
CERN

Formal methods for critical control systems at CERN

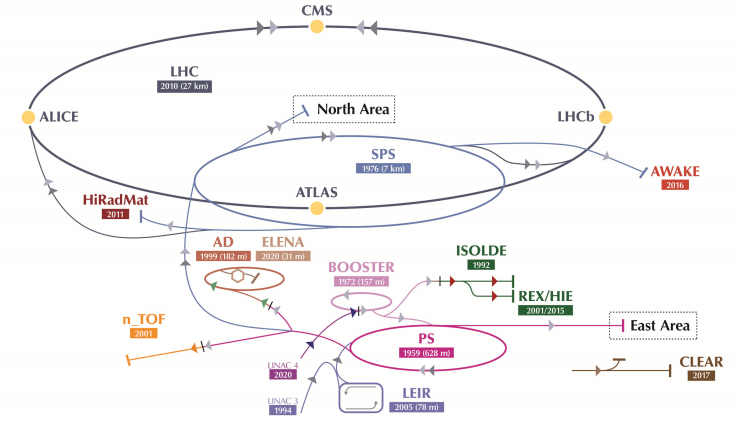
Xaver Fink, Borja Fernández Adiego
Alpine Verification Meeting 26
22nd Sept 2026

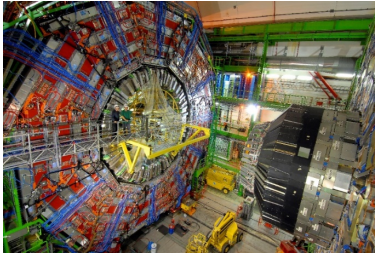
on behalf of the CERN BE-ICS group

Introduction – CERN

CERN: Conseil **e**uropéen pour la **r**echerche **n**ucléaire

- Biggest particle accelerator complex in the world

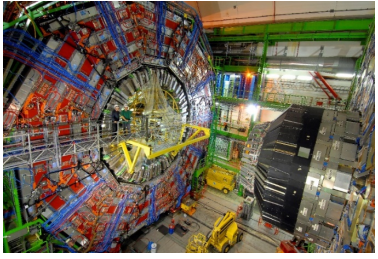




Introduction – Control systems at CERN

CERN: Conseil **e**uropéen pour la **r**echerche **n**ucléaire

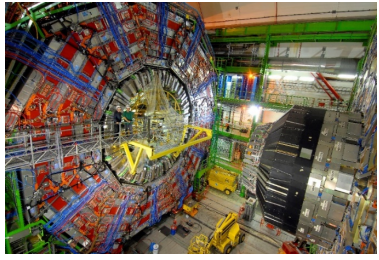
- Biggest particle accelerator complex in the world
- Numerous **advanced control applications**
  - Industrial control systems (e.g. cooling towers)
  - Beam-control & steering
  - Robotics (e.g. automated maintenance)
  - ...

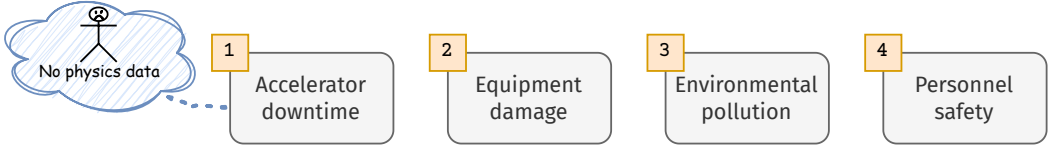


Introduction – Control systems at CERN

CERN: Conseil **e**uropéen pour la **r**echerche **n**ucléaire

- Biggest particle accelerator complex in the world
- Numerous **advanced control applications**
  - Industrial control systems (e.g. cooling towers)
  - Beam-control & steering
  - Robotics (e.g. automated maintenance)
  - ...

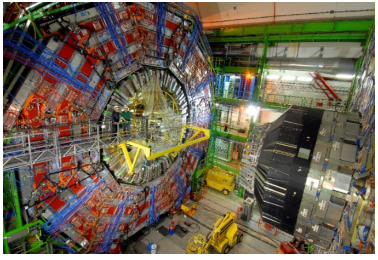




Introduction – Control systems at CERN

CERN: Conseil **e**uropéen pour la **r**echerche **n**ucléaire

- Biggest particle accelerator complex in the world
- Numerous **advanced control applications**
  - Industrial control systems (e.g. cooling towers)
  - Beam-control & steering
  - Robotics (e.g. automated maintenance)
  - ...



CERN BE-ICS Group Mission

Technology, frameworks, engineering for control systems and projects at CERN

- Applying functional safety standards → Formal Methods for software/system verification

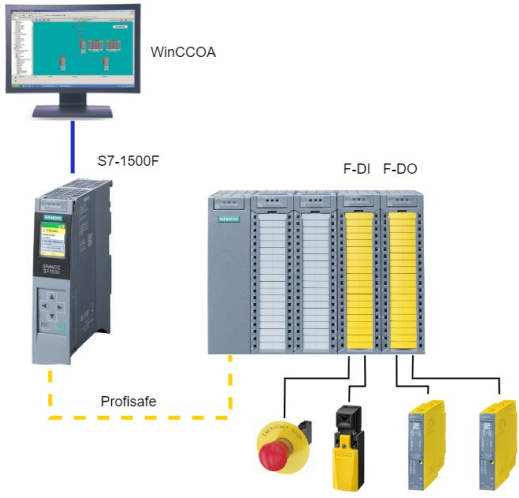
Risks and control challenges

- Ultra-high vacuum (10^{-10} **to** 10^{-11} **mbar**)
- Cryogenic temperatures (**-271°C**)
- $\approx$ **12,000 A** to create the necessary magnetic field
- Beam energy $\approx$ **362 megajoules** (MJ)
- **Radiation**
- Optimize the **energy consumption**
- Timing and synchronization
- ...

Risks and control challenges

- Ultra-high vacuum (10^{-10} **to** 10^{-11} **mbar**)
- Cryogenic temperatures (**-271°C**)
- $\approx$ **12,000 A** to create the necessary magnetic field
- Beam energy $\approx$ **362 megajoules** (MJ)
- **Radiation**
- Optimize the **energy consumption**
- Timing and synchronization
- ...

We need **reliable** control and safety systems

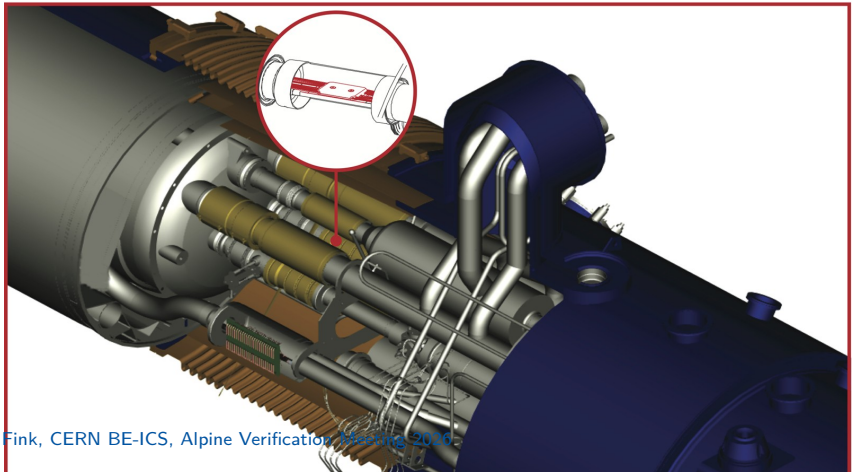


Consequences of failure

- Why? Equipment failure can be extremely expensive

Consequences of failure

- Why? Equipment failure can be extremely expensive
- LHC helium leak failure in 2008 $\rightarrow$ Delay by 1 year
  - Safety systems performed as expected and minimized consequences



Central question

How can we obtain strong guarantees that the systems work correctly?

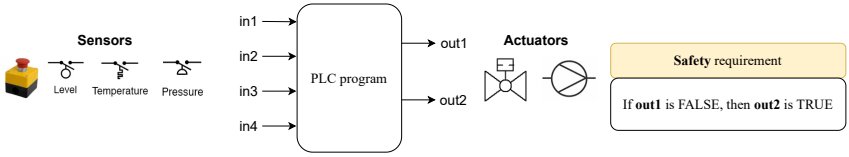
Central question

How can we obtain strong guarantees that the systems work correctly?

A system works correctly if we have both

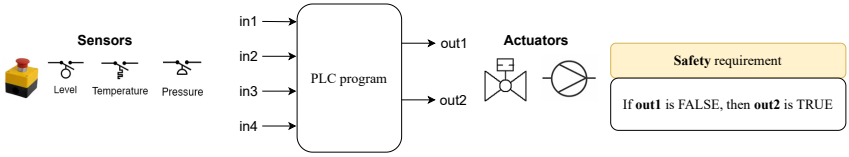
- Safety: nothing bad ever happens
- Functional/liveness: eventually something good happens

Testing?



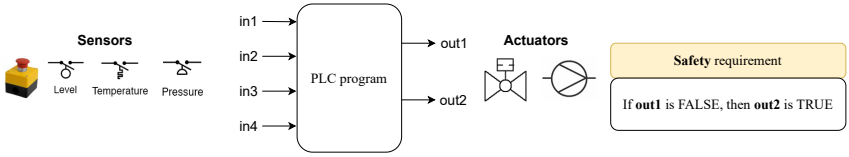
- If "**in1**", "**in2**", "**in3**" and "**in4**" are **BOOL**, then we need to check $2^4 = 16$ combinations
- If the inputs are **INT** (16-bit), then $2^{16*4} \approx 1.8 * 10^{19}$ combinations
- For real (large) systems (many variables), such requirements cannot (practically) be checked by using testing techniques

Testing?



- If "**in1**", "**in2**", "**in3**" and "**in4**" are **BOOL**, then we need to check $2^4 = 16$ combinations
- If the inputs are **INT** (16-bit), then $2^{16*4} \approx 1.8 * 10^{19}$ combinations
- For real (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**

Testing?



- If "**in1**", "**in2**", "**in3**" and "**in4**" are **BOOL**, then we need to check $2^4 = 16$ combinations
- If the inputs are **INT** (16-bit), then $2^{16*4} \approx 1.8 * 10^{19}$ combinations
- For real (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**
- **Solution**: **abstraction-based formal verification**

Topic 1: PLC control logic

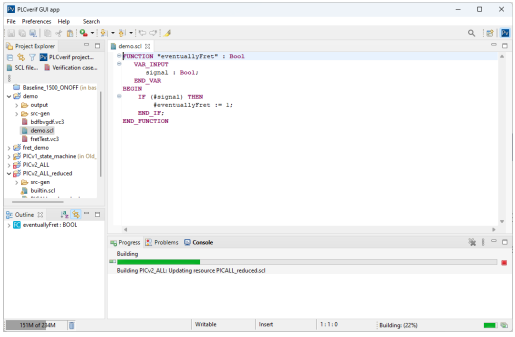
1. Programmable Logic Controllers



Figure: Siemens S7-1500 F PLC

PLCverif: gitlab.com/plcverif-oss

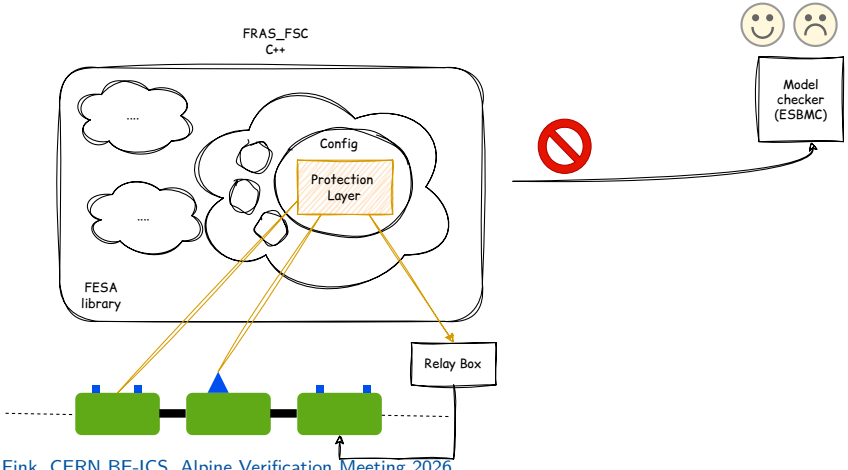
- Formal verification tool for PLC programs actively developed and used at CERN



Topic 2: C++ equipment-control software

Not all project use PLCs! Standard C++ software

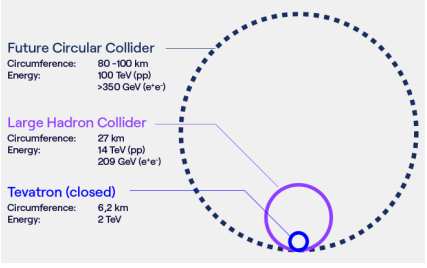
- Partial verification using autogenerated library sims



Topic 3: AI-enabled control

Problem: Systems are becoming larger and more complex!

- Next-Generation particle accelerators
  - FCC: $\sim 4\times$ the number of controls
- Increasing use of sensor feedback and optimization-based control



"The next generation accelerator control system will have to [...] provide integration of artificial intelligence (AI)." [BBB$^+$25]

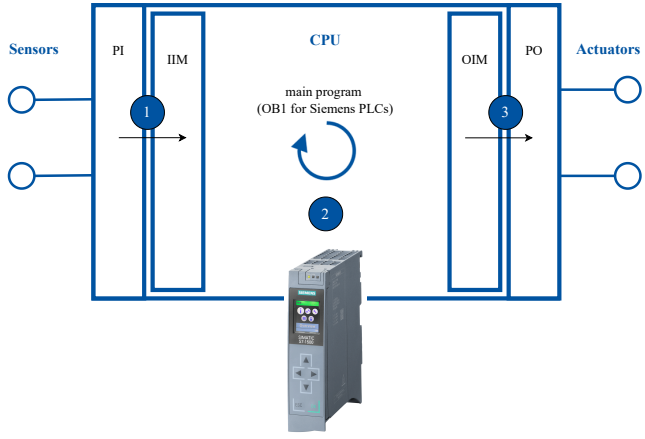
PLC verification at CERN

What we will cover

- What is a PLC?
- How to verify a PLC program?
- Challenges for real projects?

PLC scan cycle and programming languages

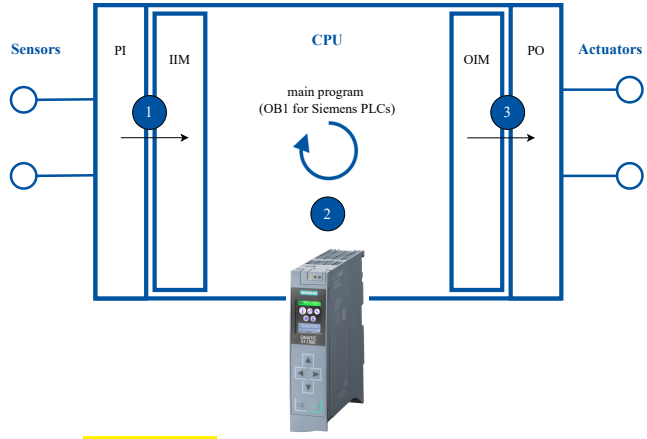
1. Read the sensor value through the input periphery
2. Execute the main program
3. Send order to actuators through the input periphery



PLC scan cycle and programming languages

1. Read the sensor value through the input periphery
2. Execute the main program
3. Send order to actuators through the input periphery

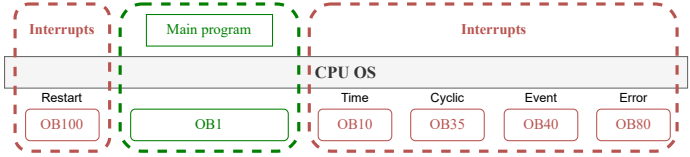
- There are 5 different PLC languages defined by the **IEC 61131-3**



There are also cyclic, event and hardware interrupts, communication routines, etc.

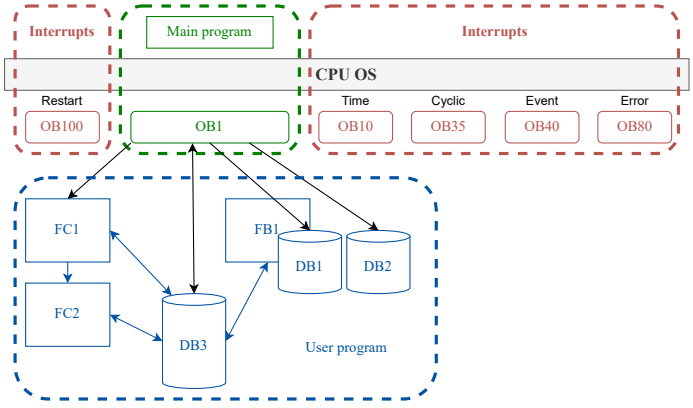
(main) PLC programming blocks

1. Organization Block (OB)



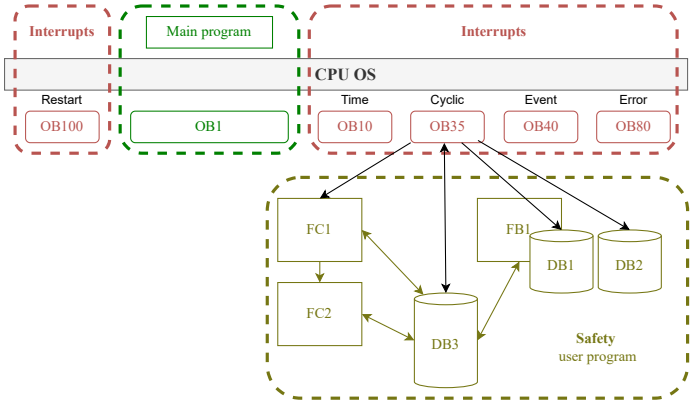
(main) PLC programming blocks

1. Organization Block (OB)
2. Function (FC)
3. Function Block (FB)
4. Data Block (DB) (**global variables**)
   - Instance DB of a FB
   - Global DB



(main) PLC programming blocks

1. Organization Block (OB)
2. Function (FC)
3. Function Block (FB)
4. Data Block (DB) (**global variables**)
   - Instance DB of a FB
   - Global DB



PLC programming languages (IEC 61131-3)

- **Structured Text (ST)**
  ≈ Siemens SCL

```
(* MODE MANAGER *)

IF NOT (#HLD AND #PHLD) THEN

    (* Forced Mode *)
    IF (#AuMoSt_aux OR #MMoSt_aux OR #SoftLDSt_aux) AND
        #E_MFoMoR AND NOT (#AuInhFoMo) 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 #AuInhFoMo 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_MMMoR AND NOT (#AuInhMo) THEN

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

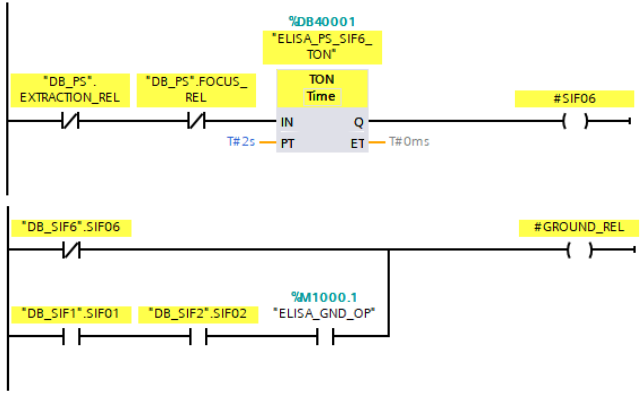
        (* Auto Mode *)
    ELSIF (#MMoSt_aux AND (#E_MAuMoR OR #E_AuAuMoR)) OR
        (#FoMoSt_aux AND #E_MAuMoR) OR
        (#SoftLDSt_aux AND #E_MAuMoR) OR
        (#MMoSt_aux AND #AuInhMo) OR
        (#FoMoSt_aux AND #AuInhFoMo) OR
        (#SoftLDSt_aux AND #AuInhFoMo) OR
        NOT (#AuMoSt_aux OR #MMoSt_aux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

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

    (* Status setting *)
    #LDSt := FALSE;
    #AuMoSt := #AuMoSt_aux;
    #MMoSt := #MMoSt_aux;
    #FoMoSt := #FoMoSt_aux;
    #SoftLDSt := #SoftLDSt_aux;
ELSE
    (* Local Drive Mode *)
    #AuMoSt := FALSE;
    #MMoSt := FALSE;
    #FoMoSt := FALSE;
    #LDSt := TRUE;
    #SoftLDSt := FALSE;
END_IF;
```

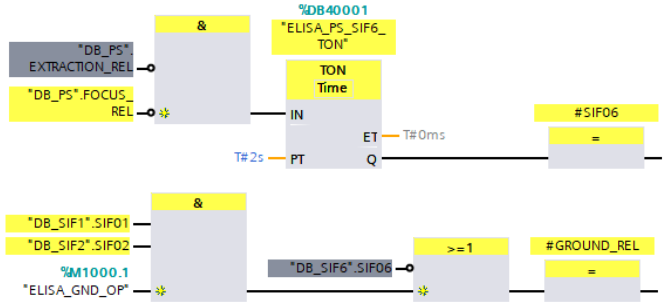
PLC programming languages (IEC 61131-3)

- Structured Text (ST) ≈ Siemens SCL
- **Ladder Diagram (LD)**



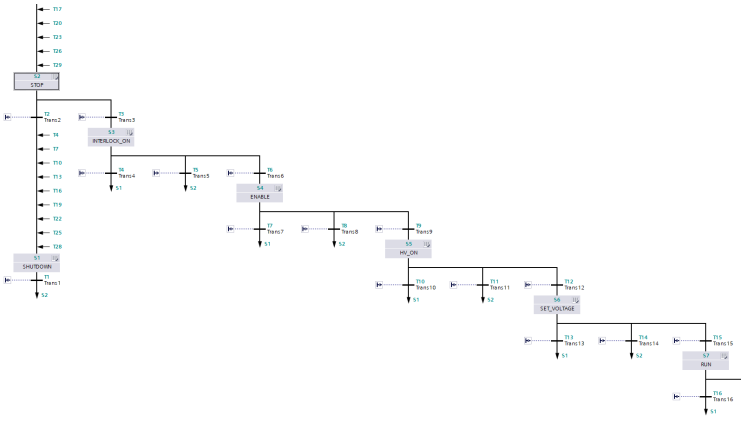
PLC programming languages (IEC 61131-3)

- Structured Text (ST) ≈ Siemens SCL
- Ladder Diagram (LD)
- **Function Block Diagram (FBD)**



PLC programming languages (IEC 61131-3)

- Structured Text (ST) ≈ Siemens SCL
- Ladder Diagram (LD)
- Function Block Diagram (FBD)
- **Sequential Function Chart (SFC)** ≈ Siemens GRAPH

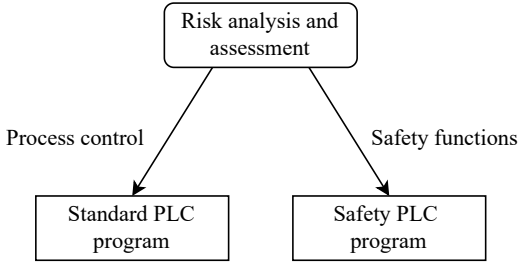


PLC programming languages (IEC 61131-3)

- Structured Text (ST) ≈ Siemens SCL
- Ladder Diagram (LD)
- Function Block Diagram (FBD)
- Sequential Function Chart (SFC) ≈ Siemens GRAPH
- **Instruction List (IL)** ≈ Siemens STL

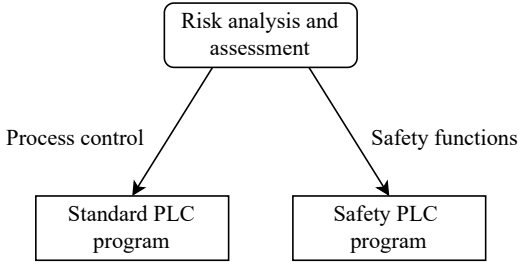
```
NETWORK
TITLE = 'SIF1 logic'
      A     #COM_1;
      A     #CON_A;
      A(    ;
      ON    #TSH1;
      ON    #TSH2;
      ON    #FSL1;
      ON    #FSL2;
      )     ;
      =     #SIF1;
NETWORK
TITLE = 'PC1_PP logic'
      AN    #SIF1;
      AN    #SIF2;
      AN    #SIF4;
      A     #PC1_OPER;
      =     #PC1_PP;
END_FUNCTION
```

Standard and Safety PLC programs



Standard and Safety PLC programs

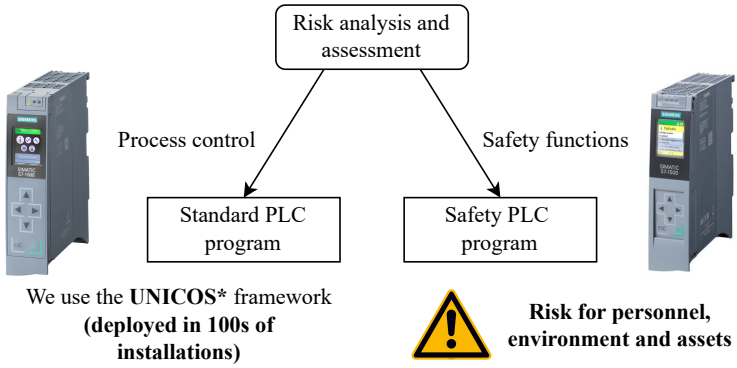




We use the **UNICOS*** framework **(deployed in 100s of installations)**

*unicos.web.cern.ch

Standard and Safety PLC programs



*unicos.web.cern.ch

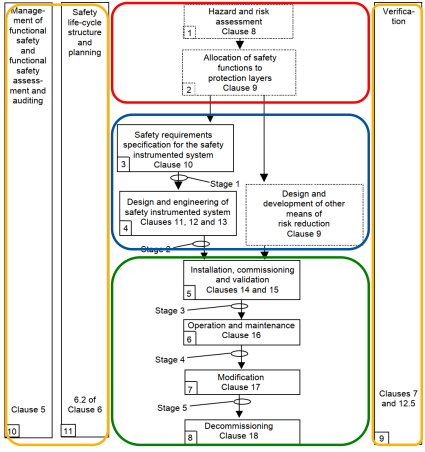
The **PLC program logic** must be compliant with the **specifications**

Functional Safety standards

- Provide **guidelines** to **manage** such projects and **requirements** to **design**, **develop** and **validate** safety systems
- IEC 61508, **IEC 61511**, IEC 62061, etc.

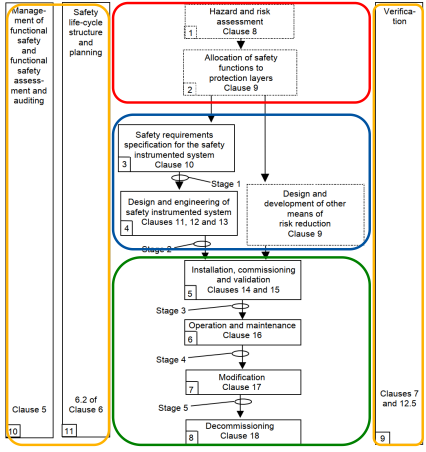
Functional Safety standards

- Provide **guidelines** to **manage** such projects and **requirements** to **design**, **develop** and **validate** safety systems
- IEC 61508, **IEC 61511**, IEC 62061, etc.
- IEC 61511 safety life-cycle:
  - Risk analysis and assessment
  - Design and development of the safety system
  - Commissioning, operation, etc.
  - Management and verification



Functional Safety standards

- Provide **guidelines** to **manage** such projects and **requirements** to **design**, **develop** and **validate** safety systems
- IEC 61508, **IEC 61511**, IEC 62061, etc.
- IEC 61511 safety life-cycle:
  - Risk analysis and assessment
  - Design and development of the safety system
  - Commissioning, operation, etc.
  - Management and verification
- Requirements for the safety system:
  - Hardware
  - Architecture
  - **Software** (Application program)
  - Testing and validation



Functional Safety standards about safety programs

IEC 61508-3: Software requirements

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 61511-2 (Annex B): Guidelines for the application of IEC 61511-1

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

How can we apply model checking to PLC programs?

PLC program

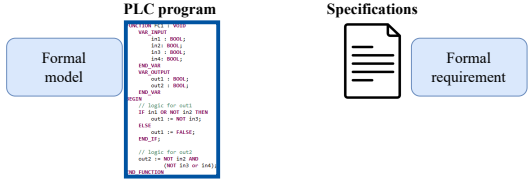
```
FUNCTION PLC : VOID
   VAR_INPUT
        in1 : BOOL;
        in2: BOOL;
        in3 : BOOL;
        in4: BOOL;
   END_VAR
   VAR_OUTPUT
        out1 : BOOL;
        out2 : BOOL;
   END_VAR
BEGIN
   // logic for out1
   IF in1 OR NOT in2 THEN
        out1 := NOT in3;
   ELSE
        out1 := FALSE;
   END_IF;

   // logic for out2
   out2 := NOT in2 AND
           (NOT in3 or in4);
END_FUNCTION
```

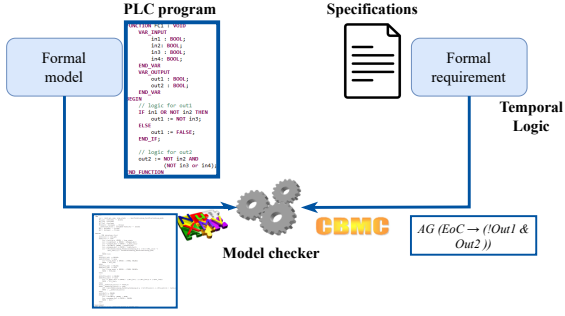
Specifications



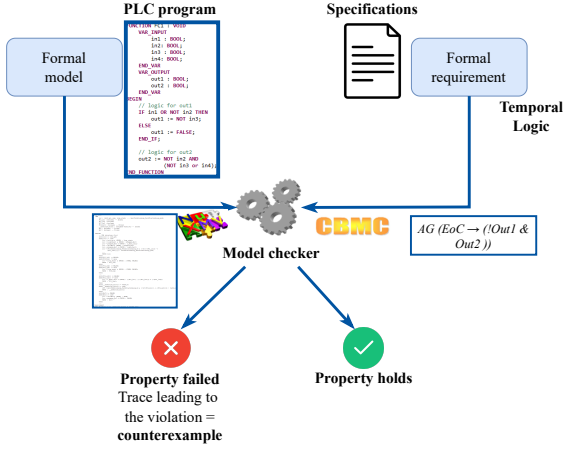
How can we apply model checking to PLC programs?



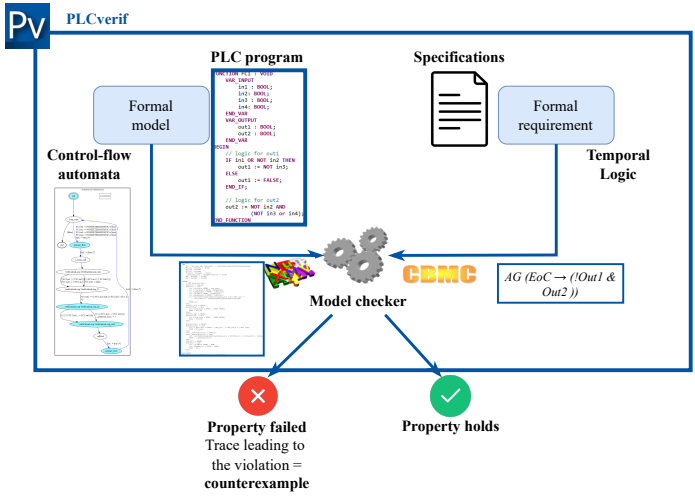
How can we apply model checking to PLC programs?



How can we apply model checking to PLC programs?



How can we apply model checking to PLC programs?



PLCverif workflow

PLC program

```
FUNCTION FC1 : VOID
    VAR_INPUT
        in1 : BOOL;
        in2: BOOL;
        in3 : BOOL;
        in4: BOOL;
    END_VAR
    VAR_OUTPUT
        out1 : BOOL;
        out2 : BOOL;
    END_VAR
BEGIN
    // logic for out1
    IF in1 OR NOT in2 THEN
        out1 := NOT in3 OR in4;
    ELSE
        out1 := FALSE;
    END_IF;

    // logic for out2
    out2 := (in1 OR NOT in2) AND
            (NOT in3 or in4);

END_FUNCTION
```

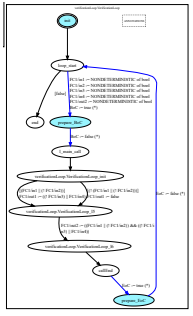
IEC 61131-3 languages (manufacturer variations of) ST, IL, FBD, LD

Specification

- Assertion
- Pattern
- FRETish

via FRET

CFA model
+ model **reductions**
+ **property** parsing / formalization



CTLSPEC *AG((EoC & !out1) -> out2)*

Verification

CBMC

Theta

ESBMC

Variety of external tools

Reporting

Counterexample

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

User interaction PLCverif internals

PLCverif workflow

PLC program

```
FUNCTION FC1 : VOID
    VAR_INPUT
        in1 : BOOL;
        in2: BOOL;
        in3 : BOOL;
        in4: BOOL;
    END_VAR
    VAR_OUTPUT
        out1 : BOOL;
        out2 : BOOL;
    END_VAR
BEGIN
    // logic for out1
    IF in1 OR NOT in2 THEN
        out1 := NOT in3 OR in4;
    ELSE
        out1 := FALSE;
    END_IF;

    // logic for out2
    out2 := (in1 OR NOT in2) AND
            (NOT in3 or in4);

END_FUNCTION
```

IEC 61131-3 languages
(manufacturer variations of)
ST, IL, FBD, LD

Specification

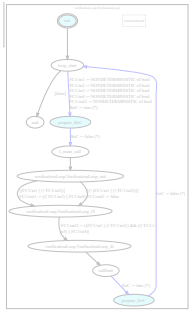
Assertion

Pattern

FRETish

via FRET

CFA model
+ model **reductions**
+ **property** parsing / formalization



CTLSPEC: *AG((EoC & !out1) -> out2)*

Verification

CBMC

theta

ESBMC

Variety of external tools

Reporting

Counterexample

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

User interaction

PLCverif internals

PLCverif workflow

PLC program

```
FUNCTION FC1 : VOID
    VAR_INPUT
        in1 : BOOL;
        in2: BOOL;
        in3 : BOOL;
        in4: BOOL;
    END_VAR
    VAR_OUTPUT
        out1 : BOOL;
        out2 : BOOL;
    END_VAR
BEGIN
    // logic for out1
    IF in1 OR NOT in2 THEN
        out1 := NOT in3 OR in4;
    ELSE
        out1 := FALSE;
    END_IF;

    // logic for out2
    out2 := (in1 OR NOT in2) AND
            (NOT in3 or in4);

END_FUNCTION
```

IEC 61131-3 languages
(manufacturer variations of)
ST, IL, FBD, LD

Specification

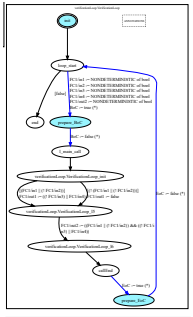
Assertion

Pattern

FRETish

via FRET

CFA **model**
+ model **reductions**
+ **property** parsing / formalization



CTLSPEC *AG((EoC & !out1) -> out2)*

Verification

CBMC

Theta

ESBMC

Variety of external tools

Reporting

Counterexample

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

User interaction

PLCverif internals

PLCverif

- **Open-source platform** for automated verification of **PLC code against property specifications**



PLCverif

- **Open-source platform** for automated verification of **PLC code against property specifications**
- Plugin-based architecture:
  - **Multiple PLC languages** (and PLC brands) can be verified
  - **Several property specification** methods can be integrated
  - **Several third-party verification back-ends** can be integrated



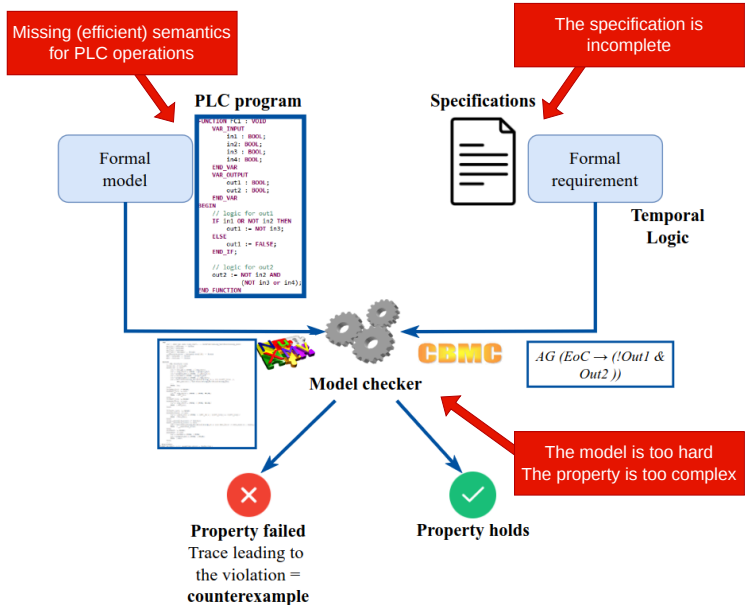
PLCverif



PLCverif: Status of a Formal Verification Tool for Programmable Logic Controller
I. D. López-Miguel et al.
18th ICALEPCS 2021

- **Open-source platform** for automated verification of **PLC code against property specifications**
- Plugin-based architecture:
  - **Multiple PLC languages** (and PLC brands) can be verified
  - **Several property specification** methods can be integrated
  - **Several third-party verification back-ends** can be integrated
- Developed within the **Eclipse** ecosystem: **Java** (Java 11), **EMF** (Eclipse Modeling Framework), **Xtend** and **Xtext**

Problems in verification



Case studies

UNICOS baseline objects



Z. Ádám et al.

From Natural Language Requirements to the Verification of Programmable Logic Controllers: Integrating FRET into PLCverif

15th NFM 2023

SPS Personnel Protection System



B. Fernández Adiego et al.

Applying model checking to highly-configurable safety critical software: the SPS-PPS PLC program

18th ICALEPCS 2021

GSI-FAIR Personnel Access System



I. D. López-Miguel et al.

Working Together for Safer Systems: A Collaboration Model for Verification of PLC Code

19th ICALEPCS 2023

ELISA Safety System



X. Fink et al.

Verifying PLC Programs via Monitors: Extending the Integration of FRET and PLCverif

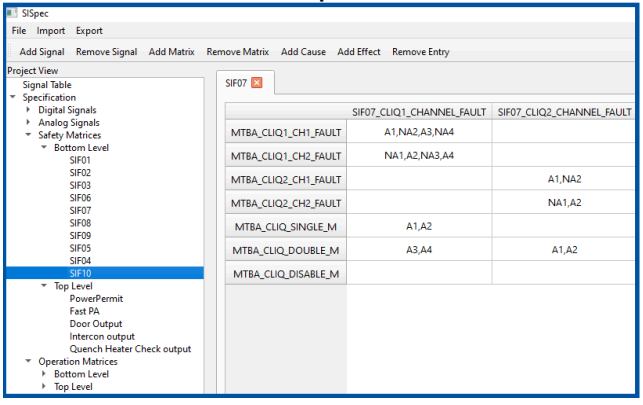
NFM 2024 .

SM18 project(s)



SM18 verification properties

SISpec



B. Fernández Adiego et al.
Cause-and-Effect Matrix Specifications for Safety Critical Systems at CERN
17th ICALEPCS 2019

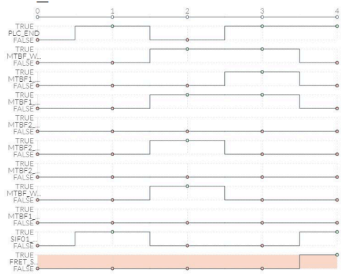
SM18 verification properties

```
F_MAIN shall always satisfy if (PLC_END & (!MTBF_WCC_PCLSW20_FSL | !MTBF_WCC_PCLSW20_TSH |
      (MTBF1_LSW20_POS & (!MTBF1_WCC_LSW20_FSL | !MTBF1_WCC_LSW20_TSH)) | (MTBF2_LSW20_POS &
      (!MTBF2_WCC_LSW20_FSL | !MTBF2_WCC_LSW20_TSH)))) then SIF01_DB.SIF01_PC20K
```

(a) Property of SIF01_PC20K in FRETish.

```
G ((PLC_END and ((((!
  MTBF_WCC_PCLSW20_FSL) or
  (! MTBF_WCC_PCLSW20_TSH))
  or (MTBF1_LSW20_POS and
  ((! MTBF1_WCC_LSW20_FSL)
  or (! MTBF1_WCC_LSW20_TSH)
  ))) or (MTBF2_LSW20_POS and
  ((!MTBF2_WCC_LSW20_FSL) or (!
  MTBF2_WCC_LSW20_TSH)))))
  -> SIF01_DB.SIF01_PC20K)
```

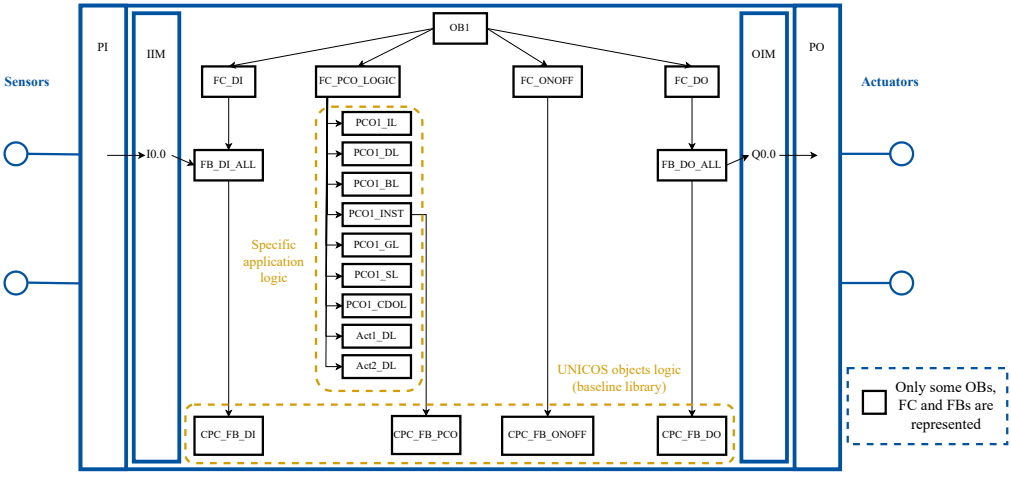
(b) Generated LTL property by FRET.



(c) Future LTL simulation

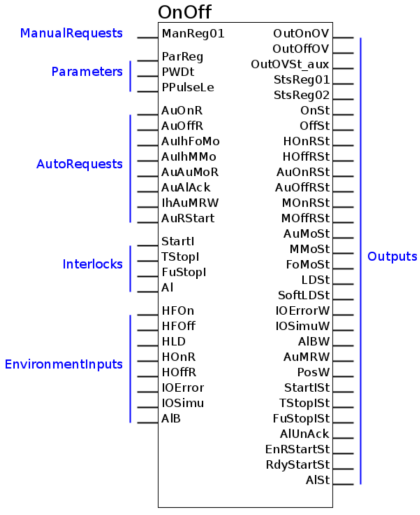
Z. Ádám et al.
From Natural Language Requirements to the Verification of Programmable Logic Controllers: Integrating FRET into PLCverif
15th NFM 2023 .

UNICOS Baseline objects



UNICOS Baseline objects

- Common library to all UNICOS PLC programs
- 23 baseline objects
- Main Function Block and auxiliary Functions
- Complex properties to verify



UNICOS Baseline objects

Metric	OnOff PLC program
PLC language	SCL (ST IEC 61131-3)
Lines of code	$\approx$ 820
Program blocks	1 main FB, 2 timers and 3 FCs
Statements	418
Function calls	21
Input variables	29 (20 BOOL, 3 WORD, 2 ARRAY of 16 BOOL and 4 TIME)
Output variables	31 (27 BOOL, 2 WORD and 2 ARRAY of 16 BOOL)
Internal variables	82 (73 BOOL, 1 TIME, 2 TP timer instances, 1 TON timer instance, 1 REAL and 4 INT)
PLC data types	BOOL, INT, REAL, WORD, TIME, STRUCT and ARRAY
Timers	3 instances

Table: OnOff PLC program metrics

UNICOS Baseline objects

PLC program

CPC_FB_ONOFF.scl

```
(* MODE MANAGER *)

IF NOT (#MLD AND #PHLD) THEN

(* Forced Mode *)
    IF (#AuMoSt_aux OR #PMoSt_aux OR #SoftLDSt_aux) AND
       #_FMoSt AND NOT(#AuIhFoMo) THEN

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

(* Software Local Mode *)
    ELSIF (#AuMoSt_aux OR #PMoSt_aux) AND #E_HSoftLDR AND NOT #AuIhFoMo THEN

        #AuMoSt_aux   := FALSE;
        #PMoSt_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;
        #PMoSt_aux    := TRUE;
        #FoMoSt_aux   := FALSE;
        #SoftLDSt_aux := FALSE;

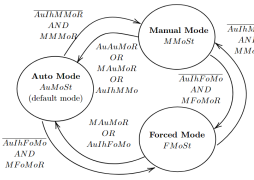
(* Auto Mode *)
    ELSIF (#PMoSt_aux AND (#E_HAuMoR OR #E_AuAuMoR )) OR
          (#FoMoSt_aux AND #E_HAuMoR) OR
          (#SoftLDSt_aux AND #E_HAuMoR) OR
          (#PMoSt_aux AND #AuIhMMo) OR
          (#FoMoSt_aux AND #AuIhFoMo)OR
          (#SoftLDSt_aux AND #AuIhFoMo) OR
          NOT(#AuMoSt_aux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

        #AuMoSt_aux   := TRUE;
        #PMoSt_aux    := FALSE;
        #FoMoSt_aux   := FALSE;
        #SoftLDSt_aux := FALSE;
    END_IF;

    (* Status setting *)
    #LDSt    := FALSE;
    #AuMoSt  := #AuMoSt_aux;
    #PMoSt   := #PMoSt_aux;
```

"Complex" specification properties

Original specification



Extracted properties

When (MMoSt & AuAuMoR) the CPC_FB_OnOff shall eventually satisfy AuMoSt & PLC_END

```
(G (((!(MMoSt and AuAuMoR)) and (X (MMoSt
 and AuAuMoR))) -> (X(F(AuMoSt and
 PLC_END))))) and ((MMoSt and
 AuAuMoR) -> (F(AuMoSt and PLC_END)))
```

the CPC_FB_OnOff shall always satisfy if (MMoSt & AuAuMoR & !PLC_END) then at the next occurrence of PLC_END, AuMoSt

```
G (((MMoSt and AuAuMoR) and (! PLC_END)) ->
 (X ((((! PLC_END) ) U PLC_END) ->
 (((! PLC_END)) U (PLC_END and AuMoSt)))))
```

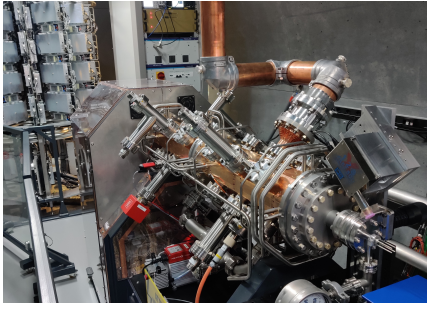
Z. Ádám et al.
From Natural Language Requirements to the Verification of Programmable Logic Controllers: Integrating FRET into PLCverif
15th NFM 2023

ELISA Safety System

Verifying PLC Programs via Monitors: Extending the Integration of FRET and PLCverif
X Fink, A Mavridou, A Katis, BF Adiego
NFM 2024

ELISA (Experimental LInear accelerator for Surface Analysis)

- PLC running a safety program: protection against voltage, etc.



ELISA Safety System



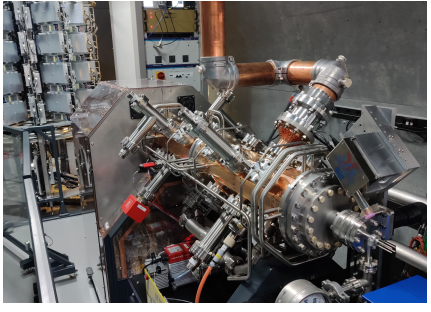
Verifying PLC Programs via Monitors: Extending the Integration of FRET and PLCverif
X Fink, A Mavridou, A Katis, BF Adiego
NFM 2024

ELISA (Experimental LInear accelerator for Surface Analysis)

- PLC running a safety program: protection against voltage, etc.

Challenge: Complex properties with reasoning over PLC time

Upon persisted(2000,!ELISA_RF_PRESSURE_SWSt) the FC_MAIN shall at the next timepoint satisfy (!RF_CMD & !RF_AMP_CMD)



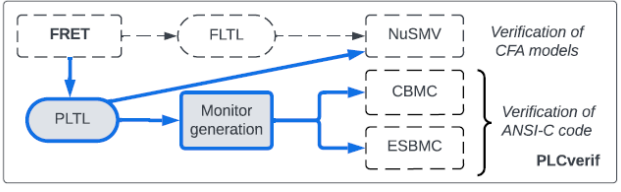
ELISA Safety System

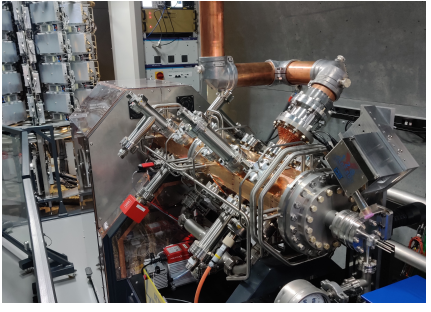
Verifying PLC Programs via Monitors: Extending the Integration of FRET and PLCverif
X Fink, A Mavridou, A Katis, BF Adiego
NFM 2024

ELISA (Experimental LInear accelerator for Surface Analysis)

- PLC running a safety program: protection against voltage, etc.

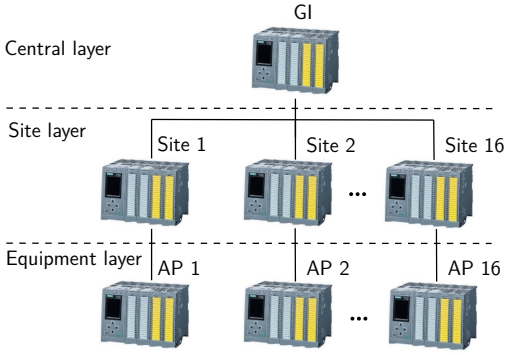
Challenge: Complex properties with reasoning over PLC time





SPS Personnel Protection Systems

- The The Super Proton Synchrotron (**SPS**) is one of the CERN particle accelerators
- The SPS Personnel Protection System is **highly distributed** and **configurable** PLC-based access and safety system
- **Complex module specifications** including many input and **configuration** variables



SPS Personnel Protection Systems

Original module specification

Specification 2 Partial, simplified requirement of SIF-X1.

if (

$$\prod_{i=1}^{N_i} (I_EISa_Pos_Stat[i] \vee I_EISa_Bypass[i])$$
$$\forall i \in [1,N_i] : (I_EISa_Pos[i]=1 \wedge SC\text{-}S[i][i]=1$$

$$\wedge$$

$$\prod_{i=1}^{N_i} (I_EISa_PU_Stat[i] \vee I_EISa_Bypass[i])$$
$$\forall i \in [1,N_i] : (I_EISa_PU[i]=1 \wedge SC\text{-}S[i][i]=1$$

$$\wedge$$

$$S0_AP_Pos[j] \wedge S0_AP_PU[j] \wedge S0_AP_Key_Distrib[j]$$

) **then**

$N_EISa_Safe[j] \leftarrow 1$

else

$N_EISa_Safe[j] \leftarrow 0$

end if

PLCverif property

```
//#ASSERT
(((I_EISa_Pos_Stat OR I_EISa_Bypass) AND
(SC-S_0 AND I_EISa_Pos))
= (SC-S_0 OR I_EISa_Pos)) AND
(((I_EISa_PU_Stat OR I_EISa_Bypass) AND
(SC-S_0 AND I_EISa_PU))
= (SC-S_0 OR I_EISa_PU)) AND
(S0_AP_Key_Distrib.%X0 AND
S0_AP_PU.%X0 AND S0_AP_Pos.%X0)
= N_EISa_Safe.%X0
```

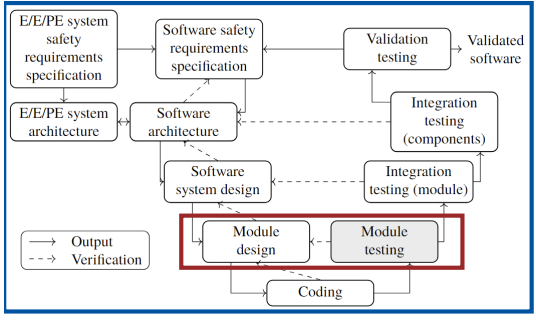
B. Fernández Adiego et al.
Applying model checking to highly-configurable safety critical software: the SPS-PPS PLC program

18th ICALEPCS 2021

SPS Personnel Protection Systems

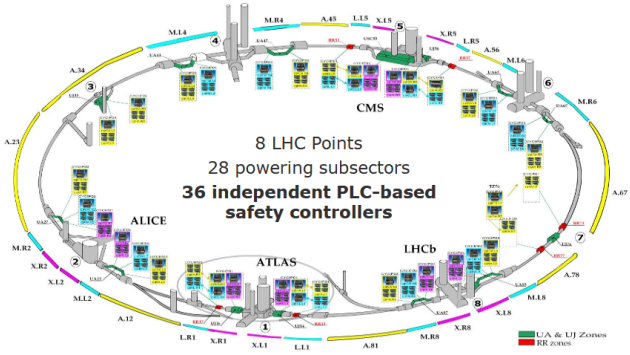
- We applied **PLCverif** to individual program **modules** (FBs) (large number of input and configuration variables)
- Testing was applied to the whole system to validate hardware, software and communications
- **Formal verification** helped our engineers to improve the specification and to **better understand the program behavior**

V-model: Formal verification of PLC modules



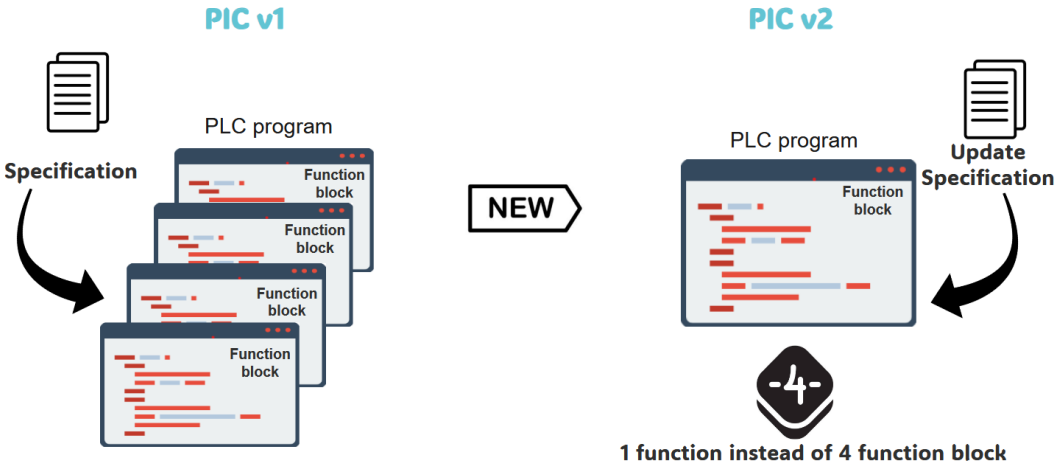
The Powering Interlock Controller (PIC)

- Central safety system ensuring the safe powering of the LHC superconducting circuits



- Authorize powering only when all safety conditions are fulfilled
- Continuously supervise the powering state
- Request a beam dump through the Beam Interlock System whenever required

Upgrade of the PIC PLC controls

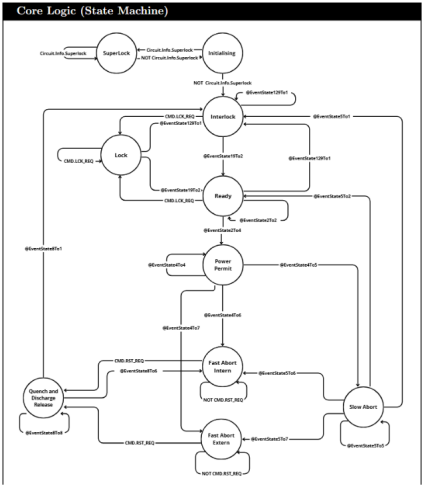


Formal verification of the new PIC controls

Formal Verification and Specification of the PIC PLC program
Mohammad Massi Rashidi
Master's thesis project

PLCverif helps with the requirements engineering

- Equivalence checking
- Specification updates



Remaining challenges

- PLCverif itself is not verified
  - no formal semantics of all intermediate models and transformations

Remaining challenges

- PLCverif itself is not verified
  - no formal semantics of all intermediate models and transformations
- Large PLC programs do not work out-of-the-box
  - parsing of language features
  - abstraction of proprietary functions
  - state-space explosion

Remaining challenges

- PLCverif itself is not verified
  - no formal semantics of all intermediate models and transformations
- Large PLC programs do not work out-of-the-box
  - parsing of language features
  - abstraction of proprietary functions
  - state-space explosion
- PLCverif still mainly used by verification experts
  - model checking = push button, but model creation is hard
  - defining properties correctly
  - going from PLC software to PLCverif

C++ Component Verification

Front-end computers at CERN

PLCs cannot be used for everything at CERN

- Compute capabilities
- Radiation exposure
- ...



Figure: Siemens FEC

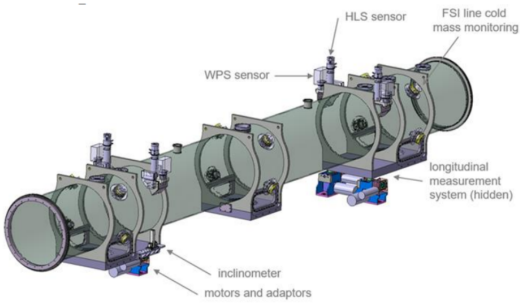
Verification challenges: larger systems, complex software, ...

Can we use existing tools?

Case study: FRAS Protection Layer

Full Remote Alignment System (FRAS) for remote alignment of HL-LHC magnets

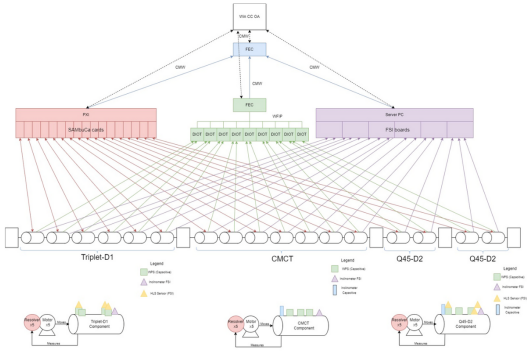
- 17 components
- 21 bellows
- Sensors:
  - 54 WPS
  - 10 Cap. Inclin.
  - 27 HLS FSI
  - 16 FSI Inclin.
- 85 motors
- 85 resolvers
- Various controllers



Case study: FRAS Protection Layer

Full Remote Alignment System (FRAS) for remote alignment of HL-LHC magnets

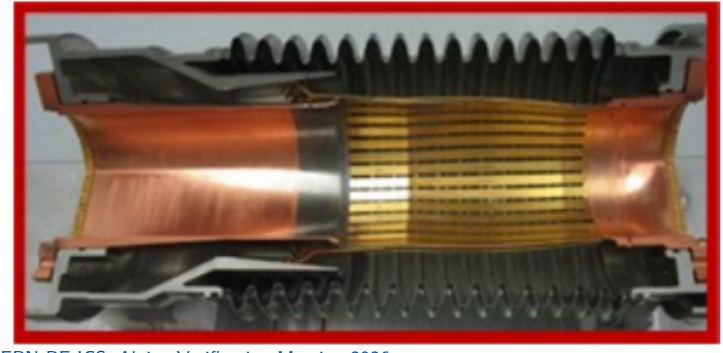
- 17 components
- 21 bellows
- Sensors:
  - 54 WPS
  - 10 Cap. Inclin.
  - 27 HLS FSI
  - 16 FSI Inclin.
- 85 motors
- 85 resolvers
- Various controllers



Risk and Protection Layer

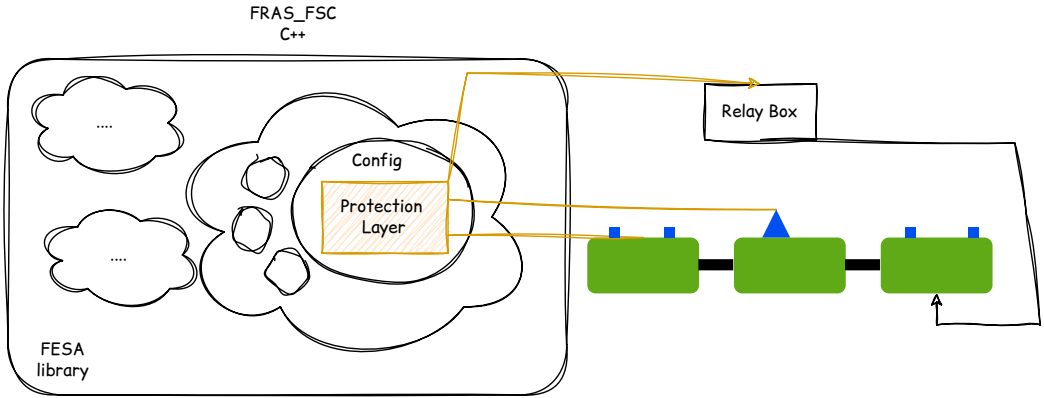
A failure in the FRAS control system can drive an excessive relative displacement between neighbouring components.

- Damage to interconnecting bellows
- Potential consequence of up to a year of LHC stop.



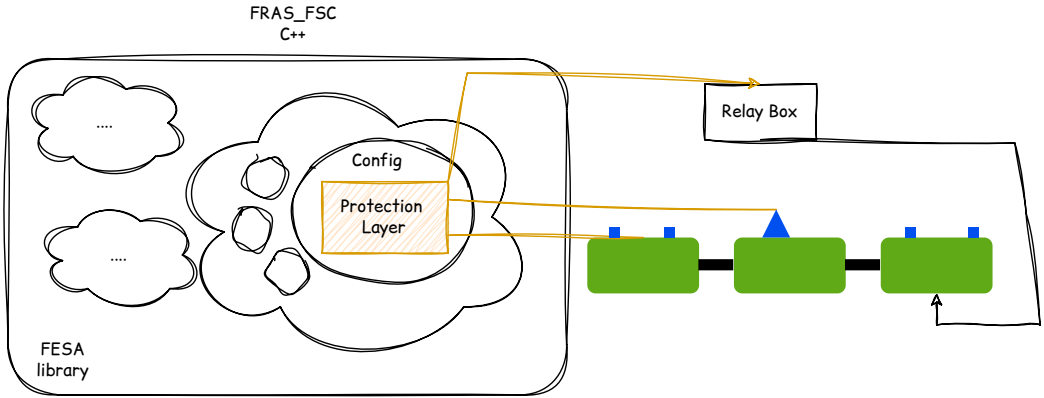
Risk and Protection Layer

- Software protection layer within framework
- Independently calculate position and stop motors when risk occurs



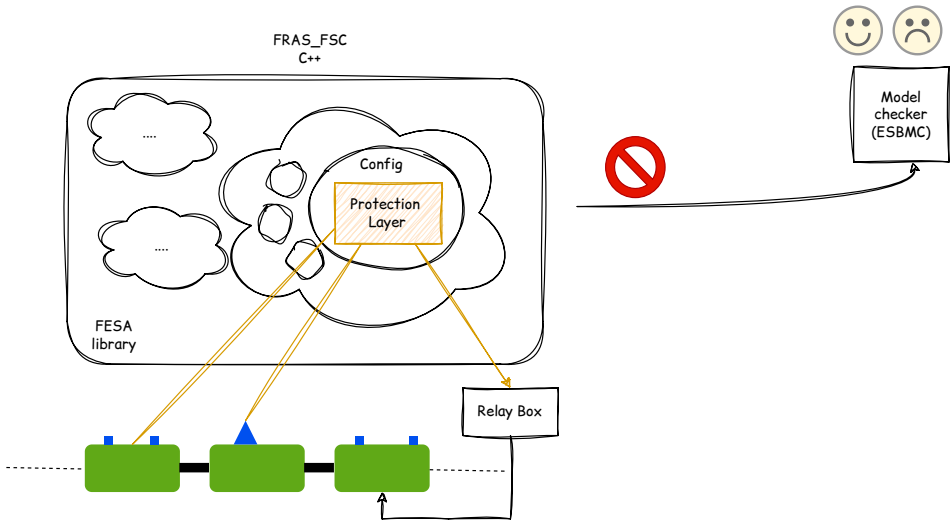
Risk and Protection Layer

- Software protection layer within framework
- Independently calculate position and stop motors when risk occurs
- Architecture constraints: protection layer needs to be embedded into functional software



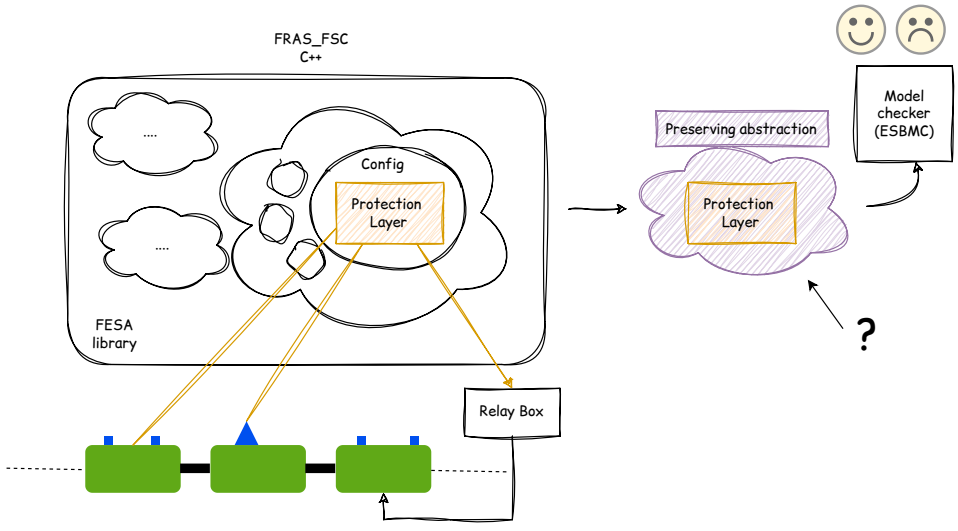
Protection-layer verification

Improving Software Quality and CI/CD Workflows for FRAS
Thanapat Kemthong, Ricardo Ampudia
Internal project



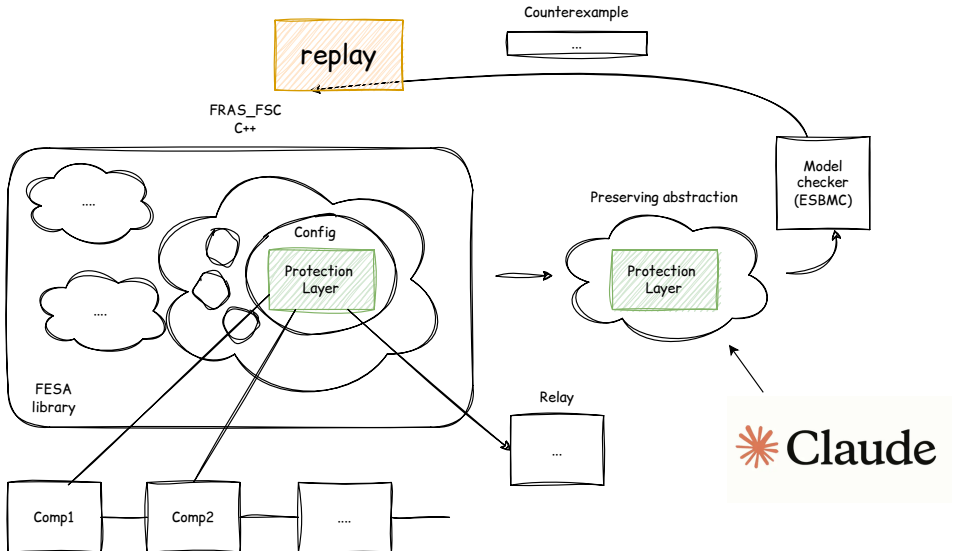
Protection-layer verification

Improving Software Quality and CI/CD Workflows for FRAS
Thanapat Kemthong, Ricardo Ampudia
Internal project



Protection-layer verification

Improving Software Quality and CI/CD Workflows for FRAS
Thanapat Kemthong, Ricardo Ampudia
Internal project



LLM Generated Library Abstractions

Library shims

FsiExamplePLLogic__execute.cpp

```
#include <deps1>
#include <deps2>

#include ".../FsiExamplePLLogic.h"

/* Harness helpers */
struct {};
namespace FsiRack {
    void func1(...) {....}
}
void func2(...) {...}

#include ".../FsiExamplePLLogic.cpp"

int main() {
    FsiExamplePL::Device device;

    const int exampleProperty1 = __VERIFIER_nondet_int();
    __VERIFIER_assume(exampleProperty1 >= 0 && exampleProperty1 <= 100);
    device.property1Int = exampleProperty1;
    ...

    FsiRack::examplePL::execute(&device, ...);

    __ESBMC_assert(some_property, "Some Property Violated");
}
```

→ Library shims + Shadow headers

formal-verification/include

cmw-fwk/client/rbac	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
cmw-log	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
fesa-core	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
algorithm	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
bdaqctrl.h	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
chrono	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
fsi-edfa.h	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
fsi-laser.h	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
fsi.h	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
fsi_tmr.h	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
functional	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
memory	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
mutex	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
numeric	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
shadow_config.schema.json	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
shadow_header.py	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
shadow_header_config.schema.json	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
shadow_stub.py	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
sstream	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
stdexcept	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
stdio.h	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
string	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
tuple	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
unistd.h	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
vector	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago
wchar.h	feat: import the shared engine content (shims, generators, probes, templat...	6 days ago

Takeaways from this experiment

- C++ verification tools can help in projects similar as for PLC verification
  - Improved formal specification
  - Revealing bugs in configuration and implementation
- LLMs can make these tools usable for non-experts
  - Speeding up modeling time
  - Fully automated GitLab pipelines

Takeaways from this experiment

- C++ verification tools can help in projects similar as for PLC verification
  - Improved formal specification
  - Revealing bugs in configuration and implementation
- LLMs can make these tools usable for non-experts
  - Speeding up modeling time
  - Fully automated GitLab pipelines

Future direction

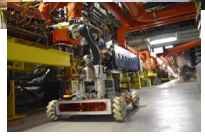
- How to trust that the LLM generated abstraction does not remove relevant code?

Verification of neural networks and AI-enabled control

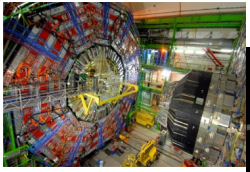
AI at CERN

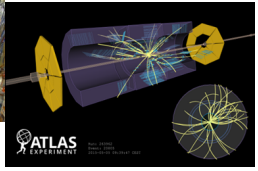




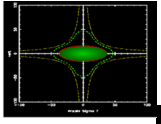


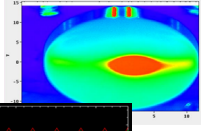
Robotics and Mechatronics

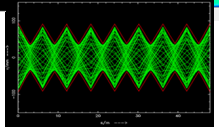




Data analysis of the physics experiments







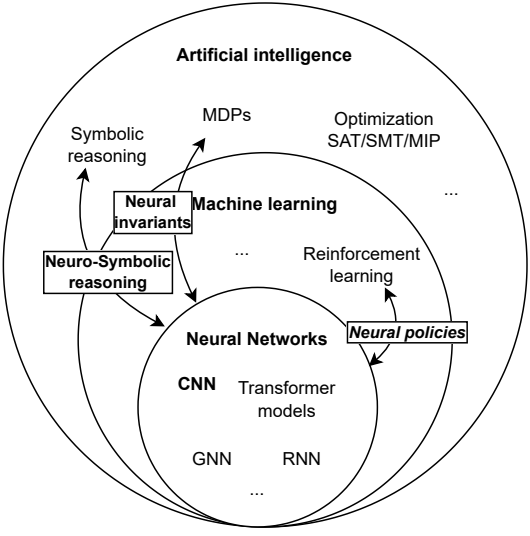
Particle accelerators operation and **beam optimization**





Industrial controls

Neural networks



Safety-critical control systems with neural network components

Many of these systems are safety-critical!

- Neural network adoption not feasible unless risk can be quantified
- CERN AI strategy includes research on trustworthy and explainable AI

Functional-safety standards

Existing standards are not applicable to neural-networks. Emerging:
ISO/PAS 8800 (automotive), ISO/IEC 24029, ISO/IEC 5469 (general)

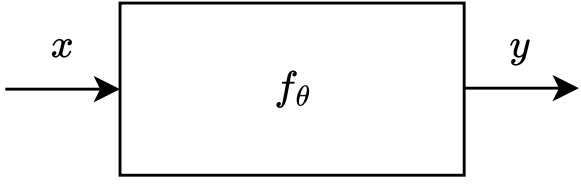
What we will cover

- What is neural network verification?
- What are we doing at CERN?
  - Robustness verification of time series data
    - Crystal collimator alignment case study
  - Safety/Stability verification of closed-loop control
    - Industrial controls HVAC/cooling-tower case studies

Feed-forward Neural network verification

What is a neural network?

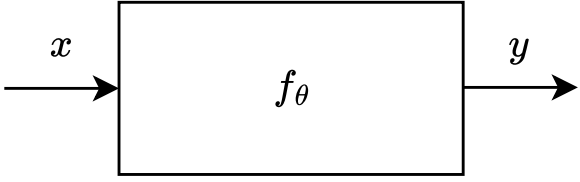
- Nonlinear layered parameterized function $\mathbb{R}^n \to \mathbb{R}^{\geq}$
- Usually optimizable by gradient descent



Feed-forward Neural network verification

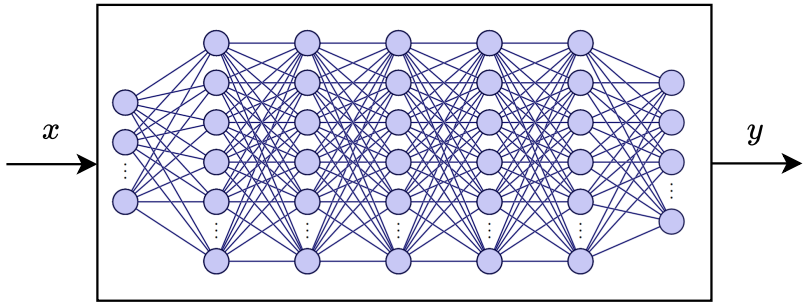
What is a neural network?

- Nonlinear layered parameterized function $\mathbb{R}^n \to \mathbb{R}^{>}$
- Usually optimizable by gradient descent



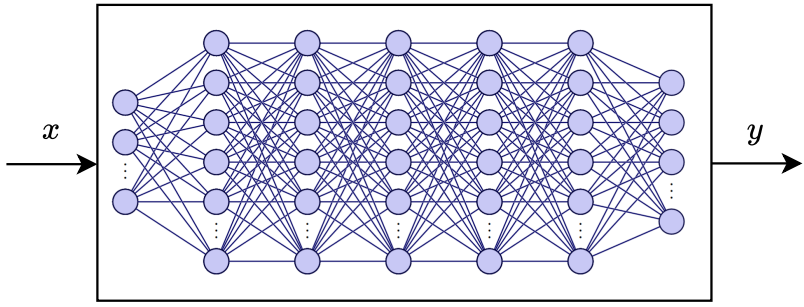
- Properties: reachability, robustness, fairness $\to$ simple

Feed-forward Neural network verification



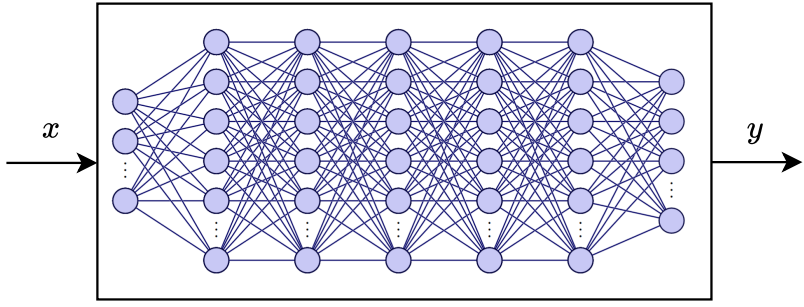
- Properties: reachability, robustness, fairness $\rightarrow$ simple
- Functions:
  - Linear layers: $f_{\text{linear}}(x) = Ax + b$ *(A and b are learned)*
  - Nonlinear layers: e.g. $f_{\text{ReLU}}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$

Feed-forward Neural network verification



- Properties: reachability, robustness, fairness $\rightarrow$ simple
- Functions:
  - Linear layers: $f_{\text{linear}}(x) = Ax + b$ *(A and b are learned)*
  - Nonlinear layers: e.g. $f_{\text{ReLU}}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$

Feed-forward Neural network verification



Even simple properties can be extremely hard to answer

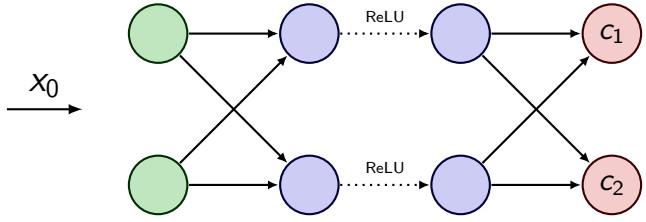
- Reachability is NP-complete [Saelzer & Lange, 2021]

Pointwise Robustness Problem

Neural Network Classification

Input: Vector x_0 and set of possible classes $\mathcal{C} = \{c_1, \ldots, c_n\}$

$$f(x_0) = y = (y_1, \ldots, y_n)^T \rightarrow c^{x_0} = c_{\arg\max y_i}$$

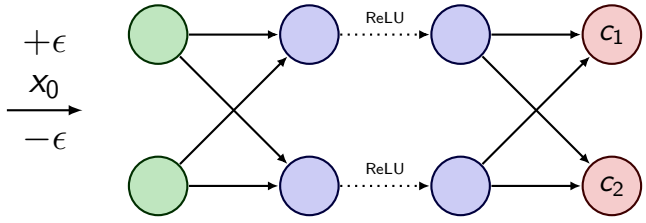


Pointwise Robustness Problem

Neural Network Classification

Input: Vector x_0 and set of possible classes $\mathcal{C} = \{c_1, \ldots, c_n\}$

$$f(x_0) = y = (y_1, \ldots, y_n)^T \rightarrow c^{x_0} = c_{\arg\max y_i}$$



ϵ-region to model perturbation

Pointwise Robustness Problem

Neural Network Classification

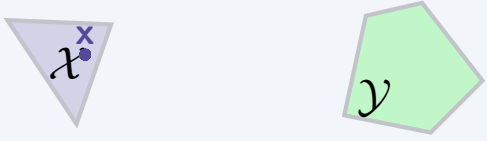
Input: Vector x_0 and set of possible classes $\mathcal{C} = \{c_1, \ldots, c_n\}$

$$f(x_0) = y = (y_1, \ldots, y_n)^T \rightarrow c^{x_0} = c_{\arg\max y_i}$$

Local Adversarial Robustness Problem

Input: Tuple (f, x, ϵ, C, i^*)

$$\mathcal{X} = \{x : ||x - x_0||_p < \epsilon\} \qquad \mathcal{Y} = \{y : y_{i^*} > y_j, \forall j \neq i^*\}$$



Problem: Decide $\quad x \in \mathcal{X} \Rightarrow y = f(x) \in \mathcal{Y}$

Neural network verification methods



Neural Network Verification Tutorial
Huan Zhang, Kaidi Xu, Shiqi Wang and Cho-Jui Hsieh
https://neural-network-verification.com/

Optimization-based

- (MI)LP, SMT, CHC
- Complete
- Generally do not scale

Abstract domain propagation

- (partial) polyhedra (CROWN), star-set, polynomial
- Incomplete

Abstract propagation + search + optimization

- Combine abstract domains with optimization and search (BaB)
- State-of-the-art
- Computationally expensive

Verification using Reachability

Setup: Input x, Perturbation area $\mathcal{X}$, Correct classification area $\mathcal{Y}$

Goal: Compute reachable set $\mathcal{R}$ or overapproximation $\mathcal{R}^*$

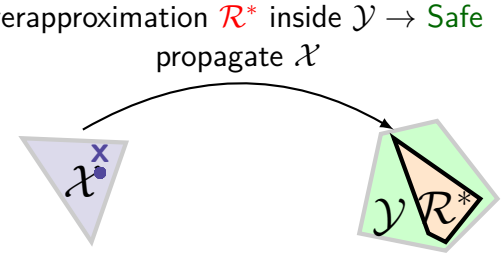




Verification using Reachability

Setup: Input x, Perturbation area $\mathcal{X}$, Correct classification area $\mathcal{Y}$

Goal: Compute reachable set $\mathcal{R}$ or overapproximation $\mathcal{R}^*$



- Different abstract domains
- Optimizations to reduce overapproximation

Refutation via Gradient Descent

Setup: Input x, Perturbation area $\mathcal{X}$, Correct classification area $\mathcal{Y}$

Goal: Find $\hat{x} \in \mathcal{X}$ with $f(\hat{x}) \notin \mathcal{Y}$

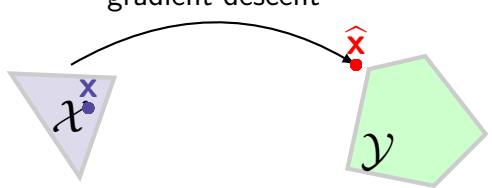


Refutation via Gradient Descent

Setup: Input x, Perturbation area $\mathcal{X}$, Correct classification area $\mathcal{Y}$

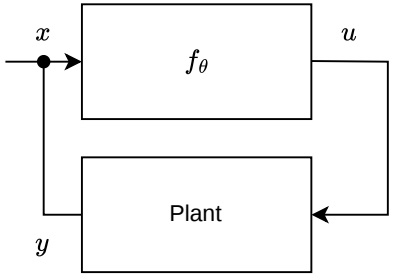
Goal: Find $\hat{x} \in \mathcal{X}$ with $f(\hat{x}) \notin \mathcal{Y}$

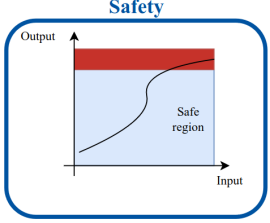
Adversarial example $\hat{x}$ outside $\mathcal{Y} \rightarrow$ Unsafe

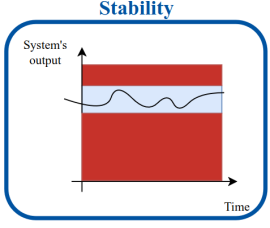


- Incomplete!

Neural networks in closed-loop control



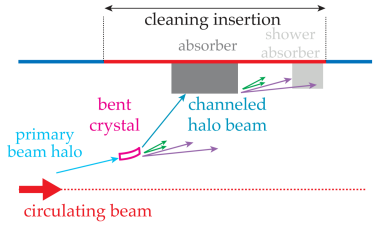




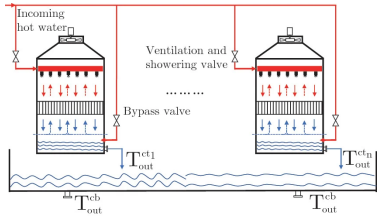
- Bounded reachability, inductive invariants (certificates)
- Reuses local neural network verification methods + domains to reason about plant dynamics (e.g. Flow*)

Our work on it at CERN

1. Beam collimator alignment

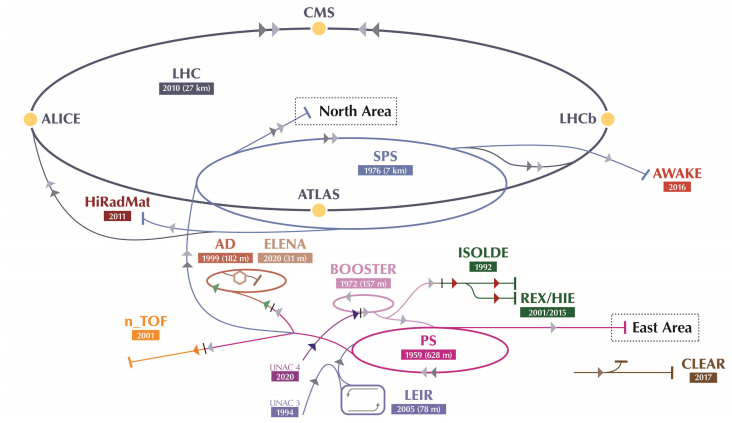


2. Cooling towers / HVAC control



Case study: Neural network based crystal collimator alignment

Context: Particle beam cleaning (collimation)



Case study: Neural network based crystal collimator alignment

Context: Particle beam cleaning (collimation)

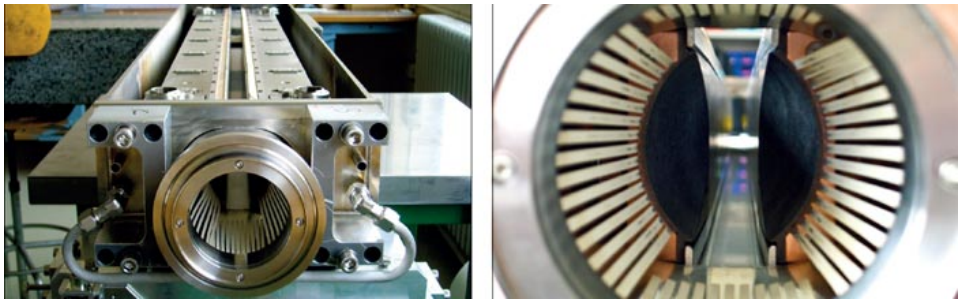
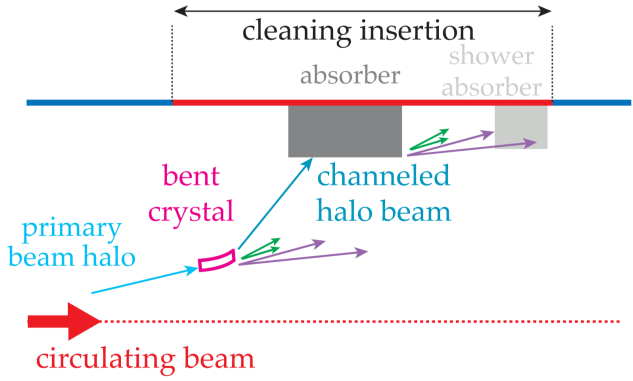


Figure: https://cerncourier.com/a/the-collimation-system-defence-against-beam-loss/

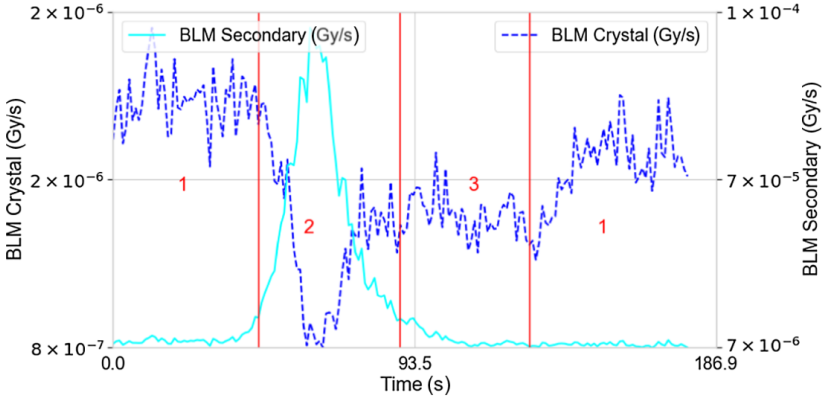
Case study: Neural network based crystal collimator alignment



Nerual network based alignment

Machine learning based crystal collimator alignment optimization
G Ricci et al
APS 2024

Alignment of the crystal based on beam loss monitor (BLM) signals

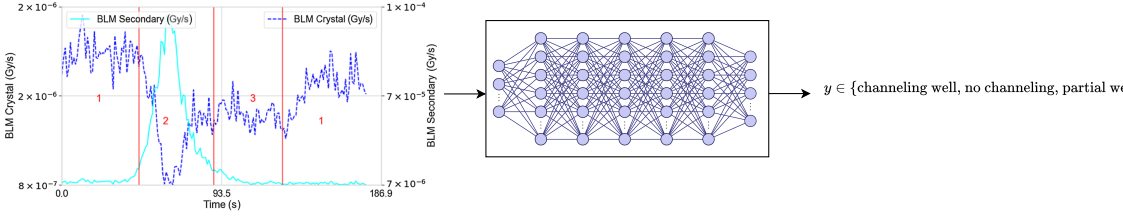


Nerual network based alignment

Machine learning based crystal collimator alignment optimization
G Ricci et al
APS 2024

Alignment of the crystal based on beam loss monitor (BLM) signals

- Windows of length n are classified by a neural network

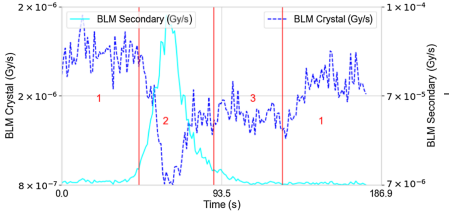


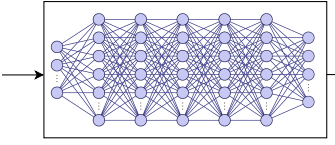
Nerual network based alignment

Machine learning based crystal collimator alignment optimization
G Ricci et al
APS 2024

Alignment of the crystal based on beam loss monitor (BLM) signals

- Windows of length n are classified by a neural network
- Iterative procedure
  1. Rotate crystal between angle $[\theta, \theta + \delta]$
  2. Collect beam loss monitor feedback and slice into windows
  3. Preprocess and classify each window with the CNN





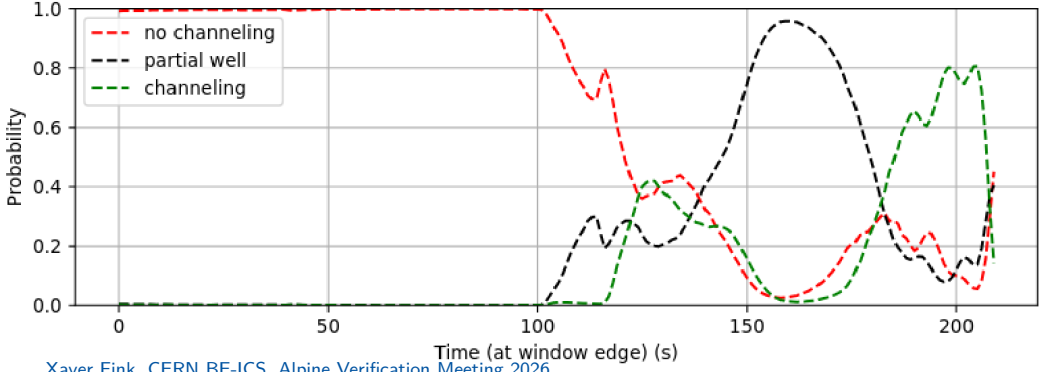
$y \in \{$channeling well, no channeling, partial we

Nerual network based alignment

Machine learning based crystal collimator alignment optimization
G Ricci et al
APS 2024

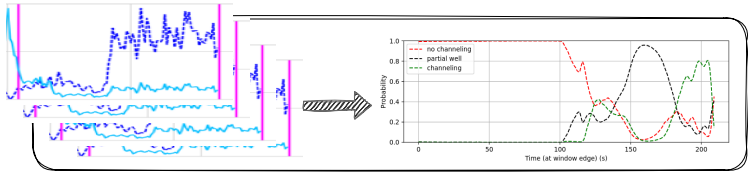
Alignment of the crystal based on beam loss monitor (BLM) signals

- Windows of length n are classified by a neural network
- k window classifications accumulate into classification graph



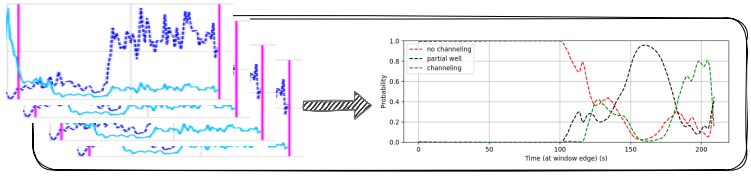
Verification problems in two layers

1. **Data layer:** Neural network predicts on data

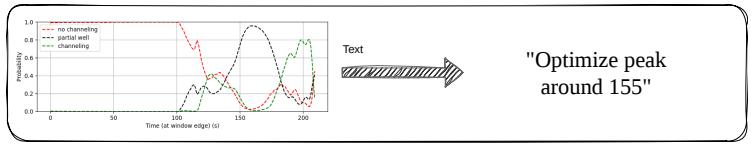


Verification problems in two layers

1. **Data layer:** Neural network predicts on data



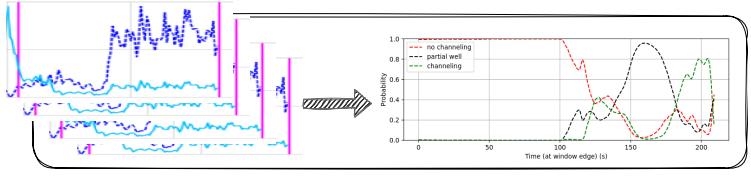
2. **Decision layer:** Human or simple rules evaluates temporal data



Verification problems in two layers

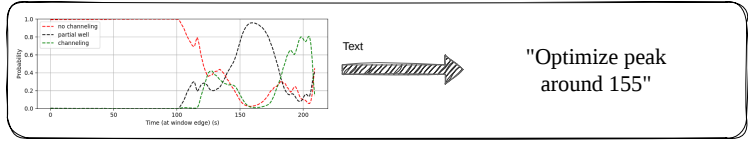
1. **Data layer:** Neural network predicts on data

Property: NN robustness against noise (local adversarial robustness)



2. **Decision layer:** Human or simple rules evaluates temporal data

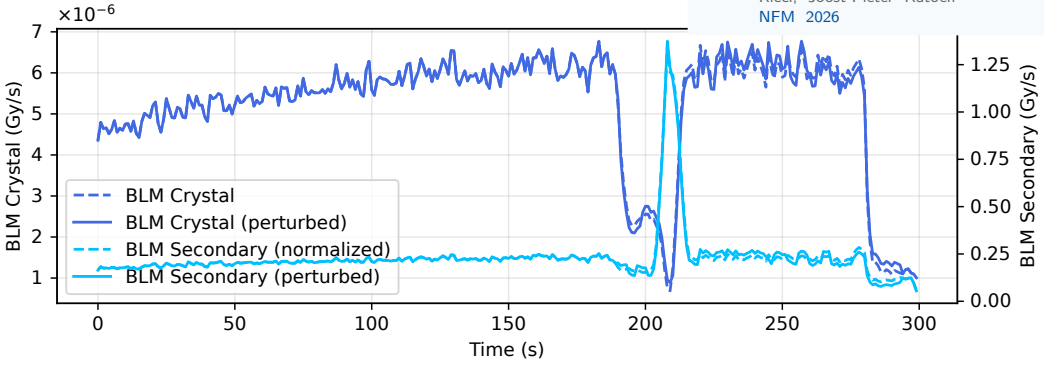
Property: Satisfaction of temporal logic specifications



1. Local adversarial robustness

Question: How robust is the neural network to signa

Adversarial Robustness of Time-Series Classification for Crystal Collimator Alignment
Xaver Fink, Borja Fernández Adiego, Daniele Mirarchi, Eloise Matheson, Álvaro García Gonzáles, Gianmarco Ricci, Joost-Pieter Katoen
NFM 2026

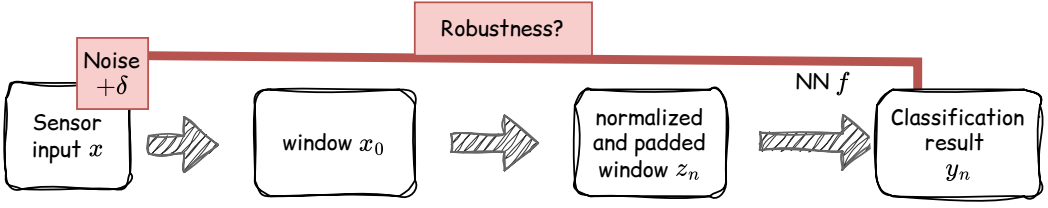


1. Local adversarial robustness

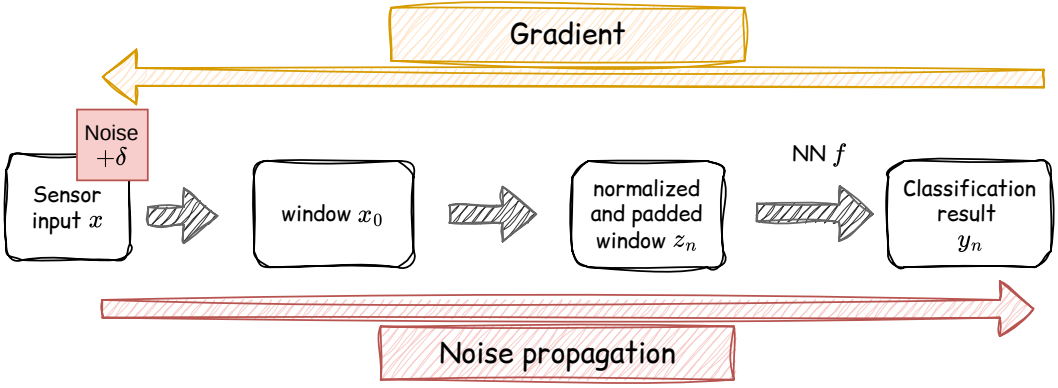
Adversarial Robustness of Time-Series Classification for Crystal Collimator Alignment
Xaver Fink, Borja Fernández Adiego, Daniele Mirarchi, Eloise Matheson, Álvaro García Gonzáles, Gianmarco Ricci, Joost-Pieter Katoen
NFM 2026

Question: How robust is the neural network to signa

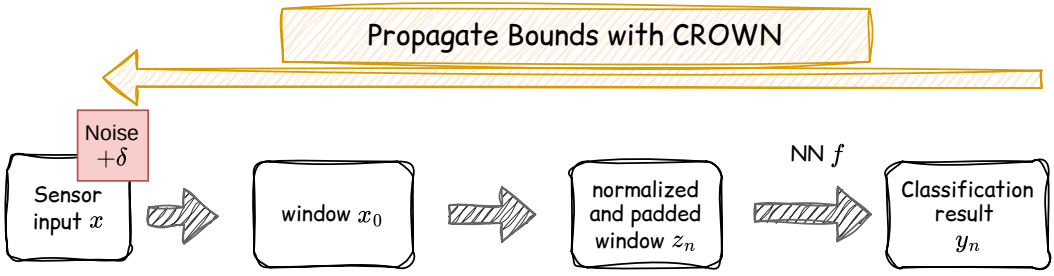
Problem: Preprocessing steps between signal and neural network



Finding counterexamples



Verifying robustness



Problem: large overapproximation because of $N(x) = \frac{x-\mu(x)}{\sigma(x)}$

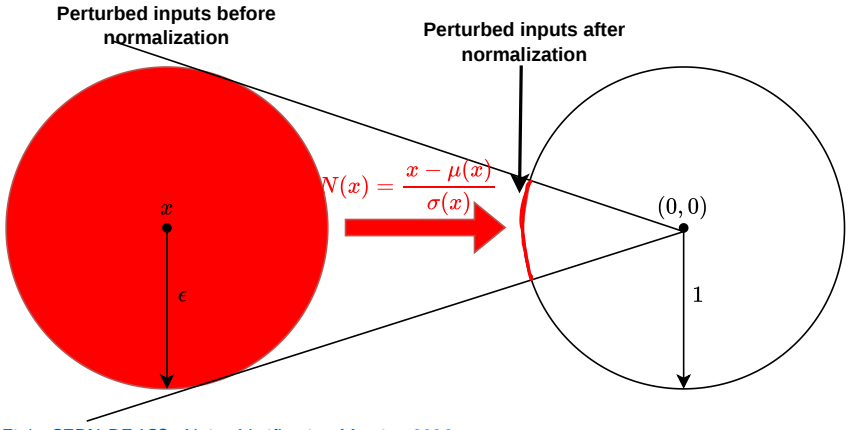
- cannot be bounded well via partial polyhedra
- more problems because of numerical instability

Verification over normalization

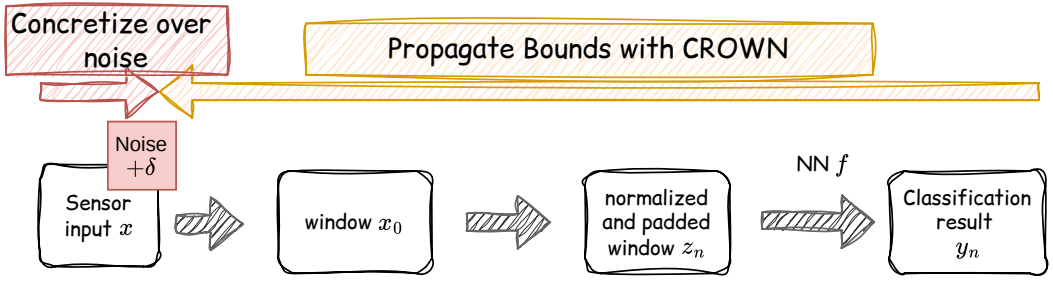
Paper under review
Xaver Fink, Borja Fernandez Adiego, Joost-Pieter-Katoen
Under review

Idea: Use our knowledge about the normalization function

- induces mean of zero, variance of one



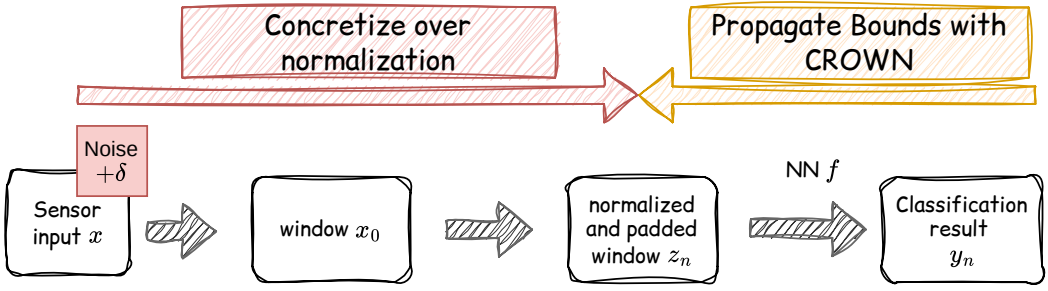
Concretization-based verification



Verification procedure:

- Propagation of bounds towards x: $g(x) \geq a^\top x + b$
- Concretize over noise-set: $L = \inf_\delta a^\top (x + \delta) + b$

Concretization-based verification



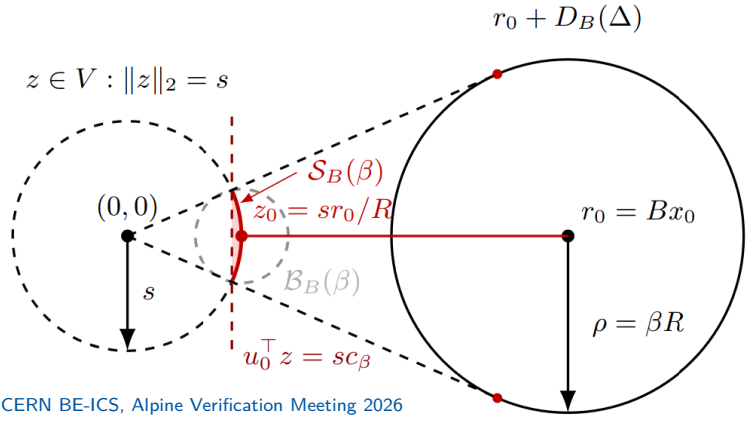
Verification procedure:

- Propagation of bounds towards z: $g(z) \geq a^\top z + b$ *easier*
- Concretize over noise-set and preprocessing:
  $L = \inf_\delta a^\top (N(W(x+\delta))) + b$ *harder*

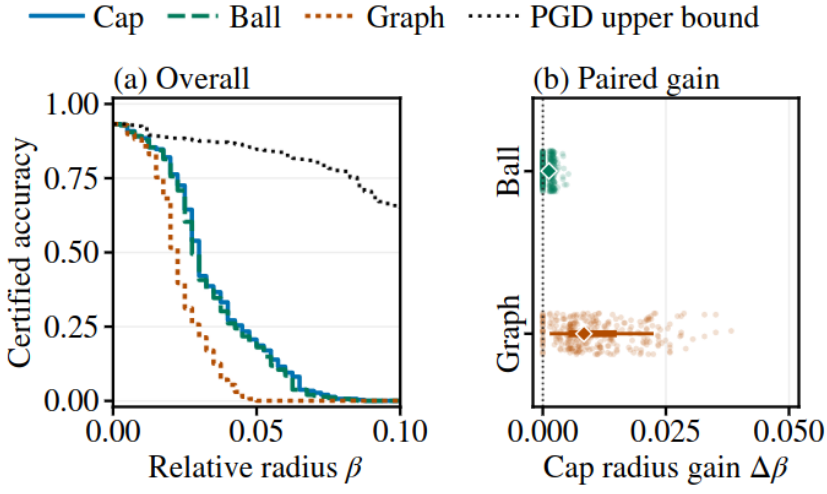
Closed-form for concretization over noise-set and preprocessing

Good news: The concretization has a closed-form optimal solution (under some assumptions)

- 1. Construct convex hull 2. Derive suppport function



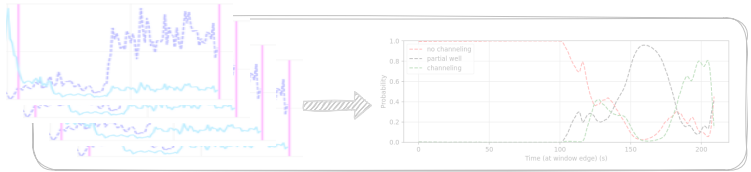
Results


Cap
Ball
Graph
PGD upper bound
(a) Overall
Certified accuracy
1.00
0.75
0.50
0.25
0.00
0.00
0.05
0.10
Relative radius β
(b) Paired gain
Ball
Graph
0.000
0.025
0.050
Cap radius gain $\Delta\beta$

Verification problems in two layers

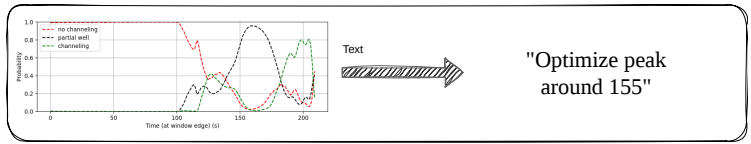
1. **Data layer:** Neural network predicts on data

Property: NN robustness against noise (local adversarial robustness)



2. **Decision layer:** Human or simple rules evaluates temporal data

Property: Satisfaction of temporal logic specifications



Problem setup



Verifying STL Properties of Sliding-Window Neural Classifiers under Adversarial Perturbations
Xaver Fink, Borja Fernandez Adiego, Joost-Pieter Katoen
RExAI Workshop 2026

Question:

Can small perturbations on the input signal violate a process-level temporal property?

local adversarial robustness + signal temporal logic (STL) satisfaction

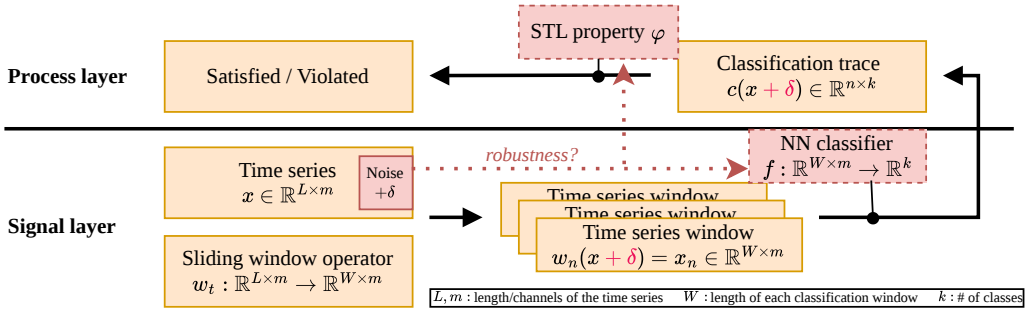
Problem setup

Verifying STL Properties of Sliding-Window Neural Classifiers under Adversarial Perturbations
Xaver Fink, Borja Fernandez Adiego, Joost-Pieter Katoen
RExAI Workshop 2026

Question:

Can small perturbations on the input signal violate a process-level temporal property?

local adversarial robustness + signal temporal logic (STL) satisfaction



Adversarially robust STL satisfaction

Goal: reason about all perturbations of the original signal x with $c_{w,f}(x) \models \varphi$

$$\forall \hat{x},\ \|\hat{x} - x\|_p \leq \varepsilon \implies c_{w,f}(\hat{x}) \models \varphi$$

$c_{w,f}(x)$: induced classification trace by windowing w, DNN f, input x

Adversarially robust STL satisfaction

Goal: reason about all perturbations of the original signal x with $c_{w,f}(x) \models \varphi$

$$\forall \hat{x},\ \|\hat{x} - x\|_p \leq \varepsilon \implies c_{w,f}(\hat{x}) \models \varphi$$

$c_{w,f}(x)$: induced classification trace by windowing w, DNN f, input x

Robustness score using quantitative STL semantics:

$$R_{\varphi,w,f}(x,\varepsilon) = \min_{\|\hat{x}-x\|_p \leq \varepsilon} \rho_{\varphi}(c_{w,f}(\hat{x}), 0)$$

- $R_{\varphi,w,f}(x,\varepsilon) \geq 0$: all perturbations preserve φ
- $R_{\varphi,w,f}(x,\varepsilon) < 0$: a violation-inducing perturbation exists

Adversarially robust STL satisfaction

Using quantitative STL semantics:

$$R_\varphi(x, \varepsilon) = \min_{\|\hat{x} - x\|_p \leq \varepsilon} \rho_\varphi(c(\hat{x}), 0)$$

- $R_\varphi(x, \varepsilon) \geq 0$: all admissible perturbations preserve φ
- $R_\varphi(x, \varepsilon) < 0$: a violating perturbation exists

Two questions:

- If a violating perturbation exists: How do we find it?
- If no violating perturbation exists: How do we prove it?

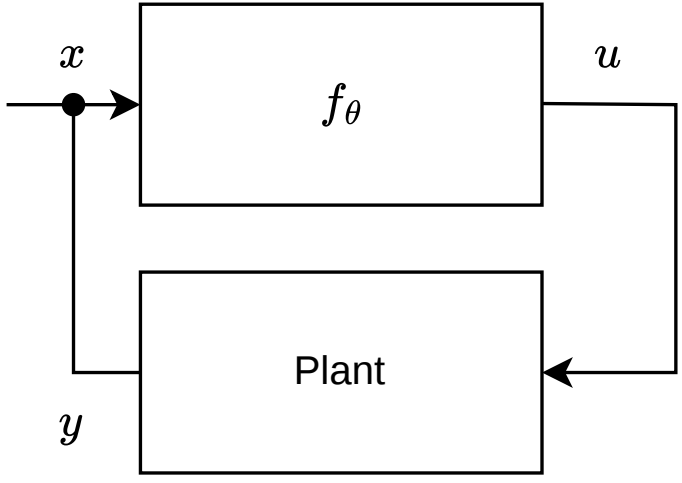
Overview

- Adversarial robustness verification is often not directly applicable
- Methods can be adapted!
- Interesting results can follow

Future interests

- Beyond adversarial robustness
  - Stochastic robustness
  - Conformal prediction
- Shielding

Verification of neural network controlled closed loop systems

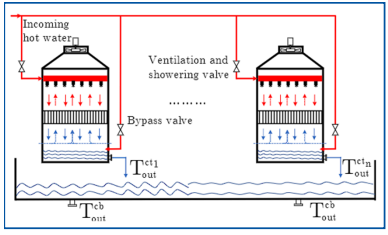


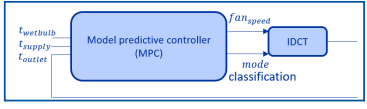
Case study: LHC cooling towers

Induced draft cooling towers (**IDCTs**)

- They provide **cold water** for different **LHC subsystems** (e.g. cryogenics, chillers, air handling units, etc.)

I. D. López-Miguel et al.
Verification of Neural Networks Meets PLC Code: An LHC Cooling Tower Control System at CERN
EANN 2023





Case study: LHC cooling towers

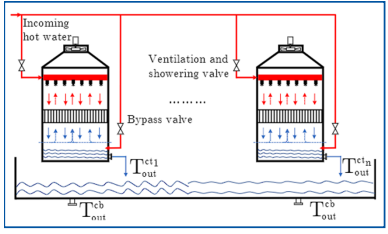
Induced draft cooling towers (**IDCTs**)

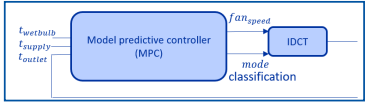
- They provide **cold water** for different **LHC subsystems** (e.g. cryogenics, chillers, air handling units, etc.)

Control actions:

- **Mode selection** (ventilation, showering or bypass)
- **Fan speed**

I. D. López-Miguel et al.
Verification of Neural Networks Meets PLC Code: An LHC Cooling Tower Control System at CERN
EANN 2023





Case study: LHC cooling towers

Induced draft cooling towers (**IDCTs**)

- They provide **cold water** for different **LHC subsystems** (e.g. cryogenics, chillers, air handling units, etc.)

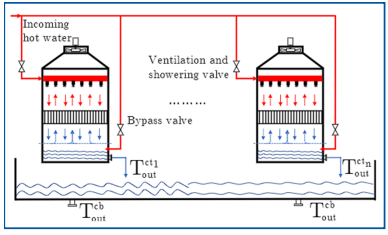
Control actions:

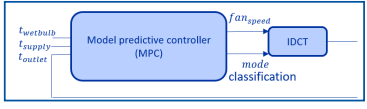
- **Mode selection** (ventilation, showering or bypass)
- **Fan speed**

Control objectives:

- Keep outlet water temperature within limits
- **Minimize** energy consumption

I. D. López-Miguel et al.
Verification of Neural Networks Meets PLC Code: An LHC Cooling Tower Control System at CERN
EANN 2023



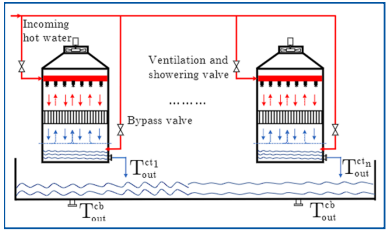


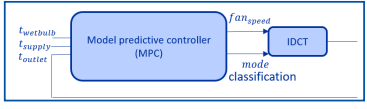
LHC cooling towers

Controls architecture:

- A **Model Predictive Controller** (MPC) was deployed to improve the existing controls
- We are evaluating to **replace the MPC** by **Neural Network running on a PLC** (less computationally demanding and easier to integrate in the existing controls hardware)

I. D. López-Miguel et al.
Verification of Neural Networks Meets PLC Code: An LHC Cooling Tower Control System at CERN
EANN 2023



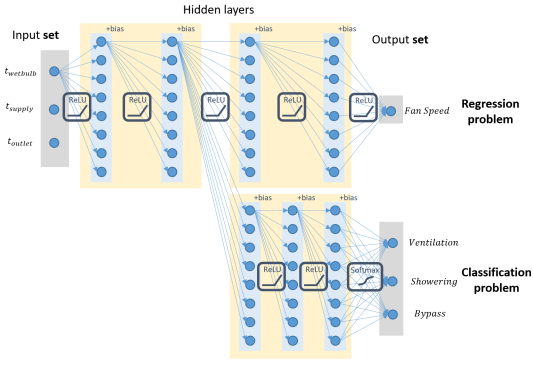


LHC cooling towers

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

Is it safe?

I. D. López-Miguel et al.
Verification of Neural Networks Meets PLC Code: An LHC Cooling Tower Control System at CERN
EANN 2023



LHC cooling towers



I. D. López-Miguel et al.
Verification of Neural Networks Meets PLC Code: An LHC Cooling Tower Control System at CERN
EANN 2023

- We verified **reachability**, **monotonicity**, **robustness**, etc.
- We applied methods to verify the design, the PLC source code and the executable
- Prototype $\rightarrow$ not in production

Problem: The same approach is not scalable for larger neural networks and more complex dynamics!



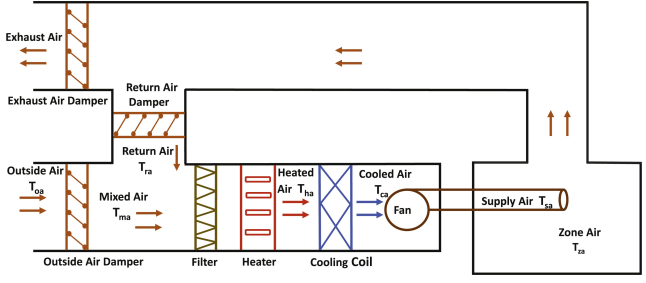


Case study: Approximate HVAC control

Design and experimental evaluation of energy-optimal control of air handling units
Faiq Ghawash, Brad Schofield, Morten Hovd, Lasse Bækdal Ahrendsen
CEP

HVAC control for the LHC SR1 building

- Goal: keep building temperature in safe range
- Method: model predictive control using CaSaDi



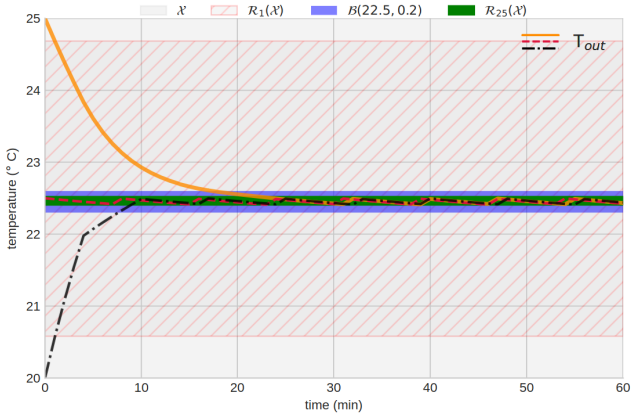
Case study: Approximate HVAC control



Approximate Model Predictive Control of Switched Affine Systems using Multitask Learning with Safety and Stability Guarantees
Faiq Ghawash, Morten Hovd and Brad Schofield
CDC 2023

Can we replace the MPC with approximate MPC?

- Reachability + inductive invariant style proof
- Prototype! → small neural network, stabilizing region



Approximate HVAC control

The real control problem is more complex!

Our plans for this:

- Evaluate state-of-the-art tools, e.g. CROWNReach
- Inductive invariant learning
  - Neural barrier functions
- Fallback: shielding
  - Model predictive
  - State-space abstraction

Summary

- We systematically apply **PLCverif** to our **Safety (critical) PLC programs**
- We have applied PLCverif to our **UNICOS** baseline objects and other standard program blocks

Summary

- We systematically apply **PLCverif** to our **Safety (critical) PLC programs**
- We have applied PLCverif to our **UNICOS** baseline objects and other standard program blocks
- **First steps** on the **verification of Neural Network controllers**

Summary

- We systematically apply **PLCverif** to our **Safety (critical) PLC programs**
- We have applied PLCverif to our **UNICOS** baseline objects and other standard program blocks
- **First steps** on the **verification of Neural Network controllers**
- Several research collaboration (namely with **NASA Ames**, **RWTH Aachen University**, **EPFL Lausanne**, and others) on topics like **compositional verification**, **NN-controller verification**, formal specifications, model-based software engineering, etc.
- External collaborations providing a formal verification service (e.g. **GSI**, ITER, etc.)

Benefits of using formal methods for BE-ICS

- We systematically **find discrepancies** between the code and the specifications (even in previously tested PLC programs) at the development phase:
  - Program bugs
  - Wrong or incomplete specifications

Benefits of using formal methods for BE-ICS

- We systematically **find discrepancies** between the code and the specifications (even in previously tested PLC programs) at the development phase:
  - Program bugs
  - Wrong or incomplete specifications
- Formal verification helped to better **understand the behavior** of the PLC programs

Benefits of using formal methods for BE-ICS

- We systematically **find discrepancies** between the code and the specifications (even in previously tested PLC programs) at the development phase:
  - Program bugs
  - Wrong or incomplete specifications
- Formal verification helped to better **understand the behavior** of the PLC programs
- It **reduces** significantly the **cost** of the **testing** and **validation phases**
- The cost of using formal methods is minimized by PLCverif (significant initial investment)

Benefits of using formal methods for BE-ICS

- We systematically **find discrepancies** between the code and the specifications (even in previously tested PLC programs) at the development phase:
  - Program bugs
  - Wrong or incomplete specifications
- Formal verification helped to better **understand the behavior** of the PLC programs
- It **reduces** significantly the **cost** of the **testing** and **validation phases**
- The cost of using formal methods is minimized by PLCverif (significant initial investment)
- We hope for similar results with **safe Neural Network controllers**

References (I)

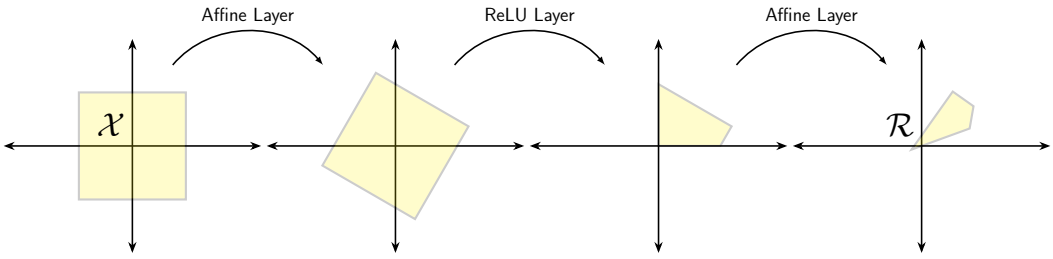


Michael Benedikt, Wolfgang Bartmann, Jean-Paul Burnet, Christian Carli, Antoine Chance, Paolo Craievich, Massimo Giovannozzi, Christophe Grojean, Johannes Gutleber, Klaus Hanke, Andre Henriques, Patrick Janot, Carlos Lourenco, Michelangelo Mangano, Thomas Otto, John Howard Poole, Srini Rajagopalan, Tor Raubenheimer, Ezio Todesco, Luisa Ulrici, Timothy Paul Watson, Guy Wilkinson, and Frank Zimmermann.

Future circular collider feasibility study report volume 2: Accelerators, technical infrastructure and safety, 2025.


CERN

Interval Propagation



Goal: Propagate $\mathcal{X}$ through FFNN layers

$\rightarrow$ Abstract interpretation for interval propagation