From Natural Language Requirements to the Verification of Programmable Logic Controllers: Integrating FRET into PLCverif

Zsófia Ádám[1], Ignacio D. Lopez-Miguel[3], Anastasia Mavridou[4], Thomas Pressburger[5], Marcin Bęś[2], Enrique Blanco Viñuela[2], Andreas Katis[4], Jean-Charles Tournier[2], Khanh V. Trinh[4], and Borja Fernández Adiego[2]

[1] Department of Measurement and Information Systems, Budapest University of Technology and Economics, Budapest, Hungary adamzsofi@edu.bme.hu
[2] European Organization for Nuclear Research (CERN), Geneva, Switzerland borja.fernandez.adiego@cern.ch
[3] TU Wien, Vienna, Austria ignacio.lopez@tuwien.ac.at
[4] KBR at NASA Ames Research Center, Moffett Field, CA, USA anastasia.mavridou@nasa.gov
[5] NASA Ames Research Center, Moffett Field, CA, USA tom.pressburger@nasa.gov

Abstract. PLCverif is an actively developed project at CERN, enabling the formal verification of Programmable Logic Controller (PLC) programs in critical systems. In this paper, we present our work on improving the formal requirements specification experience in PLCverif through the use of natural language. To this end, we integrate NASA's FRET, a formal requirement elicitation and authoring tool, into PLCverif. FRET is used to specify formal requirements in structured natural language, which automatically translates into temporal logic formulae. FRET's output is then directly used by PLCverif for verification purposes. We discuss practical challenges that PLCverif users face when authoring requirements and the FRET features that help alleviate these problems. We present the new requirement formalization workflow and report our experience using it on two critical CERN case studies.

1 Introduction

Over the past few years, formal verification has become a crucial part in the process of software development for critical applications. To this end, CERN's open-source tool PLCverif [3] opened the door for the verification of Programmable Logic Controller (PLC) programs [13,6,12] and it has successfully been used to verify several safety-critical applications [7,8,9].

Given PLC code and a set of requirements formalized in either computation tree logic (CTL) or future-time linear temporal logic (LTL), PLCverif automatically transforms these to an intermediate mathematical model, i.e., control flow automata (CFA) [5]. Once the intermediate model is obtained, PLCverif

Zsófia Ádám and Ignacio D. Lopez-Miguel – Work performed while at CERN.

supports its translation into the input language of various model checkers for verification. Finally, analysis results are presented to the user in a convenient and easy-to-understand format.

The aforementioned process relies on control and safety engineers formalizing requirements. Prior to this work, PLCverif already supported the use of natural language templates, which are a set of pre-made templates with "blanks" where expressions containing variables of the PLC program can be added (e.g., "{expression1} is always true at the end of the PLC cycle"). During formalization, pattern instantiations are translated to LTL or CTL based on the pre-made templates. However, the expressive power of the existing templates is limited.

In this paper, we present the integration of NASA Ames' Formal Requirements Elicitation Tool (FRET [1]) into PLCverif, which provides a structured natural requirements language with an underlying temporal logic semantics. The integration of FRET within PLCverif helps users express and formalize a greater range of requirements and understand their semantics. The toolchain was successfully used in two critical CERN case studies: a safety program of a superconducting magnet test facility and a module of a process control library.

2 Integrating FRET into PLCverif

FRET is an open source project for writing, understanding, formalizing, and analyzing requirements [1,10,11]. FRET's user interface was designed with usability in mind; engineers with varying levels of experience in formal methods can express requirements using a restricted natural language plus standard Boolean/arithmetic expressions, called FRETISH with precise, unambiguous meaning. For a FRETISH requirement, FRET produces textual and diagrammatic explanations of its exact meaning and temporal logic formalizations in LTL. FRET also supports interactive simulation of the generated logical formulae to increase confidence that they capture the intended semantics.

2.1 Limitations of Requirement Formulation in PLCverif

As already anticipated in the introduction, *patterns* have significant limitations:

1. they offer a *limited set of 9 pre-made templates* only,
2. they do not offer any tool for *validation* of complex requirements; i.e., methods for checking if the created requirement is the same as the intended one.

FRET is able to improve on the current limitations the following ways:

Expressive Power. Users are able to formulate requirements in FRETISH as constrained and unambiguous sentences, which are then automatically transformed to LTL expressions.

Validation. FRET has a built-in simulator, allowing the user to check different temporal variable valuations. Furthermore, FRET generates a textual and diagrammatic description of the requirement to further help precise understanding.

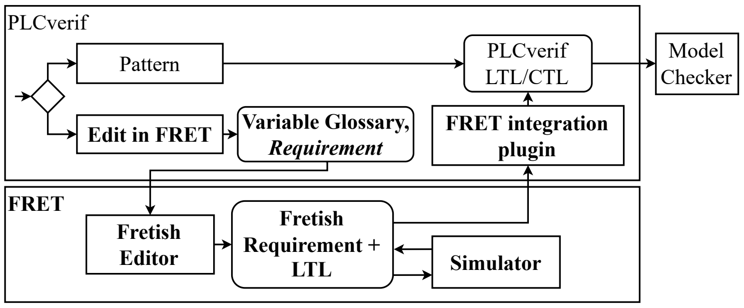


Fig. 1: Requirement formalization workflow in PLCverif. Features integrated into PLCverif in this work are shown in bold.

2.2 Realization of the New Workflow with FRET

The bold parts of Figure 1 show how FRET fits into the workflow of PLCverif. If "Edit in FRET" is chosen, FRET is opened and the user can work in the requirement editor. Once the requirement is written in FRETish and formalized, it is then sent back to PLCverif.

The External Mode of FRET. The feature set of FRET covers much more than what PLCverif could currently utilize (e.g., requirement hierarchies). PLCverif handles each requirement in a separate verification case, thus only the following features are utilized: the requirement editor, the formalization component and the requirements simulator. To facilitate integration with external tools including PLCverif, we developed a Node.js script for running the aforementioned features as a standalone tool. This new mode also implements the ability to import a variable glossary into the requirement editor of FRET.

Variable Glossary. PLCverif extracts the list of variable names and types by parsing the PLC code. Now the resulting *variable glossary* can also be exported to a JSON file for FRET to use. This enables features such as *autocompletion* of variable names in the FRET requirement editor. It also facilitates the process of creating formalized properties that can be directly used by PLCverif for verification as these requirement variables directly match PLC code variables.

Since variable names used in PLC code may include other characters besides alphanumeric, we extended the FRET editor to support identifiers with periods, percents, or double-quoted identifiers that can contain any special character.

The supported data types differ between the two tools (e.g., PLC programs might use arrays, while FRET only has scalar types). Assigning a data type to a variable is not mandatory in FRET and in this work it is only utilized by the simulator to show what possible values a variable can be assigned. Thus implementing a best-effort mapping proved to be adequate (e.g., sending each array element as a separate variable).

Preparing FRET for PLCs. The working principle of a PLC is described by the so-called *PLC Scan Cycle*, consisting of three steps: (1) read sensor values, (2) execute the PLC program and (3) write the actuator values. PLC requirements must be able to express cycles, e.g. "next cycle" or "at the end of the cycle".

To enable this, FRET's formalization algorithm was extended to express new built-in predicates `nextOcc(p,q)` and `prevOcc(p,q)`, and FRETISH was extended with the phrases *"at the next/previous occurrence of* `p`, `q`*"* meaning: at the next (previous) time point where `p` holds (if there is such), `q` also holds. These are expanded into the following LTL formulae, where L (R) is the formula that specifies the left (right) endpoint of the scope interval:

- Future for nextOcc
  `($R$ | (X (((!p & !$R$) U p) => ((!p & !$R$) U (p & q)))))`
- Past for prevOcc
  `($L$ | (Y (((!$L$ & !p) S p) => ((!$L$ & !p) S (p & q)))))`

Based on our experience, many PLC program requirements are checked at the end of PLC cycles. This is the most critical moment since the calculated values are sent to the actuators of the system. In the FRET PLC requirements, a variable called `PLC_END` can be used to express this exact moment.

In addition, FRET allows the creation of templates to help users write common requirements. For this work, we have created new FRET templates for PLCverif (e.g., *`In PLC_END`* *`[COMPONENT]`* `shall` *`always`* *`satisfy [RESPONSE]`*). For further details on the workflow, we refer readers to our technical report [14].

3 Case studies

The integration has been used in two CERN critical systems. The PLC programs and the properties are real cases at CERN. The first case study utilizes the validation capabilities, while the second employs the improved expressiveness. Only two properties per case study are shown due to a lack of space.

We give a brief description of the FRETISH syntax used in the case studies. For a complete description please refer to [11]. A FRETISH requirement is composed of six sequential fields: *scope*, *condition*, *component*, shall, *timing* and *response*. The optional *condition* field is a Boolean expression preceded by the word *when* that triggers the *response* Boolean expression to be satisfied when the condition expression becomes true from false. FRETISH provides a variety of timings. In this case study, we use *always* and *eventually*.

3.1 Safety PLC program

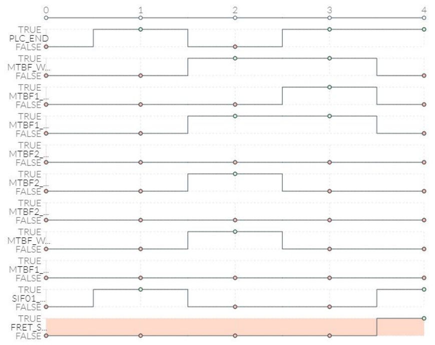
The safety PLC program of the *SM18 Cluster F*, a superconducting magnet test facility at CERN, is meant to protect the personnel and the equipment of this installation. The *SM18 cluster F* is dedicated to test the new superconducting magnet technology for the High Luminosity Large Hadron Collider (HL-LHC) [2], an upgrade of the existing LHC particle accelerator. Its main risks are related to the cryogenic system and the powerful power converters up to 20.000 Amps.

```
F_MAIN shall always satisfy if (PLC_END & (!MTBF_WCC_PCLSW20_FSL | !MTBF_WCC_PCLSW20_TSH |
     (MTBF1_LSW20_POS & (!MTBF1_WCC_LSW20_FSL | !MTBF1_WCC_LSW20_TSH)) | (MTBF2_LSW20_POS &
     (!MTBF2_WCC_LSW20_FSL | !MTBF2_WCC_LSW20_TSH)))) then SIF01_DB.SIF01_PC20K
```

(a) Property of SIF01_PC20K in FRETish.

```
G ((PLC_END and ((((!
  MTBF_WCC_PCLSW20_FSL) or
  (! MTBF_WCC_PCLSW20_TSH))
  or (MTBF1_LSW20_POS and
  ((! MTBF1_WCC_LSW20_FSL)
  or (! MTBF1_WCC_LSW20_TSH)
  ))) or (MTBF2_LSW20_POS and
  ((!MTBF2_WCC_LSW20_FSL) or (!
  MTBF2_WCC_LSW20_TSH)))))
  -> SIF01_DB.SIF01_PC20K)
```

(b) Generated LTL property by FRET.



(c) Future LTL simulation

Fig. 2: Property of the SIF01_PC20K safety function.

Error Property. The property of this case study corresponds to the expected logic of one of the safety functions (*SIF01_PC20K*):

if at the end of the PLC cycle (PLC_END) the flow (*_FSL) or thermo (*_TSH) switches monitoring the cooling system detect a low flow or a high temperature, and the power converter (PC20k) is connected to the magnet (*_LSW20_POS), then the safety function should shut down the power converter (SIF01_PC20K).

Figure 2 shows how the property is expressed in FRETish and in LTL.

Validation. Before verifying this property with PLCverif, the user should make sure that the formalized property behaves as expected. The main challenge is the number of operators and parentheses, making manual validation difficult. FRET's simulator aids by allowing the user to check any temporal valuation and whether it satisfies the property or not as shown in Figure 2c.

3.2 Standard PLC program

This case study is concerned with UNICOS [4], a CERN framework for the development of hundreds of industrial control systems. The selected program library is called the *OnOff* object. Its purpose is to control physical equipment driven by digital signals, which can be composed of different types of devices. This makes its PLC program highly configurable and its associated logic complex.

Error Property. The property presented here is related to the transitions between two operation modes shown in Figure 3: (1) *Auto mode*, where the *OnOff* object is driven by the control logic of a higher object of the hierarchy of the program, and (2) *Manual mode*, where the operator drives the object.

The property extracted from the specification is: When the OnOff object is in Manual mode (MMoSt) and the control logic requests to move to Auto mode (AuAuMoR) at any point in the PLC cycle, the OnOff object should move to the Auto mode (AuMoSt). Figure 4 shows the property in FRETISH and the LTL formula.

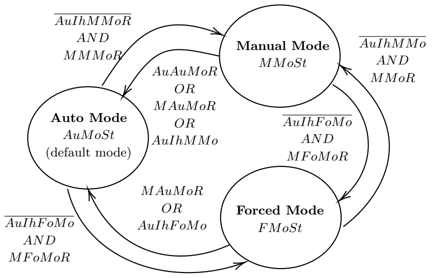

$\overline{AuIhMMoR}$ AND MMMoR
Manual Mode
MMoSt
$\overline{AuIhMMo}$ AND MMoR
AuAuMoR OR MAuMoR OR AuIhMMo
Auto Mode
AuMoSt
(default mode)
$\overline{AuIhFoMo}$ AND MFoMoR
MAuMoR OR AuIhFoMo
Forced Mode
FMoSt
$\overline{AuIhFoMo}$ AND MFoMoR

Fig. 3: OnOff operation modes specification.

```
When (MMoSt & AuAuMoR) the
    CPC_FB_OnOff shall eventually
    satisfy AuMoSt & PLC_END
```

(a) Property expressed in FRETISH.

```
(G (((!(MMoSt and AuAuMoR)) and (X (MMoSt
  and AuAuMoR))) -> (X(F(AuMoSt and
  PLC_END))))) and ((MMoSt and
  AuAuMoR) -> (F(AuMoSt and PLC_END)))
```

(b) Generated LTL property by FRET

Fig. 4: PLCverif property for the *OnOff* object.

```
the CPC_FB_OnOff shall always satisfy
    if (MMoSt & AuAuMoR & !PLC_END)
    then at the next occurrence of
    PLC_END, AuMoSt
```

(a) Property expressed in FRETISH.

```
G (((MMoSt and AuAuMoR) and (! PLC_END)) ->
  (X ((((! PLC_END) ) U PLC_END) ->
  (((! PLC_END)) U (PLC_END and AuMoSt)))))
```

(b) Generated LTL property by FRET

Fig. 5: PLCverif property for the *OnOff* object using the `nextOcc` predicate

Thanks to the new *"at the next occurrence of"* phrase, we can verify that this property is satisfied at the end of the current cycle (which is more precise and strict), as shown in Figure 5.

These properties can not be expressed with the current PLCverif patterns, but they are expressible with the restricted natural language of FRET. Now PLCverif users can express a large variety of requirements in a natural language and validate these in the simulator.

4 Conclusion

The creation of requirements in FRET fits well into the verification workflow and improves both usability and expressiveness, as shown by the case studies. PLCverif [3] and FRET [1] are both open source.

Plans for improvements include the support of verification of time properties in PLCverif and the support for different type widths in the FRET simulator (*e.g. 16/32 bit integers*).

To the best of our knowledge, this is the first attempt to specify formal requirements using a structured natural language for PLC program verification.

References

1. FRET: Formal requirements elicitation tool, `https://github.com/NASA-SW-VnV/fret`
2. High luminosity large hadron collider webpage., `https://home.cern/science/accelerators/high-luminosity-lhc`
3. PLCverif: A tool to verify PLC programs based on model checking techniques, `https://gitlab.com/plcverif-oss`
4. UNICOS webpage., `https://unicos.web.cern.ch/`
5. Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software model checker Blast. International Journal on Software Tools for Technology Transfer **9**(5-6), 505–525 (Sep 2007). `https://doi.org/10.1007/s10009-007-0044-z`, `https://doi.org/10.1007/s10009-007-0044-z`
6. Darvas, D., Fernández Adiego, B., Blanco Viñuela, E.: PLCverif: A tool to verify PLC programs based on model checking techniques. In: Proc. ICALEPCS'15 (10 2015). `https://doi.org/10.18429/JACoW-ICALEPCS2015-WEPGF092`
7. Fernández Adiego, B., Blanco Viñuela, E.: Applying model checking to critical PLC applications : An ITER case study. In: Proc. ICALEPCS'17 (10 2017). `https://doi.org/10.18429/JACoW-ICALEPCS2017-THPHA161`
8. Fernández Adiego, B., Darvas, D., Blanco Viñuela, E., Tournier, J.C., Bliudze, S., Blech, J., González, V.: Applying model checking to industrial-sized PLC programs. IEEE Transactions on Industrial Informatics **11**, 1400–1410 (12 2015). `https://doi.org/10.1109/TII.2015.2489184`
9. Fernández Adiego, B., Lopez-Miguel, I.D., Tournier, J.C., Blanco Viñuela, E., Ladzinski, T., Havart, F.: Applying model checking to highly-configurable safety critical software: The SPS-PPS PLC program. In: Proc. ICALEPCS'21 (03 2022). `https://doi.org/10.18429/JACoW-ICALEPCS2021-WEPV042`
10. Giannakopoulou, D., Pressburger, T., Mavridou, A., Rhein, J., Schumann, J., Shi, N.: Formal requirements elicitation with FRET. In: REFSQ Tools (2020)
11. Giannakopoulou, D., Pressburger, T., Mavridou, A., Schumann, J.: Automated formalization of structured natural language requirements. Information and Software Technology **137**, 106590 (2021)
12. Lopez-Miguel, I.D., Tournier, J.C., Fernández Adiego, B.: PLCverif: Status of a formal verification tool for programmable logic controller. In: Proc. ICALEPCS'21 (03 2022). `https://doi.org/10.18429/JACoW-ICALEPCS2021-MOPV042`
13. Viñuela, E.B., Darvas, D., Molnár, V.: PLCverif Re-engineered: An Open Platform for the Formal Analysis of PLC Programs. In: Proc. ICALEPCS'19. pp. 21–27. No. 17 in International Conference on Accelerator and Large Experimental Physics Control Systems, JACoW Publishing, Geneva, Switzerland (08 2020). `https://doi.org/10.18429/JACoW-ICALEPCS2019-MOBPP01`, `https://jacow.org/icalepcs2019/papers/mobpp01.pdf`, https://doi.org/10.18429/JACoW-ICALEPCS2019-MOBPP01
14. Ádám, Z., Lopez-Miguel, I.D., Mavridou, A., Pressburger, T., Bęś, M., nuela, E.B.V., Katis, A., Tournier, J.C., Trinh, K.V., Adiego, B.F.: Automated verification of programmable logic controller programs against structured natural language requirements. Tech. Rep. NASA/TM 0230003752 (2023)