Automated Formal Verification of PLC (Programmable Logic Controller) Programs

ICS – ESTEREL meeting
21/01/2016
Outline

- Context – Idea (goal)
- Background & state of the art
- Approach - Novelty
- State of the project – results
- Conclusions
Context

- ICS group: Industrial Control and Safety.
- PLC control systems:
  - Standard and Safety Instrumented Systems.

Need: Guarantee that the PLC code is compliant with the specifications.

Currently: manual and automated testing
- Useful, but not efficient for every type of requirements.
- Difficult to test safety requirements:
  "if out1 is true, out2 should be false"
**Context**

Even for SIL1 it is recommended to use [Semi]-formal methods

**IEC 61508: Software design and dev. (table A.2)**

<table>
<thead>
<tr>
<th>Technique/Measure</th>
<th>Ref</th>
<th>SIL1</th>
<th>SIL2</th>
<th>SIL3</th>
<th>SIL4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Fault detection and diagnosis</td>
<td>C.3.1</td>
<td>---</td>
<td>R</td>
<td>R</td>
<td>HR</td>
</tr>
<tr>
<td>2 Error detecting and correcting codes</td>
<td>C.3.2</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>HR</td>
</tr>
<tr>
<td>3a Failure assertion programming</td>
<td>C.3.3</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>HR</td>
</tr>
<tr>
<td>3b Safety bag techniques</td>
<td>C.3.4</td>
<td>---</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>3c Diverse programming</td>
<td>C.3.5</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>HR</td>
</tr>
<tr>
<td>3d Recovery block</td>
<td>C.3.6</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>3e Backward recovery</td>
<td>C.3.7</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>3f Forward recovery</td>
<td>C.3.8</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>3g Re-try fault recovery mechanisms</td>
<td>C.3.9</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>HR</td>
</tr>
<tr>
<td>3h Memorising executed cases</td>
<td>C.3.10</td>
<td>---</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>4 Graceful degradation</td>
<td>C.3.11</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>5 Artificial intelligence - fault correction</td>
<td>C.3.12</td>
<td>---</td>
<td>NR</td>
<td>NR</td>
<td>NR</td>
</tr>
<tr>
<td>6 Dynamic reconfiguration</td>
<td>C.3.13</td>
<td>---</td>
<td>NR</td>
<td>NR</td>
<td>NR</td>
</tr>
<tr>
<td>7a Structured methods including for example, ISD, MASCOT, SADT, and Yourdon</td>
<td>C.2.1</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>7b Semi-formal methods</td>
<td>Table B.7</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>7c Formal methods including for example, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z</td>
<td>C.2.4</td>
<td>---</td>
<td>R</td>
<td>R</td>
<td>HR</td>
</tr>
<tr>
<td>8 Computer-aided specification tools</td>
<td>B.2.4</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
</tbody>
</table>

a) Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. Only one of the alternate or equivalent techniques/measures has to be satisfied.

b) The measures in this table concerning fault tolerance (control of failures) should be considered with the requirements for architecture and control of failures for the hardware of the programmable electronics in part 2 of this standard.
Idea – Problems – State of the art

Applying formal verification to PLC programs (new developments and existing systems independently of the purpose)

- But…
  - Why formal verification is not widely used in industry yet?
  - How can we fill the gap between the automation and formal verification worlds?

- Other industries using formal methods:
  - **NASA**: Remote Agent spacecraft control system (Deep Space 1 mission).
  - **Aircraft** industry: Airbus A340 flight control, etc.
  - **Communication** protocols: IPv6 protocol.
  - Etc.

- What about formal verification in PLC based control systems?
  - Industry: ESTEREL (SCADE), Siemens and ABB doing research.
  - Academia: RWTH Aachen University, TU Dortmund, ENS Cachan, …
Why model checking is not widely used in automation?

- How to get models?
  - Automated generation
- Which model checker should be used?
  - Multiple (general method.)
- How to formalize requirements?
  - Formal requirement
- How to proceed with a counterexample?
  - Analysis & Demonstration
- How to make it efficient?
  - Reductions
  - Pattern

Formal Model

Real System (hardware, software)

Specifications

Counter-example

satisfied

Model checker

not satisfied
Model checking vs. Testing

- MC checks the specifications against a **model** instead of the real system.

- Allows to **check properties** that are almost **impossible** to test (e.g. liveness properties).

- Checks all **possible combinations**.

- Gives a **counterexample** when a discrepancy is found.

- Possible to **automatize** (can be used by non-formal method experts).

- State space **explosion**.
Our approach: methodology overview

- **General** method for applying formal verification:
  - Generate formal models **automatically** out of PLC code.
  - Includes **several input PLC languages** (IEC 61131-3: SFC, ST, IL, Ladder, FBD).
  - Easy integration of **different formal verification** tools.
Model example

1. IF ia > 0 THEN
   xa := TRUE;
ELSE
   xa := FALSE;
   IF ib > 0 THEN
   xb := TRUE;
   ELSE
   xb := FALSE;
   END_IF;
END_IF;
c := c + 1;

2. 

3. init(loc) := initial;
   next(loc) := case
       loc = end : initial;
       loc = initial : l1;
       loc = 11 & ((IA > 0sd16_0)) : l3;
       loc = 11 & !((IA > 0sd16_0)) : l2;
       loc = 12 & ((IB > 0sd16_0)) : l3;
       loc = 12 & !((IB > 0sd16_0)) : l2;
       loc = 13 : end;
   TRUE: loc;
esac;
next(XA) := case
       loc = 11 & ((IA > 0sd16_0)) : TRUE;
       loc = 11 & !((IA > 0sd16_0)) : FALSE;
   TRUE : XA;
esac;
...
Our approach: methodology overview

Traditional PLC program development

How can we be sure that a bug found by this methodology is real?

We can use the counterexample in the real system and prove it.
Project status: CASE tool prototype
Project status & Results

The methodology has been applied to PLC programs at CERN:
- UNICOS library: Bugs have been found in previously tested PLC programs.
- Full UNICOS applications: Cryogenic control system (QSDN).

Future development & research:
- Extending the PLC grammars.
- Production-ready CASE tool.
- More abstraction and reduction techniques.
- Control system specifications.
- Applying extensively this approach to CERN PLC programs.
- Static analysis.
Conclusion and summary

- Model checking can be applied to PLC programs.

- **Contribution → Difficulty can be hidden:**
  - Automated model generation, requirement patterns, automatic reduction techniques and counterexample analysis.

- We have **found discrepancies in our systems:**
  - Incomplete or incorrect **specification**.
  - Mistake in the **implementation**.

- Bugs can be proved in the real systems (counterexample info).

- Models are built out of the **PLC programs.** No standards for PLC program specification.