Practice-Oriented Formal Methods for PLC Programs of Industrial Control Systems

Dániel Darvas[1,2]

[1] CERN – European Organization for Nuclear Research, Geneva, Switzerland
ddarvas@cern.ch
[2] Budapest University of Technology and Economics, Budapest, Hungary
darvas@mit.bme.hu

Abstract

In many cases the precise verification of industrial control systems is desired, but the available resources are limited, making the usage of formal methods typically unaffordable. This paper describes three different, practice-oriented, easy-to-use approaches to apply model checking, conformance checking and code generation based on formal specification. These approaches do not require deep expertise in formal methods, nor radical changes in the development process.

1 Introduction

Industrial control systems (ICS) control a wide variety of plants, e.g. water treatment plants, chemical plants, or auxiliary systems of particle accelerators. Very often they are implemented using programmable logic controllers (PLCs) that are specialised programmable hardware. The correctness of the system depends on both the hardware and the software design. Here we focus on the quality of the PLC programs.

Very often ICS are used together with separated, dedicated safety systems, therefore the PLC programs are typically not safety-critical and thus limited resources are allocated to their verification. However, the failure of such control systems can still cause significant economic losses if e.g. the plant has to be stopped.

Formal methods are typically considered too expensive and too difficult to use in these scenarios. In contrast, we claim that practice-oriented, adapted, easy-to-use formal methods can improve the quality without excessive resource requirements. In this paper we overview three proposed methods for different use cases, all of them are used at CERN.

2 Use Cases and Methods

This section introduces the use cases and our proposed solutions. They are summarized in Table 1. Note that UC1 is the least, UC3 is the most intrusive to the development process.

Table 1: Use cases for application of lightweight formal methods

	Available			
	Informal spec.	Manual impl.	Formal spec.	Action
UC1	+	+		Model checking of the manual implementation
UC2	+	+	+	Conformance checking between manual impl. and formal spec.
UC3	+		+	Code generation based on checked formal specification

2.1 Model Checking PLC Programs (UC1)

In the first use case our goal is to find hidden faults in the manual implementation using model checking, without having a formal specification available. We use our tool PLCverif[1] to generate formal models based on the PLC source codes automatically. Then the users can check certain requirements on this model that are extracted from the informal specifications [4].

Challenge 1.1 It is difficult to use various model checkers without extensive knowledge. *Our solution*: PLCverif translates the source code to an intermediate verification representation (IM). Wrappers for various model checkers (e.g. NuSMV, ITS) take care of the concrete representation and the execution of the tools [4], thus these tools remain hidden.

Challenge 1.2 Even if the models are generated from the source code automatically, they are often huge, making the verification impossible. *Our solution*: PLCverif includes generic and domain-specific reduction rules that can efficiently reduce the verification run time without altering the result.

Challenge 1.3 Most often no formal requirements are available, and it is difficult to use temporal logic to describe the requirements that the developer wants to check. *Our solution*: Requirement patterns are provided to help the user. They bridge the gap between the intuitive, informal requirements and the requirements formalised in temporal logic [4, 5].

PLCverif provides full support for this use case, the complete workflow is automated and hidden from the user. Based on the source code and the filled requirement patterns, the model checking is automatically performed and a verification report is produced.

2.2 Checking Conformance with Specification (UC2)

The second use case considers a next step, when a lightweight, behaviour-oriented formal specification is available besides the manual implementation. Our specification language, PLCspecif[2] [1] can help the users to express their detailed requirements precisely. Then our toolchain provides a way to automatically generate IM representations of the implementation and the specification. Their behaviours are then compared using a model checker tool. The counterexamples can show traces where the behaviours differ. This method provides more precise analysis, but it requires a formal specification in addition to the implementation.

Challenge 2.1 The formal specification methods are typically based on complex mathematical notations. It is often difficult to express typical PLC behaviours (e.g. edge-driven signals, occurrence of various events at the same time) [2]. *Our solution*: The PLCspecif specification method builds on semi-formal languages (state machines and data-flow diagrams) that are known by the developers. PLCspecif assigns a unified, formal semantics to these semi-formal languages that is adapted to the PLC and ICS domain [1].

Challenge 2.2 The widely-known conformance relations are too strict, therefore they identify many differences which are not relevant in practice. These "practically false positives" (acceptable differences) are undermining the usability. *Our solution*: New, more permissive conformance relations and algorithms to check their satisfaction were defined for PLC artefacts. They allow the user to focus on differences that are interesting in practice based on our experience.

[1] http://cern.ch/plcverif/
[2] http://cern.ch/plcspecif/

2.3 Code Generation Based on Formal Specification (UC3)

The third use case, similarly to UC2, assumes that a formal, detailed behaviour specification is available. Based on that the implementation can be generated automatically. The code generation based on PLCspecif is designed to have a semantics matching to the PLCspecif formal semantics.

Challenge 3.1 The generated code should be understandable. As for availability reasons it might be needed to modify parts of the PLC programs while the PLC is in operation, the code should be open to manual modifications. *Our solution*: The code generation of PLCspecif can be customised, adapted to the local coding conventions.

Challenge 3.2 The generated code should have the behaviour described by the specification. *Our solution*: The code generation is defined based on the formal semantics of the specification and their equivalence can be checked.

Challenge 3.3 Certain requirements are difficult to be described using "imperative" specifications, e.g. a state machine or a data-flow diagram. *Our solution*: PLCspecif allows the users to describe additional invariant and safety properties using requirement patterns. Based on the formal semantics of PLCspecif, model checkers can analyse the satisfaction of the invariant properties directly on the specification.

In light of this use case, UC2 might look rarely beneficial. The UC3 method not only provides the same level of correctness as UC2, but it reduces the manual resource needs by generating the implementation. On the other hand, UC3 alters the usual development process that can be blocker for acceptance at the moment. UC1 and UC2 can be used independently from the regular development process as additional steps. Furthermore, in certain cases the usage of some languages and code generation methods are forbidden, thus UC3 is not applicable [3].

3 Conclusion

Based on our model checking and formal specification solutions for PLC programs, we drew up three different use cases. They show that our verification and specification solutions are adaptable, and they can be introduced step by step. All these solutions aim to be easy to use in practice, thus their usage should not need an extensive training. They have different advantages and prerequisites, therefore for each project the most appropriate one can be chosen and applied. At CERN we use all three: some in production [3], some for experimental projects.

References

[1] D. Darvas, E. Blanco, and I. Majzik. Syntax and semantics of PLCspecif. Report EDMS 1523877, CERN, 2015. https://edms.cern.ch/file/1523877/.

[2] D. Darvas, I. Majzik, and E. Blanco. Requirements towards a formal specification language for PLCs. In *Proc. 22th PhD Mini-Symposium*, pages 18–21. BUTE, 2015. DOI: 10.5281/zenodo.14907.

[3] D. Darvas, I. Majzik, and E. Blanco. Formal verification of safety PLC based control software. In *iFM 2016*, volume 9681 of *LNCS*. Springer, 2016. DOI: 10.1007/978-3-319-33693-0_32.

[4] B. Fernández, D. Darvas, E. Blanco, et al. Applying model checking to industrial-sized PLC programs. *IEEE Trans. Ind. Informat.*, 11(6):1400–1410, 2015. DOI: 10.1109/TII.2015.2489184.

[5] B. Fernández, D. Darvas, J-C. Tournier, E. Blanco, and V.M. González. Bringing automated model checking to PLC program development – A CERN case study. In *12th Int. Workshop on Discrete Event Systems*, pages 394–399. IFAC, 2014. DOI: 10.3182/20140514-3-FR-4046.00051.