Dániel Darvas (CERN / TU Budapest)

Practice-oriented formal methods for PLC programs of industrial control systems

PhD Symposium at iFM2016
05/06/2016, Reykjavík, Iceland

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

http://go.cern.ch/6Lgf
CERN  *European Org. for Nuclear Research*

- Largest *particle physics laboratory*
- Large Hadron Collider (LHC)
  - Proton* beams with high energies

© CERN
Industrial control systems (@ CERN)

**PLC**: Programmable Logic Controller

Control hardware

Control software

Physical process (plant)

measure

manipulate

@CERN:
- Cryogenics,
- Vacuum,
- Cooling and ventilation,
- Gas mixture, …
Verification for medium-critical systems

Resources for verification vs. Criticality

We are here.
Verification for medium-critical systems

What is the proper verification solution here?
Three challenges of the domain

- **Medium criticality**
  - Cannot afford special verification experts
  - Developers do the verification too

- **Typically non-IT background**
  - Cannot build on "base IT knowledge" (mathematics, UML, …)

- **Special devices**
  - Special programming languages
  - Lack of tools
  - Behind mainstream IT
How to ensure the quality of the software?

- **Testing**
  - Traditional, but not enough

- **Formal methods**
  - Without involving formal methods experts!
## Three proposed approaches

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>✦</td>
<td>✦</td>
<td></td>
<td>Model checking</td>
</tr>
<tr>
<td>#2</td>
<td>✦</td>
<td>✦</td>
<td>✦</td>
<td>Conformance checking</td>
</tr>
<tr>
<td>#3</td>
<td>✦</td>
<td></td>
<td>✦</td>
<td>Code generation</td>
</tr>
</tbody>
</table>
Challenges

- **Approach #1 (Model checking)**
  - Difficulty of using model checkers
  - Verification performance on huge models
  - Specifying requirements in temporal logic

- **Approach #2 (Conformance checking)**
  - Difficulty of using formal specification methods
  - Too strict conformance relations

- **Approach #3 (Code generation)**
  - Generated code should be understandable, modifiable
  - Generated code should match the specification
  - Difficult to describe invariant properties
Appr. #1: Model checking (PLCverif)

- PLC code
- Requirement patterns
- Formal model
- Formal requirement
- Model checker
- Satisfied
- Not satisfied
- Counter-example
- Verification report

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

Reductions
Appr. #1: Model checking (PLCverif)

PLC code

Requirement patterns

Verification report

- 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
PLCverif

- ST code and verification case editor
- One-click verification
- Multiple model checkers under hood
- Verification report

http://cern.ch/plcverif
Challenges

- **Approach #1 (Model checking)**
  - Difficulty of using model checkers
  - Verification performance on huge models
  - Specifying requirements in temporal logic

- **Approach #2 (Conformance checking)**
  - Difficulty of using formal specification methods
  - Too strict conformance relations

- **Approach #3 (Code generation)**
  - Generated code should be understandable, modifiable
  - Generated code should match the specification
  - Difficult to describe invariant properties
PLCspecif (example)

**Example Module**

**Assigned inputs:**
- ValueReq : INT16
- EnableReq_fromLogic : BOOL
- EnableReq_fromScada : BOOL
- EnableReq_fromField : BOOL
- DisableReq : BOOL
- PMin : INT16 param
- PMax : INT16 param

**Assigned outputs:**
- Value : INT16
- Status : BOOL

**Input definitions:** — (none)

**Event definitions:**
- @disable <- rising_edge(DisableReq) (pri=1)
- @enable <- EnableReq_fromLogic OR EnableReq_fromScada
  OR EnableReq_fromField (pri=2)

**Core logic (state machine)**

- Enabled
- Disabled

**Output definitions:**

1. \( _\text{Value} = \begin{array}{c|c|c|c|c}
    \text{ValueReq} & < \text{PMin} & > \text{PMax} & \text{result} \\
    \hline
    T & \cdot & \text{PMin} \\
    F & F & \text{PMax} \\
    F & T & \text{ValueReq} \\
  \end{array} \)

2. \( \text{Value} = \text{in\_state}(\text{Enabled}) \) result

3. \( \text{Status} = \text{in\_state}(\text{Enabled}) \)

**Invariant properties:**

- ALWAYS \( \text{PMin} \leq \text{Value} \leq \text{PMax} \) ASSUMING \( \text{PMin} \leq \text{PMax} \)

**Details:**

- Detailed behaviour specification
- Structured, hierarchical
- Separation of concerns
- Domain-specific semantics
- Unifies different semi-formal formalisms
- Supports verification
Appr. #2: Formal specification + conformance checking

- PLC code
  - Formal verif. model
    - Reductions
    - Composite model
    - Model checker
      - Satisfied
      - Not satisfied
        - ✓
        - Counter-example
          - Verification report
  - Formal specification
    - Formal verif. model
      - Reductions
      - Conformance relation
Challenges

- **Approach #1 (Model checking)**
  - Difficulty of using model checkers
  - Verification performance on huge models
  - Specifying requirements in temporal logic

- **Approach #2 (Conformance checking)**
  - Difficulty of using formal specification methods
  - Too strict conformance relations

- **Approach #3 (Code generation)**
  - Generated code should be understandable, modifiable
  - Generated code should match the specification
  - Difficult to describe invariant properties
Appr. #3: Code generation

- Code generation defined based on the **formal semantics** of PLC specifications

- **Special needs:**
  - Configurable code generation
  - Possibility to edit manually!
    - Understandable code needed
    - Conformance checking required

- Not possible for **fail-safe** PLCs
Which one is the best? – Comparison

− “Coverage”
  • *Model checking*: only the specified requirements
  • *Other methods*: complete specification

− “Intrusiveness”
  • *Model checking*: development process is extended only
  • *Conformance checking*: not intrusive, but formal specification needed
  • *Code generation*: radical change with mixed feelings

− Complementary methods for different purposes
Challenges and our solutions

- **Approach #1 (Model checking)**
  - Wrappers for model checkers, generic intermediate model
  - Generic and domain-specific reductions
  - Requirement patterns

- **Approach #2 (Conformance checking)**
  - Building on semi-formal languages with unified semantics
  - Permissive conformance relations

- **Approach #3 (Code generation)**
  - Customisable code generation
  - Code generation based on the formal semantics
  - Additional support for defining and checking invariants
Use case: Magnet testing plant

- **Goal**: ensuring safety by allowing/forbidding tests
- **Verified in development phase**
- **Problems found**:
  - Requirement misunderstanding
  - Functionality problems
  - Safety problems (some well-hidden problems)

→ More details in our iFM2016 paper
For more information…

− Project website (with publication list)
  http://cern.ch/project-plc-formalmethods/

− PLCverif tool’s website
  http://cern.ch/plcverif

− PLCspecif’s website
  http://cern.ch/plcspecif

− CERN website – http://home.cern

− Contact me
  daniel.darvas@cern.ch
  http://cern.ch/ddarvas
Model checking at CERN


D. Darvas et al. **Formal verification of safety PLC based control software.** Integrated Formal Methods (LNCS 9681), pp. 508-522, Springer, 2016. [http://dx.doi.org/10.1007/978-3-319-33693-0_32](http://dx.doi.org/10.1007/978-3-319-33693-0_32)