Formal verification of UNICOS-CPC PLC Baseline Objects

L. Dao

University of Gothenburg, Gothenburg, Sweden

Supervisor: *B. Fernández Adiego*. CERN, Geneva, Switzerland

Abstract

During my time at CERN, I worked on the application of formal verification to UNICOS-CPC baseline objects using PLCverif to enhance framework reliability, improve object understanding and minimize extensive testing efforts. The project demonstrated that formal verification can uncover code bugs, difficult to detect through conventional testing. Through analysis of the OnOff object, specific instances of safety requirement violations were identified. Furthermore, clearer specifications were created, including for example the OnOff mode manager and an automated verification pipeline built to streamline testing for new object versions. This project showcases that formal verification not only improves the quality of UNICOS-CPC but also provides a deeper understanding of their behavior, potentially enabling a future model-based development process to build more robust systems and reduce testing workloads.

Keywords

CERN, summer student report, formal verification, UNICOS-CPC, PLC, baseline objects, safety requirements, PLCverif

1 Introduction

Developing reliable control systems is crucial for the operation of CERN's industrial facilities. A key part of this process is verification, which involves checking whether a system meets its specified requirements. Verification is traditionally achieved through testing, but this approach has inherent limitations. Unlike testing, formal verification relies on building a mathematical model of the program and using mathematical methods and logic to prove that the program satisfies the specified requirements [1].

2 Background

Testing has been the most common method for ensuring a program behaves as intended. However, even automated tests cannot cover all possible combinations of inputs and execution paths within a program. While testing can detect the presence of errors, it cannot prove their absence. This approach is particularly challenging when verifying security requirements (such as preventing dangerous states) or maintaining invariants (conditions that must always hold true) [2].

Formal verification offers a more powerful complement. The most popular formal verification technique is model checking, where the code is translated into a formal model and requirements are expressed as logical formulas. The model checker then systematically examines whether the requirements hold in all possible situations. If a violation is found, the model checker provides a counterexample, illustrating the specific conditions that trigger the error. This allows for the discovery of corner cases and subtle errors that would be difficult to identify through testing alone, leading to a better understanding of the system's behavior and potentially prompting adjustments to the specification [3].

PLC programs are widely used in industrial facilities, from process industries to scientific installations like CERN, where they control essential systems such as cooling, cryogenics, ventilation and gas management [4]. To standardize the development of PLC programs in these control systems, CERN has

developed the UNICOS-CPC (Unified Industrial Control System - Common PLC Core) framework. A key element of this framework is the baseline objects, which are reusable program libraries for typical automation components. These objects model real equipment, manage output signals and process feedback from sensors [5]. Errors within these programs can propagate throughout hundreds of installations, potentially leading to operational disruptions, material damage, or, critically, risks to personnel safety [2]. Therefore, formal verification of the CPC baseline objects is an important step in increasing the reliability and robustness of their complex behavior.

In this project, PLCverif is used to enable formal verification of PLC programs. PLCverif, a tool developed here at CERN, automates the translation of PLC code into a formal model, formalizes the property specifications, reduces the formal model and calls state of the art model checkers to verify the PLC program.

3 The project

Each year, a new version of UNICOS-CPC is released, requiring extensive testing to ensure the functionality of all components including the baseline objects. This requires a significant time and resources.

This project aimed to investigate how formal verification could reduce the testing costs. By integrating continuous verification pipelines, it is possible to automatically check that requirements remain valid. This approach offers the potential to minimize testing resources while simultaneously enhancing reliability and ensuring that new changes do not compromise existing functionality, as verification systematically explores all possible combinations of input data.

3.1 Goals

- Verify the behavior of selected parts of the baseline objects and ensure that key requirements are met.
- Integrate verification cases into continuous integration/continuous delivery (CI/CD) pipelines to enhance reliability and guarantee bug free code for new releases.
- Gain a deeper understanding of the baseline object's logic and develop clear, unambiguous specifications that experts can review and developers can use as reference material.

3.2 Methods

To achieve these goals, the tool PLCverif was utilized. The working method can be divided into the following steps:

1. Import the PLC code from a baseline object.
2. Define requirements that the object should satisfy, using one of the specification methods provided by PLCverif (referred to as assertions). These assertions can address safety concerns (e.g. preventing simultaneous activation of conflicting positions) or logical conditions (e.g. ensuring that a full stop sequence always triggers an interlock).
3. Check these requirements using a supported model checker.
4. Reverse engineer the code and analyze counterexamples when requirements are not met to understand the underlying cause.
5. Integrate the verification requirements from PLCverif into CI/CD pipelines, enabling automated execution of the PLCverif command-line interface (CLI) and verification of a defined set of assertions. This allows verification to occur automatically whenever new code is committed to the version control system.
6. Formulate new formal specifications for the baseline objects based on previous work at CERN [6]. The core logic within these specifications was structured using state machines and logic

diagrams, which not only improves visual clarity and reduces ambiguity but also better positions the specifications for future verification efforts.

4 Case study

A central case study in this project was the OnOff object, a PLC library component used to control the digital actuators, like an OnOff valve or a digital pump. A simplified diagram of the OnOff program interface and a property to be verified is shown in Figure 1.

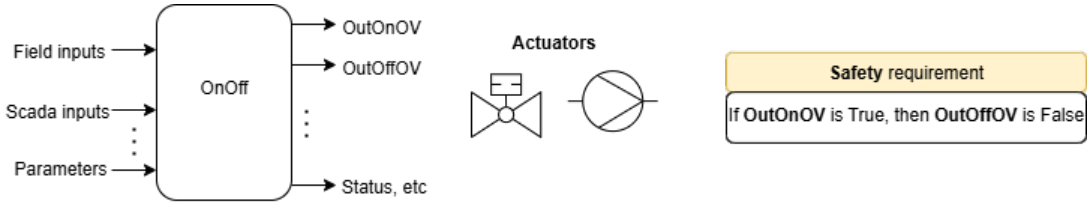


Figure 1: OnOff Object

The OnOff logic is considerable complex: it comprises over 800 lines of code, a large number of inputs and outputs and numerous internal variables and functions. It incorporates logic for operating modes, timers, interlocks and warnings. This complexity renders exhaustive testing impractical as the sheer number of possible states grows to astronomical figures, as presented in Table 1.

Metric	**OnOff PLC program**
PLC language	SCL (ST IEC 61131-3)
Lines of code	≈ 820
Program blocks	1 main FB, 2 timers and 3 FCs
Statements	418
Function calls	21
Input variables	29 (20 BOOL, 3 WORD, 2 ARRAY of 16 BOOL and 4 TIME), i.e. $2^{228} \approx 4.3 \times 10^{68}$ combinations
Output variables	31 (27 BOOL, 2 WORD and 2 ARRAY of 16 BOOL)
Internal variables	82 (73 BOOL, 1 TIME, 2 TP timer instances, 1 TON timer instance, 1 REAL and 4 INT)
PLC data types	BOOL, INT, REAL, WORD, TIME, STRUCT and ARRAY
Timers	3 instances

Table 1: Metrics of the OnOff PLC program

4.1 Verification of a safety requirement

The verification of the OnOff object revealed that a critical safety requirements (shown in Figure 1) that were not consistently met. Intuitively, the requirements stems from the fact that if the OnOff object sends the order to open the valve (*OutOnOv* equals to *TRUE*), the OnOff object should not send an order to close the valve (*OutOffOv* equals to *TRUE*) at the same time. However, this property was not guaranteed in the original code. Through analysis of the generated counterexamples and subsequent reverse engineering of the code, the variables responsible for the issue were precisely localized.

The first instance happened in Local Drive Mode, under a specific combination of inputs HFON and HFOFF, together with parameters such as POutOff and PHLDCmd. Both outputs could become active (or inactive) simultaneously. These findings were subsequently validated using a testing tool developed by a colleague at CERN, as shown in Figure 2. This highlights a clear example of how verification can uncover rare cases that are frequently missed by conventional testing methodologies.

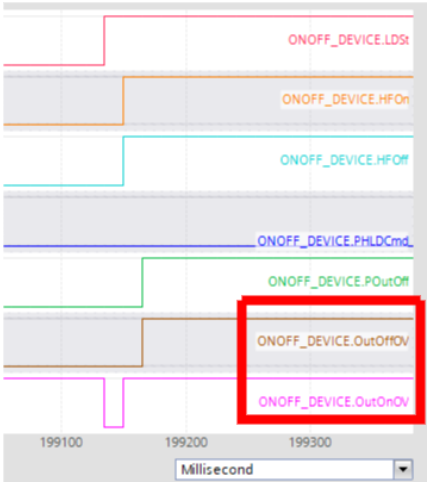


Figure 2: Trace in Siemens Simulation

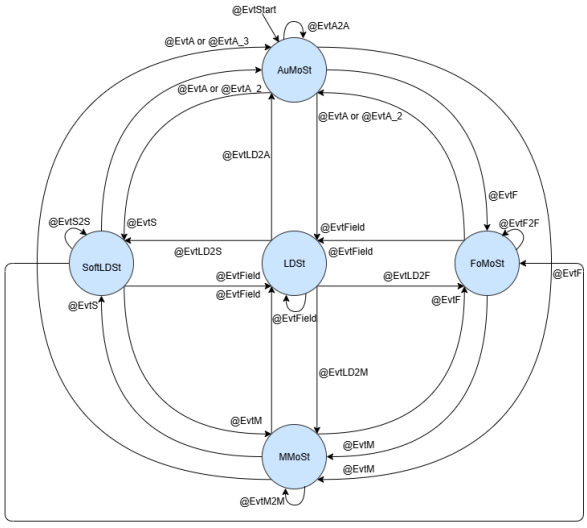
The second instance involved Interlock logic of the OnOff object. This logic is designed to enforce a safe state by transmitting the correct output signal when the system encounters abnormal or potentially dangerous inputs (e.g pressure limits, wrong system states, or bad sensor reading). However, with a particular configuration of parameters, both outputs could become active (or inactive) at the same time, when an interlock is engaged. Demonstrating that even seemingly straightforward logic can contain unforeseen exceptions depending on the implementation.

The third instance concerned the Position Manager of the outputs OutOnOV and OutOffOV. A specific parameter configuration resulted in the logic resetting only one of the signals, leaving the other in a previous state. This resulted in a situation where both outputs are active simultaneously. This is exactly the kind of subtle behavior that can cause problems in practice, but is very difficult to detect with testing.

4.2 Mode manager specification

Formal verification extends beyond individual safety checks, it can aid the creation of a comprehensive formal specification. Take for example the Mode Manager, a component responsible for managing operating modes such as Auto, Local Drive or Manual.

Here the new specification explicitly states that only one mode can be active at any given time. It also incorporates modes that exist within the code but were absent from previous documentation, thereby creating a more complete description of the object's functionality. Critically, the new specification clarifies mode priority, e.g when multiple mode requests are active, it explains which mode will be selected in the so called event definition. A detail previously unaddressed in the original documentation yet already implemented in the code. The new specification explicitly details this hierarchy, enhancing understanding and facilitating validation, as illustrated in the Figures 3a and 3b.



(a) Core Logic - state machine

Event definitions:

- @EvtField ⇐ HLD and PHLD
  Highest priority
- @EvtStart ⇐ $\overline{\text{@EvtField}}$ and $\overline{\text{AuMoSt\_aux}}$ and $\overline{\text{SoftLDSt\_aux}}$ and $\overline{\text{MMoSt\_aux}}$ and $\overline{\text{FoMoSt\_aux}}$
  At the begining, when everything is false
- @EvtLD2S ⇐ $\overline{\text{@EvtField}}$ and SoftLDSt_aux and $\overline{\text{AuIhFoMo}}$
  Previous mode and not inhibited
- @EvtLD2M ⇐ $\overline{\text{@EvtField}}$ and MMoSt_aux and $\overline{\text{AuIhMMo}}$
  Previous mode and not inhibited
- @EvtLD2F ⇐ $\overline{\text{@EvtField}}$ and FoMoSt_aux and $\overline{\text{AuIhFoMo}}$
  Previous mode and not inhibited
- @EvtLD2A ⇐ $\overline{\text{@EvtField}}$ and {AuMoSt_aux or ((FoMoSt_aux or SoftLDSt_aux) and AuIhFoMo) or (MMoSt_aux and AuIhMMo)}
  Previous mode or inhibited
- @EvtF ⇐ $\overline{\text{@EvtField}}$
  and E_MFoMoR and $\overline{\text{AuIhFoMo}}$
- @EvtS ⇐ $\overline{\text{@EvtField}}$
  and E_MSoftLDR and $\overline{\text{AuIhFoMo}}$
  and E_MFoMoR
  input request and condition for not @EvtF
- @EvtM ⇐ $\overline{\text{@EvtField}}$
  and E_MMMoR and $\overline{\text{AuIhMMo}}$
  and {E_MFoMoR or E_MSoftLDR or AuIhFoMo}
  input request and condition for not {@EvtF or @EvtS}
- @EvtA ⇐ $\overline{\text{@EvtField}}$
  and E_MAuMoR
  and {E_MFoMoR or E_MSoftLDR or AuIhFoMo}
  and {E_MMoR or AuIhMMo}
  input request and condition for not {@EvtF or @EvtS or @EvtM}
- @EvtA_2 ⇐ $\overline{\text{@EvtField}}$
  and AuIhFoMo and {E_MMoR or AuIhMMo}
  input request and condition for not {@EvtM}
- @EvtA_3 ⇐ $\overline{\text{@EvtField}}$
  and {AuIhMMo or E_AuAuMoR)
  and {E_MFoMoR or E_MSoftLDR or AuIhFoMo}
  input request and condition for not {@EvtF or @EvtS}

(b) Event Definition

Figure 3: Formal specification of the OnOff model manager

5 Results

Throughout the project, I achieved significant technical and methodological results. Beyond the detailed case study of the OnOff object, the formal verification approach was extended to other baseline components, including the PID, ANALOG and ANADIG UNICOS-CPC objects. This work finished with a fully automated verification pipeline designed for seamless integration with new versions of these baseline objects. Upon code modification by a developer, this pipeline executes verification cases and promptly reports any deviations from established requirements, enabling the integration of formal verification into a CI/CD process.

Furthermore, I contributed to the formalization of multiple parts of these objects. Prior documentation lacked clarity or contained ambiguities, but through rigorous analysis and verification efforts, new specifications that accurately describe the object's behavior were produced. All relevant pipelines and specifications are available at the following link [7].

Finally, I developed a Python script that translates simple PLC Boolean code into Python syntax, which is then converted into Boolean expressions. These Boolean expressions can be imported into a drawing and simulation tool for automated logic circuit generation called DIGITAL [8]. This minimizes the time to reverse engineer the PLC programs and produce the formal specifications using Logic Diagrams. This process is visualized in Figure 4.

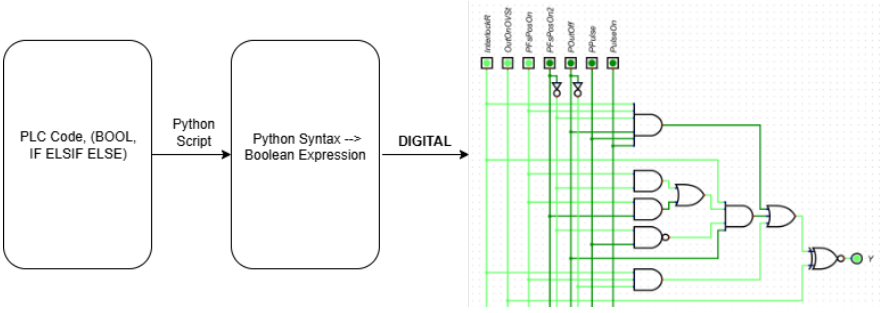


Figure 4: PLC Boolean code to logic diagram

6 Discussion

This project has demonstrated the potential of formal verification to enhance the development life cycle of UNICOS-CPC baseline objects. Key benefits include the discovery of subtle bugs that would otherwise be difficult to detect. Additionally, an automated verification within CI/CD pipelines which can be expected to yield substantial time savings compared to current testing activities. The creation of formal specifications, will allow the UNICOS experts to review systematically the objects behavior and therefore, any unusual or inconsistent cases outside the desired specification can be detected more easily.

However, challenges remain. Despite the benefits, formal requirement specification requires a significant investment of time and demands a deep understanding of both the object's logic and the basic principles of formal methods. This may present a barrier for those unfamiliar with these areas.

6.1 Future directions

In the short term, UNICOS-CPC group experts could review the specifications and verification cases and expand them to improve the reliability of upcoming releases.

In the long run, a model-based software development approach can be adopted. A complete formal model (specification) could be used not only for verification but also for automated code generation, test case creation, documentation refinement and more. This would effectively promote a "correct by design" development process.

Another promising direction involves the automated generation of user documentation directly from the formal specifications. By translating these specifications into natural language or simple state machines, documentation that is both easily understandable and guaranteed to be accurate could be automatically generated. This would improve the current documentation for the UNICOS-CPC users.

Bibliography

[1] L. Dao, A. Davoodi and E. Samuelsson, "How do we know that a computer program does what it says it does? Formal verification of hypergeometric recurrence relations with polynomial coefficients," Bachelor's thesis in Mathematics, Chalmers University of Technology & University of Gothenburg, 2024. Available: gupea.ub.gu.se/handle/2077/85570

[2] I. D. Lopez-Miguel, C. Betz, E. Blanco Viñuela, B. Fernández Adiego and M. Salinas, "Working Together for Safer Systems: A Collaboration Model for Verification of PLC Code," in *Proceedings of the 19th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2023)*, Cape Town, South Africa, 7–13 Oct. 2023, paper TUPDP001, JACoW ICALEPCS 2023, doi:10.18429/JACoW-ICALEPCS2023-TUPDP001, 2023. Available: cds.cern.ch/record/2895134

[3] B. Fernández Adiego, E. Blanco Viñuela, F. Havart, T. Ladzinski, I. D. Lopez-Miguel and J.-C. Tournier, "Applying Model Checking to Highly-Configurable Safety Critical Software: The SPS-PPS PLC Program," in *Proceedings of the 18th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2021)*, Online, 18–22 Oct. 2021, pp. 759–763, JACoW ICALEPCS2021, 2022. doi:10.18429/JACoW-ICALEPCS2021-WEPV042. Available: cds.cern.ch/record/2809709

[4] D. Darvas, B. Fernández Adiego and E. Blanco Viñuela, "PLCverif: A Tool to Verify PLC Programs Based on Model Checking Techniques," in *Proceedings of the 15th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2015)*, Melbourne, Australia, paper WEPGF092, 4 pp., ISBN 978-3-95450-148-9, 2015. Available: accelconf.web.cern.ch/ICALEPCS2015/papers/wepgf092.pdf

[5] CERN BE-ICS, "UNICOS-CPC — UNICOS Continuous Process Control: Resources Package 1.18.1 Documentation," documentation, CERN, 2016. Available: ucpc-resources.web.cern.ch/latest/

[6] D. Darvas, E. Blanco Viñuela and I. Majzik, "A Formal Specification Method for PLC-Based Applications," in *Proceedings of the 15th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2015)*, Melbourne, Australia, paper WEPGF091, 4 pp., ISBN 978-3-95450-148-9, 2015. Available: cds.cern.ch/record/2213506/files/wepgf091.pdf

[7] Pipelines, gitlab.cern.ch/plcverif-internal/baseline_1500/baseline_1500_onoff/-/pipelines. Specifications, indico.cern.ch/event/1579460/

[8] H. Neemann, *Digital: A digital logic designer and circuit simulator* [Computer software]. GitHub, 2024. [Online]. Available: github.com/hneemann/Digital