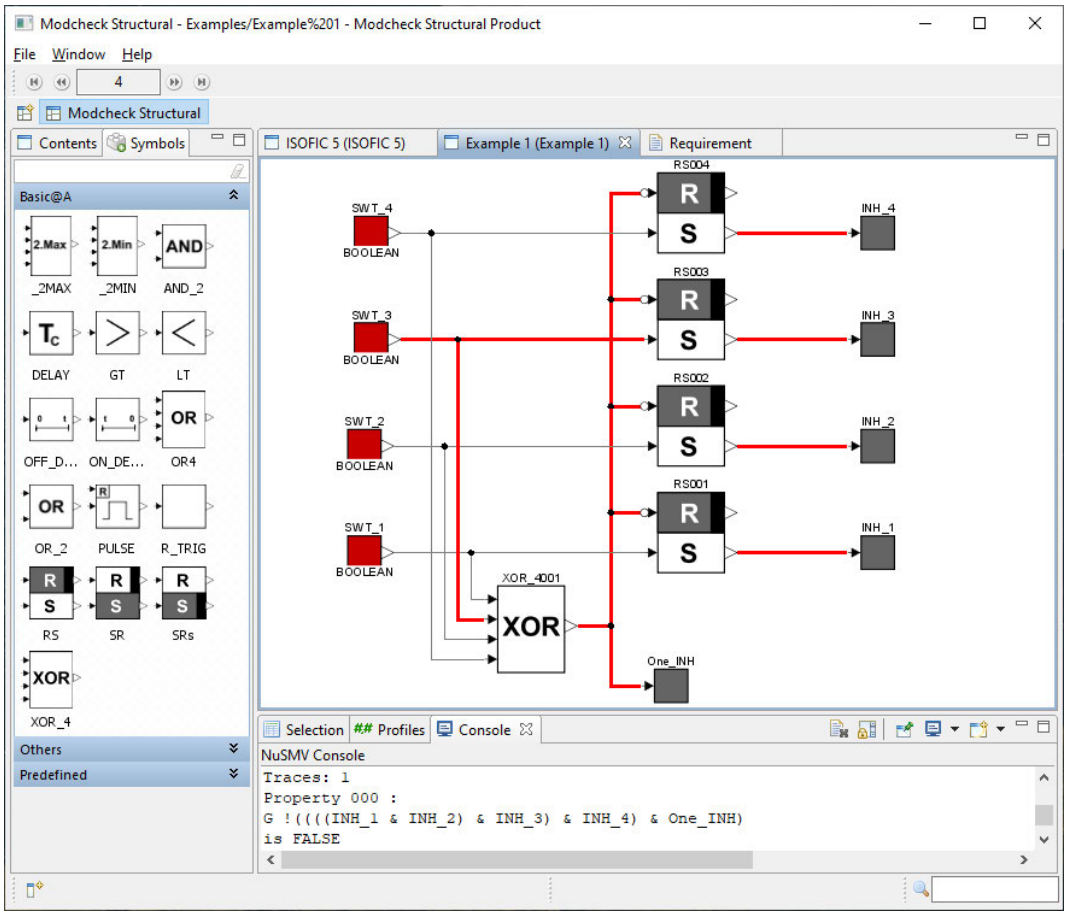

Figure 1: MODCHK is a graphical front-end for NuSMV, providing, e.g., counterexample visualization for function block diagrams.

Table 2: The scope of practical model-checking projects in Finnish nuclear industry.

Project	Project type	Plant type	Verification scope	Safety class
Hanhikivi 1	new-build (cancelled)	AES-2006	Functional architecture	SC2, SC3
Loviisa 1&2	I&C renewal	VVER-440	RTS, RPCS, RPLS, PAIS, PPS, NFS and MBS	SC2, SC3, non-safety
Olkiluoto 1&2	I&C renewal	BWR	Functional architecture	SC2, SC3, non-safety
Olkiluoto 3	new-build	EPR	PS and PACS	SC2

with spurious actuation, contradictory actuation commands, or in general, incorrect response to input. We also reported ambiguity issues in the specifications.

For the Loviisa 1&2 renewal project ELSA, we verified the detailed design of several I&C systems (Reactor

Trip System (RTS), Reactor Power Control System (RPCS), Reactor Power Limitation System (RPLS), Preventive Actuation and Indication System (PAIS), Preventive Protection System (PPS), Neutron Flux Measurement System (NFS)) and the functional design of the Manual Backup System (MBS). We detected issues leading to the redesign of RPCS and RTS logics.

For the Olkiluoto 1&2, we verified the functional architecture, where—similarly to the Hanhikivi 1 project—the vendor-independent logic of the functions was specified using block diagrams. We detected issues dealing with spurious actuation, contradictory actuation commands, or in general, incorrect response to input. We also reported ambiguity issues in the specifications.

For the Olkiluoto 3 new-build, we verified different revisions of the detailed design of the Protection System (PS) and the Priority and Actuator Control System (PACS), of which the PACS is based on FPGA technology. This work was done on commission from the regulator STUK, and the results are confidential.

In all the above-mentioned projects combined, we have detected a total of 75 confirmed design issues. 22 (29 %) of the issues deal with spurious actuation. In 5 (8 %) of the issues, the logic permanently froze to some output state. Other recurring features of the issues include:

1. Improper or ill-timed human actions played a role in 25 (33 %) of the issues.
2. Very exact (millisecond-level) timing played a role in 23 (31 %) of the issues.
3. Uncharacteristic input data played a role in 16 (21 % of the issues).

(By "uncharacteristic input data", we mean scenarios where, in the counterexample, the model variables that represent process measurements are given combinations and/or sequences of values that are not likely if consider the real physical and chemical process of the plant [15]. Such input data could in some cases still be attributed to sensor device malfunction.)

The detected issues have varying safety relevance and probability. In [14], we have analyzed the potential end effects of the issues (detected by 2020), and in Table 3, we list examples of issues with either high or low safety significance. The issues with low impact on safety could still be detrimental to the plant's operation, or even cause damage to equipment. Issues where the I&C fails to the safe direction (e.g., "trip signal cannot be reset", or "safety function is permanently on" [14]) can have a negative impact on plant availability, and therefore cause financial loss.

Based on the scope and the results of these projects, we can draw several conclusions:

1. Model checking reveals design issues in logics already subjected to more conventional verification methods such as testing. We have even found design issues that were confirmed to be possible by manual review, but not practically reproducible in a simulator (due to the exact timing required).
2. Model checking reveals I&C software design issues that could lead to spurious actuation [3].
3. The issues found with model checking often feature unlikely, complicated, and counter-intuitive scenarios, which highlights the difficulty in covering every conceivable scenario in, e.g., testing.
4. It is likely that nuclear I&C systems currently in use that have not been subjected to formal verification contain hidden design errors that model checking would reveal.
5. Model checking is of use in different life cycle phases, and in different design stages—as functional design is used as input for later detailed design, catching errors at an early stage is crucial.
6. Verification of non-safety system functions is also justified, as failures of such systems could lead to production loss or damage to expensive equipment, even if safety would not be directly compromised.
7. Applicability and usefulness of model checking does not depend on the plant type, the technology, the supplier, or the vendor-specific programming or specification language.

Table 3: Descriptions of design issues identified in VTT projects [14].

Issues with high safety relevance	Issues with low safety relevance
Spurious actuation of safety function *may lead to leak of radionuclides*.	Periodic test fails, no effect on plant operation.
Spurious actuation of safety functions *reduces options for controlling accident*.	Plant remains in safe state, exceptional state of I&C logic due to maintenance action.
System remains in non-safe state after initiating event.	Testing logic of preventive functions fails.
Safety function lost due to exceptional state of delay processing in I&C logic.	Short equipment transient due to test, plant remains in safe state.
Component protection failure may lead to *loss of safety function later.*	2-out-of-4 voting reduced to 2-out-of-3 vote.
Safety function can be inhibited on several channels by maintenance action.	Indicative function fails.
...	...

4. PRACTICAL EXAMPLES OF DESIGN ISSUES

In this section, we illustrate the usefulness of model checking by presenting two examples. Both examples are based on an actual design issues VTT detected in the practical industry project we described in Section 3. In both examples, the failing property describes a state that should never occur. To mask the origin, we have simplified the example to include only the blocks needed to reproduce the issue, and we have changed the graphical symbols representing the basic function blocks.

We have previously published fourteen similar examples. The reader can find seven of them, and the links to the rest of them in [15].

Both examples deal with binary logic, but NuSMV can also handle integer variables, and model checking can even be used for infinite-domain verification, where the models can contain real number math [16]. Examples of analog logic issues we have detected are found in [3,15,16].

4.1. Example 1: Control Mode Selection Logic

In our first example, we have a logic for selecting one of two control modes (CTRL_A or CRTL_B) (see Fig. 2). The intended functionality is that the operator can select the operation mode (by the SET_A and SET_B commands). Once a LIMIT is reached, the control mode is automatically switched off. One requirement to verify is that both control modes shall not be activated simultaneously. The analyst specifies:

$$\mathbf{G}\,\neg(\text{CTRL\_A} \wedge \text{CTRL\_B})$$

NuSMV then returns a counterexample, where the model is initialized in a state where both contradictory inputs SET_A and SET_B are set at the initial state. Due to the processing order of the feedback loop, both outputs are then set (see step 2 in Fig. 2) (and remain set until LIMIT is set). The analyst rules such an initial state out by rewriting the property as:

$$\neg(\text{SET\_A} \wedge \text{SET\_B}) \rightarrow \mathbf{G}\,\neg(\text{CTRL\_A} \wedge \text{CTRL\_B})$$

NuSMV then returns a counterexample (see Fig. 2), where both SET_A and SET_B are false at the initial state, but are then activated simultaneously. Again, due to the processing order of the feedback loop, both outputs are set.

If we assume that there exists some external mechanism that prevents the user from switching SET_A and SET_B on at the same exact time, the analyst can again re-write the property to rule such executions out:

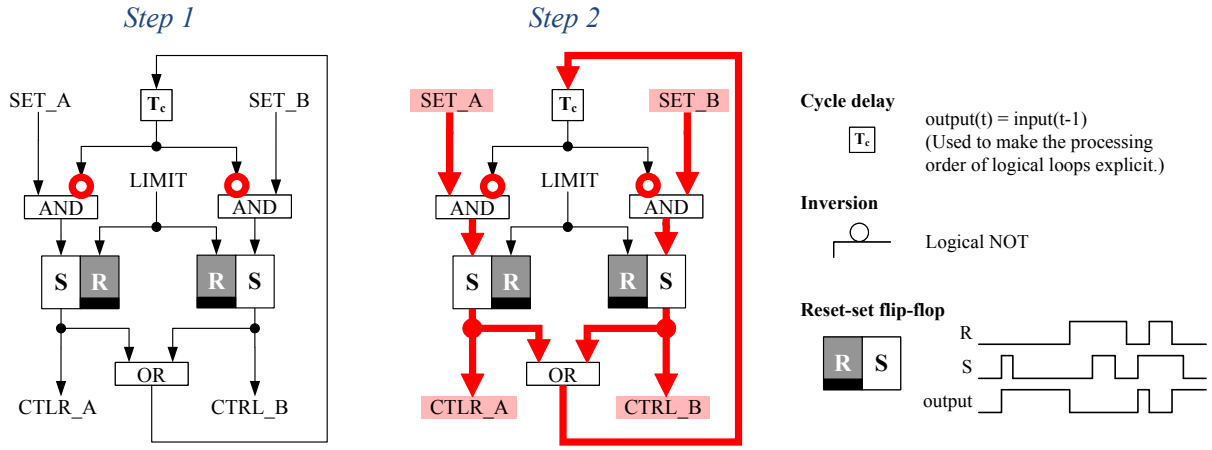

Figure 2: The logic, the second counterexample, and the block key for our Example 1. The first counterexample consists of only one step, which is the same as step 2, here.

$$\neg(\text{SET\_A} \wedge \text{SET\_B}) \wedge \mathbf{G}\,\neg((\neg\text{SET\_A} \wedge \mathbf{X}\,\text{SET\_A}) \wedge (\neg\text{SET\_B} \wedge \mathbf{X}\,\text{SET\_B})) \rightarrow \mathbf{G}\,\neg(\text{CTRL\_A} \wedge \text{CTRL\_B})$$

NuSMV then returns a third counterexample (see Fig. 3), where first, the operator switches SET_A, and CTRL_A is therefore set. Next, LIMIT causes CTLR_A to be reset. If, on the same execution cycle where LIMIT resets, SET_B is also switched on (while SET_A is still active), both control modes again set simultaneously.

The originally verified model consisted of 58 function blocks, and the NuSMV model contained $3{,}6 \cdot 10^{19}$ reachable states. NuSMV took 0.3 seconds to produce the counterexamples (recorded on an Intel Core i7-6600U CPU with a clock rate of 2.6 GHz).

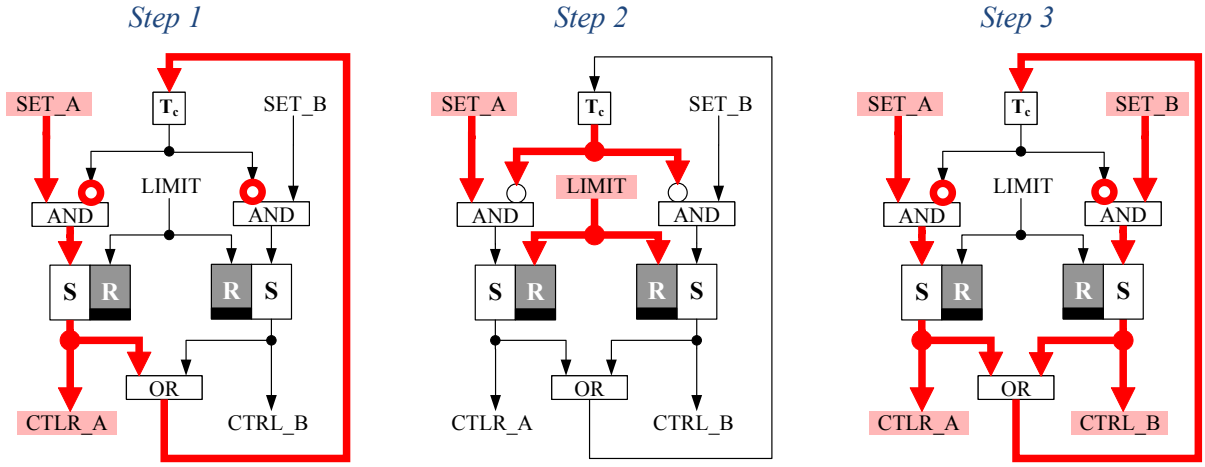

Figure 3: The third counterexample for Example 1, requiring no simultaneous switching commands from the operator.

This example highlights that (1) the first counterexample is not necessarily the most likely one, but (2) the analyst can look for alternative counterexamples by ruling out unrealistic states and execution paths. The third counterexample in Fig. 3 also requires very exact timing to occur. SET_B needs to be switched on the same processing cycle (within milliseconds) as LIMIT is deactivated——any sooner and the LIMIT

signal prevents CTRL_B from being set, any later and the re-activated CTRL_A causes the delayed signal to prevent CTRL_B from being set. Coming up with such test cases is hard. Getting a simulator to reproduce such a scenario might also be hard in practice.

For illustrative purposes, the NuSMV input file for Example 1 is shown below. Note that the script shown here does not follow our modelling approach [3], since we have emitted the validity processing logic [3], and "flattened" the AND and OR blocks to shorten the script. (Signal validity is a relevant factor in six of the real-world issues we have detected, even though it is not used in the functional diagrams.)

```
MODULE main()
  VAR
    SET_A : boolean;
    SET_B : boolean;
    LIMIT : boolean;

    RS001 : RS(LIMIT, AND001);
    RS002 : RS(LIMIT, AND002);
    DELAY001 : DELAY(OR001);
  DEFINE
    AND001 := SET_A & !DELAY001.OUT1;
    AND002 := SET_B & !DELAY001.OUT1;
    OR001 := RS001.OUT1 | RS002.OUT1;

    CTRL_A := RS001.OUT1;
    CTRL_B := RS002.OUT1;

    --LTLSPEC G !(CTRL_A & CTRL_B);
    --LTLSPEC !(SET_A & SET_B) -> G !(CTRL_A & CTRL_B);
    LTLSPEC !(SET_A & SET_B)
       & G !((!SET_A & X SET_A) & (!SET_B & X SET_B))
      -> G !(CTRL_A & CTRL_B);

MODULE DELAY(IN1)
  VAR
    mem : boolean;
  DEFINE
    OUT1 := mem;
  ASSIGN
    init(mem) := FALSE;
    next(mem) := IN1;

MODULE RS(RESET, SET)
  VAR
    mem : boolean;
  DEFINE
    OUT1 :=
    case
      RESET : FALSE;
      SET : TRUE;
      TRUE : mem;
    esac;
    OUT2 := !OUT1;
  ASSIGN
    init(mem) := FALSE;
    next(mem) := OUT1;
```

4.2. Example 2: Subsystem Inhibition Logic

In our second example, we have a logic for inhibiting one of three redundant subsystems for testing or maintenance purposes (see Fig. 4). The operator can switch at most one subsystem at a time for inhibition—if more than one switch is on at the same time, no inhibition signal is set. One requirement to verify is that all three subsystems shall not be inhibited simultaneously (the worst case scenario). The analyst specifies:

$$\mathbf{G}\,\neg(\text{INH\_1} \wedge \text{INH\_2} \wedge \text{INH\_3} \wedge \text{One\_INH})$$

NuSMV returns a counterexample (see Fig. 4), where each switch (SWT_1 to SWT_3) is activated in succession, so that exactly on the processing cycle where one switch is reset, the next switch is set. The logic only ever gets input combinations where exactly one switch is on.

The original logic had four redundant subsystems, and the original model consisted of 49 function blocks, and the NuSMV model contained $9{,}9 \cdot 10^{14}$ reachable states. NuSMV took 0.4 seconds to produce the counterexample.

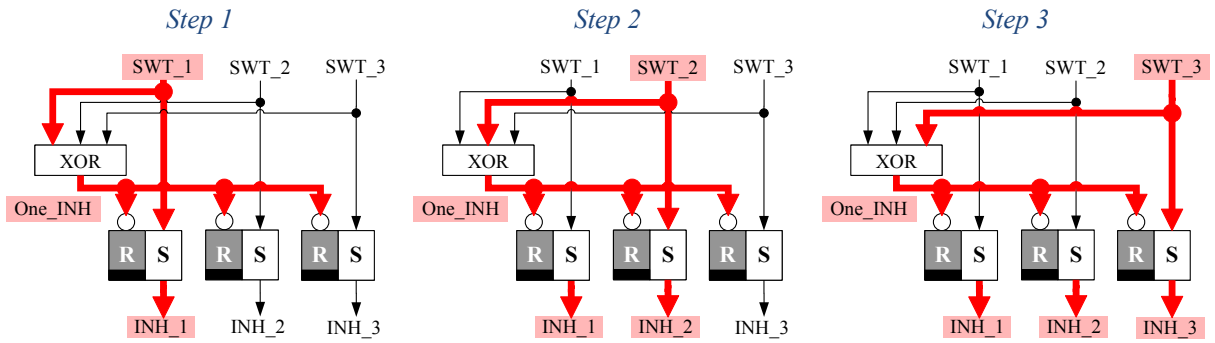

Figure 4: The logic and the counterexample for Example 2.

Like our last counterexample for the first example, the counterexample in Fig. 4 requires very exact timing, but here, the scenario is even more unlikely to ever occur. It is obvious that a corresponding test case would not be among the easiest to come up with.

5. CONCLUSIONS

In the words of VTT's customers, model checking is "truly beneficial in nuclear I&C projects", "found issues which would have otherwise been left undetected", and the results are "remarkable". Using the method has become an industry practice in Finland, regardless of the life-cycle phase or the design stage.

The examples shown in this paper illustrate how easy and fast it is to exhaustively verify that the *unwanted* scenario can never occur, be it spurious actuation, contradictory commands, or in general, the incorrect response from I&C.

The nuclear industry should always strive for improving safety. That fact alone would justify the broader application of model checking. But really, it is an issue of cost. Hidden design errors, by manifesting during operation, can cause—besides accidents—unnecessary shutdowns, damage to equipment, and then re-design and re-licensing efforts. Costs from such setbacks are nowhere near comparable to the cost of applying model checking in the design phase, to catch the errors then. Furthermore, as the analysis itself is so fast, nuclear I&C vendors could increase their performance in project delivery with the fast and easy solution iteration that model checking enables.

ACKNOWLEDGEMENTS

Our research on model checking is currently funded by the National Nuclear Safety and Waste Management Research Programme 2023-2028 (SAFER2028). We thank our customers (Fennovoima, Fortum, STUK and TVO) for helping us promote the use of the method.

REFERENCES

[1] E. M. Clarke, O. Grumberg and D. Peled, *Model Checking*, MIT Press, Cambridge, MA (1999).

[2] IAEA, "Introduction to Systems Engineering for the Instrumentation and Control of Nuclear Facilities", IAEA Nuclear Energy Series No. NR-T-2.14 (2022).

[3] 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*, **205**, 107237 (2021).

[4] 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," *2002 International Conference on Computer Aided Verification (CAV)*, LNCS 2404, pp. 359-364 (2002).

[5] 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: A. Biere and R. Bloem (Eds.), *Computer Aided Verification*, Springer International Publishing, Cham, pp. 334-342 (2014).

[6] L. Lamport, "Proving the correctness of multiprocess programs," *IEEE Transactions on Software Engineering*, **SE-3** (2), pp. 125-143 (1977).

[7] M. Benedetti and A. Cimatti, "Bounded Model Checking for Past LTL," *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003)*, LNCS 2619, pp. 18-33 (2003).

[8] I.D. Lopez-Miguel, J-C. Tournier and B. Fernandez, "PLCverif: Status of a formal verification tool for programmable logic controller," *18th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2021)*, Shanghai, China, Oct. 14-22 (2021).

[9] T. Ausberger, K. Kubíček, Pavla 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," *27th IEEE Intenational Conference on Emerging Technologies and Factory Automation (ETFA 2022)*, Stuttgart, Germany, Sept. 6-9 (2022).

[10] S. Gautham, A. V. Jayakumar, A. D. Rajagopala and C. Elks, "Accessible Formal Verification for Design Assurance of Model Based Critical I&C Systems," *12th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human-Machine Interface Technologies (NPIC & HMIT 2021)*, Providence, RI, USA, June 14-17, pp. 802-811 (2021).

[11] A. Wakankar, A. Kabra, A.K. Bhattacharjee and G. Karmakar, "Architectural model driven dependability analysis of computer based safety system in nuclear power plant" *Nuclear Engineering and Technology*, **51** (2), pp. 463-478 (2019).

[12] V. Kumar, K.C. Mishra, P. Singh, A.N. Hati, M.R. Mamdikar, L. K. Singh and R.N.R. Parida, "Reliability analysis and safety model checking of Safety-Critical and control Systems: A case study of NPP control system" *Annals of Nuclear Energy*, **166**, 108813 (2022).

[13] WNA, "Safety Classification for I&C Systems in Nuclear Power Plants - Current Status & Difficulties", World Nuclear Association Report No. 2015/008 (2015).

[14] A. Helminen and A. Pakonen, "Potential applications of model checking in probabilistic risk assessments", VTT Research Report VTT-R-00017-20 (2020).

[15] A. Pakonen, "Oops! Examples of I&C design issues detected with model checking," *International Symposium on Future I&C for Nuclear Power Plants (ISOFIC 2021)*, Okoyama, Japan, Nov. 15-17 (2021).

[16] A. Pakonen, "Model-checking infinite-state nuclear safety I&C systems with nuXmv," *19th IEEE International Conference on Industrial Informatics (INDIN 2021)*, Palma de Mallorca, Spain, July 21-23 (2021).