Bringing Deductive Verification to Factory Automation developers

Denis Cousineau – Researcher at MERCE

Frama-C & SPARK day

THE OPEN SOURCE INNOVATION SPRING 2019
1. Introduction

2. How we can *easily* improve Ladder debugging with Why3

3. From the IDE... back to the IDE

4. Conclusion
WHO WE ARE?

Research centre located in Rennes

Expertise in advanced communications and energy sciences like power electronics

Open Innovation with European academic laboratories, consortiums and standardisation bodies

60 people in total, of which 35 researchers, mostly PhDs
Mitsubishi Electric Corporate R&D

Mitsubishi Electric R&D Centre Europe (MERCE)
Communication & Information Technology
Power Electronic Technology
Environment & Energy Technology
(Rennes, France and Livingston, UK)

Mitsubishi Electric Research Laboratories (MERL)
Mechatronics
Information & Communication
(Massachusetts, USA)

Mitsubishi Electric (China) (R&D Dept.)

R&D in Japan (2 locations)

Advanced Technology R&D Center
- Power Electronics Technology
- Electrical Technology
- Environmental, Energy and Materials Technology
- Device Technology
- System Technology
- Image Technology (Hyogo Prefecture)

Information Technology R&D Center
- Information Technology
- Communications Technology
- Multi-Media Technology
- Optical and Electrical Wave Technology

Industrial Design Center
- Design Technology
  (Kanagawa Prefecture)
Mitsubishi Electric Corporation

... is **not** Mitsubishi Motors
Mitsubishi Electric Corporation

Source: MELCO website. 2017 results.
Industrial automation is a key market for MELCO $10B annual sales, 2\textsuperscript{nd} vendor worldwide\(^*\)

MELCO sells AC servo systems, inverters, industrial robots, processing machines and Programmable Logic Controllers (PLC) to control them

Ladder Logics

Ladder program = graphical diagram with circuits diagrams of relay logic hardware

Inputs

Logical combination of inputs values

Outputs

Instructions calls

~80% of PLC programs

Hard to debug manually
State of the art of Ladder debugging

1. Industry:
   – PLC manufacturers provide IDEs with
     • Simulation mode (possibly with breakpoints)
     • Testing tools

   ... But “real debugging” is still often done live in the factory

2. Research
   – Numerous papers on verifying complex properties of Ladder programs
     • Coq: Safety properties verification of ladder diagram programs, Roussel & al., 2009
     • Model-checking: PLC verification using symbolic model checking, Bhoi & al., 2008

   ... But those solutions do not scale, hence are not implemented yet in IDEs
Plan

1. Introduction

2. How we can easily improve Ladder debugging with Why3

3. From the IDE... back to the IDE

4. Conclusion
How we can *easily* improve Ladder debugging

• “improve Ladder debugging”
  – Targets useful errors detection

• “easily”
  – Easy to develop for us
    • Use off-the-shelf components like Why3, CVC4, etc...
  – Easy to use for Ladder programmers
    • Fully automatic tool
    • Easy-to-understand feedback
    • Scalable to industrial projects size

Target is runtime errors detection
  (division by 0, integers overflows, instructions errors, etc...)
• the **BCD** instruction
  – Converts 16-bit binary data to BCD 4-digit data for display purpose

  ![Ladder example](image)

  – Possible range error when calling BCD instruction \(\rightarrow \text{[0;9999]}\)

<table>
<thead>
<tr>
<th>Error code (3D6)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Data in the device specified by (6) is out of the range, 0 to 9999.</td>
</tr>
<tr>
<td>9999</td>
<td>[description]</td>
</tr>
</tbody>
</table>
Ladder example

• A simple program using **BCD** instruction

```plaintext
while true do
    if X0 then {
        D0 = D0 + 1;
        D1 = BCD(D0);
    }

done
```

Program is executed continuously every XXms
1st order modelization of Ladder **single-scan**

• Modelizing a single scan allows to
  – use *only* first order logic to specify
    • instructions calls
    • arithmetic operations, etc...
  – Detect *automatically*
    • instructions errors
    • arithmetic overflows, etc...

→ Considering only the loop body allows to bypass user interaction that would be needed otherwise (e.g. stating loop invariants)
1. Introduction

2. How we can easily improve Ladder debugging with Why3

3. From the IDE... back to the IDE

4. Conclusion
From the IDE…

Research Prototype

GX Works 3 → AST → OCaml

```ocaml
type front =
| Contact of ckind * string
| Sequential_front of front * front
| Parallel_front of front * front
| Arith_test of arith_op * string * string

type output =
| Coil of bool * string
| Timer of string * int
| Pulse of string
| Set of bool * string
| Reset of bool * string
| Bcd of bool * string * string
| Mov of bool * string * string
| BitShift of bool * string
| Inc of bool * string
| FMov of bool * string * string * int
| BMov of bool * string * string * int
| BKMinus of bool * string * string * string * int
| (...)

type rear =
| Output of output * int
| Dependent_rear of front * rear
| Parallel_rear of rear * rear

type rung = {front : front; rear : rear}

type diagram = rung list
```
From the IDE...

Translated program with code locations informations as labels

Errors conditions as pre-conditions
Errors explanations as labels

Research Prototype

GX Works 3

let n253_2 = 'expl:42 common_181 in
let common_181 = pulse and
(pulse and (n250_1) (pulse and
(n251_2) (pulse and (n252_2) (n253 2)))) in

let bcd (is_P : bool)
(input : pulse)
(src : int)
(prev_val : int) : int
requires { "model.vc"
"expl:BCD: out of [0...9999] range call"
(inactive is P input \ (0 <= src / src <= 9999))
... back to the IDE

Counter Example

Research Prototype

or

Mitsubishi Electric R&D Centre Europe

Non Confidential / Export Control: NLR
... back to the IDE
... back to the IDE

Research Prototype

or

Counter Example

AST initial config

AST full execution

```haskell
type pulse = Pulsing | Downing | Still.False | Still.True

type front_v =
  | Contact_v of pulse * Ladder.ast.ckind * bit.dev * pulse
  | Sequential_front_v of pulse * front_v * front_v * pulse
  | Parallel_front_v of pulse * front_v * front_v * pulse
  | Arith_test_v of pulse * Ladder.ast.arith_op * word.dev * word.dev * bool
```
... back to the IDE

Research Prototype

Counter Example

AST initial config

AST full execution

GUI
... back to the IDE

Error reason
(labels in Ladder library)

Error scenario
(symbolic execution from error initial configuration)

Error location
(labels in program model)
Plan

1. Introduction

2. How we can easily improve Ladder debugging with Why3

3. From the IDE... back to the IDE

4. Conclusion
Conclusion

• **Objective 1**: improve Ladder programming debugging
  – We targeted runtime errors detection

• **Objective 2**: easy to use for Ladder programmers
  – Fully automatic tool
  – Give all useful information when finding an error (thanks to Why3 labels)

• **Objective 3**: scalable to industrial projects size
  – Our prototype (~10k LOC OCaml) handles them in a few seconds
Feedback

• Very positive feedback using Why3

  – API for file loading, WP, strategies and running provers
    (did not use it for creating Why3 modules,
     we used our own library to produce Why3 text files)

  – Labels for contextualizing proof obligations
    (code locations, errors reasons, etc...)

  – Counter-examples handling with CVC4
    (improvement track: counter-examples relevance)

... many thanks to the Why3 team!
Thank you for your attention