ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP01

PLCverif RE-ENGINEERED: AN OPEN PLATFORM FOR THE FORMAL ANALYSIS OF PLC PROGRAMS

D. Darvas, E. Blanco, CERN, Geneva, Switzerland
V. Molnár, Budapest University of Technology and Economics, Budapest, Hungary

Abstract

Programmable Logic Controllers (PLC) are widely used for industrial automation in industry and at CERN. The reliability of PLC software is crucial, but typically only testing is used to validate it. Our work targets the use of formal verification in practical ways for many years, which showed that it can be beneficial and practically applicable to various PLC programs. In this paper, we present PLCverif, our platform for formal analysis of PLC programs which has largely enhanced the quality of the deployed PLC software. By re-engineering the previous internal prototype tool, we built PLCverif to be an open, extensible platform that can be used not only for CERN's specific PLC programs. PLCverif is licensed under an open source license, allowing the interested parties to use and extend it.

INTRODUCTION AND MOTIVATION

Programmable Logic Controllers (PLCs) are widely used to implement process control systems and interlock systems. The incorrect behaviour of PLCs can cause service disruptions, consequently significant financial losses and injuries too in some cases, therefore ensuring their correct behaviour is essential.

Testing (mainly acceptance or system testing) represents the state of the art in PLC software quality assurance. While testing is effective in finding certain types of errors, it is often not sufficient as the sole verification method. Testing cannot be exhaustive, thus cannot prove the correctness of a system. In addition, it is very difficult to test the following types of requirements: *safety* (the system will never reach an unsafe state) or *invariant* (formulas which shall be true over all possible system run), as these errors may occur in very particular cases only.

Model checking can overcome some of the weaknesses of testing. This is a formal verification technique, which checks the satisfaction of a formalised requirement on a mathematical model of the system under analysis. It checks the requirement's satisfaction with every input combination, with every possible execution trace. In addition, if a violation is found, typically a trace leading to the violation (a counterexample) is provided. However, model checking is difficult to use by PLC developers who are not experts in this domain. The target of our work is to make model checking more practically applicable to the software of PLC-based systems by hiding the formal details, simplifying the user interaction and automating the process.

Our work is not the first that targets the formal verification of PLC programs. Among others, Arcade.PLC [1] and the toolset developed by VTT Technical Research Centre of Finland [2] both offer PLC program verification. However, none of the publicly available tools was applicable to the real-world PLC programs used at CERN, due for example, to the lack of support for the Siemens SCL language. In addition, we did not find possible to extend or adapt these tools for our use cases.

Previously, we have presented a methodology for practical model checking of PLC programs [3], a prototype tool that implements this workflow [4], as well as real-life case studies where model checking was proven to be beneficial [5, 6]. This paper reports about our re-engineering efforts done during the last two years and presents the final tool officially. This development made PLCverif richer in features, more robust and open to extensions. In addition, the paper discusses how *we* did benefit from PLCverif and how *users* can adapt it to their use cases.

DEVELOPMENT OF PLCverif

The first plans to evaluate the use of formal verification to PLC programs at CERN date back to 2012. After the initial experimentation phase, the design and development of the methodology used in PLCverif started in mid-2013. Within a year, a prototype version of PLCverif was developed. Already during the development, PLCverif was used to analyse parts of systems in production.

This prototype version was sufficient for our internal use cases. However, to make PLCverif more generally applicable, it had to be more robust, more generic and more extensible. To obtain the resources needed for this additional development, a CERN Knowledge Transfer Fund was requested and awarded in 2016. The proposed two-year-long re-engineering project was selected to be funded in mid-2016 [7]. The development project started in June 2017 and ended in May 2019. During that time, PLCverif was rebuilt from scratch, taking the previous experiences and knowledge into account.

The goal of this re-engineering work was to make PLCverif usable by any automation engineer at CERN and other interested parties outside the organisation.

PLCverif FOR USERS

This section discusses the principal use case of PLCverif from the users' point of view.

Verification Workflow

Out of the box, PLCverif offers a model checking workflow for the analysis of PLC programs. The verification workflow is shown in Figure 1 and it has the following main steps:

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP01

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

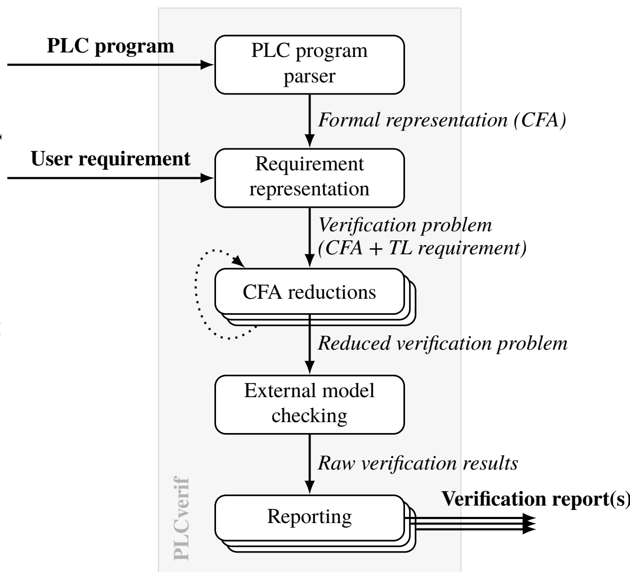

Figure 1: Formal verification workflow of PLCverif.

1. **PLC program parsing.** First, PLCverif parses the PLC program (located in one or several files) to be analysed. By choosing the entry point of the verification, the analysis can be limited to a part of the program. The parsed PLC program is automatically translated into a mathematical, control flow-based representation, producing so-called control flow automata (CFA). This precise description will serve as the base for analysis.

   Currently, the Siemens STL and SCL V5.3 input languages are supported[1]. PLCverif provides an editor as shown in Figure 2.

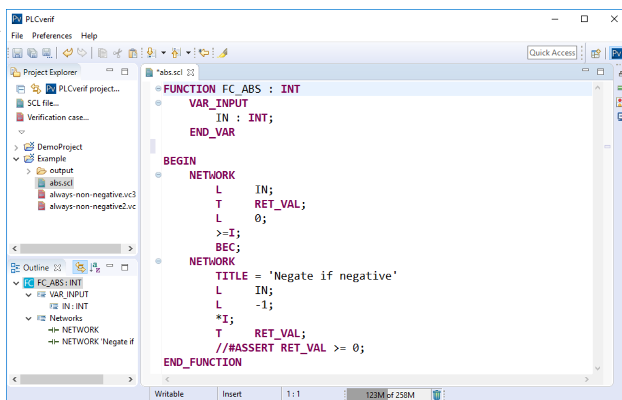

Figure 2: STL code editor in PLCverif.

2. **Requirement representation.** The user should describe the precise requirement to be checked. This, however, does not mean that the user needs to describe the requirement using mathematical formulae. Currently, two requirement description methods are supported out of the box:

   - Assertion-based requirements: special comments in the source code (e. g. `//#ASSERT On <> Off`) can describe expressions (invariants) which are expected to be always satisfied at a given location of the program. The verification job will then check if the violation of any of the selected assertions is possible.
   - Pattern-based requirements: the user chooses a requirement pattern that is a precisely phrased plain text sentence with some placeholders, e. g. "*If α is true at the end of a PLC cycle, β shall be true at the end of the same cycle.*". The gaps in the requirement pattern (α and β in the previous example) should be filled with expressions over the PLC variables. For each requirement pattern, a defined temporal logic representation is defined which will be used in the next steps.
   - If needed, new types of requirement representations can be defined, adapted to the specific needs.

3. **CFA reductions.** The formal, precise CFA representation of the program, including also the requirement, may need to be reduced in order to make the verification feasible and efficient. These reductions will not change the verification result for the given requirement; however, they may remove parts of the program which do not influence the result of the currently checked requirement.

4. **External model checking.** The model checking itself is performed by widely used model checker tools. In this step, *(i)* the reduced CFA will be translated into the input syntax of the chosen model checker tool, *(ii)* the model checker tool is executed, and *(iii)* its output, notably the counterexample if available, is parsed to PLCverif's internal representation.

   Currently the following external model checkers are supported: NuSMV [8], nuXmv [9], Theta [10] and CBMC [11]. The different model checkers have different strengths and weaknesses. In addition, not every feature is supported by every model checker. For example, Theta does not support bitwise operators. CBMC often provides good performance, but it is a bounded model checker and this may cause false negative results. If needed, new model checkers can be integrated into PLCverif easily.

5. **Reporting.** The last step of the formal verification workflow is to produce verification reports. Some of these reports are in human-readable form and target the user of PLCverif. Other reports are machine-readable and serve as descriptions for the execution environment or as artefacts for later summary reports. Figure 3 shows an example HTML verification report.

The verification workflow is guided by a *verification case*, which describes the code and the requirement to be analysed,

[1] The novelties introduced in Siemens TIA Portal are not supported by PLCverif yet.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP01

PLCverif — Verification report

Generated on 2019-03-26 07:48:49 | PLCverif v3.0 | (C) CERN BE-ICS-AP | Show/hide expert details

ID:	always-non-negative
Name:	Absolute value is always non-negative
Description:	The return value of the absolute value function is always non-negative
Source file(s):	C:\ICALEPCS2019_PLCverif\workspace\Example\abs.scl
Requirement:	None of the assertions is violated.
Result:	Violated
Verification backend:	NusmvBackend (nuxmv-Classic-dynamic-df)
Total run time:	262 ms
Backend run time:	194 ms

Counterexample

	Variable	End of Cycle 1
INPUT INT	FC_ABS.IN	-32768
signed int16	FC_ABS.RET_VAL	-32768

Diagnosis

The assertion 'RET_VAL>=0' has been violated in Cycle 1.

Show/hide more details

Figure 3: Verification report.

the verification tools and reporters to be used. Additionally, it permits the fine-tuning of every element in the verification workflow. PLCverif provides a convenient graphical editor for the user to describe the verification case, as shown in Figure 4.

New Features with Respect to the Prototype Tool

At first glance, the feature set of PLCverif looks similar to the feature set of the prototype tool, discussed in [4]. Hence, we highlight the main novelties from the user point of view below.

- Native support of STL code, together with an editor, content assist, validation, etc.
- Support for assertions in the SCL and STL programs and formal assertion violation checking.
- Integration of new model checkers (CBMC and Theta). CBMC is a software model checker for C programs [12], thus a C representation of the CFA is produced for verification purposes by PLCverif. Theta is a framework for abstraction refinement-based model checking [13], including several state-of-the-art algorithms.
- Full command line support. Every verification performed via the graphical interface can also be done through the command line interface. This permits the automation of using PLCverif, for example the automatic re-execution upon every requirement change or addition. This is an essential feature to integrate PLCverif in a continuous integration practice.
- Extensibility. Most features of PLCverif can be extended or adapted using plug-ins, even by 3rd parties, without any modification required in the platform itself. This is discussed in detail in the next section.

Verification case

Metadata
General description of the verification case.
ID: always-non-negative2
Name: Absolute value is always non-negative
Description: The return value of the absolute value function is always non-negative, if the input is not -32768.

Source files
Here the scope of the verification (i.e., the included source files) needs to be selected.
Source files: ☑ abs.scl ☐ *.scl (all scl files in this project's root)
Reload source files
Language frontend: STEP 7
Entry block: FC_ABS

Verification backend
Selection and configuration of the external verification tool to be used.
Backend: NuSMV
Algorithm: (default)
Advanced settings

Requirement
Description of the requirement to be verified.
Requirement type: Pattern requirement
Pattern: If {1} is true at the end of the PLC cycle, then {2} should always ?
1: FC_ABS.IN ≠ -32768
2: FC_ABS.RET_VAL ≥ 0
If **FC_ABS.IN ≠ -32768** is true at the end of the PLC cycle, then **FC_ABS.RET_VAL ≥ 0** should always be true at the end of the same cycle.

Requirement – advanced

Reporters

Advanced settings (0)

Verify
Everything is ready? Buckle up and hit the 'Verify' button!
Verify!
Last result: Satisfied
Last execution: 2019-03-26 07:47:47
Last duration: 189 ms
Open report

Figure 4: Verification case editor.

PLCverif FOR DEVELOPERS

The majority of the re-engineering work done is only visible for the developers. Here, *developer* does not only mean the core platform developers, but also those who may eventually extend and adapt PLCverif to their verification needs.

It was known from the beginning of the project that it is not possible to cover all the potential needs with PLCverif: every potential PLC language, every requirement representation method, every external model checker tool, every verification report format, etc. Therefore, PLCverif was designed as a generic, open (PLC) code analysis platform. This section overlooks the main features of this platform and its extensibility.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP01

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

PLCverif Platform Features

At the highest level, PLCverif is a program analysis platform that provides three main components:

- Metamodels and common data structures to describe the programs under analysis and to provide a common "language" between the different parts of the defined workflows,
- An interface for a program analysis job, i. e. a workflow that takes a formal representation of a PLC program and produces some artefacts,
- A homogenized way to handle (read, load, save) the settings of each part of the platform.

Metamodels PLCverif contains an *expression metamodel* to describe logic, arithmetic and temporal logic (Computation Tree Logic and Linear Temporal Logic) expressions. These expressions can be used in requirements and in the CFA representations of the PLC programs as well. An expression parser and an expression editor are both included too to make it easier to handle the textual representation of the expressions.

An automata-based representation of PLC programs is defined by the platform too. Briefly, in an automaton, *locations* and *transitions* represent the behaviour of the program. The transitions may have *guard conditions* and *actions* (typically variable assignments). In an automaton, at most one of the locations is active. A transition, connecting a source and a target location, may *fire* if its source location is active and its guard condition is satisfied. After firing, its target location will become active and the associated actions will be executed. Several automata can be grouped together in a *network*, and special *call transitions* allow their interactions.

Relying on the expression metamodel, two control flow automata metamodels are defined. A *control flow network declaration* (CFD) models the control flow automata at the level of declarations. One advantage of the CFD is that it is easy to generate a CFD for a PLC program, also it is easy to handle by model checkers whose input language is close to a general-purpose programming language. However, it is difficult to do reductions on this representation and most model checkers cannot handle the rich data structures and the need for instantiation.

This is why a second, *control flow network instance* (CFI) metamodel is included too. In a CFI, the automaton instances do not need instantiation anymore, the rich data structures are instantiated and flattened. This representation can be more efficiently reduced and it is much closer to the syntax of NuSMV and Theta for example. The instantiation of a CFD (the conversion to a CFI) is performed by PLCverif. In addition, a trace will be generated that establishes a link between the components of the result CFI and the source CFD.

Program analysis job The unit of work in PLCverif is an execution of a *job*. A job is a defined set of steps on a parsed PLC code that produces some artefacts (job results). The principal job of PLCverif is the *verification job*. It implements the workflow that was shown in Figure 1.

Settings loading and saving The *settings*, as defined in PLCverif, are a generic way to represent various information that is needed for the execution of a given job. They will describe for example the selected job and selected verification tool, the detailed settings of the verification tool, the timeout to be used, etc. Essentially every information that can influence the execution of a job is called a setting. The settings are stored as hierarchical key-value pairs.

Some of the settings will be defined for each individual execution (e. g. metadata of the current verification job, requirement to be checked). The effective settings consist of them merged with the default settings and the installation-specific settings (e. g. location of the external verification tools on a given machine). The effective settings can be saved to make the job execution reproducible.

The settings can be defined textually (in a file or as command line arguments), or by using graphical editors (e. g. the verification case editor shown in Figure 4 is such a graphical interface).

Example. The verification case shown in Figure 4 corresponds to the following settings.

```
-id = always-non-negative2
-description = "The return value of the absolute
     value function is always non-negative, if
     the input is not -32768."
-name = "Absolute value is always non-negative"
-sourcefiles.0 = abs.scl
-lf = step7
-lf.entry = FC_ABS
-job = verif
-job.backend = nusmv
-job.req = pattern
-job.req.pattern_id = pattern-implication
-job.req.pattern_params.1 = "FC_ABS.IN != -32768"
-job.req.pattern_params.2 = "FC_ABS.RET_VAL >= 0"
```

Note that the path to the NuSMV binary is not defined anywhere. This is taken from the installation specific settings, but could be included above explicitly too, in the following way:

```
-job.backend.binary_path = C:\NuSMV\nusmv.exe
```

As the verification cases (and other settings) are defined in such simple format, they can be easily generated if needed.

Extensibility

As mentioned earlier, PLCverif is designed to be an open, extensible PLC code analysis platform. The feature set can be extended with plug-ins that essentially implement given interfaces (more precisely, PLCverif plug-ins are Eclipse extensions, implementing given extension points). There are various extension points defined in PLCverif:

- The **job** extensions can describe and implement custom PLC program analysis workflows. They can define their own extension points too.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP01

- The **language frontend** extensions are responsible for parsing certain types of PLC programs and translating them into corresponding control flow automata declarations.
- The **CFA reduction** extensions implement various algorithms to make the formal models of the PLC programs smaller and therefore to improve the verification performance.

In addition, the *verification job* described before defines several extension points too:

- The **requirement representation** extensions are responsible to represent the requirements described by the user in a defined format into temporal logic representation. These extensions can also modify the CFA (e. g. by adding some monitors) in order to make the requirement representable in temporal logic.
- The **verification backend** extensions solve the verification problem (check a temporal logic requirement on the reduced CFA model) and provide the result of the verification. Typically, these plug-ins provide mapping from one of the CFA formalisms to the input syntax of the external model checker. Next, the model checker is executed and its output, notably the verification result and the counterexample if available, will be parsed.
- The **verification reporter** extensions represent a subset of the verification result in a human-readable or machine-readable format.

The extensions of the verification job may have graphical representations too, contributing to the verification case editor with custom editor components.

Further Extensions

In addition to the built-in extensions, which are part of the extension library, new extensions can be easily developed to adapt PLCverif to custom needs. For example, new requirement representation plug-ins can be developed to provide an interface for specific requirements. If needed due to some particularities of programs under analysis, new CFA reductions can be implemented. If the result is expected in a particular format, a new reporter plug-in can be implemented.

For example, the two latest custom extensions built: an implicit requirement representation plug-in that can be used for checking whether division by zero may occur in the program, and a verification reporter plug-in that can translate the counterexample to the format expected by one of our testing tools [14].

USE CASES

To demonstrate the usability of our tool, some of our recent success stories are listed below. This section aims to point to the reader to possible fields of application of the tool with real industrial installations.

ITER

In [6], we have already reported about our work targeting the verification of a PLC program that implements ITER's High Integrity Operator Commands (HIOC) protocol. Even though the re-engineering of the PLCverif platform has just started, we already benefited from the convenience of the assertion-based requirement description and the performance of CBMC. Although it is difficult to prove the correctness of a cyclic program using CBMC due to its boundedness, it was able to provide us interesting property violations quickly. This method complemented well the pattern-based requirement verification with NuSMV using our prototype tool.

Interlocks for Magnet Test Benches

CERN develops, maintains and operates various superconducting magnet test benches. The safety interlocks of these benches are typically implemented using fail-safe PLCs. PLCverif was used in the development of at least 5 different PLC-based interlock systems. In the frame of the collaboration with GSI Helmholtz Centre for Heavy Ion Research, CERN built a test facility for the superconducting magnets [15] to be used in the Super Fragmentation Separator (Super-FRS). As these magnet tests involve various risks (e. g. high voltage, high current, cryogenic fluids), the PLC-based interlock and personal protection system is critical. We have used PLCverif to analyse the fail-safe PLC program with respect to the semi-formal specification provided by our internal client.

The fail-safe PLC program, written in Siemens LAD language, is approximately 10,000 lines long when exported to STL. It contains about 120 (digital and analogue) inputs and 80 outputs.

The specification has been formalised mainly as assertions. In total, 163 assertions have been written or generated from the specification. Some of the generated assertions are very complex, up to 10,000 characters long. PLCverif was able to check the assertions corresponding to the key safety functions in less than 15 minutes. Verifying all formalised assertions takes more time, approximately 4 hours. As for different requirements different model checkers provided the best performance, we used all three of NuSMV, CBMC and Theta.

The formalisation of the requirements, the model checking results and the code reviews conducted consequently revealed important, interesting problems with varying severity. Attributed directly or indirectly to formal verification, 17 issues have been identified altogether. Some of these were critical flaws in the implementation of the fail-safe logic. The use of PLCverif helped to improve the quality of the interlock logic and to increase the confidence of both the client (domain experts) and the PLC program developer.

SPS Personnel Protection System

PLCverif is being applied to the Personnel Protection System (PPS) of the Super Proton Synchrotron (SPS) ac-

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP01

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

celerator [16]. The associated control system has a highly distributed architecture based on several Siemens S7-1500F PLCs which implement about 30 different Safety Functions. The PLC program has been designed with a configurable approach where a generic software is instantiated for each access zone. This generic safety program, written in LAD programming language, is structured in different modules corresponding to each of the specified Safety Instrumented Functions (SIFs). As a consequence of the configurable program design, the verification activities become very challenging as the PLC program contains many extra configuration variables which increases exponentially the number of combinations to explore. Initially, PLCverif has been applied to verify the individual generic modules of these PLC programs. The first verification campaign has shown very promising results and some safety critical discrepancies between the PLC program and the specification have been found. Currently all except one module could be successfully verified by PLCverif. Further investigations are needed to overcome the enormous state space of the created formal model and to be able to verify all the SIFs.

OPEN SOURCE LICENSING

There is a wade spectrum of software licences ranging from protective to public domain (e.g. GPL, MIT, BSD, EPL). The main idea behind the PLCverif project was to contribute improving the quality of the developed software to control industrial installations for all interested people. This naturally fits with an open and permissive license type. We analysed different options and confronted them with our needs which basically can be summarised as making a tool usable for a large community while allowing third-party contributors to include their latest research developments which could help to increase the functionality and performance of the tool.

The expressions of interest we have gathered during the project showed three main types of potential contributors:

- **Academia**: interested in closing the gap between research and practical applications of formal methods who identified a tool where they could try their latest research (e.g. reduction methods, model checkers, symbolic execution)
- **Industrial integrator**: interested in providing error-free industrial control systems to their clients
- **Project responsible**: interested in making a final and sound validation of the software to be deployed for their critical systems (e.g. safety instrumented systems)

In collaboration with the Knowledge Transfer (KT) group at CERN, we decided to release the PLCverif tool under the Eclipse Public License 2.0 (EPL [17]) which is similar to the GNU General Public License (GPL [18]) but allows to link the code under this licence to proprietary applications. This allows both, the use and the extension of the tool, even for commercial purposes. We believe that this choice will foster the collaboration between the interested contributors in all the three cases described previously.

The licence selection was finally narrowed to two possibilities: EPL and Apache [19] as they are permissive licences. This type of licence places few restrictions on users and often only requires that the original creators are attributed in any distribution or derivative of the software. Apache seemed most popular but, on the other hand, most of the PLCverif components are already under EPL license. Additionally the choice was appropriate taking into account that licensing in EPL imposes the condition to disclose the modifications done to any EPL code and if a component under EPL licence is distributed as part of a derivative in binary form, the modified source must be available as well. This is the main concept of copyleft licences and what guarantees perpetual open source of the work.

CONCLUSION

Our work on improving the quality assurance and complementing testing led to PLCverif, an automated formal verification platform for PLC programs. Already in the prototype phase, it had demonstrated its real-life usability in various CERN projects. The described used cases reinforce this fact and demonstrate its utility. The re-engineering work of the last two years made it possible to make PLCverif a robust and open platform for PLC program analysis. It provides strong verification capabilities for Siemens PLC programs out of the box, but it can also be extended by adding third-party plug-ins to support new types of program analysis, additional programming languages, more model checkers, reductions and verification report formats.

PLCverif demonstrated the possibility of using formal verification for real-life projects. In addition, it is a successful example of knowledge transfer at CERN, permitting to use the results of our development outside of CERN as well. We encourage the reader to download and try PLCverif from our website, `http://cern.ch/plcverif/` or the open source repository.

ACKNOWLEDGEMENTS

We thank the CERN Knowledge Transfer Fund for providing the generous financial support that made this re-engineering work possible and their precious help in finding the best strategy to make this project open source. We also thank the different collaborations and contacts along the project, notably the EPFL (Switzerland) and the University of Oviedo (Spain) during the initial phases of our research and the BME (Hungary) during the re-engineering phase.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP01

REFERENCES

[1] S. Biallas, J. Brauer, and S. Kowalewski, "Arcade.PLC: a verification platform for programmable logic controllers", in *Proc. 27th IEEE/ACM Int. Conf. on Automated Software Engineering*, 2012, pp. 338–341, doi:10.1145/2351676.2351741

[2] A. Pakonen, T. Matasniemi, J. Lahtinen and T. Karhela, "A toolset for model checking of PLC software", in *Proc. 2013 IEEE 18th Int. Conf. on Emerging Technologies & Factory Automation*, 2013, pp. 338–341, doi:10.1109/ETFA.2013.6648065

[3] B. Fernández, D. Darvas, E. Blanco, J.C. Tournier, S. Bliudze, J.O. Blech, and V.M. González, "Applying Model Checking to Industrial-Sized PLC Programs", *IEEE Transactions on Industrial Informatics*, 2015, pp. 1400–1410, doi:10.1109/TII.2015.2489184

[4] D. Darvas, E. Blanco Vinuela, and B. Fernández Adiego, "PLCverif: A Tool to Verify PLC Programs Based on Model Checking Techniques", in *Proc. 15th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'15)*, Melbourne, Australia, Oct. 2015, pp. 911–914. doi:10.18429/JACoW-ICALEPCS2015-WEPGF092

[5] D. Darvas, M. István and E. Blanco, "Formal Verification of Safety PLC Based Control Software", *Integrated Formal Methods, ser. Lecture Notes in Computer Science*, 2016, pp. 508–522, doi:10.1007/978-3-319-33693-0_32

[6] B. Fernandez Adiego *et al.*, "Applying Model Checking to Critical PLC Applications: An ITER Case Study", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 1792–1796. doi:10.18429/JACoW-ICALEPCS2017-THPHA161

[7] *Six new projects will bridge gap between CERN and society*, CERN Bulletin, 2016, pp. 24–25.

[8] NuSMV: a new symbolic model checker, http://nusmv.fbk.eu

[9] The nuXmv Model Checker, http://nuxmv.fbk.eu

[10] Theta repository, https://github.com/FTSRG/theta/

[11] CBMC, Bounded Model Checker, https://www.cprover.org/cbmc/

[12] E. Clarke, D. Kroening, and F. Lerda, "A Tool for Checking ANSI-C Programs", *Tools and Algorithms for the Construction and Analysis of Systems*, 2004, pp. 168–176, doi:10.1007/978-3-540-24730-2_15

[13] T. Tóth, A. Hajdu, A. Vörös, Z. Micskei, and M. István, "Theta: a Framework for Abstraction Refinement-Based Model Checking", in *Proc. 17th Conf. on Formal Methods in Computer-Aided Design*, 2017, pp. 176–179, doi:10.23919/FMCAD.2017.8102257

[14] E. Blanco Vinuela, D. Darvas, and Gy. Sallai, "Testing Solution for Siemens PLCs Based on PLCSIM Advanced", presented at the 17th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'19), New York, NY, USA, Oct. 2019, paper WEPHA018.

[15] E.S. Fischer *et al.*, "Superconducting Magnets at FAIR", in *Proc. 8th Int. Particle Accelerator Conf. (IPAC'17)*, Copenhagen, Denmark, May 2017, pp. 2546–2549. doi:10.18429/JACoW-IPAC2017-WEOCB2

[16] T. Ladzinski, F. Havart, and B. Fernandez Adiego, "Renovation of the SPS Personnel Protection System: A Configurable Approach", presented at the 17th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'19), New York, NY, USA, Oct. 2019, paper MOPHA078.

[17] Eclipse, https://www.eclipse.org/legal/epl-2.0/

[18] GNU General Public License, https://www.gnu.org/licenses/gpl-3.0.en.html

[19] Apache License, version 2.0, https://www.apache.org/licenses/LICENSE-2.0

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.