Verifying PLC Programs via Monitors: Extending the Integration of FRET and PLCverif*

Xaver Fink[1][0009−0006−2890−2809], Anastasia Mavridou[2][0000−0002−3943−9753], Andreas Katis[2][0000−0001−7013−1100], and Borja Fernández Adiego[1]

[1] European Organization for Nuclear Research (CERN), Geneva, Switzerland
{xaver.eugen.fink, borja.fernandez.adiego}@cern.ch
[2] KBR at NASA Ames Research Center, Moffett Field, CA, USA
{anastasia.mavridou,andreas.katis}@nasa.gov

Abstract. Verification of Programmable Logic Controller (PLC) programs requires reasoning about propositions qualified in terms of time. CERN's PLCverif, an open-source tool for the analysis of safety-critical PLC systems, uses Linear Temporal Logic (LTL) for the specification of properties. Until now, PLCverif depended on third-party tools that accept LTL specifications to perform verification. However, our experience with industrial PLC programs shows that, to overcome analysis limitations, a wide range of techniques are needed to successfully verify complex properties. In this paper, we extend PLCverif to enable PLC program verification of pure-past LTL (PLTL) safety properties with assertion-based verification tools. To this end, we take an algorithm from the runtime-monitoring domain, apply it to bounded model checking of PLC programs, and implement it in PLCverif. We extend the integration of NASA's Formal Requirements Elicitation Tool (FRET) into PLCverif to use PLTL properties generated with FRET. In addition, we leverage the program structure induced by the PLC scan-cycle for a state-space reduction. Finally, we expose the algorithm to a real-world case study of critical systems at CERN.

1 Introduction

Programmable Logic Controllers (PLCs) are heavily used in industrial control systems, even in safety-critical industrial installations, where a failure of the control system may have catastrophic consequences. To minimize risks, functional safety standards like IEC 61508 and IEC 61511 recommend the use of formal

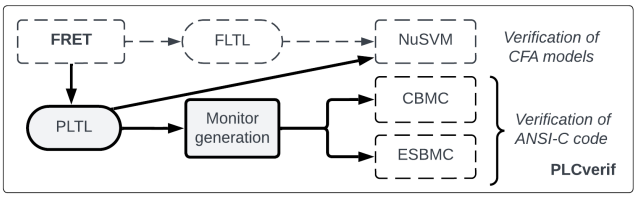

Fig. 1. PLCverif-FRET updated workflow.

methods to verify the correct behaviour of safety-critical PLC programs against formally specified *safety properties*. To this end, the PLCverif tool[3], actively developed at CERN, provides a platform for automated verification of PLC code against property specifications, utilizing various third-party verification backends. A critical part of the verification process in PLCverif is the specification of properties. Currently, this can be done either in the form of *boolean assertions* directly in the program code, or *patterns* (i.e., premade LTL templates), or *structured natural language* through the integration with NASA's FRET[4], a formal requirement authoring tool that can express a variety of pure past-time and pure future-time *LTL* formulas [1].

Case studies on industrial programs have shown that ensembles of verification tools enable a greatly improved verification coverage [8,3]. Notably, cutting-edge commercial tools are known to employ a wide variety of verification methods [2]. Employing a diverse portfolio of tools comes with its challenges though, especially with regards to supporting different specification languages. For instance, state-of-the-art formal verification tools for ANSI C programs focus on the support of code assertions and a limited set of LTL properties [3]. Thus, very often such tools are incompatible with properties expressed using the full expressibility of LTL. Related work has focused on bridging this gap, via means of generating assertions from LTL specifications, in the form of *monitors* [7,6].

In this paper, we leverage the monitoring work by Havelund & Roşu [7] to enable the verification of safety PLTL properties on infinite-loop PLC programs. For this, we transform PLTL into ordered monitor assignments over which a Boolean assertion is equivalent to the evaluation of the property on finite traces of the program. The property can then be verified by, the integrated in PLCverif, CBMC and ESBMC[5] assertion-based verification tools.

Figure 1 shows the workflow between the FRET and PLCverif tools. The first workflow (FRET to NuSMV[6]), presented in our prior work [1], supports the verification of pure-future LTL (FLTL) properties on control flow automaton (CFA) models generated from PLC code. However, this presents limitations since NuSMV does not natively support the verification of properties over explicit time (only time-steps). To overcome this limitation and also support verification of

[3] PLCverif is publicly available under https://gitlab.com/plcverif-oss.
[4] FRET is publicly available under https://github.com/NASA-SW-VnV/fret.
[5] https://www.cprover.org/cbmc/, http://www.esbmc.org.
[6] https://nuxmv.fbk.eu

ANSI-C code against LTL properties, the second workflow (bold continuous lines in Figure 1) represents the contributions of this work, which can be summarized as follows: (1) an extension of [7] with bounded temporal operations and an interpretation of the algorithm for assertion-based Bounded Model Checking (BMC) on PLC programs; (2) an implementation of the algorithm in the open source tool PLCverif, as well as an extension of the existing FRET integration; and (3) a case study of applying the workflow to safety-critical projects at CERN.

2 Preliminaries: PLC Verification and Specification

PLC program verification with PLCverif. PLCs are standardized robust industrial systems, performing computations in a well-defined cyclic structure known as *scan cycle*. The scan cycle consists of several phases, but for the purpose of program verification only three are relevant: (1) read and store sensor values, (2) execute the main PLC program and (3) send the calculated values to the actuators. This cycle is repeated continuously until the PLC is stopped. Most PLC safety properties are verified at the end of the scan cycle, since it is the most critical point in the system's execution. PLCverif models the three phases with a *control flow automaton* (CFA), with the end of the scan cycle being represented as a location in the CFA, called *EoC*.

PLC program specification with FRET. In this work, we use and extend the PLCverif-FRET integration [1] to allow users to author requirements using the structured natural language of FRET, called FRETish, which circumvents the need for expertise on writing temporal logic specifications. Once a requirement is written in FRETish, FRET generates equivalent PLTL and FLTL formulas.

The FRETish language provides a rich set of features to express temporal and functional constraints. The requirements presented later in this paper exercise the following subset[7]: 1) `condition` is a Boolean expression that either a) Upon keyword: triggers the `response` to occur at the time the expression's value becomes true, or is true at the beginning of the execution trace, or b) Whenever keyword: triggers the `response` to occur every time the expression's value is true 2) `component` is the system component that the requirement is levied upon, 3) `timing` specifies when the response shall happen, and 4) `response` is the Boolean expression that the component's behavior must satisfy.

We also use the `persisted(n, p)` special predicate offered by FRET that describes the persistence of a condition. Here `p` is any Boolean expression, which may include temporal predicates itself. The predicate `persisted` means that `p` has held true for the previous `n` time points, as well as the current time point.

3 Generating Monitors for PLTL Properties

Figure 2 shows the BMC pipeline workflow implemented for this work. The original PLCverif-FRET integration only considered FLTL formulas. As part of

[7] We exclude FRET's scope field [5] as it is not used in this paper.

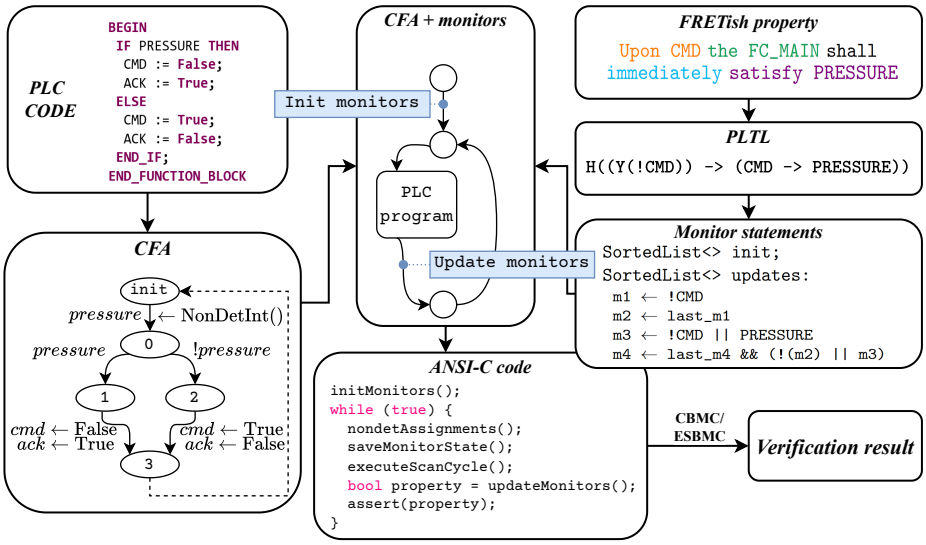

Fig. 2. Verification Pipeline in PLCverif via Monitor Generation.

this work, we implemented the appropriate means in PLCverif to retrieve the PLTL variant of a given FRET requirement. For the next step in the pipeline, the generation of monitor statements, we implemented the monitor synthesis algorithm by Havelund & Roşu [7].

The algorithm reduces the verification problem of PLTL to a Boolean query over runtime monitors. Essentially, for each finite unrolling of the model, the acceptance of a PLTL formula only depends on the current and previous variable state, allowing a simple recursive computation procedure.

We chose this algorithm (opposed to other LTL-to-Büchi approaches e.g., [6]) due to its simplicity for implementation and extension to specifics of the PLC-domain. This also extends to the evaluation of the monitors during runtime. For the algorithm by Havelund & Roşu this becomes a simple boolean query which is natively supported by BMC tools for ANSI C programs; monitoring alternatives such as Büchi automata require a more complex decision procedure based on infinite-state visits which was out-of-scope for this study.

In its original version, the algorithm does not provide the capability of monitoring metric properties, such as *"upon a, after two seconds b should hold"*. This is a key requirement for verifying PLC programs that include timer modules. Additionally, while the monitors are trivial to compute, they can add significant overhead to the verification task. Let n be the number of temporal operators in the PLTL formula, and m the number of variable state updates in the model. The algorithm by Havelund and Roşu adds $2 \cdot n$ boolean variables as monitors for the state change between cycles $\{t; t-1\}$, and $2 \cdot n \cdot m$ variable assignments to update the monitor variables after each variable value change. To tackle these problems, we propose two modifications of the algorithm.

Extending the algorithm to metric PLTL. To model PLC Timers, we extend the algorithm by Havelund & Roşu with recursive procedures for comput-

ing bounded temporal operations by utilizing Integer counters. As an example, the PLTL *"bounded historically"* operation intuitively corresponds to the FRET predicate `persisted(n, p)` with its semantics given by:

$\mathbf{H}[0, n]\,\varphi$ is true at time t if φ holds in all previous time steps $t - n \leq t' \leq t$.

The satisfaction relation can be calculated on the fly via a recursive definition:

$$timer_{n,\varphi} : \mathit{Trace} \to \mathbb{N}, \pi_i \mapsto \begin{cases} \max(timer_{n,\varphi}(\pi_{i-1}) - 1, 0), & \text{if } \pi_i \models \varphi \\ n, & \text{otherwise} \end{cases}$$

$$\pi_i \models \mathrm{H}[0, n]\,\varphi \textit{ iff } \pi_i \models \neg(timer_{n,\varphi}(\pi_i) > 0).$$

Upon violation of φ by trace π_i, a counter variable is activated, blocking the property for n steps. Other metric PLTL operators are similarly derived.

Restricting the value-range of the counters to known regions and the caching of previously computed values improves efficiency.

Overall, this algorithm allows for an elegant implementation and extension to our use case, while staying within the scope of this case study.

Abstracting over the scan cycle. We perform an abstraction over the scan cycle to tackle the high number of added variable assignments by the monitor generation. Instead of updating the monitors after each individual variable-update, the whole cycle is treated as one system-state update-function. This means, that the monitor variables are only updated once at the end of each cycle. Note, that this abstraction requires consideration while formulating the LTL requirements. To give an example, $\mathrm{Y}\,\varphi$ *("phi holds at the previous state")* on the abstracted state-model, is equivalent to *"φ holds at the end of the previous cycle"* on the original model.

Intuitively, this makes reasoning over cycle-internal properties impossible. To motivate the abstraction withstanding this limitation, consider that almost all PLC safety properties are evaluated at the end of the cycle as explained in Section 2. The abstraction reduces the number of added variable assignments from $2 \cdot n \cdot m$ to $2 \cdot n$.

Remark 1. The execution time of the program inside of the scan cycle is bounded by design. This means that if the scan cycle time exceeds the limit, a system failure would stop the PLC and any such program would not pass into production systems. This results in the abstraction having a nice interpretation under BMC of PLC programs. Since only the outermost scan cycle loop has to be unrolled, each unrolling represents a true-prefix of all possible program execution traces.

4 Case study

This new workflow has been applied to two critical PLC programs at CERN. Table 1 shows the FRETish requirements and their equivalent PLTL formulas generated by FRET for both PLC programs. Due to their temporal operators all properties were only directly verifiable via NuSMV until now.

ELISA safety program. ELISA (**E**xperimental **LI**near accelerator for **S**urface **A**nalysis) is a proton accelerator that will allow CERN visitors to observe a particle beam with their naked eyes and apply Ion Beam Analyses in different fields. The ELISA safety PLC program is in charge of protecting the CERN visitors and the expensive accelerator equipment from the risks related to the high voltage and radio frequency power needed to produce the particle beam. As part of the PLC program design phase, fifteen functional safety requirements were formalized. The verification revealed bugs in the original design (subsequently fixed), as well as inaccuracies in the requirement formalization. For this paper we selected one violated, one satisfied, and one timed property:

- **[EL1]**: a state change for the `RF_CMD` output should only occur when the correct input (`RF_PRESSURE_SWSt`) is given. This requirement was violated due to an incomplete functional safety specification. According to the risk analysis of the ELISA experiment, the proton beam can reduce the levels of O_2 and increase the levels of O_3 in the confined space of the experiment. A software failure related to this requirement could have serious safety implications for the CERN personnel and visitors.
- **[EL2]**: if the operator requests grounding (`OP_RQ`) while no safety condition is triggered (`PS_FarSt` and `VAC_PT06St`), the power supplies should remain ungrounded (`GND_CMD`). This requirement holds.
- **[EL3]**: the system is allowed to be grounded (`!GND_CMD`) only if both power supplies were off (`!EX_CMD` and `!FO_CMD`) for at least t milliseconds. We verified the property for $t \in \{500, 2000, 3000\}$, assuming 100 ms of cycle time. The property holds for the first two cases and a violation is expected for $t = 3000$, as the PLC code TON timer is set to trigger after 2 s.

The UNICOS OnOff Library. The OnOff PLC program is part of the CERN-made UNICOS framework [8] for the development of industrial control applications. This program was analyzed in a previous study by Ádám et al. [1]. It is used in hundreds of industrial installations, which means that a program failure may have enormous economical impact. We present two properties extracted from the functional documentation of the OnOff object. The properties were chosen as they were found to be violated due to correct, yet undocumented, edge case behavior, which until now was only verifiable via NuSMV.

- **[OnOff1]**: while in Manual operation mode (`MMoSt`), a request to move to the Automatic operation mode (`AuAuMoR`) arrives, the OnOff shall be in Automatic operation mode (`AuMoSt`) at the next cycle.
- **[OnOff2]**: if an interlock (`FuStopI`) triggers, the OnOff output (`OutOnOV`) should be in the fail-safe position (`PFsPosOn`) until the interlock is acknowledged (`AuAlAck` or `MAlAckR`).

Results. Table 2 shows the benchmark results of our monitoring-based technique together with the CBMC/ESBMC assertion-based verification tools

[8] UNICOS website https://unicos.web.cern.ch.

Table 1. ELISA and UNICOS OnOff requirements in FRETish and PLTL.

Req	FRETish & PLTL
[EL1]	Upon RF_CMD the FC_MAIN shall immediately satisfy RF_PRESSURE_SWSt (H ((RF_CMD and (Z (!RF_CMD))) -> RF_PRESSURE_SWSt))
[EL2]	Upon (OP_RQ & PS_FarSt & VAC_PT06St) the FC_MAIN shall immediately satisfy (GND_CMD) H ((((OP_RQ and PS_FarSt) and VAC_PT06St) and (Z (!(OP_RQ and PS_FarSt and VAC_PT06St)))) -> GND_CMD)
[EL3]	Whenever !GND_CMD & !OP_RQ the FC_MAIN shall immediately satisfy persisted($\langle t \rangle$,!EX_CMD & !FO_CMD) H (((!GND_CMD) and (!OP_RQ)) -> (H[0,$\langle t \rangle$] ((!EX_CMD) and (!FO_CMD))))
[OnOff1]	Whenever MMoSt & AuAuMoR the OnOff shall at the next timepoint satisfy AuMoSt (H ((Y (MMoSt and AuAuMoR)) -> (AuMoSt or (!(Y (TRUE))))))
[OnOff2]	Upon FuStopI the OnOff shall until (AuAlAck | MAlAckR) satisfy (PFsPosOn -> OutOnOV) (H ((H (!FuStopI)) or ([(!(AuAlAck or MAlAckR)) S ((!(AuAlAck or MAlAckR)) and (FuStopI and ((Y (!FuStopI)) or (!(Y (TRUE))))))] -> (PFsPosOn -> OutOnOV))))

against the native PLTL verification with NuSMV. The BMC runs were performed with an incrementally increasing unwinding limit; this is natively supported by ESBMC and NuSMV, and was implemented via a wrapper-script for CBMC. A time limit of one hour was set for the verification experiments and all the experiments were run on a Lenovo X1 Carbon (7th Gen) with an Intel i5 8365U processor and 16 GB RAM, running on Windows 11. Note, that BDD and k-induction based algorithms are sound and complete. Thus, timeouts of these algorithms on properties with known counterexamples are failures (e.g. [EL1] with the NuSMV BDD algorithm), as the tool failed to find the (existing) counterexample within the time limit (represented as TO in Table 2). Conversely, BMC is known to be sound but only complete up to a specified loop-unrolling limit. For benchmarks it is common to utilize unlimited incremental loop-unrolling until either a counterexample is found or the execution times out. Thus, a timeout on properties that are known to be correct should be interpreted as *"correct up to unrolling x"*. Alternatively, the reachability diameter of a program can be computed which acts as an upper bound for the unrolling-limit with which BMC is sound and complete [4].

Two main conclusions can be extracted from Table 2: (1) Overall, our approach shows better results than NuSMV on the OnOff program (a larger program than ELISA), and competitive results on ELISA. We encountered a false-positive in the k-induction run of ESBMC for the [OnOff1] property, confirmed by the ESBMC developers[9]. This reinforces the necessity for a portfolio of ver-

[9] https://github.com/esbmc/esbmc/issues/1735

Table 2. Verification results ($\top$:*"property holds"*, $\top^-(n)$:*"holds up to n-th unrolling"*, $-$: *"unknown"*, $\bot$:*"counterexample"*) and time (in seconds) for the ELISA and UNICOS OnOff case studies. [TO]: "timeout"; [×]: "cannot be verified".

		ELISA										**UNICOS OnOff**			
		EL1		*EL2*		*EL3*						*OnOff1*		*OnOff2*	
						$t = 500$		$t = 2000$		$t = 3000$					
Tool		time	result	time	result	time	result	time	result	time	result	time	result	time	result
CBMC	BMC	3.34	$\bot$	TO	$\top^-(206)$	TO	$\top^-(209)$	TO	$\top^-(194)$	4.74	$\bot$	7.03	$\bot$	25.95	$\bot$
ESBMC	k-Ind	1.47	$\bot$	1.42	$\top$	2.23	$\top$	5.64	$\top$	7.74	$\bot$	4.18	$\top$	4.57	$\bot$
	BMC	1.62	$\bot$	TO	$\top^-(810)$	TO	$\top^-(258)$	TO	$\top^-(186)$	5.62	$\bot$	2.78	$\bot$	3.21	$\bot$
NuSMV	BDD	TO	$-$	0.91	$\top$		×		×		×	4.37	$\bot$	TO	$-$
	BMC	1.20	$\bot$	TO	$\top^-(155)$		×		×		×	7.44	$\bot$	42.83	$\bot$

ification algorithms. (2) Verification of properties using explicit time units (e.g. seconds) is now possible in PLCverif, shown by the fact that some [EL3] variants are provable with CBMC/ESBMC, using our monitor-based approach.

5 Conclusion

We presented an implemented extension of the PLCverif-FRET workflow to enable assertion-based verification of PLC programs via monitors of PLTL formulas. Algorithmic adaptations of the monitor generation algorithm to the PLC domain allow the efficient verification of metric PLTL properties. The new workflow was applied to critical PLC programs at CERN and showed that leveraging heterogeneous verification techniques provides stronger analysis capabilities. In the future, we plan to extend this work to finite-trace FLTL monitors.

References

1. Ádám, Z., Lopez-Miguel, I.D., Mavridou, A., Pressburger, T., Beś, M., Blanco Viñuela, E., Katis, A., Tournier, J.C., Trinh, K.V., Fernández Adiego, B.: From natural language requirements to the verification of programmable logic controllers: Integrating fret into plcverif. In: Rozier, K.Y., Chaudhuri, S. (eds.) NASA Formal Methods. pp. 353–360. Springer Nature Switzerland, Cham (2023)
2. Afzal, M., Asia, A., Chauhan, A., Chimdyalwar, B., Darke, P., Datar, A., Kumar, S., Venkatesh, R.: Veriabs : Verification by abstraction and test generation. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 1138–1141 (2019). https://doi.org/10.1109/ASE.2019.00121
3. Beyer, D.: Progress on software verification: SV-COMP 2022. In: Proc. TACAS (2). pp. 375–402. LNCS 13244, Springer (2022)
4. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Handbook of satisfiability **185**(99), 457–481 (2009)
5. Giannakopoulou, D., Pressburger, T., Mavridou, A., Schumann, J.: Automated formalization of structured natural language requirements. Information and Software Technology **137**, 106590 (2021). https://doi.org/https://doi.org/10.1016/j.infsof.2021.106590, https://www.sciencedirect.com/science/article/pii/S0950584921000707

6. Havelund, K., Peled, D.: Runtime verification: from propositional to first-order temporal logic. In: Runtime Verification. pp. 90–112. Springer (2018)
7. Havelund, K., Roşu, G.: Synthesizing monitors for safety properties. In: TACAS. pp. 342–356. Springer (2002)
8. Westhofen, L., Berger, P., Katoen, J.P.: Benchmarking software model checkers on automotive code. In: NFM. pp. 133–150. Springer (2020)