CERN

B. Fernández, D. Darvas, E. Blanco

Formal methods appliedto PLC code verification

Automation seminar CERN – IFAC (CEA)

02/06/2014

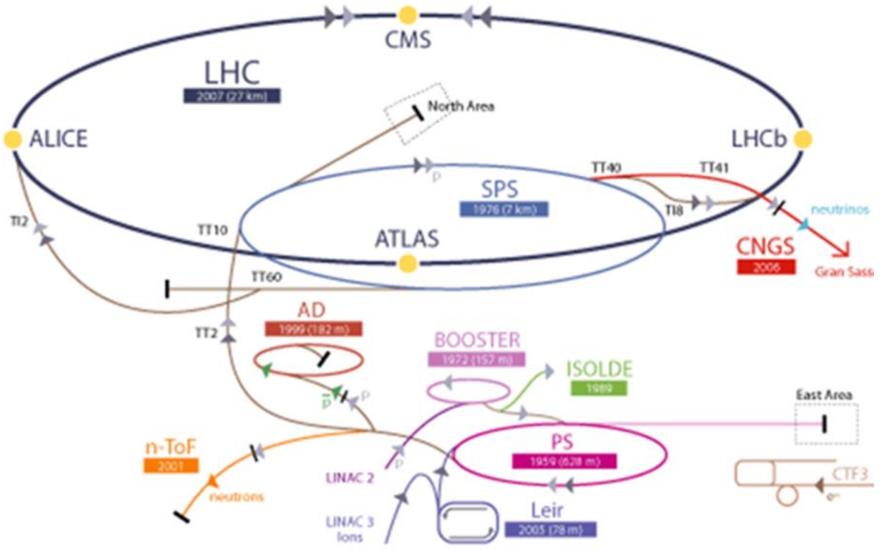
Outline

- Context
- Formal verification
- Methodology overview
- Some results
- Conclusions

Context

- ❑ CERN: European Organization for Nuclear Research

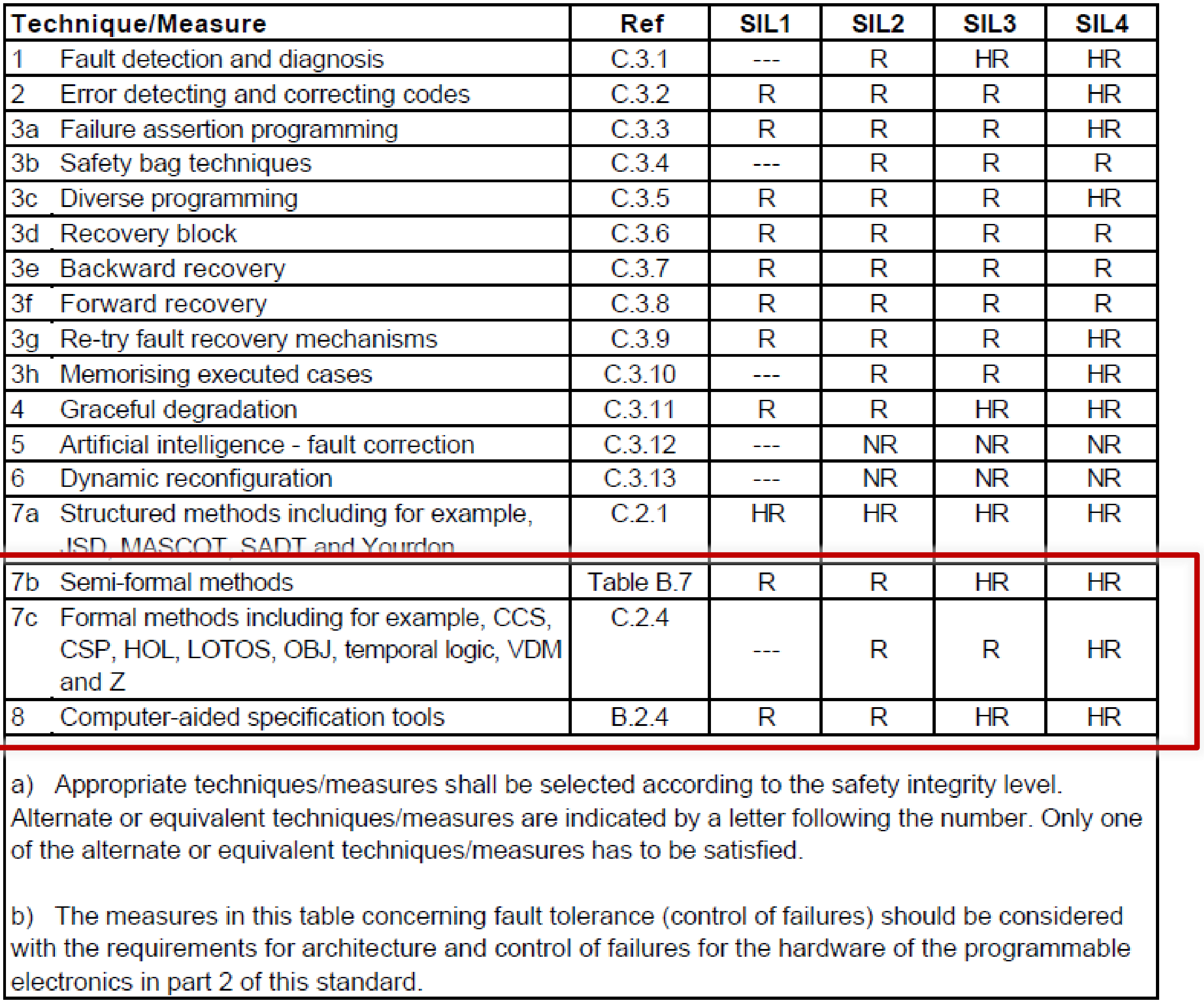
  The biggest particle physics research institute



- ❑ PLCs at CERN:
  - ❑ Widely used in many different systems.
  - ❑ E.g.: cryogenics, vacuum systems, gas systems, C&V.
  - ❑ Common structure (UNICOS framework).

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

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



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

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.

Verification of CERN’s PLC programs

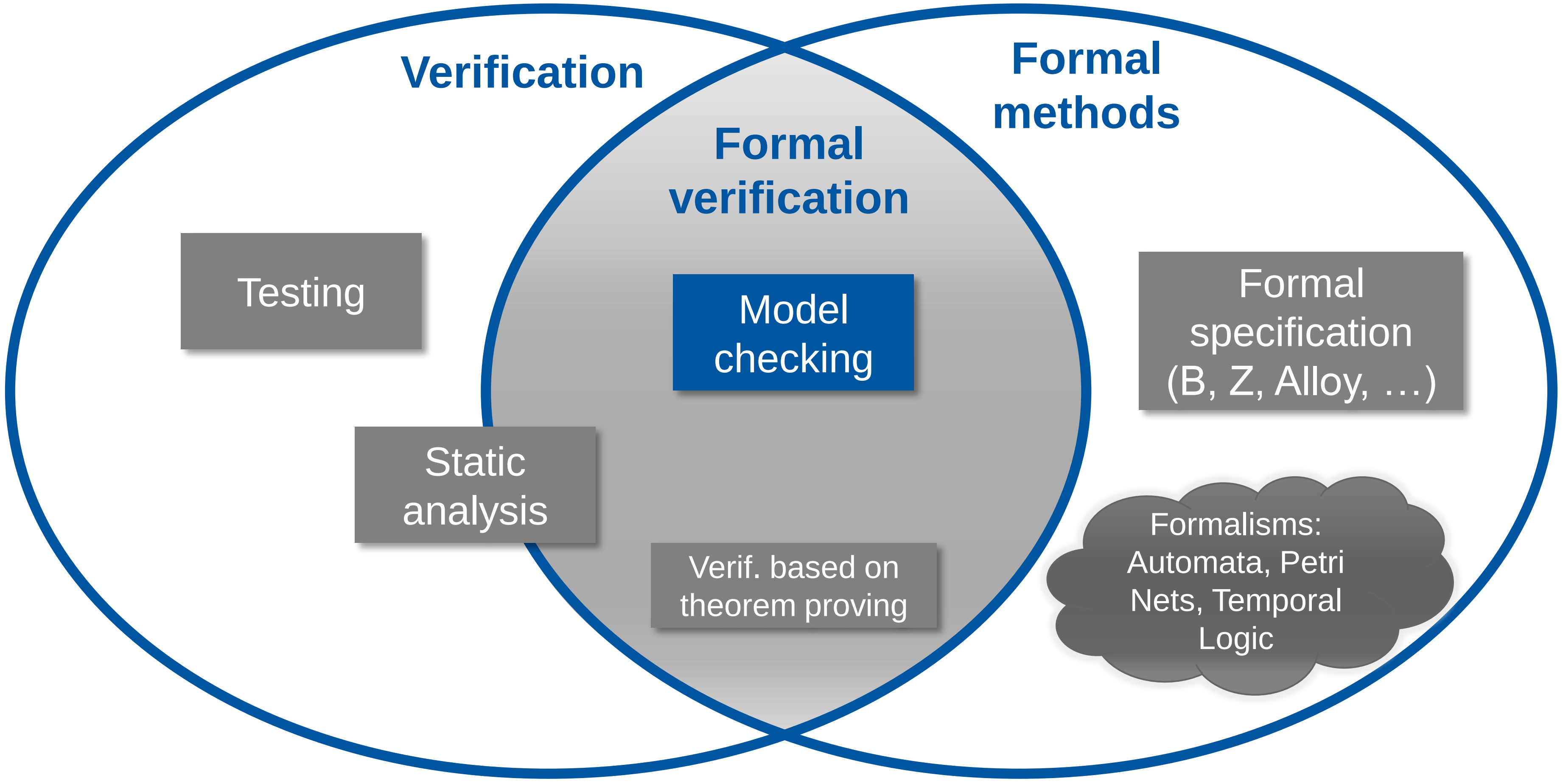
- Currently: manual and automated **testing**
  - Useful, but not efficient for every type of requirements
  - Difficult to test **safety** requirements:
    “if out1 is true, out2 should be false”

- **Model checking** can complement testing
  - Can check large amount of combinations.
  - Formal method.

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

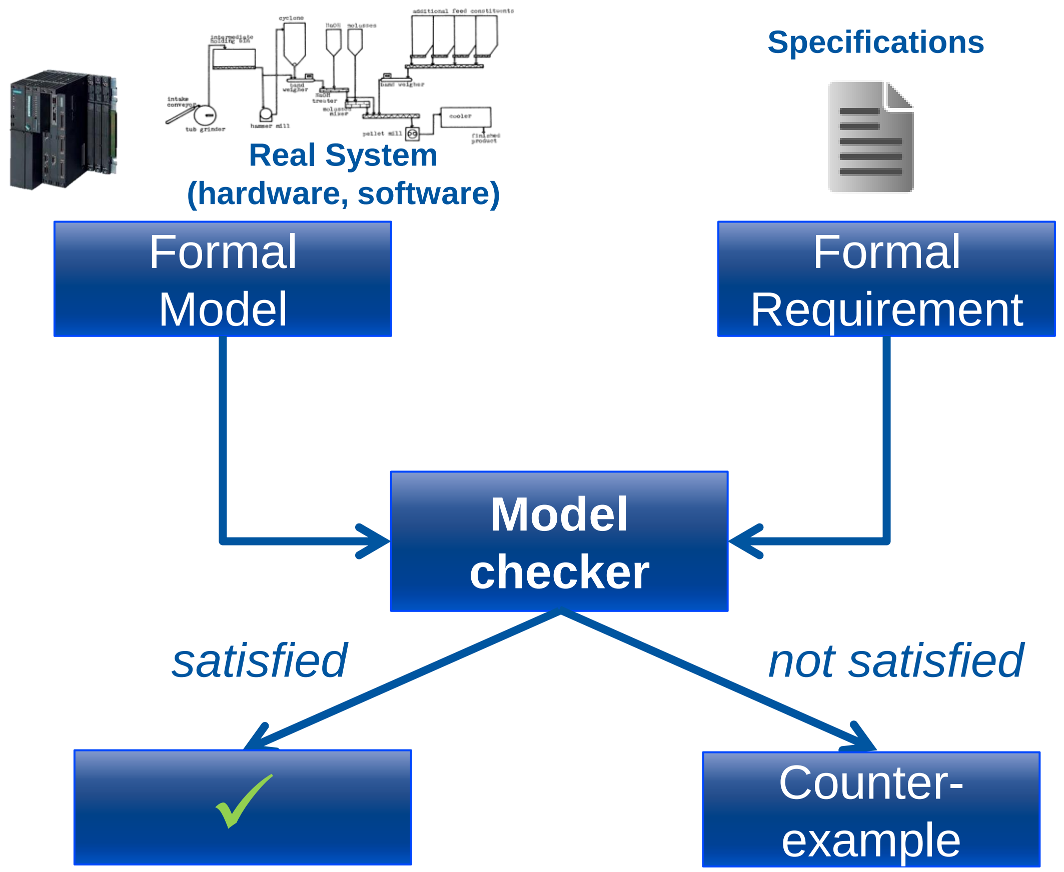
About formal verification

How to classify model checking?

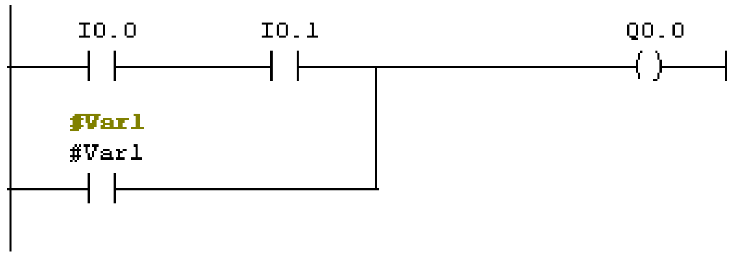

Verification
Formal methods
Formal verification
Testing
Model checking
Static analysis
Verif. based on theorem proving
Formal specification (B, Z, Alloy, …)
Formalisms: Automata, Petri Nets, Temporal Logic

YEARS / ANS CERN

Model checking



Testing vs. model checking



Q0.0 := (I0.0 AND I0.1) OR Var1

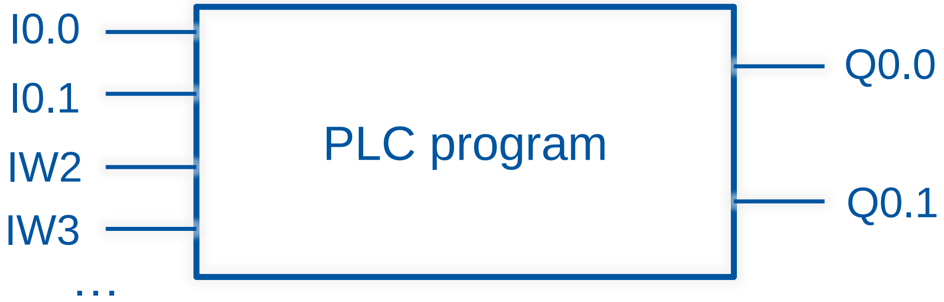
Requirement

*If **I0.0** is FALSE and **I0.1** is FALSE , then **Q0.0** is FALSE*

(Incomplete) testing **may** answer that this property is correct.

Model checking will answer that this property is **not** correct and it will provide a **counterexample**: Var1 == 1

Testing vs. model checking



Safety Requirement

*If **Q0.0** is TRUE, then **Q0.1** is FALSE*

Model checking will **explore all input combinations** and will verify the safety property

It's a **extremely complicated task for Testing**.

Model checking

1. **How to build the formal models?**

   Automata, Petri nets, Timed automata, …

2. **How to build the formal requirement?**

   Temporal Logic

Temporal logic

Boolean logic

operators

AND, OR, NOT

predicates

input=TRUE, temp>100

+

Temporal operators

in the future …

always …

once …

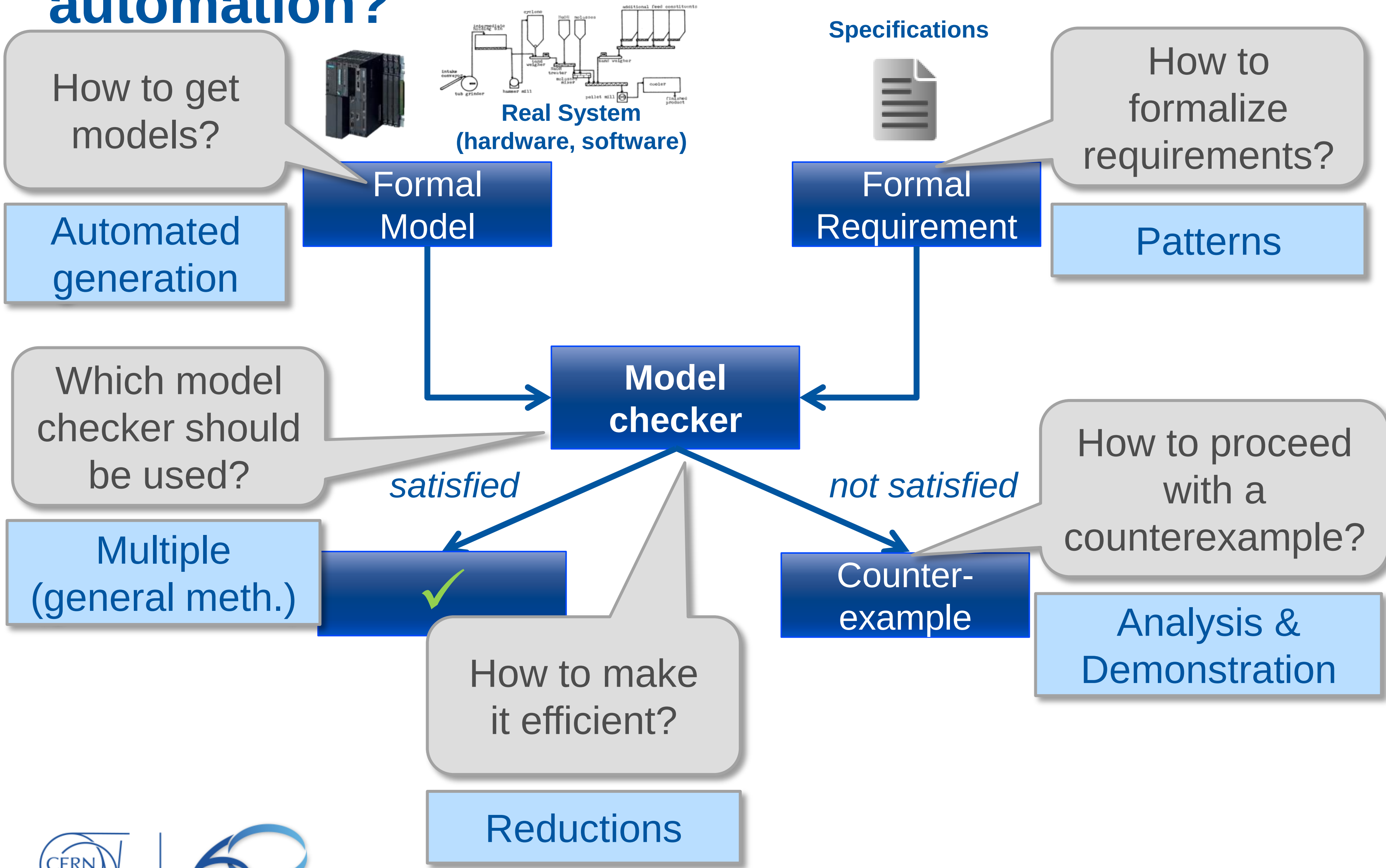
until…

Model checking

- 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**

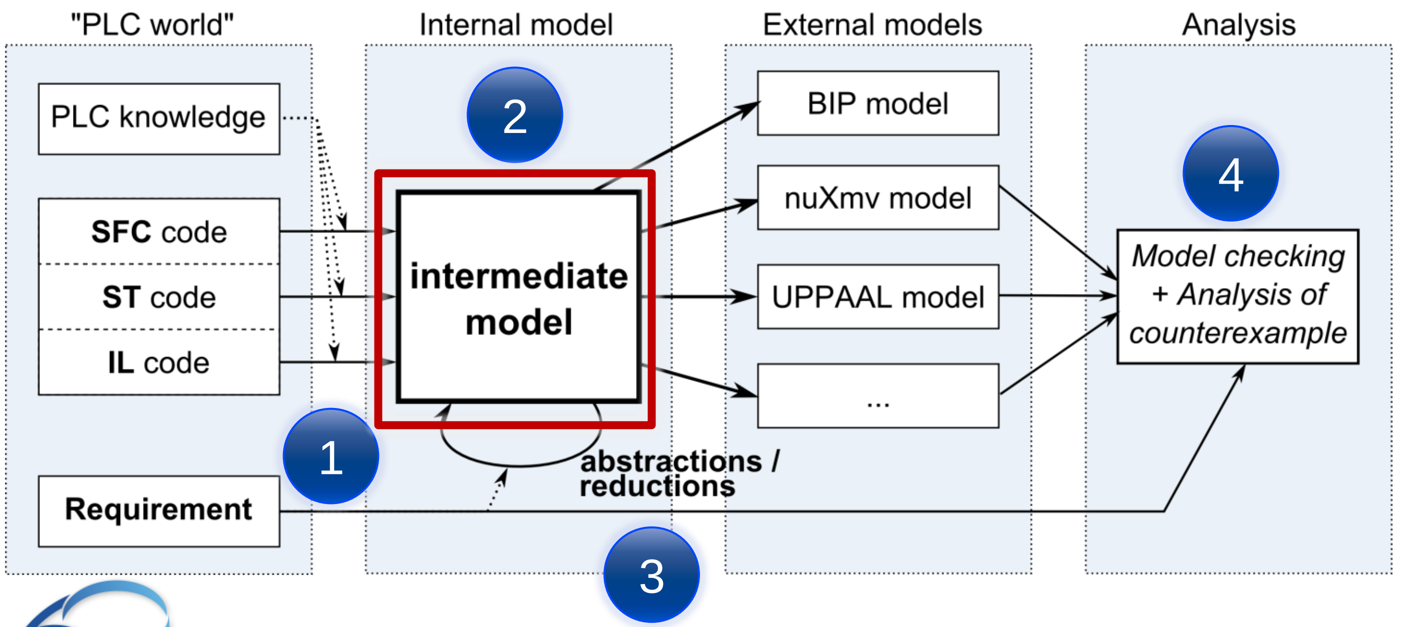
About our methodology

Why is not model checking widely used in automation?



Our approach: methodology overview

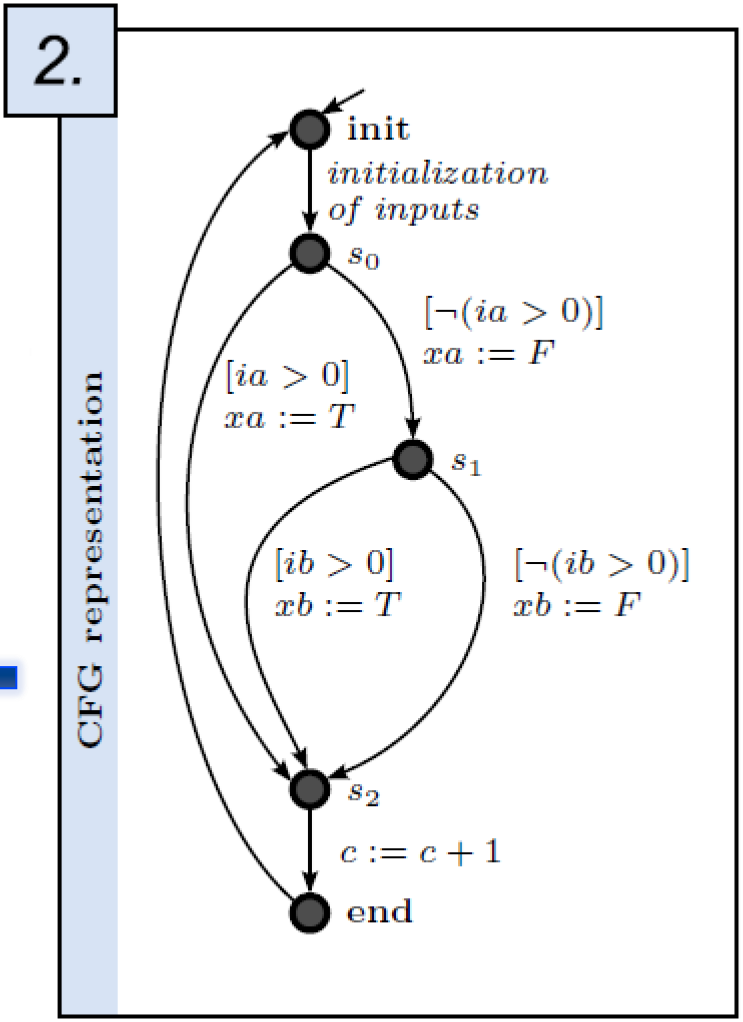
- **General** method for applying formal verification:
  - **Generate** formal models **automatically** out of PLC code.
  - Includes **several input PLC languages** (IEC 61131-3: SFC, ST, IL, Ladder, FBD).
  - Easy integration of **different formal verification** tools.



Model example

1. ST code

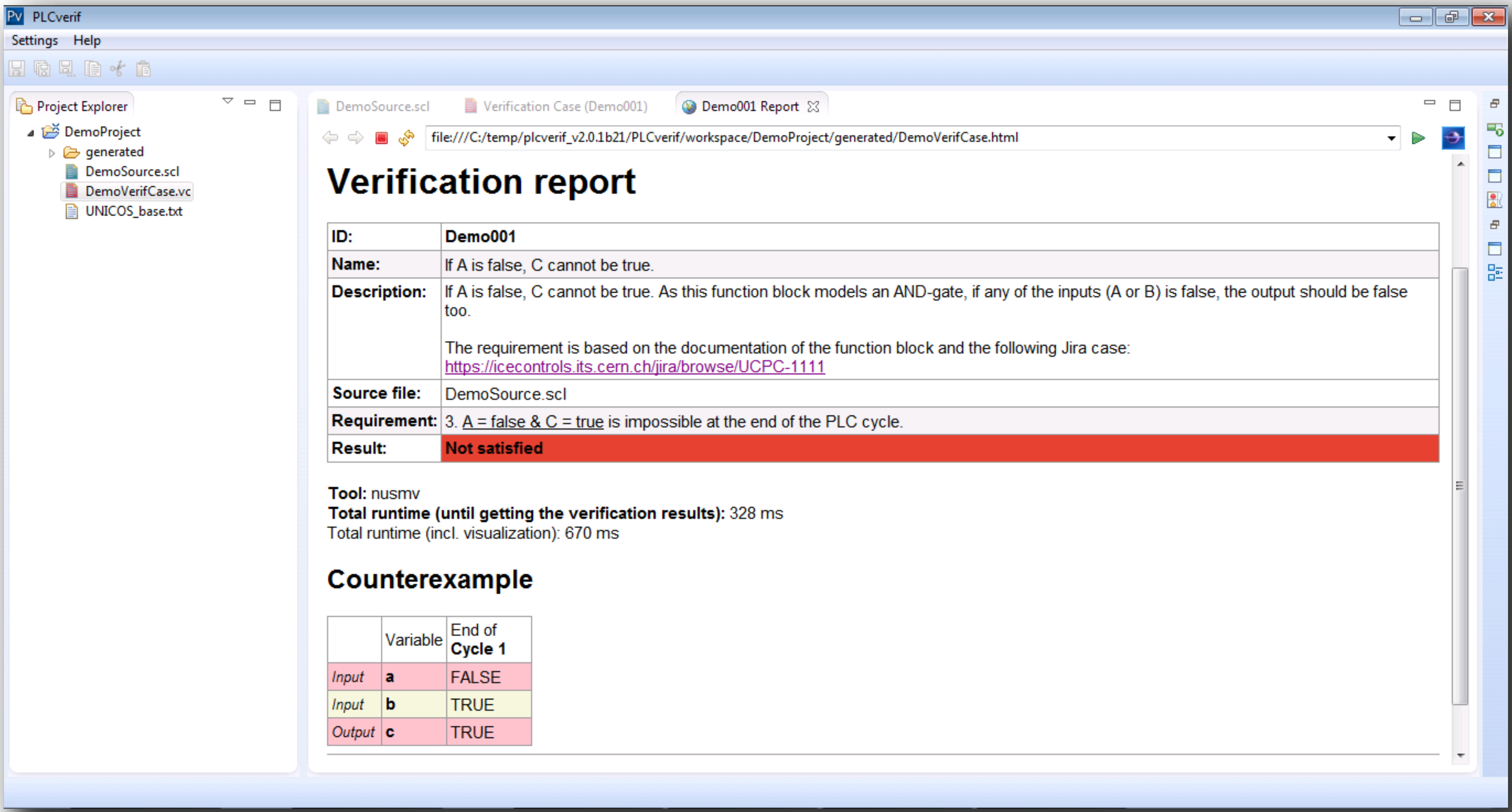
```
IF ia > 0 THEN
     xa := TRUE;
ELSE
     xa := FALSE;
     IF ib > 0 THEN
          xb := TRUE;
     ELSE
          xb := FALSE;
     END_IF;
END_IF;
c := c + 1;
```



3. NuSMV model

```
init(loc) := initial;
next(loc) := case
   loc = end : initial;
   loc = initial : s0;
   loc = s0 & ((IA > 0sd16_0)) : s2;
   loc = s0 & (!((IA > 0sd16_0))) : s1;
   loc = s1 & ((IB > 0sd16_0)) : s2;
   loc = s1 & (!((IB > 0sd16_0))) : s2;
   loc = s2 : end;
   TRUE: loc;
esac;
next(XA) := case
   loc = s0 & ((IA > 0sd16_0)) : TRUE;
   loc = s0 & (!((IA > 0sd16_0))) : FALSE;
   TRUE : XA;
esac;
...
```

Automatic generation tool



file:///C:/temp/plcverif_v2.0.1b21/PLCverif/workspace/DemoProject/generated/DemoVerifCase.html

Verification report

ID:	**Demo001**
Name:	If A is false, C cannot be true.
Description:	If A is false, C cannot be true. As this function block models an AND-gate, if any of the inputs (A or B) is false, the output should be false too. The requirement is based on the documentation of the function block and the following Jira case: https://icecontrols.its.cern.ch/jira/browse/UCPC-1111
Source file:	DemoSource.scl
Requirement:	3. A = false & C = true is impossible at the end of the PLC cycle.
Result:	**Not satisfied**

Tool: nusmv
Total runtime (until getting the verification results): 328 ms
Total runtime (incl. visualization): 670 ms

Counterexample

	Variable	End of **Cycle 1**
Input	**a**	FALSE
Input	**b**	TRUE
Output	**c**	TRUE

Some results

Results with the UNICOS library

- CERN PLC programs developed with the UNICOS Framework:
  - **Library** of objects (representing the logic of real equipment)
  - Expressed in PLC code: **ST** language.
- Metrics of the PLC program

Metric	OnOff PLC code
Lines of code	600
Input variables	60
Output variables	62
Data types	Booleans, **integers**, **floats**, **time**, etc.
Timers	3

YEARS / ANS CERN

Results with the UNICOS library

Metric	Non-reduced model	Reduced model	Specific Model *
Potential state space	$\sim 10^{218}$	$\sim 10^{36}$	$\sim 10^{10}$
# Variables	255	118	33
Generation	0.3 s	11.3 s	12.6 s
NuSMV Verification (with cex.)	–	160.8 s	0.5 s

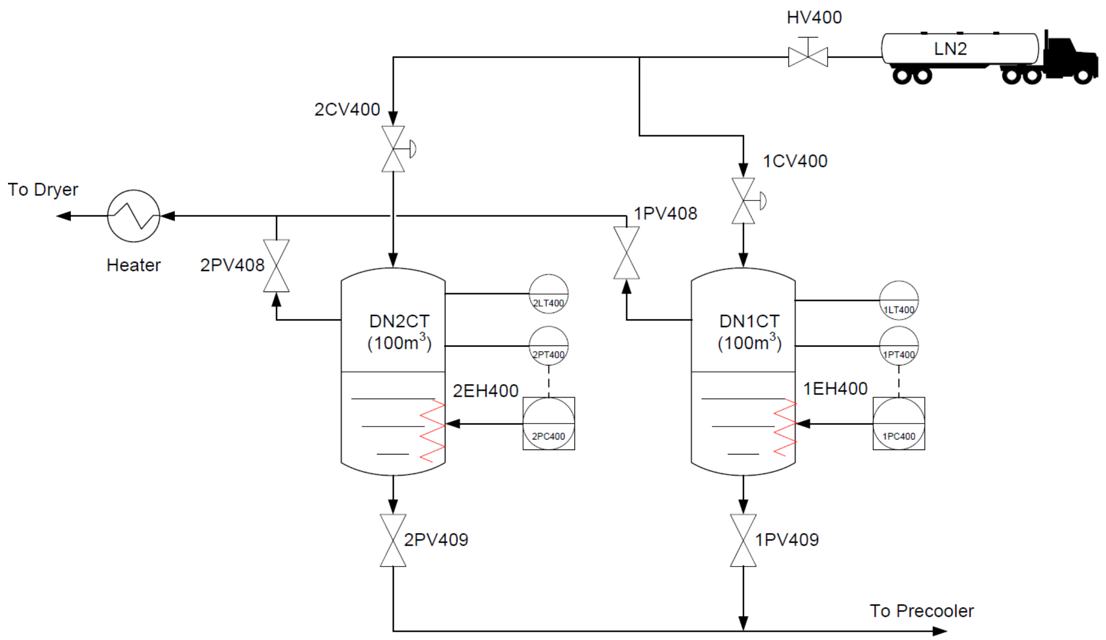
* Based on a real requirement about the mode manager of the OnOff object

Cone of Influence algorithm
(**property preserving reduction techniques**)

CERN | YEARS / ANS CERN

Results with an UNICOS control system

QSDN control system



PLC code

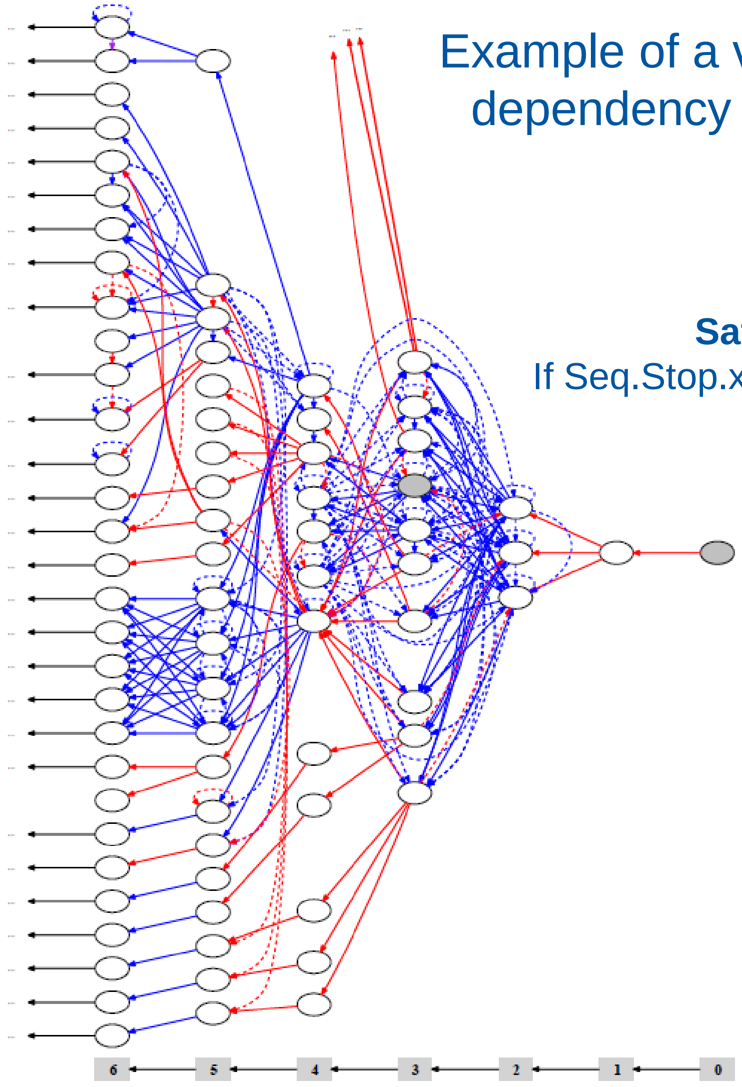
- ❑ 110 FBs and FCs
- ❑ 17,500 lines of code

Formal model

- ❑ 302 automata
- ❑ PSS = **~10^{31985}**

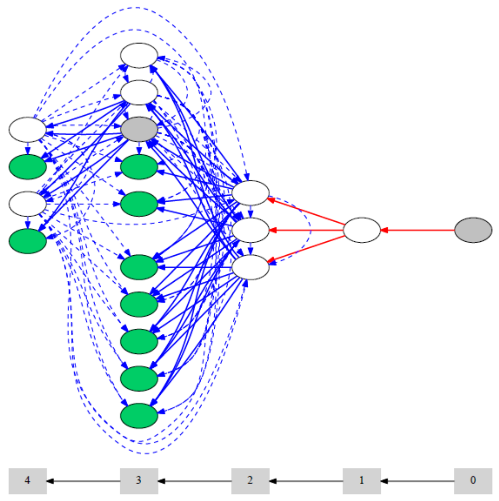
Goal: Verify the **specific logic** of the application

Results with an UNICOS control system



Example of a variable dependency graph

Reduced variable dependency graph



Safety req.
If Seq.Stop.x → Valve1.AuOffR

Metric	Non-reduced model	Reduced Model*	Abstract Model **
PSS	$\sim 10^{31985}$	$\sim 10^{5048}$	$\sim 10^{13}$
# Variables	31,402	3757	20
Generation	4.2 s	15.3 s	5.4 s
NuSMV Verification	–	–	0.25 s

* Using **property preserving** reduction techniques
** Using **non-property preserving** reduction techniques

Conclusion and summary

- Model checking can be applied to PLC programs.
- **Difficulty can be hidden** from the control engineers:
  - Automated model generation, requirement patterns, automatic reduction techniques and counterexample analysis.
- We have **found discrepancies in our systems**.

  Sources of problems:
  - Incomplete or incorrect **specification.**
  - Mistake in the **implementation.**
- Bugs can be proved and “help” is provided to find the source of the bug.
- Future work: Concurrency + formal specifications + improvement of reduction techniques.

CERN

www.cern.ch