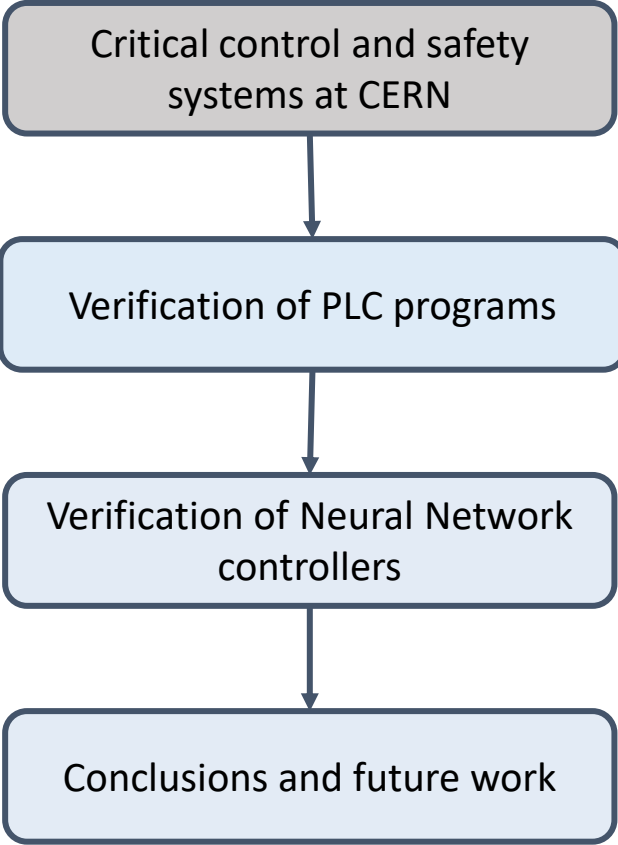


# ***Formal verification of critical PLC programs and neural network controllers at CERN***

Borja Fernández Adiego

*Contains joint work of several members and former members of the **BE-ICS group at CERN***

# Roadmap



## **Context – CERN (European Organization for Nuclear Research)**

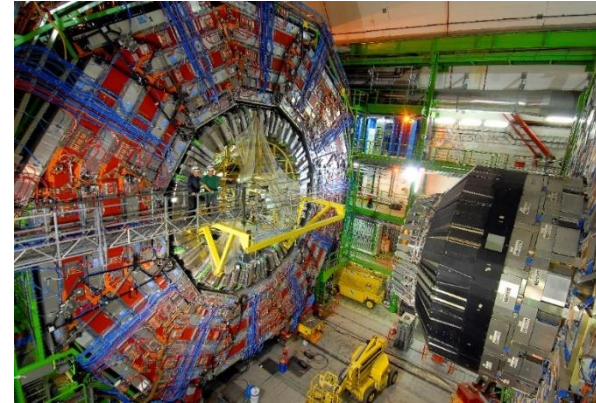
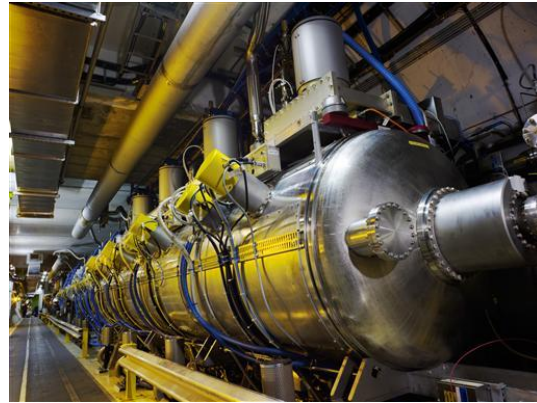
## Context – CERN (European Organization for Nuclear Research)

- CERN has the biggest particle accelerator complex in the world and **many critical and complex industrial installations**

## Context – CERN (European Organization for Nuclear Research)

- CERN has the biggest particle accelerator complex in the world and **many critical and complex industrial installations**

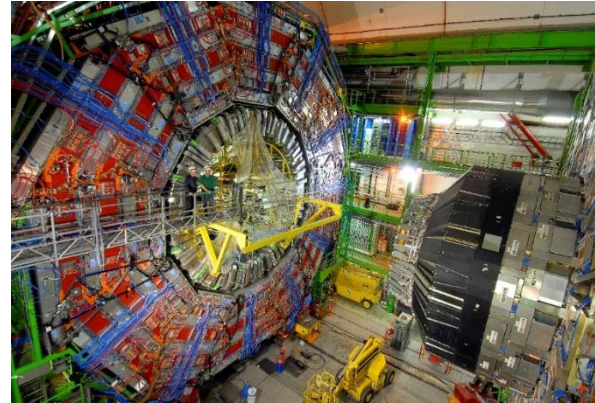
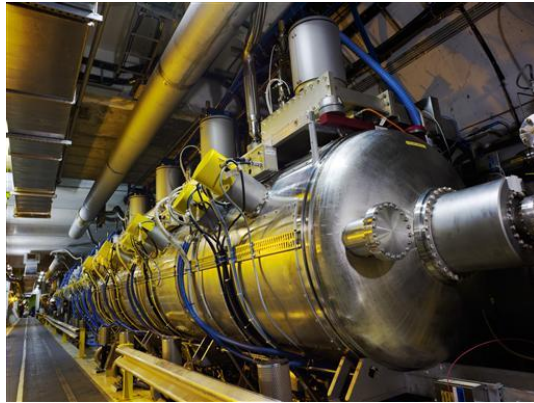
LHC  
(Large Hadron Collider)



## Context – CERN (European Organization for Nuclear Research)

- CERN has the biggest particle accelerator complex in the world and **many critical and complex industrial installations**

LHC  
(Large Hadron Collider)



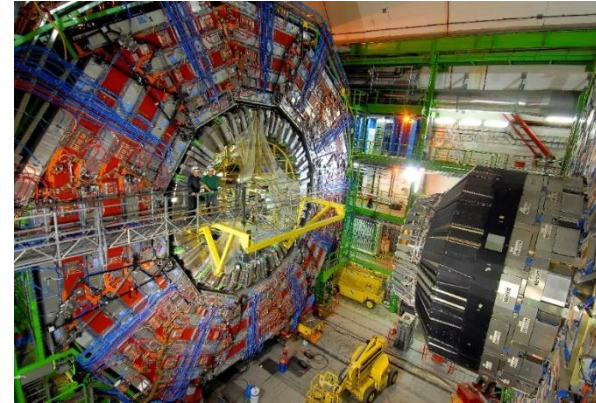
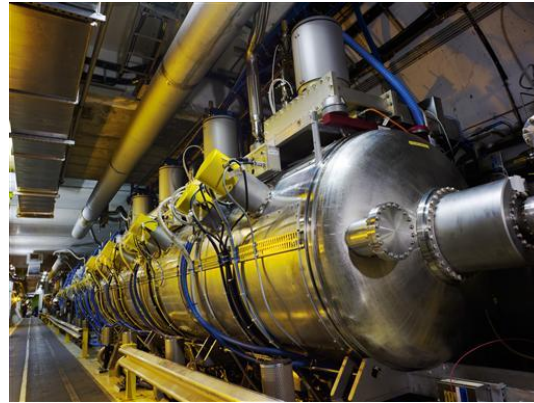
Access and  
protection  
systems



## Context – CERN (European Organization for Nuclear Research)

- CERN has the biggest particle accelerator complex in the world and **many critical and complex industrial installations**

LHC  
(Large Hadron Collider)



Access and  
protection  
systems

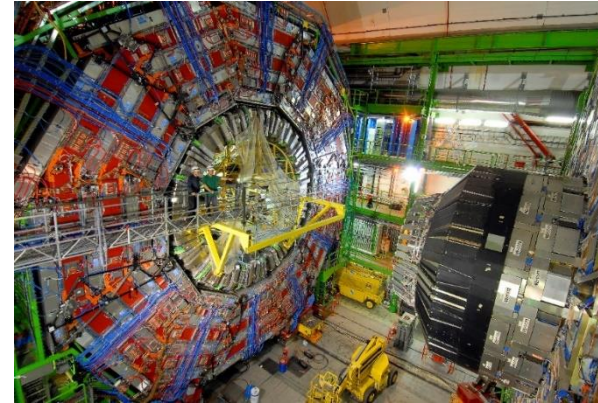
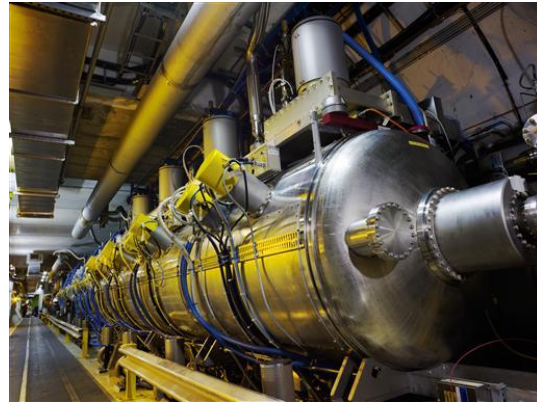


Cryogenics plants

## Context – CERN (European Organization for Nuclear Research)

- CERN has the biggest particle accelerator complex in the world and **many critical and complex industrial installations**

LHC  
(Large Hadron Collider)



Access and  
protection  
systems



Cryogenics plants



Cooling and ventilation  
systems

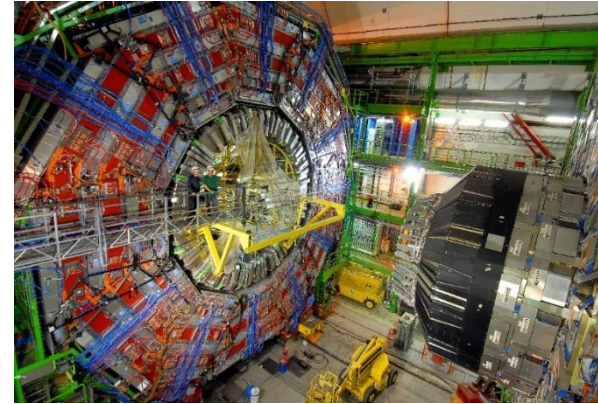
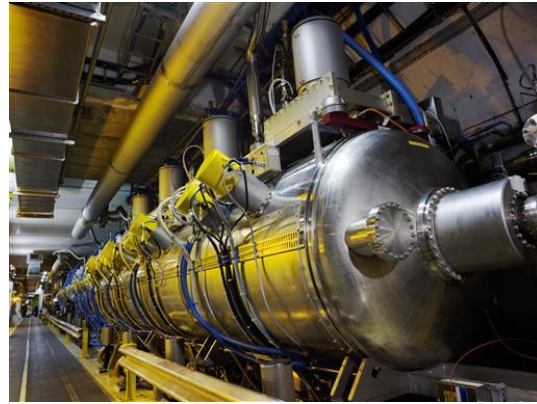




## Context – CERN (European Organization for Nuclear Research)

- CERN has the biggest particle accelerator complex in the world and **many critical and complex industrial installations**

LHC  
(Large Hadron Collider)



Access and  
protection  
systems



Cryogenics plants



Cooling and ventilation  
systems



Superconducting magnet test benches

**BE-ICS group in a nutshell**

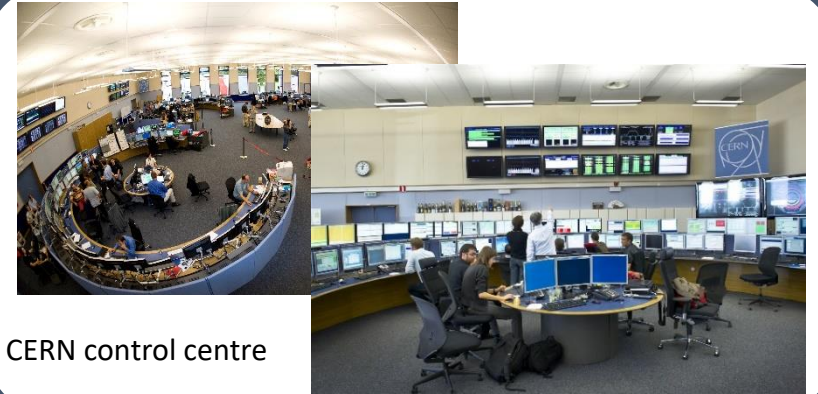
## BE-ICS group in a nutshell

- “BE-ICS provides the **technology, frameworks, engineering** and CERN-wide **support** for systems and projects in all domains using standard **industrial control** solutions” <https://be-dep-ics.web.cern.ch>



## BE-ICS group in a nutshell

- “BE-ICS provides the **technology, frameworks, engineering** and CERN-wide **support** for systems and projects in all domains using standard **industrial control** solutions” <https://be-dep-ics.web.cern.ch>
- BE-ICS is in charge of the design and development of industrial **control** and **safety** systems



CERN control centre



Cooling and ventilation systems



Superconducting magnet test benches



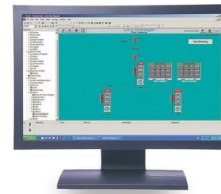
Cryogenics plants



## BE-ICS group in a nutshell

- “BE-ICS provides the **technology, frameworks, engineering** and CERN-wide **support** for systems and projects in all domains using standard **industrial control** solutions” <https://be-dep-ics.web.cern.ch>
- BE-ICS is in charge of the design and development of industrial **control** and **safety** systems

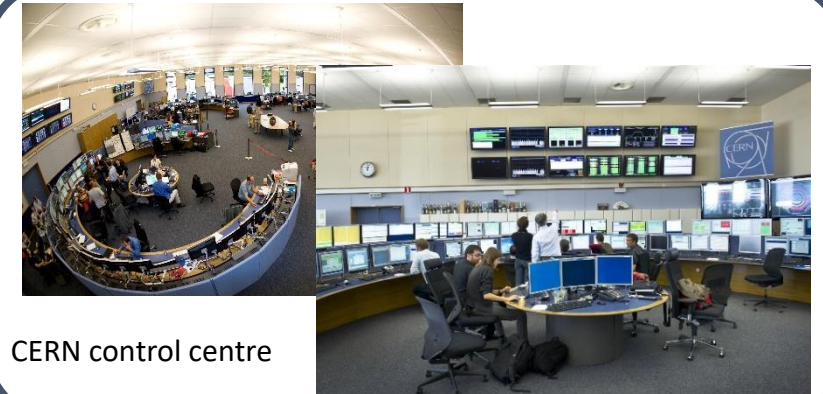
WinCCOA



Industrial PCs (FEC)



Programmable Logic Controllers (PLC)



CERN control centre



Cooling and ventilation systems



Superconducting magnet test benches



Cryogenics plants

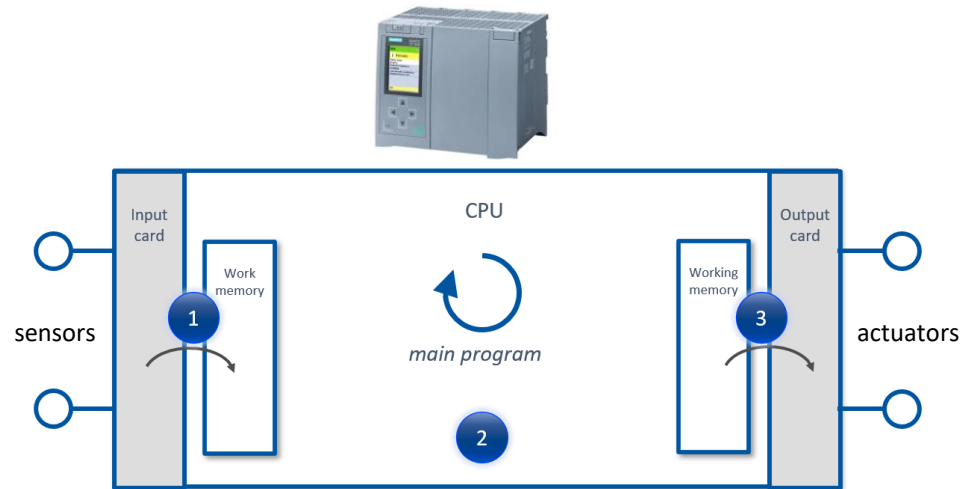
**Context – PLCs at CERN**

## Context – PLCs at CERN

- At CERN, more than 3000 **PLCs** (Programmable Logic Controllers) are installed to **control** and/or **protect** the installations

## Context – PLCs at CERN

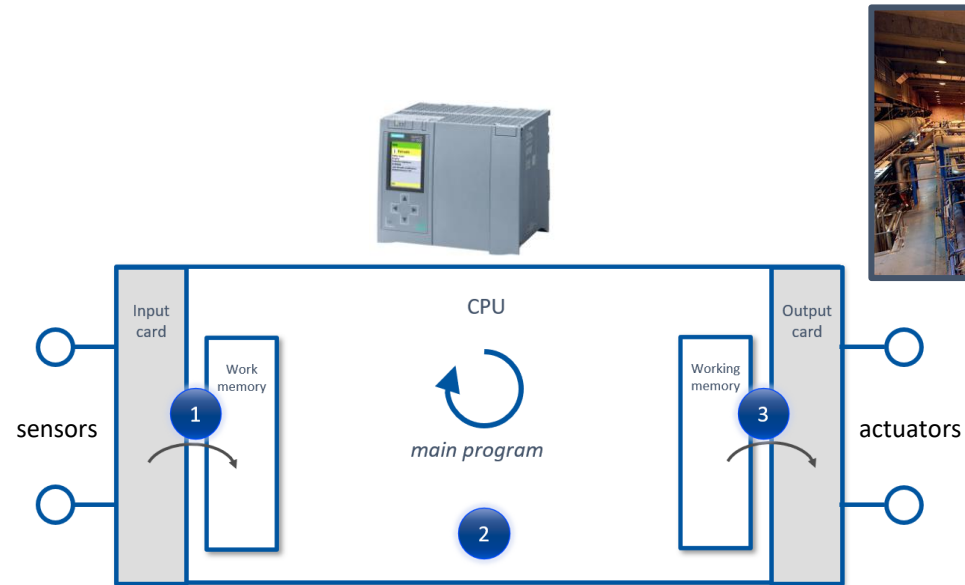
- At CERN, more than 3000 **PLCs** (Programmable Logic Controllers) are installed to **control** and/or **protect** the installations





## Context – PLCs at CERN

- At CERN, more than 3000 **PLCs** (Programmable Logic Controllers) are installed to **control** and/or **protect** the installations



### Example of PLC program

```
(* MODE MANAGER *)
IF NOT (#HLD AND #PHLD) THEN
  (* Forced Mode *)
  IF (#AuMoSt_aux OR #MMoSt_aux OR #SoftLDSt_aux) AND
    #EMFoMoR AND NOT (#AuIhFoMo) THEN

    #AuMoSt_aux := FALSE;
    #MMoSt_aux := FALSE;
    #FoMoSt_aux := TRUE;
    #SoftLDSt_aux := FALSE;

  (* Software Local Mode *)
  ELSIF (#AuMoSt_aux OR #MMoSt_aux) AND #EMSoftLDR AND NOT #AuIhFoMo THEN

    #AuMoSt_aux := FALSE;
    #MMoSt_aux := FALSE;
    #FoMoSt_aux := FALSE;
    #SoftLDSt_aux := TRUE;

  (* Manual Mode *)
  ELSIF (#AuMoSt_aux OR #FoMoSt_aux OR #SoftLDSt_aux) AND
    #EMMoR AND NOT (#AuIhMo) THEN

    #AuMoSt_aux := FALSE;
    #MMoSt_aux := TRUE;
    #FoMoSt_aux := FALSE;
    #SoftLDSt_aux := FALSE;

  (* Auto Mode *)
  ELSIF (#MMoSt_aux AND (#EMAuMoR OR #EAuMoR)) OR
    (#FoMoSt_aux AND #EMAuMoR) OR
    (#SoftLDSt_aux AND #EMAuMoR) OR
    (#MMoSt_aux AND #AuIhMo) OR
    (#FoMoSt_aux AND #AuIhFoMo) OR
    (#SoftLDSt_aux AND #AuIhFoMo) OR
    NOT (#AuMoSt_aux OR #MMoSt_aux OR #FoMoSt_aux OR #SoftLDSt_aux) THEN

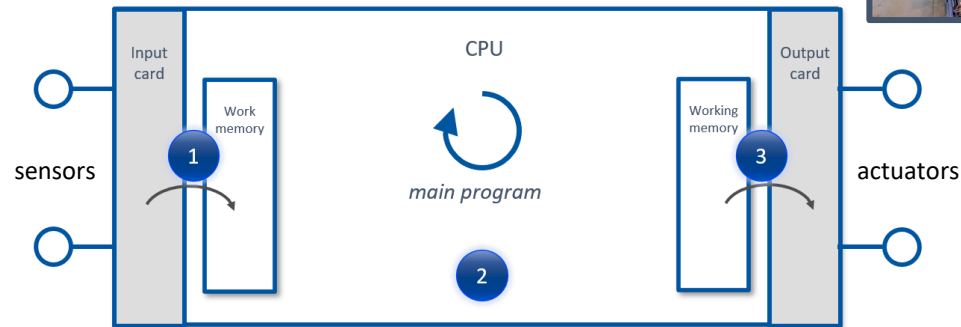
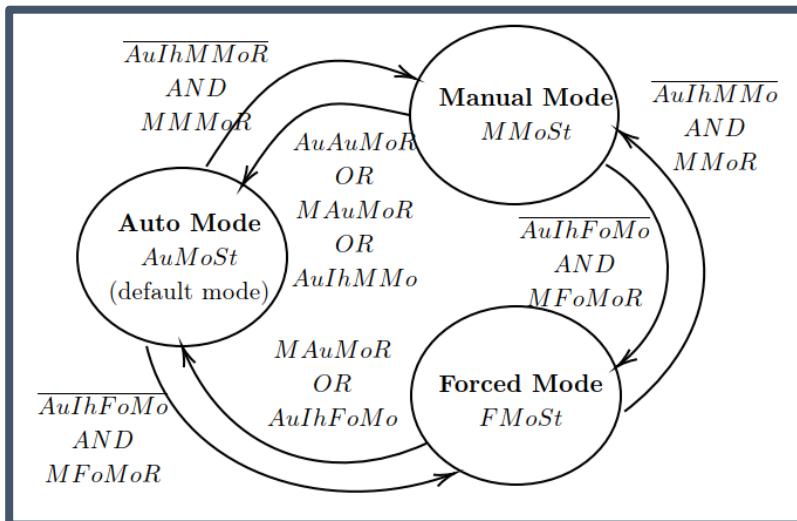
    #AuMoSt_aux := TRUE;
```

## Context – PLCs at CERN

- At CERN, more than 3000 **PLCs** (Programmable Logic Controllers) are installed to **control** and/or **protect** the installations



### Example of specification (ambiguous and incomplete)



### Example of PLC program

```

(* MODE MANAGER *)
IF NOT (#HLD AND #PHLD) THEN
  (* Forced Mode *)
  IF (#AuMoSt_aux OR #MMoSt_aux OR #SoftLDSt_aux) AND
    #EMFoMoR AND NOT (#AuIhFoMo) THEN

    #AuMoSt_aux := FALSE;
    #MMoSt_aux := FALSE;
    #FoMoSt_aux := TRUE;
    #SoftLDSt_aux := FALSE;

    (* Software Local Mode *)
    ELSIF (#AuMoSt_aux OR #MMoSt_aux) AND #E_MSoftLDR AND NOT #AuIhFoMo THEN

    #AuMoSt_aux := FALSE;
    #MMoSt_aux := FALSE;
    #FoMoSt_aux := FALSE;
    #SoftLDSt_aux := TRUE;

    (* Manual Mode *)
    ELSIF (#AuMoSt_aux OR #FoMoSt_aux OR #SoftLDSt_aux) AND
      #EMMoR AND NOT (#AuIhMMo) THEN

    #AuMoSt_aux := FALSE;
    #MMoSt_aux := TRUE;
    #FoMoSt_aux := FALSE;
    #SoftLDSt_aux := FALSE;

    (* Auto Mode *)
    ELSIF (#MMoSt_aux AND (#EMAuMoR OR #E_AuAuMoR)) OR
      (#FoMoSt_aux AND #E_MAuMoR) OR
      (#SoftLDSt_aux AND #E_MAuMoR) OR
      (#MMoSt_aux AND #AuIhMMo) OR
      (#FoMoSt_aux AND #AuIhFoMo) OR
      (#SoftLDSt_aux AND #AuIhFoMo) OR
      NOT (#AuMoSt_aux OR #MMoSt_aux OR #FoMoSt_aux OR #SoftLDSt_aux) THEN

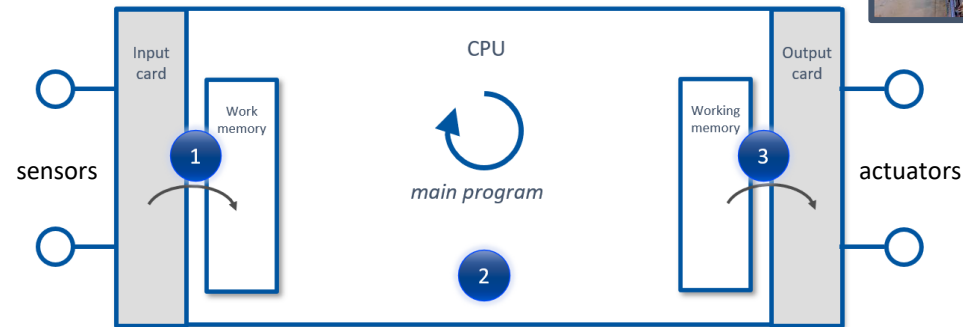
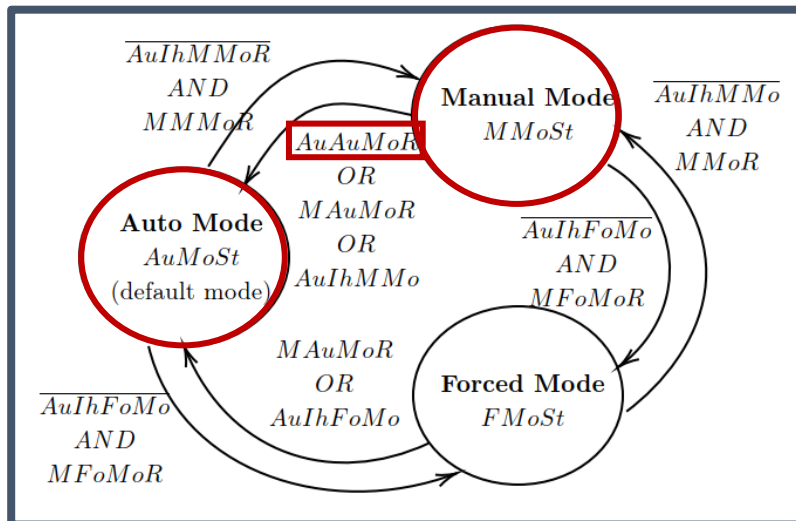
    #AuMoSt_aux := TRUE;
  
```

## Context – PLCs at CERN

- At CERN, more than 3000 **PLCs** (Programmable Logic Controllers) are installed to **control** and/or **protect** the installations



### Example of specification (ambiguous and incomplete)



### Need to guarantee “complex” properties

if **MMoSt** and **AuAuMoR** at any time before the end of the PLC cycle,  
then **AuMoSt** should be true at the end of the same PLC cycle

### Example of PLC program

```

(* MODE MANAGER *)
IF NOT (#HLD AND #PHLD) THEN
  (* Forced Mode *)
  IF (#AuMoSt_aux OR #MMoSt_aux OR #SoftLDSt_aux) AND
    #E_MFoMoR AND NOT (#AuIhFoMo) THEN

    #AuMoSt_aux := FALSE;
    #MMoSt_aux := FALSE;
    #FoMoSt_aux := TRUE;
    #SoftLDSt_aux := FALSE;

  (* Software Local Mode *)
  ELSIF (#AuMoSt_aux OR #MMoSt_aux) AND #E_MSoftLDR AND NOT #AuIhFoMo THEN

    #AuMoSt_aux := FALSE;
    #MMoSt_aux := FALSE;
    #FoMoSt_aux := FALSE;
    #SoftLDSt_aux := TRUE;

  (* Manual Mode *)
  ELSIF (#AuMoSt_aux OR #FoMoSt_aux OR #SoftLDSt_aux) AND
    #E_MMoR AND NOT (#AuIhMMo) THEN

    #AuMoSt_aux := FALSE;
    #MMoSt_aux := TRUE;
    #FoMoSt_aux := FALSE;
    #SoftLDSt_aux := FALSE;

  (* Auto Mode *)
  ELSIF (#MMoSt_aux AND (#E_MAuMoR OR #E_AuAuMoR)) OR
    (#FoMoSt_aux AND #E_MAuMoR) OR
    (#SoftLDSt_aux AND #E_MAuMoR) OR
    (#MMoSt_aux AND #AuIhMMo) OR
    (#FoMoSt_aux AND #AuIhFoMo) OR
    (#SoftLDSt_aux AND #AuIhFoMo) OR
    NOT (#AuMoSt_aux OR #MMoSt_aux OR #FoMoSt_aux OR #SoftLDSt_aux) THEN

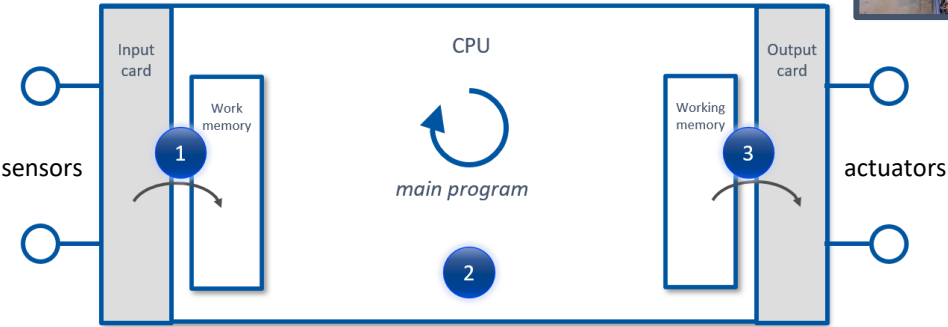
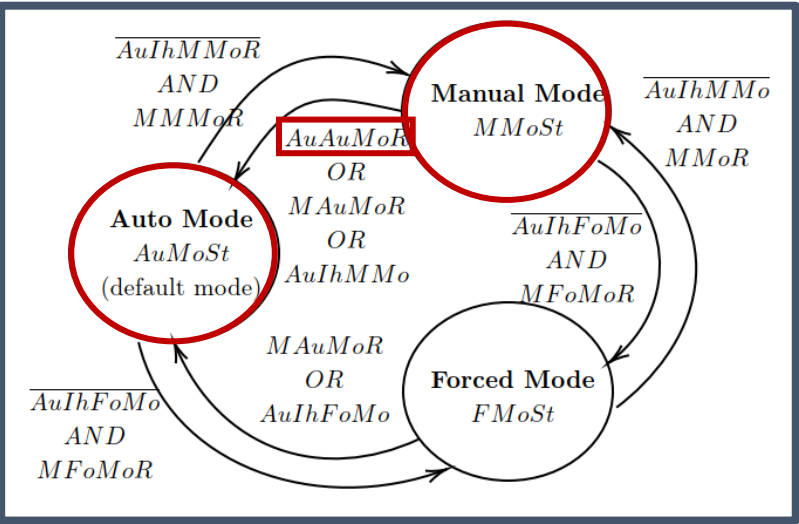
    #AuMoSt_aux := TRUE;
  
```

# Context – PLCs at CERN

- At CERN, more than 3000 **PLCs** (Programmable Logic Controllers) are installed to **control** and/or **protect** the installations



## Example of specification (ambiguous and incomplete)



## Example of PLC program

```
(* MODE MANAGER *)
IF NOT (#HLD AND #PHLD) THEN
  (* Forced Mode *)
  IF (#AuMoSt_aux OR #MMoSt_aux OR #SoftLDSt_aux) AND
    #EMFoMoR AND NOT (#AuIhFoMo) THEN

    #AuMoSt_aux := FALSE;
    #MMoSt_aux := FALSE;
    #FoMoSt_aux := TRUE;
    #SoftLDSt_aux := FALSE;

  (* Software Local Mode *)
  ELSIF (#AuMoSt_aux OR #MMoSt_aux) AND #EMSoftLDR AND NOT #AuIhFoMo THEN

    #AuMoSt_aux := FALSE;
    #MMoSt_aux := FALSE;
    #FoMoSt_aux := FALSE;
    #SoftLDSt_aux := TRUE;

  (* Manual Mode *)
  ELSIF (#AuMoSt_aux OR #FoMoSt_aux OR #SoftLDSt_aux) AND
    #EMMoR AND NOT (#AuIhMo) THEN

    #AuMoSt_aux := FALSE;
    #MMoSt_aux := TRUE;
    #FoMoSt_aux := FALSE;
    #SoftLDSt_aux := FALSE;

  (* Auto Mode *)
  ELSIF (#MMoSt_aux AND (#EMAuMoR OR #EAuMoR)) OR
    (#FoMoSt_aux AND #EMAuMoR) OR
    (#SoftLDSt_aux AND #EMAuMoR) OR
    (#MMoSt_aux AND #AuIhMo) OR
    (#FoMoSt_aux AND #AuIhFoMo) OR
    (#SoftLDSt_aux AND #AuIhFoMo) OR
    NOT (#AuMoSt_aux OR #MMoSt_aux OR #FoMoSt_aux OR #SoftLDSt_aux) THEN

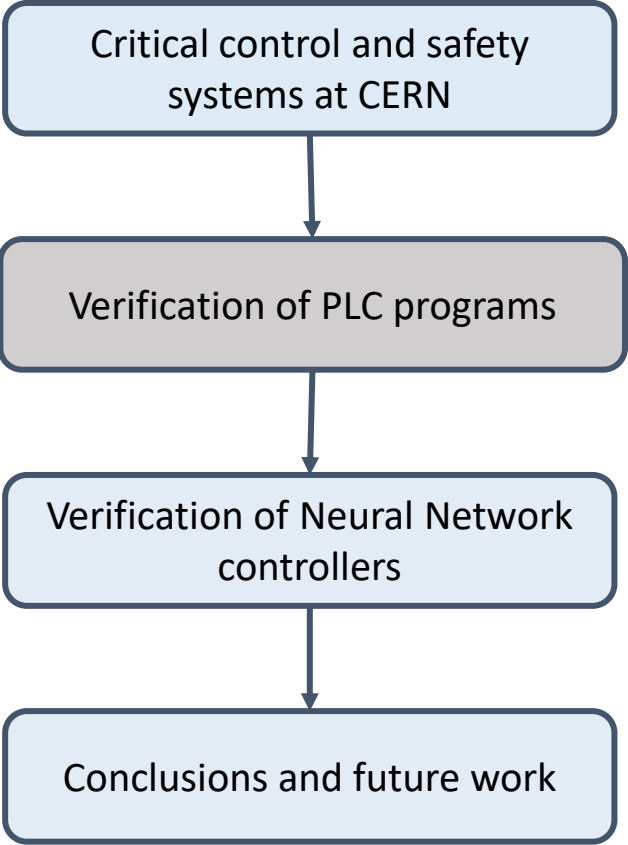
    #AuMoSt_aux := TRUE;
```

Need to guarantee “complex” properties  
if **MMoSt** and **AuAuMoR** at any time before the end of the PLC cycle,  
then **AuMoSt** should be true at the end of the same PLC cycle

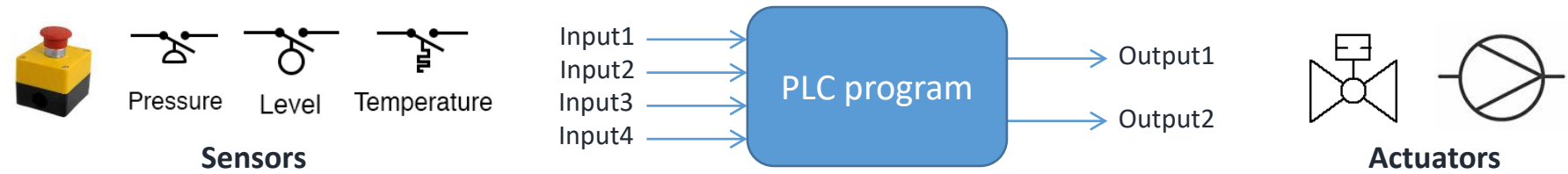
Formal verification  
(e.g. model checking)



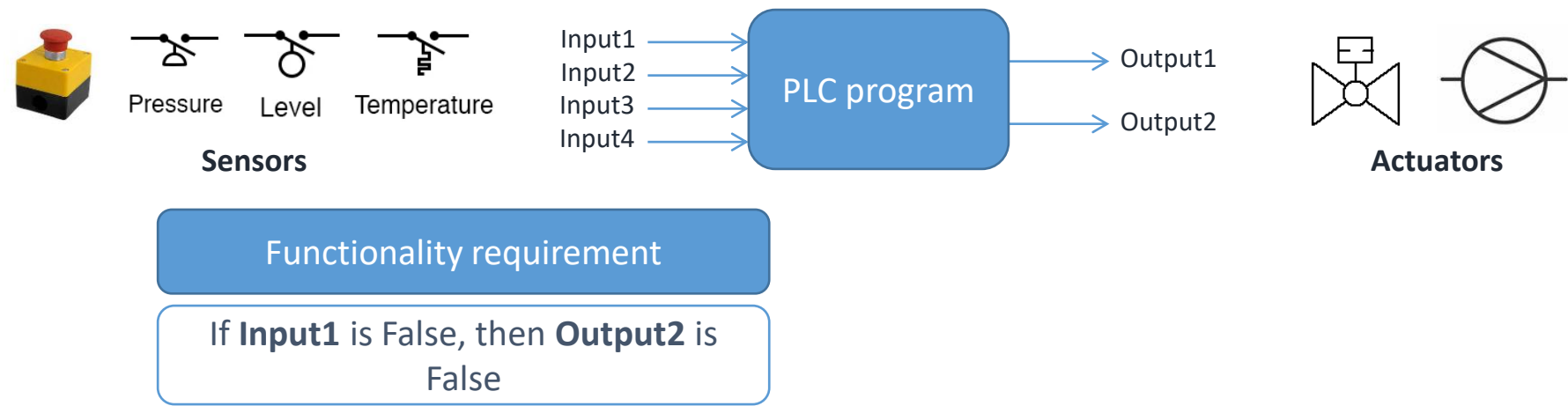
# Roadmap



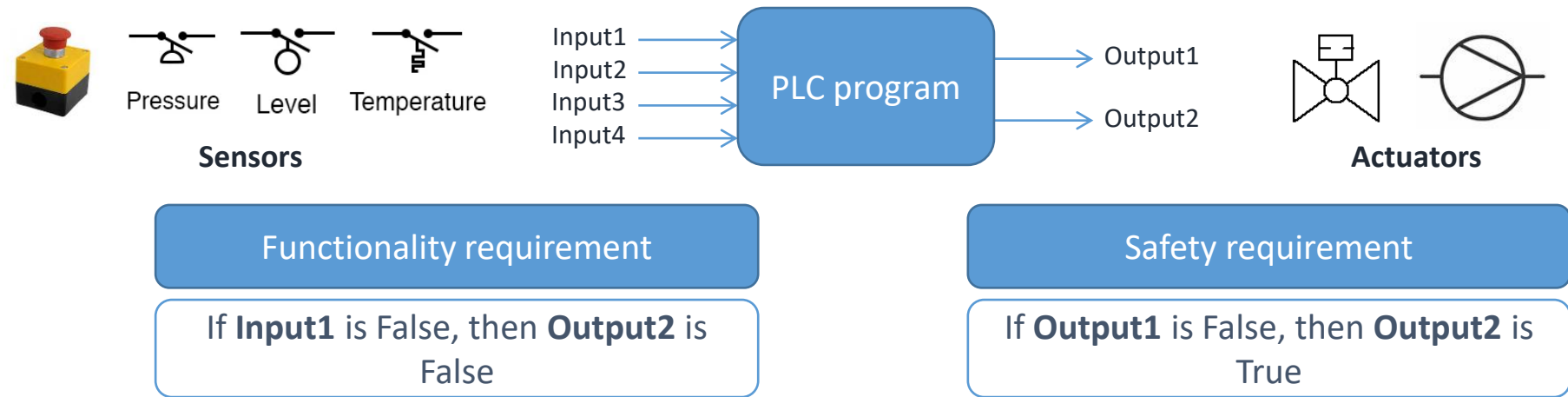
# Why formal verification?



# Why formal verification?

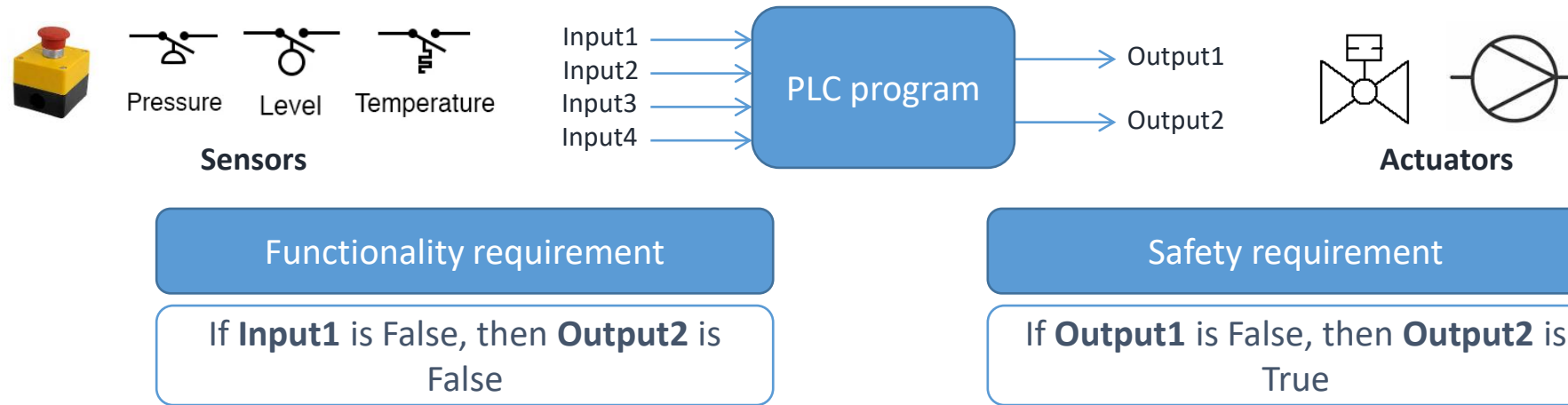


# Why formal verification?



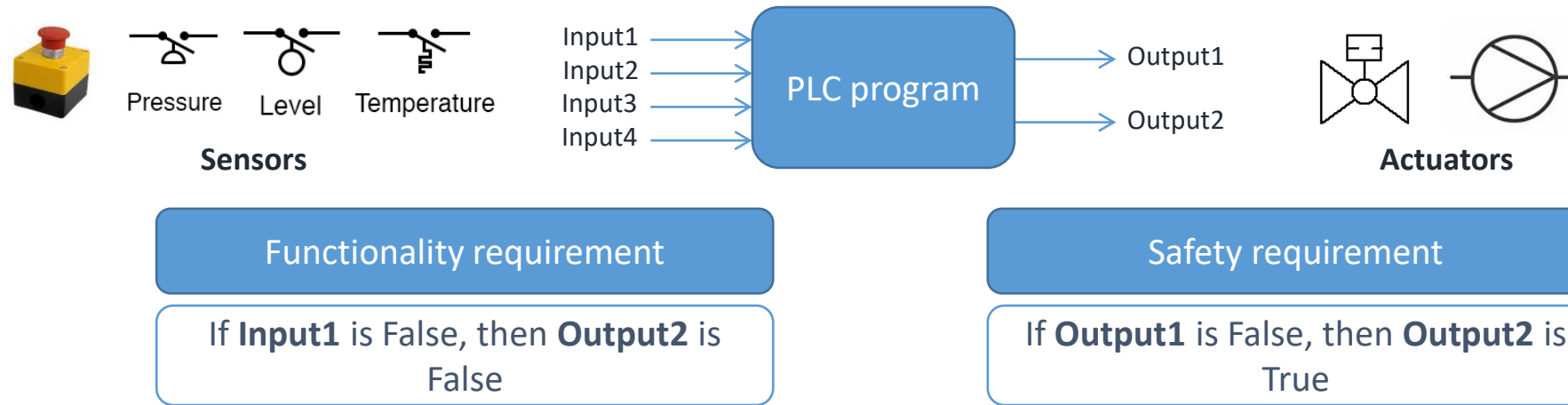


## Why formal verification?



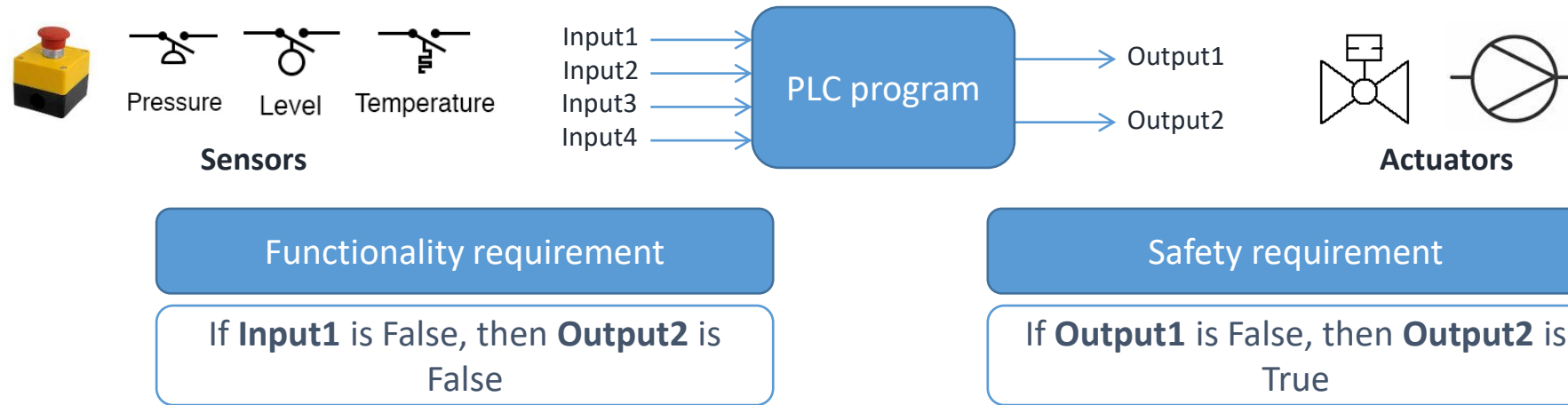
- If "*Input1*", "*Input2*", "*Input3*" and "*Input4*" are **BOOL**, then we need to check  $2^4 = 16$  combinations

## Why formal verification?



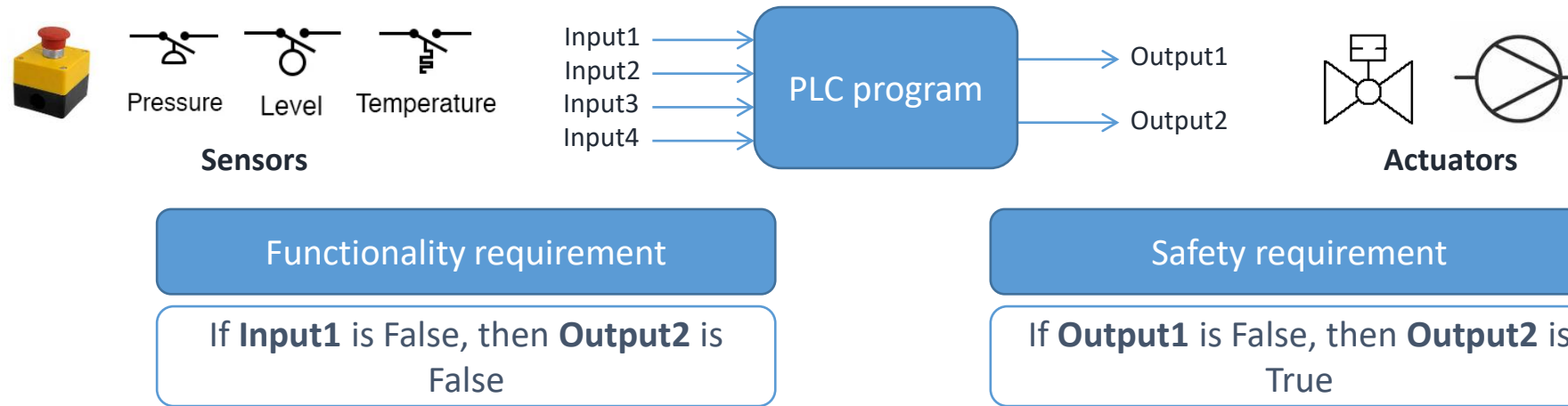
- If "*Input1*", "*Input2*", "*Input3*" and "*Input4*" are **BOOL**, then we need to check  $2^4 = 16$  combinations
- If they are **INT** (16-bit), then  $2^{16*4} \approx 1.8*10^{19}$  combinations

## Why formal verification?



- If "*Input1*", "*Input2*", "*Input3*" and "*Input4*" are **BOOL**, then we need to check  $2^4 = 16$  combinations
- If they are **INT** (16-bit), then  $2^{16*4} \approx 1.8*10^{19}$  combinations
- for large systems (many variables), such requirements **cannot** (practically) **be checked by using testing techniques**

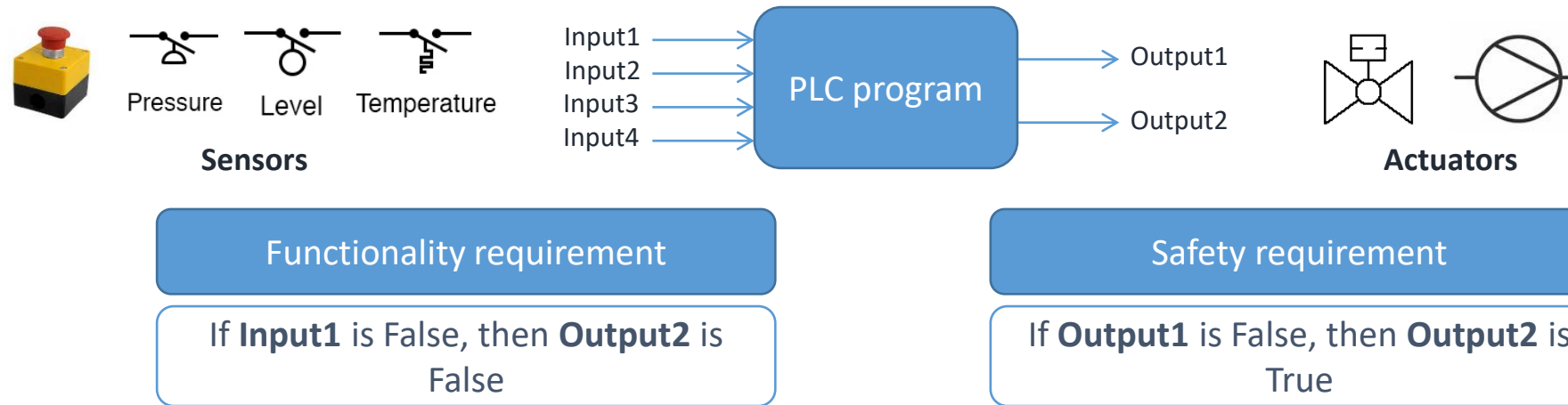
## Why formal verification?



- If "*Input1*", "*Input2*", "*Input3*" and "*Input4*" are **BOOL**, then we need to check  $2^4 = 16$  combinations
- If they are **INT** (16-bit), then  $2^{16*4} \approx 1.8*10^{19}$  combinations
- for large systems (many variables), such requirements **cannot** (practically) **be checked by using testing techniques**
- **Peer reviews** and **testing** can (normally) catch most of the "problems" (e.g. code bugs), but not the **corner cases**

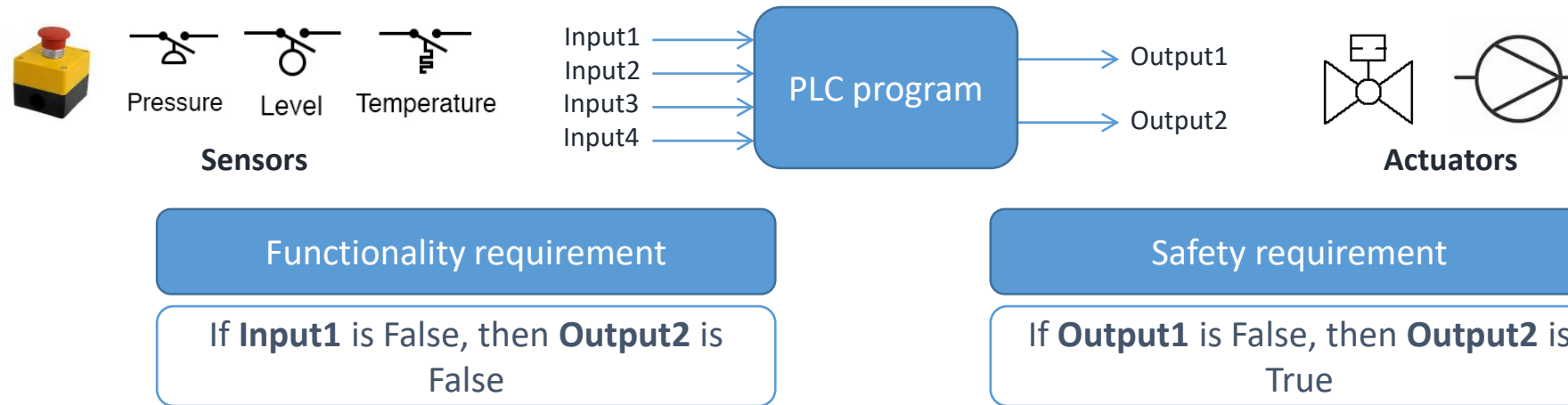


## Why formal verification?



- If “*Input1*”, “*Input2*”, “*Input3*” and “*Input4*” are **BOOL**, then we need to check  $2^4 = 16$  combinations
- If they are **INT** (16-bit), then  $2^{16*4} \approx 1.8*10^{19}$  combinations
- for large systems (many variables), such requirements **cannot** (practically) **be checked by using testing techniques**
- **Peer reviews** and **testing** can (normally) catch most of the “problems” (e.g. code bugs), but not the **corner cases**
  - **E.g. Ariane 5 rocket explosion** (more than 500 millions US\$ cost due to a software flaw in control software)

## Why formal verification?



- If “*Input1*”, “*Input2*”, “*Input3*” and “*Input4*” are **BOOL**, then we need to check  $2^4 = 16$  combinations
- If they are **INT** (16-bit), then  $2^{16 \cdot 4} \approx 1.8 \cdot 10^{19}$  combinations
- for large systems (many variables), such requirements **cannot** (practically) **be checked by using testing techniques**
- **Peer reviews** and **testing** can (normally) catch most of the “problems” (e.g. code bugs), but not the **corner cases**
  - **E.g. Ariane 5 rocket explosion** (more than 500 millions US\$ cost due to a software flaw in control software)

Solution: **Model checking**

## What are Formal Methods?

## What are Formal Methods?

Techniques based on **mathematics** and **formal logic (precise semantics)**

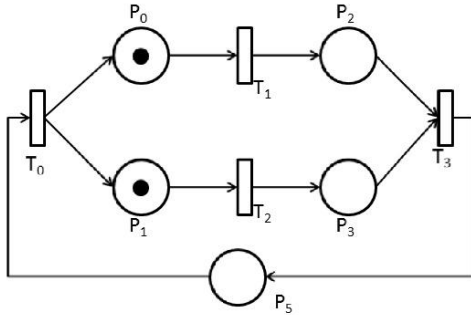
# What are Formal Methods?

Techniques based on **mathematics** and **formal logic** (precise semantics)

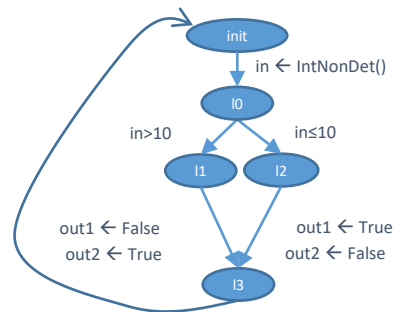
Graphical languages

Petri nets, automata, ...

*Petri net*



*Automata*



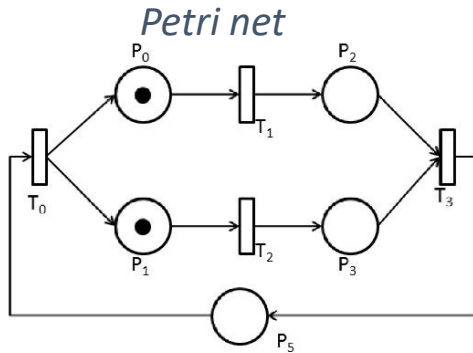


# What are Formal Methods?

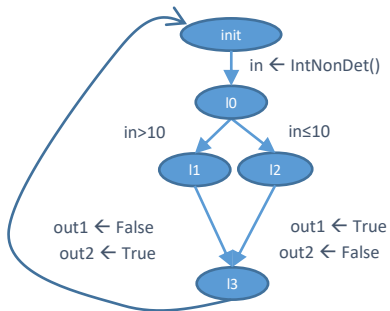
Techniques based on **mathematics** and **formal logic** (precise semantics)

Graphical languages

Petri nets, automata, ...



*Automata*



Textual languages

B-method, VDM, TLA+,...

*B-method*

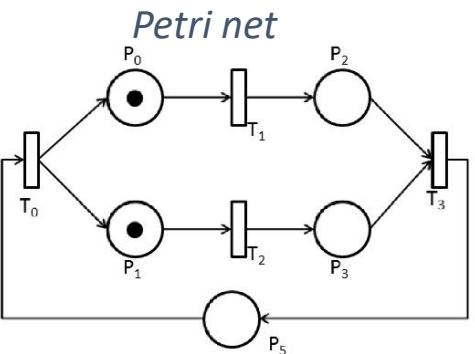
```
MACHINE
  Switch
SETS
  STATE = {closed, open}
VARIABLES
  state
INVARIANT
  state : STATE
INITIALISATION
  state := open
OPERATIONS
  toggle =
    IF state = open
    THEN
      state := closed
    ELSE
      state := open
    END ;
END
```

# What are Formal Methods?

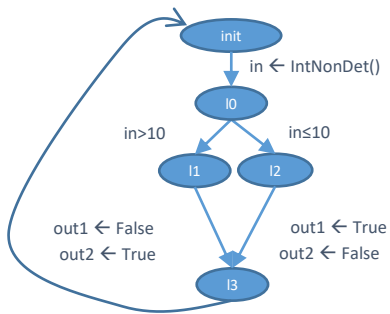
Techniques based on **mathematics** and **formal logic** (precise semantics)

Graphical languages

Petri nets, automata, ...



*Automata*



Textual languages

B-method, VDM, TLA+,...

*B-method*

```
MACHINE
  Switch
SETS
  STATE = {closed, open}
VARIABLES
  state
INVARIANT
  state : STATE
INITIALISATION
  state := open
OPERATIONS
  toggle =
    IF state = open
    THEN
      state := closed
    ELSE
      state := open
    END ;
END
```

Mathematical languages

Temporal logic,  
propositional logic, Z  
notation,...

*Temporal logic*

$AG((a \wedge b) \rightarrow c)$

*Propositional logic*

$(A \rightarrow B) \vdash (\neg B \rightarrow \neg A)$

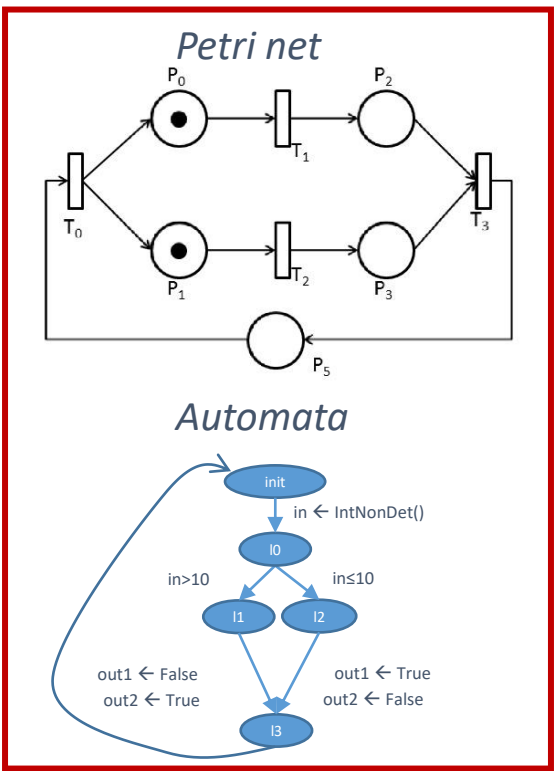
# What are Formal Methods?

Techniques based on **mathematics** and **formal logic** (precise semantics)

Graphical languages

Petri nets, automata, ...

e.g. **system model**  
(model checking)



Textual languages

B-method, VDM, TLA+, ...

*B-method*

```
MACHINE
  Switch
SETS
  STATE = {closed, open}
VARIABLES
  state
INVARIANT
  state : STATE
INITIALISATION
  state := open
OPERATIONS
  toggle =
    IF state = open
    THEN
      state := closed
    ELSE
      state := open
    END ;
END
```

Mathematical languages

Temporal logic,  
propositional logic, Z  
notation, ...

e.g. **properties**  
(model checking)

*Temporal logic*

$AG((a \wedge b) \rightarrow c)$

*Propositional logic*

$(A \rightarrow B) \vdash (\neg B \rightarrow \neg A)$

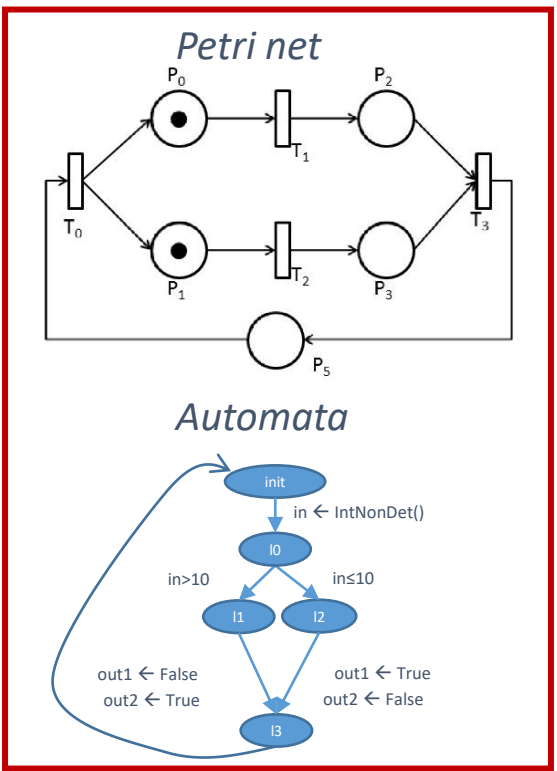
# What are Formal Methods?

Techniques based on **mathematics** and **formal logic** (precise semantics)

Graphical languages

Petri nets, automata, ...

e.g. **system model**  
(model checking)



Textual languages

B-method, VDM, TLA+, ...

*B-method*

```
MACHINE
  Switch
SETS
  STATE = {closed, open}
VARIABLES
  state
INVARIANT
  state : STATE
INITIALISATION
  state := open
OPERATIONS
  toggle =
    IF state = open
    THEN
      state := closed
    ELSE
      state := open
    END ;
END
```

Mathematical languages

Temporal logic,  
propositional logic, Z  
notation, ...

e.g. **properties**  
(model checking)

*Temporal logic*

$AG((a \wedge b) \rightarrow c)$

*Propositional logic*

$(A \rightarrow B) \vdash (\neg B \rightarrow \neg A)$

They can be used for **specification, verification, simulation, test case generation**, etc.

## Where are Formal Methods being used?

Formal specification

Formal verification



## Where are Formal Methods being used?

### Formal specification



COMPASS

Correctness, Modelling and Performance of Aerospace Systems

<http://www.compass-toolset.org>



Using **TLA+** to create a clear and concise specification, leading to a subsequent code reduction <https://cacm.acm.org/magazines/2015/4/184701-how-amazon-web-services-uses-formal-methods/fulltext>



Use of the formal specification language **VDM** to specify industrial applications

[https://www.researchgate.net/publication/2879682\\_The\\_IFAD\\_VDM-SL\\_toolbox](https://www.researchgate.net/publication/2879682_The_IFAD_VDM-SL_toolbox)



Formal Verification of Critical Aerospace Software <https://hal.archives-ouvertes.fr/hal-01184099/document>

### Formal verification



Integration of their **static analyser** INFER into their **software development** process <https://www.inf.ed.ac.uk/teaching/courses/sp/2019/lects/distefano-scaling-2019.pdf>



NASA AMES Robust Software Engineering group

<https://www.nasa.gov/isd-robust-software-engineering>

Use of the model checker SPIN to verify the model of a software

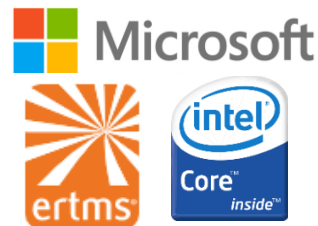
<http://spinroot.com/gerard/pdf/spin04.pdf>



Verification of **neural-network-based control** systems in non-towered airports to avoid collisions at landing

[https://www.researchgate.net/publication/356096882\\_Formal\\_Analysis\\_of\\_Neural\\_Network-Based\\_Systems\\_in\\_the\\_Aircraft\\_Domain](https://www.researchgate.net/publication/356096882_Formal_Analysis_of_Neural_Network-Based_Systems_in_the_Aircraft_Domain)

And many more ...



## Why aren't Formal Methods widely used?

Pros	Cons
<i>Unambiguity</i> (well-defined semantics)	<i>High cost</i> (time)
<i>Precision</i> (e.g. software verification)	<i>Limitation of computational models</i> (state space explosion in model checking)
...	<i>Usability</i>

## Why aren't Formal Methods widely used?

Pros	Cons
<i>Unambiguity</i> (well-defined semantics)	<i>High cost</i> (time)
<i>Precision</i> (e.g. software verification)	<i>Limitation of computational models</i> (state space explosion in model checking)
...	<i>Usability</i>

- Using formal methods is **more “expensive”** than traditional alternatives in engineering
- Real-life system models may be too large **to be handled by simulators or model checkers**
- We should **apply them when the cost of a failure is higher than the cost of using them** (tool support)

## Formal methods and the standards (e.g. functional safety)

**Formal methods and the standards (e.g. functional safety)**

**IEC 61508:** Functional safety of electrical/electronic/programmable electronic safety-related systems



# Formal methods and the standards (e.g. functional safety)

## IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems

Table A.1 – Software safety requirements specification

(See 7.2)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1a	Semi-formal methods	Table B.7	R	R	HR	HR
1b	Formal methods	B.2.2, C.2.4	---	R	R	HR
2	Forward traceability between the system safety requirements and the software safety requirements	C.2.11	R	R	HR	HR
3	Backward traceability between the safety requirements and the perceived safety needs	C.2.11	R	R	HR	HR
4	Computer-aided specification tools to support appropriate techniques/measures above	B.2.4	R	R	HR	HR

# Formal methods and the standards (e.g. functional safety)

## IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems

Table A.1 – Software safety requirements specification

(See 7.2)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1a	Semi-formal methods	Table B.7	R	R	HR	HR
1b	Formal methods	B.2.2, C.2.4	---	R	R	HR
2	Forward traceability between the system safety requirements and the software safety requirements	C.2.11	R	R	HR	HR
3	Backward traceability between the safety requirements and the perceived safety needs	C.2.11	R	R	HR	HR
4	Computer-aided specification tools to support appropriate techniques/measures above	B.2.4	R	R	HR	HR

## Formal methods and the standards (e.g. functional safety)

### IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems

**Table A.1 – Software safety requirements specification**

(See 7.2)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1a	Semi-formal methods	Table B.7	R	R	HR	HR
1b	Formal methods	B.2.2, C.2.4	---	R	R	HR
2	Forward traceability between the system safety requirements and the software safety requirements	C.2.11	R	R	HR	HR
3	Backward traceability between the safety requirements and the perceived safety needs	C.2.11	R	R	HR	HR
4	Computer-aided specification tools to support appropriate techniques/measures above	B.2.4	R	R	HR	HR

**Table A.5 – Software design and development – software module testing and integration**

(See 7.4.7 and 7.4.8)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Probabilistic testing	C.5.1	---	R	R	R
2	Dynamic analysis and testing	B.6.5 Table B.2	R	HR	HR	HR
3	Data recording and analysis	C.5.2	HR	HR	HR	HR
4	Functional and black box testing	B.5.1 B.5.2 Table B.3	HR	HR	HR	HR
5	Performance testing	Table B.6	R	R	HR	HR
6	Model based testing	C.5.27	R	R	HR	HR
7	Interface testing	C.5.3	R	R	HR	HR
8	Test management and automation tools	C.4.7	R	HR	HR	HR
9	Forward traceability between the software design specification and the module and integration test specifications	C.2.11	R	R	HR	HR
10	Formal verification	C.5.12	---	---	R	R

## Formal methods and the standards (e.g. functional safety)

### IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems

**Table A.1 – Software safety requirements specification**

(See 7.2)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1a	Semi-formal methods	Table B.7	R	R	HR	HR
1b	Formal methods	B.2.2, C.2.4	---	R	R	HR
2	Forward traceability between the system safety requirements and the software safety requirements	C.2.11	R	R	HR	HR
3	Backward traceability between the safety requirements and the perceived safety needs	C.2.11	R	R	HR	HR
4	Computer-aided specification tools to support appropriate techniques/measures above	B.2.4	R	R	HR	HR

**Table A.5 – Software design and development – software module testing and integration**

(See 7.4.7 and 7.4.8)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Probabilistic testing	C.5.1	---	R	R	R
2	Dynamic analysis and testing	B.6.5 Table B.2	R	HR	HR	HR
3	Data recording and analysis	C.5.2	HR	HR	HR	HR
4	Functional and black box testing	B.5.1 B.5.2 Table B.3	HR	HR	HR	HR
5	Performance testing	Table B.6	R	R	HR	HR
6	Model based testing	C.5.27	R	R	HR	HR
7	Interface testing	C.5.3	R	R	HR	HR
8	Test management and automation tools	C.4.7	R	HR	HR	HR
9	Forward traceability between the software design specification and the module and integration test specifications	C.2.11	R	R	HR	HR
10	Formal verification	C.5.12	---	---	R	R

## Formal methods and the standards (e.g. functional safety)

### IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems

**Table A.1 – Software safety requirements specification**

(See 7.2)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1a	Semi-formal methods	Table B.7	R	R	HR	HR
1b	Formal methods	B.2.2, C.2.4	---	R	R	HR
2	Forward traceability between the system safety requirements and the software safety requirements	C.2.11	R	R	HR	HR
3	Backward traceability between the safety requirements and the perceived safety needs	C.2.11	R	R	HR	HR
4	Computer-aided specification tools to support appropriate techniques/measures above	B.2.4	R	R	HR	HR

**Table A.5 – Software design and development – software module testing and integration**

(See 7.4.7 and 7.4.8)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Probabilistic testing	C.5.1	---	R	R	R
2	Dynamic analysis and testing	B.6.5 Table B.2	R	HR	HR	HR
3	Data recording and analysis	C.5.2	HR	HR	HR	HR
4	Functional and black box testing	B.5.1 B.5.2 Table B.3	HR	HR	HR	HR
5	Performance testing	Table B.6	R	R	HR	HR
6	Model based testing	C.5.27	R	R	HR	HR
7	Interface testing	C.5.3	R	R	HR	HR
8	Test management and automation tools	C.4.7	R	HR	HR	HR
9	Forward traceability between the software design specification and the module and integration test specifications	C.2.11	R	R	HR	HR
10	Formal verification	C.5.12	---	---	R	R

### IEC 61511: Functional safety – Safety instrumented systems for the process industry sector

## Formal methods and the standards (e.g. functional safety)

### IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems

**Table A.1 – Software safety requirements specification**

(See 7.2)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1a	Semi-formal methods	Table B.7	R	R	HR	HR
1b	Formal methods	B.2.2, C.2.4	---	R	R	HR
2	Forward traceability between the system safety requirements and the software safety requirements	C.2.11	R	R	HR	HR
3	Backward traceability between the safety requirements and the perceived safety needs	C.2.11	R	R	HR	HR
4	Computer-aided specification tools to support appropriate techniques/measures above	B.2.4	R	R	HR	HR

**Table A.5 – Software design and development – software module testing and integration**

(See 7.4.7 and 7.4.8)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Probabilistic testing	C.5.1	---	R	R	R
2	Dynamic analysis and testing	B.6.5 Table B.2	R	HR	HR	HR
3	Data recording and analysis	C.5.2	HR	HR	HR	HR
4	Functional and black box testing	B.5.1 B.5.2 Table B.3	HR	HR	HR	HR
5	Performance testing	Table B.6	R	R	HR	HR
6	Model based testing	C.5.27	R	R	HR	HR
7	Interface testing	C.5.3	R	R	HR	HR
8	Test management and automation tools	C.4.7	R	HR	HR	HR
9	Forward traceability between the software design specification and the module and integration test specifications	C.2.11	R	R	HR	HR
10	Formal verification	C.5.12	---	---	R	R

### IEC 61511: Functional safety – Safety instrumented systems for the process industry sector

- several references to model checking. For example from IEC 61511-2:2016 Annex B:

*“... specification should be implemented in the graphical language of the **model checking** workbench environment...”*



## Formal methods and the standards (e.g. functional safety)

### IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems

**Table A.1 – Software safety requirements specification**

(See 7.2)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1a	Semi-formal methods	Table B.7	R	R	HR	HR
1b	Formal methods	B.2.2, C.2.4	---	R	R	HR
2	Forward traceability between the system safety requirements and the software safety requirements	C.2.11	R	R	HR	HR
3	Backward traceability between the safety requirements and the perceived safety needs	C.2.11	R	R	HR	HR
4	Computer-aided specification tools to support appropriate techniques/measures above	B.2.4	R	R	HR	HR

**Table A.5 – Software design and development – software module testing and integration**

(See 7.4.7 and 7.4.8)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Probabilistic testing	C.5.1	---	R	R	R
2	Dynamic analysis and testing	B.6.5 Table B.2	R	HR	HR	HR
3	Data recording and analysis	C.5.2	HR	HR	HR	HR
4	Functional and black box testing	B.5.1 B.5.2 Table B.3	HR	HR	HR	HR
5	Performance testing	Table B.6	R	R	HR	HR
6	Model based testing	C.5.27	R	R	HR	HR
7	Interface testing	C.5.3	R	R	HR	HR
8	Test management and automation tools	C.4.7	R	HR	HR	HR
9	Forward traceability between the software design specification and the module and integration test specifications	C.2.11	R	R	HR	HR
10	Formal verification	C.5.12	---	---	R	R

### IEC 61511: Functional safety – Safety instrumented systems for the process industry sector

- several references to model checking. For example from IEC 61511-2:2016 Annex B:

*“... specification should be implemented in the graphical language of the **model checking** workbench environment...”*

## Introduction to model checking (for PLC programs)

## Introduction to model checking (for PLC programs)

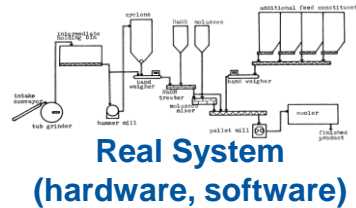
Given a **global model** of the system and a **formal property**, the **model checking algorithm checks exhaustively** that the model meets the property

Clarke and Emerson (1982) and Queille and Sifakis (1982)

# Introduction to model checking (for PLC programs)

Given a **global model** of the system and a **formal property**, the **model checking algorithm checks exhaustively** that the model meets the property

Clarke and Emerson (1982) and Queille and Sifakis (1982)



Specifications

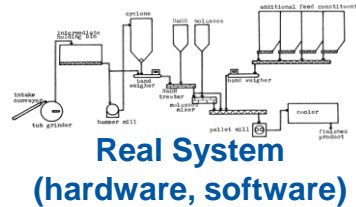


## Introduction to model checking (for PLC programs)

Given a **global model** of the system and a **formal property**, the **model checking algorithm** **checks exhaustively** that the model meets the property

Clarke and Emerson (1982) and Queille and Sifakis (1982)

**Formal  
model**



**Specifications**

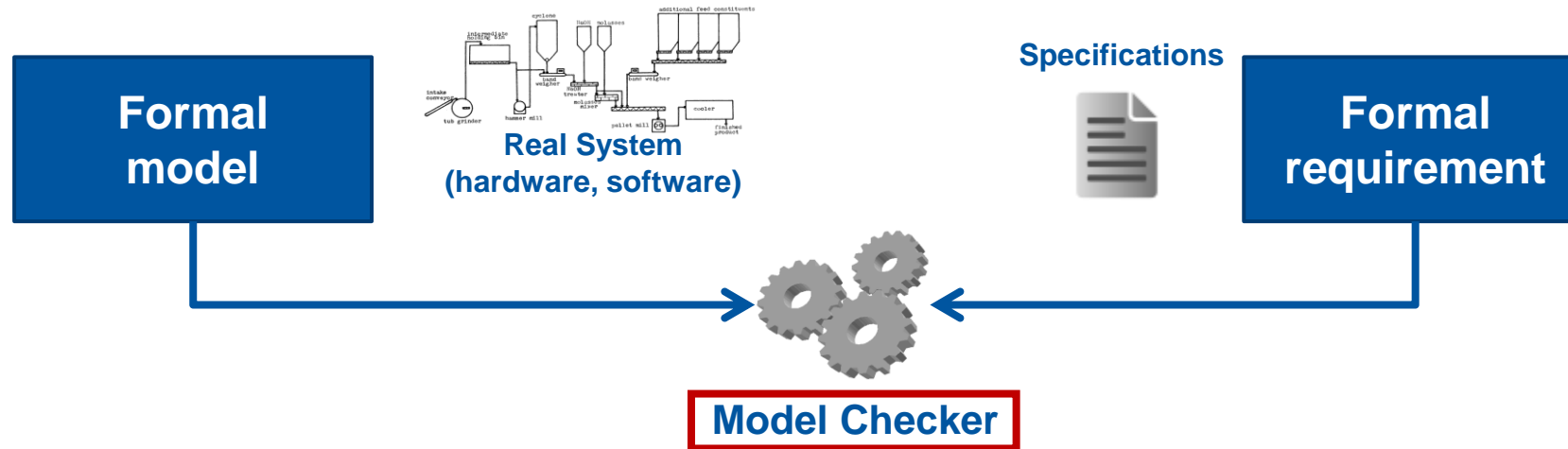


**Formal  
requirement**

## Introduction to model checking (for PLC programs)

Given a **global model** of the system and a **formal property**, the **model checking algorithm** **checks exhaustively** that the model meets the property

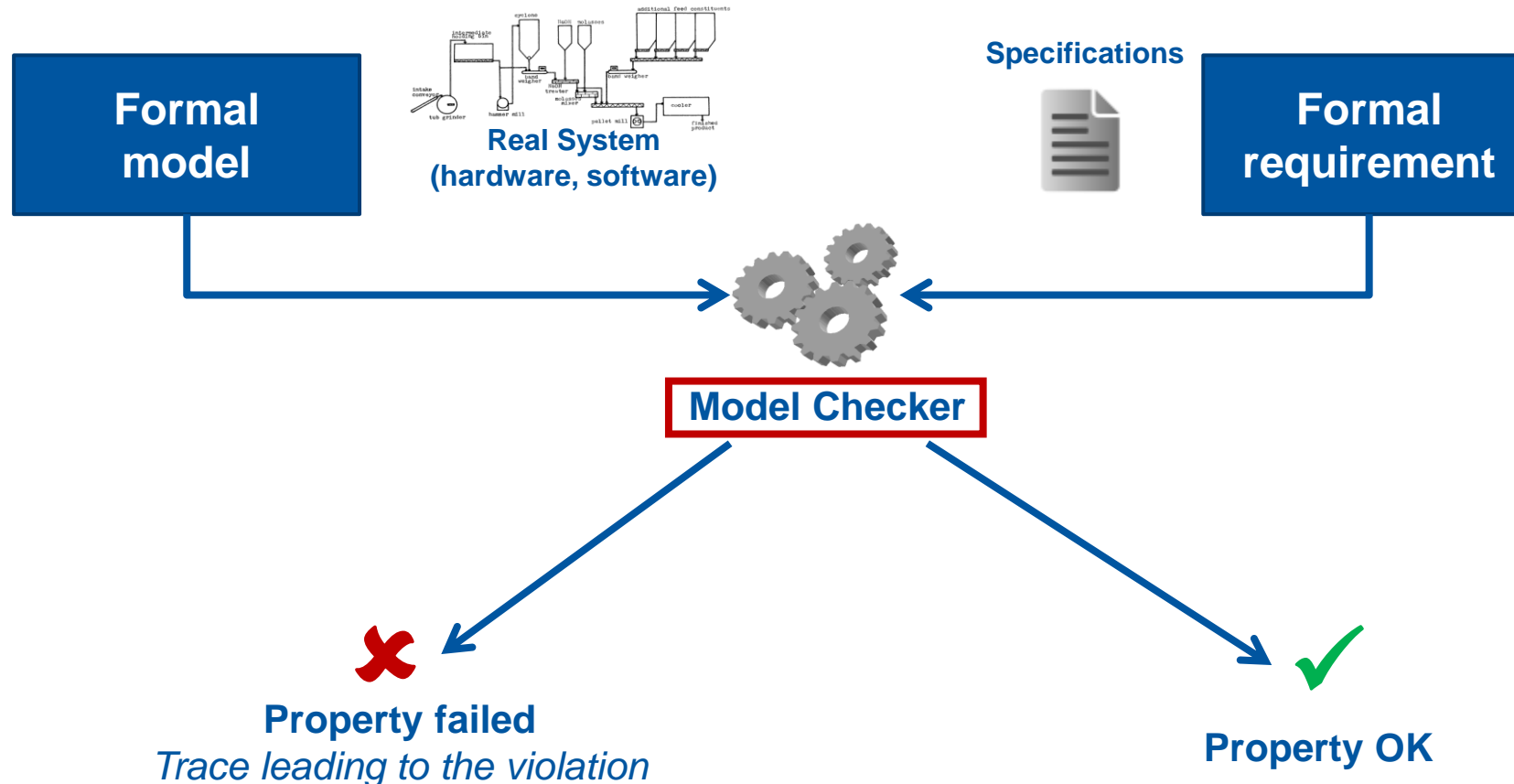
Clarke and Emerson (1982) and Queille and Sifakis (1982)



## Introduction to model checking (for PLC programs)

Given a **global model** of the system and a **formal property**, the **model checking algorithm** **checks exhaustively** that the model meets the property

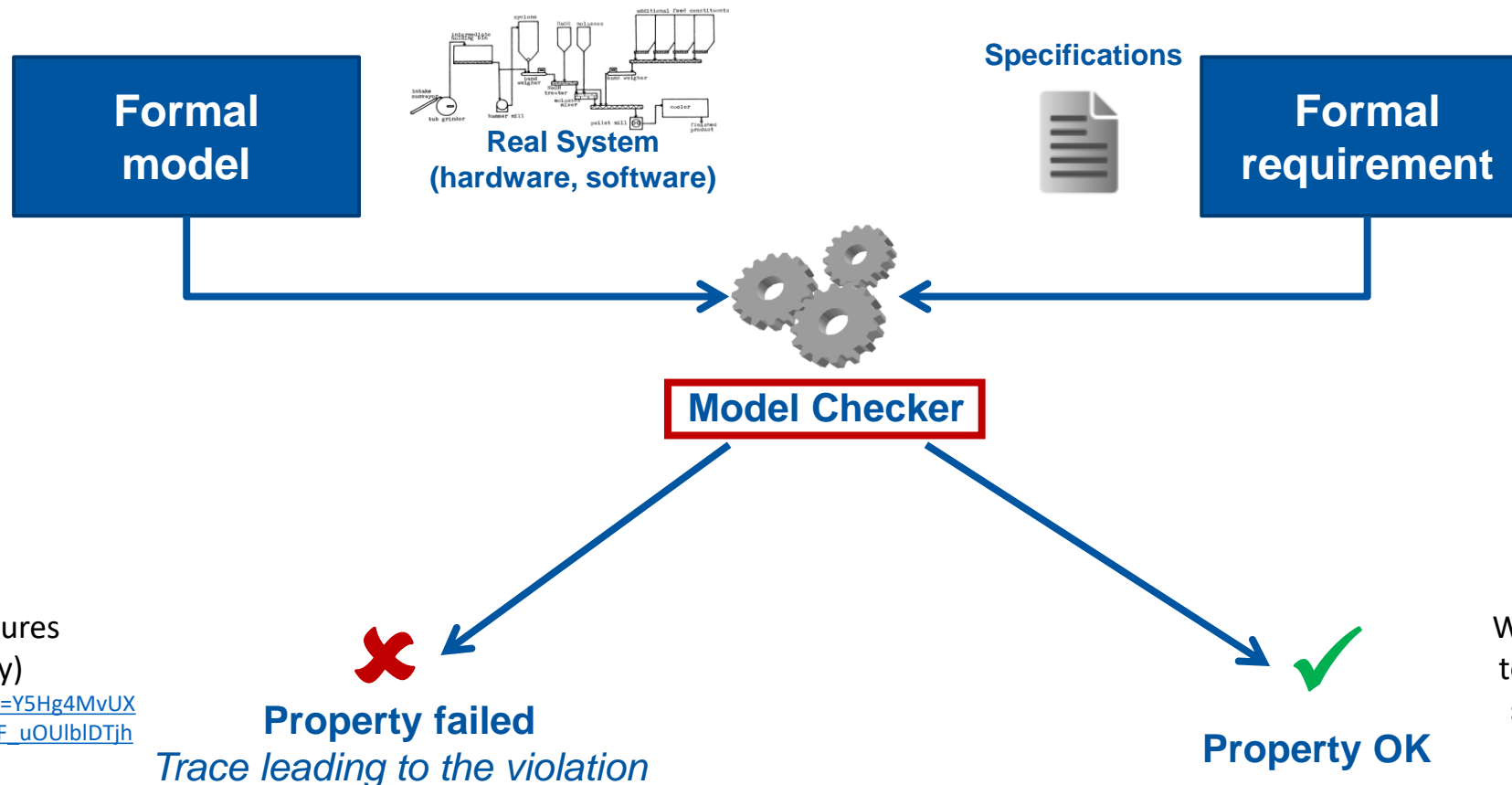
Clarke and Emerson (1982) and Queille and Sifakis (1982)



## Introduction to model checking (for PLC programs)

Given a **global model** of the system and a **formal property**, the **model checking algorithm** **checks exhaustively** that the model meets the property

Clarke and Emerson (1982) and Queille and Sifakis (1982)



Model Checking lectures  
(Aachen university)

[https://www.youtube.com/watch?v=Y5Hg4MvUXc4&list=PLwabKnOFhE38C0o6z\\_bhIF\\_uOUIbIDTjh](https://www.youtube.com/watch?v=Y5Hg4MvUXc4&list=PLwabKnOFhE38C0o6z_bhIF_uOUIbIDTjh)

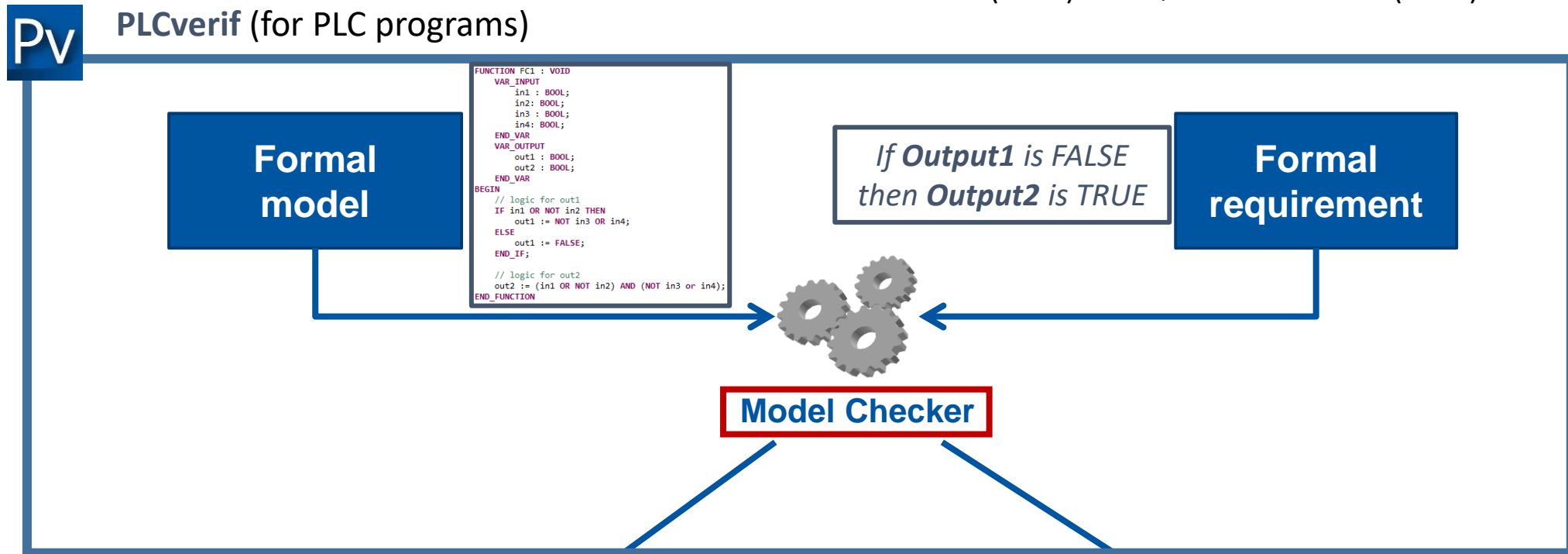
Was born for **hardware design**,  
today it is used extensively for  
**software verification** as well



# Introduction to model checking (for PLC programs)

Given a **global model** of the system and a **formal property**, the **model checking algorithm checks exhaustively** that the model meets the property

Clarke and Emerson (1982) and Queille and Sifakis (1982)



Model Checking lectures  
(Aachen university)

[https://www.youtube.com/watch?v=Y5Hg4MvUXc4&list=PLwabKnOFhE38C0o6z\\_bhIF\\_uOUIbIDTjh](https://www.youtube.com/watch?v=Y5Hg4MvUXc4&list=PLwabKnOFhE38C0o6z_bhIF_uOUIbIDTjh)

**Property failed**  
*Trace leading to the violation*

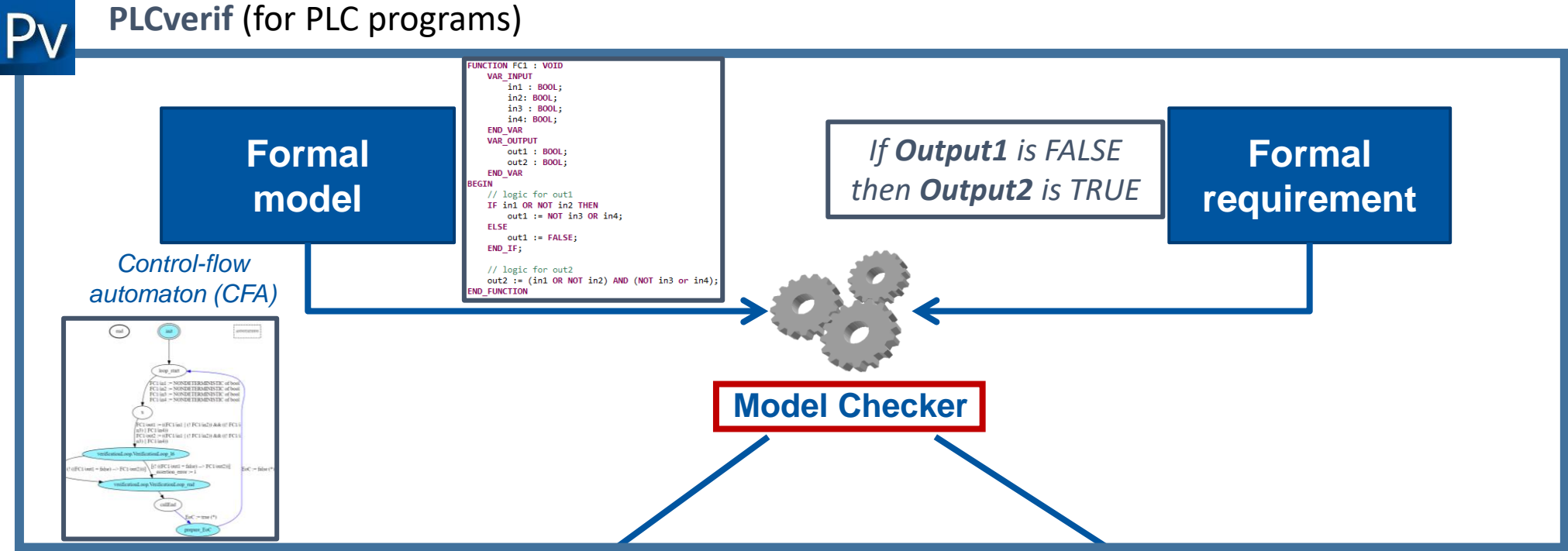
**Property OK**

Was born for **hardware design**,  
today it is used extensively for  
**software verification** as well

# Introduction to model checking (for PLC programs)

Given a **global model** of the system and a **formal property**, the **model checking algorithm checks exhaustively** that the model meets the property

Clarke and Emerson (1982) and Queille and Sifakis (1982)



Model Checking lectures  
(Aachen university)

[https://www.youtube.com/watch?v=Y5Hg4MvUXc4&list=PLwabKnOfhE38C0o6z\\_bhIF\\_uOUIbIDTjh](https://www.youtube.com/watch?v=Y5Hg4MvUXc4&list=PLwabKnOfhE38C0o6z_bhIF_uOUIbIDTjh)

**Property failed**  
*Trace leading to the violation*

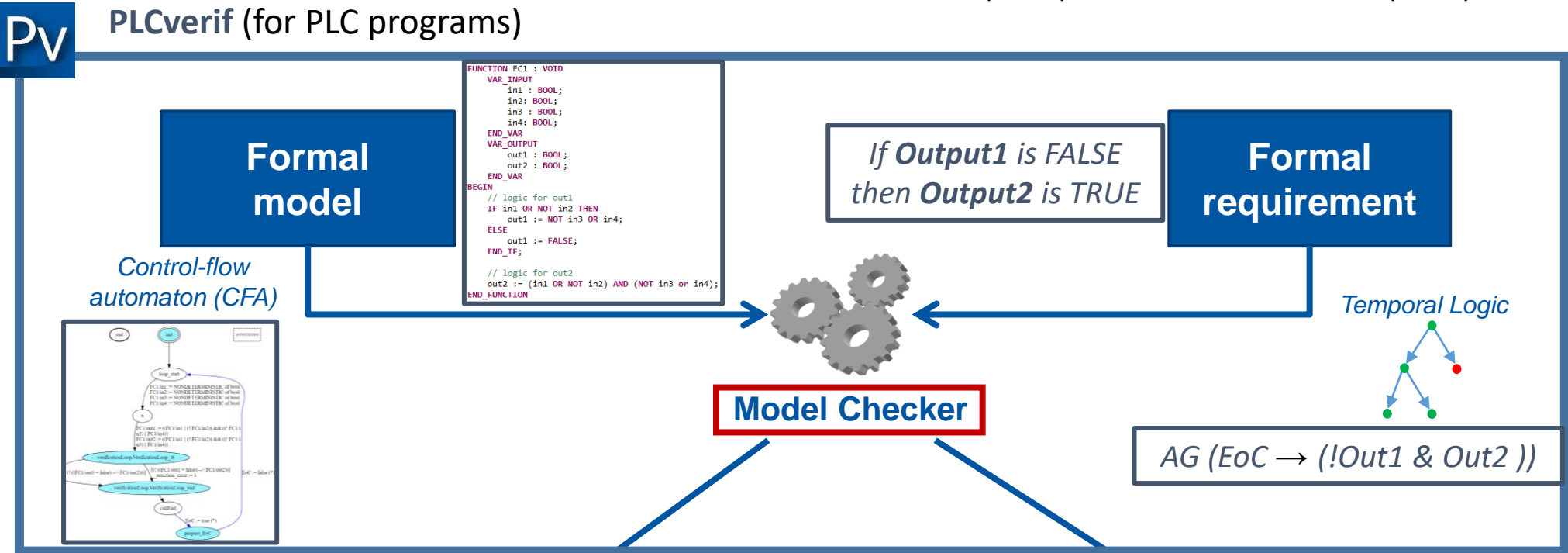
**Property OK**

Was born for **hardware design**,  
today it is used extensively for  
**software verification** as well

# Introduction to model checking (for PLC programs)

Given a **global model** of the system and a **formal property**, the **model checking algorithm checks exhaustively** that the model meets the property

Clarke and Emerson (1982) and Queille and Sifakis (1982)



Model Checking lectures  
(Aachen university)  
[https://www.youtube.com/watch?v=Y5Hg4MvUXc4&list=PLwabKnOFhE38C0o6z\\_bhIF\\_uOUIbIDTjh](https://www.youtube.com/watch?v=Y5Hg4MvUXc4&list=PLwabKnOFhE38C0o6z_bhIF_uOUIbIDTjh)

 **Property failed**  
*Trace leading to the violation*

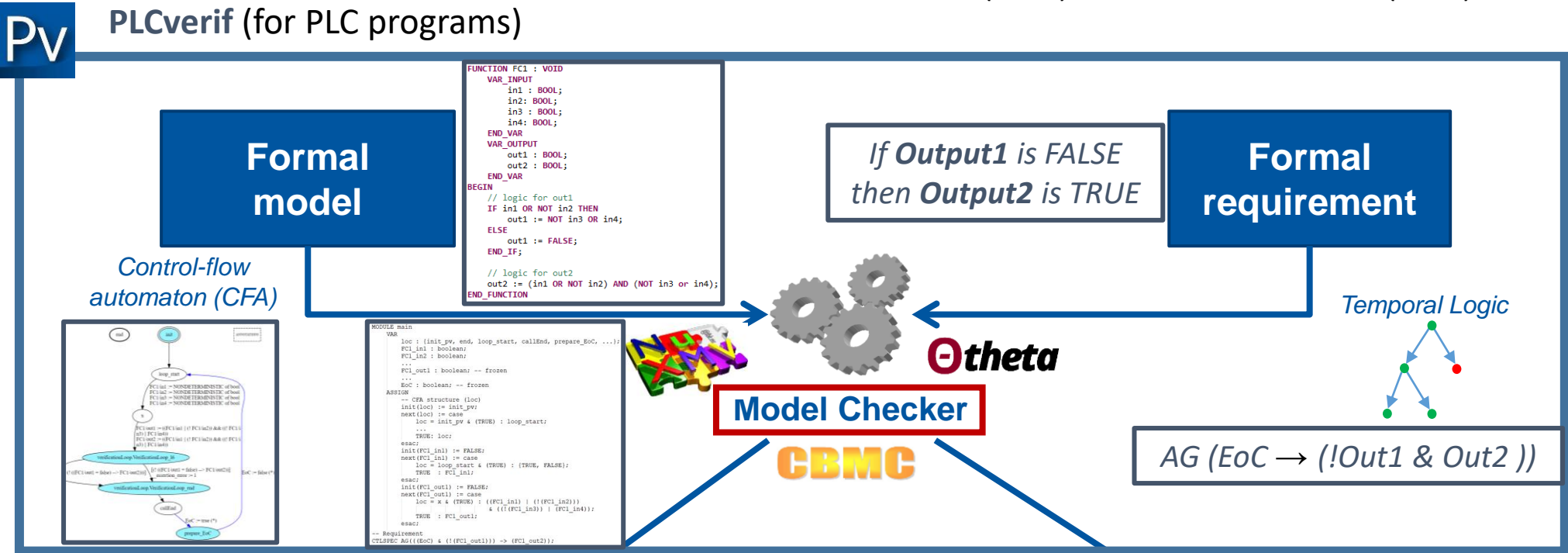
 **Property OK**

Was born for **hardware design**,  
today it is used extensively for  
**software verification** as well

# Introduction to model checking (for PLC programs)

Given a **global model** of the system and a **formal property**, the **model checking algorithm checks exhaustively** that the model meets the property

Clarke and Emerson (1982) and Queille and Sifakis (1982)

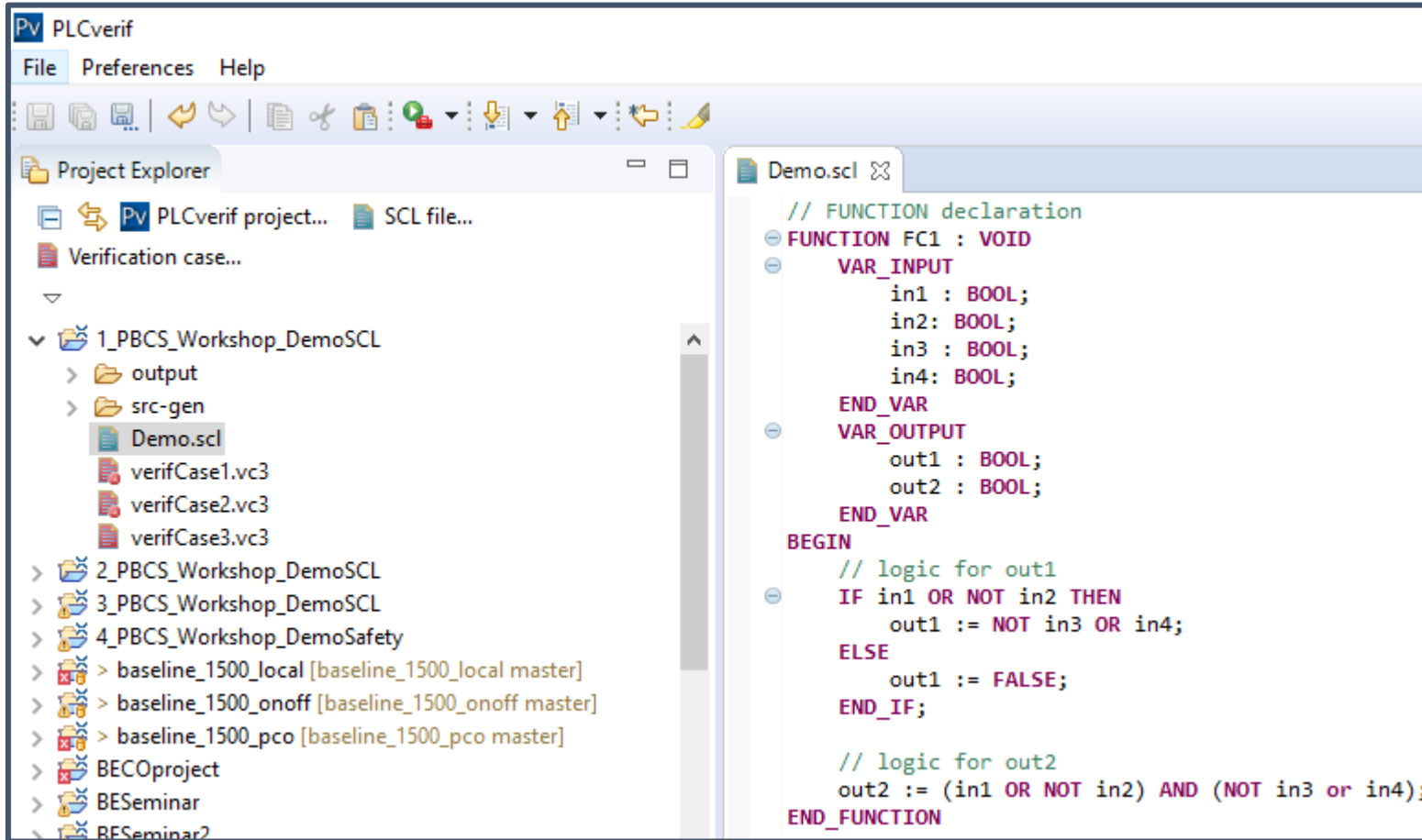


Model Checking lectures  
(Aachen university)  
[https://www.youtube.com/watch?v=Y5Hg4MvUXc4&list=PLwabKnOFhE38C0o6z\\_bhlf\\_uOUIbIDTjh](https://www.youtube.com/watch?v=Y5Hg4MvUXc4&list=PLwabKnOFhE38C0o6z_bhlf_uOUIbIDTjh)

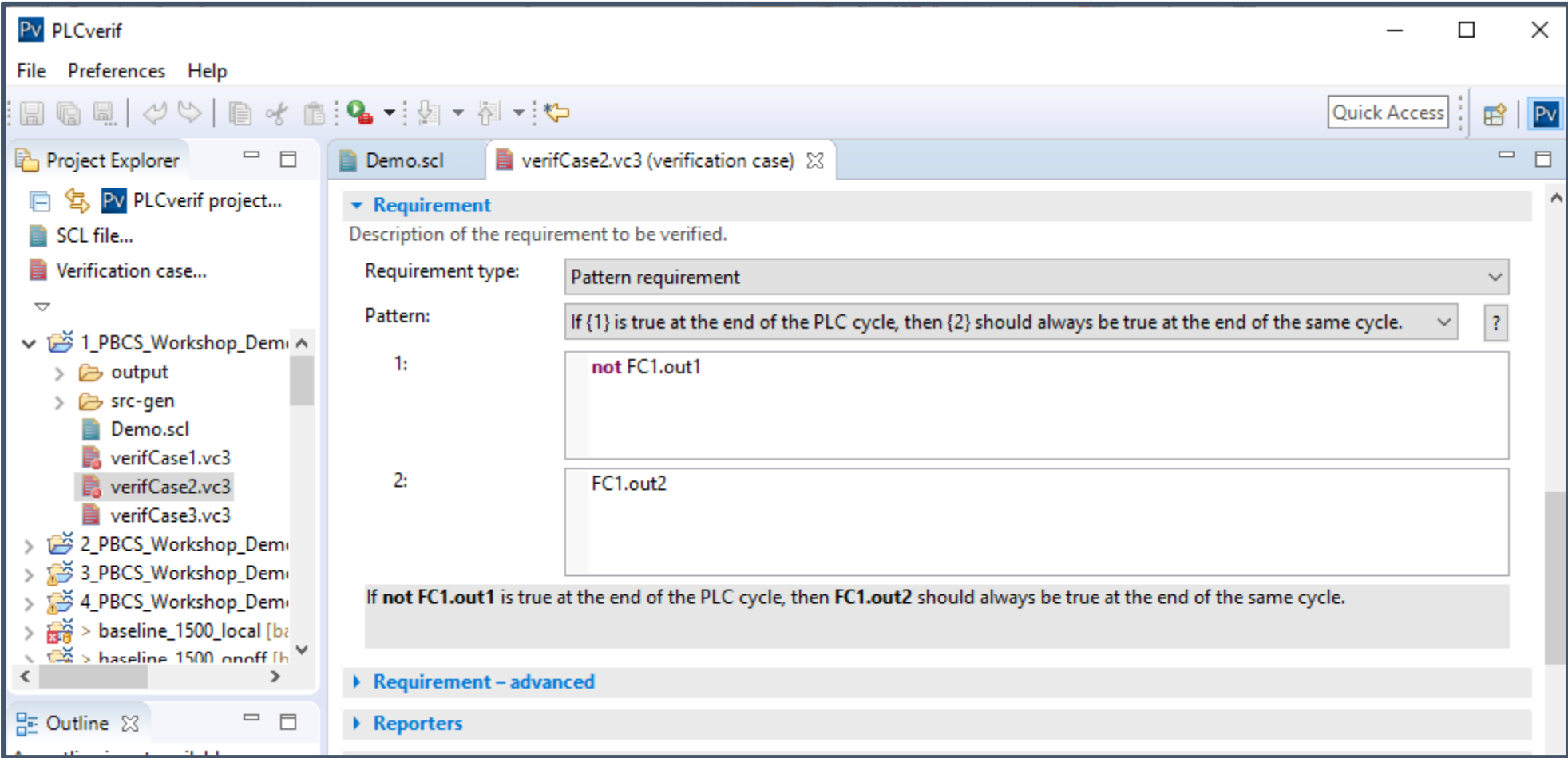
Was born for **hardware design**,  
today it is used extensively for  
**software verification** as well

**PLCverif (for users)**

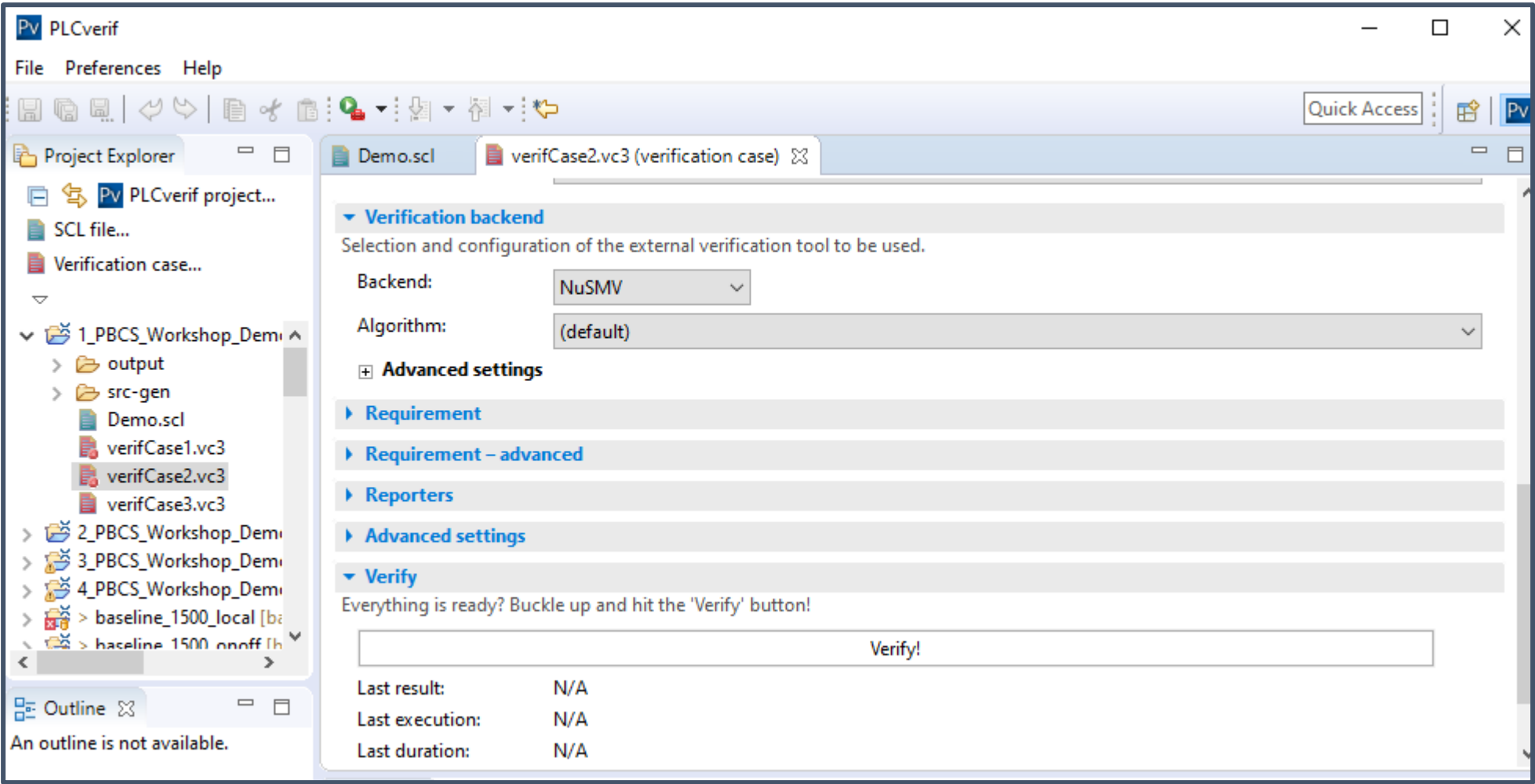
## PLCverif (for users)



# PLCverif (for users)

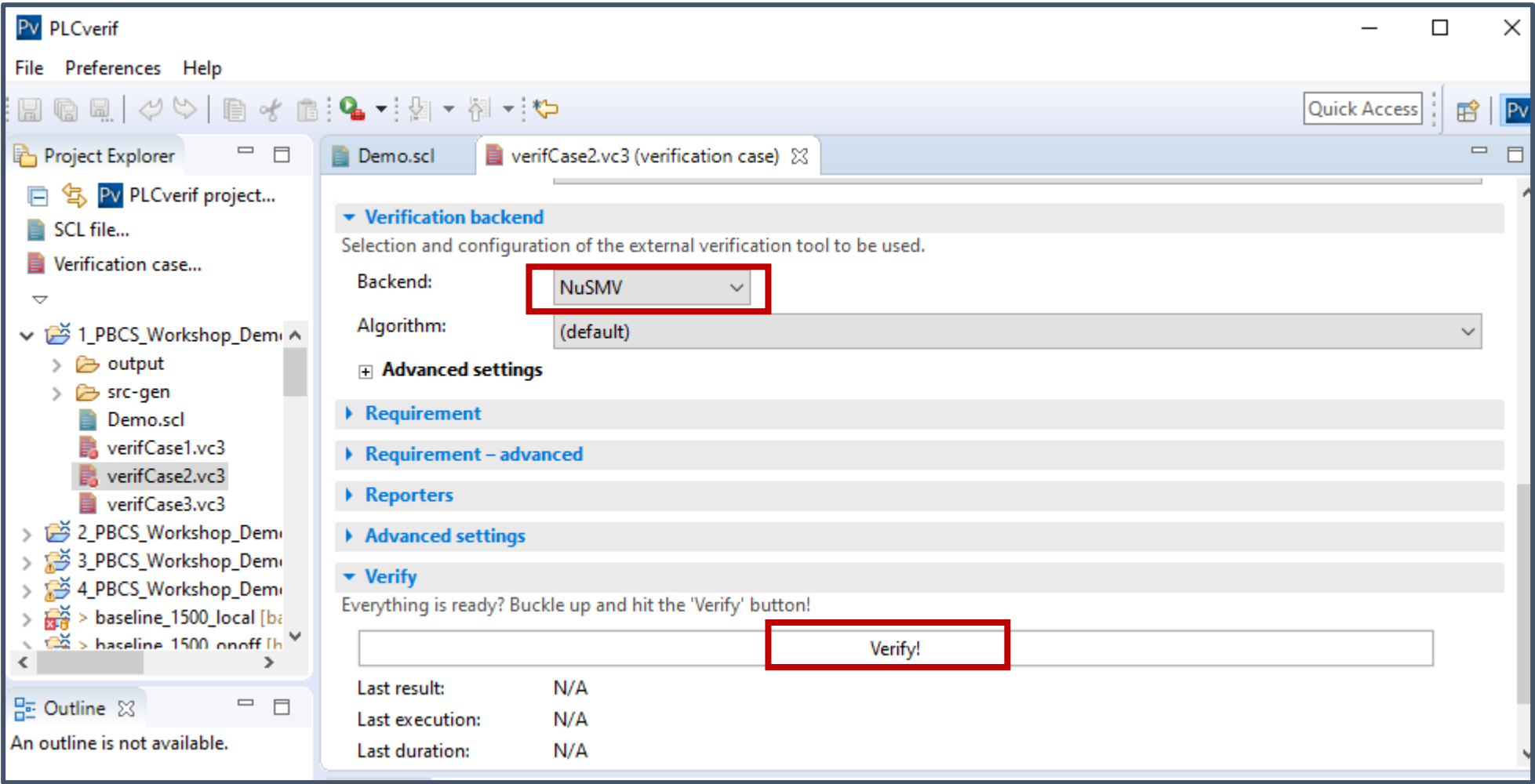


PLCverif (for users)





PLCverif (for users)



# PLCverif (for users)

PLCverif

File Preferences Help

Project Explorer

PLCverif project...

SCL file...

Verification case...

1\_PBCS\_Workshop\_DemoSCL

output

src-gen

Demo.scl

verifCase1.vc3

verifCase2.vc3

verifCase3.vc3

2\_PBCS\_Workshop\_DemoSCL

3\_PBCS\_Workshop\_DemoSCL

4\_PBCS\_Workshop\_DemoSafety

> baseline\_1500\_local [baseline\_1500\_local master]

> baseline\_1500\_onoff [baseline\_1500\_onoff master]

> baseline\_1500\_pco [baseline\_1500\_pco master]

BECOpject

BEseminar

BEseminar2

> demo-unicos [demo-unicos master]

> demo1 [demo1 master]

DemoProject

> demoproject\_plcverif [demoproject\_plcverif master]

DemoSIF1

DemoSummerStudent

DemoTheta

ESO\_Program

ESO\_Program2

ESO\_Pnriect

Outline

An outline is not available.

Demo.scl verifCase2.vc3 (verification case) verifCase2 verification report

file:///C:/dev/PLCverif/workspace/1\_PBCS\_Workshop\_DemoSCL/output/verifCase2.report.html

## PLCverif — Verification report

Generated on 2021-12-06 11:50:27 | PLCverif v3.0 | (C) CERN BE-ICS-AP | [Show/hide expert details](#)

ID:	verifCase2
Name:	
Description:	
Source file(s):	<ul style="list-style-type: none"><li>C:\dev\PLCverif\workspace\1_PBCS_Workshop_DemoSCL\Demo.scl</li><li>C:\dev\PLCverif\workspace\builtin_Siemens S7-300\builtin.scl</li></ul>
Requirement:	If not FC1.out1 is true at the end of the PLC cycle, then FC1.out2 should always be true at the end of the same cycle.
Result:	Violated
Verification backend:	NusmvBackend (nuxmv-Classic-dynamic-df)
Total run time:	449 ms
Backend run time:	346 ms

### Counterexample

	Variable	End of Cycle 1
INPUT BOOL	FC1.in1	true
INPUT BOOL	FC1.in2	false
INPUT BOOL	FC1.in3	true
INPUT BOOL	FC1.in4	false
OUTPUT BOOL	FC1.out1	false
OUTPUT BOOL	FC1.out2	false

# PLCverif (for users)

PLCverif

File Preferences Help

Project Explorer

PLCverif project...

SCL file...

Verification case...

1\_PBCS\_Workshop\_DemoSCL

output

src-gen

Demo.scl

verifCase1.vc3

verifCase2.vc3

verifCase3.vc3

2\_PBCS\_Workshop\_DemoSCL

3\_PBCS\_Workshop\_DemoSCL

4\_PBCS\_Workshop\_DemoSafety

> baseline\_1500\_local [baseline\_1500\_local master]

> baseline\_1500\_onoff [baseline\_1500\_onoff master]

> baseline\_1500\_pco [baseline\_1500\_pco master]

BECOpject

BEseminar

BEseminar2

> demo-unicos [demo-unicos master]

> demo1 [demo1 master]

DemoProject

> demoproject\_plcverif [demoproject\_plcverif master]

DemoSIF1

DemoSummerStudent

DemoTheta

ESO\_Program

ESO\_Program2

ESO\_Pnriect

Outline

An outline is not available.

Demo.scl

verifCase2.vc3 (verification case)

verifCase2 verification report

file:///C:/dev/PLCverif/workspace/1\_PBCS\_Workshop\_DemoSCL/output/verifCase2.report.html

## PLCverif — Verification report

Generated on 2021-12-06 11:50:27 | PLCverif v3.0 | (C) CERN BE-ICS-AP | [Show/hide expert details](#)

ID:	verifCase2
Name:	
Description:	
Source file(s):	<ul style="list-style-type: none"><li>C:\dev\PLCverif\workspace\1_PBCS_Workshop_DemoSCL\Demo.scl</li><li>C:\dev\PLCverif\workspace\builtin_Siemens S7-300\builtin.scl</li></ul>
Requirement:	If not FC1.out1 is true at the end of the PLC cycle, then FC1.out2 should always be true at the end of the same cycle.
Result:	Violated
Verification backend:	NusmvBackend (nuxmv-Classic-dynamic-df)
Total run time:	449 ms
Backend run time:	346 ms

### Counterexample

	Variable	End of Cycle 1
INPUT BOOL	FC1.in1	true
INPUT BOOL	FC1.in2	false
INPUT BOOL	FC1.in3	true
INPUT BOOL	FC1.in4	false
OUTPUT BOOL	FC1.out1	false
OUTPUT BOOL	FC1.out2	false

PLCverif references: <https://gitlab.com/plcverif-oss> and [www.cern.ch/plcverif](http://www.cern.ch/plcverif)

The **complexity** of using formal methods **is hidden** by the PLCverif

**Some references**

## Some references

### Real case studies



B. Fernández et al. **“Applying model checking to industrial-sized PLC programs”**. In *IEEE Transactions on Industrial Informatics*  
<https://ieeexplore.ieee.org/document/7295624>



B. Fernandez et al. **“Applying model checking to critical PLC applications : An ITER case study”** in *Proc. of the 17<sup>th</sup> ICALEPCS*  
<https://cds.cern.ch/record/2305319/files/thpha161.pdf>




B. Fernandez et al. **“Applying model checking to highly-configurable safety critical software: The SPS-PPS PLC program”** in *Proc. of the 18<sup>th</sup> ICALEPCS*  
<https://cds.cern.ch/record/2809709/files/document.pdf>




B. Fernandez et al. **“Cause-and-Effect Matrix specifications for safety critical systems at CERN”** in *Proc. of the 17<sup>th</sup> ICALEPCS*  
<https://accelconf.web.cern.ch/icalepcs2019/papers/mopha041.pdf>

# Some references


## Real case studies




B. Fernández et al. **“Applying model checking to industrial-sized PLC programs”**. In IEEE Transactions on Industrial Informatics <https://ieeexplore.ieee.org/document/7295624>



B. Fernandez et al. **“Applying model checking to critical PLC applications : An ITER case study”** in Proc. of the 17<sup>th</sup> ICALEPCS <https://cds.cern.ch/record/2305319/files/thpha161.pdf>




B. Fernandez et al. **“Applying model checking to highly-configurable safety critical software: The SPS-PPS PLC program”** in Proc. of the 18<sup>th</sup> ICALEPCS <https://cds.cern.ch/record/2809709/files/document.pdf>




B. Fernandez et al. **“Cause-and-Effect Matrix specifications for safety critical systems at CERN”** in Proc. of the 17<sup>th</sup> ICALEPCS <https://accelconf.web.cern.ch/icalepcs2019/papers/mopha041.pdf>


## Research activities




Ignacio D. Lopez-Miguel et al. **“Simplification of numeric variables for PLC model checking”**. In Proc. of the MEMOCODE '21 <https://dl.acm.org/doi/abs/10.1145/3487212.3487334>




Milán Mondok. **“Evaluating compositional verification options for PLCverif”**. In CERN internal note [https://cds.cern.ch/record/2780057/files/compositional\\_verification.pdf](https://cds.cern.ch/record/2780057/files/compositional_verification.pdf)




B. Fernández et al. **“Modelling and formal verification of timing aspects in large PLC programs”**. In Proc. of IFAC World Congress'14 <http://cds.cern.ch/record/1956687/files/CERN-ACC-2014-0226.pdf>



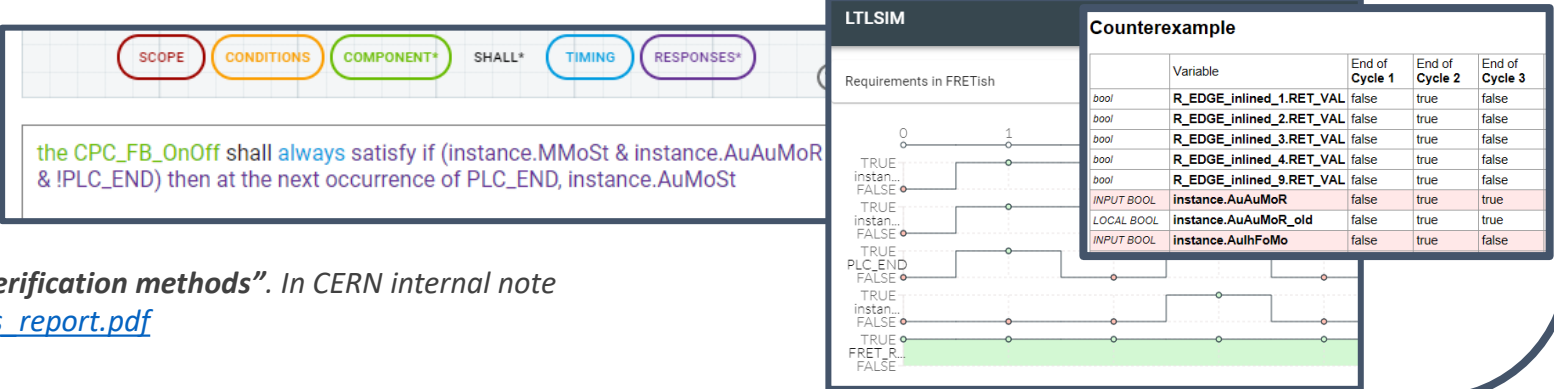
D. Darvas et al. **“A formal specification method for PLC-based applications”** in Proc. of the 15<sup>th</sup> ICALEPCS <https://accelconf.web.cern.ch/ICALEPCS2015/papers/wepqf091.pdf>



Zsófia Ádám et al. **“From Natural Language Requirements to the Verification of Programmable Logic Controllers: Integrating FRET into PLCverif”**. In NASA Formal Methods Symposium [https://link.springer.com/chapter/10.1007/978-3-031-33170-1\\_21](https://link.springer.com/chapter/10.1007/978-3-031-33170-1_21)



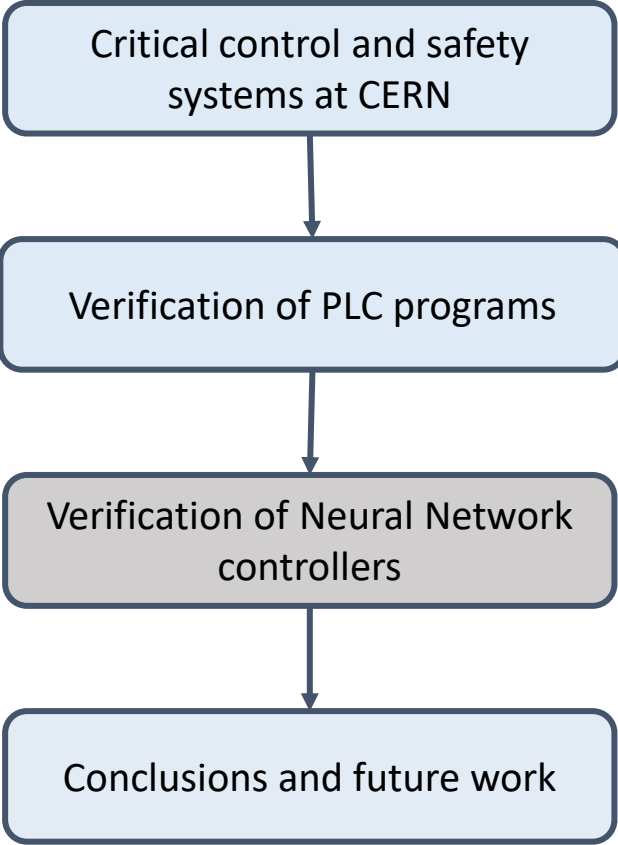
Mihály Dobos-Kovács. **“Counterexample analysis of formal verification methods”**. In CERN internal note [https://cds.cern.ch/record/2779411/files/MihalyDobosKovacs\\_report.pdf](https://cds.cern.ch/record/2779411/files/MihalyDobosKovacs_report.pdf)



The screenshot displays the FRETish tool interface. At the top, there are tabs for SCOPE, CONDITIONS, COMPONENT\*, SHALL\*, TIMING, and RESPONSES\*. The main area shows a requirement: "the CPC\_FB\_OnOff shall always satisfy if (instance.MMoSt & instance.AuAuMoR & !PLC\_END) then at the next occurrence of PLC\_END, instance.AuMoSt". Below this, there is a section titled "Counterexample" which includes a table of variables and their values across different cycles.

Variable	End of Cycle 1	End of Cycle 2	End of Cycle 3
bool R_EDGE_inlined_1.RET_VAL	false	true	false
bool R_EDGE_inlined_2.RET_VAL	false	true	false
bool R_EDGE_inlined_3.RET_VAL	false	true	false
bool R_EDGE_inlined_4.RET_VAL	false	true	false
bool R_EDGE_inlined_9.RET_VAL	false	true	false
INPUT BOOL instance.AuAuMoR	false	true	true
LOCAL BOOL instance.AuAuMoR_old	false	true	true
INPUT BOOL instance.AuHfMo	false	true	false

# Roadmap



First steps

## **Context – CERN (European Organization for Nuclear Research)**

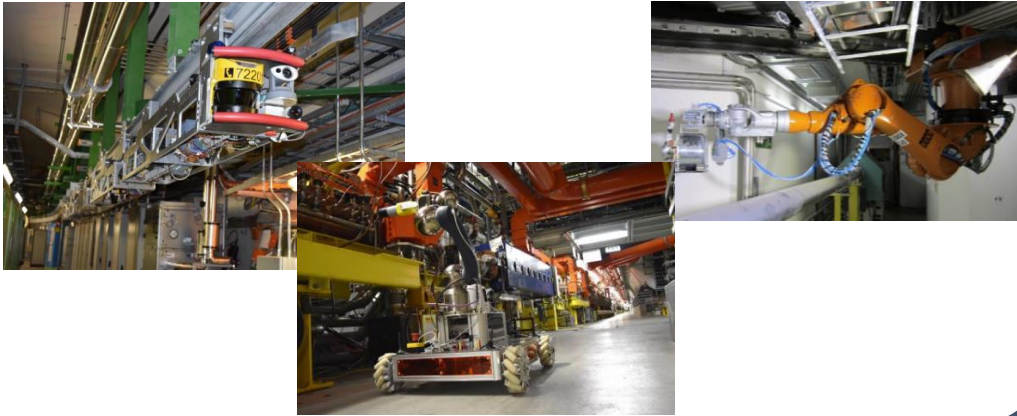
- Many applications of machine learning at CERN



## Context – CERN (European Organization for Nuclear Research)

- Many applications of machine learning at CERN

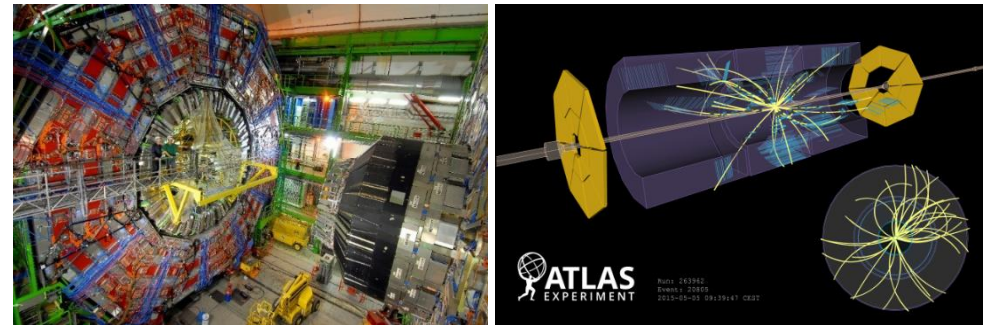
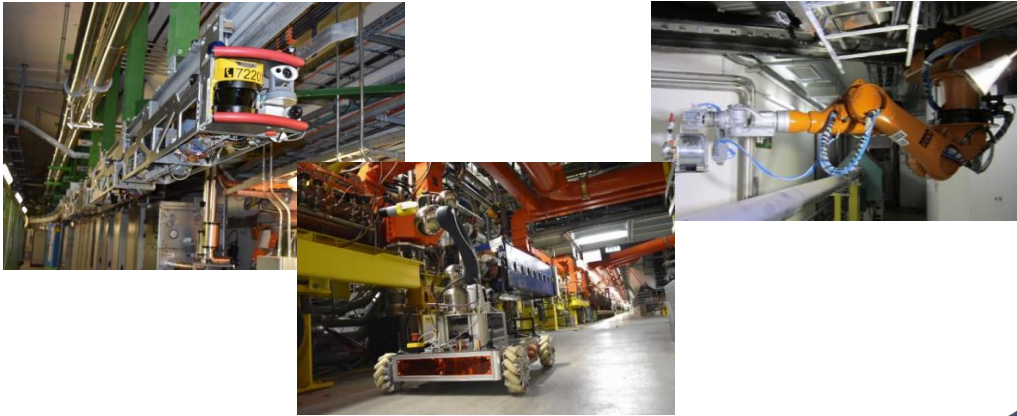
Robotics



## Context – CERN (European Organization for Nuclear Research)

- Many applications of machine learning at CERN

Robotics

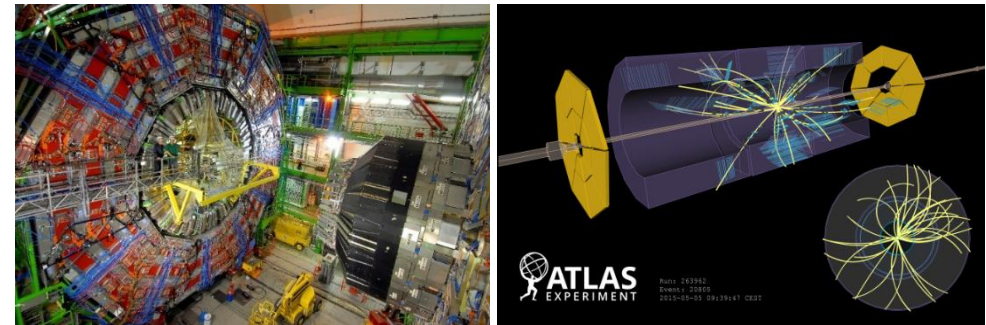
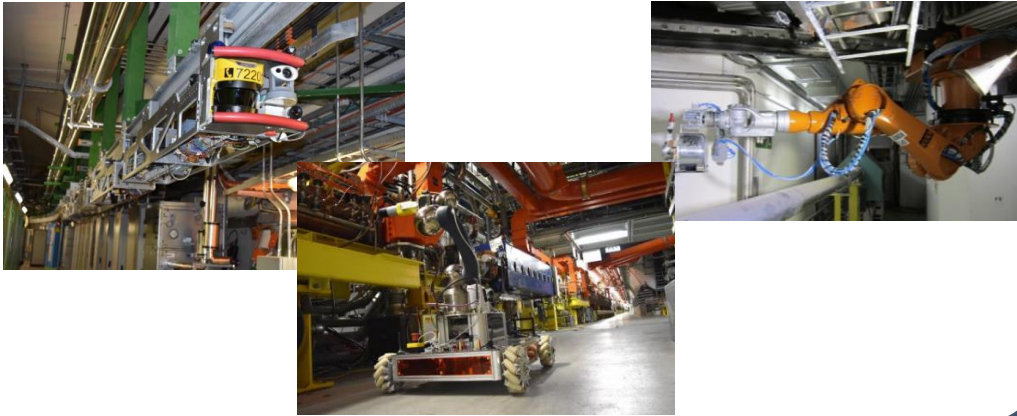


Data analysis of  
the physics  
experiments

## Context – CERN (European Organization for Nuclear Research)

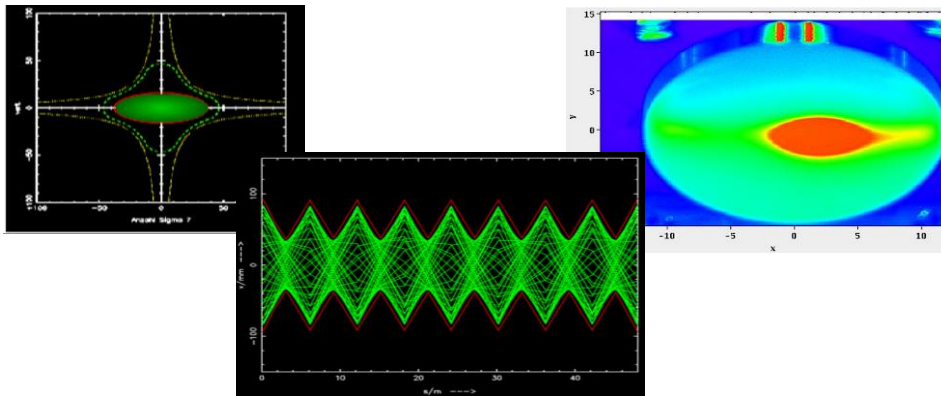
- Many applications of machine learning at CERN

Robotics



Data analysis of  
the physics  
experiments

Particle  
accelerators  
Operation and  
beam  
optimization

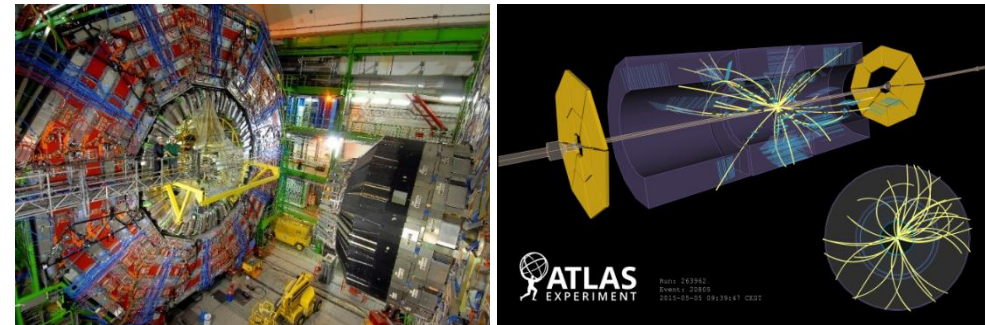
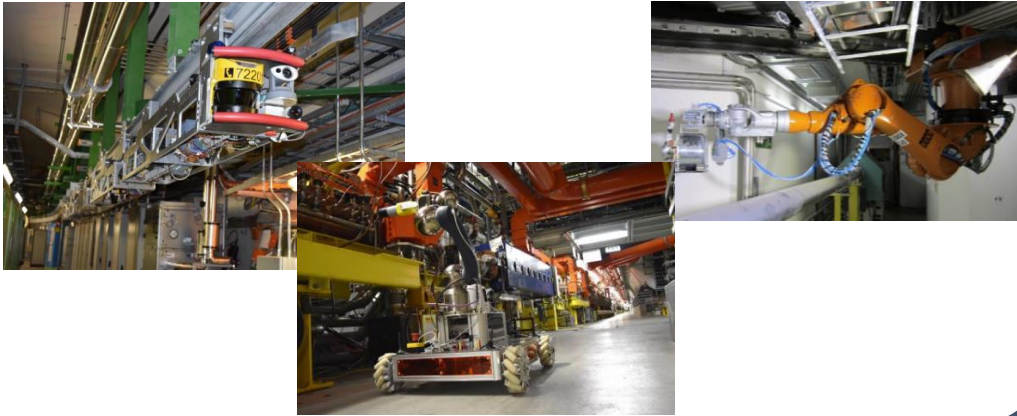




## Context – CERN (European Organization for Nuclear Research)

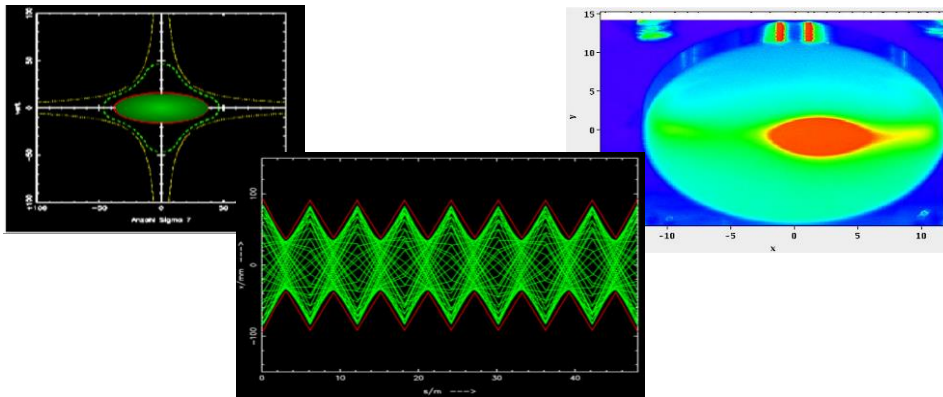
- Many applications of machine learning at CERN

Robotics



Data analysis of  
the physics  
experiments

Particle  
accelerators  
Operation and  
beam  
optimization



Industrial  
controls

## **Context – Neural-network-based controllers**

## **Context – Neural-network-based controllers**

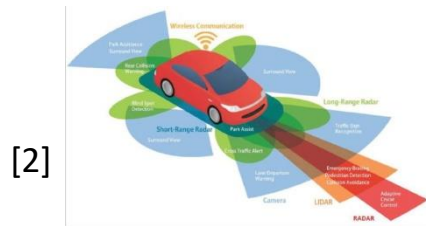
Why NN-based controllers?

## Context – Neural-network-based controllers

Why NN-based controllers?

### Tasks hard to specify

- Autonomous driving



Several rule exceptions

## Context – Neural-network-based controllers

Why NN-based controllers?

### Tasks hard to specify

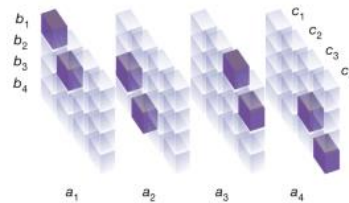
- Autonomous driving



Several rule exceptions

### Computationally fast

- Matrix multiplication





## Context – Neural-network-based controllers

## Why NN-based controllers?

## Tasks hard to specify

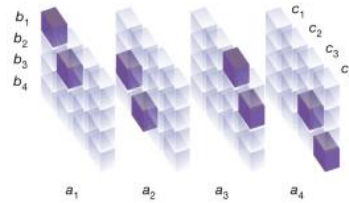
- Autonomous driving



## Several rule exceptions

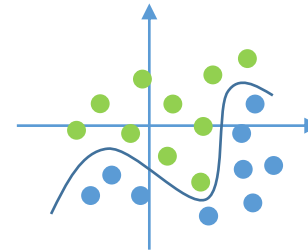
## Computationally fast

- Matrix multiplication



## Versatile

- Non-linearities
- No need to linearize

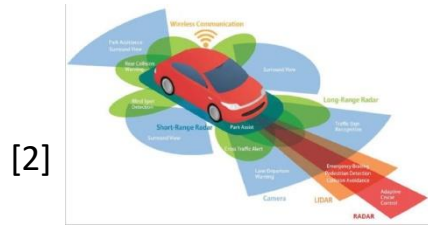


## Context – Neural-network-based controllers

### Why NN-based controllers?

#### Tasks hard to specify

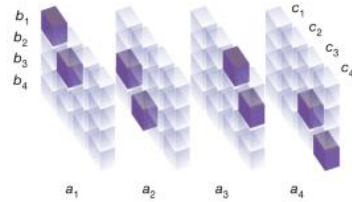
- Autonomous driving



Several rule exceptions

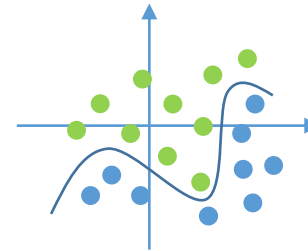
#### Computationally fast

- Matrix multiplication



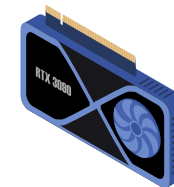
#### Versatile

- Non-linearities
- No need to linearize



#### Only data needed

- No physical modelling required
- Collect data

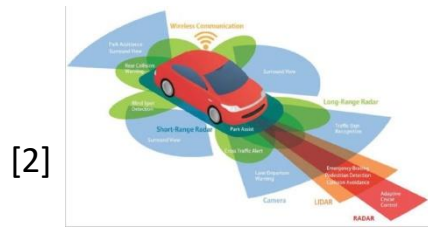


## Context – Neural-network-based controllers

### Why NN-based controllers?

#### Tasks hard to specify

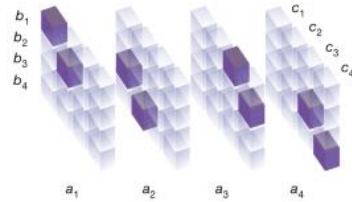
- Autonomous driving



Several rule exceptions

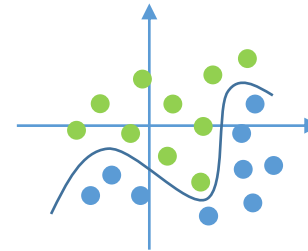
#### Computationally fast

- Matrix multiplication



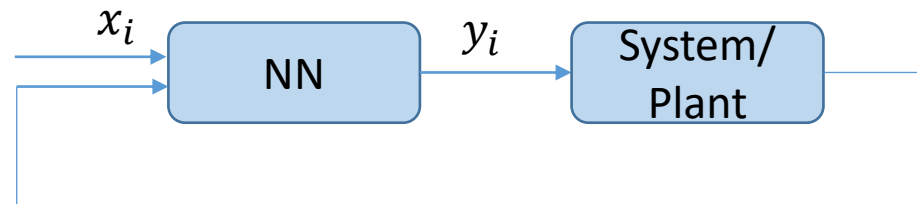
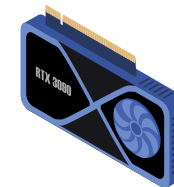
#### Versatile

- Non-linearities
- No need to linearize



#### Only data needed

- No physical modelling required
- Collect data

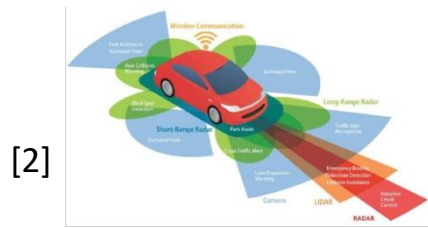


## Context – Neural-network-based controllers

### Why NN-based controllers?

#### Tasks hard to specify

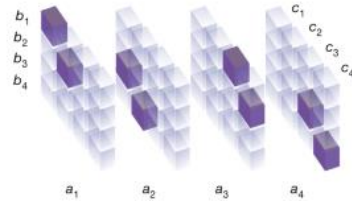
- Autonomous driving



Several rule exceptions

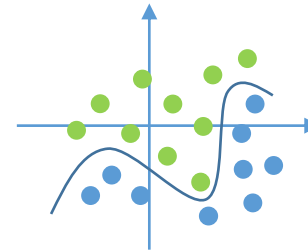
#### Computationally fast

- Matrix multiplication



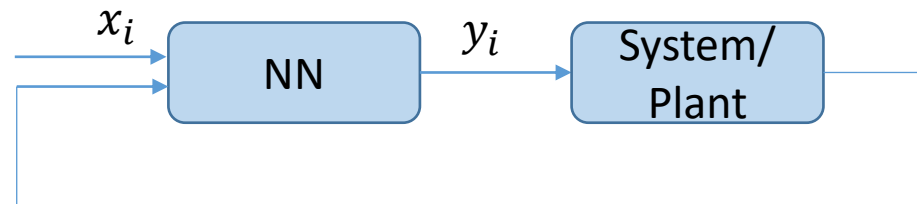
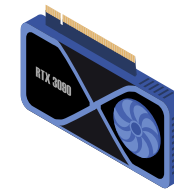
#### Versatile

- Non-linearities
- No need to linearize



#### Only data needed

- No physical modelling required
- Collect data



But a failure in the controller can be fatal



## **Context – Verification of neural-network-based controllers**

Why verification of NNs?

## Context – Verification of neural-network-based controllers

Why verification of NNs?

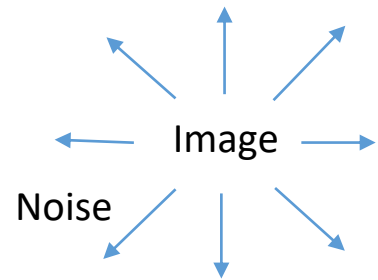
- **Guaranteeing properties**

## Context – Verification of neural-network-based controllers

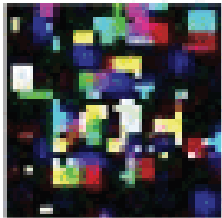
Why verification of NNs?

- **Guaranteeing properties**

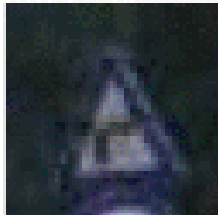
**Robustness**



Dangerous  
curve to the  
right



+ noise



Children  
crossing



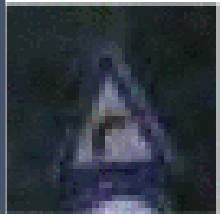
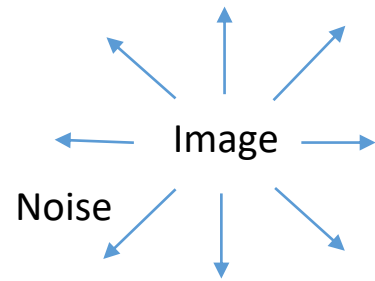
## Context – Verification of neural-network-based controllers

Why verification of NNs?

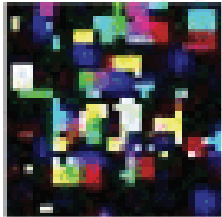
- **Guaranteeing properties**



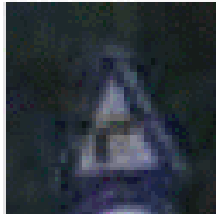
### Robustness



Dangerous  
curve to the  
right



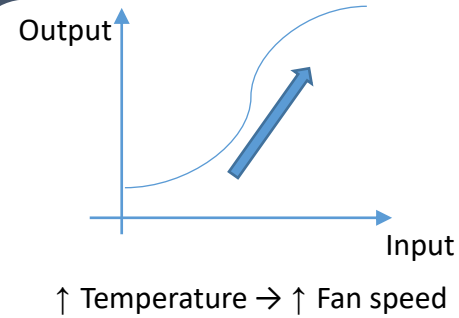
+ noise



Children  
crossing



### Monotonicity





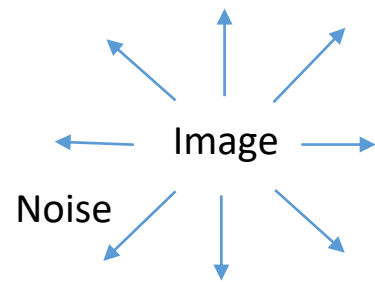
## Context – Verification of neural-network-based controllers

Why verification of NNs?

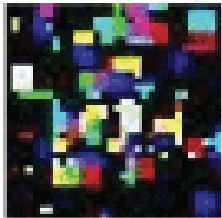
- **Guaranteeing properties**



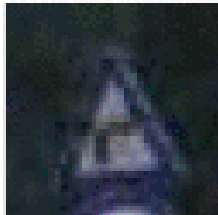
### Robustness



Dangerous  
curve to the  
right



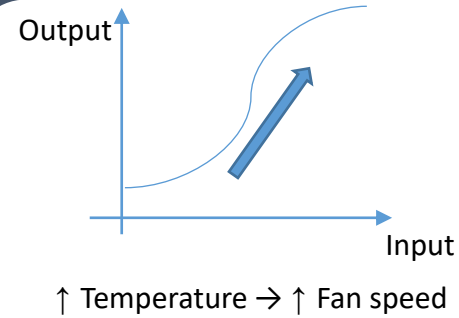
+ noise



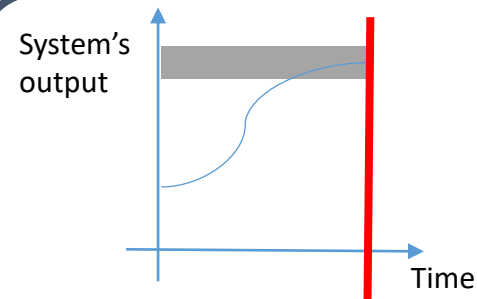
Children  
crossing



### Monotonicity



### Reachability



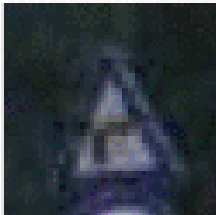
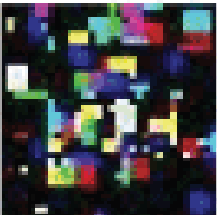
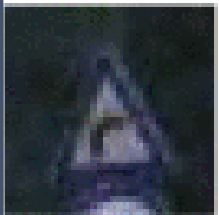
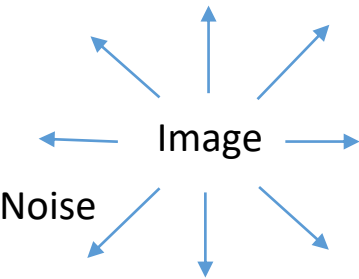
# Context – Verification of neural-network-based controllers

Why verification of NNs?

- **Guaranteeing properties**



## Robustness



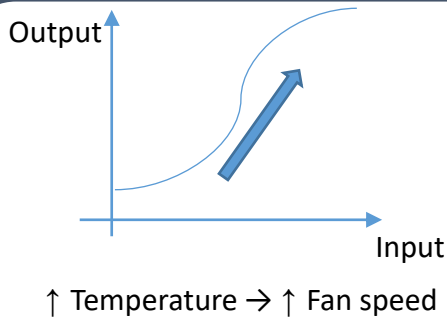
Dangerous  
curve to the  
right

+ noise

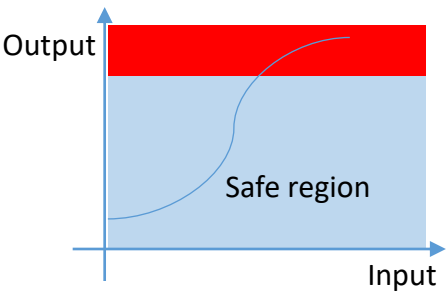
Children  
crossing



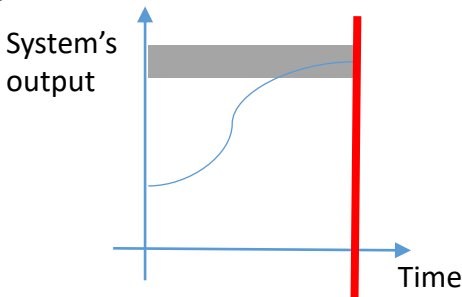
## Monotonicity



## Safety



## Reachability



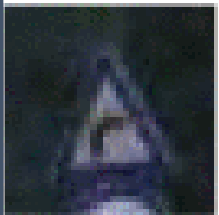
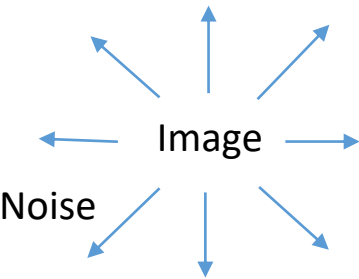
# Context – Verification of neural-network-based controllers

Why verification of NNs?

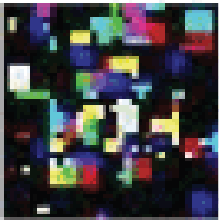
- **Guaranteeing properties**



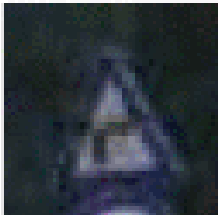
## Robustness



Dangerous curve to the right



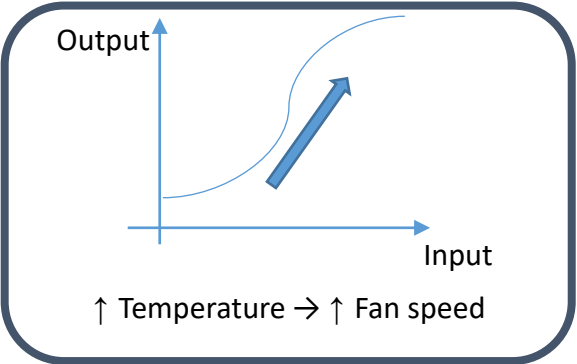
+ noise



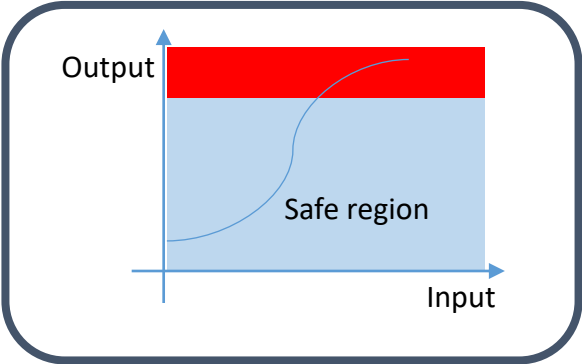
Children crossing



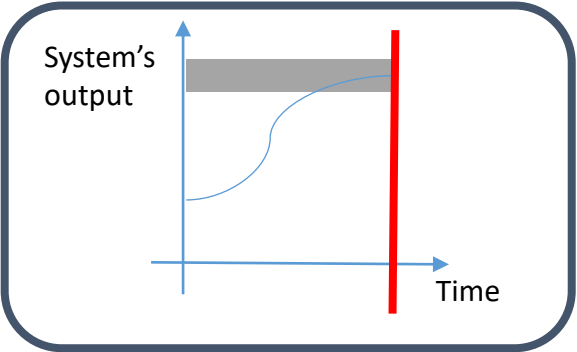
## Monotonicity



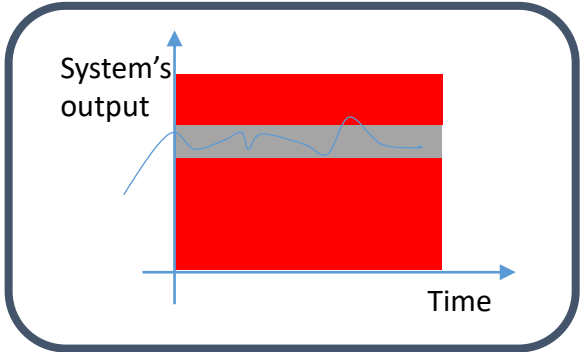
## Safety



## Reachability



## Stability



## Context – Verification of neural-network-based controllers

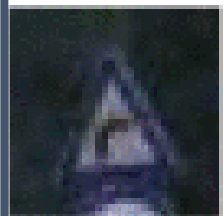
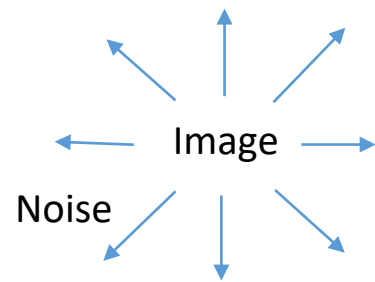
Why verification of NNs?

- **Guaranteeing properties**

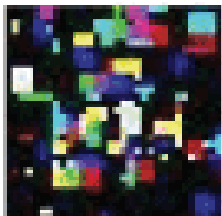


- **Explainability**

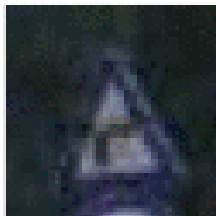
### Robustness



Dangerous curve to the right



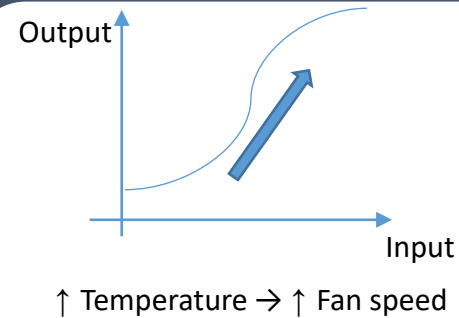
+ noise



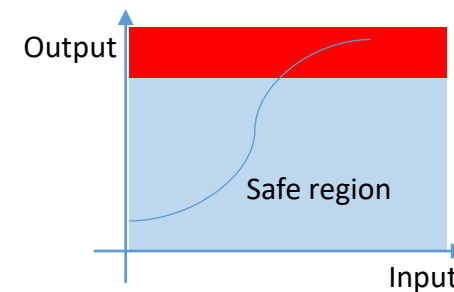
Children crossing



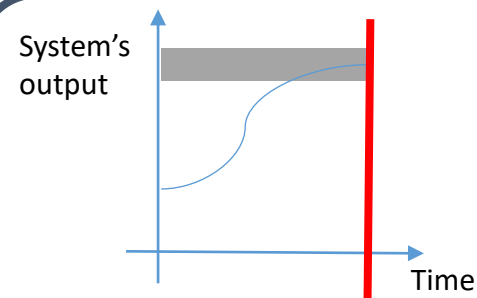
### Monotonicity



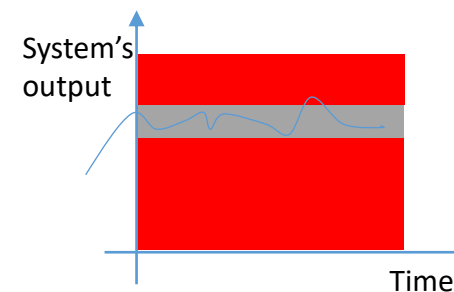
### Safety



### Reachability



### Stability



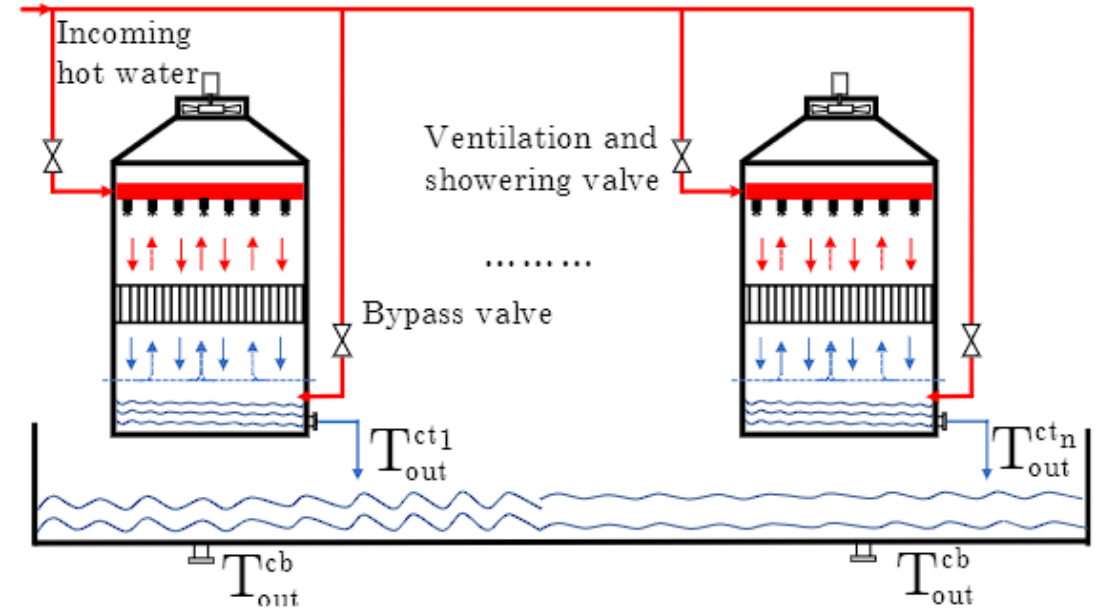
- **Is it possible** that a given situation ever occurs?
- **Is there a combination** of inputs that leads to a given output?



## Case study – LHC cooling towers controller

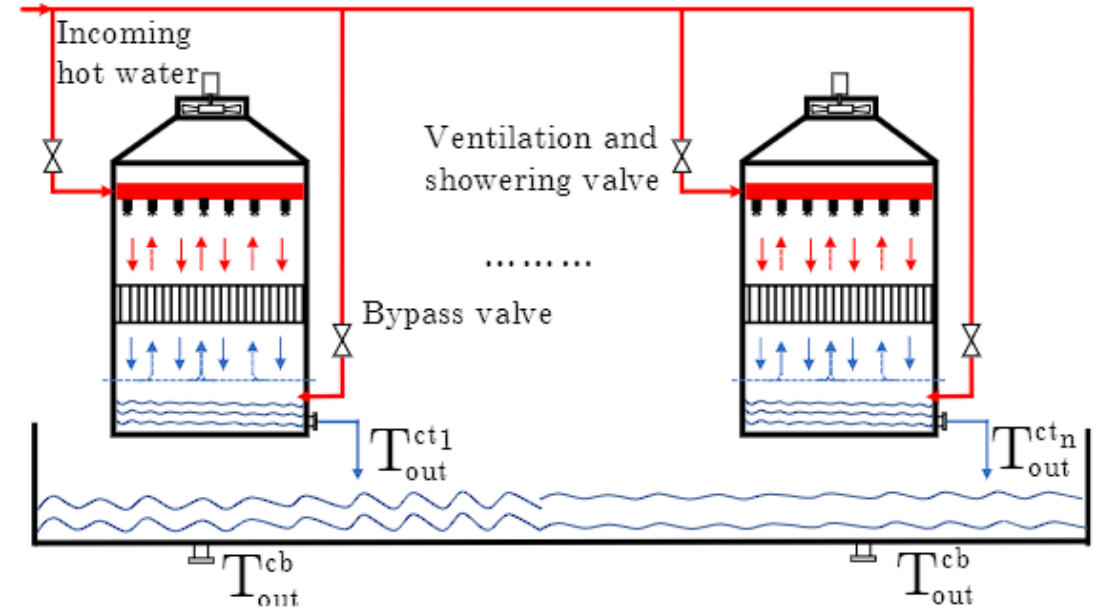
## Case study – LHC cooling towers controller

- Induced draft cooling towers (IDCTs)
- **Provide cold water** for different LHC subsystems (e.g. cryogenics, chillers, air handling units, etc.)



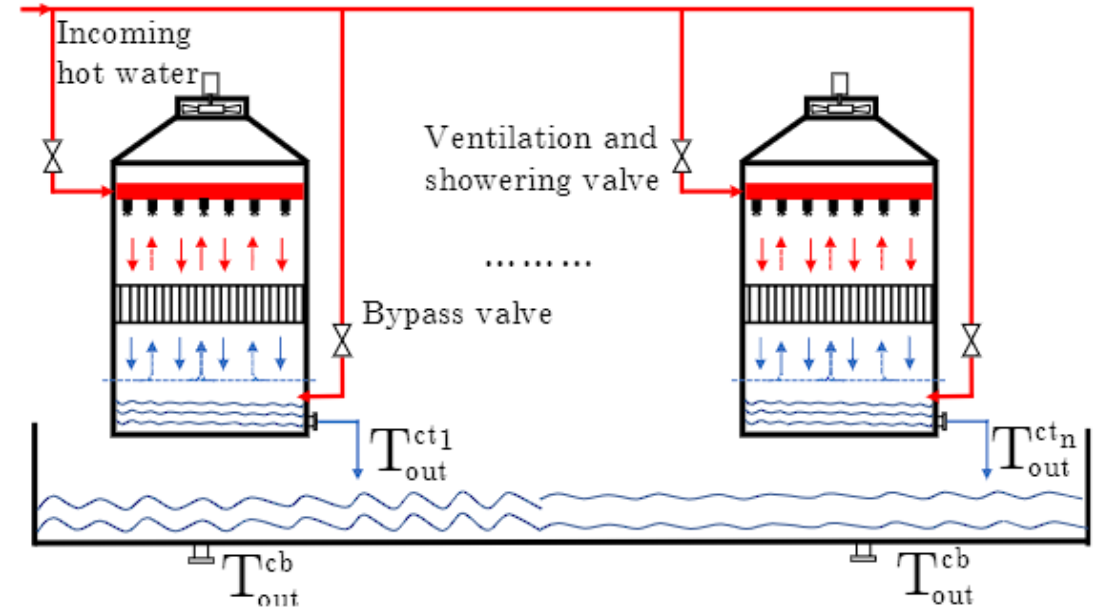
## Case study – LHC cooling towers controller

- Induced draft cooling towers (IDCTs)
- **Provide cold water** for different LHC subsystems (e.g. cryogenics, chillers, air handling units, etc.)
- **Control actions:**
  - **Mode selection:**
    1. Ventilation
    2. Showering
    3. Bypass
  - **Fan speed**



## Case study – LHC cooling towers controller

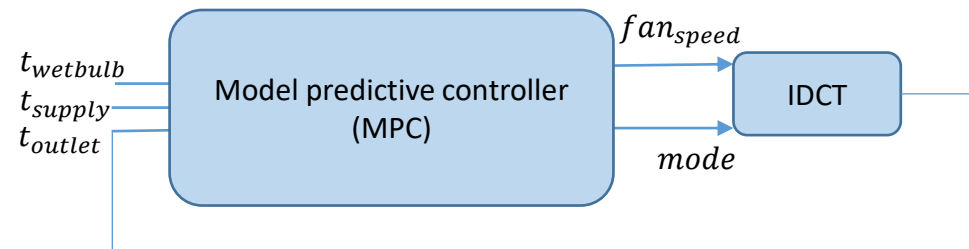
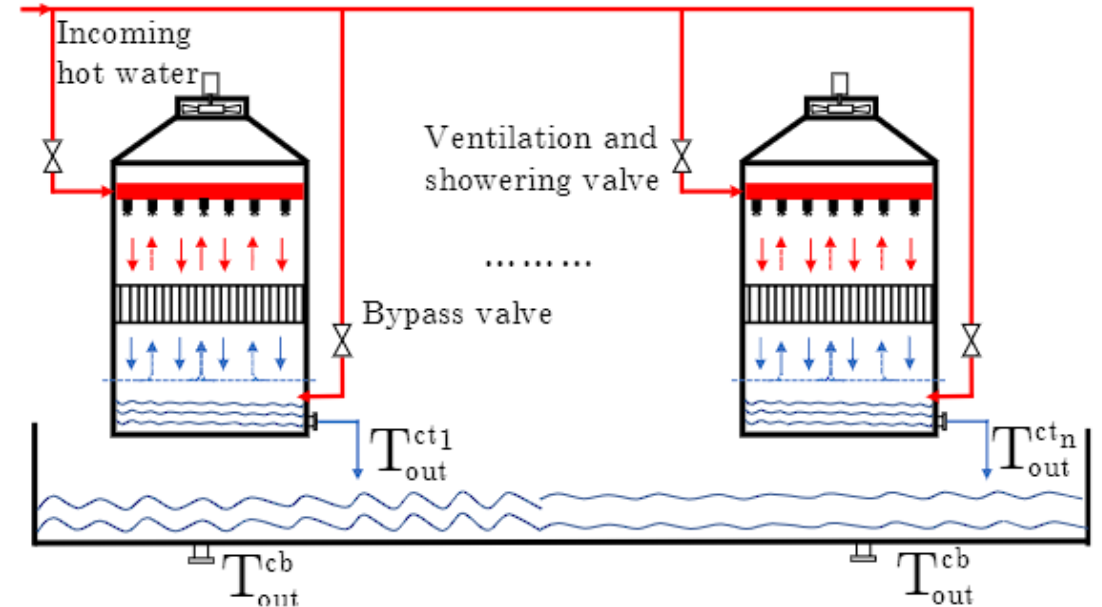
- Induced draft cooling towers (IDCTs)
- **Provide cold water** for different LHC subsystems (e.g. cryogenics, chillers, air handling units, etc.)
- **Control actions:**
  - **Mode selection:**
    1. Ventilation
    2. Showering
    3. Bypass
  - **Fan speed**
- **Control objective:**
  - Keep **outlet water temperature within strict limits**
  - Utilize **minimum amount of energy**





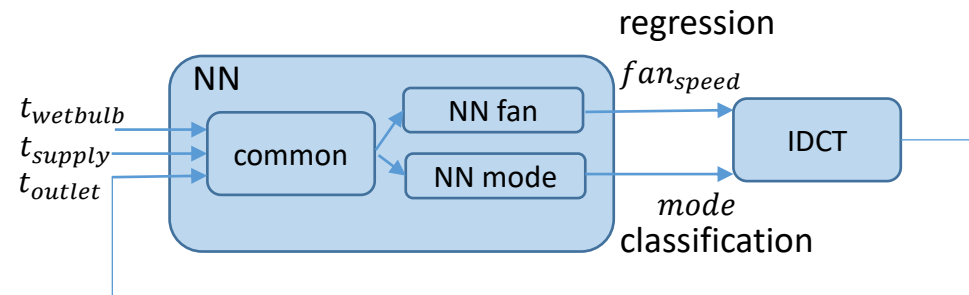
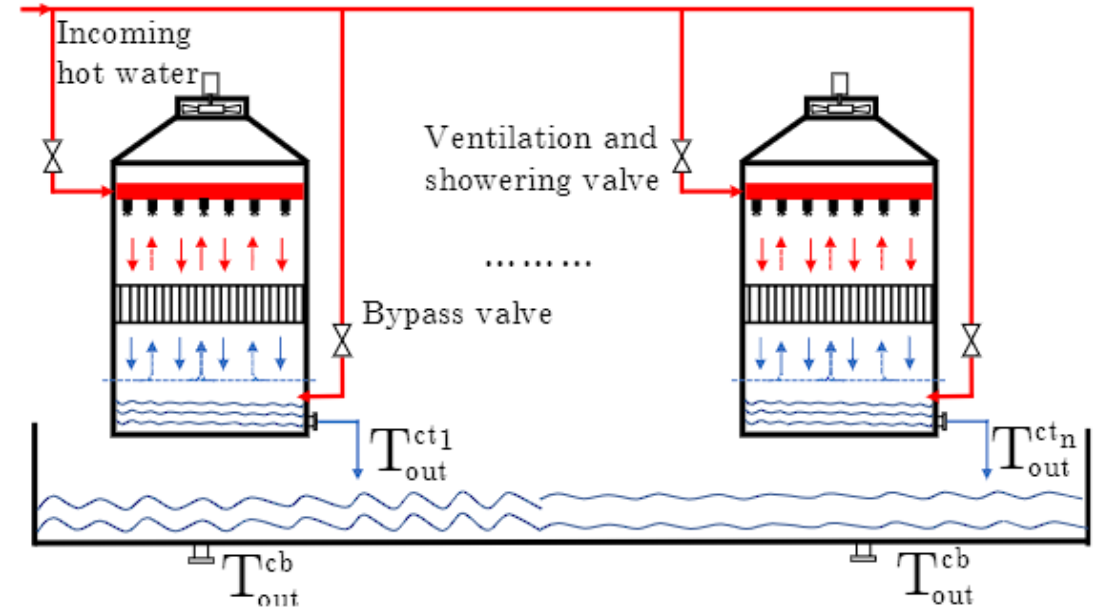
## Case study – LHC cooling towers controller

- Induced draft cooling towers (IDCTs)
- **Provide cold water** for different LHC subsystems (e.g. cryogenics, chillers, air handling units, etc.)
- **Control actions:**
  - **Mode selection:**
    1. Ventilation
    2. Showering
    3. Bypass
  - **Fan speed**
- **Control objective:**
  - Keep **outlet water temperature within strict limits**
  - Utilize **minimum amount of energy**



## Case study – LHC cooling towers controller

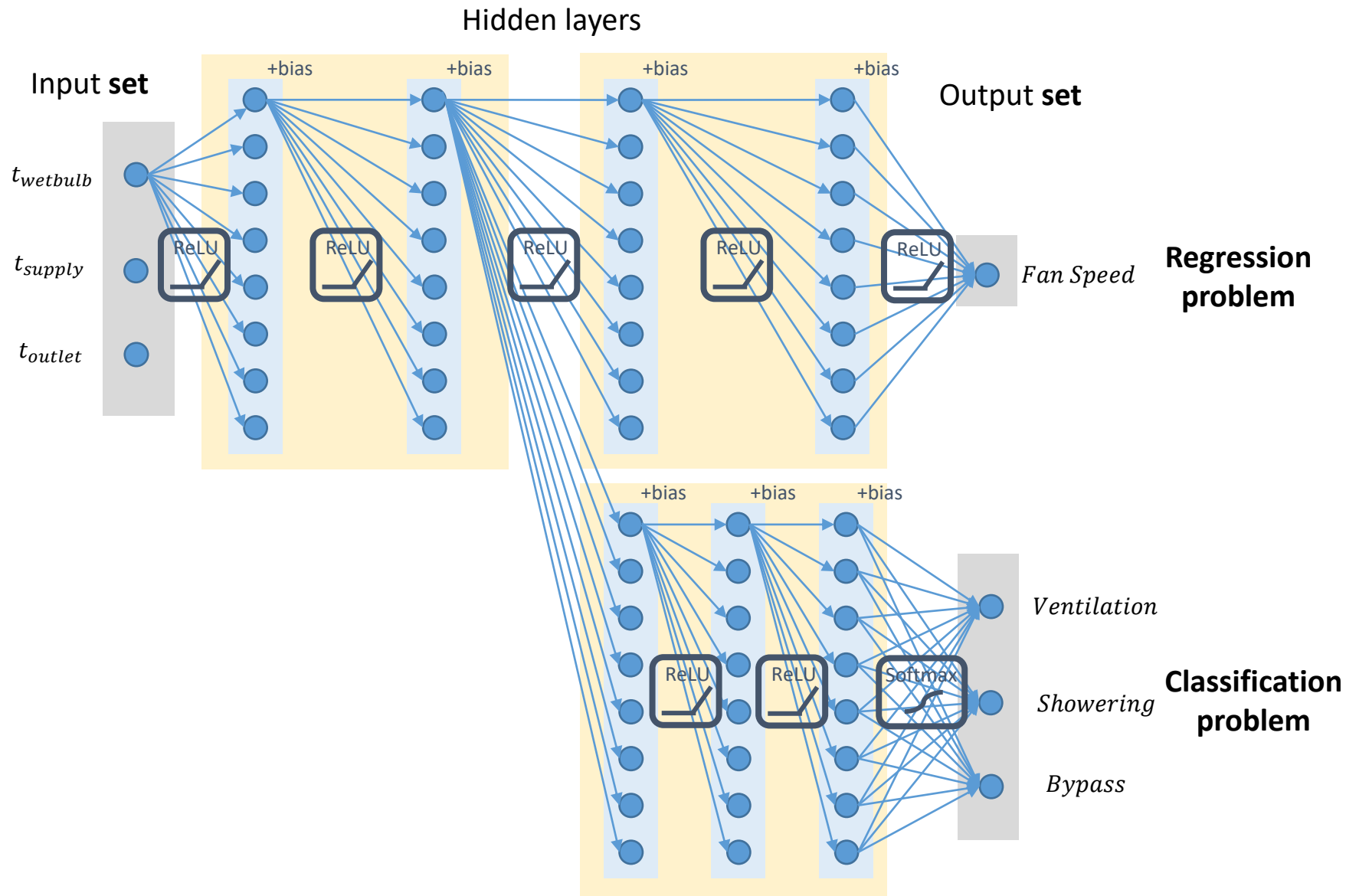
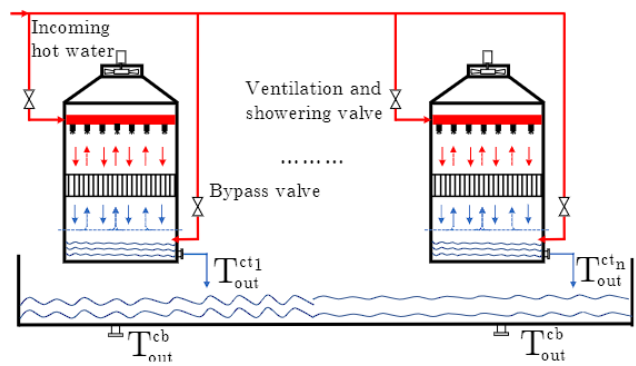
- Induced draft cooling towers (IDCTs)
- **Provide cold water** for different LHC subsystems (e.g. cryogenics, chillers, air handling units, etc.)
- **Control actions:**
  - **Mode selection:**
    1. Ventilation
    2. Showering
    3. Bypass
  - **Fan speed**
- **Control objective:**
  - Keep **outlet water temperature within strict limits**
  - Utilize **minimum amount of energy**



**Case study – NN description**

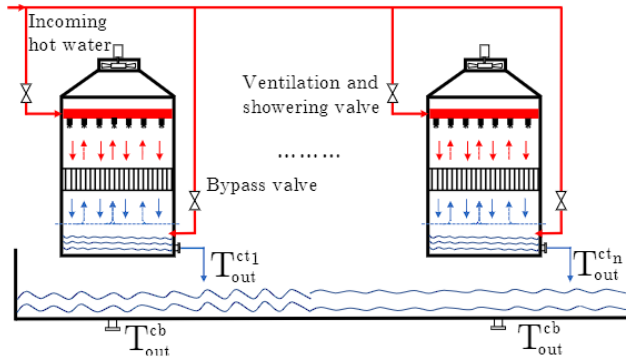
# Case study – NN description

## Initial non-conventional NN architecture

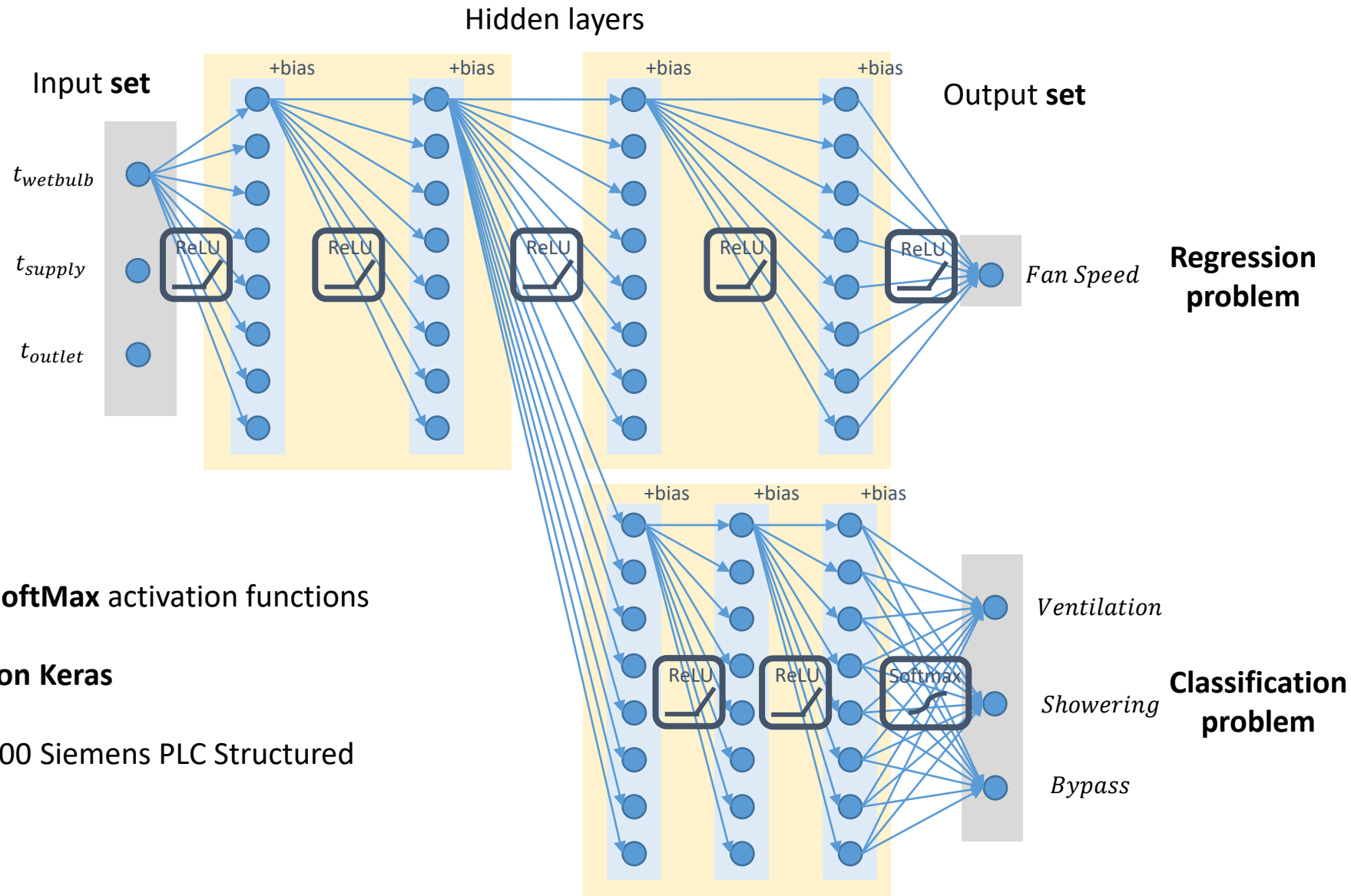


## Case study – NN description

### Initial non-conventional NN architecture



- Feedforward NN with **ReLU** and **SoftMax** activation functions
- Developed and trained with **Python Keras**
- Automatically translated to S7-1200 Siemens PLC Structured Control Language (**SCL**)



**Case study**

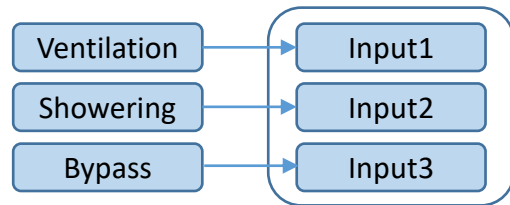
## Case study

Properties to be verified

## Case study

Properties to be verified

Operational modes reachability



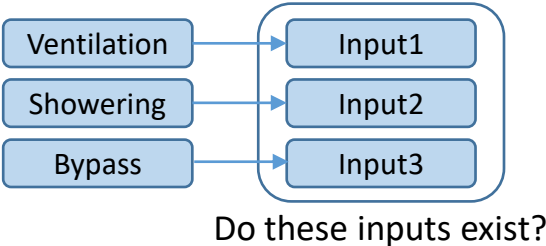
Do these inputs exist?



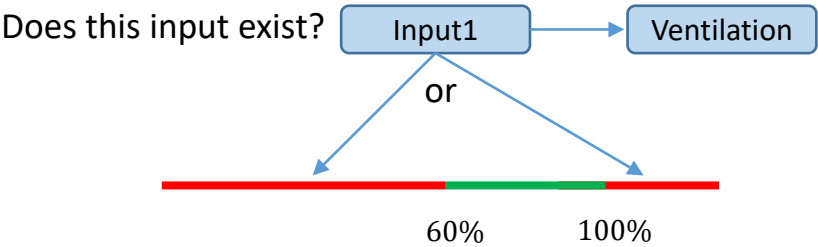
# Case study

Properties to be verified

Operational modes reachability



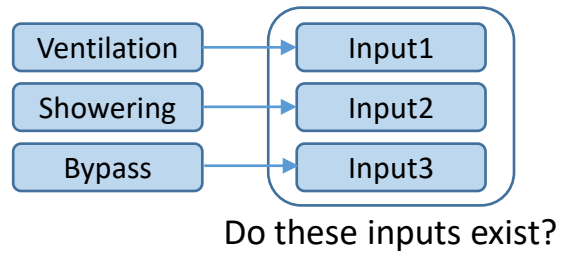
Fan speed constraint satisfaction



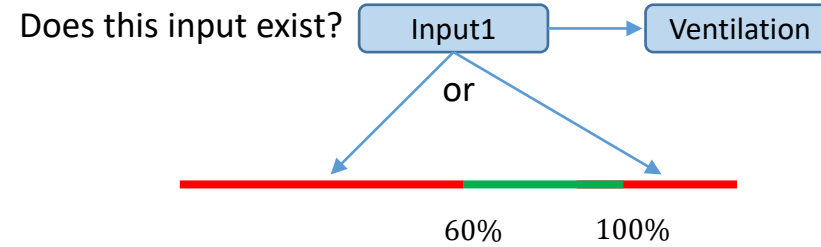
## Case study

Properties to be verified

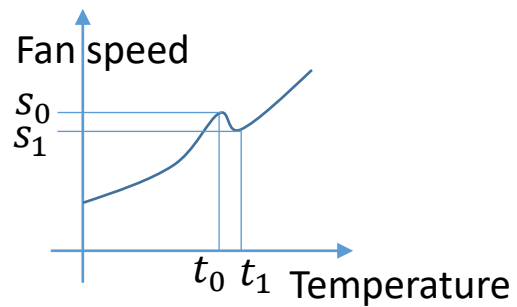
Operational modes reachability



Fan speed constraint satisfaction



Monotonicity

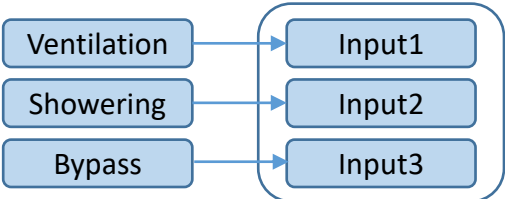


Do  $t_0 < t_1$  exist such  
that  $s_0 > s_1$ ?

# Case study

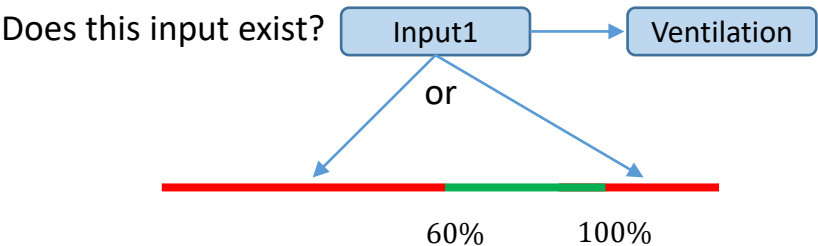
## Properties to be verified

### Operational modes reachability

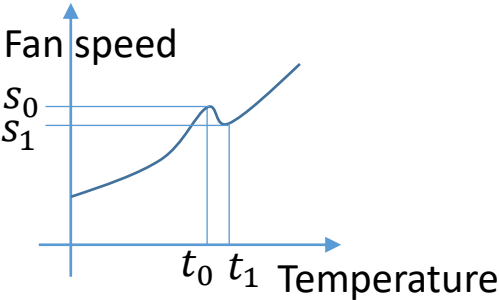


Do these inputs exist?

### Fan speed constraint satisfaction

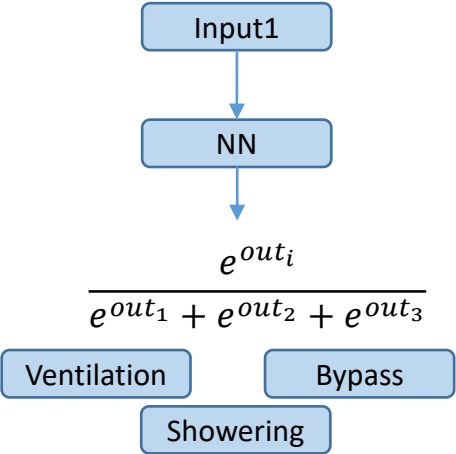


### Monotonicity



Do  $t_0 < t_1$  exist such that  $s_0 > s_1$ ?

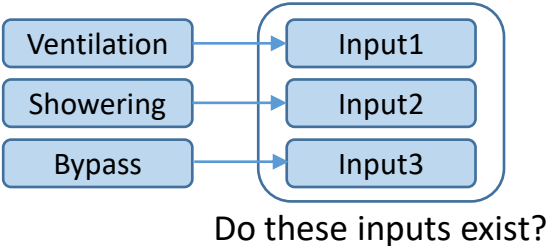
### Softmax overflow



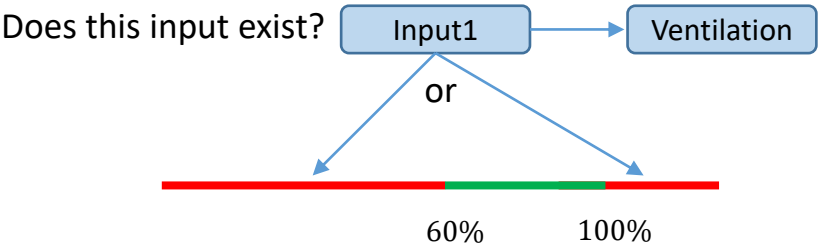
# Case study

## Properties to be verified

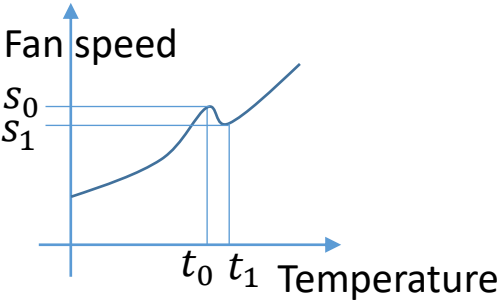
### Operational modes reachability



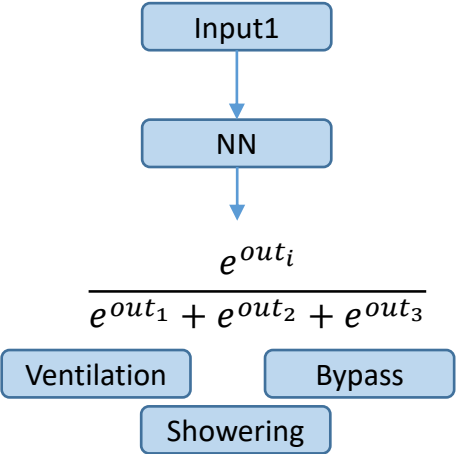
### Fan speed constraint satisfaction



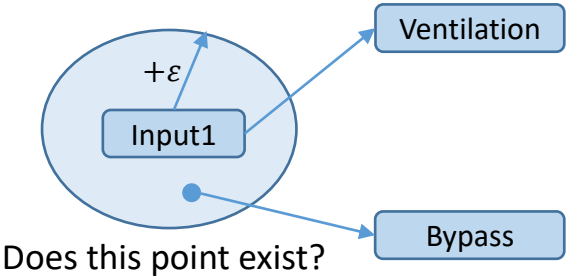
### Monotonicity



### Softmax overflow



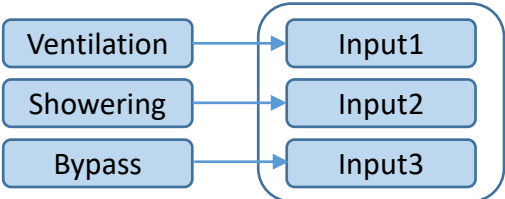
### Robustness



# Case study

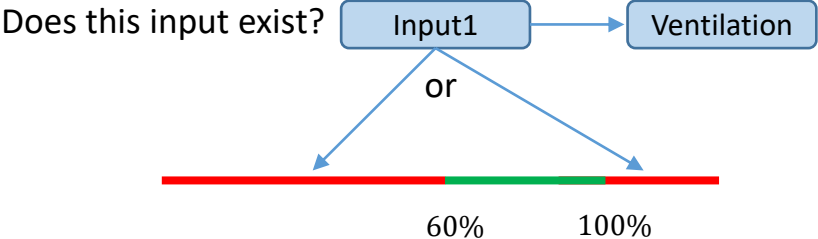
## Properties to be verified

### Operational modes reachability

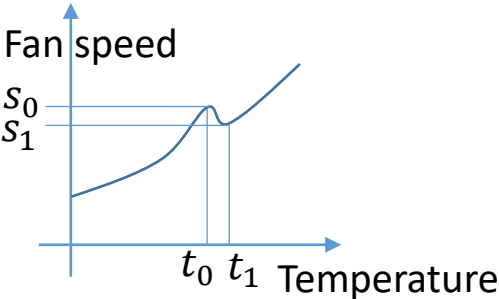


Do these inputs exist?

### Fan speed constraint satisfaction

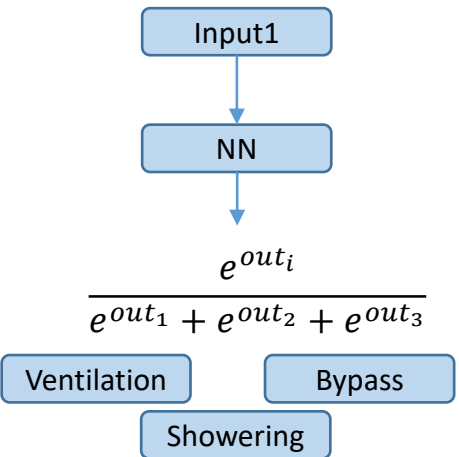


### Monotonicity

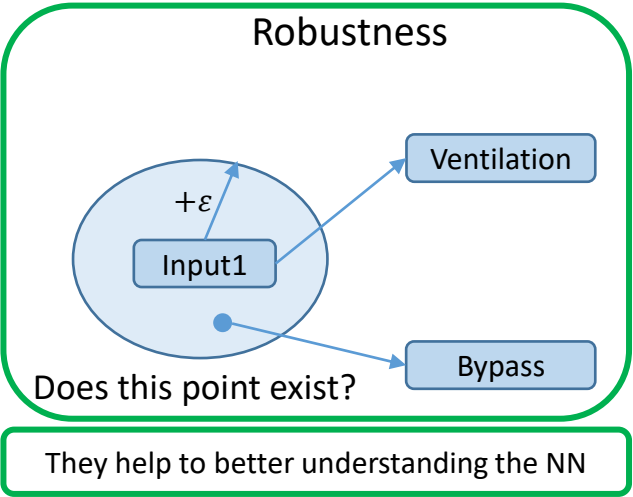


Do  $t_0 < t_1$  exist such that  $s_0 > s_1$ ?

### Softmax overflow



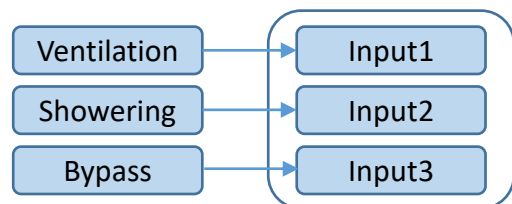
### Robustness



## Case study

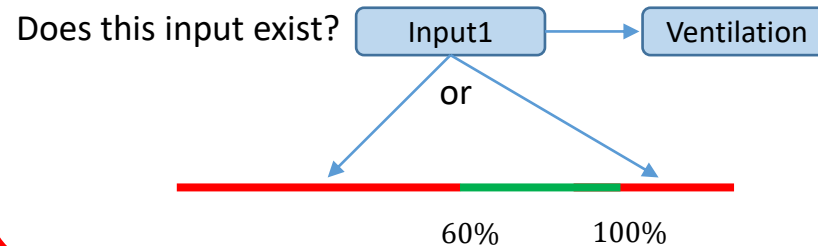
Properties to be verified

Operational modes reachability

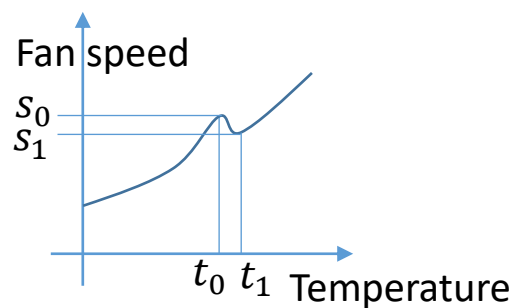


Do these inputs exist?

Fan speed constraint satisfaction

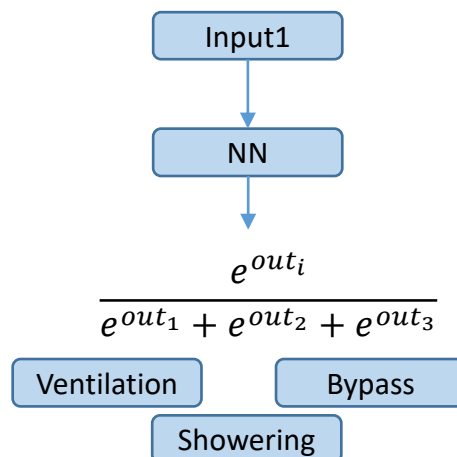


Monotonicity

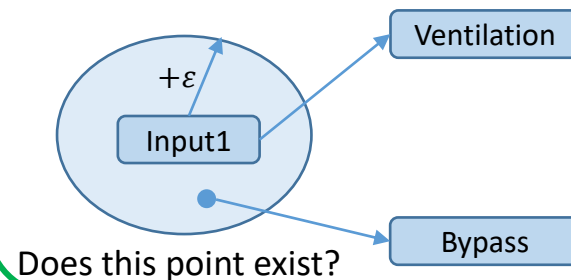


Do  $t_0 < t_1$  exist such that  $s_0 > s_1$ ?

Softmax overflow



Robustness



They help to better understanding the NN

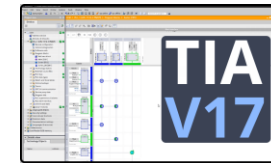
They represent problems

**Approaches to verify these properties**

## Approaches to verify these properties



NN design



PLC source code



Executable

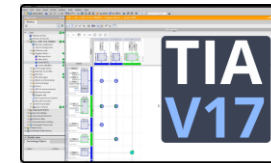


## Approaches to verify these properties

Different methods were applied and compared:



NN design



PLC source code



Executable

## Approaches to verify these properties

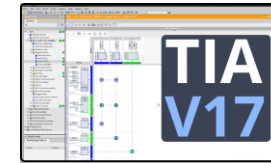
Different methods were applied and compared:

1. **nnenum**: an open-source **NN verification** tool for **ReLU** NNs from Stony Brook University  
<https://github.com/stanleybak/nnenum>



**nnenum**

NN design



PLC source code



Executable

## Approaches to verify these properties

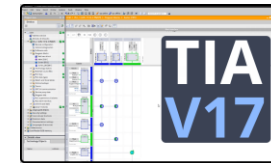
Different methods were applied and compared:

1. **nnenum**: an open-source **NN verification** tool for **ReLU** NNs from Stony Brook University  
<https://github.com/stanleybak/nnenum>
2. **PLCverif**: an open-source formal **verification tool** for **PLC programs** from CERN <https://gitlab.com/plcverif-oss>



**nnenum**

NN design



PLC source code



Executable

## Approaches to verify these properties

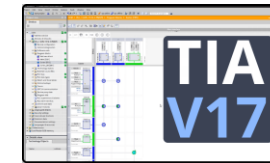
Different methods were applied and compared:

1. **nnenum**: an open-source **NN verification** tool for **ReLU** NNs from Stony Brook University  
<https://github.com/stanleybak/nnenum>
2. **PLCverif**: an open-source formal **verification tool** for **PLC programs** from CERN <https://gitlab.com/plcverif-oss>
3. **Z3**: an open-source **theorem prover** from Microsoft Research <https://github.com/Z3Prover/z3>
4. **Testing**: traditional testing techniques



**nnenum**

NN design



PLC source code



Executable



Ignacio D. Lopez-Miguel et al. “**Verification of Neural Networks Meets PLC Code: An LHC Cooling Tower Control System at CERN**”. In *EANN 2023: Engineering Applications of Neural Networks conference*  
[https://link.springer.com/chapter/10.1007/978-3-031-34204-2\\_35](https://link.springer.com/chapter/10.1007/978-3-031-34204-2_35)

nnenum



## nenum



## Property encoding (vnnlib)

```
(declare-const X_0 Real)
(declare-const X_1 Real)
(declare-const X_2 Real)
(declare-const Y_0 Real)
(declare-const Y_1 Real)
(declare-const Y_2 Real)
(assert (>= X_0 20.0))
(assert (<= X_0 25.0))
(assert (>= X_1 23))
(assert (<= X_1 27))
(assert (>= X_2 8.0))
(assert (<= X_2 21.0))
(assert (>= Y_0 Y_1))
(assert (>= Y_0 Y_2))
```

# nnenum



## Property encoding (vnnlib)

```
(declare-const X_0 Real)
(declare-const X_1 Real)
(declare-const X_2 Real)
(declare-const Y_0 Real)
(declare-const Y_1 Real)
(declare-const Y_2 Real)
(assert (>= X_0 20.0))
(assert (<= X_0 25.0))
(assert (>= X_1 23))
(assert (<= X_1 27))
(assert (>= X_2 8.0))
(assert (<= X_2 21.0))
(assert (>= Y_0 Y_1))
(assert (>= Y_0 Y_2))
```

Inputs

# nenum



## Property encoding (vnnlib)

```
(declare-const X_0 Real)
(declare-const X_1 Real)
(declare-const X_2 Real)
(declare-const Y_0 Real)
(declare-const Y_1 Real)
(declare-const Y_2 Real)

(assert (>= X_0 20.0))
(assert (<= X_0 25.0))
(assert (>= X_1 23))
(assert (<= X_1 27))
(assert (>= X_2 8.0))
(assert (<= X_2 21.0))
(assert (>= Y_0 Y_1))
(assert (>= Y_0 Y_2))
```

Inputs

Outputs



# nnenum



## Property encoding (vnnlib)

```
(declare-const X_0 Real)
(declare-const X_1 Real)
(declare-const X_2 Real)
(declare-const Y_0 Real)
(declare-const Y_1 Real)
(declare-const Y_2 Real)
(assert (>= X_0 20.0))
(assert (<= X_0 25.0))
(assert (>= X_1 23))
(assert (<= X_1 27))
(assert (>= X_2 8.0))
(assert (<= X_2 21.0))
(assert (>= Y_0 Y_1))
(assert (>= Y_0 Y_2))
```

Inputs

Outputs

Range of the inputs:

$25 \geq x_0 \geq 20$

$27 \geq x_1 \geq 23$

$21 \geq x_2 \geq 8$

# nnenum



## Property encoding (vnnlib)

```
(declare-const X_0 Real)
```

```
(declare-const X_1 Real)
```

```
(declare-const X_2 Real)
```

```
(declare-const Y_0 Real)
```

```
(declare-const Y_1 Real)
```

```
(declare-const Y_2 Real)
```

```
(assert (>= X_0 20.0))
```

```
(assert (<= X_0 25.0))
```

```
(assert (>= X_1 23))
```

```
(assert (<= X_1 27))
```

```
(assert (>= X_2 8.0))
```

```
(assert (<= X_2 21.0))
```

```
(assert (>= Y_0 Y_1))
```

```
(assert (>= Y_0 Y_2))
```

Inputs

Outputs

Range of the inputs:

$25 \geq x_0 \geq 20$

$27 \geq x_1 \geq 23$

$21 \geq x_2 \geq 8$

Property to verify:

$y_0 \geq y_1 \wedge y_0 \geq y_2$

*It is possible that the  
selected mode is  
**ventilation***

## nnenum



### Property encoding (vnnlib)

```
(declare-const X_0 Real)
(declare-const X_1 Real)
(declare-const X_2 Real)
(declare-const Y_0 Real)
(declare-const Y_1 Real)
(declare-const Y_2 Real)
(assert (>= X_0 20.0))
(assert (<= X_0 25.0))
(assert (>= X_1 23))
(assert (<= X_1 27))
(assert (>= X_2 8.0))
(assert (<= X_2 21.0))
(assert (>= Y_0 Y_1))
(assert (>= Y_0 Y_2))
```

Inputs

Outputs

Range of the inputs:

$$25 \geq x_0 \geq 20$$

$$27 \geq x_1 \geq 23$$

$$21 \geq x_2 \geq 8$$

Property to verify:

$$y_0 \geq y_1 \wedge y_0 \geq y_2$$

*It is possible that the  
selected mode is  
**ventilation***

### Goal

Find an example that satisfies all conditions, i.e.,  
a set of  $\{x_0, x_1, x_2\}$  such that  $(y_0 \geq y_1 \wedge y_0 \geq y_2)$

# nnenum



## Property encoding (vnnlib)

```
(declare-const X_0 Real)
(declare-const X_1 Real)
(declare-const X_2 Real)
(declare-const Y_0 Real)
(declare-const Y_1 Real)
(declare-const Y_2 Real)
(assert (>= X_0 20.0))
(assert (<= X_0 25.0))
(assert (>= X_1 23))
(assert (<= X_1 27))
(assert (>= X_2 8.0))
(assert (<= X_2 21.0))
(assert (>= Y_0 Y_1))
(assert (>= Y_0 Y_2))
```

Inputs

Outputs

Range of the inputs:

$25 \geq x_0 \geq 20$

$27 \geq x_1 \geq 23$

$21 \geq x_2 \geq 8$

Property to verify:

$y_0 \geq y_1 \wedge y_0 \geq y_2$

*It is possible that the  
selected mode is  
ventilation*

## Goal

Find an example that satisfies all conditions, i.e.,  
a set of  $\{x_0, x_1, x_2\}$  such that  $(y_0 \geq y_1 \wedge y_0 \geq y_2)$

## Execution

```
root@7d4848b3ea5f:/work# python3 -m nnenum.nnenum sharedDocker/final/modes.onnx sharedDocker/
/final/modesReachability_0.vnnlib
Running in parallel with 16 processes
(0.1 sec) Q: 0, Sets: 0/1 (0.0%) ETA: - (expected 1 stars)

Pre-overapproximation simulation found a confirmed counterexample.

Unsafe Base Branch: (Mode: 0)

Total Stars: 1 (0 exact, 1 approx)
Runtime: 0.1 sec
Completed work frac: 1.0
Num Stars Copied Between Processes: 0
Num Lps During Enumeration: 0
Total Num Lps: 0

Result: network is UNSAFE with confirmed counterexample in result.cinput and result.coutput
Input: [22.5, 25.0, 14.5]
Output: [0.7701750993728638, -0.5644218325614929, -1.8442020416259766]
root@7d4848b3ea5f:/work#
```

# nnenum



## Property encoding (vnnlib)

```
(declare-const X_0 Real)
(declare-const X_1 Real)
(declare-const X_2 Real)
(declare-const Y_0 Real)
(declare-const Y_1 Real)
(declare-const Y_2 Real)
(assert (>= X_0 20.0))
(assert (<= X_0 25.0))
(assert (>= X_1 23))
(assert (<= X_1 27))
(assert (>= X_2 8.0))
(assert (<= X_2 21.0))
(assert (>= Y_0 Y_1))
(assert (>= Y_0 Y_2))
```

Inputs

Outputs

Range of the inputs:

$25 \geq x_0 \geq 20$

$27 \geq x_1 \geq 23$

$21 \geq x_2 \geq 8$

Property to verify:

$y_0 \geq y_1 \wedge y_0 \geq y_2$

*It is possible that the  
selected mode is  
ventilation*

## Goal

Find an example that satisfies all conditions, i.e.,  
a set of  $\{x_0, x_1, x_2\}$  such that  $(y_0 \geq y_1 \wedge y_0 \geq y_2)$

## Execution

```
root@7d4848b3ea5f:/work# python3 -m nnenum.nnenum sharedDocker/final/modes.onnx sharedDocker/
/final/modesReachability_0.vnnlib
Running in parallel with 16 processes
(0.1 sec) Q: 0, Sets: 0/1 (0.0%) ETA: - (expected 1 stars)

Pre-overapproximation simulation found a confirmed counterexample.

Unsafe Base Branch: (Mode: 0)

Total Stars: 1 (0 exact, 1 approx)
Runtime: 0.1 sec
Completed work frac: 1.0
Num Stars Copied Between Processes: 0
Num Lps During Enumeration: 0
Total Num Lps: 0

Result: network is UNSAFE with confirmed counterexample in result.cinput and result.coutput
Input: [22.5, 25.0, 14.5]
Output: [0.7701750993728638, -0.5644218325614929, -1.8442020416259766]
root@7d4848b3ea5f:/work#
```

**Evidence**

# nnenum



## Property encoding (vnnlib)

```
(declare-const X_0 Real)
(declare-const X_1 Real)
(declare-const X_2 Real)
(declare-const Y_0 Real)
(declare-const Y_1 Real)
(declare-const Y_2 Real)
(assert (>= X_0 20.0))
(assert (<= X_0 25.0))
(assert (>= X_1 23))
(assert (<= X_1 27))
(assert (>= X_2 8.0))
(assert (<= X_2 21.0))
(assert (>= Y_0 Y_1))
(assert (>= Y_0 Y_2))
```

Inputs

Outputs

Range of the inputs:

$$25 \geq x_0 \geq 20$$

$$27 \geq x_1 \geq 23$$

$$21 \geq x_2 \geq 8$$

Property to verify:

$$y_0 \geq y_1 \wedge y_0 \geq y_2$$

*It is possible that the  
selected mode is  
ventilation*

## Goal

Find an example that satisfies all conditions, i.e.,  
a set of  $\{x_0, x_1, x_2\}$  such that  $(y_0 \geq y_1 \wedge y_0 \geq y_2)$

## Execution

```
root@7d4848b3ea5f:/work# python3 -m nnenum.nnenum sharedDocker/final/modes.onnx sharedDocker
/final/modesReachability_0.vnnlib
Running in parallel with 16 processes
(0.1 sec) Q: 0, Sets: 0/1 (0.0%) ETA: - (expected 1 stars)

Pre-overapproximation simulation found a confirmed counterexample.

Unsafe Base Branch: (Mode: 0)

Total Stars: 1 (0 exact, 1 approx)
Runtime: 0.1 sec
Completed work frac: 1.0
Num Stars Copied Between Processes: 0
Num Lps During Enumeration: 0
Total Num Lps: 0

Result: network is UNSAFE with confirmed counterexample in result.cinput and result.coutput
Input: [22.5, 25.0, 14.5]
Output: [0.7701750993728638, -0.5644218325614929, -1.8442020416259766]
root@7d4848b3ea5f:/work#
```

**Evidence**

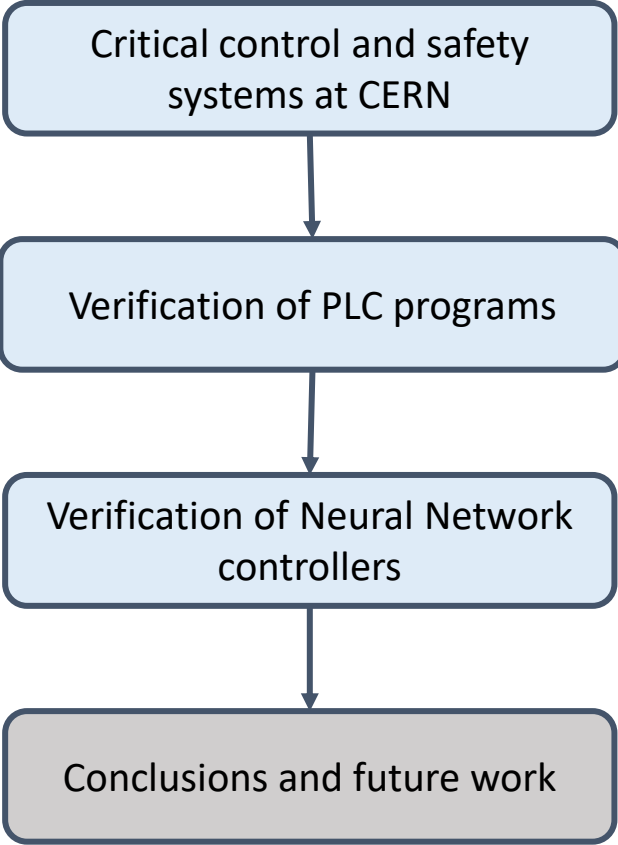
## Pros:

- Very efficient
- Scalable

## Cons:

- Limited to certain architectures
- No loops
- No complex properties


# Roadmap




**Conclusions (1)**



## Conclusions (1)

- **Formal verification** (e.g. model checking) can be used **to verify critical software** (critical PLC programs)
- There are not commercial tools (yet) for PLC programs, this is why we developed **PLCverif** 
- **PLCverif has been applied to many critical PLC programs** at CERN and outside CERN

## Conclusions (1)

- **Formal verification** (e.g. model checking) can be used **to verify critical software** (critical PLC programs)
- There are not commercial tools (yet) for PLC programs, this is why we developed **PLCverif** 
- **PLCverif has been applied to many critical PLC programs** at CERN and outside CERN
- Still many **challenges**:
  - **State space explosion** problem (verification performance)
  - Properties **specification**
  - Automatic generation of models
  - **Counterexample analysis** (what do we do when we find a problem?)











**Conclusions (2)**

## Conclusions (2)

1. **Important to verify neural networks** in critical systems to:
  - **guarantee properties** such as robustness, stability, safety, etc.
  - have a **better understanding** of the behavior of the NN

# Conclusions (2)

1. Important to verify neural networks in critical systems to:
- **guarantee properties** such as robustness, stability, safety, etc.
  - have a **better understanding** of the behavior of the NN

*Simulation*.input1	Floating-point nu...	21.3	21.3		
*Simulation*.input2	Floating-point nu...	23.0	23.0		
*Simulation*.input3	Floating-point nu...	8.0	8.0		
*NN_Result_DB*.fan_speed	Floating-point nu...	0.00262890317651313			
*NN_Result_DB*.modes[0]	Floating-point nu...	6.01462636744791E-08			
*NN_Result_DB*.modes[1]	Floating-point nu...	0.00348845387816139			
*NN_Result_DB*.modes[2]	Floating-point nu...	0.996511485975575			

2. We use **simulators** (e.g. Siemens PLCSIM advanced) **to confirm the property violations** (counterexamples)

# Conclusions (2)

1. Important to verify neural networks in critical systems to:
- **guarantee properties** such as robustness, stability, safety, etc.
  - have a **better understanding** of the behavior of the NN

*Simulation*.input1	Floating-point nu...	21.3	21.3	<input checked="" type="checkbox"/>	
*Simulation*.input2	Floating-point nu...	23.0	23.0	<input checked="" type="checkbox"/>	
*Simulation*.input3	Floating-point nu...	8.0	8.0	<input checked="" type="checkbox"/>	
*NN_Result_DB*.fan_speed	Floating-point nu...	0.00262890317651313		<input type="checkbox"/>	
*NN_Result_DB*.modes[0]	Floating-point nu...	6.01462636744791E-08		<input type="checkbox"/>	
*NN_Result_DB*.modes[1]	Floating-point nu...	0.00348845387816139		<input type="checkbox"/>	
*NN_Result_DB*.modes[2]	Floating-point nu...	0.996511485975575		<input type="checkbox"/>	

2. We use **simulators** (e.g. Siemens PLCSIM advanced) **to confirm the property violations** (counterexamples)

3. We analyzed **different verification tools**

	performance	scalability	expressiveness	same types?	plug-and-play?
PLCverif	low	low	<b>high</b>	<b>yes</b>	<b>yes</b>
nnenum	<b>very high</b>	<b>high</b>	low	no	no
Z3	medium	medium	low	no	no
Testing	high	very low	medium	no	no

# Conclusions (2)

1. Important to verify neural networks in critical systems to:
- **guarantee properties** such as robustness, stability, safety, etc.
  - have a **better understanding** of the behavior of the NN

*Simulation*.input1	Floating-point nu...	21.3	21.3	<input checked="" type="checkbox"/>	
*Simulation*.input2	Floating-point nu...	23.0	23.0	<input checked="" type="checkbox"/>	
*Simulation*.input3	Floating-point nu...	8.0	8.0	<input checked="" type="checkbox"/>	
*NN_Result_DB*.fan_speed	Floating-point nu...	0.00262890317651313		<input type="checkbox"/>	
*NN_Result_DB*.modes[0]	Floating-point nu...	6.01462636744791E-08		<input type="checkbox"/>	
*NN_Result_DB*.modes[1]	Floating-point nu...	0.00348845387816139		<input type="checkbox"/>	
*NN_Result_DB*.modes[2]	Floating-point nu...	0.996511485975575		<input type="checkbox"/>	

2. We use **simulators** (e.g. Siemens PLCSIM advanced) **to confirm the property violations** (counterexamples)

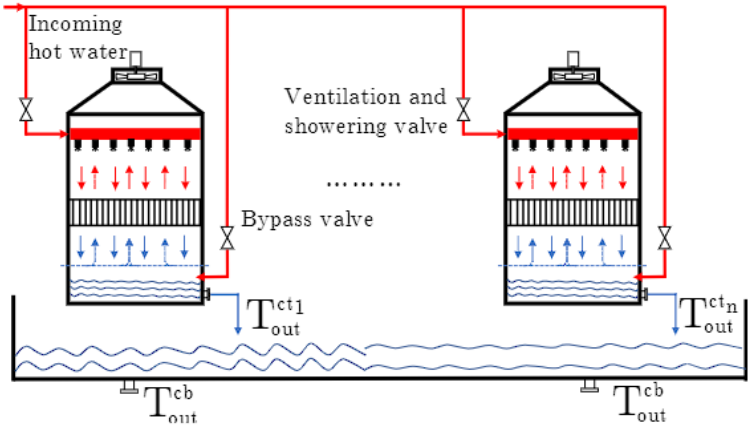
3. We analyzed **different verification tools**

	performance	scalability	expressiveness	same types?	plug-and-play?
PLCverif	low	low	<b>high</b>	<b>yes</b>	<b>yes</b>
nnenum	<b>very high</b>	<b>high</b>	low	no	no
Z3	medium	medium	low	no	no
Testing	high	very low	medium	no	no



Ignacio D. Lopez-Miguel et al. “**Verification of Neural Networks Meets PLC Code: An LHC Cooling Tower Control System at CERN**”. In EANN 2023: Engineering Applications of Neural Networks conference [https://link.springer.com/chapter/10.1007/978-3-031-34204-2\\_35](https://link.springer.com/chapter/10.1007/978-3-031-34204-2_35)

4. We have applied to a real **case study** for industrial controls at **CERN**



**Future work**



Future work

“Traditional” PLC-based controllers

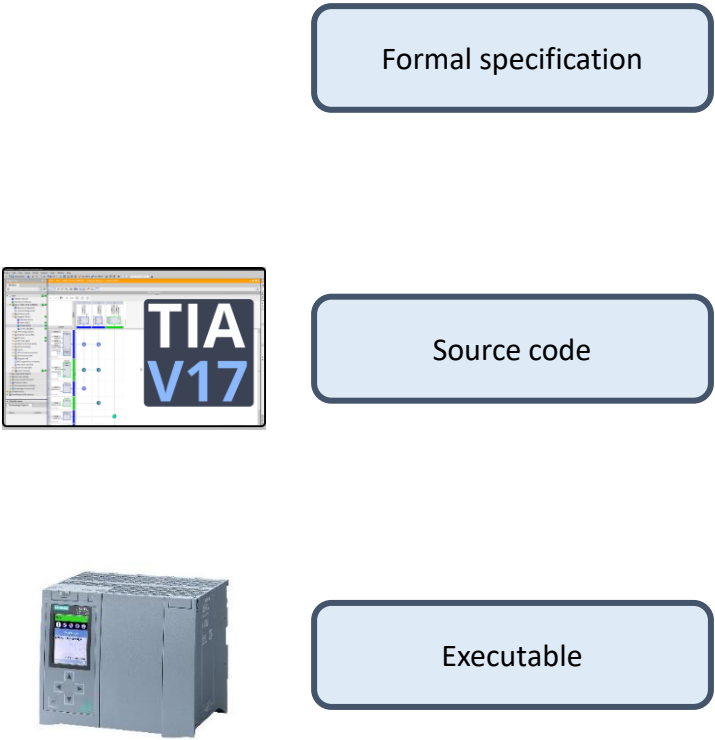
NN-based controllers



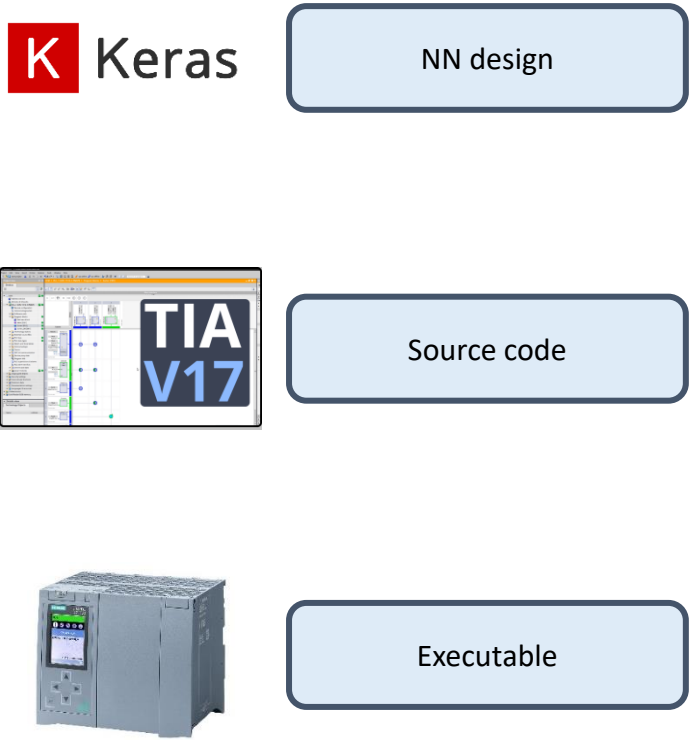
Towards **reliable and safe control software**

# Future work

## “Traditional” PLC-based controllers



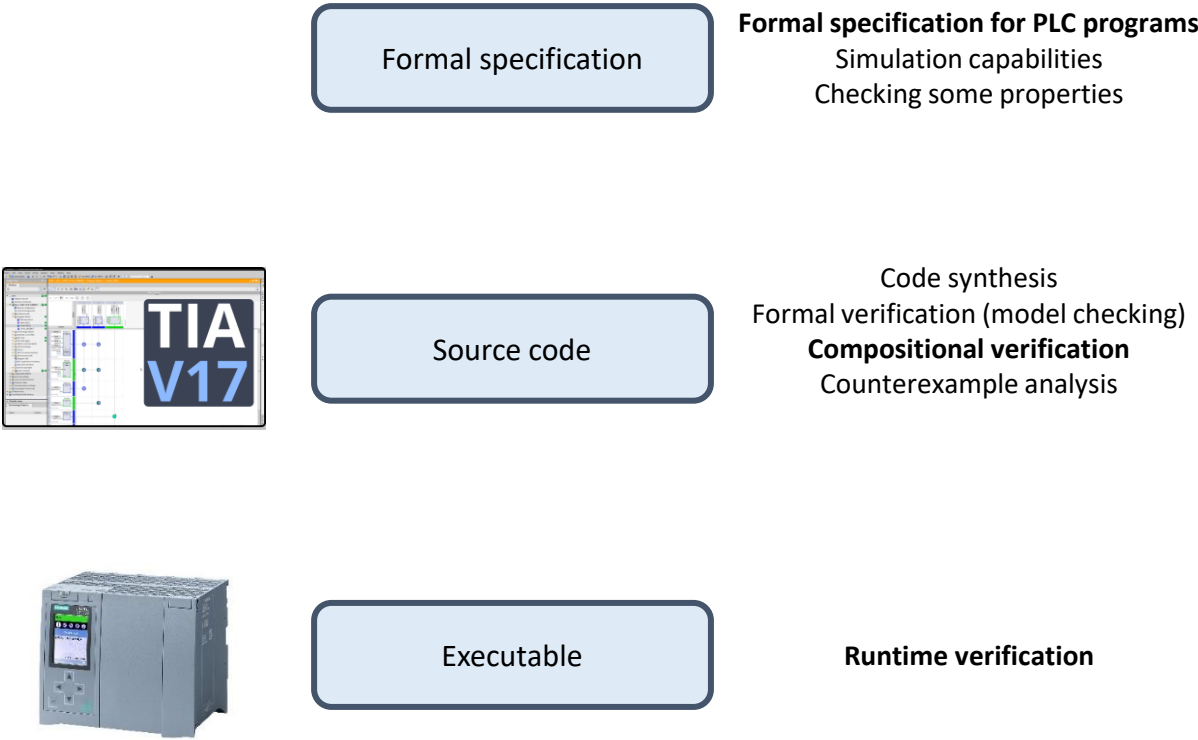
## NN-based controllers



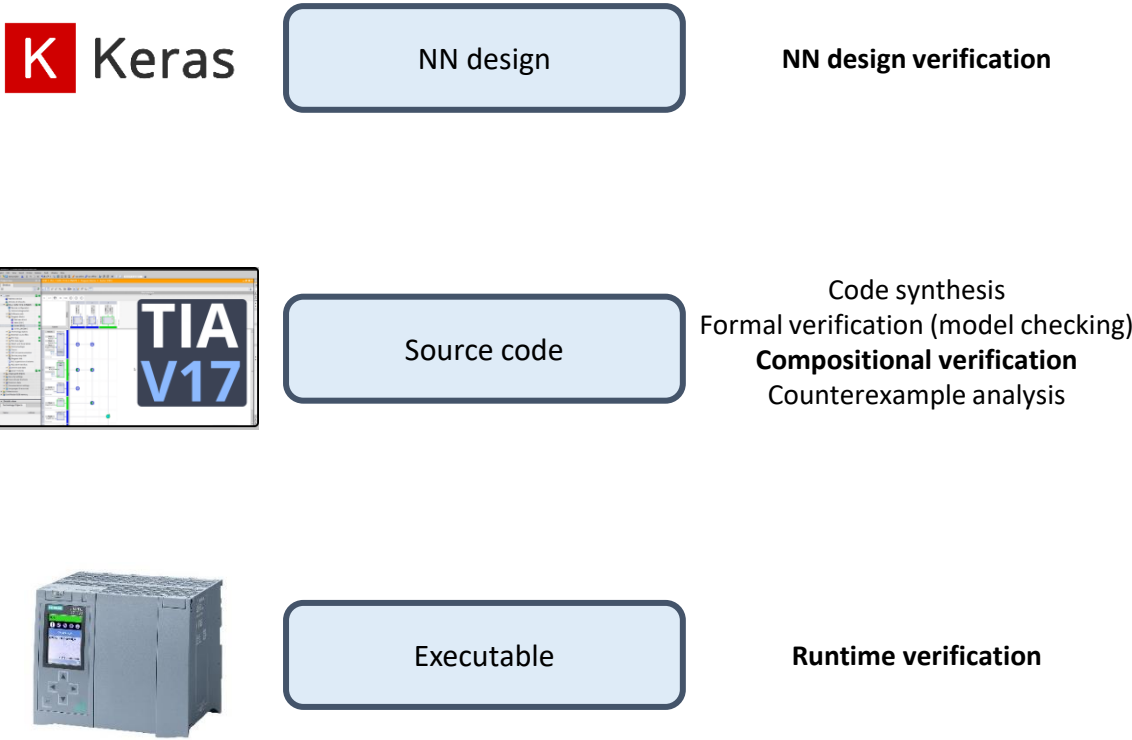
Towards **reliable and safe control software**

# Future work

## “Traditional” PLC-based controllers



## NN-based controllers



Towards **reliable and safe control software**