Applying Formal verification on Industrial Control systems

Borja Fernández
Daniel Darvas
Enrique Blanco
Outline

- Motivation & context

- Formal verification
  - Definition
  - Model checking vs. Testing
  - Some Myths & “Real” examples

- Formal verification of PLC programs
  - Challenges
  - Some results

- Conclusion & Brainstorming
Motivation & Context
Motivation & Context

- Solution for Safety Systems according to the IEC 61508
  - Safety life cycle (Risk analysis, SIL level, …)
  - Verification hardware. SIL (Safety Integrated Level)
- Verification software:
  - PLC code meets the specification
  - But how?
Even for SIL1 is recommended to use [Semi]-formal methods.
About formal verification
How to classify model checking?
Some definitions

- **Verification**: ensuring system complies with the specification

- **Formal methods**: methods with *mathematical* basis, typically given by a formal specification language.

- **Formal verification**: *both* together 😊
How to classify model checking?

Verification

Model checking

Formal verification

Verif. based on theorem proving

Testing

Static analysis

Formal methods

Formal specification (B, Z, Alloy, …)

Formalisms: Automata, Petri Nets, Temporal Logic
Testing vs. model checking

Model checking tools: algorithms to check that a global model (hardware, software, process, etc.) meets given requirements.

- e.g. NuSMV, UPPAAL, DFinder (BIP), SPIN, KRONOS, etc.

**Testing**

- Inputs are known, outputs are checked

**Model checking**

- Usually: The possibility of an output combination is checked.
- Can be used for other ways too.
Testing vs. model checking

**Requirement**

If \( I0.0 \) is FALSE and \( I0.1 \) is FALSE, then \( Q0.0 \) is FALSE

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

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

Model checking will answer that this property is **not** correct and it will provide a **counterexample**: \( \text{Var1} == 1 \)
Testing vs. model checking

Safety Requirement
If $Q_{0.0}$ is TRUE, then $Q_{0.1}$ is FALSE

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

It’s a **extremely complicated task** for Testing.
What is model checking?

Formal model

Real System (hardware, software)

Specifications

Formal requirement

Model Checker

Property failed

Trace leading to the violation

Property OK
Model checking

1. How to build the formal models?
   Automata, Petri nets, Timed automata, …

2. How to build the formal requirement?
   Temporal Logic

Temporal logic

<table>
<thead>
<tr>
<th>Boolean logic</th>
<th>Temporal operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>operators: AND, OR, NOT</td>
<td>in the future …</td>
</tr>
<tr>
<td>predicates: input=TRUE, temp&gt;100</td>
<td>once ...</td>
</tr>
<tr>
<td></td>
<td>always ...</td>
</tr>
<tr>
<td></td>
<td>until ...</td>
</tr>
</tbody>
</table>
Requirements

- They are the two big groups of requirements.

- **Safety:**
  - Something bad is never true.
  - *(or: Something good is always true.)*
  - *On and Off is never true at the same time.*
  - Counterexample is finite.

- **Liveness:**
  - Something good will eventually happen.
  - *If there is a request, it will be served.*
  - *(We don’t know when, but eventually it will be served.)*
  - Counterexample is infinite.

- Deadlock-freeness, reachability, fairness…
How does a model checker work?

We don’t know!!! It’s a **black box**!!!  But …

Internal representation in a Model checker: BDD, MDD, etc.

<table>
<thead>
<tr>
<th>I0.0</th>
<th>I0.1</th>
<th>Var1</th>
<th>Q0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

At the end of a PLC cycle, when I0.0 = I0.1 = 0, then Q0.0 is always 0.

PSS: Potential Space State

RSS: Reachable Space State
Good things about model checking

- Explores **all possible combinations**
- Allows to check properties that are almost **impossible** to test.
- **Counterexamples**
- Possible to **automatize** (can be used by non-formal method experts)
“Problems” with model checking

- We check a **model**, not the real system
  
  - **How can we be sure** that it represents the real system?

- State space **explosion**

- Formal **requirement**

- Formal **model**
Myths (published papers)

1. Seven Myths of Formal Methods


- They require highly trained mathematicians
- They *increase the cost* of development
- They *are not used on real, large scale software*

2. Seven more Myths of Formal Methods


- Formal methods *delay* the development process
- Formal methods are *not supported* by tools
- Formal methods only apply to software
- Formal methods people always use formal methods
Real cases (many)

- **NASA Remote agent** -- The Remote Agent spacecraft control system was used by NASA on the Deep Space 1 mission (launched in 1998, ended in 2001).

- **Intel processors**.

- **Subway**: Paris, Line 14 (MÉTÉOR) -- The safety critical parts (the automatic pilot and signalling subsystem, called PA-SIG) were developed using B Method.

- **IPv6**: The new IP protocol was verified formally at an early stage of the standardisation process. Bugs were found.

- Etc.
Goals

Automated **model checking**

**Hide the complexity from control engineers**

- Complexity of building the formal **models**.
- Complexity of using temporal logic for requirement **specification**.
Goals

1. Verification of Standard PLC control systems:
   - ST and SFC code: Verification of the UNICOS library and the real applications.

2. Verification of Safety PLC control systems:
   - IL code, exported from Ladder or FBD.

Verification of “complex” properties: Safety, liveness, “sequences”, timed properties, etc.
About formal verification of PLC programs
Challenges

1. How to create automatically the formal models?
   - Methodology & automatic generation tool
     - Xtext & EMF technologies
   - Intermediate model (IM) based on automata. Why?
     - Simple formalism but strong enough to model “all” the features of a PLC system.
     - Many model checking tools use modelling languages close to automata.
     - Split the transformation in 2 parts: Semantic (PLC language to IM) and Syntactic (IM to different model checker languages) transformations.
   - Different input languages (ST, IL, SFC, etc.) & different formal models (NuSMV, UPPAAL, BIP, etc.)
Challenges

1. How to create automatically the formal models?
   - Our simplified approach

\[ N = (A, I) \]
\[ a = (L, T, l_0, V, Val_0) \in A \]
\[ t = (l, g, amt, i, l') \in T \]
\[ i = (t, t') \in I \]
Challenges

2. How to model time & timers in PLCs?

- "Realistic" approach
  - Model time explicitly
  - Need of a monitor for verifying timed properties

- "Abstract" approach
  - No time
  - Introducing non-determinism

If there is a rising edge on “A”, then 3 seconds later “B” will be true

If there is a rising edge on “A”, then eventually “B” will be true
Challenges

3. How to reduce the state space explosion problem?

Reductions & abstraction techniques

ST code

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

Property

always c > 0
Challenges

4. How to define the requirements?

This is what we need!!!

Requirement can be expressed with LTL:

\[ G((PLC\_END \& FuStopI) \rightarrow ((PLC\_END \rightarrow FuStopI) \ \underline{U} \ (PLC\_END \& !FuStopI \& ((!AuOnR \& !MOnR \& !HOnR) \rightarrow !OutOnOV)))) ; \]

This is what we get!!!

Original requirement:

“After falling edge on FuStopISt, the OutOnOV MUST remain FALSE IF AuOnR=FALSE AND MOnR=FALSE”

HOnR = FALSE should be also included

intermediate compromise (pattern) 😊

Semi-formal:

If in the last cycle, FuStopI=T and in the current cycle FuStopI=F AND AuOnR=F, MOnR=F, HOnR=F AND PFsPosOn=F, then OutOnOV = false.
Challenges

5. How to find the source of the problem?

- Counterexample
- But … it can be long:
  - text file (e.g. 1.4 MB, 34.000 lines)
- Content:
  - values for every variable, for every PLC cycle, after every instruction
- Can be reduced (automatically):
  - only the ends of the cycles are interesting
  - only 780 lines, like:

We can proof that the bug exists in the real system!!

**Cycle 1**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HLD</td>
<td>FALSE</td>
</tr>
<tr>
<td>HONR</td>
<td>FALSE</td>
</tr>
<tr>
<td>HOFFR</td>
<td>FALSE</td>
</tr>
<tr>
<td>STARTI</td>
<td>FALSE</td>
</tr>
<tr>
<td>TSTOPi</td>
<td>FALSE</td>
</tr>
<tr>
<td>FUSTOPI</td>
<td>FALSE</td>
</tr>
<tr>
<td>AUONR</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

...
Challenges

6. How to model the PLC platform?
   - Study of **concurrency** problems in PLCs.
   - Modelling Scan cycle, interrupts, restarts, etc.
   - E.g. It will allow us to verify if safety properties are preserved when an interrupt appears.

7. Modelling the process for PLC code verification?
   - Simple information used to **reduce** the state space (Data abstraction)
   - IW0: 16 bits variable $2^{16}$ combinations. They can be reduced using process information (assumptions).
Our approach
Small example

1. ST code

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

2. CFG representation

3. NuSMV model

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

CERN
Small example

UNICOS
Local object
**Results: UNICOS ONOFF object**

**Safety requirement about mode manager**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Non-reduced model</th>
<th>Reduced Model</th>
<th>Specific model</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSS</td>
<td>$1.61 \times 10^{218}$</td>
<td>$4.57 \times 10^{36}$</td>
<td>$3.65 \times 10^{10}$</td>
</tr>
<tr>
<td>Variables</td>
<td>255</td>
<td>118</td>
<td>33</td>
</tr>
<tr>
<td>Generation</td>
<td>0.3 s</td>
<td>11.3 s</td>
<td>12.6 s</td>
</tr>
</tbody>
</table>

**Verification Run-time**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Non-reduced model</th>
<th>Reduced Model</th>
<th>Specific model</th>
</tr>
</thead>
<tbody>
<tr>
<td>No parameters</td>
<td>–</td>
<td>–</td>
<td>8.398 s</td>
</tr>
<tr>
<td>-dynamic</td>
<td>–</td>
<td>$\approx 7$ h</td>
<td>1.334 s</td>
</tr>
<tr>
<td>-dynamic -df</td>
<td>–</td>
<td>160.8 s</td>
<td>0.547 s</td>
</tr>
<tr>
<td>-dynamic -df -dcx</td>
<td>–</td>
<td>3.787 s</td>
<td>0.141 s</td>
</tr>
</tbody>
</table>
If mode is Forced and AuAuMoR=true, no other requests, Forced mode is not inhibited, the mode will be set to Auto.

**Specification:**

<table>
<thead>
<tr>
<th>AuAuMoR</th>
<th>BOOLEAN</th>
<th>Auto Mode Request</th>
<th>Auto Mode Request. The control logic requests Auto Mode on the object</th>
</tr>
</thead>
<tbody>
<tr>
<td>ManReg01.MAuMoR</td>
<td>BIT1</td>
<td>Manual Auto Mode Request</td>
<td>Manual Auto Mode Request: The operator requests the Auto Mode.</td>
</tr>
</tbody>
</table>

**Result:** false

```plaintext
(* Auto Mode *)
IF (MMoSt_aux AND (E_MAuMoR OR E_AuAuMoR)) OR
   (FoMoSt_aux AND E_MAuMoR)) OR
   (SoftLDSt_aux AND E_MAuMoR) OR
   (MMoSt_aux AND AuIhMmo) OR
   (FoMoSt_aux AND AuIhFoMo) OR
   (SoftLDSt_aux AND AuIhFoMo) OR
NOT(AuMoSt_aux OR MMoSt_aux OR FoMoSt_aux OR SoftLDSt_aux) THEN
   AuMoSt_aux := TRUE;
   MMoSt_aux := FALSE;
   FoMoSt_aux := FALSE;
   SoftLDSt_aux := FALSE;
END_IF;
```
Just after a Full Stop Interlock, the OutOnOV is false, if there is no On request (and the FsPos is Off).

**Specification**

In LTL:

\[ G((PLC\_END \& FuStopI) \rightarrow ((PLC\_END \rightarrow FuStopI) U (PLC\_END \& !FuStopI \& (!AuOnR \& !MOnR \& !HOnR) \rightarrow !OutOnOV))) ];

**Result:** false

Results: we found some bugs!! 😊 ☹️
Ongoing and future work

1. Concurrency problems: Shared memory
   - ST instructions are **not atomic**
   - Variable assignments from separate OBs
   - Call the same function block from multiple OBs
   - UNICOS problems

   ```
   ST code
   ...
   a := b*b;
   ...
   
   IL code
   ...
   L b;
   L b;
   *I;
   T a;
   ...
   ```

2. Verification of full UNICOS application
   - More abstraction techniques
   - Compositional verification
Conclusions

- **Modelling**: automatic generation of formal models for 3 different verification tools. ST grammar (not 100%).

- **Reduction techniques**: property preserving reduction techniques applied automatically to the IM.

- **Requirements**: partial solution to formalize specifications.

- **Identify the source of the problem**: counterexample analysis

Automated model checking can be used to find bugs on PLC programs: improving the quality of our PLC code.
“Problems” with model checking

- We check a model, not the real system
  - How we can be sure that it represents the real system?
    - Counterexample + PLC demonstrator

- State space explosion?
  - Abstractions

- Formal requirement?
  - patterns!?

- Formal model?
  - Methodology + tool
Conclusions

- We need more reduction and abstraction techniques.

- **Compositional verification**? Could be applied to our intermediate model?

- **Specification**: currently it is our bottleneck!
Brainstorming: Open questions

- What about Specifications?
  - How do you test?

- Could Formal verification be useful for WinCC OA, Labview, communication protocols, Java code?

- Do you have concurrency problems in your systems?