What is the method in applying formal methods to PLC applications?

Angelika Mader*and Hanno Wupper
Computing Science Institute, University Nijmegen, The Netherlands, email: mader | wupper @cs.kun.nl

Abstract

The question we investigate is how to obtain PLC applications with confidence in their proper functioning. Especially, we are interested in the contribution that formal methods can provide for their development. Our maxim is that the place of a particular formal method in the total picture of system development should be made very clear. Developers and customers ought to understand very well what they can rely on or not, and we see our task in trying to make this explicit. Therefore, for us the answer to the question above leads to the following questions: Which parts of the system can be treated formally? What formal methods and tools can be applied? What does their successful application tell (or does not) about the proper functioning of the whole system?

Keywords: formal methods, PLC, embedded systems, verification, testing, specification

1 INTRODUCTION

The question we investigate is how to get PLC applications with confidence in their proper functioning. Especially, we are interested in the potential contribution from formal methods for their development.

They way we approach this question is via a schematic presentation of an overall picture of PLC applications (see figure 1). We assume that this schema in some form or other is familiar to everyone working in the field of PLC applications. Making it explicit helps us to classify our own and others' existing work, evaluating its relevance, and identifying new research questions that could lead to useful contributions.

PLC applications typically consist of a variety of different parts: an environment that has to be controlled (e.g. a plant), a PLC, a PLC program, connections between a PLC and its environment, connections between a PLC and a PC, etc. During its design a PLC application is influenced by the programming environment on a PC, a compiler, etc. A PLC application only works correctly if all its parts and their interactions work correct.

A methodical development of PLC applications can be supported by formal methods such as program synthesis, a posteriori verification, formal testing, specification and program design methods. All of them have their limits, which are e.g. the complexity of the system, the simplification of reality, and the inherent restriction to small segments of reality. However, we are optimistic about the usefulness of formal methods: we strongly believe that it is the only way to get reliable applications. Our maxim is that the place of a particular formal method in the total picture of system development should be made very clear. Developers and customers ought to understand very well what they can rely on and what not, and we see it as our task to try to make this explicit. Therefore, for us the answer to the question above leads to the following questions:

- Which steps in the development process can be supported by formal methods?
- What formal methods and tools can be applied?
- What does their successful application tell (or not) about the proper functioning of the whole system?

Although many PLC-specific issues are addressed, our results are not confined to PLCs. Whenever possible, we try to present them in such a general way that they can also be used for the development of other embedded software.

The paper is structured as follows: in section 2 we sketch a schema of PLC applications, i.e. the components of PLC applications, their different abstraction levels and the interconnections between them. Referring to this schema, we describe in section 3 formal methods that support development and validation of PLC applications.

2 A SCHEMATIC PRESENTATION OF PLC APPLICATIONS

In this section we sketch an overall schema of PLC applications (see figure 1). Using this schema we motivate and discuss different abstraction levels on which PLC applications can be treated, and their interconnections. Each formal method applies to a fragment of the whole system at a certain level of abstraction. The result obtained using a particular method therefore also has to be interpreted relative to the relevant fragment and abstraction level.

System requirement: Specification of the goal.

A specification of the overall goal of a system or at least of its relevant properties is a prerequisite to make any

*supported by a NWO postdoc grant and the EC LTR project VHS (project nr. 26270)

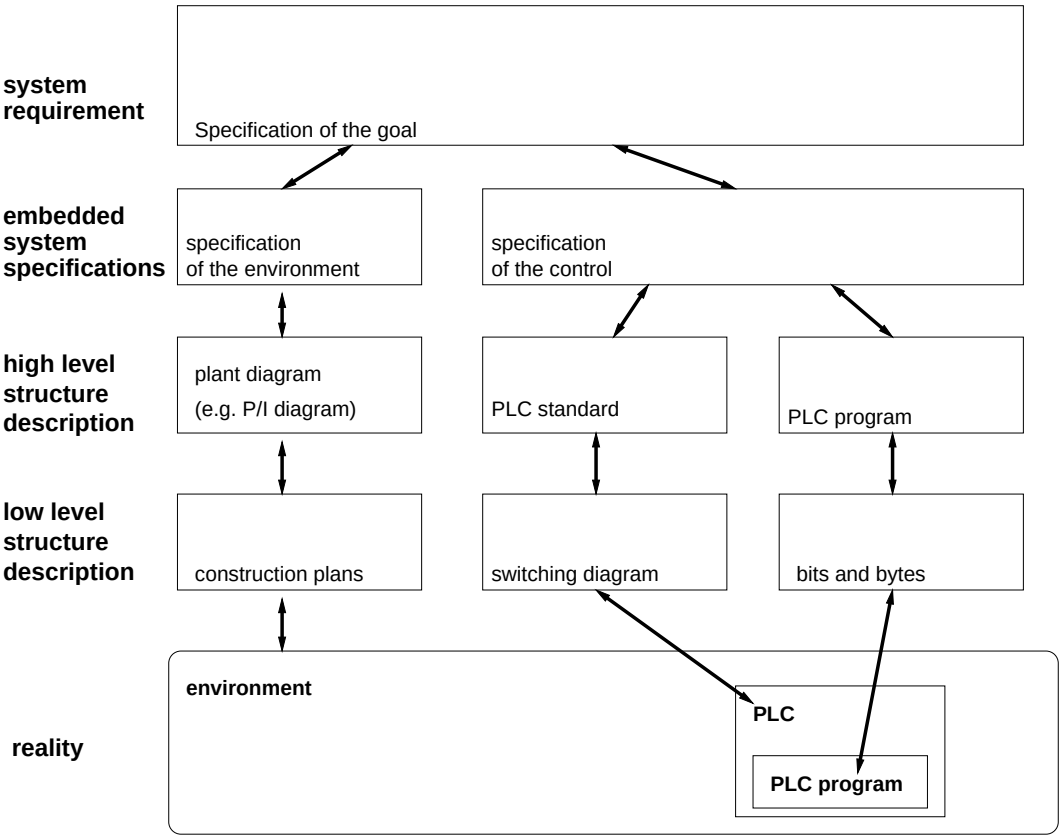

Figure 1: Abstraction levels of PLC applications

useful statement about its correctness. Ideally, such a specification is formal and exact, if we want to apply formal methods. On the other hand it should be clear and simple, in order to be readable to a customer or at least to a domain expert. In practice, such specifications are hardly ever given. Methods that allow to derive them are a research topic.

Embedded system specifications.

An embedded system consists of a program, a computer on which the program runs, and an environment to be controlled (a machine, vehicle, aircraft, or a whole factory). Correctness of one component is only meaningful with respect to the other components and with respect to a given overall goal. On the most abstract level this means that a specification theorem must hold of the following form:

$$(*) \quad \mathcal{SPEC}_{\text{environment}} \wedge \mathcal{SPEC}_{\text{control}} \Rightarrow \mathcal{SPEC}_{\text{overall\_goal}}$$

Theorem $(*)$ describes a correctness question on the specification level: e.g., given a specification of an overall goal and a specification of the environment a control is only correct if it satisfies a specification that makes implication $(*)$ true.

Specification of the environment.

A specification should, as simply as possible, state only those assumptions about the environment that are necessary to establish that it, together with the specified control, meets the overall goal. The availability of such an environment specification is useful for the following reasons:

- It provides guidance for the derivation of the control specification.
- It can be used for purposes of simulation.

Specification of the control.

A specification of a controller should only state how the environment has to be controlled. It should not state anything about PLCs or PLC languages in particular. It should be sufficiently general so that any machine that satisfies it is acceptable as a controller in the given environment. The availability of such a control specification is useful for the following reasons:

- It allows to divide the correctness proof for the PLC program into two parts: correctness of the program w.r.t. the PLC standard and the control specification and correctness of the control specification w.r.t. the environment and the overall goal.
- It is a prerequisite for systematic development and can provide valuable guidance during the development of the program.
- It can be used for the construction of meaningful tests.

High level structure description.

In principle, from the descriptions of the lowest level machine program, the lowest level PLC and the environment it should be provable that the overall specification holds. In practice, this is far too complex to be feasible. Therefore, more abstract formal descriptions at higher levels have to be used.

Schema of environment.

Rather than reasoning about a physical plant with its pipes, bolts, and nuts of specific geometry, we might prefer something like a "piping and instrumentation diagram" that abstracts from irrelevant information. Being computer scientists, we must leave the verification of the actual machines with respect to such schemas to other engineers and believe that the appropriate pipes, bolts, etc. have been connected in the right way.

PLC program.

Rather than reasoning about bits we can reason in terms of higher-level formalisms. In the case of PLC programs this would be a text in one or more of the standard PLC programming languages. Of course, it must be clear that the actual bits in the actual PLC are a correct implementation of that program. This implementation will, by the way, be different for different PLC models. Within the framework of our research we assume that the PLC manufacturer provides compilers that correctly translate the higher level languages to machine code.

PLC standard.

If the PLC languages of the standard IEC 1131-3 [5] are to have the same semantics independent of the model of PLC, one single formal specification of all PLCs is needed. It should be the most important part of the PLC standard. Up to now, it does not exist. We must therefore base our research on as few assumptions as possible about a well-chosen fragment of the PLC languages, and make these assumptions explicit as a "parameter" of our results. Of course, it must be clear that any actual PLC is an implementation of this standard. Again, within the framework of our research we assume that the PLC manufacturer provides hard- and firmware that is a correct implementation of the standard.

Low level structure description.

Lowest level machine program.

The physical program in the PLC is some electromagnetic representation of a large number of bits in the PLC's memory. The corresponding mathematical model is the corresponding long sequence of zeros and ones.

Lowest level PLC.

A PLC is a huge and complex electro-magnetic circuit that forms a programmable computer, together with an operating system. Both can be modelled exactly by means of suitable formalisms.

Environment.

The system in which the PLC with its program is embedded can be complex mechano-electrical machine, or a chemical plant etc.. All relevant properties of the environment and its (relevant) behaviour can be modelled exactly.

3 Formal methods contributing to reliable PLC applications.

In this section we try to put formal approaches for PLC applications in the context of our schema of figure 1. Certainly, the classification below is not complete, and, moreover, there may be other useful classifications.

Before getting to the concrete formal approaches we want to consider the principal possibilities for validation. At a very basic level, we distinguish between four ways to get confidence in the correct behaviour of a system. Typically, in system development we rely in a combination of all of them.

Insight is based on the simplicity of a representation. Some of the PLC languages have been designed to support insight. In general, to provide insight for complex systems requires also a notion of hierarchy. In section 3.3, we will mention a method for control design that for one part is based on insight combined with a hierarchical decomposition.

Induction. If a phenomenon can be observed again and again, one may conclude that it will also observable in the future - unless one has overlooked some essential causalities. This has led to a sophisticated culture of experimentation in the natural sciences. In computer science systems are mainly *tested*, which in some cases is a rather poor application of induction. Testing theory attempts to help to construct meaningful tests and interpretations of their results. With respect to PLCs applications it will be discussed in section 3.2.

Deduction in its most precise form, logical reasoning and mathematical proofs, helps us to know for sure that something is the case. That is why correctness proofs, supported by model checkers and proof tools, are applied. This process is called verification. A basis for verification is a formal model of a real system. Naturally, each formal model is only an abstraction of a real system. The knowledge we obtain from such formal methods can never be more meaningful than the abstractions on which they were based. Only aspects of the reality that are taken into account in the model can be verified. Therefore, verification can help to eliminate only certain classes of errors.

Proving correctness for arbitrary programs is very difficult, because of its complexity. In order to deal with the complexity, hierarchical and compositional approaches can be applied ("divide et impera"). A precondition for such approaches is a structured formal description of a program that, in the ideal case, is obtained by a structured program development process. In section 3.1 we treat verification approaches for PLC appli-

cations.

Belief. Often, we simply believe that something is the case, without being able to establish its truth by insight, induction or deduction. In such cases it is essential to be conscious of what one believes and what not. Where system development is based on belief, this should be made explicit. Correctness proofs for PLC programs, for example, are based on the belief that the correct semantics of the PLC languages is built into correct compilers. On the other hand, testing of PLC behaviour mentioned earlier, on the other hand, can tell us something meaningful without having to believe in the correct implementation of the PLC languages.

3.1 Program verification.

Most work done in formal methods for PLC applications is done in the area of program verification. Here, we want to emphasize the relations of verification approaches with both, specification level and physical level.

Program verification is band on the high level structure description and its relations to the embedded system specification or the system requirement. Typically, from a high level description of the program a model of the behaviour of the program is derived. In most cases we believe that the modelled behaviour coincides with the real behaviour in the reality. If a semantics of a high level description is defined there are two questions to answer: does the model match the semantics, and do the compiler and hardware behave according to the semantics? For PLC languages semantics are defined in the standard IEC 1131-3 [5]. However, from a formal point of view, these semantics are to some extent ambiguous and incomplete. Therefore, it is necessary for program verification to restrict to the "safe" fragments of the languages, i.e. fragments whose semantics are unambiguous. For most models introduced in the literature the question whether the they match with the semantics is answered on basis of insight. Furthermore, we trust in the manufacturers for having produced hardware and compilers according to the semantics.

We distinguish three levels of program verification: verification of a program together with its environment with respect to the system specification, verification of a program with respect to the control specification, and static program analysis.

In **static program analysis** errors such as unreachable code, forbidden control structures in SFCs, etc. can be detected. Here, a program specification is not required explicitly, but a detected error indicates that a program is not an implementation of any "reasonable" specification. An example of this approach can be found in [2].

Having verified a **program against the control specification** still leaves the correctness question of the specification open as given by theorem $(*)$ in section 2. In most cases not a complete program specification is considered, but some properties that seem to be relevant are verified, e.g. with help of a model checker. In this case one should make sure that the properties checked are implied by the control specification. Moreover, one should be aware of that it may be difficult to obtain a proof of a complete specification by means of model checking: the number of properties to verify can possibly be too large. Therefore, model checking is often used like a debugger: the property checked describes one kind of errors.

Most existing work about verification of PLC programs is done in this class (see e.g. [9]).

A special case is, as in static program analysis, to check "specification-independent" properties as e.g. in [1]. There, a property that is verified is the absence of unwanted history. If this property does not hold this might be an indication that the program is not an implementation of any "intended" specification.

A further restriction is not to verify a whole program, but only parts of it, such as the function blocks used by the program. Advantage here is that a function block used several times only needs to be verified once.

The most general case is **verification of a program together with its environment with respect to the system specification**. There, a model of the program together with a model of the controlled system are shown to satisfy the overall goal or relevant properties implied by the overall goal.

For all these levels of verification the choice and derivation of a certain model or class of models raises the following questions.

- First of all, in present examples the process of deriving a model is mainly based on experience and intuition. The first verification attempts have mainly the function to correct errors in the model. At a certain point when the model designer *believes* that the model is correct the results of verification are interpreted with respect to the physical system. A discussion of this aspect can be found in [3].

- Second, there are many classes of useful models, such as described by timed or hybrid automata, Petri-nets, process algebras etc. A survey can be found in [9] and in the long version of [7]. It is useful to have different classes of models available: a model needs not contain more information than the property to verify requires. Consequently, given a property it would be best to use always the most abstract model that allows for verification. An easy example is that an untimed property may not require a timed model. The question that arises is about the relation between different models. Its answer provides a basis to make use of different models during one verification process. In [7] we report on criteria that classify PLC applications and allow to find suitable models for each class and property to be verified. The goal of that

work is to make the derivation process for PLC models clearer.

- Originally, PLCs had a very simple architecture, operating system and programming languages. Meanwhile, they have become almost as complex as "usual" computers: they allow multi-tasking, interrupts, cyclical or periodical program execution, languages with complex features, etc. In this context it is anyway questionable whether the models we use can capture the complexity of the systems considered.

3.2 Testing PLC applications.

The authors are not aware of work done on formal methods based testing of PLC applications. However, for synchronous models of PLC programs, as e.g. in [6], the coupling to the synchronous theory and tool machinery inclusive testing approaches should be relatively straightforward.

Testing is the only validation method that can be used to find out whether a real, physical object (plant or control) satisfies its specification. Hence, all arrows connecting to the lowest level in figure 1 can be justified only by testing methods. In principle, testing approaches are possible between the reality level on one hand, and each of the higher levels on the other hand. For an introduction to testing using formal methods see e.g. [11].

Testing the integrated embedded system (reality) against the system requirement is known as *conformance testing*. For several PLC applications, e.g. control of chemical or nuclear plants, it is dangerous to test the whole system. To overcome this problem there are (at least) two possible ways: one is to substitute the environment by a simulation. This approach introduces new correctness questions, i.e. does the simulation capture the relevant properties of the physical plant? Another possibility is reducing the object under test to the PLC program and testing it against the program specification. In this case (as with verification approaches) we need to be sure about the correctness of the program specification: does the program specification together with an environment specification imply the system requirement.

It seems to be the case that for PLC applications several different testing disciplines could provide useful methods:

- A typical application area for PLCs is process control. In this context questions such as *are the processes activated in the right order?* are relevant. In the area of **protocol testing** much research is done with respect to testing of control structures. The results obtained there could be useful for PLC applications.

- Often it is useful to consider the input/output behaviour of PLC programs. One example is a whole scan cycle, where input consists of the "old" state of memory together with the new data on the input points, and output consists of a new memory state and new values on the output points. Also on a higher level, the one of function blocks and functions, it is reasonable to investigate the input/output behaviour. With this point of view results from **testing of software systems** could be transferred to PLC applications.

- Finally, also **hardware testing** may be useful for PLC applications. One difficulty of programming PLCs is due to the fact that for input and output variables hardware addresses are used, which easily leads to errors. Testing that the internal variables are connected in the correct way to the hardware addresses and furtheron to the input and output points provides certainly useful information. This is similar to boundary scan testing for hardware, where e.g. pins of chips are tested for being connected correctly.

3.3 Program derivation.

Program derivation is a long investigated subject in computer science. In the context of PLCs we want to mention two approaches.

The work of [4] introduces a method to generate PLC programs starting from a control specification in duration calculus. There exists a tool [10] supporting this process. Concerning the general picture, still the correctness question of theorem $(*)$ is open. However, the tool [10] also allows to generate timed automata models of the program and verification techniques can be used to show that program (model) and environment (model) satisfy the overall specification.

In [12] we investigated the design process of the specifications. We suggested a method that is based on on theorem $(*)$. The goal is that in the end such a theorem can be proved. To get such a theorem we use an approximative method. Starting point is a very general statement about the system requirement, which may be too strong initially. During the approximation process the system requirement is weakened stepwise as information about the controlled system and the control is added. This way of introducing information supports insight in the sense discussed earlier. We applied the method in a PLC case study [8]. It turned out that even when not striving for a complete formal proof that the program implements the control specification, the understanding of the overall problem we obtained by formulating the hierarchy of theorems gives guidance in designing the program.

4 Conclusion

In this paper we have raised the question what the role of formal methods for increasing confidence in PLC applications is. Of course, we have given only a partial answer. There are many possible applications of formal

methods for validation not mentioned here, such as simulation, synthesis, verification and testing approaches on other fragments of the whole picture, or other specification approaches. However, we believe that the structure presented helps to find out which aspects of the whole picture are treated by a certain method and how to evaluate results gained by formal methods with respect to the rest of the picture. Relating the structure presented with more existing and possible formal approaches is ongoing work.

The necessity of formal descriptions is obvious for (formal) validation. Moreover, we claim that they are also necessary for the design process: a control design can only be as good as the environment specification (environment description) is. Often, in control design, these descriptions are implicit. It is also obvious that making them explicit increases insight and the quality of the control program.

Additionally, we claim that formal descriptions are only useful if they are simple enough. One reason is that only then they go together with insight. Another reason is that the tools available cannot deal with arbitrary unstructured complexity ("spaghetti programs"), but can be useful for well-structured problems.

Finally, we are convinced that the usefulness of any formal method for industrial size systems is determined by the availability of tools supporting the method.

ACKNOWLEDGMENT

The authors want to thank Ed Brinksma for useful discussions, Judy Romijn and the referees for their critical remarks.

REFERENCES

[1] A. Aiken, M. Fähndrich, and Z. Su. Detecting races in Relay Ladder programs. In *Proceddings of TACAS'98*, volume 1384 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 1998.

[2] Sébastien Bornot, Ralf Huuck, Yassine Lakhnech, and Ben Lukoschus. Utilizing static analysis for programmable logic controllers. In *ADPM 2000: 4th International Conference on Automation of Mixed Processes: Hybrid Dynamic Systems, September 18-19, 2000, Dortmund, Germany*, 2000.

[3] Ed Brinksma. Verification is experimentation! In *Proceedings of CONCUR 2000*, Lecture Notes in Computer Science, 2000.

[4] H. Dierks. Synthesising controllers from real-time specifications. In *Proceedings of Tenth International Symposium on System Synthesis*, pages 126–133. IEEE CS Press, 1997.

[5] International Electrotechnical Commission. *IEC International Standard 1131-3, Programmable Controllers, Part 3, Programming Languages*, 1993.

[6] Fernando Jiménez-Fraustro and Eric Rutten. Hybrid simulation of IEC-61131 PLC programs using Signal and Simulink. In *Proceedings of the 4th International Conference on Automation of Mixed Processes, ADPM'00*, 18-19 September 2000, Dortmund, Germany, 2000.

[7] A. Mader. A classification of PLC models and applications. In *WODES 2000: 5th Workshop on Discrete Event Systems, August 21-23, 2000, Gent, Belgium*, 2000.

[8] A. Mader, E. Brinksma, H. Wupper, and N. Bauer. Design of a PLC control program for a batch plant- VHS case study 1. submitted for publication, 2000.

[9] O. Rossi, J. J. Lesage, and J. M. Roussel. Formal validation of PLC programs: a survey. In *Proceedings of European Control Conference 1999 (ECC'99)*, 1999.

[10] J. Tapken. Moby/PLC - A Design Tool for Hierarchical Real-Time Automata. In *Proceedings of FASE'98*, volume 1382 of *Lecture Notes in Computer Science*, pages 326–329. Springer Verlag, 1998. available at: http://theoretica.Informatik.Uni-Oldenburg.DE/ moby/.

[11] J. Tretmans. Testing concurrent systems: A formal approach. In J.C.M Baeten and S. Mauw, editors, *CONCUR'99 – 10^{th} Int. Conference on Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 46–65. Springer-Verlag, 1999.

[12] H. Wupper and A. Mader. System design as a creative mathematical activity. Technical Report CSI-R9919, University of Nijmegen, 1999.