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;

**BEGIN**
- **out1**: = NOT in1;

**END_FUNCTION_BLOCK**

**DATA_BLOCK** inst Test

**BEGIN**

**END_DATA_BLOCK**

### "Equivalent" Java code

```java
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;
    }
}
```

```java
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
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
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;
```

![](image)
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 \( \alpha \) and \( \beta \) are true, then \( \alpha \) shall stay true until \( \beta \) becomes true.

\[ AG((\alpha \land \beta) \rightarrow A[\alpha U \neg\beta]) \]
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

PLC program → Intermediate model

Reductions

Intermediate model → Formal requirement → Requirement patterns → Model checker

Model checker:
- Satisfied
- Not satisfied → Counter-example

Verification report

Based on the implementation

User-friendly requirement specification

Heavily automated reductions

Replaceable external model checker

Self-contained report with counterexample

Overview of the workflow

Based on the implementation
User-friendly requirement specification
Heavily automated reductions
Replaceable external model checker
Self-contained report with counterexample
Tool hiding the formal details

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

Requirement patterns (needs no formal verification knowledge)

Instead of:

\[ AG((\text{PLC\_START} \& (\text{instance/fomost\_aux} = \text{TRUE} \& \text{instance/auaumor} = \text{TRUE} \& \text{instance/manreg01b/8} = \text{FALSE})) \rightarrow X(\text{!PLC\_END} U \text{PLC\_END} \& (\text{instance/aumost} = \text{TRUE}) )) \]
The PLCverif tool

PLCverif — Verification report

Generated at Mon Jul 07 15:19:22 CEST 2014 | PLCverif v2.0.11 | (C) CERN EN-ICE-PLC | Show/hide expert details

<table>
<thead>
<tr>
<th>ID:</th>
<th>Demo001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td>If A is false, C cannot be true.</td>
</tr>
<tr>
<td>Description:</td>
<td>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.</td>
</tr>
<tr>
<td>Source file:</td>
<td>DemoSource.scl</td>
</tr>
<tr>
<td>Requirement:</td>
<td>3. A = false &amp; C = true is impossible at the end of the PLC cycle.</td>
</tr>
<tr>
<td>Result:</td>
<td>Not satisfied</td>
</tr>
</tbody>
</table>

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

Counterexample

<table>
<thead>
<tr>
<th>Variable</th>
<th>End of Cycle 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input a</td>
<td>FALSE</td>
</tr>
<tr>
<td>Input b</td>
<td>TRUE</td>
</tr>
<tr>
<td>Output c</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Source LOC</th>
<th>Unreduced PSS</th>
<th>Reduced PSS</th>
<th>Verification time (NuSMV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) 12</td>
<td>24</td>
<td>24</td>
<td>0.04 s</td>
</tr>
<tr>
<td>(2) 1000</td>
<td>$3.8 \times 10^{242}$</td>
<td>$2.2 \times 10^8$</td>
<td>0.24 s</td>
</tr>
<tr>
<td>(3) 1000</td>
<td>$3.8 \times 10^{242}$</td>
<td>$5.8 \times 10^6$</td>
<td>0.23 s</td>
</tr>
<tr>
<td>(4) 17,700</td>
<td>$10^{32446}$</td>
<td>$7.9 \times 10^{35}$</td>
<td>21.7 s</td>
</tr>
<tr>
<td>(5) 10,000</td>
<td>$10^{978}$</td>
<td>$1.6 \times 10^{84}$</td>
<td>~7 min</td>
</tr>
</tbody>
</table>

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/forbidding tests

Core:
- selected test
- switch statuses
- current voltages
- cryo conditions

SM18 PLCSE safety logic

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

Challenges in the verification

- Complex, semi-formal (ambiguous) requirements
Semi-formal specification

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

<table>
<thead>
<tr>
<th>Input</th>
<th>Selected test</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Voltage</strong></td>
<td>&gt;100 V</td>
<td>&gt;50 V</td>
<td></td>
</tr>
<tr>
<td><strong>Overheating</strong></td>
<td>FALSE</td>
<td></td>
<td><strong>don’t care</strong></td>
</tr>
<tr>
<td><strong>Cryo OK</strong></td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>TestEnabled</th>
<th>TRUE</th>
<th>TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SpecialTest</strong></td>
<td>TRUE</td>
<td>FALSE</td>
<td></td>
</tr>
</tbody>
</table>

- Not bad, but ambiguous
  - Colours have undefined additional meanings
  - Some ambiguous values in cells, e.g. “1 / NA / NA / 0”
From M. Charronniere
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

```plaintext
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;

0 ;
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
SM18 PLCSE safety logic

- 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_1_MAIN
- TBC_1_CD
- TBC_1_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
- TBC1_INTERC
- TBC1_INTERC_POWER
- TBC2_INTERC
- TBC2_INTERC_POWER
- TBC_INTERC_CC
- TBC_FLASHBOX_adj_ON
- TBC_CRYO_1_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
0 IN1
0 IN2
0 IN3
= OUT1
STL-to-SCL transformation

NETWORK

0 IN1

0 IN2

0 IN3

= OUT1
STL-to-SCL transformation

// 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

// 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;

In reality it is the intermediate model which is reduced, not the code...
“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;

// O IN2 (OR STL instruction)
 IF regNFC = FALSE THEN regRLO := IN2; ELSE regRLO := regRLO OR IN2; END_IF;
regOR := FALSE; regSTA := IN2;

// O IN3 (OR STL instruction)
 IF regNFC = FALSE THEN regRLO := IN3; ELSE regRLO := regRLO OR IN3; END_IF;
regOR := FALSE; regSTA := IN3;

// = 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;
// 0 IN1 (OR STL instruction)
IF TRUE THEN regRLO := IN1; ELSE regRLO := regRLO OR IN1; END_IF;
regOR := FALSE; regSTA := IN1; regNFC := TRUE;
// 0 IN2 (OR STL instruction)
IF FALSE THEN regRLO := IN2; ELSE regRLO := regRLO OR IN2; END_IF;
regOR := FALSE; regSTA := IN2; regNFC := TRUE;
// 0 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; // NETWORK (Start of STL network)
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;
// 0 IN1 (OR STL instruction)
regRLO := IN1;
regOR := FALSE; regSTA := IN1;
// 0 IN2 (OR STL instruction)
regRLO := regRLO OR IN2;
regOR := FALSE; regSTA := IN2;
// 0 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;
// 0 IN1 (OR STL instruction)
regRLO := IN1;
regOR := FALSE; regSTA := IN1;
// 0 IN2 (OR STL instruction)
regRLO := regRLO OR IN2;
regOR := FALSE; regSTA := IN2;
// 0 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;

// 0 IN1 (OR STL instruction)
regRLO := IN1;
regOR := FALSE; regSTA := IN1;

// 0 IN2 (OR STL instruction)
regRLO := regRLO OR IN2;
regOR := FALSE; regSTA := IN2;

// 0 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)

```plaintext
regOR := FALSE;  regSTA := TRUE;  regRLO := TRUE;
// 0 IN1 (OR STL instruction)
regRLO := IN1;
regOR := FALSE;  regSTA := IN1;
// 0 IN2 (OR STL instruction)
regRLO := regRLO OR IN2;
regOR := FALSE;  regSTA := IN2;
// 0 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;

// 0 IN1 (OR STL instruction)
regRLO := IN1;

// 0 IN2 (OR STL instruction)
regRLO := regRLO OR IN2;

// 0 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)

regRLO := IN1 OR IN2;

// O IN2 (OR STL instruction)

regRLO := regRLO OR IN3;

// = OUT1 (STORE STL instruction)

OUT1 := regRLO;
“Expression propagation”

// NETWORK (Start of STL network)

regRLO := TRUE;
// 0 IN1 (OR STL instruction)

// 0 IN2 (OR STL instruction)
regRLO := IN1 OR IN2;

// 0 IN3 (OR STL instruction)
regRLO := regRLO OR IN3;

// = OUT1 (STORE STL instruction)
OUT1 := regRLO;
“Expression propagation”

// NETWORK (Start of STL network)

regRLO := TRUE;
// 0 IN1 (OR STL instruction)
// 0 IN2 (OR STL instruction)
// 0 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;
// 0 IN1 (OR STL instruction)

// 0 IN2 (OR STL instruction)

// 0 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;

// 0 IN1 (OR STL instruction)

// 0 IN2 (OR STL instruction)

// 0 IN3 (OR STL instruction)

// = OUT1 (STORE STL instruction)

OUT1 := IN1 OR IN2 OR IN3;
Non-read variable elimination (regRLO)

// NETWORK (Start of STL network)

// 0 IN1 (OR STL instruction)

// 0 IN2 (OR STL instruction)

// 0 IN3 (OR STL instruction)

// = OUT1 (STORE STL instruction)
OUT1 := IN1 OR IN2 OR IN3;
Result of the reductions

\[
\text{OUT1} := \text{IN1 OR IN2 OR IN3};
\]
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
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


Formal specification at CERN


