Considerations on operator behavior modeling in I&C system model checking

Antti Pakonen
Semantum Ltd.
Espoo, Finland
antti.pakonen@semantum.fi

***Abstract*—Model checking has been proven to detect design issues in industrial instrumentation and control (I&C) system logics. During 17 years of practical projects in the Finnish nuclear industry, VTT has identified 98 cases of the I&C logic violating functional requirements. 30 of these scenarios feature human operators making different types of errors. By also modeling the actions of the operators, we could potentially filter out unrealistic scenarios, reveal design issues otherwise hidden, and lower the computational cost of model checking by liming the execution paths of the I&C logic model.**

This paper studies the implications of modeling human operator actions in the context of I&C logic model checking. Based on practical examples from real nuclear industry projects, it is not necessarily advisable to model the operator's actions. Also, based on our experiments, the addition of the operator model will not by default make the analysis faster, even if the reachable state space of the I&C logic model is reduced.

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

I. Introduction

Model checking [1] is a formal verification method that can be used to determine if a model of a system satisfies stated formal properties. When deriving the properties from functional requirements, model checking serves as a powerful verification method. The result is either a proof that the property holds or a counterexample trace demonstrating a model execution path that violates the property.

In Finland, VTT[1] has applied model checking in the verification of nuclear power plant (NPP) instrumentation and control (I&C) logics since 2008 [2]. A total of 98 design issues were found by 2025. In 30 issues, the counterexample features human users—operators in the control room, or maintenance personnel in the field—making different types of errors.

In open-loop modeling, the environment of the verified system is not included. When the operator's actions (and the measurements from the controlled plant) are modeled as free, non-deterministic inputs, the potentially resulting "spurious counterexamples" [3] can feature input sequences that are impossible in the real world. Closing the loop by modeling the operator actions (or the plant dynamics [4]) can limit the execution paths of the I&C model, which could help mask unrealistic scenarios and lower the computational cost.

This paper aims to address three research questions:

1) What are the safety implications of modeling the operator's expected actions?
2) What kind or properties could be violated by the addition of an "operator model"?
3) Does an "operator model" necessarily reduce model state space and shorten analysis times?

Many previous works focus on developing methods of modeling erroneous human actions to discover emergent overall system behavior. In this paper, our aim is not to present techniques for operator behavior modeling, but to study the implications of adding such a model on (1) the reliability of the verification results and on (2) the practical applicability of model checking. The analysis is based on real nuclear industry case study examples.

Section II describes the fundamentals of model checking. Related research is summarized in Section III. Section IV describes the detected issues related to operator errors and Section V the case studies. The results are discussed in Section VI, and the conclusions are presented in Section VII.

II. Model checking

A. Tools and algorithms

As the objective is a logical proof that no model state or execution path can violate a stated property, the key challenge in model checking is to avoid the so-called state space explosion [1], where the number of states becomes too high to reason over in feasible time. To avoid the problem, symbolic model checkers such as NuSMV [5] use Decision Diagrams (BDD) [6] for canonical representation of Boolean formulas, avoiding the need to process every state explicitly. Bounded model checking (BMC) [7] is a technique based on Boolean satisfiability (SAT) solvers, where a limit is set for the length of checked sequences of state transitions. While useful for bug hunting, BMC can also produce a false positive result.

The infinite-state model checker nuXmv [8] uses a combination of techniques to extend the scope to real and unbounded integer number math.

VTT's practical work has been based on NuSMV and nuXmv, but other tools of interest include SPIN [9] for explicit-state analysis, Uppaal [10] for continuous time analysis, and PRISM [11] for probabilistic model checking.

This work has been funded by the Finnish National Nuclear Safety and Waste Management Research Programme 2023-2028 (SAFER2028). The research was conducted while the author was still affiliated with VTT.

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

B. Formal property specification

The verified properties are specified using *temporal logic* languages. Linear Temporal Logic (LTL) [1] specifies properties that hold for all execution paths. *Temporal operators* such as "Globally" ($\mathbf{G}\ \phi$: ϕ is true at every state) or "Finally" ($\mathbf{F}\ \phi$: ϕ is true at some future state) are used with logical operators to describe statements about model variables over time.

In contrast, Computation Tree Logic (CTL) [1] is based on branching time. It uses *path quantifiers* $\mathbf{A}$ (all execution paths) and $\mathbf{E}$ (some execution path) in conjunction with temporal operators such as $\mathbf{G}$ and $\mathbf{F}$. Some CTL properties we have found useful in our work [12]—such as the *reachability property* [13] $\mathbf{AG}\,\mathbf{EF}\ \phi$—cannot be formulated in LTL.

Property Specification Language (PSL) [14] extends LTL and CTL, and, in addition to syntactic sugar (e.g., `never` p for $\mathbf{G}\ \neg p$), provides the Sequential Regular Expressions (SERE) "style" convenient for expressing properties on multi-cycle behavior of I&C systems [12].

III. Related research

Modeling operator actions in the context of model checking has been studied with different motivations. The approach can be to model either erroneous or intended user actions.

In [15], Bolton et al. present an approach of generating models of operator errors based on a task-based taxonomy of erroneous human behavior. They claim that a formal model allowing any operator behavior to manifest at any time "will not be interesting to analysts" and therefore place a limit on how many erroneous behaviors will be allowed. The Enhanced Operator Function Model (EOFM) is based on XML representation, which is then translated to SAL [16] for verification.

In [17], Akarpour et al. model human-robot collaboration by extending the human operator model with states describing normative or erroneous actions. As in [15], the number of possible human errors is limited. $\mathbb{Z}$ot [18] is then used to verify the safety of working in close proximity with robots.

In [19], Basuki et al. capture emergent system behavior arising from unexpected operator actions. The Maude system [20] then supports LTL model checking.

In [21], Aït-Ameur & Baron suggest an approach for the validation of user interface design by modeling the intended user tasks using a type of tree notation. The event B [22] technique then supports formal verification of reachability and observability properties, and validation of user tasks.

In [23], Muhammad specifies an operator model accounting for, e.g., age, experience and fatigue, and then uses PRISM to assess the impact on the operation of a drone system.

Beyond formal verification, there are many works on modeling the actions of operators in an industrial context. To mention one recent example with a different motivation: In [24], Markaj et al. specify operator behavioral models using stochastic state machines and then integrate those models with a process plant model in Simulink. The objective is to identify a suitable degree of plant autonomy in an early design phase.

In contrast with the above-mentioned research, this paper focuses on the implications of human action modeling on (1) safety and (2) the practical applicability of model checking, based on industrial experience.

IV. Analysis of industrial data

In the practical nuclear industry projects mentioned above, VTT has by 2025 found a total of 98 confirmed design issues, 94 with NuSMV and four with nuXmv. Based on our analyses, the actions of operators or maintenance personnel are relevant in 43 issues. In 30 issues (31.6%), human error is a key factor.

To identify and classify recurring features in the issues, we can use action error modes from Systematic Human Error and Prediction Approach (SHERPA) [25]. Grounded on cognitive models of human behavior, SHERPA works by indicating error modes in tasks based on analysis of work activity [26]. There are different variants of SHERPA error mode taxonomies, the one adopted here is from [26]. The most common error modes for our issues are "A2 Operation mistimed" and "A7 Wrong operation on right object" (see Table I).

TABLE I
The occurrence of SHERPA action error modes [26]

Action error mode	Count
A1 Operation too long/short	4
A2 Operation mistimed[a]	10
A3 Operation in wrong direction[b]	4
A4 Operation too little/much	
A5 Misalign	
A6 Right operation on wrong object	
A7 Wrong operation on right object[c]	10
A8 Operation omitted	1
A9 Operation incomplete	
A10 Wrong operation on wrong object	1

[a] See example 2 in Section V-A.
[b] See example 1 in Section V-A.
[c] See example 3 in Section V-B.

We also categorized the issues using error modes from Cognitive Reliability and Error Analysis Method (CREAM) [27]. The error taxonomy in CREAM is intended to support human reliability analyses, accident analyses, and system design. Under this categorization, the most common error mode in the data was now "Object", followed by "Timing" (see Table II).

TABLE II
The occurrence of CREAM error modes [27]

Error mode	Count
E1 Timing (too early, too late, omission)	7
E2 Duration (too long, too short)	4
E3 Sequence (reversal, repetition, commission, intrusion)[a]	4
E4 Object (wrong action, wrong object)[b]	11
E5 Force (too much, too little)	
E6 Direction (wrong direction)[c]	4
E7 Distance (too short, too far)	
E8 Speed (too fast, too slow)	

[a] See example 2 in Section V-A.
[b] See example 3 in Section V-B.
[c] See example 1 in Section V-A.

The online annex [28] to this paper contains short descriptions of each issue and their classification based on both the SHERPA action error modes and the CREAM error modes.

V. Case studies

The addition—or the omission—of the operator model can have either useful or detrimental effects to the analysis results. The risks considered below are that a (1) real issue is masked, (2) a spurious counterexample will prove unrealistic, or (3) the computational cost will prove excessive.

A. Masking of relevant issues

A conservative operator model will limit the execution paths of the connected I&C model. We therefore run a risk of accidentally preventing a crucial execution path.

To illustrate this risk, let us consider two running examples. Both are from VTT's practical projects, where the analysis revealed a problem in a real-world design. The examples are masked and simplified from their original context.

For example 1, the original logic consisted of 152 blocks. Fig. 2 shows the simplified example, along with a simple operator model F_O. For simplicity in illustration, same type of function block diagrams are used to represent F_O and the I&C logic model F_I. The blocks are described in Fig. 1.

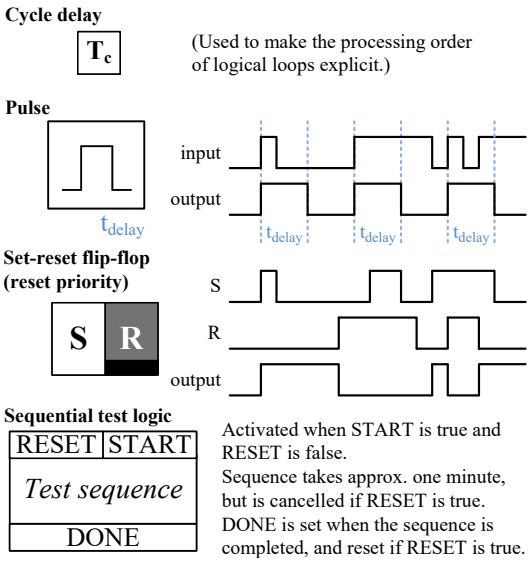

Fig. 1. Key for the function blocks used in the example logics

In the example, a `decrease` command from the operator leads to the `close` output being set until a `low` signal from the plant leads to a stop of the function. However, a PSL safety property to exclude rapid oscillation (**never** `{!close; close; !close; close; !close; close}`), results in a counterexample where the operator commands `decrease` although `low` is already true (see Fig. 3).

For example 2, the original logic consisted of 34 function blocks. Fig. 4 shows the simplified example, along with a simple operator model F_O. Here, a `start` command initiates a sequential test logic, taking about a minute to process. The test sequence is deactivated upon reaching its completion or by the operator's `stop` command. However, a CTL property to ensure that a state where the test is not "done" is always

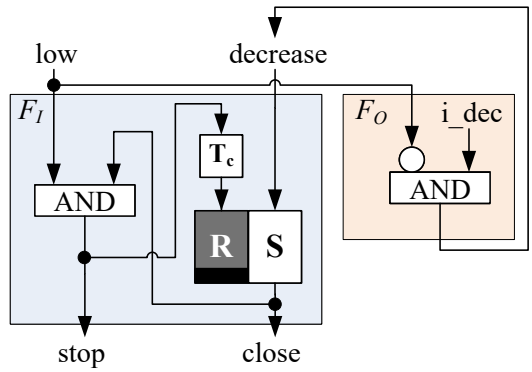

Fig. 2. Example 1: an I&C logic model F_I, where, if an operator model F_O is added, a design issue in F_I is masked. The circular element at the AND block input represents a negation.

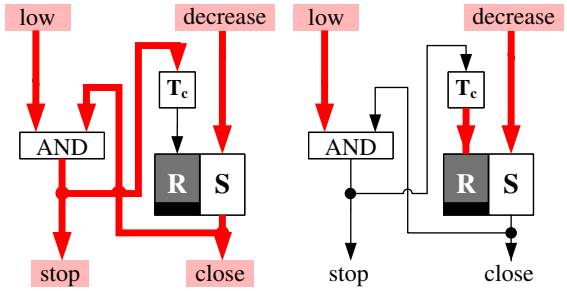

Fig. 3. If the operator commands `decrease` even though `low` is true, the I&C logic in example 1 will enter a rapid oscillation between two states.

reachable (**AG EF** $\neg$`done`), results in a counterexample illustrated in Fig. 5, where a rapid succession of back-and-forth commands from the operator places the logic in a permanently stuck state.

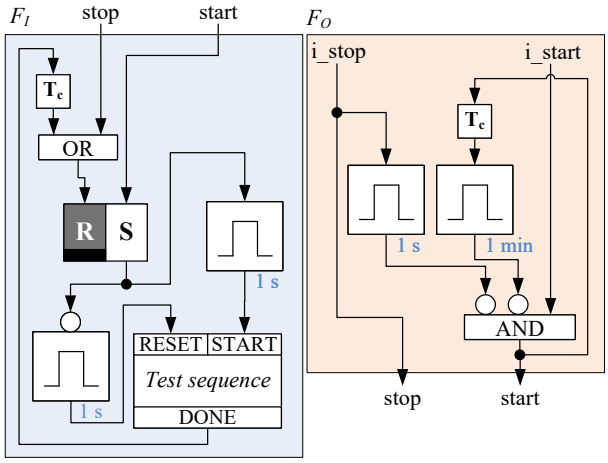

Fig. 4. Example 2: an I&C logic model F_I, where, if an operator model F_O is added, a design issue in F_I is masked.

What is notable about these real-world examples is that the operator models F_O shown in Fig. 2 and Fig. 4 are added, the counterexample execution paths are no longer reachable, and the above properties now hold. The issues

are effectively masked from being discovered, and although the operator models are justifiable (the operator should not command `decrease` if `low` is true, or command `start` immediately after `stop`), it cannot be guaranteed that all the excluded execution paths are truly irrelevant.

On the other hand, we can also consider the option where including the operator model would reveal a relevant design issue not valid for the I&C logic model alone.

If a LTL property holds for all the execution paths of the I&C model F_I, it will also hold after adding an arbitrary operator model F_O. F_O can introduce additional model variables and add to the number of reachable states in the joint model, but it cannot add execution paths to F_I.

CTL properties, however, do not have to hold for all execution paths. Therefore, a CTL property could be violated by adding the operator model.

Fig. 6 shows a semi-fictitious (see [29]) example, where F_I satisfies the LTL property $\mathbf{G}((\texttt{ack\_a} \wedge \texttt{set\_b}) \rightarrow \texttt{act\_b})$, but the overall logic $F_O + F_I$—while still satisfying the above LTL property—breaks the CTL reachability property $\mathbf{AG\ EF}\ \neg\texttt{act\_b}$, because at the end of the counterexample, we can no longer reach a state where $(\texttt{ack\_a} \wedge \texttt{set\_b})$ would hold. At the end of the scenario, the above LTL property is trivially true and essentially meaningless.

However, F_O in Fig. 6, if we assume that the operator is free to act, then regardless of any feedback from the I&C or the plant, any operator action ψ is always reachable ($\mathbf{AG\ EF}\ \psi$ holds). If every operator action is always reachable, then any violation of the reachability property ($\mathbf{AG\ EF}\ \phi$) for the outputs of F_I will be revealed by the verification of F_I, alone.

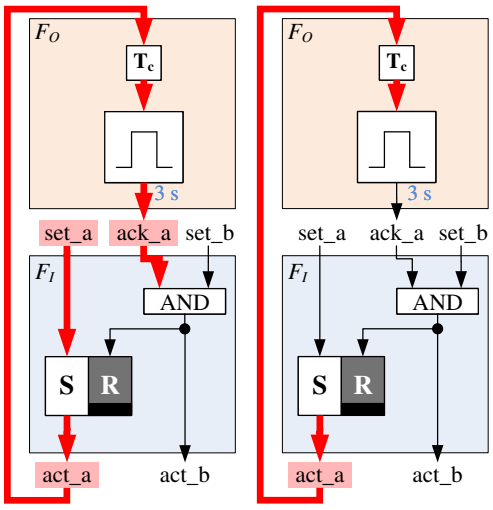

Fig. 6. A semi-fictitious logic, where adding the operator model causes the logic in the depicted counterexample ending up in a frozen state

B. Spurious counterexamples

A less severe risk is that the addition or the omission of the operator model will result in a spurious counterexample depicting an impossible scenario, in which case unnecessary extra time could be spent on processing such counterexamples.

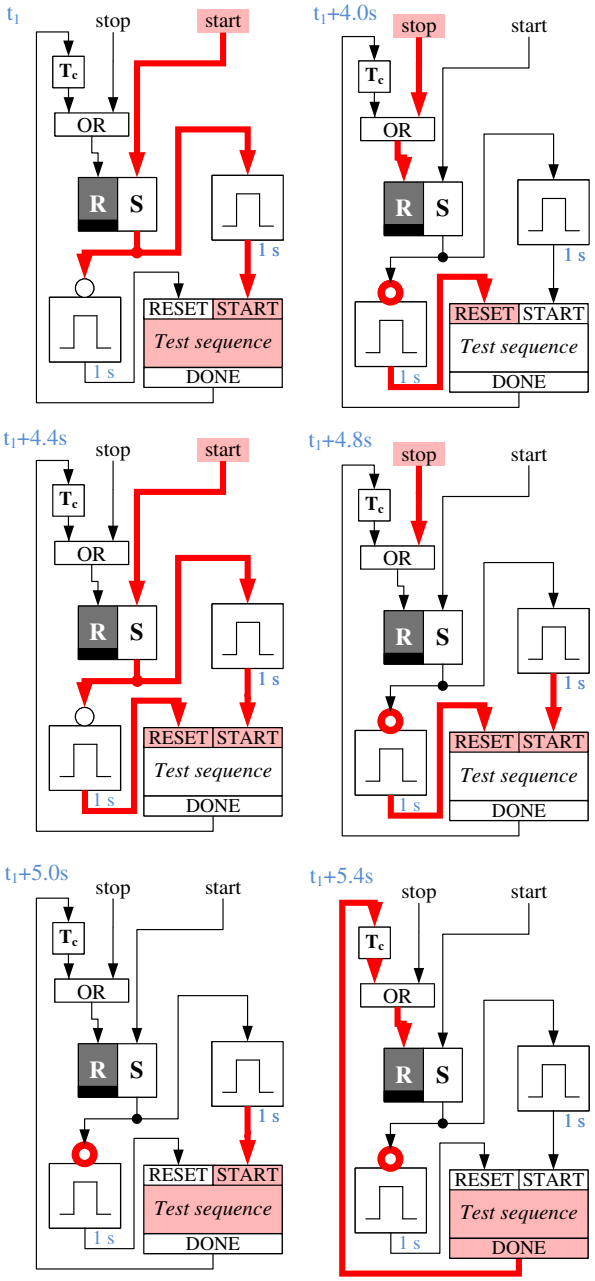

Fig. 5. Counterexample for the second logic: if the second `stop` is received before a second has passed from the previous `stop`, a `start` in between the stops will place the I&C logic in a frozen state, where the test cannot be reset or restarted.

In our online annex [28], issues where operators set several redundant subsystems into test or manual mode at the same time are common. These counterexamples can be considered spurious, if we assume that operators would not deliberately act in such a fashion (or if some procedure, like a key, prevents simultaneous access). In this case, an operator model could filter unrealistic counterexamples.

Example 3 illustrates useful counterexample filtering. Fig. 7 shows an I&C logic F_I for inhibiting one of three redundant subsystems for testing and maintenance. As shown in [30],

the operator can inhibit all three subsystems simultaneously through a specific quick succession of commands. However, we can prevent that by adding the operator model F_O in Fig. 7.

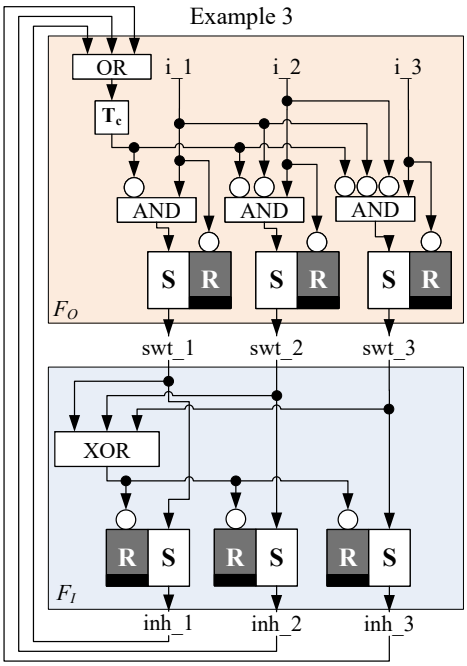

Fig. 7. An operator model F_O added to the third exemplar I&C logic F_I.

If, on the other hand, the operator model is flawed, and the analysis results in a counterexample caused by such a flaw, then it is the added operator model that causes extra work.

However, in each case, the effect is that extra time is spent by the analyst to rule out spurious counterexamples. While practically inconvenient, this effect is not detrimental to safety, in that it does not limit the verification results.

C. Computational overload

In VTT's projects, the complexity of some models has meant that the analyst has had to use BMC, which is unable to produce verification proofs, but has revealed several issues.

Intuitively, the addition of the operator model should constrain the execution paths of the connected I&C model, which could then decrease the analysis time and/or enable more comprehensive analysis of logics otherwise too complex.

To experiment with the effect of the operator model on the computational load, we collected real-world industrial NuSMV models from VTT's projects. In Table III, Logic 1 is the original logic where example 1 (Fig. 2) comes from, logic 2 is where example 2 (Fig. 4) comes from, and logic 3 is where example 3 (Fig. 7) comes from. As stated above and in [30], in their original context, these models contain 152, 34, and 49 function blocks, respectively. Due to their confidentiality, the original models and their sources cannot be shared.

To each I&C logic, we then added the type of operator model simplified in Fig. 2, Fig. 4 and Fig. 7. If NuSMV did not finish within 12 hours, the process was terminated.

The results are summarized in Table III.

The number of reachable states is not always a good measure of computational complexity, at least for symbolic model checkers. A more practical measure is the computation time. As seen in Table III, the number of reachable states in Logic 3 is seven orders of magnitude higher than in Logic 2, but the analysis time is only fractions when compared with Logic 2. (The times were recorded with NuSMV 2.6.0 on an Intel Core i5-1235U CPU with a clock rate of 1.30 GHz.)

Our experiments were rather limited. However, they show that the addition of the operator model does not necessarily make the analysis faster, and the analysis time can in certain cases even grow prohibitively long. Presumably, this is due to the additional variables and logic introduced.

VI. Discussion

In Section V-A, we pointed out that a LTL property holding for the I&C model F_I alone will still hold for F_I even if we add an operator model F_O to limit the states reachable for F_I. This principle does not hold for CTL properties, which represent 10% of properties specified in VTT's practical projects [12]. However, the only type of CTL property used three times or more—or the only type to have revealed design issues—is the reachability property $\mathbf{AG\ EF}\ \phi$. As discussed above, for the combination of $F_O + F_I$ in unison to violate $\mathbf{AG\ EF}\ \phi$, the operator would have to be unable to perform some action. However, human operators can always use their intuition and experience in unforeseen situations [24].

A recurring feature in the issues is that they feature operators behaving in ways that one could claim they would not, or that the instructed procedures should prevent. From a safety point of view, it would still be preferable to design the I&C systems in a way that the unwanted scenarios can simply be excluded by the I&C, alone. If users are deliberately left with a known avenue for misuse, there are few guarantees that a bad actor would never try to abuse such a possibility. Given enough time, the risk also grows that an operator acting in good faith might simply make the right type of error. From the defense-in-depth principle, it does not follow that it is acceptable to have "holes" in the I&C just because the operators are trusted to always act in a certain way.

For simplicity, our examples used function block diagrams to specify the operator models. Some other language is likely to be more appropriate for a real implementation.

Our analyses are limited by only working with the symbolic model checkers NuSMV. The results for computational load could differ for explicit-state model checkers. In addition, we only used three models to calculate the effect on analysis time.

VII. Conclusion

Reflecting to the research questions posed in Section I:

1) An operator model can obviously filter out unrealistic execution paths and therefore spurious counterexamples. However, the concern is that an overly conservative (or just plain incorrect) operator model could accidentally mask a realistic and relevant scenario.

TABLE III
COMPARISON BETWEEN MODELS WITH I&C LOGIC (F_I) ONLY AND MODELS WITH OPERATOR MODEL (F_O) ADDED

	Logic 1		Logic 2		Logic 3	
	F_I	$F_O + F_I$	F_I	$F_O + F_I$	F_I	$F_O + F_I$
Reachable states	$2.34 \cdot 10^{51}$	$1.85 \cdot 10^{52}$	$3.52 \cdot 10^{7}$	$8.23 \cdot 10^{7}$	$2.63 \cdot 10^{14}$	$3.44 \cdot 10^{14}$
CTL checking time / property (s)	36.3	—[a]	5.5	2.6	0.2	0.3
LTL/PSL checking time / property (s)	61.0	—[a]	31.7	27.1	0.3	0.3
Properties satisfied	27 / 31	—[a] / 31	6 / 10	7 / 10	6 / 8	8 / 8

[a]Process was terminated after 12 hours.

2) As an LTL property holding for a standalone I&C model will always hold for the joint model of the operator and the I&C, it is difficult to come up with a scenario where the addition of the operator model would "break" the I&C logic. Violation of CTL reachability properties (or the introduction of deadlocks) would require the operator to be unable to perform some action.
3) The additional logic introduced by the operator model does not necessarily limit the number of overall reachable states, and the analysis times can increase.

The first step in formal verification is the specification of model boundaries. When verifying I&C logics, the decision to include or exclude the operator actions in the model carries a risk of (1) unintentionally masking what would in reality be a relevant counterexample, (2) causing a spurious (irrelevant) counterexample, and/or (3) causing computational overload.

More holistic formal verification approaches are needed. Still, from a safety point of view, it seems justifiable to apply the open-loop approach, where the actions of human operators (or plant feedback) are *not* included in the formal model of the I&C system. A downside is having to process counterexamples that would not necessarily be realistic in the broader context, but it is preferable to be safe rather than sorry.

ACKNOWLEDGMENT

The author thanks Hanna Koskinen and Sami Karadeniz of VTT for their help with Section IV.

REFERENCES

[1] E. Clarke, O. Grumber, and D. Peled, *Model checking*, 2nd ed. Cambridge, Massachusetts, US: MIT press, 2001.
[2] 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, Jan. 2021, 107237.
[3] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *Proc. CAV*, ser. LNCS, vol. 1855, 2000, pp. 154–169.
[4] S. Preuße, H. Lapp, and H. Hanisch, "Closed-loop system modeling, validation, and verification," in *Proc. ETFA*, Sept. 2012, pp. 1–8.
[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 *Proc. CAV*, ser. LNCS, vol. 2404, 2002, pp. 359–364.
[6] 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.
[7] 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, July 2001.
[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 *Proc. CAV*, ser. LNCS, vol. 8559, 2014, pp. 334–342.
[9] G. J. Holzmann, "The model checker SPIN," *IEEE Trans. Softw. Eng.*, vol. 23, no. 5, pp. 279–295, 1997.
[10] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on Uppaal," in *Proc. SFM-RT*, ser. LNCS, vol. 3185, 2004, pp. 200–236.
[11] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. CAV*, ser. LNCS, vol. 6806, 2011, pp. 585–591.
[12] A. Pakonen, I. Buzhinsky, and V. Vyatkin, "Evaluation of visual property specification languages based on practical model-checking experience," *J. Syst. Softw.*, vol. 216, Oct. 2024, 112153.
[13] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen, and P. McKenzie, *Systems and Software Verification – Model-Checking Techniques and Tools*. Heidelberg: Springer, 2001.
[14] C. Eisner and D. Fisman, *A Practical Introduction to PSL*. US: Springer, 2006.
[15] M. L. Bolton, K. A. Molinaro, and A. M. Houser, "A formal method for assessing the impact of task-based erroneous human behavior on system safety," *Reliab. Eng. Syst.*, vol. 188, pp. 168–180, 2019.
[16] L. de Moura, S. Owre, and N. Shankar, "The SAL language manual," CSL, SRI International, Tech. Rep., 2003. [Online]. Available: https://sal.csl.sri.com/doc/language-report.pdf
[17] M. Askarpour, D. Mandrioli, M. Rossi, and F. Vicentini, "Modeling operator behavior in the safety analysis of collaborative robotic applications," in *Proc. SAFECOMP*, ser. LNCS, vol. 10488, 2017, pp. 89–104.
[18] M. Pradella. (2013) A user's guide to Zot. [Online]. Available: https://github.com/fm-polimi/zot/blob/master/doc/zot-man.pdf
[19] T. A. Basuki, A. Cerone, A. Griesmayer, and R. Schlatte, "Model-checking user behaviour using interacting components," *Form. Asp. Comp.*, vol. 21, pp. 571–599, 2009.
[20] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott, "The Maude 2.0 system," in *Proc. RTA*, ser. LNCS, vol. 2706, 2003, pp. 76–87.
[21] Y. Ait-Ameur and M. Baron, "Formal and experimental validation approaches in HCI systems design based on a shared event B model," *Int. J. Softw. Tools Technol. Transf.*, vol. 8, pp. 547–563, 2006.
[22] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering*. Cambridge, UK: Cambridge University Press, 2010.
[23] S. Muhammad, "Modeling operator performance in human-in-the-loop autonomous systems," *IEEE Access*, vol. 9, pp. 102 715–102 731, 2021.
[24] A. Markaj, F. Pelzer, N. Richter, A. Fay, and M. Mercangöz, "Development and integration of operator behavior models for the evaluation of autonomous plants," in *Proc. ETFA*, Sept. 2024.
[25] D. E. Embrey, "SHERPA: A systematic human error reduction and prediction approach," in *Proc. International Topical Meeting on Advances in Human Factors in Nuclear Power Systems*, Apr. 1986, pp. 234–251.
[26] N. A. Stanton and C. Baber, "Error by design: methods for predicting device usability," *Des. Stud.*, vol. 23, no. 4, pp. 363–384, 2002.
[27] E. Hollnagel, *Cognitive Reliability and Error Analysis Method – CREAM*. Oxford, UK: Elsevier Science, 1998.
[28] A. Pakonen, "Classification of I&C issues based on SHERPA and CREAM," Zenodo, 2026, v1. [Online]. Available: https://doi.org/10.5281/zenodo.19396196
[29] ——, "Oops! Examples of I&C design issues detected with model checking," in *Proc. ISOFIC*, Nov. 2021.
[30] ——, "Model-checking I&C logics – practical examples," in *Proc. NPIC & HMIT*, July 2023, pp. 1610–1619.