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

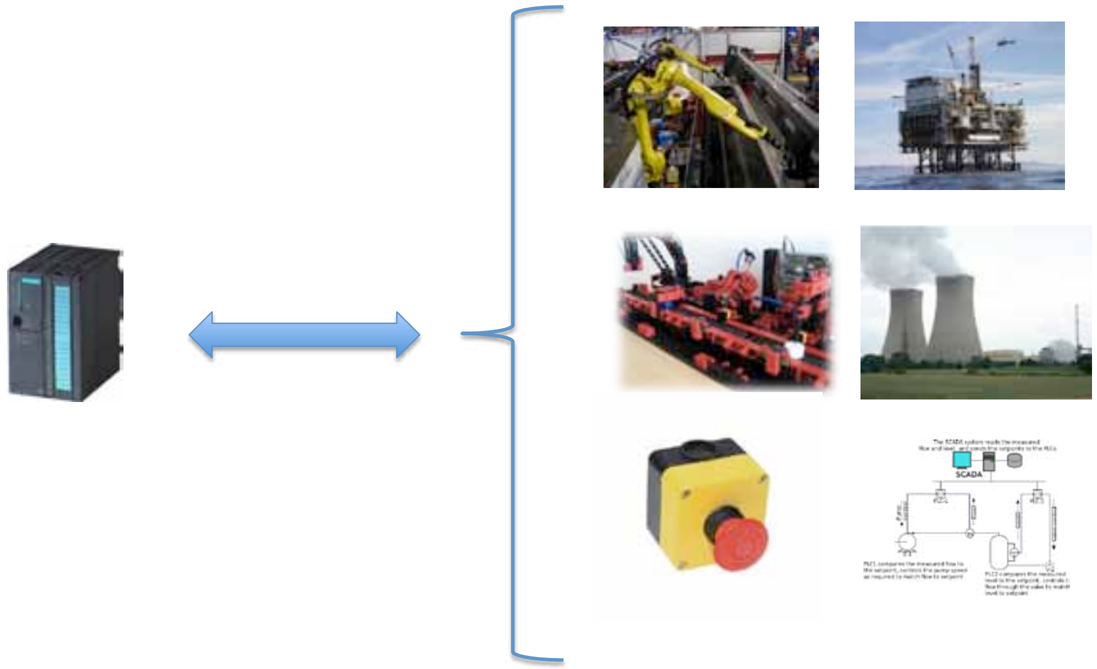
Myself

- 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)

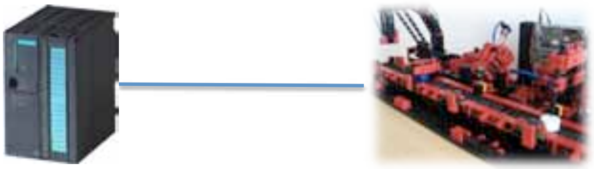
Overview

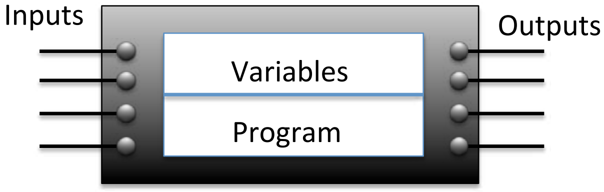
- Introduction & motivation
  - PLCs
  - Formal verification
- Formal Methods
  - Model checking of PLC programs
  - Refinement techniques
- [mc]square
- Demonstration of [mc]square

Programmable Logic Controllers



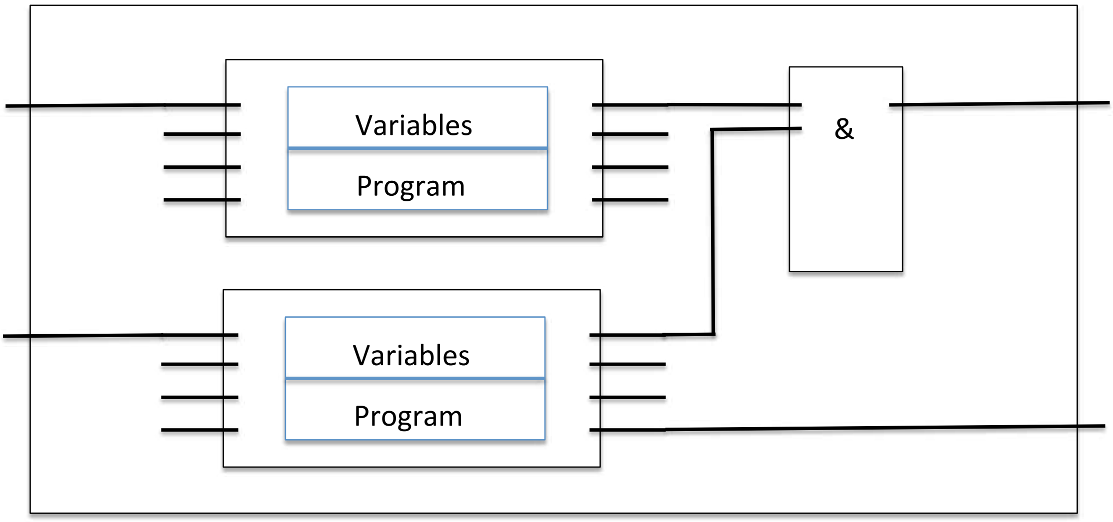
PLCs: Cyclic Scanning Mode





1. Sense inputs
2. Process
3. Write outputs

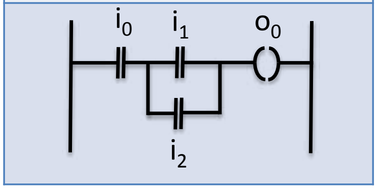
PLCs: General Layout



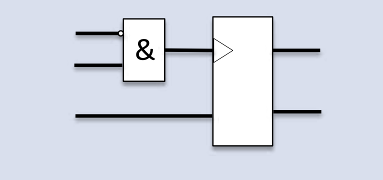
- Defined in the standard IEC 61131

PLCs: Programming Languages

Ladder Diagram



Function Block Diagram



Instruction List

```
LD      input0
ADD   50
GT      100
JMPC label
```

Structured Text

```
IF input0+50 > 100 THEN
        output0 := 1;
ELSE
        output0 := 0;
ENDIF;
```

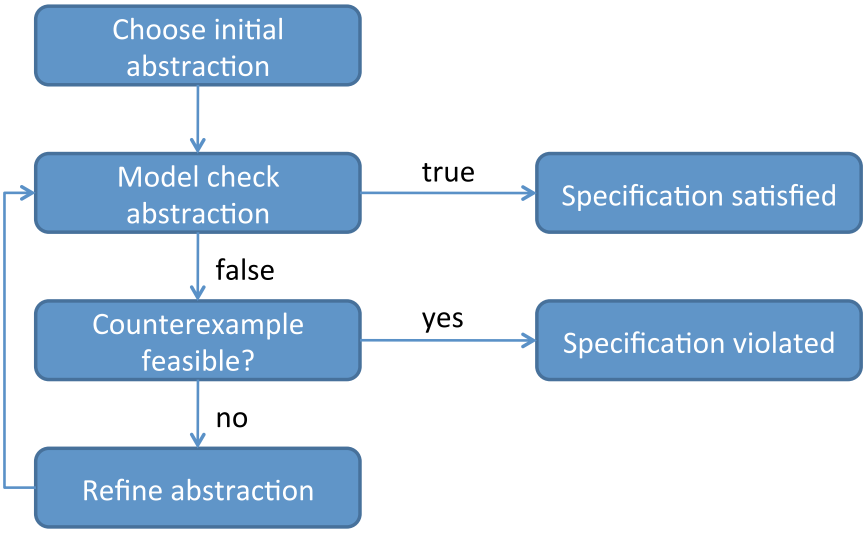
Formal Verification

- *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



Choose initial abstraction

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

input0, input1	*INPUT*
output0	*OUTPUT*
var0	*GLOBAL*
Type BYTE	0..255

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

```
          LD      input0
          ADD     50
          GT      100
          JMPC    Label
          LD      input1
          ST      var0
          RET

Label:
          LD      var0
          ST      output0
          RET
```

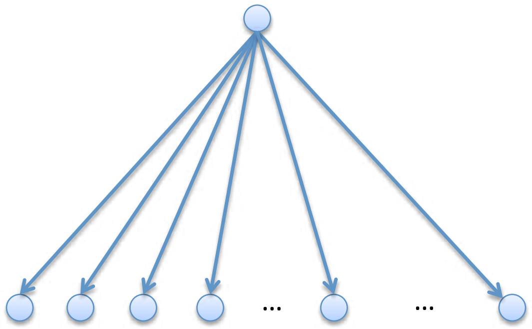
Building the State Space

input0, input1	*INPUT*
output0	*OUTPUT*
var0	*GLOBAL*
Type BYTE	0..255

```
        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!

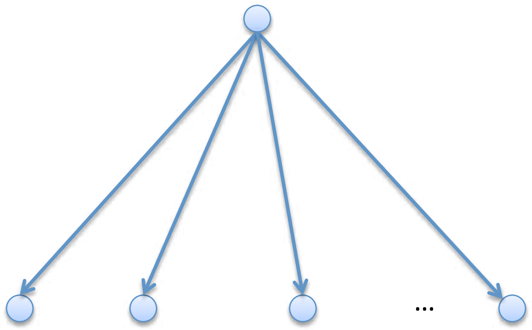
Building the Abstract State Space

input0, input1	*INPUT*
output0	*OUTPUT*
var0	*GLOBAL*
Type BYTE	0..255

```
        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.)

Program

```
        LD    input0
        ADD   50
        GT    100
        JMPC  Label
...
```

Accumulator

[0, 255]
[50, 305]
[0, 1]

- Let’s start with input0 = [0, 255]
- Condition jump (JMPC) 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

- $cs_f(v)$:⇔ 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*

- $cs_{sing}(v)$:⇔ v represents a single value

- Idea:
  - Extend constraints to expressions
  - Guarding conditional jumps, etc
  - Next: Formal model for PLC programs

SSA Form

Program

```
      LD    input0
      ADD  50
      GT     100
      JMPCLabel
...
```

SSA form

$acc^{(0)} := input_0^{(0)}$

$acc^{(1)} := acc^{(0)} + 50$

$acc^{(2)} := acc^{(1)} > 100$

guard(cs_{sing} ($acc^{(2)}$)

…

- If $cs_{sing}(acc^{(2)})$ is not fulfilled, $input0^{(0)}$ should be split
- Next step: Transform $cs_{sing}(acc^{(2)})$ into a constraint on $input0^{(0)}$

Transforming Constraints

- $\text{cs}_{f1}(e_1) \vdash \text{cs}_{f2}(e_2)$ holds iff $\text{cs}_{f1}(e_1)$ implies the consistency of $\text{cs}_{f2}(e_2)$
- E.g. $\text{cs}_{>50}(a + 5) \vdash \text{cs}_{>45}(a)$

```
$acc^0 := input_0^0$
$acc^1 := acc^0 + 50$
$acc^2 := acc^1 > 100$
$guard(cs_{sing}(acc^2))$
...
```

$\text{cs}_{sing}(\text{acc}^2)$

Constraint Guards

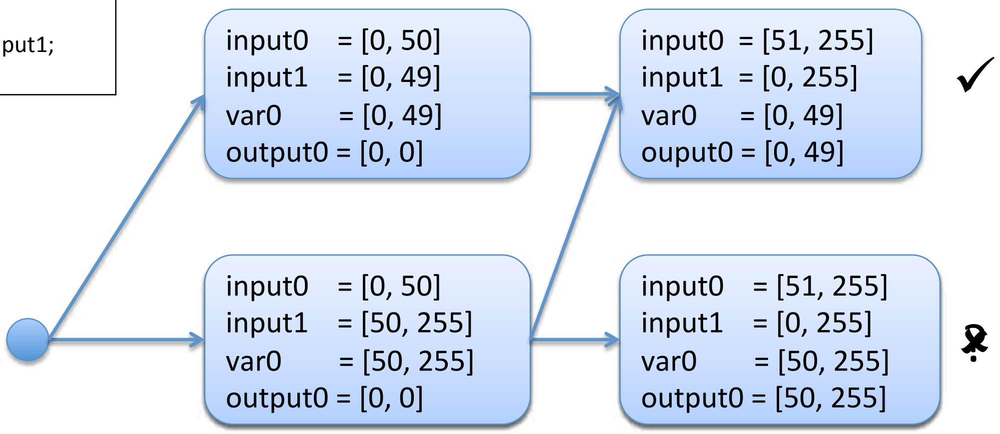
- Constraint guards are needed
  - for deterministic control flow
  - for some hardware function blocks (e.g. timers) that require concrete values
  - to guarantee that the atomic propositions of the model checker have a consistent truth value
- If those constraints are not fulfilled they are transformed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



input0 = [0, 50]
input1 = [0, 49]
var0 = [0, 49]
output0 = [0, 0]

input0 = [51, 255]
input1 = [0, 255]
var0 = [0, 49]
ouput0 = [0, 49]

✓

input0 = [0, 50]
input1 = [50, 255]
var0 = [50, 255]
output0 = [0, 0]

input0 = [51, 255]
input1 = [0, 255]
var0 = [50, 255]
output0 = [50, 255]

✗

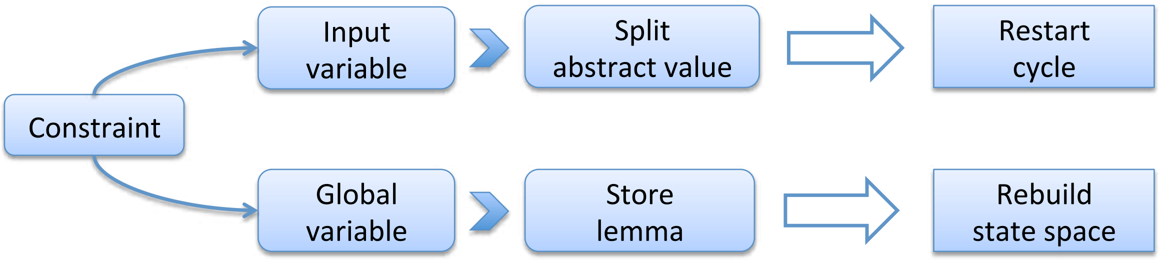
Lemma: cs$_{< 50}$(var0)

AG 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



Constraint

Input variable

Split abstract value

Restart cycle

Global variable

Store lemma

Rebuild state space

Case Studies

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

- 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: 9{ I Q 9 v ²

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

- Infer *interesting* properties using a SAT solver

SAT Based Range Analysis

- What is the lower bound x_l for x in

$$\mathrm{f}'(\langle \mathrm{x}_7, \ldots, \mathrm{x}_0, \mathrm{y}_7, \ldots, \mathrm{y}_0 \rangle) := (\mathrm{x} + \mathrm{y} < 3)$$

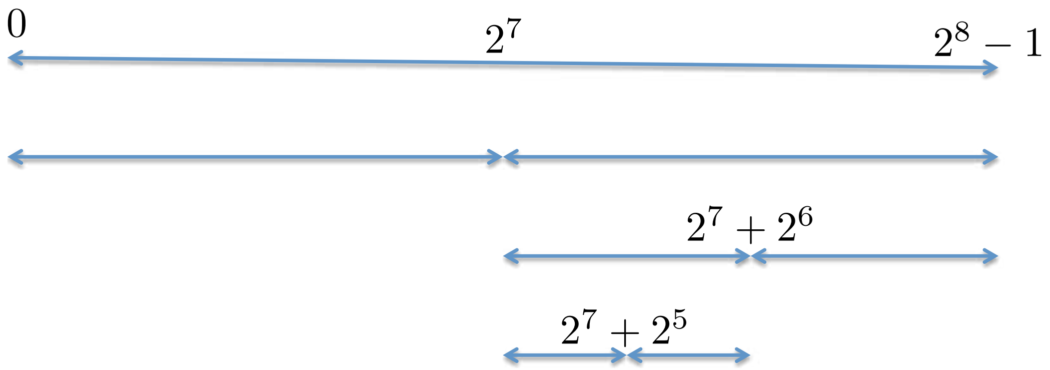
- If x is bound to the interval $[\mathrm{a}, \mathrm{b}]$, we set

$$\mathrm{f} := \mathrm{f}' \wedge (\mathrm{x} \geq \mathrm{a}) \wedge (\mathrm{x} \leq \mathrm{b}).$$

- Infer the lower bound x_l by testing whether • $:= \mathrm{f} \wedge \neg \mathrm{x}_{\mathrm{n}-1}$ is satisfiable.
  - • satisfiable $\Longrightarrow$ MSB of x_l 0.
  - • not satisfiable $\Longrightarrow$ MSB of x_l 1.
- Repeat for $\mathrm{x}_{\mathrm{n}-2}$

Algorithm by Picture

$x \leq x_u$



…

● $x_u = 2^7 + 2^5 + 2^1$

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

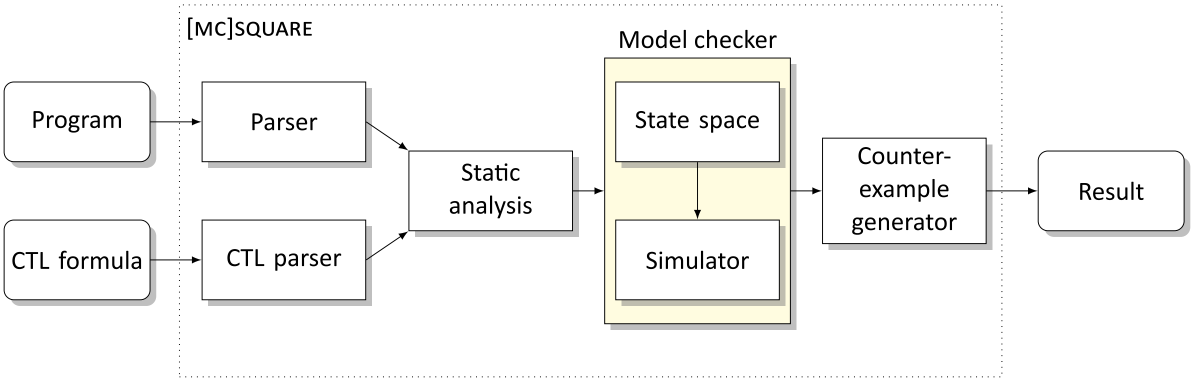
Overview

- 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**

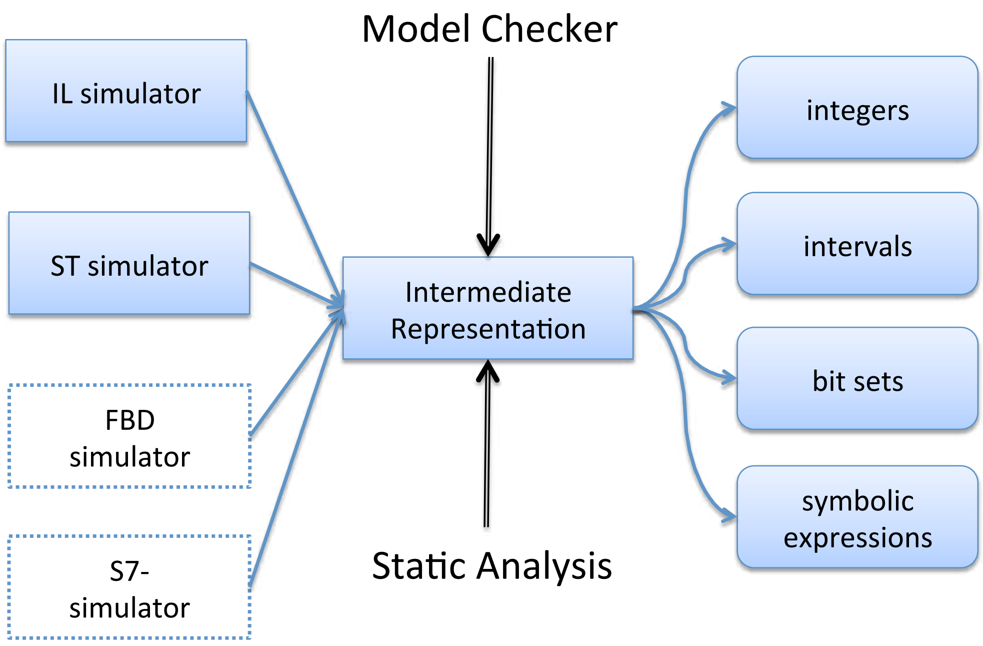
[mc]square



[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

Related Work

- [BBWK09] B. Schlich, J. Brauer, J. Wernerus, and S. Kowalewski: "Direct Model Checking of PLC Programs in IL", *DCDS 2009*
- [BFK+10] S. Biallas, G. Frey, S. Kowalewski, B. Schlich, and D. Soliman: "Formale Verifikation von Sicherheits-Funktionsbausteinen der PLCopen auf Modell- und Code-Ebene", *EKA 2010*
- [BBK10] S. Biallas, J. Brauer, and S. Kowalewski: "Counterexample-Guided Abstraction Refinement for PLCs", *SSV 2010*
- [BBSK10] S. Biallas, J. Brauer, S. Kowalewski, and B. Schlich: "Automatically Deriving Symbolic Invariants for PLC Programs Written in IL", *FORMS/FORMAT 2010*
- [BBK11] S. Biallas, J. Brauer, S. Kowalewski: SAT-Based Abstraction Refinement for Programmable Logic Controllers", *DCDS 2011*
- [BK10] J. Brauer, A. King: "Automatic Abstraction for Intervals using Boolean Formulae", SAS 2010
- [BKK10] J. Brauer, A. King, S. Kowalewski: "Range Analysis of Microcontroller Code using Bit-Level Congruences", *FMICS 2010 + SCP*
- [BK11] J. Brauer, A. King: "Transfer Function Synthesis without Quantifier Elimination", ESOP 2011 + LMCS
- [BKK11] J. Brauer, A. King, J. Kriener: "Existential Quantification as Incremental SAT", CAV 2011

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

Demonstration