Dániel Darvas (BME, CERN)

Practice-Oriented Formal Methods to Support the Software Development of Industrial Control Systems

Public Ph.D. defence
15th May 2017, Budapest

Advisors:
István Majzik, Ph.D. (BME)
Enrique Blanco Viñuela, Ph.D. (CERN)

http://mit.bme.hu/~darvas/phd/
http://cern.ch/ddarvas/phd/
Industrial control systems (ICS)

- **Goal:** safe and efficient operation of industrial plants

- **Typical implementation:** PLCs

- Often not safety-critical, but reduced availability = loss of money
Quality improvements for ICS software

- But **costs** and **benefits** have to be in balance
- Methods are needed which **can be applied in practice with reasonable effort**
- ICS domain: **special needs**, but also **special properties** that can be exploited in verification
Formal methods

- **Precise** methods for *description* and *analysis* of systems and *requirements*

- **Model checking** (*verification method*):
  
  Formal model + Formal requirement → **Model checker** → Verification result

- **Key challenges:**
  
  ![Performance](image) ![Usability](image)
Contributions

Make formal methods more applicable to ICS in practice!

1. Improving the performance of model checking: B-I-Sat – a new, integrated algorithm

2. Making model checking usable for PLC programs by developers

3. Precise behaviour specification for PLC program modules
Thesis 1

Improving the performance of model checking: Bounded Iterative Saturation (B-I-Sat)
Motivation

− Model checking is **computationally difficult**

− Different approaches to facilitate:
  - **Symbolic**: operations on encoded state sets (e.g. saturation)
  - **Bounded**: checking only a part of the model
  - **Abstract**: simplifying the model

− **Saturation + bounded**: Will it **improve** the performance?
  - **Drawback**: storage and computation overhead
B-I-Sat workflow

Th1.1: Definition of basic workflow and two strategies

State space exploration (bound $b$) → Storage conversion → Requirement evaluation

Terminate? (yes/no)

Th1.2: Definition of termination conditions

Th1.3: Compacting saturation (3rd strategy, reduced overhead)

Th1.4: Performance evaluation

Preparation for next iteration

$b := b + B$
Iteration strategies

- **Restarting**: each iteration starts from model’s initial state(s)

- **Continuing**: each iteration starts from end of previous iteration

- **Compacting**: starts from frontier set without distance
Scalability of the algorithms

Execution time of evaluating $EF(q_i = 0)$ on the Queen–10 model ($B_0 = B = 2$)

- Unbounded
-Restarting/Continuing
  - Compacting

"Shallow" requirements

"Deep" requirements

Th1.4: Performance evaluation
Summary of contributions

1. **B-I-Sat**: Bounded saturation-based model checking
   - New integrated algorithm
   - Three strategies
   - Termination conditions
   - Detailed evaluation
Publications – Thesis 1


Thesis 2
Making model checking usable for PLC programs by developers
Motivation

− Formal verification (e.g. model checking) is good to find “high quality bugs”

− Make the use of model checking as simple as possible

− Targeted approach:
Model checking workflow

Th2.1: Design of intermediate model
Support for reductions, multiple MCs

Th2.2: Design of model reductions

Th2.3: Extension to safety PLC programs (STL)

Th2.4: Implementation and real-life evaluation

PLC code → Intermediate model → Model checker

Requirements patterns → Temporal logic requirement

Reductions

Satisfied

Verification report

Counter-example

Not satisfied

Replaceable model checker
Intermediate model (IM)

- Simple, but **expressive** enough
- **Independent** from model checkers
Automated, property-preserving reductions

- **General** reductions
- **Domain-specific** reductions *(what is observable?)*
- **Requirement-specific** reductions *(what affects the result?)*

---

**Graph:**

- **Number of elements**
- **Iterations**:
  - Transitions
  - Assignments
  - Variables

- **Labels:**
  - **SM18-PLCSE, safety of the power converter**
  - **Th2.2: Design of model reductions**

- **Legend:**
  - Impossible to execute the model checker
  - Slow model checking
  - Fast model checking

---

100,000s of elements

100s of elements
Extension to safety systems

Normal programs
- No constraint
- Mainly SCL (at CERN)
  \[ r := (a \geq b); \]
- “Straightforward” semantics

Safety programs
- Additional constraints
- Low-level languages
- Imprecise semantics

Additional needs:
- Determining the semantics
- STL to IM translation
- New reductions

Th2.3: Extension to safety PLC programs (STL)
PLCverif

- Tool for the non-expert users

Th2.4: Implementation and real-life evaluation
Summary of contributions

1. **B-I-Sat**: Bounded saturation-based model checking
   - new integrated algorithm
   - three strategies
   - termination conditions
   - detailed evaluation

2. **PLCverif**: User-friendly model checking for developers
   - intermediate model
   - safety PLC programs
   - reductions
   - implementation
   - evaluation
Publications – Thesis 2


## Publications – Thesis 2

<table>
<thead>
<tr>
<th>Int. conference (7)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Tech. rep.</th>
</tr>
</thead>
</table>
Thesis 3
Precise behaviour description for PLC program modules: PLCspecif
Motivation

- **Difficult to formalise** certain requirements (even with requirement patterns)
- **Thorough analysis** needs many requirements
- **Formal specification** methods may be useful, but need **too much knowledge and effort**

→ **Describe** the expected **behaviour** in a **consistent**, **complete**, **domain-specific** way, **adapted to ICS**
Benefits of formal specification

- **Clean, unambiguous description**
- + additional methods

**Th3.1**: Formal syntax and semantics for PLCspecif

- **Invariant checking** (Th3.2)
- **Code generation** (Th3.3)
- **Conformance checking** (Th3.4)
Metamodel and semantics

Concrete syntax

Abstract syntax

Semantics (timed automaton)

Th3.1: Formal syntax and semantics definition

---

```
// Translating triggered transitions as TA transitions
24  B' ← false;
25  foreach t ∈ m.transitions do
26      if t.trigger ≠ ∅ then
27          srcActive ← sourceActive(t, activeState);
28          g ← (srcActive ∧ mapping(t.trigger, TAVariable) ∧ t.guard);
29          is true and the event is triggered
30          B' ← (B' ∨ g);
31      // After firing the target state will be activated
32      if ¬t.to is a DeepHistoryState then
33          vc ← (activeState := literal(t.to));
34          foreach h ∈ historyStatesToUpdate(t.to) do
35              // ○ is the concatenation operator
36              vc ← vc ○ (mapping(h, TAVariable) := literal(t.to));
37      else
38          vc ← (activeState := mapping(t.to, TAVariable));
39      t3 ← new TATransition(from: l2, to: l3, clock_guard: ∅, data_guard: g, var_chng: vc, ta: a);
```
PLCspecific (example)

**Example Module**

**Assigned inputs:**
- ValueReq : INT16
- EnableReq_fromLogic : BOOL
- EnableReq_fromScada : BOOL
- EnableReq_fromField : BOOL
- DisableReq : BOOL
- PMin : INT16 param
- PMax : INT16 param

**Assigned outputs:**
- Value : INT16
- Status : BOOL

**Input definitions:** — (none)

**Event definitions:**
- @disable ← rising_edge(DisableReq) (pri=1)
- @enable ← EnableReq_fromLogic OR EnableReq_fromScada OR EnableReq_fromField (pri=2)

**Core logic (state machine)**

```
<table>
<thead>
<tr>
<th>Disabled</th>
<th>@disable</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>@enable</td>
</tr>
</tbody>
</table>
```

**Output definitions:**
- _Value = ValueReq < PMin OR ValueReq > PMax OR ValueReq
- Value = in_state(Enabled) ValueReq
- Status = in_state(Enabled)

**Invariant properties:**
- ALWAYS PMin ≤ Value ≤ PMax ASSUMING PMin ≤ PMax

**Separation of concerns**

**Unifies different semi-formal formalisms**

**Supports verification**

**Th3.1:** Formal syntax and semantics definition

**Th3.2**
Invariant checking

- Complements the precise description of functionality
- Helps to generate code that matches the intentions
- Based on the workflow presented in Thesis 2

Th3.2: Invariant checking

Code generation

- Constructing SCL code with
  - Matching behaviour (correct) – based on the semantics
  - Good readability (maintainable, modifiable)
- Mapping on the level of semantics (TA – CFA)

Th3.3: Code generation
Conformance checking

- Implementation corresponds to specification?
  - Use cases: legacy code, safety PLCs, manual modification

- Contributions
  - Permissive conformance relations for PLC programs
  - Method for checking these relations, mapped to model checking problems (PLCverif)

Th3.4: Definition of conformance relations and conformance checking
## Conformance relations

<table>
<thead>
<tr>
<th>Relation</th>
<th>Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_{\text{conf}}_1$</td>
<td><strong>Strict equivalence</strong></td>
</tr>
<tr>
<td>$p_{\text{conf}}<em>2$, $p</em>{\text{conf}}_3$</td>
<td><strong>Permissive relation with fixed delay</strong>&lt;br&gt;Constant shift allowed</td>
</tr>
<tr>
<td>$p_{\text{conf}}_4$</td>
<td><strong>Permissive relation with variable delay</strong>&lt;br&gt;Local differences allowed</td>
</tr>
<tr>
<td>$p_{\text{conf}}_5$</td>
<td>$p_{\text{conf}}_4$ + total pulse length preservation</td>
</tr>
<tr>
<td>$p_{\text{conf}}_6$</td>
<td>$p_{\text{conf}}_4$ + edge number preservation</td>
</tr>
</tbody>
</table>

**Th3.4:** Definition of conformance relations and conformance checking
Summary of contributions

1. **B-I-Sat**: Bounded saturation-based model checking
   - New integrated algorithm
   - Three strategies
   - Termination conditions
   - Detailed evaluation

2. **PLCverif**: User-friendly model checking for developers
   - Intermediate model
   - Safety PLC programs
   - Reductions
   - Implementation
   - Evaluation

3. **PLCspecif**: Complete behaviour description for PLC program modules
   - Syntax, semantics
   - Invariant checking
   - Code generation
   - Conformance checking
### Publications – Thesis 3

Application examples
Paks NPP – PRISE

- **PRISE**: detection logic for pri-sec leakage
- First complete verification: with saturation
- **B-I-Sat** reduces the run time for “shallow” requirements

Photo: http://dailynewshungary.com/
CERN – Magnet testing

- Complex PLC-based safety logic
- Formal methods from design phase

- PLCverif + requirement patterns:
  12 problems identified

- Formal specification was created –
  simple structure, still less ambiguous

- Conformance checking:
  2 additional problems found

- In total 6 critical issues hidden for testing
Summary of contributions

1. **B-I-Sat:** Bounded saturation-based model checking
   - new integrated algorithm
   - three strategies
   - termination conditions
   - detailed evaluation

2. **PLCverif:** User-friendly model checking for developers
   - intermediate model
   - safety PLC programs
   - reductions
   - implementation
   - evaluation

3. **PLCspecif:** Complete behaviour description for PLC program modules
   - syntax, semantics
   - invariant checking
   - code generation
   - conformance checking
Publication overview

- **Number of publications:** 29
  - International journal articles: 3
  - Hungarian journal articles (in English): 3
  - International conference/workshop papers: 17

- **Independent citations:** 32

<table>
<thead>
<tr>
<th></th>
<th>Journal articles</th>
<th>International conf./ws.</th>
<th>Local event &amp; tech rep.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thesis 1</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Thesis 2</td>
<td>2</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Thesis 3</td>
<td>–</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>
Application of formal methods for ICS in practice is...

useful,
needed,
possible.
Summary of contributions

1. **B-I-Sat**: Bounded saturation-based model checking
   - New integrated algorithm
   - Three strategies
   - Termination conditions
   - Detailed evaluation

2. **PLCverif**: User-friendly model checking for developers
   - Intermediate model
   - Safety PLC programs
   - Reductions
   - Implementation
   - Evaluation

3. **PLCspecif**: Complete behaviour description for PLC program modules
   - Syntax, semantics
   - Invariant checking
   - Code generation
   - Conformance checking
Reviewers’ questions

dr. Tim A.C. Willemse
Among the items listed for future work on page 35 is the proposal to devise an efficient implementation for the precise, three-valued termination condition. The three-valued termination condition may lead to earlier termination, but computing this three-valued termination condition itself seems to involve some form of model checking, and, by extension, some overhead.

Is there any reason to assume that the advantages of earlier termination outweigh the disadvantages of a more complex decision procedure?

What would be typical use cases (verification problems) that would benefit from a more precise termination condition?
May the advantages of three-valued termination conditions outweigh their overhead? When could the 3VT be helpful?

- Advantage of 3VT:
  Allows early termination of bounded model checking of non-satisfied EG, EU properties
  - Without 3VT the complete state space needs to be explored

- Yes, advantages may outweigh the overhead
  - Currently: only for very shallow requirements (additional CTL expression in each iteration)
  - Future work: more efficient, on-the-fly evaluation
The requirements checked on PLC code is, as explained on page 41, given through the use of patterns rather than temporal logic, and on page 59 it is stated that these requirements are mapped onto LTL or CTL.

Are the patterns restricted to safety properties or do they also allow for stating liveness properties?

Do all properties map to a subset of $\text{CTL} \cap \text{LTL}$ or do some map to CTL and others to LTL?
Are the patterns restricted to safety properties? Do all properties map to a subset of CTL $\cap$ LTL?

- **Liveness patterns** can also be used
- In our practice, liveness properties were not common:
  - Response required in fixed time window (e.g. within 1 cycle), or
  - Non-immediate requests can be reset

- Patterns are mapped to **CTL or LTL (or both)**
  - Some patterns can only be expressed either in CTL or in LTL
  - TLs supported by the underlying model checker influence the supported patterns
The temporal logic expressions encoding the conformance relation $\text{pconf}_{5}^{v,v',K}$ employ an integer variable that essentially keeps track of how often $v$ has been ‘true’ and $v'$ has been ‘true’. It does so by incrementing and decrementing this variable. This raises some questions concerning decidability.

Is the set of values that this variable can assume bounded, i.e. will these variables always take on a value in a predetermined interval $[n, m]$ of integers?

If so, can such bounds be extracted from the PLC code or the parameters of the $\text{conf}_{5}$ relation?

If not, what does that mean for the decidability of conformance relations such as $\text{conf}_{5}$?
The CTL representation of pconf5 uses an integer variable to count the pulse length differences. Is it **bounded**? What is the bound? Is it **decidable** if not bounded?

- The **CTL representation** is defined with **unbounded** variables
- Many **model checkers** (e.g. NuSMV) require **bounded** variables (finite state space)
- It is **difficult to imagine** a PLC program pair which could **infinitely diverge**, then converge again **with finite memory**, and be conformant for any input sequence

- **Upper bound could be introduced** in practice. If it is not respected, the conformance is violated.
Reviewers’ questions

dr. Péter Battyányi
How can we describe the speed up, or any other advantage of the present decision diagram based model checking, compared to existing bounded model checking methods based on SAT procedures?
How can we describe the **speed up of B-I-Sat** compared to the **SAT-based model checker** solutions?

- **Initial goal**: improve the speed of *saturation*
  - Bounded model checking was a **tool, not a goal**

- **Comparison** of **DD-based** and **SAT-based** methods is difficult
  - B-I-Sat supports **CTL**
  - SAT-based BMC typically support **LTL**
  - Comparison would be limited to **CTL ∩ LTL**
  - **Complexity** of CTL and LTL model checking is different
Could we define a canonical form of the intermediate model underlying PLCverif?

Do the existing simplifications lead to a reduced form which is small enough in most of the typical industrial verifications?
Is it possible to define a **canonical form of IM**? Do the existing simplifications lead to a reduced form which is **small enough**?

- Yes, but first a **weight/cost function** would be needed for **minimality**
  - E.g. reducing the number of locations increases the length of expressions

- **Canonical form of IM is not needed in practice**
  - Automaton simplifications are **powerful enough already**
  - To achieve minimality the simplifications may **take more time than the gain** in verification time
  - **Difficult to have good cost functions** for the minimisation which correlate with the verification time
When talking about the translation between STL and SCLr in the Appendix, one has the impression that STL does not have a fixed, standard semantics. Would it be reasonable to make proposals for one?
Would it be reasonable to propose a fixed, standard semantics for the STL language?

- **STL has a fixed semantics** (defined by its compiler), but the precise semantics is **not publicly disclosed**.
- It would be beneficial to **determine** the already defined semantics for future verification purposes.

### 1.11 O( Or with Nesting Open

**Description**

O( (OR nesting open) saves the RLO and OR bits and a function code into the nesting stack. A maximum of seven nesting stack entries are possible.

**Status word**

<table>
<thead>
<tr>
<th></th>
<th>BR</th>
<th>CC 1</th>
<th>CC 0</th>
<th>OV</th>
<th>OS</th>
<th>OR</th>
<th>STA</th>
<th>RLO</th>
<th>/FC</th>
</tr>
</thead>
<tbody>
<tr>
<td>writes:</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>