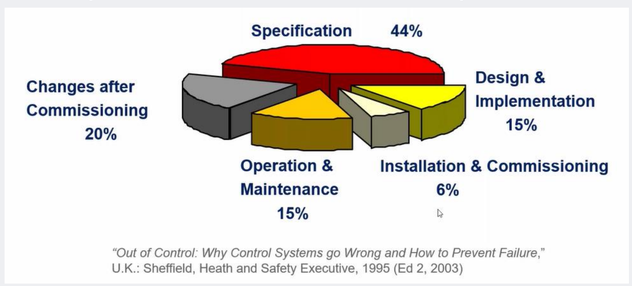
PLCverif: a tool to formally verify critical PLC programs

B. Fernández Adiego*, I. D. Lopez Miguel, M. Bres, J-C. Tournier and E. Blanco Viñuela
*borja.fernandez.adiego@cern.ch

ACCELERATOR RELIABILITY WORKSHOP ARW 2022

Software defects can be fatal and extremely costly

- Errors in the design, specification and software implementation are very common



- E.g. Ariane 5 accident (more than 500 million US$)

- In the **particle accelerators domain**, a software failure may have very serious consequences:
  - **Safety of the personnel** (risks related to radiation, electrical systems, cryogenics, etc.)
  - **High cost** (**damage** of machines and industrial facilities **or unavailability** of particle accelerator)

In industry, **the most common software verification techniques** are **peer reviews and testing**.
But they have **some limitations** (e.g. how to catch corner cases? all combinations of the program cannot be tested).

How to guarantee that critical software is compliant with the functional specifications? Formal methods

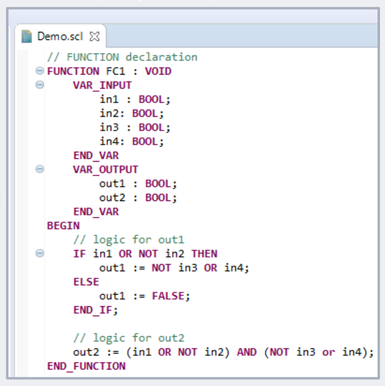
- **Formal methods** are techniques based on **mathematics and formal logic** (e.g. petri nets, automata, temporal logic, B-method, etc.)
- They can be used for **specification** and modelling, simulation, **formal verification** (e.g. **model checking**), etc.
- They provide **more effective** verification techniques (**more combinations** are explored)
- They are **popular** techniques **in critical industries** (e.g. aerospace, aircraft and railway industries)

PLCverif: Model checking for PLC programs

What is model checking? given a global **model of the system** and a **formal property**, the model checking algorithm **explores exhaustively** that the model meets the property

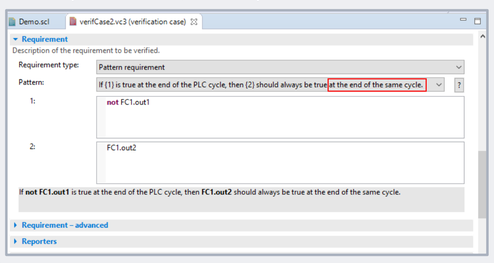
PLCverif user interface (inputs)

1 PLC program



2 Property to verify (specification patterns)

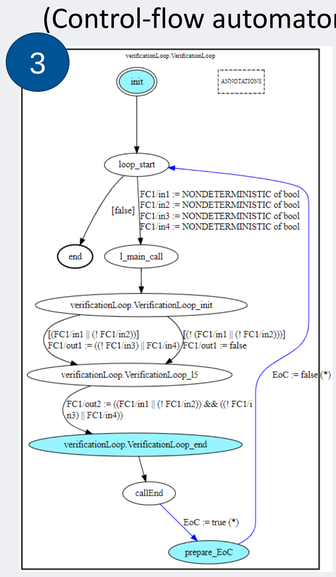
"If Output1 is FALSE, then Output2 should be TRUE at …"



PLCverif internals

Intermediate model generation (Control-flow automaton)

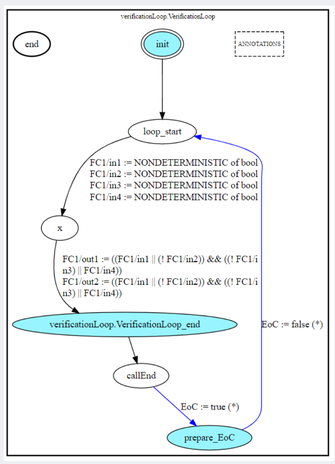
3



4 Formalized property (Temporal logic)

AG ((EoC & !FC1.Out1) → FC1.Out2)

5 Model reduction and abstraction techniques



6 Specific model generation (nuXmv model)

```
MODULE main
    VAR
        loc : {init_pv, end, loop_start, callEnd, prepare_EoC, verificationLoop_VerificationLoop_end, x};
        FC1_in1 : boolean;
        FC1_in2 : boolean;
        FC1_in3 : boolean;
        FC1_in4 : boolean;
        FC1_out1 : boolean; -- frozen
        FC1_out2 : boolean; -- frozen
        EoC : boolean; -- frozen

    ASSIGN
        -- CFA structure (loc)
        init(loc) := init_pv;
        next(loc) := case
            loc = init_pv & (TRUE) : loop_start;
            loc = loop_start & (TRUE) : x;
            loc = callEnd & (TRUE) : prepare_EoC;
            loc = prepare_EoC & (TRUE) : loop_start;
            loc = verificationLoop_VerificationLoop_end & (TRUE) : callEnd;
            loc = x & (TRUE) : verificationLoop_VerificationLoop_end;
            TRUE: loc;
        esac;

        init(FC1_in1) := FALSE;
        next(FC1_in1) := case
            loc = loop_start & (TRUE) : {TRUE, FALSE};
            TRUE  : FC1_in1;
        esac;
        init(FC1_in2) := FALSE;
        next(FC1_in2) := case
            loc = loop_start & (TRUE) : {TRUE, FALSE};
            TRUE  : FC1_in2;
        esac;
        init(FC1_in3) := FALSE;
        next(FC1_in3) := case
            loc = loop_start & (TRUE) : {TRUE, FALSE};
            TRUE  : FC1_in3;
        esac;
        init(FC1_in4) := FALSE;
        next(FC1_in4) := case
            loc = loop_start & (TRUE) : {TRUE, FALSE};
            TRUE  : FC1_in4;
        esac;
        init(FC1_out1) := FALSE;
        next(FC1_out1) := case
            loc = x & (TRUE) : ((FC1_in1) | (!(FC1_in2))) & (((FC1_in3)) | (FC1_in4));
            TRUE  : FC1_out1;
        esac;
        init(FC1_out2) := FALSE;
        next(FC1_out2) := case
            loc = x & (TRUE) : ((FC1_in1) | (!(FC1_in2))) & (((FC1_in3)) | (FC1_in4));
            TRUE  : FC1_out2;
        esac;
        init(EoC) := FALSE;
        next(EoC) := case
            loc = callEnd & (TRUE) : TRUE;
            loc = prepare_EoC & (TRUE) : FALSE;
            TRUE  : EoC;
        esac;

-- Requirement
CTLSPEC AG((EoC) & (!(FC1_out1))) -> (FC1_out2));
```

```
nuXmv > read_model -i verificase2.smv
nuXmv > flatten_hierarchy
nuXmv > encode_variables
nuXmv > build_model
nuXmv > print_reachable_states
############################################################
system diameter: 8
reachable states: 98 (2^6.61471) out of 896 (2^9.80735)
############################################################
```

7 Model checking algorithms

Model checker

CBMC, theta, nuXmv

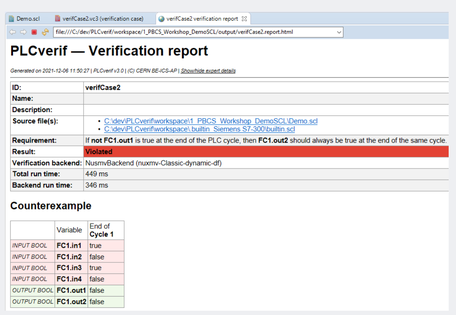
PLCverif user interface (results)

Model checking report

8

✓ Property OK

✘ Property failed
Trace leading to the violation (counterexample)



PLCverif is an **open source** project https://gitlab.com/plcverif-oss (more details in www.cern.ch/plcverif)

PLCverif relevant projects

- **CERN SPS-PPS project:** *B. Fernandez et al. "Applying model checking to highly-configurable safety critical software: The SPS-PPS PLC program" in Proc. of the 18th ICALEPCS https://accelconf.web.cern.ch/icalepcs2021/papers/wepv042.pdf*
- **CERN SM18 project:** *B. Fernandez et al. "Cause-and-Effect Matrix specifications for safety critical systems at CERN" in Proc. of the 17th ICALEPCS https://accelconf.web.cern.ch/icalepcs2019/papers/mopha041.pdf*
- **ITER:** *B. Fernandez et al. "Applying model checking to critical PLC applications: An ITER case study" in Proc. of the 17th ICALEPCS https://accelconf.web.cern.ch/icalepcs2017/papers/thpha161.pdf*
- **GSI:** collaboration to formally verify safety critical PLC programs of the FAIR particle accelerator

CERN Beams Department
Industrial Controls Systems Group (ICS)