# PLCverif: A Tool to Verify PLC Programs Based on Model Checking Techniques

2015 ICALEPCS melbourne • australia

WEPGF092

**D. Darvas, B. Fernández Adiego, E. Blanco Viñuela**, CERN, Geneva, Switzerland
daniel.darvas@cern.ch | borja.fernandez.adiego@cern.ch | enrique.blanco@cern.ch

## Motivation

### Testing is not good for everything

- **Not feasible to test all combinations**, only some selected input sequences are checked
- Testing **cannot show the absence of bugs**, it can only show their **presence**

### Model checking can complement

**Model checking**: analysing whether a *formal* system model satisfies the given formal requirements. It may…

- … check **all possible combinations**
- … prove the **absence of bugs**
- … give a **counterexample** if the requirement is violated

### But…

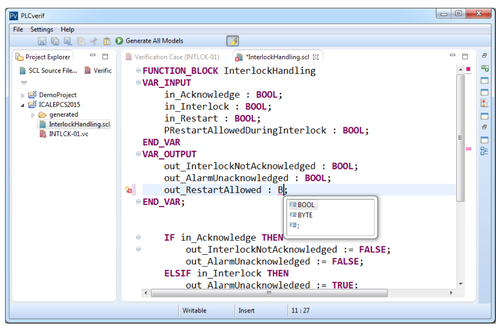
- Model checking typically needs **special expertise to** produce the formal models and requirements
- It has a **high computational complexity**

**Our goal is to overcome these issues and to *make model checking accessible* to the PLC developers.**

## Workflow

### 1) Introducing the code

- Source code can be **imported** or **locally edited**
- Supported **languages**: ST/SCL, IL/STL (partially), SFC (partially)
- Included **ST/SCL editor** with syntax highlighting, content assist, refactoring support, etc.

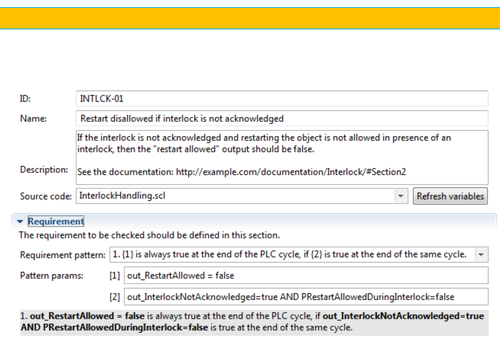


*Example*

```
IF in_Acknowledge THEN
    out_InterlockNotAcknowledged := FALSE;
    out_AlarmUnacknowledged := FALSE;
ELSIF in_Interlock THEN
    out_AlarmUnacknowledged := TRUE;
END_IF;
IF  (in_Restart AND NOT in_Interlock) OR
   (PRestartAllowedDuringInterlock AND in_Restart AND
        in_Interlock)
   AND NOT out_InterlockNotAcknowledged THEN
    out_RestartAllowed := TRUE;
END_IF;
IF in_Interlock THEN
    out_InterlockNotAcknowledged := TRUE;
    out_RestartAllowed := FALSE;
END_IF;
```

### 2) Defining the requirement

- A **verification case** contains all necessary information: metadata and the requirement
- It is difficult to use temporal logics for the non-expert users. Instead, a **verification pattern** has to be **chosen** from a pre-defined list and **filled with simple expressions**



*Example*

**Pattern**: *If* [?] *(at the end of a PLC cycle), then* [?] *is always true (at the end of the same cycle).*

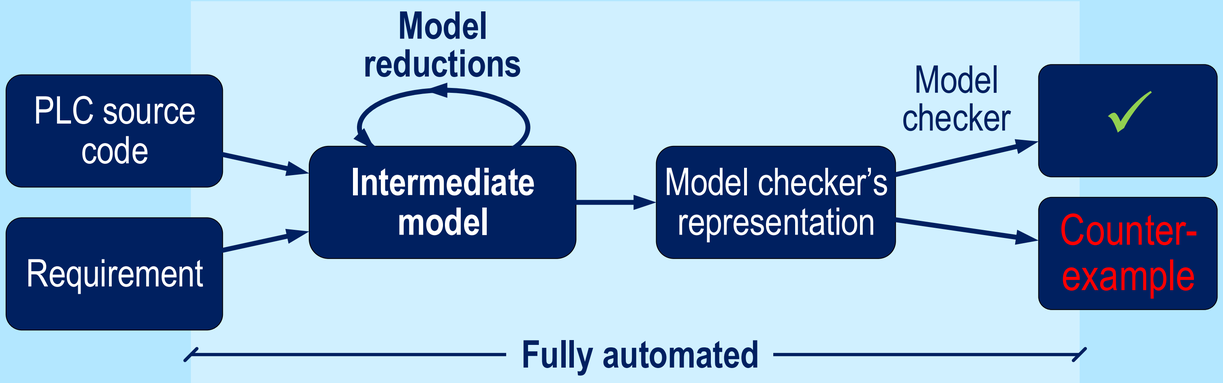
**Expressions**: out_InterlockNotAcknowledged=true AND PRestartAllowedDuringInterlock=false; out_restartAllowed = false

**Formal requirement** (used internally): *CTLSPEC AG((PLC_END & ($instance/out_interlocknotacknowledged$ = TRUE & $instance/prestartallowedduringinterlock$ = FALSE )) -> ($instance/out_restartallowed$ = FALSE ))*

### 3) Verification: Model checking

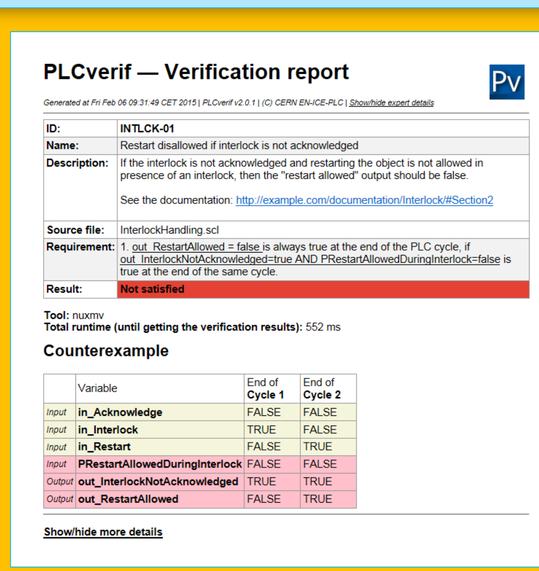
Steps performed:

1. The PLC code is parsed and translated into a **verification model**
2. The pattern-based requirement is translated into a **formal, mathematical requirement** description format (temporal logics: CTL or LTL)
3. The verification model is **reduced**
4. The verification model is **translated to the external model checker's format**
5. The **model checker tool is executed** and its result is parsed (Currently included model checkers: nuXmv, NuSMV, UPPAAL, BIP)




### 4) Report & analysis

- The output of the model checker is not easy to understand
- A **verification report summarizes the outcome** of the verification for the user in an intuitive way
- If the requirement is not satisfied, the **counterexample** shows an **example for the violation**
- Based on the counterexample, the **violation can be reached** in a controlled way or the corresponding part of the implementation can be analysed



**PLCverif — Verification report**

| | |
|---|---|
| ID: | INTLCK-01 |
| Name: | Restart disallowed if interlock is not acknowledged |
| Description: | If the interlock is not acknowledged and restarting the object is not allowed in presence of an interlock, then the "restart allowed" output should be false. See the documentation: http://example.com/documentation/Interlock/#Section2 |
| Source file: | InterlockHandling.scl |
| Requirement: | 1. out_RestartAllowed = false is always true at the end of the PLC cycle, if out_InterlockNotAcknowledged=true AND PRestartAllowedDuringInterlock=false is true at the end of the same cycle. |
| Result: | Not satisfied |

Tool: nuxmv
Total runtime (until getting the verification results): 552 ms

Counterexample

| | Variable | End of Cycle 1 | End of Cycle 2 |
|---|---|---|---|
| Input | in_Acknowledge | FALSE | FALSE |
| Input | in_Interlock | TRUE | FALSE |
| Input | in_Restart | FALSE | TRUE |
| Input | PRestartAllowedDuringInterlock | FALSE | FALSE |
| Output | out_InterlockNotAcknowledged | TRUE | TRUE |
| Output | out_RestartAllowed | FALSE | TRUE |

Show/hide more details

*Example*

The counterexample shows a **violation**:
out_InterlockNotAcknowledged=true and PRestartAllowedDuringInterlock=false, **but out_restartAllowed = true**.

Based on that the **problem can be reproduced.**

The **source** of violation **can be found** in the implementation: *a pair of parentheses is missing from the expression*.

```
IF  (in_Restart AND NOT in_Interlock) OR
   (PRestartAllowedDuringInterlock AND in_Restart AND
        in_Interlock)
   AND NOT out_InterlockNotAcknowledged THEN
    out_RestartAllowed := TRUE;
END_IF;
```

## Experiences

A **tool** was developed to **implement our methodology** and **hide the complexity** from the user: **PLCverif** (http://cern.ch/plcverif/)

- **Problems were found** in **well-tested modules** of the UNICOS framework
- PLCverif was applied in the development of a new **safety-critical control system**, giving continuous feedback and ~15 bug reports to the developers
  - It would have been **practically impossible to find** many of these bugs using testing
  - PLCverif provided **feedback** on code **before deployment** – lower correction cost

## Conclusions

- A tool hiding the complexity can **help to integrate formal verification** to the development process
- **Model reductions** make the model size smaller, thus the verification feasible
- Model checking can **complement testing** of industrial control software
- **Testing is still needed**: model checking is not universally applicable
- More work is needed in the future:
  - **Better algorithms** (to increase the set of verifiable problems)
  - **Better specification methods** (to have unambiguous requirements)
  - **Better tool** (to support more languages)

You can find this poster, the paper and more information at
**http://go.cern.ch/x8vH**
**http://cern.ch/plcverif**

CERN | EN ENGINEERING DEPARTMENT | ICE INDUSTRIAL CONTROL