Bringing Deductive Verification to Factory Automation developers

Denis Cousineau – Researcher at MERCE

Frama-C & SPARK day

THE OPEN SOURCE INNOVATION SPRING 2019

MFR19-ARC-239

Plan

1. Introduction

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

3. From the IDE… back to the IDE

4. Conclusion

Mitsubishi Electric R&D Centre Europe - France

WHO WE ARE ?

Research centre
located
in Rennes

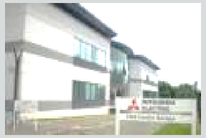
Expertise
in advanced communications
and energy sciences
like power electronics

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

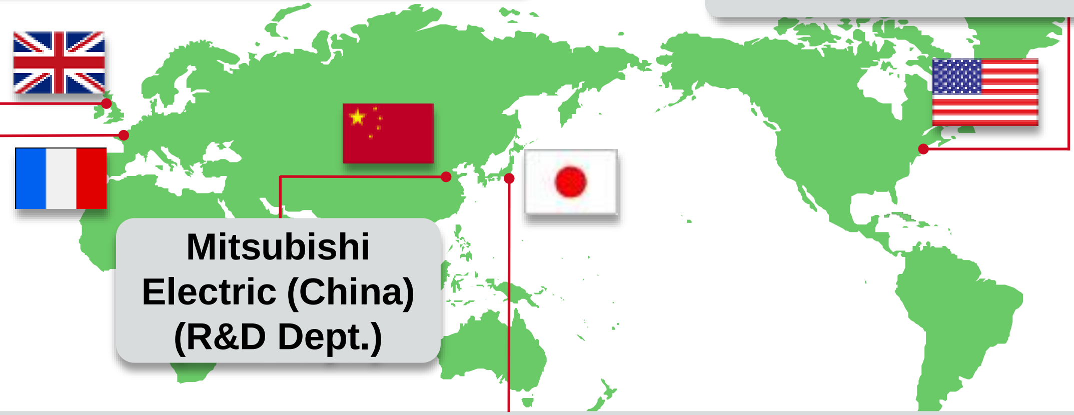
Mitsubishi Electric R&D Centre Europe
(MERCE)



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

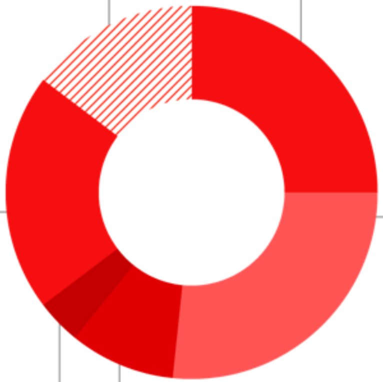
- Design Technology

(Kanagawa Prefecture)

Mitsubishi Electric Corporation

… is not Mitsubishi Motors

Mitsubishi Electric Corporation

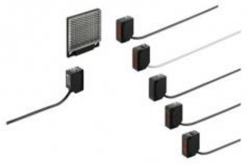


Source: MELCO website. 2017 results.

Factory Automation and PLC

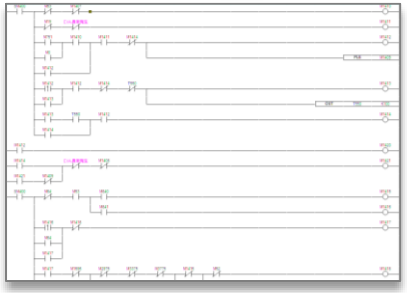
Industrial automation is a key market for MELCO
$10B annual sales, 2nd vendor worldwide*

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



Inputs
Sensors

Outputs
Actuators



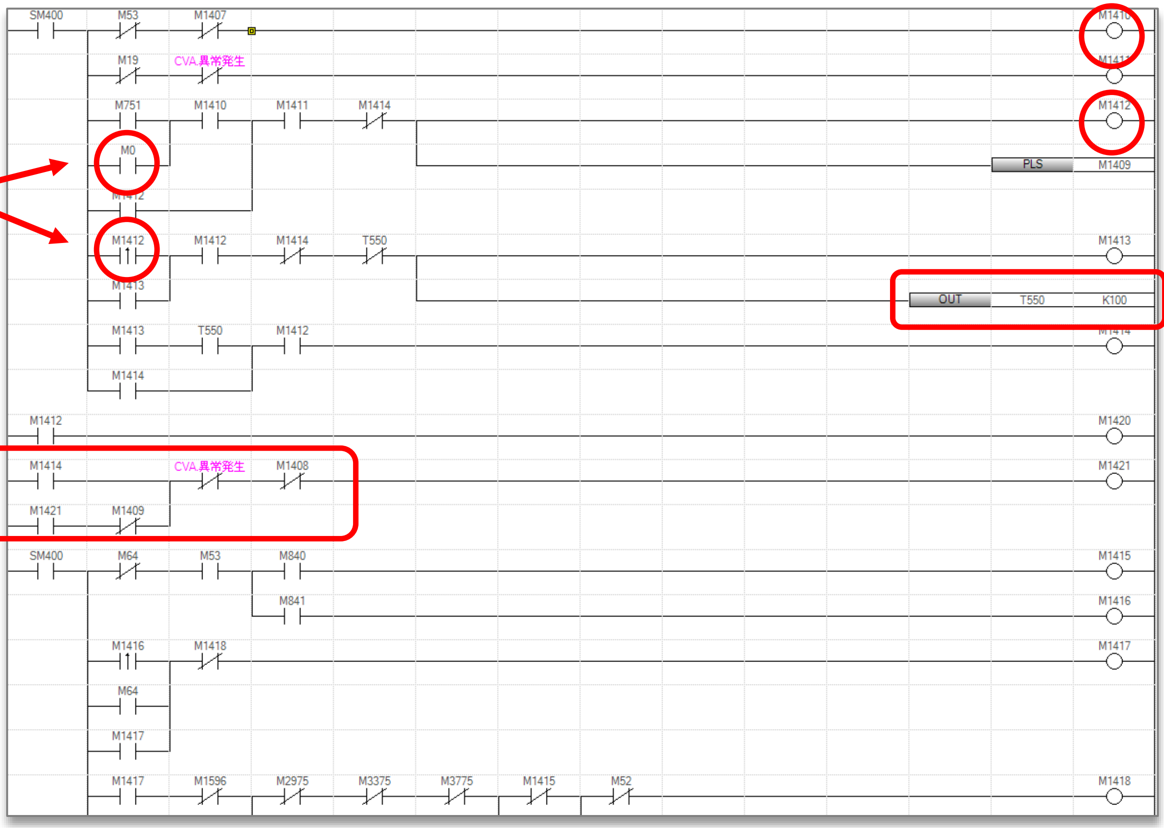
Software

(*) <u>Source</u>:
Industrial Automation Market Research Report- Forecast 2023.
ID: MRFR/SEM/1643-CRR | October, 2018

Ladder Logics

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

Inputs



Outputs

Instructions calls

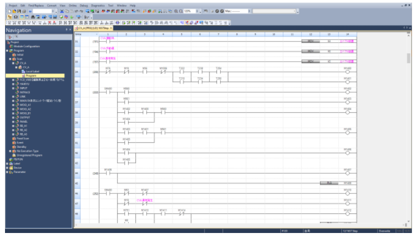
Logical combination of inputs values

~80% of PLC programs

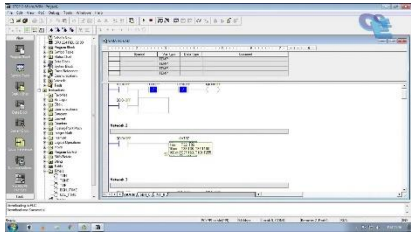
Hard to debug manually

State of the art of Ladder debugging

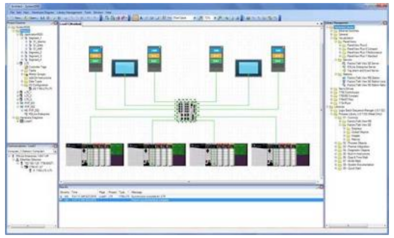
1. Industry:
   – PLC manufacturers provide IDEs with
     - Simulation mode (possibly with breakpoints)
     - Testing tools



MELCO GX Works 3



Siemens Step7



Rockwell Automation Studio 5000

... 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…)

Ladder example

- the **BCD** instruction
  - Converts 16-bit binary data to BCD 4-digit data for display purpose

16 bits integer		16 bits binary coded decimal
0000011111011100	→	0010 0000 0001 0010
		2 0 1 2

  - Possible range error when calling BCD instruction → [0;9999]

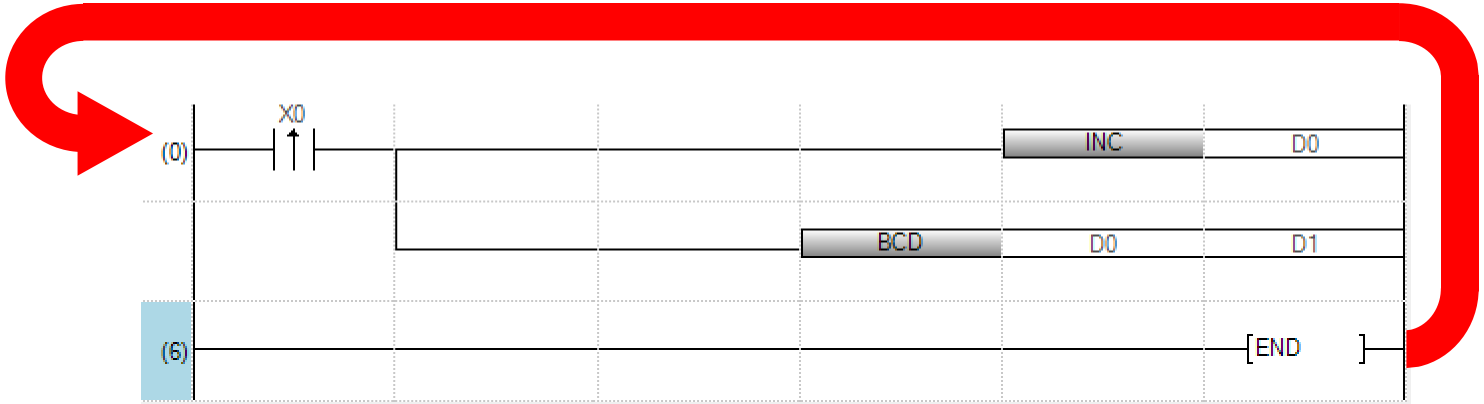
Operation error

Error code (SD0)	Description
3401H	Data in the device specified by (s) is out of the range, 0 to 9999.

0000	000000000000000	0000 0000 0000 0000
9999	010011100001111	1111 1111 1111 1111

Ladder example

- A simple program using **BCD** instruction



Program is executed continuously every XXms

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

1st order modelization of Ladder <u>single-scan</u>

- 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…

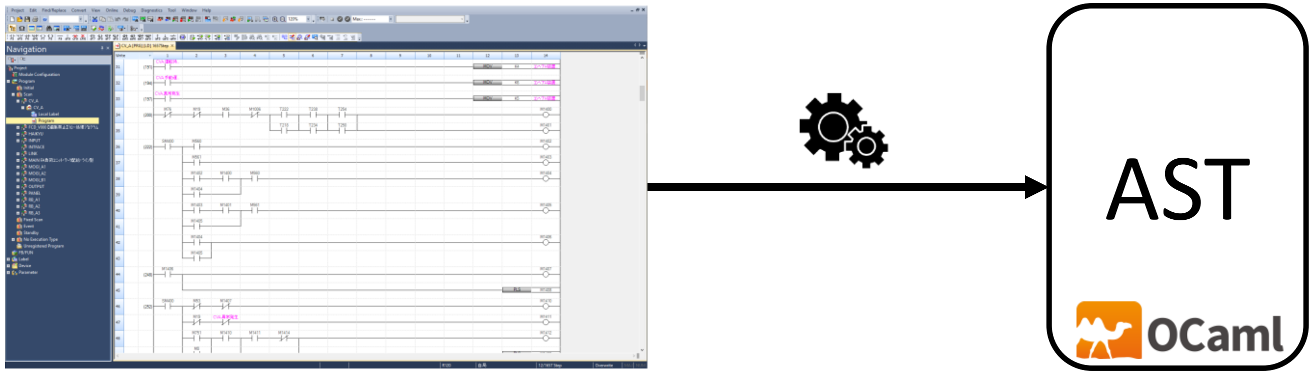
→ Considering only the loop body allows to bypass user interaction that would be needed otherwise (e.g. stating loop invariants)

Plan

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

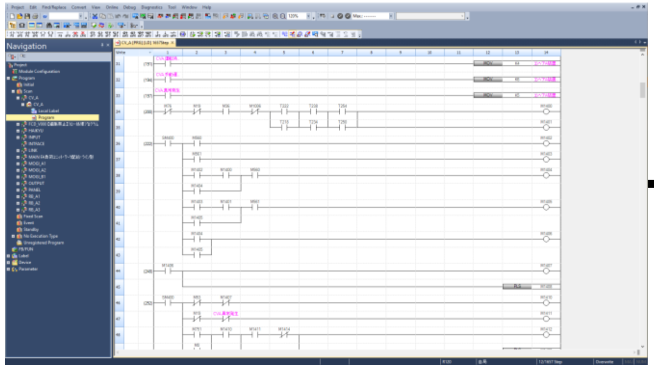
```ocaml
type front =
  | Contact of ckind * string
  | Sequential_front of front * front
  | Parallel_front of front * front
  | Arith_test of arith_op * string * string

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…

Research Prototype

```
let m253_2 = "expl:42" common_101 in
let common_102 = pulse_and
                 (merce1_1)
                 (pulse_and
                      (m250_2)
                      (pulse_and (m251_2) (pulse_and (m252_2) (m253_2)))) in
```

Translated program with code locations informations as labels



GX Works 3

AST

OCaml

Model

Why3

WP

```
use export ladder_instructions.Ladder
```

Library

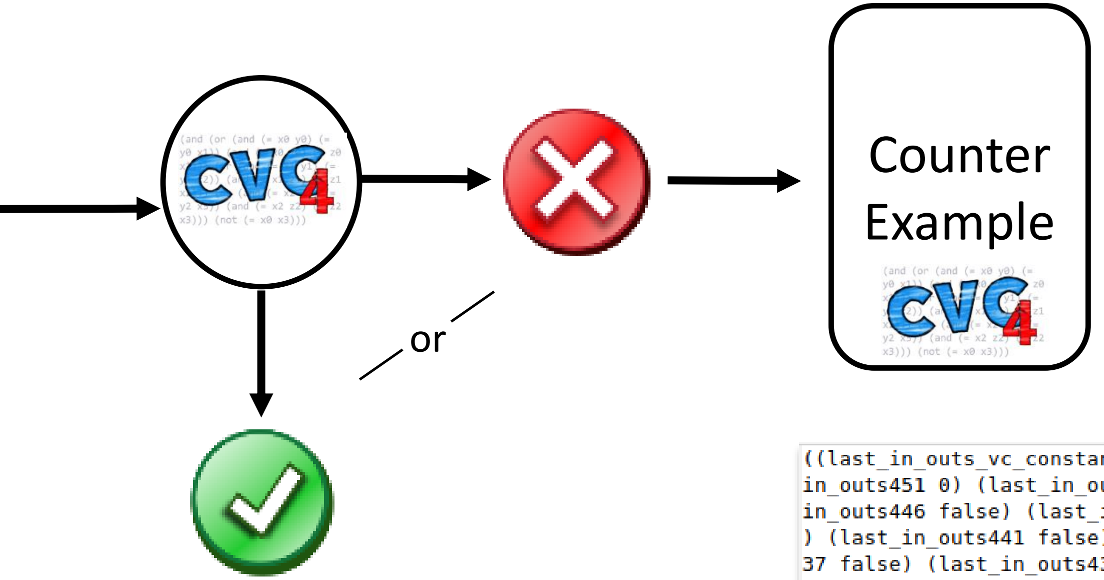
Why3

```
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))
         }
```

Errors conditions as pre-conditions
Errors explanations as labels

… back to the IDE

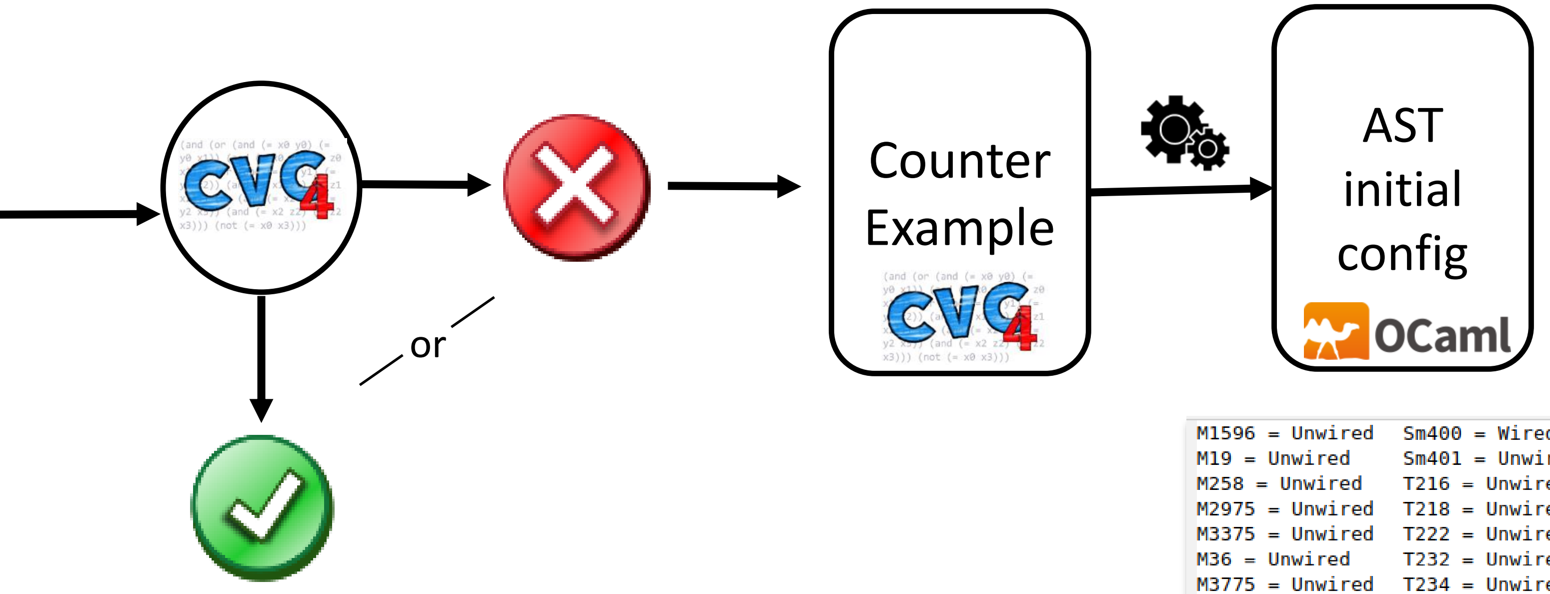
Research Prototype



((last_in_outs_vc_constant1 true) (last_in_outs_vc_constant true) (last_in_outs453 0) (last_in_outs452 0) (last_
in_outs451 0) (last_in_outs450 0) (last_in_outs449 false) (last_in_outs448 false) (last_in_outs447 false) (last_
in_outs446 false) (last_in_outs445 false) (last_in_outs444 false) (last_in_outs443 false) (last_in_outs442 false
) (last_in_outs441 false) (last_in_outs440 false) (last_in_outs439 false) (last_in_outs438 false) (last_in_outs4
37 false) (last_in_outs436 false) (last_in_outs435 false) (last_in_outs434 false) (last_in_outs433 false) (last_
in_outs432 false) (last_in_outs431 false) (last_in_outs430 false) (last_in_outs429 false) (last_in_outs428 false
) (last_in_outs427 false) (last_in_outs426 false) (last_in_outs425 true) (last_in_outs424 true) (last_in_outs423
 0) (last_in_outs422 0) (last_in_outs421 0) (last_in_outs420 0) (last_in_outs419 0) (last_in_outs418 0) (last_in
_outs417 0) (last_in_outs416 0) (last_in_outs415 0) (last_in_outs414 0) (last_in_outs413 0) (last_in_outs412 0)
(last_in_outs411 0) (last_in_outs410 0) (last_in_outs409 0) (last_in_outs408 0) (last_in_outs407 0) (last_in_out
s406 0) (last_in_outs405 0) (last_in_outs404 0) (last_in_outs403 0) (last_in_outs402 0) (last_in_outs401 0) (las
t_in_outs400 0) (last_in_outs399 0) (last_in_outs398 0) (last_in_outs397 0) (last_in_outs396 0) (last_in_outs395
 0) (last_in_outs394 0) (last_in_outs393 0) (last_in_outs392 0) (last_in_outs391 0) (last_in_outs390 0) (last_in
_outs389 0) (last_in_outs388 0) (last_in_outs387 0) (last_in_outs386 0) (last_in_outs385 0) (last_in_outs384 0)
(last_in_outs383 0) (last_in_outs382 0) (last_in_outs381 0) (last_in_outs380 0) (last_in_outs379 0) (last_in_out

… back to the IDE

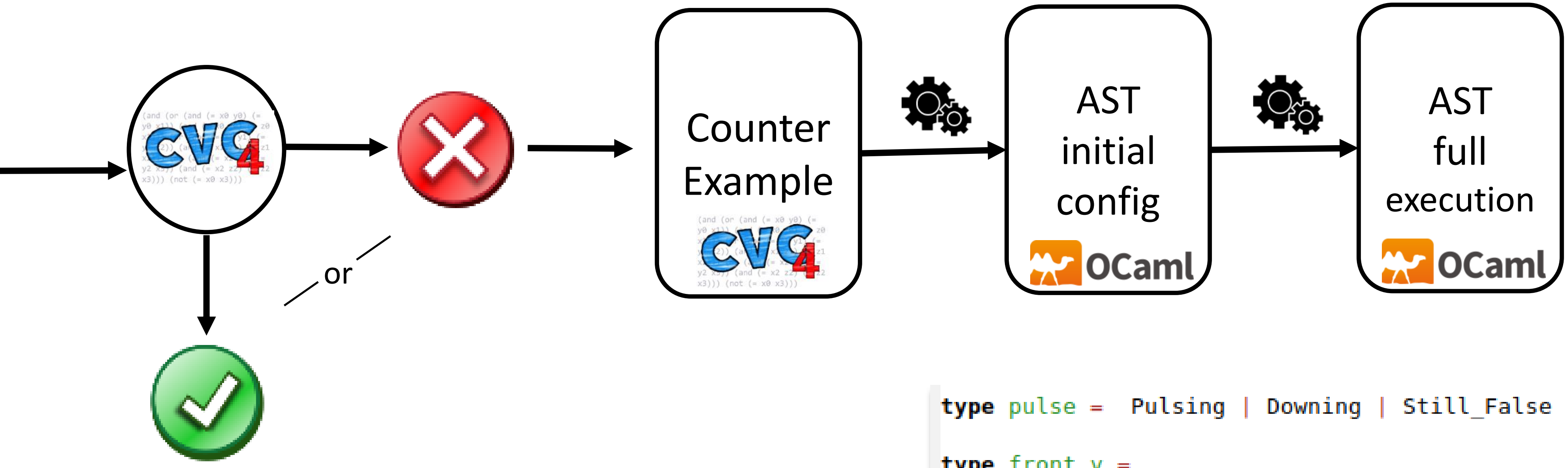
Research Prototype



M1596 = Unwired	Sm400 = Wired
M19 = Unwired	Sm401 = Unwired
M258 = Unwired	T216 = Unwired
M2975 = Unwired	T218 = Unwired
M3375 = Unwired	T222 = Unwired
M36 = Unwired	T232 = Unwired
M3775 = Unwired	T234 = Unwired
M402 = Unwired	T238 = Unwired
M52 = Unwired	T248 = Unwired
M53 = Unwired	T250 = Unwired
M560 = Pulsing	T254 = Unwired
M561 = Unwired	T264 = Unwired
M64 = Unwired	T266 = Unwired
M65 = Unwired	T550 = Unwired
M751 = Downing	T582 = Unwired
M76 = Unwired	D1400 = 0
M840 = Unwired	D1410 = 10000
M841 = Unwired	D1411 = 0

… back to the IDE

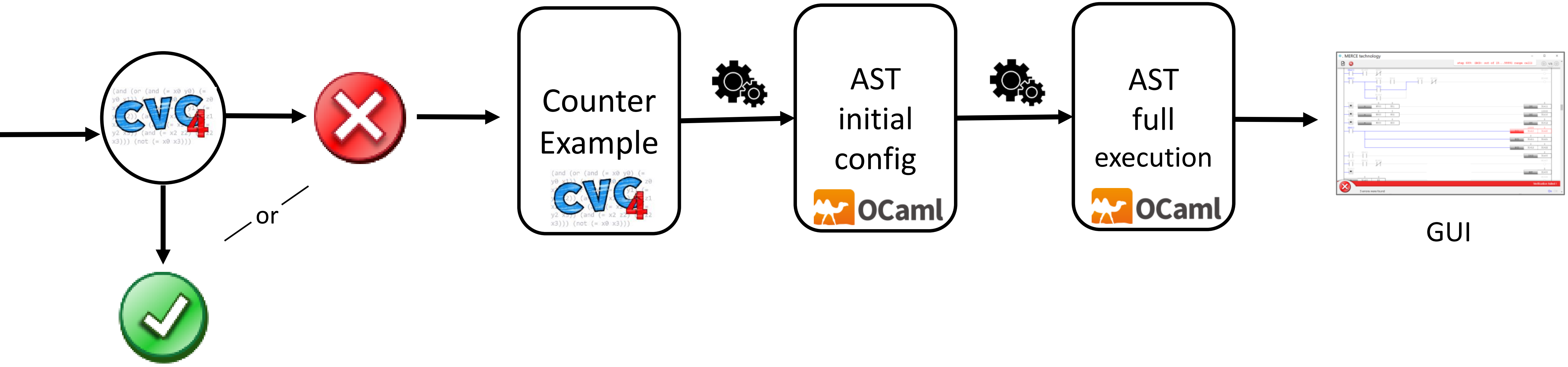
Research Prototype




```
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

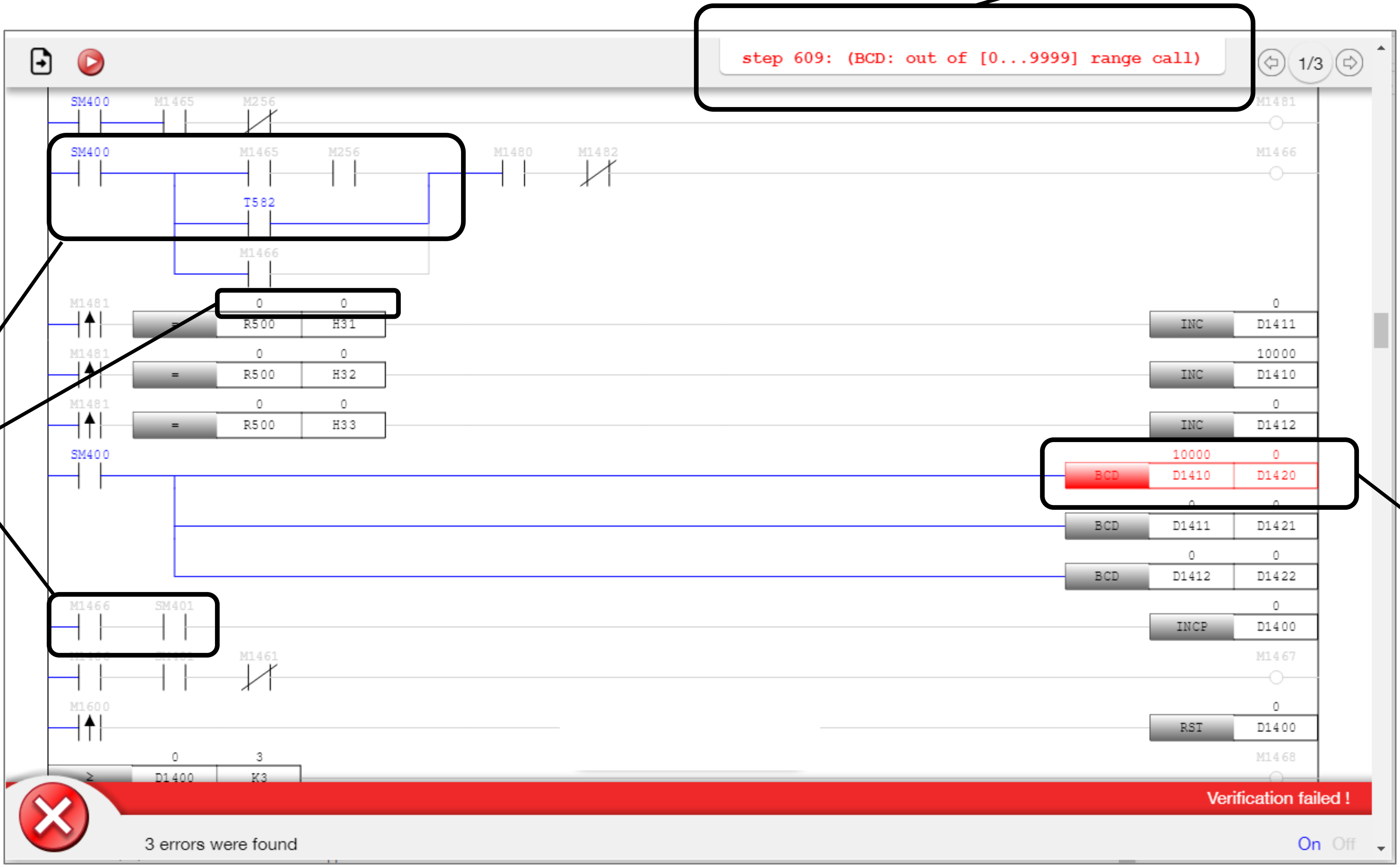


MITSUBISHI ELECTRIC
Changes for the Better

… back to the IDE

Research Prototype

Error reason
(labels in Ladder library)



Error scenario
(symbolic execution from error initial configuration)

Error location
(labels in progam model)

MITSUBISHI ELECTRIC
Changes for the Better

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 !

MITSUBISHI ELECTRIC

Changes for the Better

Thank you for your attention