Exploring the possible uses of translating PLC code to C (and more)

Contains joint work with D. Darvas, E. Blanco
Motivation
PLC Program Simulation

- Simulate the behavior of a program without hardware

- Simulators are available, but…
  - Hard to define input sequences
  - They simulate whole programs, not individual functions

- Execution of PC programs (C, Java) is straightforward
  - Ultimate goal: integration to EcosimPro
Proposed Solution
Code Generation

- **Goal:** transform PLC code to other languages
  - **C** for simulation, unit testing, and verification
  - **Java** for unit testing
  - **Scilab** for visualization

- Easily extensible with new languages
Code Generation

- **Goal:** transform PLC code to other languages
  - C for simulation, unit testing, and verification
  - Java for unit testing
  - Scilab for visualization
- Easily extensible with new languages
Example: FB_HYST

- Hysteresis block with Low and High thresholds
  - Configurable delay for rising and falling edges

- Not as trivial as it looks like
  - See the discussion at UCPC-2689

```c
// Without timers
IF in > h THEN
    q := TRUE;
ELSIF in < l THEN
    q := FALSE;
END_IF;
```
C code generation

- **Goal**: generate semantically equivalent C code

```c
IF in > h THEN
  q := TRUE;
ELSIF in < l THEN
  q := FALSE;
END_IF;
```

```c
if (in > h) goto loc1;
else if (in < l)
  goto loc2;
loc1:
  q = true;
  goto loc3;
loc2:
  q = false;
loc3:
;```
C code generation

- **Goal:** generate semantically equivalent C code

```c
IF in > h THEN
  q := TRUE;
ELSIF in < l THEN
  q := FALSE;
END_IF;
```

- Compilable, runnable C program

```c
if (in > h) {
  q = true;
} else if (in < l) {
  q = false;
}
```
Simulation

- The program reads inputs from a CSV file
- Outputs are also written into CSV

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT8</td>
<td>FLOAT</td>
<td>FLOAT</td>
<td>FLOAT</td>
<td>INT64</td>
<td>INT64</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

- Output values of every cycle
- Each row represents an execution cycle
- Compare outputs with expected values → testing
Simulation

- Visualization for inputs and outputs

- **Automatic** generation based on the input CSV
Unit Testing

- **Unit Test:** Test for small, individual units of the source
  - E.g. functions
  - No communication

- **JUnit:** Unit testing framework for Java

```java
@Test
public void test() {
    Module module = new Module(); // Module is FB_HYST
    module.h = 3;
    module.l = 1;

    module.in = 5;
    module.run();
    Assert.assertTrue(module.q);
}
```
Unit Testing

- **Unit Test**: Test for small, individual units of the source
  - E.g. functions
  - No communication

- **JUnit**: Unit testing framework for Java

```java
@Test
public void test() {
    Module module = new Module(); // Module is FB_HYST
    module.h = 3;
    module.l = 1;

    module.in = 5;
    module.run();
    Assert.assertTrue(module.q);

    module.in = 2;
    module.run();
    Assert.assertTrue(module.q);
}
```

Test for another cycle
Formal Verification

- Verify that the program complies to some requirements
- Requirements within the code: **assertions**

```
IF IN > H THEN
  Q := TRUE;
ELSIF IN < L THEN
  Q := FALSE;
END_IF;
// #ASSERT IN < L --> not Q
```

- Formal methods are useful for assertion checking
- Established practice for C programs
  - Several tools are available (CBMC, ...)
  - They support “advanced features”, e.g. **pointers**
  - They can be **more suitable for software verification**
Verification

- Check the generated C code with CBMC
  - User-defined assertions
  - Common bug causes: overflows, type conversions, ...

```
// If IN < L, then Q shall be false.
//-- ASSERT (IN < L --> not Q)
```

$ cbmc --unwind 10 module.c
[...]
** 1 of 1 failed (1 iteration)
VERIFICATION FAILED

The assertion can fail if H < L and IN < L.

H = 1, L = 5, IN = 3
Verification

- Check the generated C code with CBMC
  - User-defined assertions
  - Common bug causes: overflows, type conversions, ...

```c
// If IN < L and H > L, then Q shall be false.
// #ASSERT ((IN < L and H > L) --> not Q)
```

$ cbmc --unwind 10 module.c

[...]

** 0 of 1 failed (1 iteration)
VERIFICATION SUCCESSFUL

No assertion violation is possible for 10 cycles
Automation with Jenkins

- Automatic testing on each commit
  - Input/output CSV comparison
  - CBMC verification
Automation with Jenkins

- Automatic testing on each commit
  - Input/output CSV comparison
  - CBMC verification
Visualization

- Automatic transformation to **Scilab**
- Visualize **inputs** and **outputs**
- PLC code changes $\rightarrow$ diagrams can be **redrawn easily**
Summary

- Workflow implemented

```
$ ./pv-codegen  -in fb_hyst.scl.cfa  -out outdir  -lang cbmc
```

- Wide feature set

- Automated testing after commits
Full Workflow

- SCL
- STL

IM → Structure Model

- Simulation (C)
- CBMC Verification (C)
- Unit Testing (C)
- Unit Testing (Java)
- Visualization (Scilab)