Efficient representation for formal verification of PLC programs

Vincent Gourcuff, Olivier de Smet and Jean-Marc Faure

LURPA – ENS de Cachan
Outline

- Context of the work

- Construction of formal models

- Representation construction
  - Dependencies analysis
  - Translation of PLC programs into formal models

- Assessment of the representation efficiency

- Conclusions and prospects
Design of dependable controllers for critical systems

Requirements

Control system

Power plant

Refinery

Steam turbine

Context
Impact of the IEC 61508 standard

- Functional safety of E/E/PE safety-related systems
- Industrial need: to reach the SIL3 or SIL4 levels (SIL: Safety Integrity Level)
  - Hardware: redundancies, voting mechanisms, ....; the SIL level is obtained from analysis of probabilistic models (well-known fault forecasting techniques)
  - Software: only recommendations for development; the SIL level depends upon the methods that are employed during the software development

SIL levels and software development

<table>
<thead>
<tr>
<th>Technique</th>
<th>SIL1</th>
<th>SIL2</th>
<th>SIL3</th>
<th>SIL4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semi-formal methods</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>Formal methods including for example, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z</td>
<td>-</td>
<td>R</td>
<td>R</td>
<td>HR</td>
</tr>
</tbody>
</table>
Industrial emerging concern

- To investigate the possibilities of formal verification methods for improving the SIL level of critical systems controlled by PLCs
Formal verification of PLC programs

Scheduler
- Initialization
- Inputs reading
- Program execution
- Outputs updating

Program
- \( O_1 := I_1 \text{ OR } I_2 \);  
- \( O_2 := I_3 \text{ AND } I_4 \);  
- IF \( O_1 \) THEN \( O_3 := I_5 \text{ AND NOT}(I_4) \);  
- END_IF;  
- \( O_4 := RS(O_5, I_1) \);
- \( O_5 := O_2 \text{ AND } O_4 \);
- \( O_1 := \text{NOT}(I_2 \text{ OR } I_4) \);

Informal extrinsic (application-dependant) properties

Formal representation

Formalization

Model – Checker

Property proved or counterexample

[1] Rausch Krogh 98
[2] Frey Litz 00
[3] de Smet Rossi 02
[4] Huuck Lukoschus Bauer 03
[5] …
Technical barriers when model-checking PLC programs

<table>
<thead>
<tr>
<th>Barriers</th>
<th>Potential solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification of properties in formal language</td>
<td>Libraries of parametrized properties</td>
</tr>
<tr>
<td></td>
<td>Use of the results of previous functional or safety analysis</td>
</tr>
<tr>
<td>State space explosion</td>
<td>See what follows</td>
</tr>
<tr>
<td>Counterexamples explanation</td>
<td>Translation of counterexamples in a tailor made representation</td>
</tr>
<tr>
<td></td>
<td>Limitation of the counterexamples length</td>
</tr>
</tbody>
</table>
What is the meaning of a state?

Real world behavior

<table>
<thead>
<tr>
<th>PLC</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclic scheduler</td>
<td>Variables states</td>
</tr>
<tr>
<td>Initialization</td>
<td>Execution state</td>
</tr>
<tr>
<td>Inputs reading</td>
<td></td>
</tr>
<tr>
<td>Program execution</td>
<td></td>
</tr>
<tr>
<td>Outputs updating</td>
<td></td>
</tr>
</tbody>
</table>

Formal representations

- Collection of automata
- Transition relations on variables
  - $\text{VAR}_{n+1} = F(\text{VAR}_n)$
  - $\text{VAR}_{n+1}$ the set of variables, at the next step of calculus
  - $\text{VAR}_n$ the set of variables, at the current step of calculus
- Temporal logic statements
  - CTL
  - LTL

Requirements

- Expected and unexpected states and sequences
Detailed analysis in the case of a PLC program

**PLC program**

- $O_1 := I_1 \text{ OR } I_2$
- $O_2 := I_3 \text{ AND } I_4$
- **IF** $O_1$
- **THEN**
  - $O_3 := I_3 \text{ AND NOT}(I_4)$
  - **END_IF**
- $O_4 := \text{RS}(O_5, I_1)$
- $O_5 := O_2 \text{ AND } O_4$
- $O_1 := \text{NOT}(I_2 \text{ OR } I_4)$

**PLC scheduler**

**Initial state**

- $I_1 = 1$, $I_2 = 0$, $I_3 = 1$, $I_4 = 0$
- $O_1 = 0$, $O_2 = 1$, $O_3 = 0$, $O_4 = 0$, $O_5 = 1$

**Inputs reading**

- $I_1 = 0$, $I_2 = 0$, $I_3 = 1$, $I_4 = 0$
- $O_1 = 0$
- $O_2 = 0$
- $O_3 = 0$
- $O_4 = 1$
- $O_5 = 0$

**Outputs updating**

- $I_1 = 1$, $I_2 = 0$, $I_3 = 1$, $I_4 = 0$
- $O_1 = 1$, $O_2 = 0$, $O_3 = 0$, $O_4 = 1$, $O_5 = 0$

**PLC cycle 1**

- $I_1 = 0$, $I_2 = 1$, $I_3 = 1$, $I_4 = 0$
- $O_1 = 1$
- $O_2 = 0$
- $O_3 = 0$
- $O_4 = 1$
- $O_5 = 1$

**PLC cycle 2**
Features of the considered programs

- PLC programs are executed sequentially;
- only Boolean variables are used;
- internal variables may be included in the program;
- only the following statements of the ST language are allowed:
  - assignment;
  - Boolean operators defined in IEC 61131-3 standard (NOT, AND, OR, XOR)
  - function block (FB) (IEC 61131-3 standard or user-made)
  - control statements, IF and CASE selection statements;
  - iteration statements (FOR, WHILE, REPEAT) are forbidden;
- multiple assignments of the same variable are possible.
Global method

PLC program

Static analysis

Static dependencies

Taking into account execution order

Temporal dependencies

Formal model design

NuSMV model

PLC program

O1 := I1 OR I2;
O2 := I3 AND I4;
IF O1
THEN
O3 := I3 AND NOT(I4);
END_IF;
O4 := RS(O5, I1);
O5 := O2 AND O4;
O1 := NOT(I2 OR I4);

PLC scheduler

Next(I1) := {0, 1};
Next(I2) := {0, 1};
Next(I3) := {0, 1};
Next(I4) := {0, 1};
Next(O2) := Next(I3) & ... := 
case
Next(I1) : 0;
O5 : 1;
1 : O4;
esac;
Next(O5) := Next(O2) & Next(O4);
Next(O1) :=!(Next(I2) | Next(I4));
Static and temporal dependencies construction

<table>
<thead>
<tr>
<th>PLC program</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O_1 := I_1 \text{ OR } I_2; )</td>
</tr>
<tr>
<td>( O_2 := I_3 \text{ AND } I_4; )</td>
</tr>
<tr>
<td>IF ( O_1 ) THEN</td>
</tr>
<tr>
<td>( O_3 := I_3 \text{ AND NOT}(I_4); )</td>
</tr>
<tr>
<td>END_IF;</td>
</tr>
<tr>
<td>( O_4 := RS(O_5, I_1) )</td>
</tr>
<tr>
<td>( O_5 := O_2 \text{ AND } O_4; )</td>
</tr>
<tr>
<td>( O_1 := \text{NOT}(I_2 \text{ OR } I_4); )</td>
</tr>
</tbody>
</table>

| PLC scheduler |

Static dependencies

Temporal dependencies
Translating ST controllers into NuSMV models: general algorithm

BEGIN PLC_prog_TO_NuSMV_model(Pr)
    FOR each statement $S_i$ of Pr:
        IF $S_i$ is an assignment ($V_i := \text{expression}_i$)
            THEN
                FOR each variable $V_k$ in expression$_i$:
                    Replace $V_k$ by the variable pointed out in
                    the temporal dependency ($V_{k,i}$ or $V_{k,i+1}$)
        ELIF $S_i$ is a conditional structure (if cond; then stmt$_1$; else stmt$_2$)
            FOR each variable $V_k$ in cond:
                Replace $V_k$ by the variable pointed out in
                the temporal dependency ($V_{k,i}$ or $V_{k,i+1}$)
            FOR each variable $V_m$ assigned in $S_i$:
                Replace $V_m$ assignment by:
                "case
                cond : assignment of $V_m$ in
                PLC_prog_TO_NuSMV_model(stmt$_1$);
                !cond : assignment of $V_m$ in
                PLC_prog_TO_NuSMV_model(stmt$_2$);
                esac;"
Translating ST controllers into NuSMV models: example

PLC program

\[
\begin{align*}
O_1 & := I_1 \text{ OR } I_2; \\
O_2 & := I_3 \text{ AND } I_4; \\
& \text{IF } O_1 \\
& \text{THEN} \\
& O_3 := I_3 \text{ AND NOT}(I_4); \\
& \text{END_IF;} \\
O_4 & := \text{RS}(O_5, I_1) \\
O_5 & := O_2 \text{ AND } O_4; \\
O_1 & := \text{NOT}(I_2 \text{ OR } I_4); \\
\end{align*}
\]

Next(I1) := \{0, 1\};
Next(I2) := \{0, 1\};
Next(I3) := \{0, 1\};
Next(I4) := \{0, 1\};
Next(O2) := Next(I3) & Next(I4);
Next(O3) :=
\[
\text{case} \\
\text{Next(I1) | Next(I2) : Next(I3) & !Next(I4)}; \\
\text{!Next(I1) | Next(I2) : O3}; \\
esac;
\]
Next(O4) :=
\[
\text{case} \\
\text{Next(I1) : 0;} \\
\text{O5 : 1;} \\
\text{1 : O4}; \\
esac;
\]
Next(O5) := Next(O2) & Next(O4);
Next(O1) := !Next(I2) & Next(I4);

From generic models library

Temporal dependencies

\[
\begin{align*}
O_{5,i} \\
I_{1,i+1} \\
O_{4,i+1} \\
O_{4,i} \\
\end{align*}
\]
Translating ST controllers into NuSMV models: comparison to previous approaches

Next(I1) := \{0, 1\};
Next(I2) := \{0, 1\};
Next(I3) := \{0, 1\};
Next(I4) := \{0, 1\};
Next(O2) := Next(I3) & Next(I4);
Next(O3) :=
case
    Next(I1) | Next(I2) : Next(I3) & !(Next(I4));
    !(Next(I1) | Next(I2)) : O3;
esac;
Next(O4) :=
case
    Next(I1) : 0;
    O5 : 1;
    1 : O4;
esac;
Next(O5) := Next(O2) & Next(O4);
Next(O1) := !(Next(I2) | Next(I4));

No intermediary variables states

No "line_counter" because the execution state is no more useful

No "end_of_cycle" variable; one cycle is reduced to only one state
Method for assessing the representation efficiency

- **Comparison of two representations:**
  - Representation proposed in this paper that relies upon temporal dependencies
  - Standard representation that does not use temporal dependencies and that models the sequential execution of the program (see for instance [dSR 02])

- **Comparison criteria:**
  - Size of the state space
  - Time for proving properties
  - Memory consumption when proving properties

- **Remark:** Equivalence of both representations w.r.t. inputs/outputs has been proved using behavioural equivalence techniques
Basic example

Comparison of the state spaces sizes

- state space reduction (about 15 times)
- reduction of the maximum distance between states (system diameter) (11 times shorter)

⇒ indirect consequence: trace of counterexample reduced
Fishertechnik example [Special session at ACC02]

- known tested machining line
- already written control program
- expected behavior known
- small size system
  - 15 Inputs / 15 Outputs

Comparison of the proof process duration and of the memory consumption

<table>
<thead>
<tr>
<th></th>
<th>Standard representation</th>
<th>Proposed representation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>liveness property</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$AG ((dp_head_motor_up) \Rightarrow EF (!dp_head_motor_up))$</td>
<td>5h / 526MB</td>
<td>2s / 8MB</td>
</tr>
<tr>
<td><strong>safety property</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$AG (!INI \Rightarrow !(dp_head_motor_up &amp; dp_motor_down))$</td>
<td>20min / 200MB</td>
<td>2s / 8MB</td>
</tr>
</tbody>
</table>
Conclusions

- **Efficiency of the representation assessed**
  - Previous examples
  - Examples issued from real industrial applications

- **Translation of PLC programs in ST language**
  - Fully automated
  - Fast (some seconds or tens of seconds for industrial programs)

On-going works

- **Taking into account integer and real variables**
  - PLCs do not deal only with logic variables

- **Extension of the library of function blocks**
  - Design of formally verified tailor-made function blocks
Efficient representation for formal verification of PLC programs

Thank you

for attention