Counterexample Guided Abstraction Refinement for PLCs

Jörg Brauer

Embedded Software Laboratory
RWTH Aachen University
brauer@embedded.rwth-aachen.de

University of Bremen, 15.11.2011

Myself

• Studied at CAU Kiel
• Spent 1,5 years @ NICTA in Sydney
• Diploma (computer science) in 09/2008
• Since then
  – Embedded Software Laboratory at RWTH Aachen
  – [mc]square (project lead since 01/2010)
• Research interest
  – Circles around automatic abstraction
  – PhD thesis finished (hopefully) in spring 2012
  – Supervisors: S. Kowalewski (RWTH) & A. King (Kent)
Overview

• Introduction & motivation
  – PLCs
  – Formal verification
• Formal Methods
  – Model checking of PLC programs
  – Refinement techniques
• [mc]square
• Demonstration of [mc]square
PLCs: Cyclic Scanning Mode

1. Sense inputs  
2. Process  
3. Write outputs

PLCs: General Layout

- Defined in the standard IEC 61131
PLCs: Programming Languages

<table>
<thead>
<tr>
<th>Ladder Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Ladder Diagram Image]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction List</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD input0</td>
</tr>
<tr>
<td>ADD 50</td>
</tr>
<tr>
<td>GT 100</td>
</tr>
<tr>
<td>JMPC label</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function Block Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Function Block Diagram Image]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Structured Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF input0+50 &gt; 100 THEN</td>
</tr>
<tr>
<td>output0 := 1;</td>
</tr>
<tr>
<td>ELSE</td>
</tr>
<tr>
<td>output0 := 0;</td>
</tr>
<tr>
<td>ENDIF;</td>
</tr>
</tbody>
</table>

Formal Verification

- **Does the program behave as desired?**
  - Functional requirements
  - Non-functional requirements

- **Model checking**
  - Are desired states always reachable?
  - Are critical states never reachable?
Model Checking

- Specification
  - CTL (ACTL)
  - Propositions about input, internal and output variables
- Model
  - State is a tuple (input, variables, output)
  - Transition between states for each possible input
  - Takes PLC cycle into account
- Does the model obey the specification?

Counterexample Guided Abstraction Refinement

1. Choose initial abstraction
2. Model check abstraction
   - true: Specification satisfied
   - false
     - Counterexample feasible?
       - yes: Specification violated
       - no: Refine abstraction
Overview

• Introduction & motivation
  – PLCs
  – Formal verification

• Formal Methods
  – Model checking of PLC programs
  – Refinement techniques

• [mc]square
• Demonstration of [mc]square

Example PLC Program

<table>
<thead>
<tr>
<th>input0, input1</th>
<th>input1</th>
<th>output0</th>
<th>output0</th>
<th>var0</th>
<th>Type BYTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td>OUTPUT</td>
<td>GLOBAL</td>
<td>GLOBAL</td>
<td></td>
<td>0..255</td>
</tr>
</tbody>
</table>

```plaintext
IF input1<50 THEN
  output0 := var0;
ELSE
  GT 100
  SUB <MBC:=input1;
ENDIF;
LD input1
ST var0
RET

Label:
LD var0
ST output0
RET
```
Building the State Space

<table>
<thead>
<tr>
<th>input0, input1</th>
<th>INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>output0</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>var0</td>
<td>GLOBAL</td>
</tr>
<tr>
<td>Type BYTE</td>
<td>0..255</td>
</tr>
</tbody>
</table>

LD input0
ADD 50
GT 100
JMPC Label

LD input1
ST var0
RET

Label:
LD var0
ST output0
RET

input0 = 0 1 2 3 ... 0 ... 255
input1 = 0 0 0 0 ... 1 ... 255

State space is built using simulation!

Building the Abstract State Space

<table>
<thead>
<tr>
<th>input0, input1</th>
<th>INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>output0</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>var0</td>
<td>GLOBAL</td>
</tr>
<tr>
<td>Type BYTE</td>
<td>0..255</td>
</tr>
</tbody>
</table>

LD input0
ADD 50
GT 100
JMPC Label

LD input1
ST var0
RET

Label:
LD var0
ST output0
RET

input0 = [0, 49] [50, 255] [50, 255] ... [50, 255]
input1 = [0, 255] [0, 0] [1, 1] ... [255, 255]
Abstract Domains

• Intervals
  – [1, 50] + [2, 3] = [3, 53]
• Bit sets
  – Each bit is 0, 1 or ⊥
  – 010 ⊥ ⊥ 1 & 010010 = 0100 ⊥ 0
• We use the reduced cardinal product of intervals and bit sets

Example (cont.)

<table>
<thead>
<tr>
<th>Program</th>
<th>Accumulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>[0, 255]</td>
</tr>
<tr>
<td>ADD</td>
<td>[50, 305]</td>
</tr>
<tr>
<td>GT</td>
<td>[0, 1]</td>
</tr>
<tr>
<td>J MPC</td>
<td></td>
</tr>
</tbody>
</table>

• Let’s start with input0 = [0, 255]
• Condition jump (JMP) demands a concrete value in accumulator
• This poses a constraint on the abstract value in the accumulator
• Intuitively: Restart cycle with abstract values [0, 49] and [50, 255] for input0 to satisfy constraint
Constraints on Abstract Values

- \( cs_f(v) : \Leftrightarrow \) Abstract value \( v \) is consistent under predicate \( f \)
- Example
  - \( cs_{>50}([0, 255]) \) is false
  - \( cs_{>50}([51, 101]), cs_{>50}([3, 7]) \) are true

- \( cs_{\text{sing}}(v) : \Leftrightarrow v \) represents a single value
- Idea:
  - Extend constraints to expressions
  - Guarding conditional jumps, etc
  - Next: Formal model for PLC programs

SSA Form

- If \( cs_{\text{sing}}(\text{acc}^{(2)}) \) is not fulfilled, \( \text{input}^{(0)} \) should be split
- Next step: Transform \( cs_{\text{sing}}(\text{acc}^{(2)}) \) into a constraint on \( \text{input}^{(0)} \)
Transforming Constraints

• \( cs_{f_1}(e_1) \vdash cs_{f_2}(e_2) \) holds iff \( cs_{f_1}(e_1) \) implies the consistency of \( cs_{f_2}(e_2) \)
• E.g. \( cs_{>50}(a + 5) \vdash cs_{>45}(a) \)

### Constraint Guards

- Constraint guards are needed
  - for deterministic control flow
  - for some hardware function blocks (e.g. timers) that require concrete values
  - to guarantee that the atomic propositions of the model checker have a consistent truth value
- If those constraints are not fulfilled they are transformed into constraints on variables
Refinements of Local Variables

- Refinement loop: Begin with $\top$ for all inputs
- Transform constraints to constraints on inputs
- Refine inputs and restart cycle
- Each restart refines an abstract value, so the refinement process eventually terminates
- Simple support for global variables:
  - Protect all global variables with single value constraints (no non-determinism in state space)
  - We can do better though

Refinements of Global Variables

IF input0+50 > 100 THEN
  output0 := var0;
ELSE
  var0 := input1;
ENDIF;

\[
\begin{align*}
\text{input0} & = [0, 50] \\
\text{input1} & = [0, 49] \\
\text{var0} & = [0, 49] \\
\text{output0} & = [0, 0]
\end{align*}
\]

\[
\begin{align*}
\text{input0} & = [51, 255] \\
\text{input1} & = [0, 255] \\
\text{var0} & = [0, 49] \\
\text{output0} & = [0, 49]
\end{align*}
\]

\[
\begin{align*}
\text{input0} & = [0, 50] \\
\text{input1} & = [50, 255] \\
\text{var0} & = [50, 255] \\
\text{output0} & = [0, 0]
\end{align*}
\]

\[
\begin{align*}
\text{input0} & = [51, 255] \\
\text{input1} & = [0, 255] \\
\text{var0} & = [50, 255] \\
\text{output0} & = [50, 255]
\end{align*}
\]

Lemma: $cs_{<50}(\text{var0})$

$AG \text{output0} < 50$
Refinements of Global Variables

• Storing abstract values in states possibly allows new behavior
  - A valid ACTL formula is also valid in the concrete state space
  - For an invalid ACTL formula, we have to check whether we found a real counterexample
  - This is achieved by rebuilding the state space using the lemmas as refinements

Overview
Case Studies

<table>
<thead>
<tr>
<th>Abstraction technique</th>
<th># stored states</th>
<th># created states</th>
<th>State space size [MB]</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without</td>
<td>780 172</td>
<td>199 724 033</td>
<td>1 704</td>
<td>5 633</td>
</tr>
<tr>
<td>Only inputs</td>
<td>132 242</td>
<td>3 155 467</td>
<td>351</td>
<td>326</td>
</tr>
<tr>
<td>All variables</td>
<td>75 203</td>
<td>1 098 220</td>
<td>163</td>
<td>99</td>
</tr>
</tbody>
</table>

- Function block for monitoring a guard lock (PLCopen)
- 8 Boolean inputs and 5 outputs
- We used an implementation with 300 lines of IL code and 16 internal variables

SAT Based Techniques

- For each variable $x$, write
  \[
  x = \sum_{i=0}^{n-1} 2^i x_i
  \]

- Represent program as CNF formula (bit-blasting the instructions)

- Example: $a' := a + x$

  \[
  (a, x, a') = \left\{ \begin{array}{l}
  (\bigwedge_{i=0}^{7} a'_i \leftrightarrow a_i \oplus x_i \oplus c_i) \land \neg c_0 \land \\
  (\bigwedge_{i=0}^{6} c_{i+1} \leftrightarrow a_i \land x_i) \lor (a_i \land c_i) \lor (x_i \land c_i) \end{array} \right.
  \]

- Infer interesting properties using a SAT solver
SAT Based Range Analysis

- What is the lower bound \( x_l \) for \( x \) in

\[
f'(\langle x_7, \ldots, x_0, y_7, \ldots, y_0 \rangle) := (x + y < 3)
\]

- If \( x \) is bound to the interval \([a, b]\), we set

\[
f := f' \land (x \geq a) \land (x \leq b).
\]

- Infer the lower bound \( x_l \) by testing whether \( f \land \neg x_{n-1} \) is satisfiable.
  - satisfiable \( \implies \) MSB of \( x_l \) 0.
  - not satisfiable \( \implies \) MSB of \( x_l \) 1.

- Repeat for \( x_{n-2} \)

---

Algorithm by Picture

\[
x \leq x_u
\]

\[
0 \quad 2^7 \quad 2^8 - 1
\]

\[
2^7 + 2^6
\]

\[
2^7 + 2^5
\]

\[
\ldots
\]

\[
x_u = 2^7 + 2^5 + 2^1
\]
SAT Based Refinements

• Mostly independent of abstract domain
  – Works for conjunctive linear template constraint domains
  – Disjunctive domains are tricky
• Have tried this for a few relational domains
  – Octagons
  – Bit-wise congruences

Overview

• Introduction & motivation
  – PLCs
  – Formal verification
• Formal Methods
  – Model checking of PLC programs
  – Refinement techniques
• [mc]square
• Demonstration of [mc]square
[mc]square

- [mc]square
  - Simulation, model checking, static analysis of binary code for embedded platforms
  - Support for different platforms
    - ATmega
    - Intel MCS-51
    - Renesas R8C/23 Tiny
    - PLCs
[mc]square for PLC Code

- Supported Languages
  - Instruction List (IEC, Siemens)
  - Structured Text (since February)
  - FBD (work in progress)
- Abstraction techniques
  - Intervals, bit sets
  - Abstraction refinement
- User defined environments
- Found bugs in real programs
- Mostly automatic
- Scales well (more complex programs needed!)

Intermediate Representation

Model Checker

IL simulator

ST simulator

FBD simulator

S7-simulator

Intermediate Representation

Static Analysis

integers

intervals

bit sets

symbolic expressions

© 2011 Jörg Brauer, Embedded Software Laboratory, RWTH Aachen University
Related Work

- [BBWK09] B. Schlich, J. Brauer, J. Wernerus, and S. Kowalewski: "Direct Model Checking of PLC Programs in IL", *DCDS 2009*

© 2011 Jörg Brauer, Embedded Software Laboratory, RWTH Aachen University

Conclusion & Future Work

- **Conclusion**
  - Formal methods for PLC code
    - Debugging / Simulation
    - Verification
  - Possible with [mc]square
- **Future Work**
  - User interface improvements
  - Industrial-size case studies

© 2011 Jörg Brauer, Embedded Software Laboratory, RWTH Aachen University
Demonstration