



Formal Methods and Verification (FMV)

HSE-RP

Katharina Ceesay-Seitz

Hamza Boukabache

BE-ICS

Ignacio David Lopez Miguel

Borja Fernández Adiego

Jean-Charles Tournier

Enrique Blanco Viñuela

09/12/2021

Context – Formal Methods and Verification (FMV)

- Initiative from **HSE-RP** and **BE-ICS**

- **Support of the RAS WG**

} **Formal methods** can contribute to design and develop **safe and reliable systems**

- **Objectives:**

- Build a CERN **formal methods community**
- **Inventory** and return of experience of **CERN projects** and **tools** using formal methods
- **Promote** the **collaboration** between different groups
- **Identify** the applicability of formal methods to other areas
- **Inform** other groups on the benefits of the methods
- Establish **relation with external institutes / universities / companies**
- **Report** on current research status (e.g. conference reports)

Context - Failure categories

RAS WG

1. Random Hardware Failures

- From degradation mechanism

Stochastic
methods

Measures to combat the hardware random failures (e.g. RBD, FTA, etc.)

2. Systematic Failures

- Incorrect **specification/design**
- Human errors
- **Software** errors
- Maintenance and modifications
-

Deterministic
methods

Measures to combat the systematic failures (e.g. **formal specification, formal verification**, (functional) testing, etc.)

Formal Methods

All types of failures have an impact on the **reliability** and **availability** of the global system

Contents

1. Brief **introduction** to formal methods

- **What** are/ **Where** can we apply/ **When** should we apply formal methods?
- **Why they are not very popular** in certain industries (yet)?

2. CERN examples

- **Industrial Controls** domain (BE-ICS):
 - Formal Verification (**model checking**) of PLC programs
 - Formal **Specification** of PLC programs
- **Digital electronics design** domain (HSE-RP)
 - Formal Verification (**model checking**) of VHDL code
 - Semi-formal **specification** of requirements for HDL designs

3. **Conclusions**

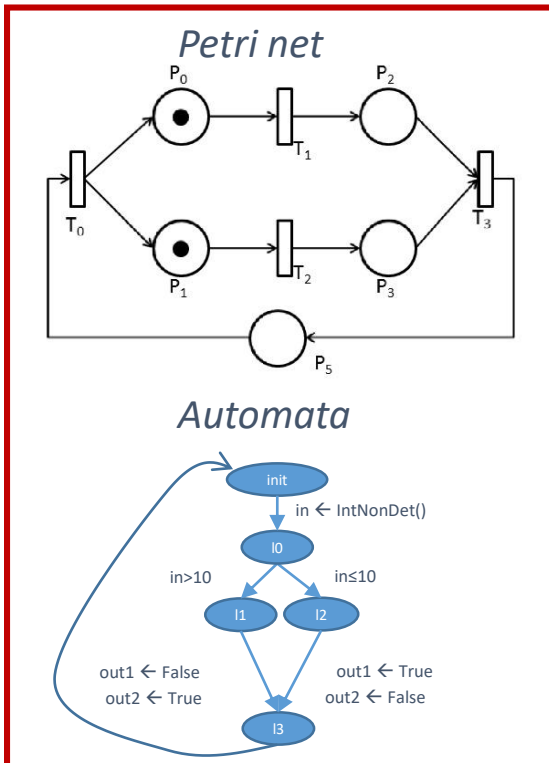
What are Formal Methods?

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

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

Where can we use Formal Methods?

Different phases of a system development, for example:

- **Specification and modelling:** use of unambiguous languages to describe a system (**precise** description, **code generation**, test case generation, etc.)
- Formal model **simulation:** formal models to simulate the behaviour of the real system (e.g. model simulation with UPPAAL)
- **Formal verification:** formalized properties checked against a formal model (e.g. **model checking**)
- Test or code generation: formal models to generate relevant test cases or the code itself
- Equivalence checking (does the implementation **match** the specification?)
- and more ...

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

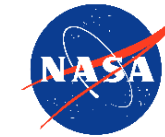


Formal Verification of Critical Aerospace Software <https://hal.archives-ouvertes.fr/hal-01184099/document>

Formal verification



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



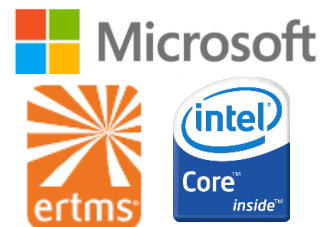
Use of the model checker SPIN to verify the model of a software <http://spinroot.com/gerard/pdf/spin04.pdf>



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

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>

- 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**

When should/could we use Formal Methods?

When the **cost of a system failure is higher than the cost of using formal methods**

Some examples:

- **Safety critical systems**
 - ✓ Damage to the environment, the installation, people
 - ✓ Damage of the reputation of the company/organization
 - ✓ Recommended by the standards (e.g. IEC 61508)
- Software libraries used in many systems
- etc.

Which Formal Method should we use?

- The most appropriate to **describe the behavior** of your system
- The most appropriate for the **final purpose** (specification, formal verification, etc.)
- A formalism **supported by tools** (e.g. simulator, **model checker**, etc.)

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

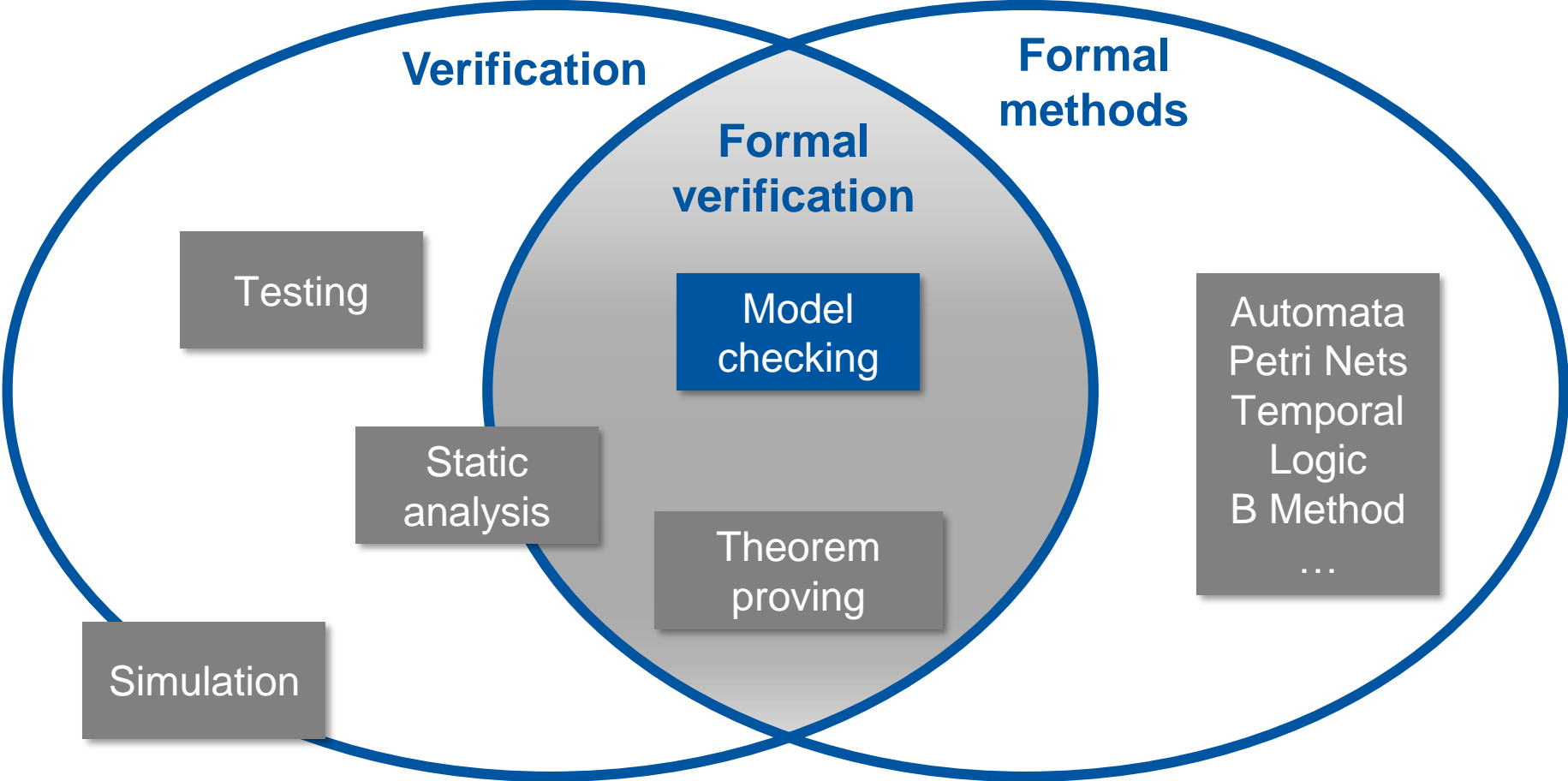
- several references to model checking. For example from IEC 61511-2:2016 Annex B:

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

Industrial Controls Domain

- Introduction to **model checking**
- Formal verification (model checking) of PLC programs
- Formal specification of PLC programs

Formal verification and model checking



Formal verification tools

Software development

Language	Tool
----------	------

Modelling

Language	Tool
----------	------

Timed automata	
Finite state machine	

No commercial tools to apply formal verification to PLC programs




Electronics design

Language	Tool
----------	------

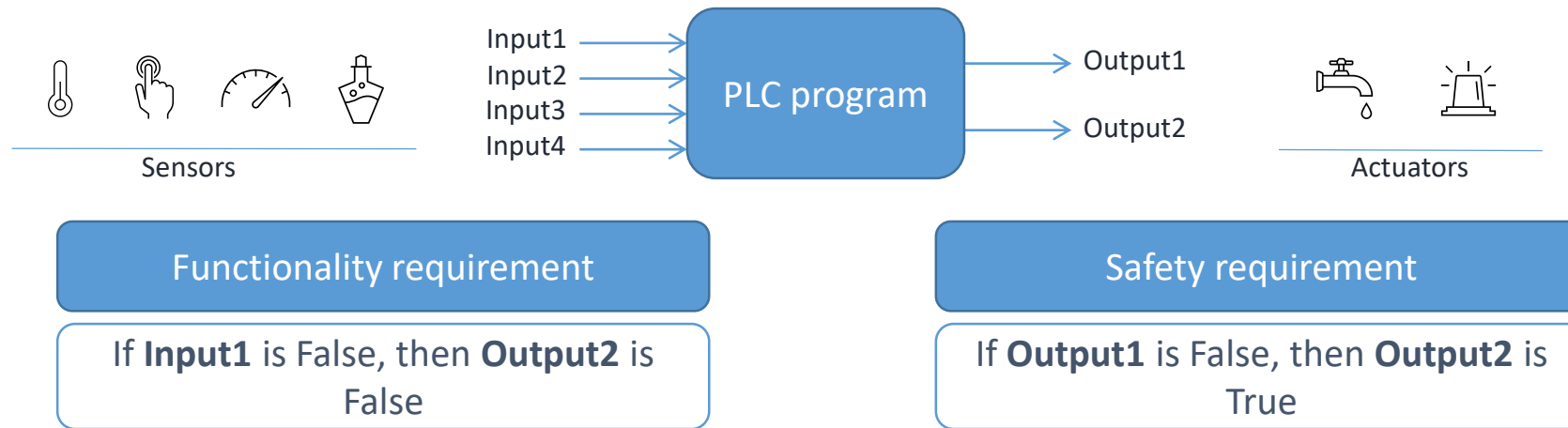
PSL, SVA	<i>SymbiYosys</i>
	
VHDL, Verilog, System Verilog	<i>cadence</i> [®]
	

Neural networks

Technology	Tool
------------	------

Closed-loop control system	VENUS
Safe reinforcement learning	

Introduction to model checking (e.g. for PLC programs)



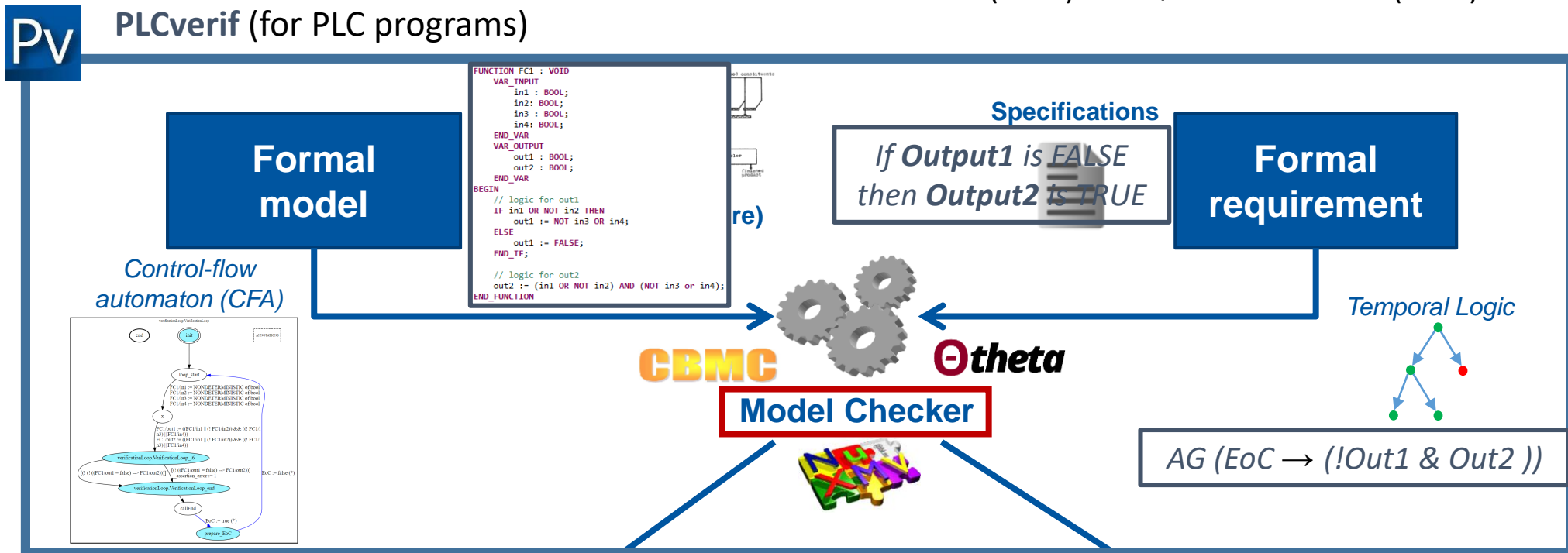
- If “*Input1*”, “*Input2*”, “*Input3*” and “*Input4*” are **BOOL**, then we need to check $2^4 = 16$ combinations
- If they are **INT** (16-bit), then $2^{16*4} \approx 1.8*10^{19}$ combinations
- for large systems (many variables), such requirements **cannot** (practically) be checked by using testing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

Solution: **Model checking**

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

Property failed
Trace leading to the violation

Property OK

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



PLCverif internals – PLC program modeling



PLC program

```

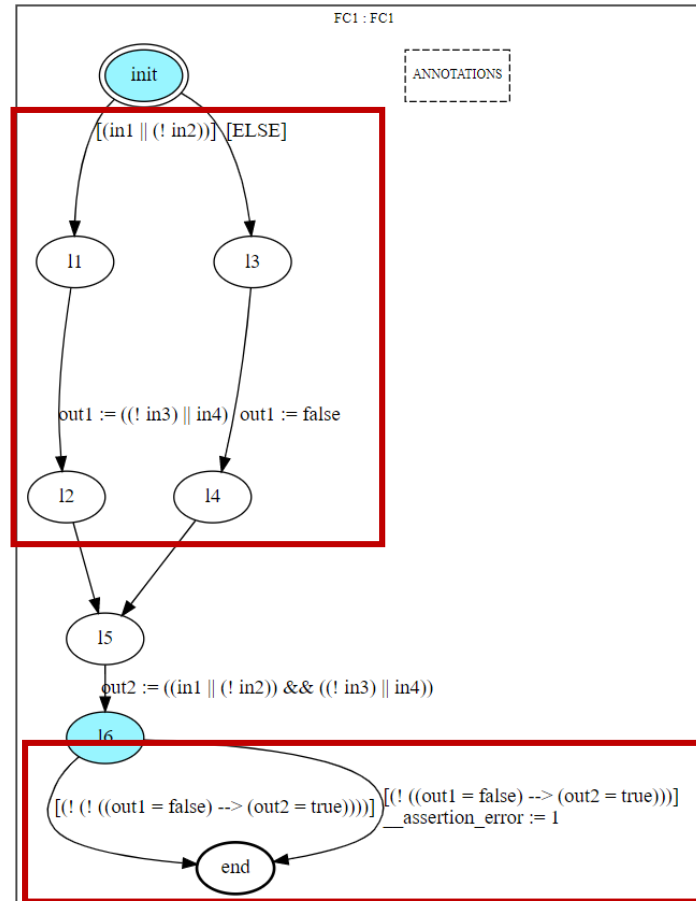
FUNCTION FC1 : VOID
  VAR_INPUT
    in1 : BOOL;
    in2: BOOL;
    in3 : BOOL;
    in4: BOOL;
  END_VAR
  VAR_OUTPUT
    out1 : BOOL;
    out2 : BOOL;
  END_VAR
BEGIN
  // logic for out1
  IF in1 OR NOT in2 THEN
    out1 := NOT in3 OR in4;
  ELSE
    out1 := FALSE;
  END_IF;

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

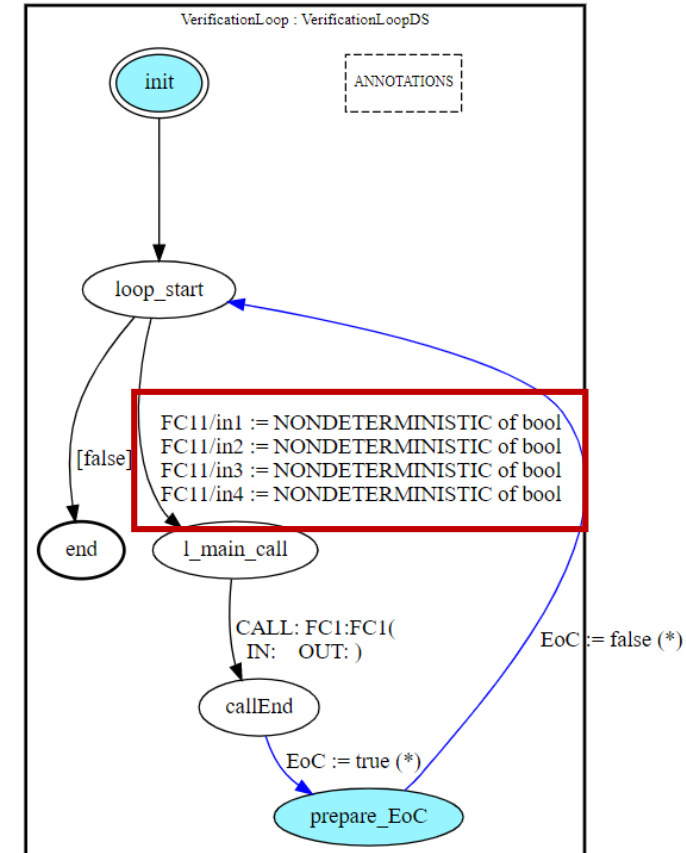
Property to verify (verification case)

If out1 is FALSE, then out2 is TRUE at the end of the PLC cycle?

Intermediate Model - Control Flow Automaton (CFA)



Formalized property (assertion error)

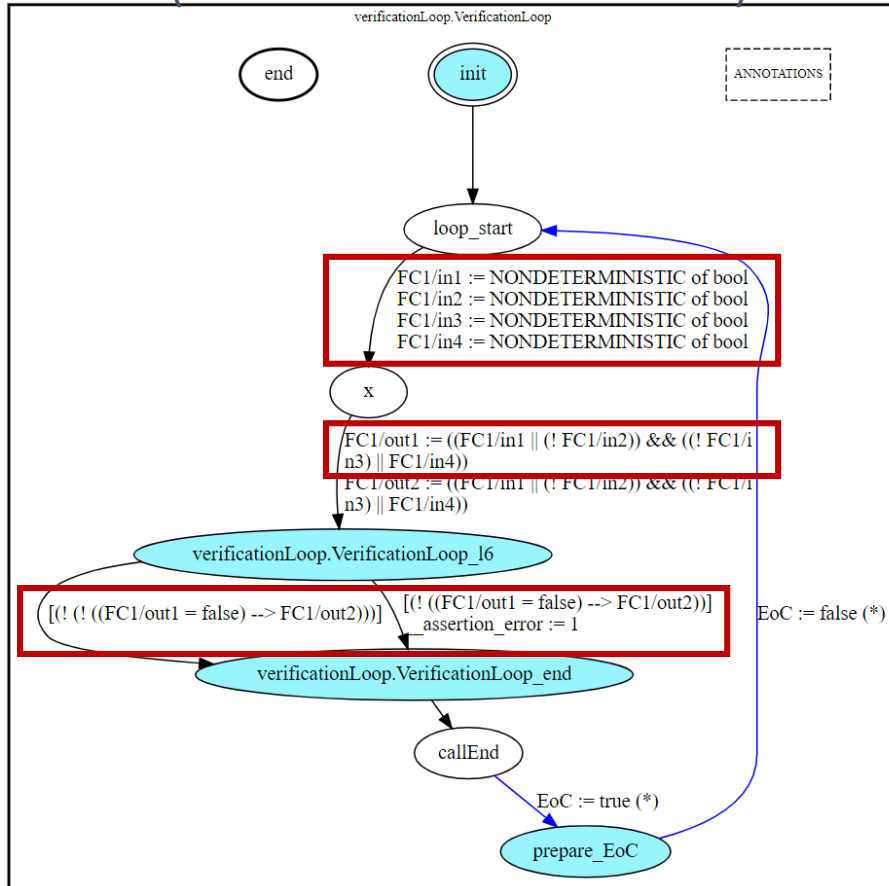


Specific to PLCs (PLC scan cycle model)

PLCverif internals – Translation to model checker input language

Inlined Control Flow Automaton (CFA)
(Inline and reduce the model)

(simplified) nuXmv model



```

MODULE main
VAR
  loc : {init_pv, end, loop_start, callEnd, prepare_EoC, ...};
  FC1_in1 : boolean;
  FC1_in2 : boolean;
  ...
  FC1_out1 : boolean; -- frozen
  ...
  __assertion_error : unsigned word[16]; -- frozen
  EoC : boolean; -- frozen
ASSIGN
  -- CFA structure (loc)
  init(loc) := init_pv;
  ...
esac;
init(FC1_in1) := FALSE;
next(FC1_in1) := case
  loc = loop_start & (TRUE) : {TRUE, FALSE};
  TRUE : FC1_in1;
esac;
init(FC1_out1) := FALSE;
next(FC1_out1) := case
  loc = x & (TRUE) : ((FC1_in1) | (!FC1_in2)) & ((!(FC1_in3)) | (FC1_in4));
  TRUE : FC1_out1;
esac;
init(__assertion_error) := 0ud16_0;
next(__assertion_error) := case
  loc = verificationLoop.VerificationLoop_l6 & (!(FC1_out1) = (FALSE)) -> (FC1_out2)) : 0ud16_1;
  TRUE : __assertion_error;
esac;
init(EoC) := FALSE;
next(EoC) := case
  loc = callEnd & (TRUE) : TRUE;
  loc = prepare_EoC & (TRUE) : FALSE;
  TRUE : EoC;
esac;
-- Requirement
CTLSPEC AG((EoC) -> ((__assertion_error) = (0ud16_0)));
  
```

Temporal logic formula

PLCverif internals – executing nuXmv

Executing the *model checking algorithm* (nuXmv)

Counterexample

trace that indicates that the model (PLC program) and the specification do not match

```
Windows PowerShell
PS C:\dev\PLCverif\tools\nuxmv> .\nuXmv.exe .\verifCase1.smv
*** This is nuXmv 1.1.1 (compiled on Wed Jun 1 10:23:30 2016)
*** Copyright (c) 2014-2016, Fondazione Bruno Kessler

*** For more information on nuXmv see https://nuxmv.fbk.eu
*** or email to <nuxmv@list.fbk.eu>.
*** Please report bugs at https://nuxmv.fbk.eu/bugs
*** (click on "Login Anonymously" to access)
*** Alternatively write to <nuxmv@list.fbk.eu>.

*** This version of nuXmv is linked to NuSMV 2.6.0.
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Copyright (C) 2010-2014, Fondazione Bruno Kessler

*** This version of nuXmv is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of nuXmv is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

*** This version of nuXmv is linked to MathSAT
*** Copyright (C) 2009-2016 by Fondazione Bruno Kessler
*** Copyright (C) 2009-2016 by University of Trento
*** See http://mathsat.fbk.eu
```

```
-- specification AG (EoC -> __assertion_error = Oud16_0) is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  loc = init_pv
  FC1_in1 = FALSE
  FC1_in2 = FALSE
  FC1_in3 = FALSE
  FC1_in4 = FALSE
  FC1_out1 = FALSE
  FC1_out2 = FALSE
  EoC = FALSE
-> State: 1.2 <-
  loc = loop_start
-> State: 1.3 <-
  loc = x
  FC1_in1 = TRUE
  FC1_in3 = TRUE
-> State: 1.4 <-
  loc = verificationLoop_VerificationLoop_16
-> State: 1.5 <-
  loc = verificationLoop_VerificationLoop_end
-> State: 1.6 <-
  loc = callEnd
-> State: 1.7 <-
  loc = prepare_EoC
  EoC = TRUE
```

PLCverif (for users)

The screenshot displays the PLCverif application interface. The 'Project Explorer' on the left shows a project structure with folders like 'output', 'src-gen', and 'Verification case...'. The main window shows the 'Verification backend' settings for a project named '1_PBCS_Workshop_Demo'. The 'Backend' dropdown is set to 'NuSMV', and the 'Algorithm' is '(default)'. Below these settings, there are sections for 'Requirement', 'Requirement - advanced', 'Reporters', and 'Advanced settings'. The 'Verify' section contains a 'Verify!' button and a status area showing 'Last result: N/A', 'Last execution: N/A', and 'Last duration: N/A'. At the bottom, a table displays verification results:

OUTPUT BOOL	FC1.out1	false
OUTPUT BOOL	FC1.out2	false

The **complexity** of using formal methods **is hidden** by the tool (PLCverif) – More details in www.cern.ch/plcverif

Formal verification of PLC programs at CERN

Functional Safety projects

- Magnet test benches:
 - **SM18** Safety PLC programs (**CEM** Specification + **PLCverif**)
 - **B180 FAIR** and **B311 Switchboard** Safety PLC programs
- **ITER** case study: verification of PLC program in charge of a safety critical communication protocol

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

- **SPS Personnel Safety System: fail-safe PLC program**

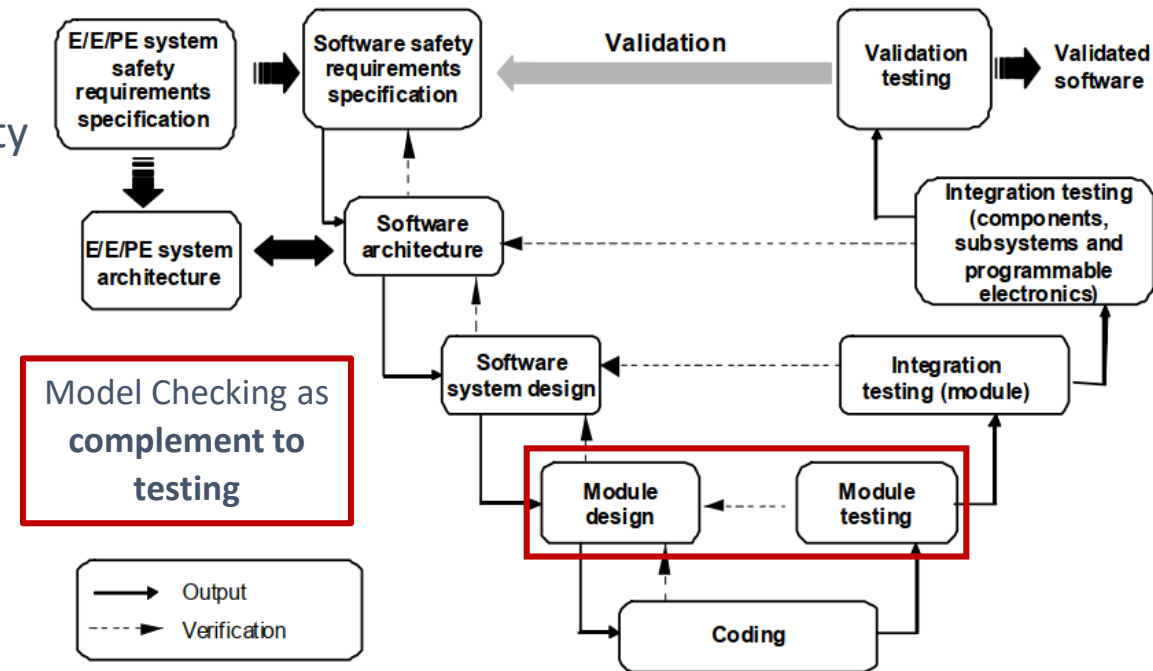
B. Fernandez et al. "Applying model checking to highly-configurable safety critical software: The SPS-PPS PLC program" in Proc. of the 18th ICALEPCS

Non-safety PLC programs but widely used at CERN:

- **UNICOS** object library (used in the LHC Cryogenics control system, many C&V plants, Gas systems, etc.)



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



Some of the models had $5.0 * 10^{91}$ and $6.5 * 10^{55}$ combinations to be checked (Potential State Space)

Formal specification of PLC programs – PLCspecif

ExampleModule <i>This module represents value limiter. If it is enabled, the output value is within the limits given as parameters. If disabled, the output is always 0.</i>																			
Assigned inputs: <ul style="list-style-type: none"> ValueReq : INT16 <i>Value to be limited.</i> EnableReq_fromLogic : BOOL EnableReq_fromScada : BOOL EnableReq_fromField : BOOL DisableReq : BOOL PMin : INT16 param <i>Lower limit.</i> PMax : INT16 param <i>Upper limit.</i> 	Assigned outputs: <ul style="list-style-type: none"> Value : INT16 Status : BOOL 																		
Input signal definitions: – (none)																			
Event input definitions: <ul style="list-style-type: none"> @disable \leftarrow rising_edge(DisableReq) (pri=1) @enable \leftarrow EnableReq_fromLogic OR EnableReq_fromScada OR EnableReq_fromField (pri=2) 																			
Core logic (state machine) <pre> stateDiagram-v2 state Disabled state Enabled Disabled --> Enabled : @enable Enabled --> Disabled : @disable </pre>																			
Output signal definitions: <ul style="list-style-type: none"> _Value = <table border="1"> <thead> <tr> <th>ValueReq < PMin</th> <th>ValueReq > PMax</th> <th>result</th> </tr> </thead> <tbody> <tr> <td>T</td> <td>.</td> <td>PMin</td> </tr> <tr> <td>F</td> <td>T</td> <td>PMax</td> </tr> <tr> <td>F</td> <td>F</td> <td>ValueReq</td> </tr> </tbody> </table> Value = <table border="1"> <thead> <tr> <th>in_state(Enabled)</th> <th>result</th> </tr> </thead> <tbody> <tr> <td>T</td> <td>_Value</td> </tr> <tr> <td>F</td> <td>0</td> </tr> </tbody> </table> Status = in_state(Enabled) 		ValueReq < PMin	ValueReq > PMax	result	T	.	PMin	F	T	PMax	F	F	ValueReq	in_state(Enabled)	result	T	_Value	F	0
ValueReq < PMin	ValueReq > PMax	result																	
T	.	PMin																	
F	T	PMax																	
F	F	ValueReq																	
in_state(Enabled)	result																		
T	_Value																		
F	0																		
Invariant properties: <ul style="list-style-type: none"> ALWAYS $PMin \leq Value \leq PMax$ ASSUMING $PMin \leq PMax$ <i>If the limit values are valid, the output is always within the limits.</i> 																			

Formal specification of PLC programs wiki:

<https://readthedocs.web.cern.ch/display/ICKB/PLC+formal+specification>

D. Darvas et al. "A formal specification method for PLC-based applications" in Proc. of the 15th

ICALEPCS <https://accelconf.web.cern.ch/ICALEPCS2015/papers/wepgf091.pdf>

Variable definition

Events and state machine (behavior)

Output variables logic

Invariants

Benefits

Precise specification

Verification cases generation (model checking properties)

Test cases generation

Future goals

PLC code generation (correctness by construction)

Contact – Formal Methods interest group

If you are interested in Formal Methods, please subscribe to the following egroup:

formal-methods-interest-group@cern.ch

(you will get updates about future events and presentations)

If you want to get started with formal methods or establish a collaboration, you can send an email to the following egroup:

formal-methods-working-group-admin@cern.ch

We have created a ***readthedocs page***, where useful information about formal methods and verification will be collected:

<https://readthedocs.web.cern.ch/display/FMVWG/Formal+methods+interest+group+Home>

