
CERN

Dániel Darvas (CERN / BME)

PLCverif:
Model checking PLC programs

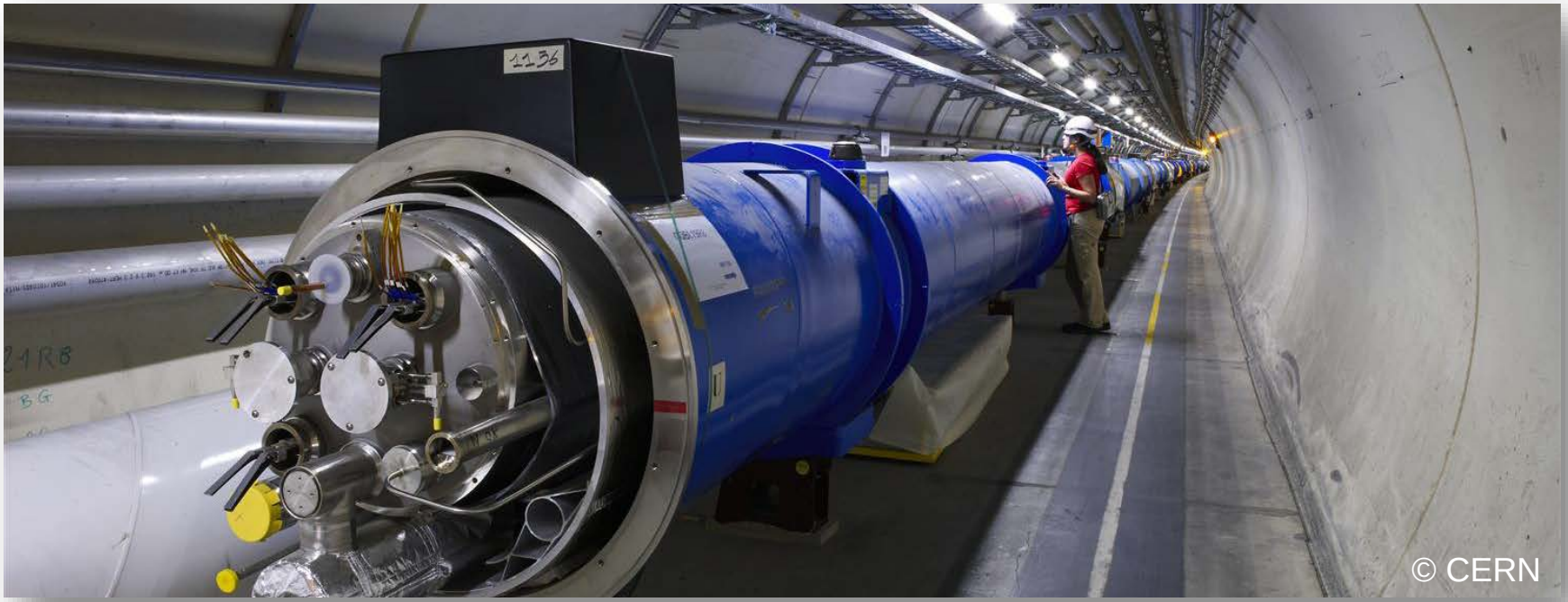
Formal Methods course, BME

22/02/2017

Contains joint work with B. Fernández Adiego, E. Blanco Viñuela, S. Bliudze, J.O. Blech, J-C. Tournier, T. Bartha, A. Vörös, R. Speroni, I. Majzik

CERN *European Org. for Nuclear Research*

- Largest **particle physics laboratory**
- Accelerator complex, incl. Large Hadron Collider (LHC)
  - Proton beams with high energies



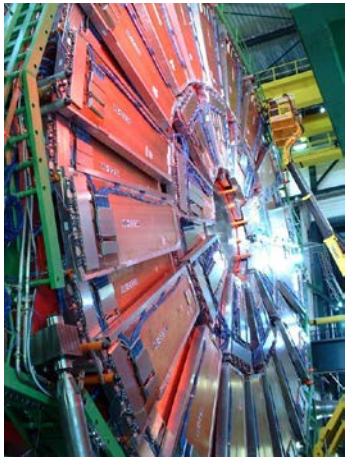
PLCs

- Programmable Logic Controllers:
  robust industrial computers, specially designed for process control tasks

- 1000+ PLCs at CERN
  - Including many **critical systems**

Cryogenics

Vacuum



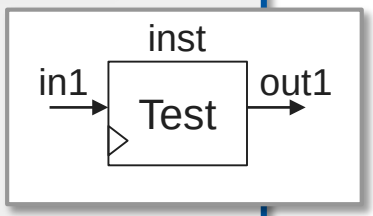
Detector control

PLC programming

- 5 standard PLC programming languages
  - Base building block: *function block*

Siemens SCL language

```
FUNCTION_BLOCK Test
  VAR_INPUT
    in1: Bool;
  END_VAR
  VAR_OUTPUT
    out1: Bool;
  END_VAR
BEGIN
  out1:= NOT in1;
END_FUNCTION_BLOCK

DATA_BLOCK  inst  Test
BEGIN
END_DATA_BLOCK
```



”Equivalent” Java code

```
final class Test {
  public boolean in1 = false;
  public boolean out1 = false;

  public void execute (boolean in1) {
    this.in1 = in1;
    execute();
  }
  public void execute () {
    out1 = !in1;
  }
}

public Test inst = new Test();
```

Motivation for formal verification

- PLCs are often not safety-critical

but

- **Expensive equipment** is operated by PLCs
- **Update** of PLC programs difficult
- The **cost of downtime** is high

Using formal methods

- Formal verification (model checking) may **complement testing** to find **more complex faults**

but

- Model checking has to be **accessible to the PLC developers**

- Required **effort** has to be in balance with the **benefits**
  - The method has to be **adapted to the available knowledge**
  - **Formal details** should be **hidden**
  - **Recurring tasks** should be **automated** or facilitated

CERN

Model checking of PLC programs

Challenges

- **Formal models**
  - Creation of formal models require lots of effort and knowledge
- **Formal requirements**
  - Formalizing requirements in e.g. CTL/LTL is difficult, they are inconvenient and ambiguous without strong knowledge
- **Model size and model checking performance**
  - “Naïve modelling” often leads to complex, large models requiring excessive resources to verify;
  - Optimization of models is difficult and tedious
- **Model checker development**
  - CERN is not a computer science research centre, development of a custom model checker would need to much effort

CERN

Can we use external tools?

- **General-purpose formal modelling and verification tools** (e.g. UPPAAL, NuSMV)
  - Usage is **too difficult**
  - Too much **repetitive tasks** in modelling

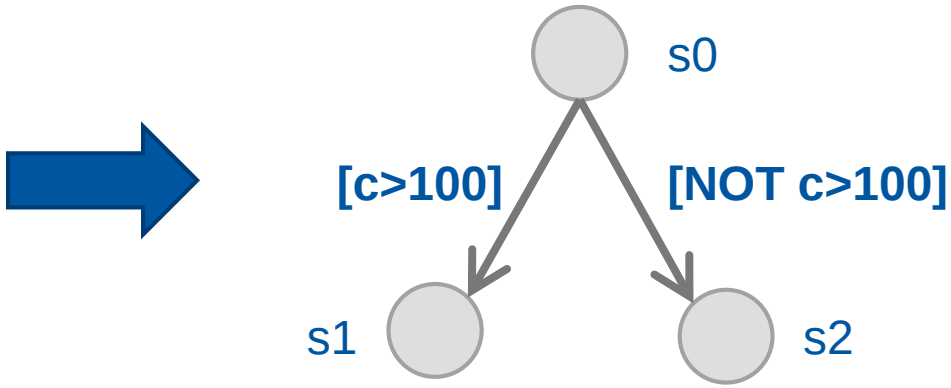
- **Software model checkers** (e.g. CBMC)
  - PLCs use **special programming languages** and execution scheme

- **PLC-specific model checkers**
  - **No industrial solution** yet
  - Some academic tools (e.g. Arcade.PLC)

Formal modelling

- **Formal models** (~automata) automatically generated **from the source code** of the PLC programs *(via AST)*

```
IF c > 100 THEN
      s1;
ELSE
      s2;
END_IF;
```



Formalizing the requirements

- Use of **CTL/LTL** is too difficult for most users

- Typical requirements were captured as **textual requirement patterns**
  - **Placeholders** to be filled by the users (using simple expressions)

If **α** and **β** are true, then **α** shall stay true until **β** becomes true.

$$\boldsymbol{AG}\big((\boldsymbol{\alpha} \wedge \boldsymbol{\beta}) \rightarrow \boldsymbol{A}[\boldsymbol{\alpha}\, \boldsymbol{U}\, \neg\boldsymbol{\beta}]\big)$$

CERN

Model size and performance

- **Size** of the generated formal model is often **huge**, **verification often impossible** (memory bottleneck)

- **Automated reductions** reduce the resource needs
  - **General-purpose, structural reductions**
  - **Domain-specific reductions**
    - Exploit the extra knowledge about the domain, the execution schema, etc.
  - **Requirement-specific reductions**
    - Removes the parts of the model which **do not influence the satisfaction** of the current requirement

External model checkers

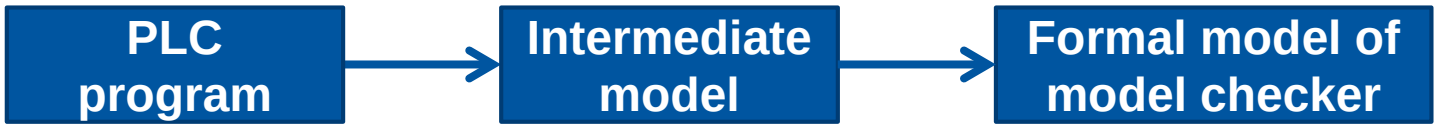
- **Development of a custom model** checker would need **excessive effort**

- Instead, we **reuse (wrap) existing** general-purpose **model checkers** as generic verification engines
  - UPPAAL
  - NuSMV / nuXmv
  - ITS
  - …

- **Input** (model+requirement) **mapping** +
  Output (counterexample) **mapping** needed

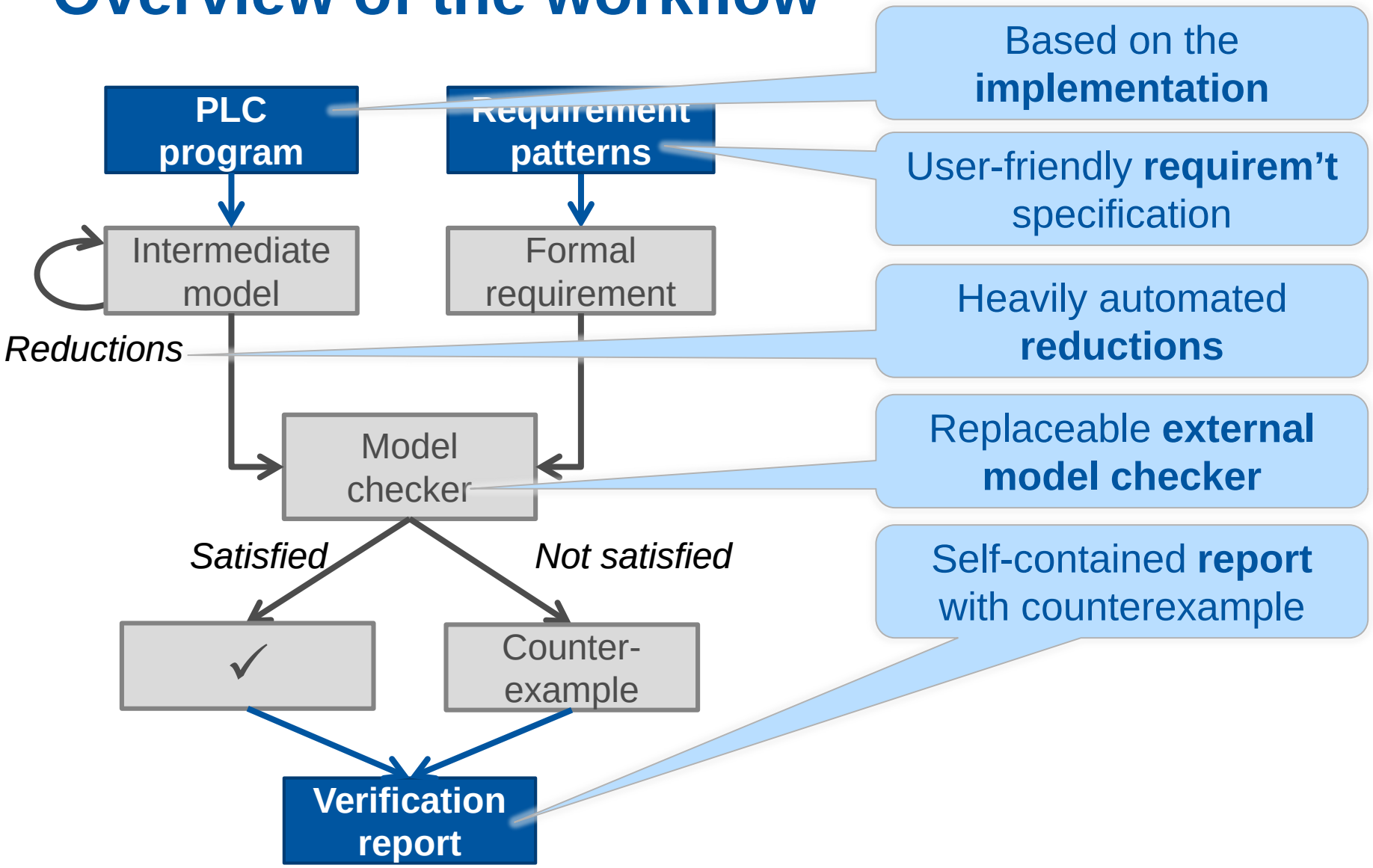
Intermediate model

- Simple, **automata-based** formalism
- Describes the **behaviour** of the PLC program



- Advantages:
  - Helps to use **model reductions** (on the IM)
  - Helps to use **various model checkers** with different syntaxes
  - **Simplifies (decouples)** the PLC program → Model checker model **transformation**, thus reduces the risk of faults

Overview of the workflow

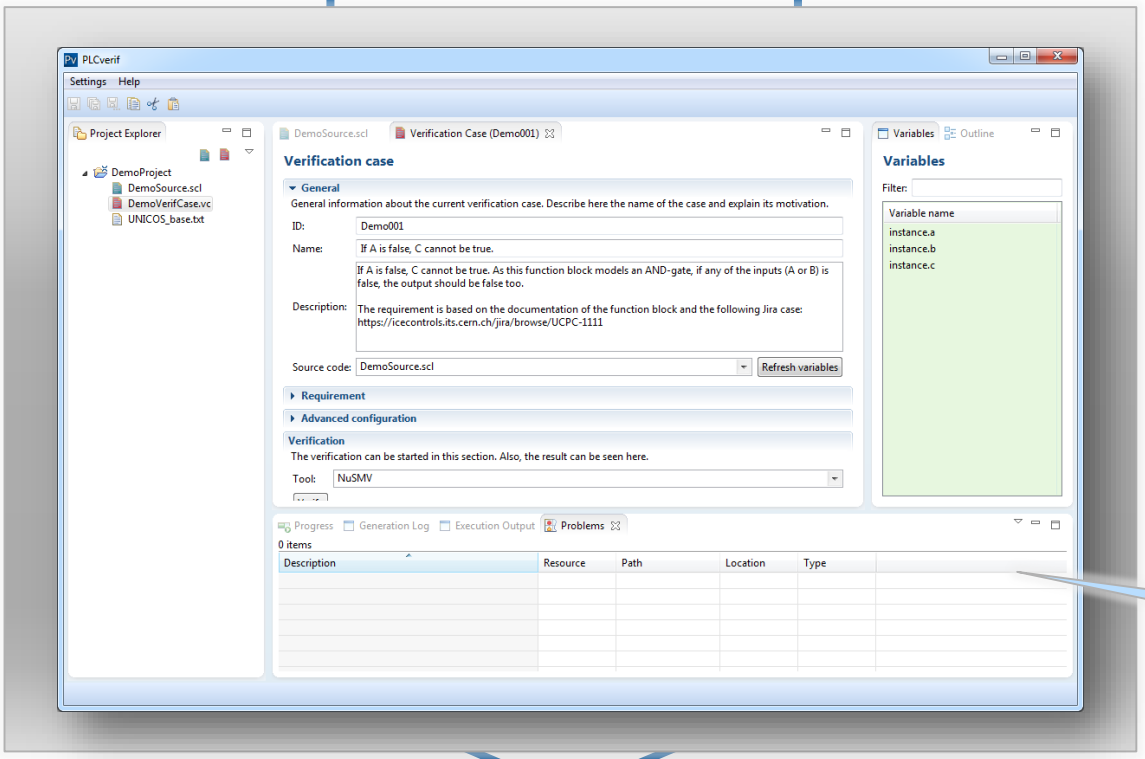


More info: B. Fernández et al. Bringing automated model checking to PLC program development - A CERN case study. In Proc. WODES 2014, pp. 394-399. IFAC, 2014. doi: 10.3182/20140514-3-FR-4046.00051

Overview of the workflow

PLC program

Requirement patterns



Verification report

Based on the **implementation**

User-friendly **requirem't** specification

Heavily automated **reductions**

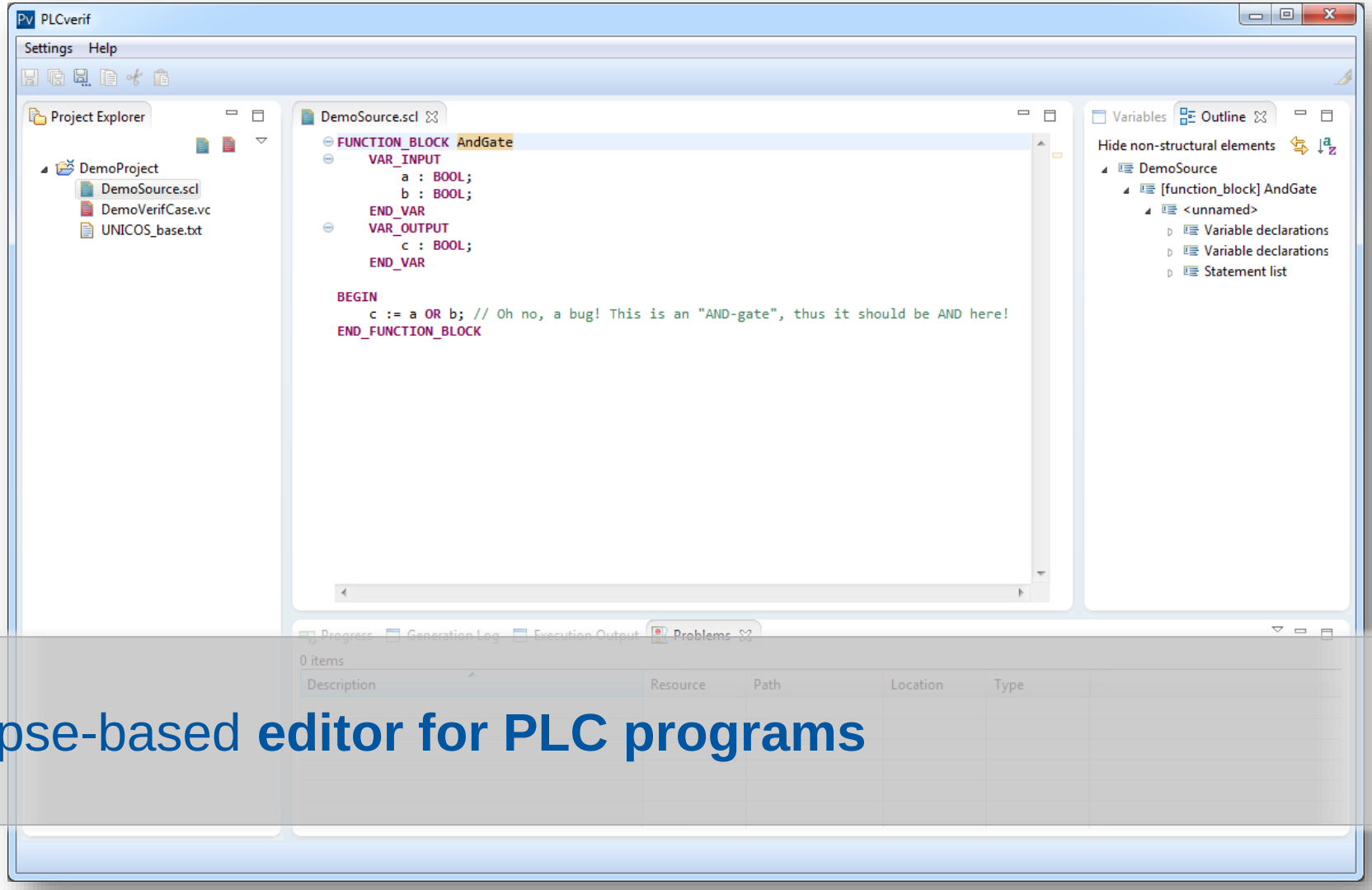
Replaceable **external model checker**

Self-contained **report** with counterexample

Tool hiding the formal details

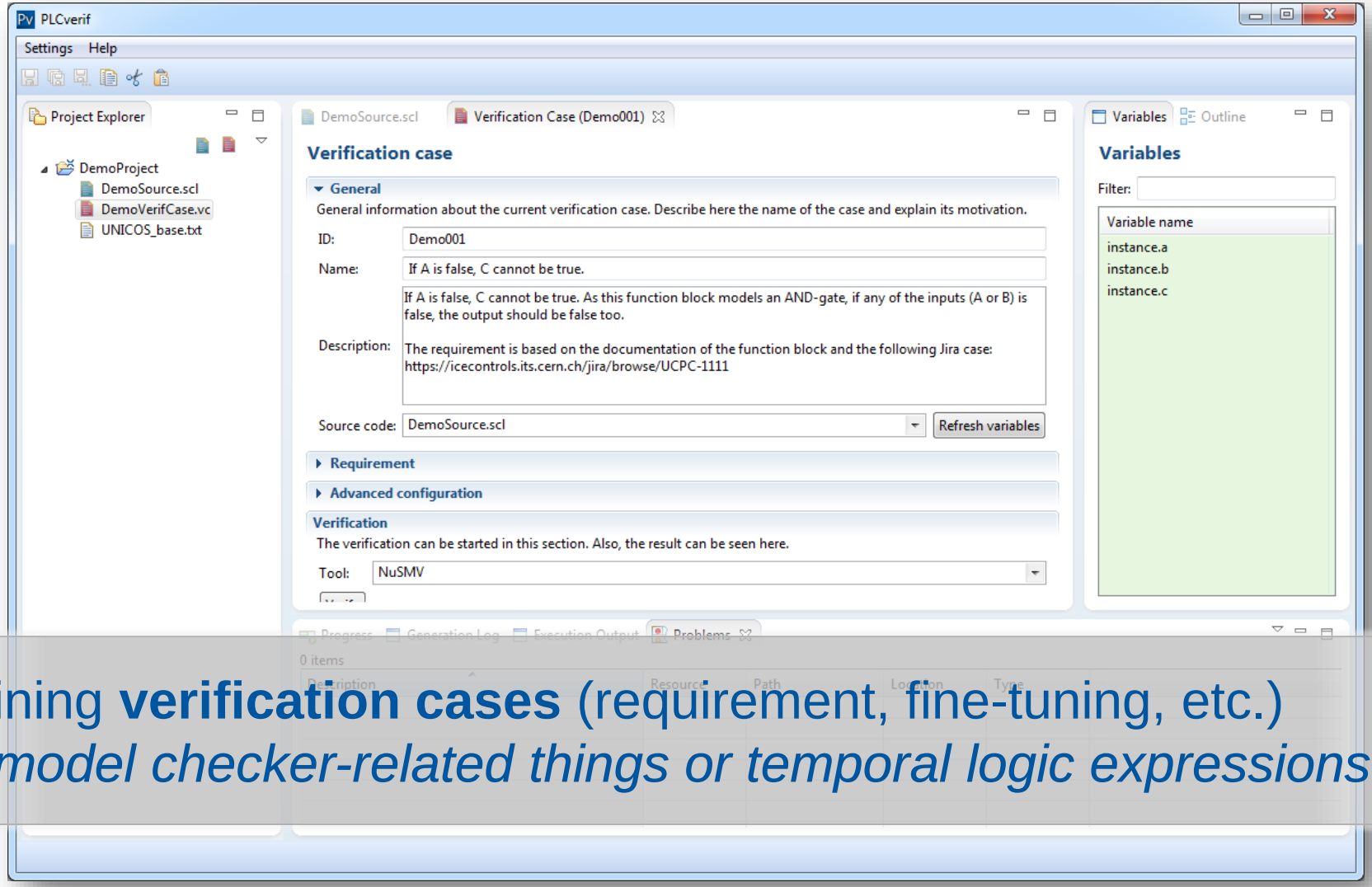
More info: B. Fernández et al. Bringing automated model checking to PLC program development - A CERN case study. In Proc. WODES 2014, pp. 394-399. IFAC, 2014. doi: 10.3182/20140514-3-FR-4046.00051

The PLCverif tool



Eclipse-based **editor for PLC programs**

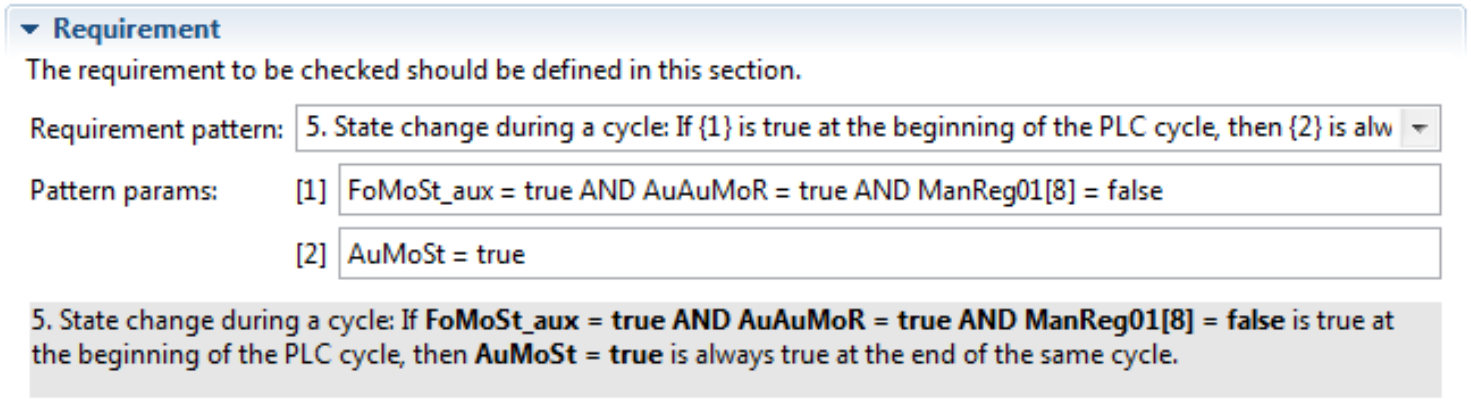
The PLCverif tool



Defining **verification cases** (requirement, fine-tuning, etc.)

No model checker-related things or temporal logic expressions

The PLCverif tool – Requirements



Instead of:

AG((PLC_START & (instance/fomost_aux = TRUE & instance/auaumor = TRUE & instance/manreg01b/8 = FALSE)) -> X(A[!PLC_END U PLC_END & (instance/aumost = TRUE)]))

Requirement patterns (needs no formal verification knowledge)

The PLCverif tool

PLCverif — Verification report

Generated at Mon Jul 07 15:19:22 CEST 2014 | PLCverif v2.0.1 | (C) CERN EN-ICE-PLC | Show/hide expert details

ID:	**Demo001**
Name:	If A is false, C cannot be true.
Description:	If A is false, C cannot be true. As this function block models an AND-gate, if any of the inputs (A or B) is false, the output should be false too. The requirement is based on the documentation of the function block and the following Jira case: https://icecontrols.its.cern.ch/jira/browse/UCPC-1111
Source file:	DemoSource.scl
Requirement:	3. A = false & C = true is impossible at the end of the PLC cycle.
Result:	**Not satisfied**

Tool: nusmv
Total runtime (until getting the verification results): 212 ms
Total runtime (incl. visualization): 361 ms

Counterexample

	Variable	End of **Cycle 1**
Input	**a**	FALSE
Input	**b**	TRUE
Output	**c**	TRUE

Click-button verification, verification **report** with the analysed counterexample

Example verification metrics

Each line represents the verification of a PLC program with a specific requirement.

	Source LOC	Unreduced PSS	Reduced PSS	Verification time (NuSMV)
(1)	12	24	24	0.04 s
(2)	1000	3.8×10^{242}	2.2×10^{8}	0.24 s
(3)	1000	3.8×10^{242}	5.8×10^{6}	0.23 s
(4)	17,700	10^{32446}	7.9×10^{35}	21.7 s
(5)	10,000	10^{978}	1.6×10^{84}	~7 min

Verification times measured on: Intel i7-3770, 8 GB RAM, Win 7 x64, Java 8
NuSMV 2.6.0 (physical PC)

Case study:
SM18 magnet testing facility

SM18 PLCSE safety controllers

Goal: ensuring **safety** by allowing/forbiddin

Core:

Safety-critical, can be dangerous: *14 kA, liquid He, –271°C, vacuum*

More info: D. Darvas, I. Majzik, E. Blanco. Formal verification of safety PLC based control software. In Integrated Formal Methods, LNCS 9681, pp. 508-522. Springer, 2016. doi: 10.1007/978-3-319-33693-0_32

Challenges in the verification

- **Complex**, semi-formal **(ambiguous) requirements**

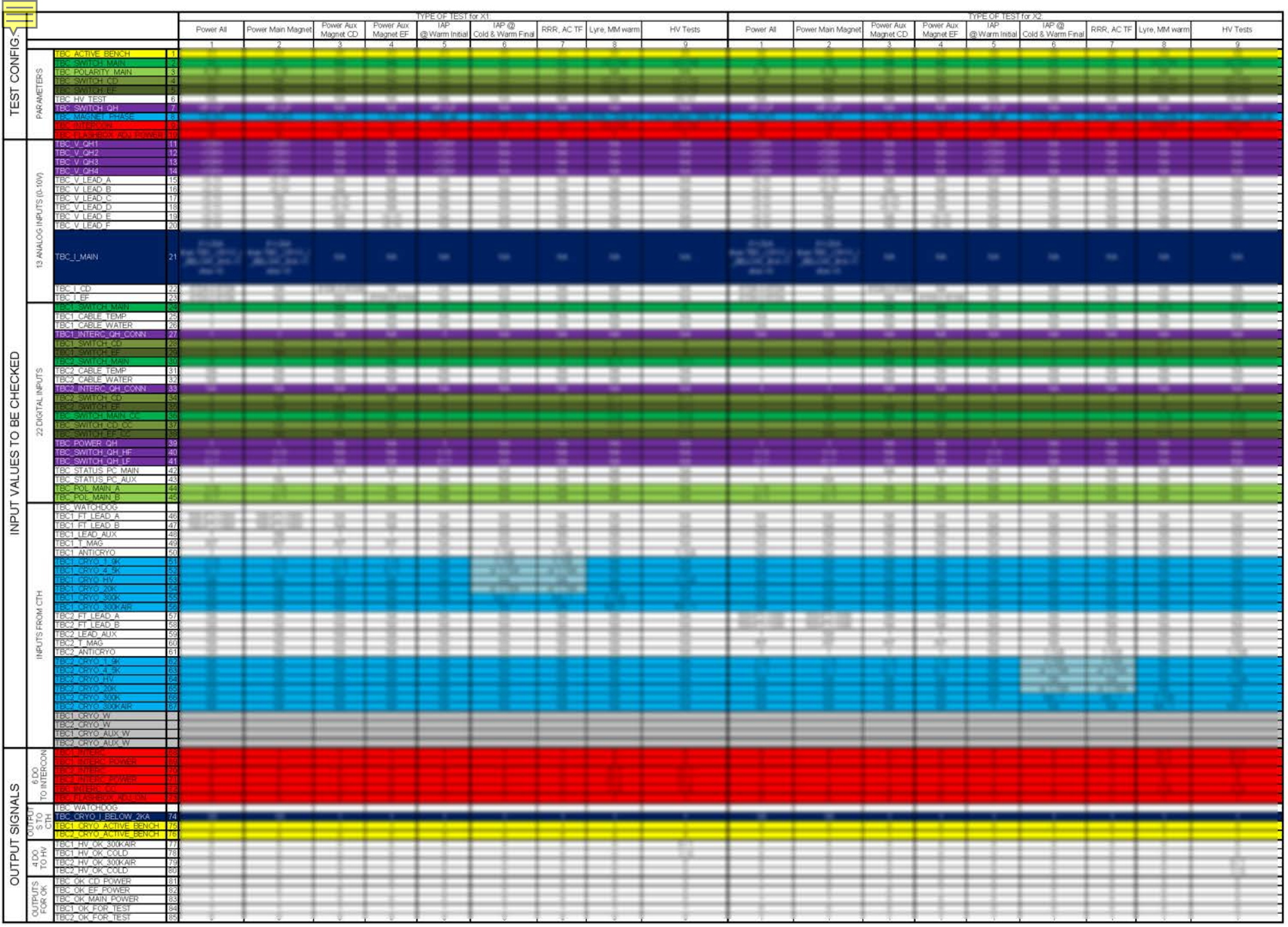
Semi-formal specification

- Allowed tests described by a **simple table**

Input	**Selected test**	1	2
	Voltage	>100 V	>50 V
	Overheating	FALSE	*don't care*
	Cryo OK	TRUE	TRUE
Output	**TestEnabled**	TRUE	TRUE
	SpecialTest	TRUE	FALSE

  - If `SelTest=1` and `Voltage>100` and not `Overh` and `CryoOk`, then `TestEnabled` shall be `true`, `SpecialTest` shall be `true`.

- Not bad, but ambiguous
  - Colours have undefined additional meanings
  - Some ambiguous values in cells, e.g. "1 / NA / NA / 0"



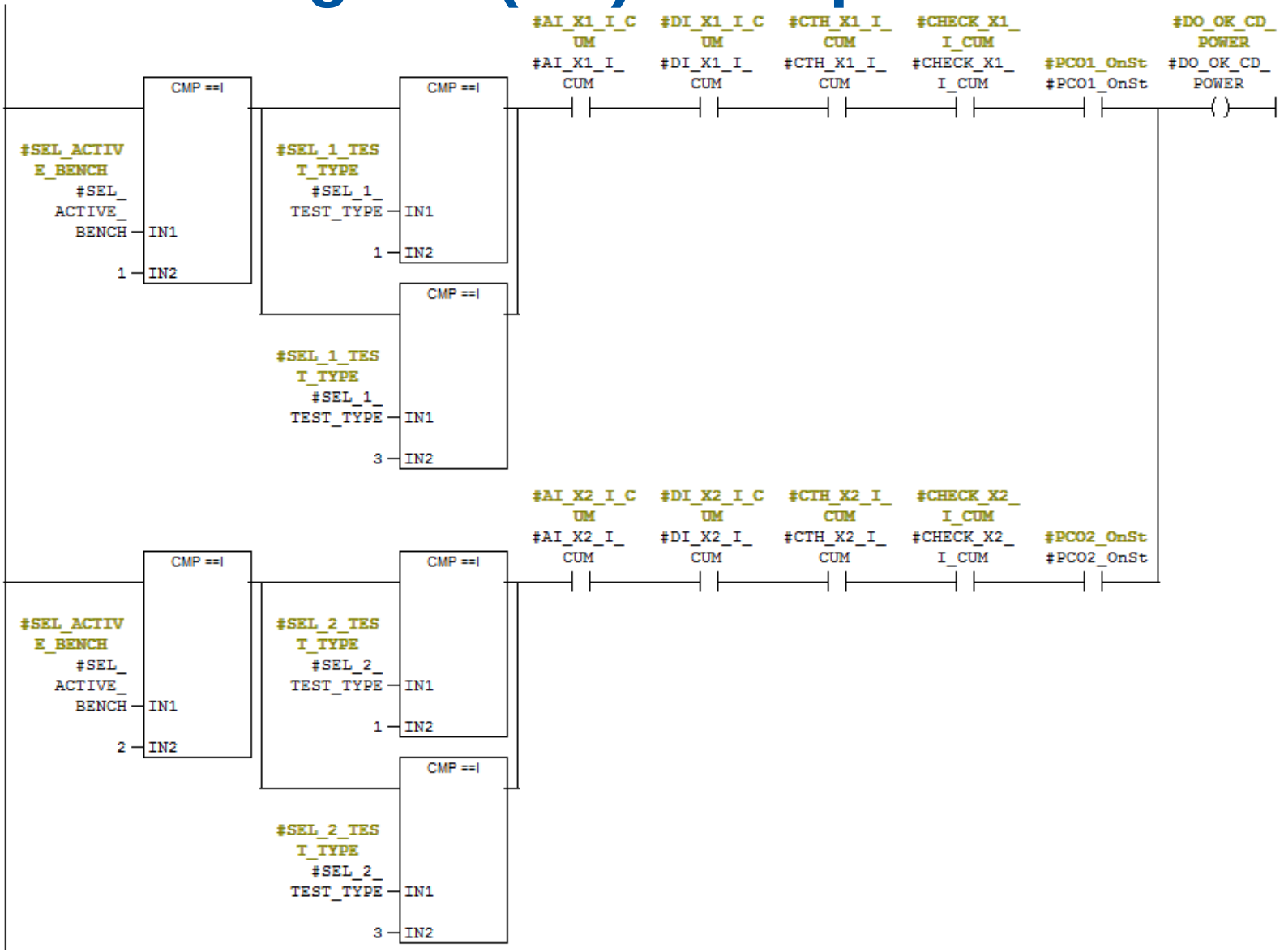
From M. Charrondiere

Challenges in the verification

- **Complex**, semi-formal **(ambiguous) requirements**

- **‘LD’ programming language**
  - Due to development restrictions for safety PLC programs
  - Has to be exported to ‘STL’ language first
  - Semantics of ‘STL’ is not precisely defined

Ladder Diagram (LD) example



Excerpt from the work of R. Speroni

Siemens Statement List (STL) example

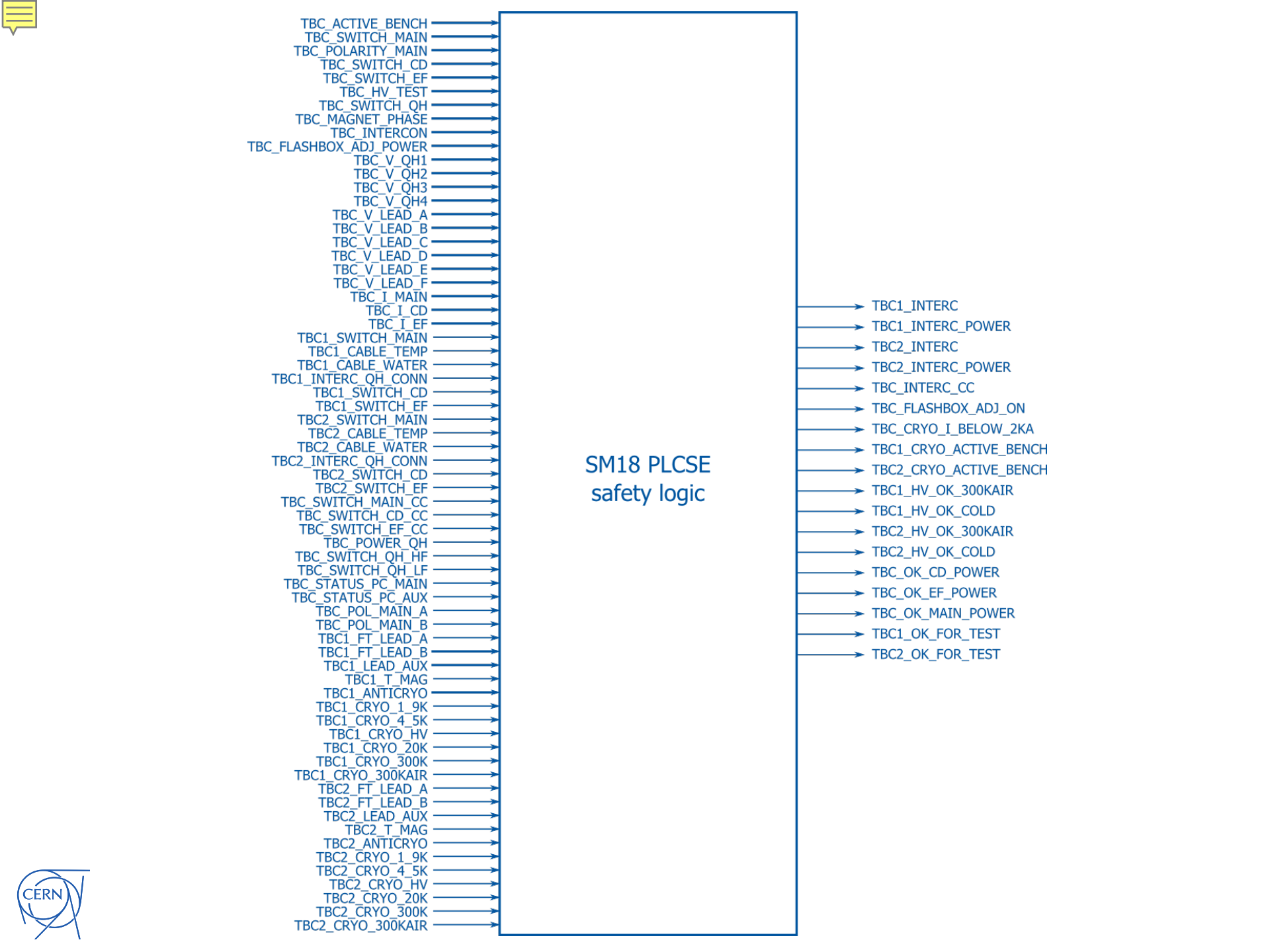
```
NETWORK
TITLE =POWER CD

      A(    ;
      L     #SEL_ACTIVE_BENCH;
      L     1;
      ==I   ;
      )     ;
      A(    ;
      O(    ;
      L     #SEL_TYPE_TEST_X1;
      L     1;
      ==I   ;
      )     ;
      O(    ;
      L     #SEL_TYPE_TEST_X1;
      L     3;
      ==I   ;
      )     ;
      )     ;
      A     #AI_X1_I_CUM;
      A     #DI_X1_I_CUM;
      A     #CTH_X1_I_CUM;
      O     ;
      A(    ;
      L     #SEL_ACTIVE_BENCH;
      L     2;
      ==I   ;
      )     ;
      A(    ;
      O(    ;
      L     #SEL_TYPE_TEST_X2;
      L     1;
      ==I   ;
      )     ;
      O(    ;
      L     #SEL_TYPE_TEST_X2;
      L     3;
      ==I   ;
      )     ;
      )     ;
      A     #AI_X2_I_CUM;
      A     #DI_X2_I_CUM;
      A     #CTH_X2_I_CUM;
      =     #DO_OK_CD_POWER;
```

Challenges in the verification

- **Complex**, semi-formal **(ambiguous) requirements**

- **‘LD’ programming language**
  - Due to development restrictions for safety PLC programs
  - Has to be exported to ‘STL’ language first
  - Semantics of ‘STL’ is not precisely defined

- **Complex safety logic**
  - Many inputs and outputs

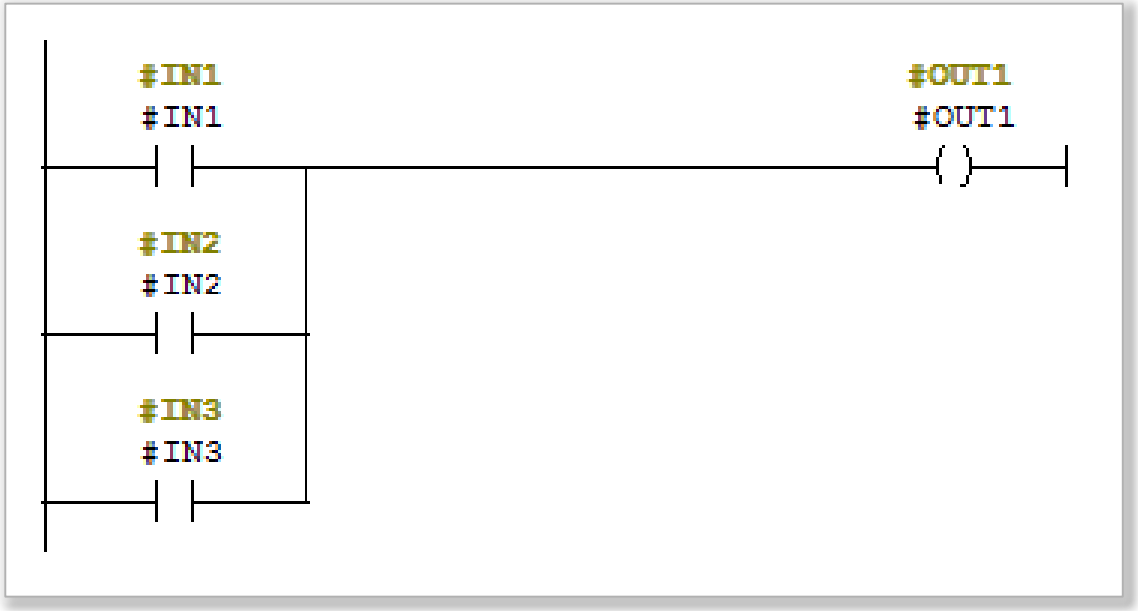

TBC_ACTIVE_BENCH
TBC_SWITCH_MAIN
TBC_POLARITY_MAIN
TBC_SWITCH_CD
TBC_SWITCH_EF
TBC_HV_TEST
TBC_SWITCH_QH
TBC_MAGNET_PHASE
TBC_INTERCON
TBC_FLASHBOX_ADJ_POWER
TBC_V_QH1
TBC_V_QH2
TBC_V_QH3
TBC_V_QH4
TBC_V_LEAD_A
TBC_V_LEAD_B
TBC_V_LEAD_C
TBC_V_LEAD_D
TBC_V_LEAD_E
TBC_V_LEAD_F
TBC_I_MAIN
TBC_I_CD
TBC_I_EF
TBC1_SWITCH_MAIN
TBC1_CABLE_TEMP
TBC1_CABLE_WATER
TBC1_INTERC_QH_CONN
TBC1_SWITCH_CD
TBC1_SWITCH_EF
TBC2_SWITCH_MAIN
TBC2_CABLE_TEMP
TBC2_CABLE_WATER
TBC2_INTERC_QH_CONN
TBC2_SWITCH_CD
TBC2_SWITCH_EF
TBC_SWITCH_MAIN_CC
TBC_SWITCH_CD_CC
TBC_SWITCH_EF_CC
TBC_POWER_QH
TBC_SWITCH_QH_HF
TBC_SWITCH_QH_LF
TBC_STATUS_PC_MAIN
TBC_STATUS_PC_AUX
TBC_POL_MAIN_A
TBC_POL_MAIN_B
TBC1_FT_LEAD_A
TBC1_FT_LEAD_B
TBC1_LEAD_AUX
TBC1_T_MAG
TBC1_ANTICRYO
TBC1_CRYO_1_9K
TBC1_CRYO_4_5K
TBC1_CRYO_HV
TBC1_CRYO_20K
TBC1_CRYO_300K
TBC1_CRYO_300KAIR
TBC2_FT_LEAD_A
TBC2_FT_LEAD_B
TBC2_LEAD_AUX
TBC2_T_MAG
TBC2_ANTICRYO
TBC2_CRYO_1_9K
TBC2_CRYO_4_5K
TBC2_CRYO_HV
TBC2_CRYO_20K
TBC2_CRYO_300K
TBC2_CRYO_300KAIR
SM18 PLCSE
safety logic
TBC1_INTERC
TBC1_INTERC_POWER
TBC2_INTERC
TBC2_INTERC_POWER
TBC_INTERC_CC
TBC_FLASHBOX_ADJ_ON
TBC_CRYO_I_BELOW_2KA
TBC1_CRYO_ACTIVE_BENCH
TBC2_CRYO_ACTIVE_BENCH
TBC1_HV_OK_300KAIR
TBC1_HV_OK_COLD
TBC2_HV_OK_300KAIR
TBC2_HV_OK_COLD
TBC_OK_CD_POWER
TBC_OK_EF_POWER
TBC_OK_MAIN_POWER
TBC1_OK_FOR_TEST
TBC2_OK_FOR_TEST

Reductions

How to make the verification of a model with ~10^{1000} states and 10,000+ lines of code feasible?

STL-to-SCL transformation

```
NETWORK
O IN1
O IN2
O IN3
= OUT1
```



STL-to-SCL transformation

```
NETWORK

O IN1

O IN2

O IN3

= OUT1
```

STL-to-SCL transformation

This transformation makes the side-effects explicit, causing a code blow up

```
// NETWORK (Start of STL network)
regMCR := TRUE;
regOR := FALSE; regSTA := TRUE; regRLO := TRUE; regNFC := FALSE;
// O IN1 (OR STL instruction)
IF regNFC = FALSE THEN regRLO := IN1; ELSE regRLO := regRLO OR IN1; END_IF;
regOR := FALSE; regSTA := IN1;   regNFC := TRUE;
// O IN2 (OR STL instruction)
IF regNFC = FALSE THEN regRLO := IN2; ELSE regRLO := regRLO OR IN2; END_IF;
regOR := FALSE; regSTA := IN2;   regNFC := TRUE;
// O IN3 (OR STL instruction)
IF regNFC = FALSE THEN regRLO := IN3; ELSE regRLO := regRLO OR IN3; END_IF;
regOR := FALSE; regSTA := IN3;   regNFC := TRUE;
// = OUT1 (STORE STL instruction)
IF regMCR THEN OUT1 := regRLO; END_IF;
regOR := FALSE; regSTA := OUT1;  regNFC := FALSE;
```

Reduction of the generated SCL code

In reality it is the intermediate model which is reduced, not the code…

```
// NETWORK (Start of STL network)
regMCR := TRUE;
regOR := FALSE; regSTA := TRUE; regRLO := TRUE; regNFC := FALSE;
// O IN1 (OR STL instruction)
IF regNFC = FALSE THEN regRLO := IN1; ELSE regRLO := regRLO OR IN1; END_IF;
regOR := FALSE; regSTA := IN1;   regNFC := TRUE;
// O IN2 (OR STL instruction)
IF regNFC = FALSE THEN regRLO := IN2; ELSE regRLO := regRLO OR IN2; END_IF;
regOR := FALSE; regSTA := IN2;   regNFC := TRUE;
// O IN3 (OR STL instruction)
IF regNFC = FALSE THEN regRLO := IN3; ELSE regRLO := regRLO OR IN3; END_IF;
regOR := FALSE; regSTA := IN3;   regNFC := TRUE;
// = OUT1 (STORE STL instruction)
IF regMCR THEN OUT1 := regRLO; END_IF;
regOR := FALSE; regSTA := OUT1;  regNFC := FALSE;
```

“Expression propagation”

```
// NETWORK (Start of STL network)
regMCR := TRUE;
regOR := FALSE; regSTA := TRUE; regRLO := TRUE; regNFC := FALSE;
// O IN1 (OR STL instruction)
IF regNFC = FALSE THEN regRLO := IN1; ELSE regRLO := regRLO OR IN1; END_IF;
regOR := FALSE; regSTA := IN1;  regNFC := TRUE;
// O IN2 (OR STL instruction)
IF regNFC = FALSE THEN regRLO := IN2; ELSE regRLO := regRLO OR IN2; END_IF;
regOR := FALSE; regSTA := IN2;  regNFC := TRUE;
// O IN3 (OR STL instruction)
IF regNFC = FALSE THEN regRLO := IN3; ELSE regRLO := regRLO OR IN3; END_IF;
regOR := FALSE; regSTA := IN3;   regNFC := TRUE;
// = OUT1 (STORE STL instruction)
IF regMCR THEN OUT1 := regRLO; END_IF;
regOR := FALSE; regSTA := OUT1;  regNFC := FALSE;
```

“Expression propagation”

```
// NETWORK (Start of STL network)
regMCR := TRUE;
regOR := FALSE; regSTA := TRUE; regRLO := TRUE; regNFC := FALSE;
// O IN1 (OR STL instruction)
IF      TRUE      THEN regRLO := IN1; ELSE regRLO := regRLO OR IN1; END_IF;
regOR := FALSE; regSTA := IN1;   regNFC := TRUE;
// O IN2 (OR STL instruction)
IF      FALSE     THEN regRLO := IN2; ELSE regRLO := regRLO OR IN2; END_IF;
regOR := FALSE; regSTA := IN2;   regNFC := TRUE;
// O IN3 (OR STL instruction)
IF      FALSE     THEN regRLO := IN3; ELSE regRLO := regRLO OR IN3; END_IF;
regOR := FALSE; regSTA := IN3;   regNFC := TRUE;
// = OUT1 (STORE STL instruction)
IF regMCR THEN OUT1 := regRLO; END_IF;
regOR := FALSE; regSTA := OUT1;  regNFC := FALSE;
```

“Dead branch elimination”

```
// NETWORK (Start of STL network)
regMCR := TRUE;
regOR := FALSE; regSTA := TRUE; regRLO := TRUE; regNFC := FALSE;
// O IN1 (OR STL instruction)
IF      TRUE      THEN regRLO := IN1; ELSE regRLO := regRLO OR IN1; END_IF;
regOR := FALSE; regSTA := IN1;   regNFC := TRUE;
// O IN2 (OR STL instruction)
IF      FALSE     THEN regRLO := IN2; ELSE regRLO := regRLO OR IN2; END_IF;
regOR := FALSE; regSTA := IN2;   regNFC := TRUE;
// O IN3 (OR STL instruction)
IF      FALSE     THEN regRLO := IN3; ELSE regRLO := regRLO OR IN3; END_IF;
regOR := FALSE; regSTA := IN3;   regNFC := TRUE;
// = OUT1 (STORE STL instruction)
IF regMCR THEN OUT1 := regRLO; END_IF;
regOR := FALSE; regSTA := OUT1;  regNFC := FALSE;
```

“Dead branch elimination”

```
// NETWORK (Start of STL network)
regMCR := TRUE;
regOR := FALSE; regSTA := TRUE; regRLO := TRUE; regNFC := FALSE;
// O IN1 (OR STL instruction)
regRLO := IN1;
regOR := FALSE; regSTA := IN1;   regNFC := TRUE;
// O IN2 (OR STL instruction)
regRLO := regRLO OR IN2;
regOR := FALSE; regSTA := IN2;   regNFC := TRUE;
// O IN3 (OR STL instruction)
regRLO := regRLO OR IN3;
regOR := FALSE; regSTA := IN3;   regNFC := TRUE;
// = OUT1 (STORE STL instruction)
IF regMCR THEN OUT1 := regRLO; END_IF;
regOR := FALSE; regSTA := OUT1;  regNFC := FALSE;
```

Non-read variable elimination (regNFC)

```
// NETWORK (Start of STL network)
regMCR := TRUE;
regOR := FALSE; regSTA := TRUE; regRLO := TRUE; regNFC := FALSE;
// O IN1 (OR STL instruction)
regRLO := IN1;
regOR := FALSE; regSTA := IN1;  regNFC := TRUE;
// O IN2 (OR STL instruction)
regRLO := regRLO OR IN2;
regOR := FALSE; regSTA := IN2;  regNFC := TRUE;
// O IN3 (OR STL instruction)
regRLO := regRLO OR IN3;
regOR := FALSE; regSTA := IN3;  regNFC := TRUE;
// = OUT1 (STORE STL instruction)
IF regMCR THEN OUT1 := regRLO; END_IF;
regOR := FALSE; regSTA := OUT1; regNFC := FALSE;
```

Non-read variable elimination (regNFC)

```
// NETWORK (Start of STL network)
regMCR := TRUE;
regOR := FALSE; regSTA := TRUE; regRLO := TRUE;
// O IN1 (OR STL instruction)
regRLO := IN1;
regOR := FALSE; regSTA := IN1;
// O IN2 (OR STL instruction)
regRLO := regRLO OR IN2;
regOR := FALSE; regSTA := IN2;
// O IN3 (OR STL instruction)
regRLO := regRLO OR IN3;
regOR := FALSE; regSTA := IN3;
// = OUT1 (STORE STL instruction)
IF regMCR THEN OUT1 := regRLO; END_IF;
regOR := FALSE; regSTA := OUT1;
```

Similar reductions (regMCR)

```
// NETWORK (Start of STL network)
regMCR := TRUE;
regOR := FALSE; regSTA := TRUE; regRLO := TRUE;
// O IN1 (OR STL instruction)
regRLO := IN1;
regOR := FALSE; regSTA := IN1;
// O IN2 (OR STL instruction)
regRLO := regRLO OR IN2;
regOR := FALSE; regSTA := IN2;
// O IN3 (OR STL instruction)
regRLO := regRLO OR IN3;
regOR := FALSE; regSTA := IN3;
// = OUT1 (STORE STL instruction)
IF regMCR THEN OUT1 := regRLO; END_IF;
regOR := FALSE; regSTA := OUT1;
```

Similar reductions (regMCR)

```
// NETWORK (Start of STL network)

regOR := FALSE; regSTA := TRUE; regRLO := TRUE;
// O IN1 (OR STL instruction)
regRLO := IN1;
regOR := FALSE; regSTA := IN1;
// O IN2 (OR STL instruction)
regRLO := regRLO OR IN2;
regOR := FALSE; regSTA := IN2;
// O IN3 (OR STL instruction)
regRLO := regRLO OR IN3;
regOR := FALSE; regSTA := IN3;
// = OUT1 (STORE STL instruction)
OUT1 := regRLO;
regOR := FALSE; regSTA := OUT1;
```

Non-read variable elimination (regOR)

```
// NETWORK (Start of STL network)

regOR := FALSE; regSTA := TRUE; regRLO := TRUE;
// O IN1 (OR STL instruction)
regRLO := IN1;
regOR := FALSE; regSTA := IN1;
// O IN2 (OR STL instruction)
regRLO := regRLO OR IN2;
regOR := FALSE; regSTA := IN2;
// O IN3 (OR STL instruction)
regRLO := regRLO OR IN3;
regOR := FALSE; regSTA := IN3;
// = OUT1 (STORE STL instruction)
OUT1 := regRLO;
regOR := FALSE; regSTA := OUT1;
```

Non-read variable elimination (regOR)

```
// NETWORK (Start of STL network)

regSTA := TRUE; regRLO := TRUE;
// O IN1 (OR STL instruction)
regRLO := IN1;
regSTA := IN1;
// O IN2 (OR STL instruction)
regRLO := regRLO OR IN2;
regSTA := IN2;
// O IN3 (OR STL instruction)
regRLO := regRLO OR IN3;
regSTA := IN3;
// = OUT1 (STORE STL instruction)
OUT1 := regRLO;
regSTA := OUT1;
```

Non-read variable elimination (regSTA)

```
// NETWORK (Start of STL network)

regSTA := TRUE; regRLO := TRUE;
// O IN1 (OR STL instruction)
regRLO := IN1;
regSTA := IN1;
// O IN2 (OR STL instruction)
regRLO := regRLO OR IN2;
regSTA := IN2;
// O IN3 (OR STL instruction)
regRLO := regRLO OR IN3;
regSTA := IN3;
// = OUT1 (STORE STL instruction)
OUT1 := regRLO;
regSTA := OUT1;
```

Non-read variable elimination (regSTA)

```
// NETWORK (Start of STL network)

regRLO := TRUE;
// O IN1 (OR STL instruction)
regRLO := IN1;

// O IN2 (OR STL instruction)
regRLO := regRLO OR IN2;

// O IN3 (OR STL instruction)
regRLO := regRLO OR IN3;

// = OUT1 (STORE STL instruction)
OUT1 := regRLO;
```

“Expression propagation”

```
// NETWORK (Start of STL network)

regRLO := TRUE;
// O IN1 (OR STL instruction)
regRLO := IN1;

// O IN2 (OR STL instruction)
regRLO := regRLO OR IN2;

// O IN3 (OR STL instruction)
regRLO := regRLO OR IN3;

// = OUT1 (STORE STL instruction)
OUT1 := regRLO;
```

“Expression propagation”

```
// NETWORK (Start of STL network)

regRLO := TRUE;
// O IN1 (OR STL instruction)


// O IN2 (OR STL instruction)
regRLO :=  IN1   OR IN2;

// O IN3 (OR STL instruction)
regRLO := regRLO OR IN3;

// = OUT1 (STORE STL instruction)
OUT1 := regRLO;
```

“Expression propagation”

```
// NETWORK (Start of STL network)

regRLO := TRUE;
// O IN1 (OR STL instruction)



// O IN2 (OR STL instruction)
regRLO :=  IN1   OR IN2;

// O IN3 (OR STL instruction)
regRLO := regRLO OR IN3;

// = OUT1 (STORE STL instruction)
OUT1 := regRLO;
```

“Expression propagation”

```
// NETWORK (Start of STL network)

regRLO := TRUE;
// O IN1 (OR STL instruction)


// O IN2 (OR STL instruction)


// O IN3 (OR STL instruction)
regRLO := IN1 OR IN2 OR IN3;

// = OUT1 (STORE STL instruction)
OUT1 := regRLO;
```

“Expression propagation”

```
// NETWORK (Start of STL network)

regRLO := TRUE;
// O IN1 (OR STL instruction)


// O IN2 (OR STL instruction)


// O IN3 (OR STL instruction)
regRLO := IN1 OR IN2 OR IN3;

// = OUT1 (STORE STL instruction)
OUT1 := regRLO;
```

“Expression propagation”

```
// NETWORK (Start of STL network)

regRLO := TRUE;
// O IN1 (OR STL instruction)


// O IN2 (OR STL instruction)


// O IN3 (OR STL instruction)


// = OUT1 (STORE STL instruction)
OUT1 := IN1 OR IN2 OR IN3;
```

Non-read variable elimination (regRLO)

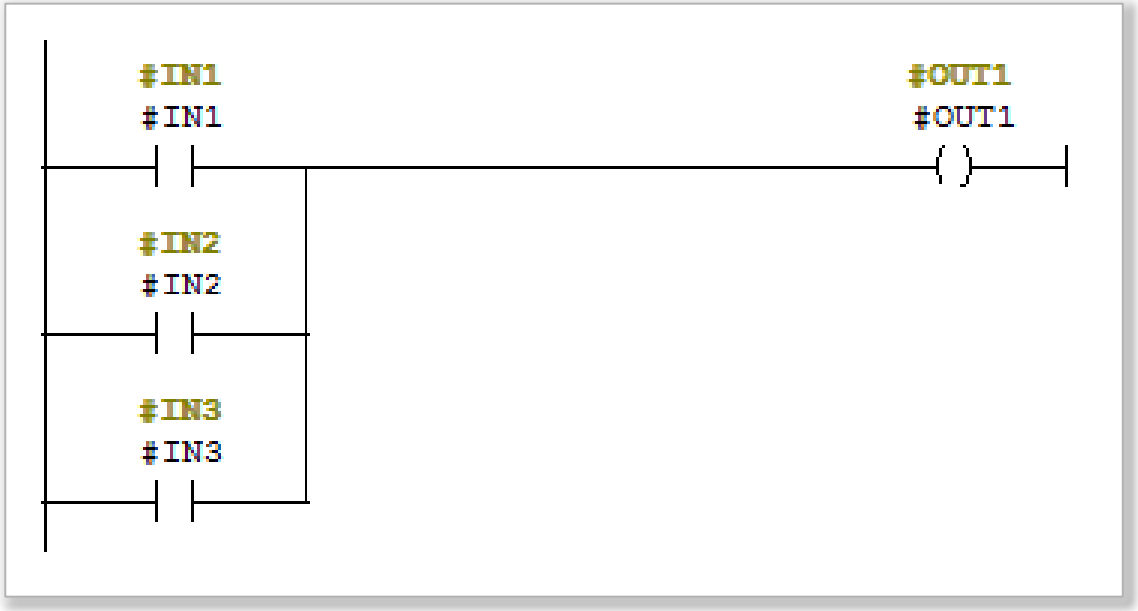
```
// NETWORK (Start of STL network)

regRLO := TRUE;
// O IN1 (OR STL instruction)


// O IN2 (OR STL instruction)


// O IN3 (OR STL instruction)


// = OUT1 (STORE STL instruction)
OUT1 := IN1 OR IN2 OR IN3;
```

Non-read variable elimination (regRLO)

```
// NETWORK (Start of STL network)


// O IN1 (OR STL instruction)


// O IN2 (OR STL instruction)


// O IN3 (OR STL instruction)


// = OUT1 (STORE STL instruction)
OUT1 := IN1 OR IN2 OR IN3;
```

Result of the reductions

```
OUT1 := IN1 OR IN2 OR IN3;
```



Problems found *(before putting in production!)*

Requirement misunderstanding

- Recognised while specifying requirements formally

Functionality problems

- “The [magnet] test should start, but it doesn’t.”

Safety problems

- “The [magnet] test **should NOT start**, but it does.”

Problems found

In total **14 issues** found

4 requirement misunderstandings

6 problems could not be found using our typical testing methods

Summary

Where are we now?

- **Model checking**: more and more used for real cases
  - Sometimes non-expert users use PLCverif **autonomously**
  - **Integration** into the development process is in progress
- Several **successful case studies**
  - Model checking revealed **interesting and potentially critical problems**
  - **Counterexample** is a huge advantage
- **Improvements** are always possible
  - New reduction methods
  - Support for new model checkers
  - Support for additional PLC languages

CERN

www.cern.ch

For more information…

- **Project** website (with publication list)
  http://cern.ch/project-plc-formalmethods/
- **PLCverif** tool's website
  http://cern.ch/plcverif
- **CERN** website – http://home.cern

Model checking at CERN

- D. Darvas et al. **Formal verification of complex properties on PLC programs**. Formal Techniques for Distributed Objects, Components, and Systems (LNCS 8461), pp. 284-299, Springer, 2014.
- B. Fernández et al. **Bringing automated model checking to PLC program development – A CERN case study**. Proc. of the 12th Int. Workshop on Discrete Event Systems, pp. 394-399, 2014.
- D. Darvas et al. **PLCverif: A tool to verify PLC programs based on model checking techniques.** Proc. of the 15th Int. Conf. on Accelerator & Large Experimental Physics Control Systems, pp. 911-914, JaCoW, 2015. http://doi.org/10.18429/JACoW-ICALEPCS2015-WEPGF092
- B. Fernández et al. **Applying model checking to industrial-sized PLC programs.** IEEE Transactions on Industrial Informatics, 11(6):1400-1410, 2015. http://doi.org/10.1109/TII.2015.2489184
- D. Darvas et al. **Formal verification of safety PLC based control software.** Integrated Formal Methods (LNCS 9681), pp. 508-522, Springer, 2016. http://doi.org/10.1007/978-3-319-33693-0_32

Formal specification at CERN

- D. Darvas et al. **Requirements towards a formal specification language for PLCs.** 2014. http://doi.org/10.5281/zenodo.14907
- D. Darvas et al. **A formal specification method for PLC-based applications.** Proc. of the 15th Int. Conf. on Accelerator & Large Experimental Physics Control Systems, pp. 907-910, JaCoW, 2015. http://dx.doi.org/10.18429/JACoW-ICALEPCS2015-WEPGF091
- D. Darvas et al. **Syntax and semantics of PLCspecif.** CERN Report, EDMS 1523877, 2015. https://edms.cern.ch/document/1523877
- D. Darvas et al. **Formal verification of safety PLC based control software.** Integrated Formal Methods (LNCS 9681), pp. 508-522, Springer, 2016. http://doi.org/10.1007/978-3-319-33693-0_32