
CERN

Gyula Sallai

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

29/08/2017

Contains joint work with D. Darvas, E. Blanco

Motivation

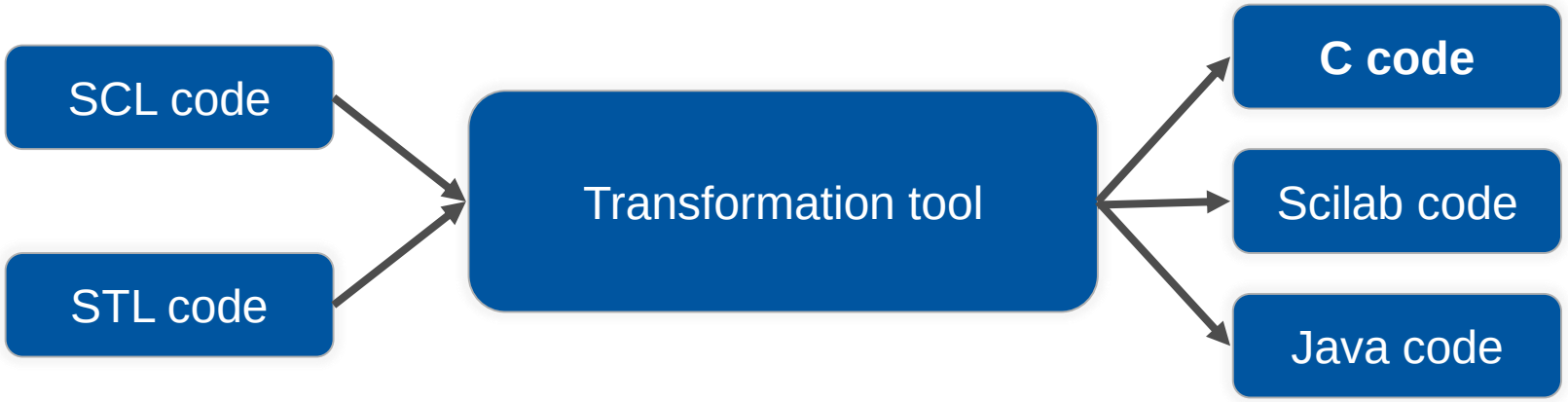
PLC Program Simulation

- 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

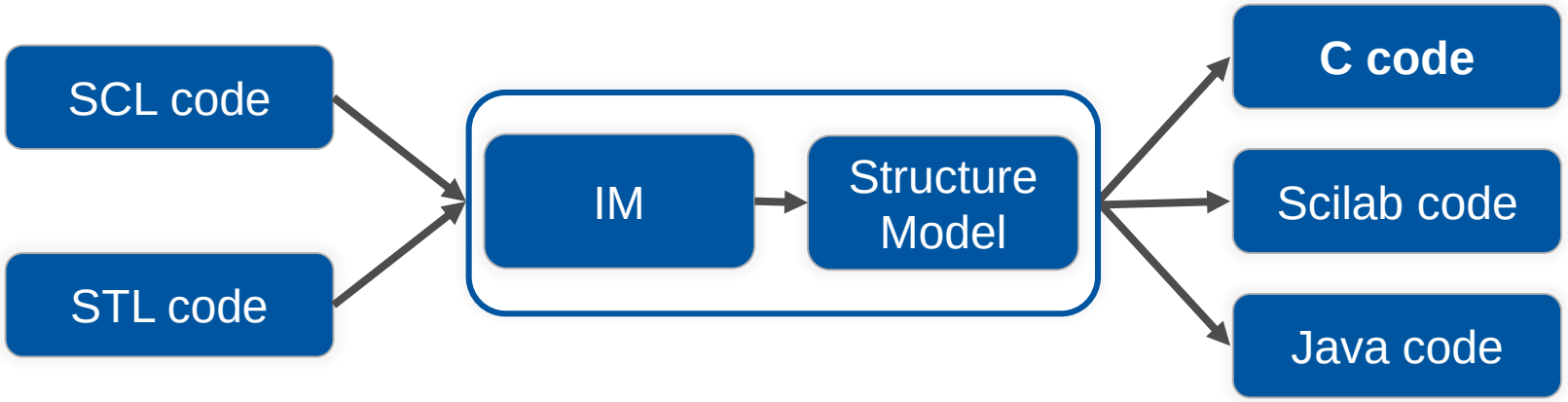
Code Generation

- **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 **L**ow and **H**igh thresholds
  - Configurable delay for rising and falling edges

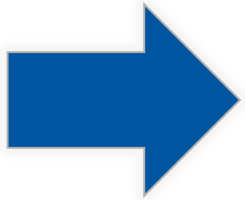
- Not as trivial as it looks like
  - See the discussion at *UCPC-2689*

```
// 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

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

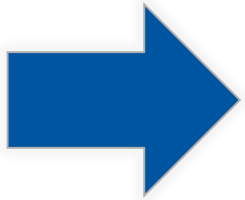


```
   if (in > h) goto loc1;
   else if (in < l)
        goto loc2;
loc1:
   q = true;
   goto loc3;
loc2:
   q = false;
loc3:
   ;
```

C code generation

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

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



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

- Compilable, runnable C program

Simulation

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

Output values of every cycle

t_cycle	instance3/in	instance3/h	instance3/l	instance3/onpt	instance3/offpt
UINT8	FLOAT	FLOAT	FLOAT	INT64	INT64
1	6	5	2	1	1
1	4	5	2	1	1
1	6	5	2	1	1
1	6	5	2	1	1
1	6	5	2	1	1
1	4	5	2	1	1
1	1	5	2	1	1
1	3	5	2	1	1
1	1	5	2	1	1
1	1	5	2	1	1
1	1	5	2	1	1
1	3	5	2	1	1

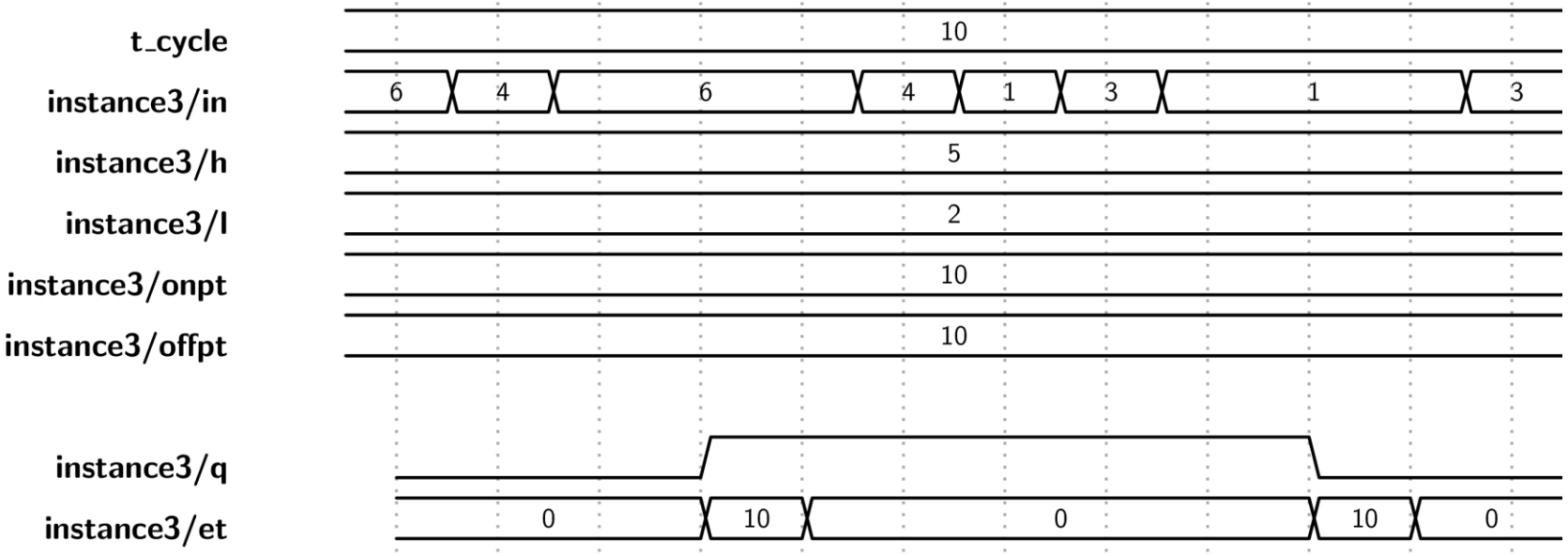
instance3/q	instance3/et
0	0
0	0
0	0
1	1
1	0
1	0
1	0
1	0
1	0
0	1
0	0
0	0

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

```
@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

```
@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
```

Special comment sequence for assert conditions

- 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, …

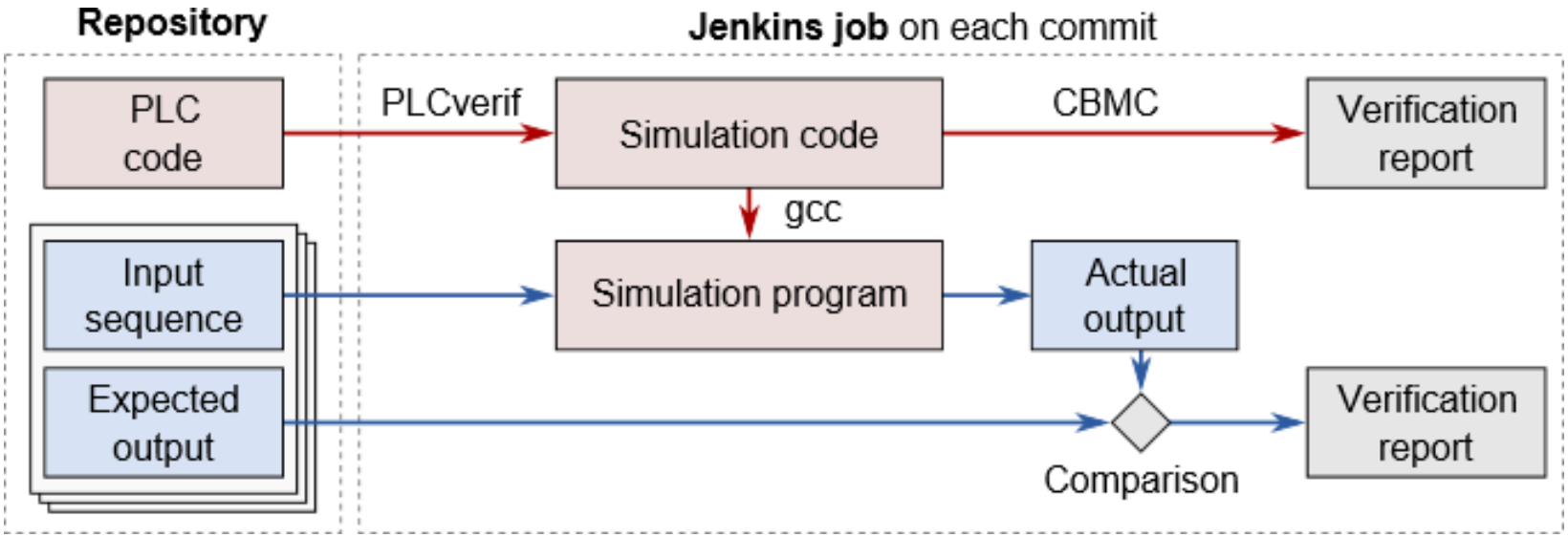
```
// If IN < L and H > L, then Q shall be false.
//#ASSERT ((IN < L and H > L) --> not Q)
```

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

No assertion violation is possible for 10 cycles

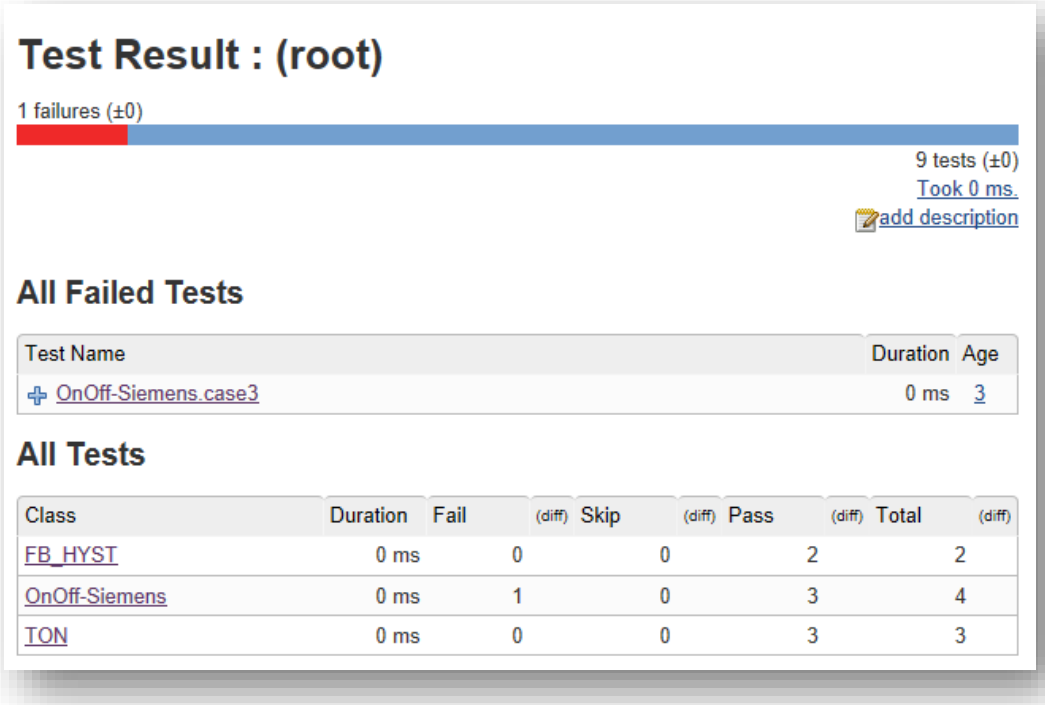
Automation with Jenkins

- 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



Test Result : (root)

1 failures (±0)

9 tests (±0)

Took 0 ms.

add description

All Failed Tests

Test Name	Duration	Age
OnOff-Siemens.case3	0 ms	3

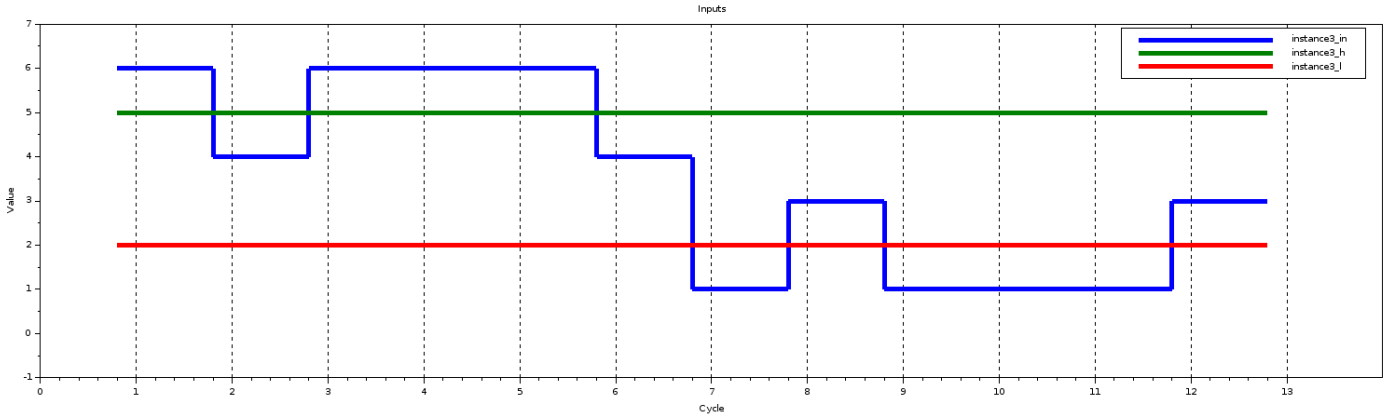
All Tests

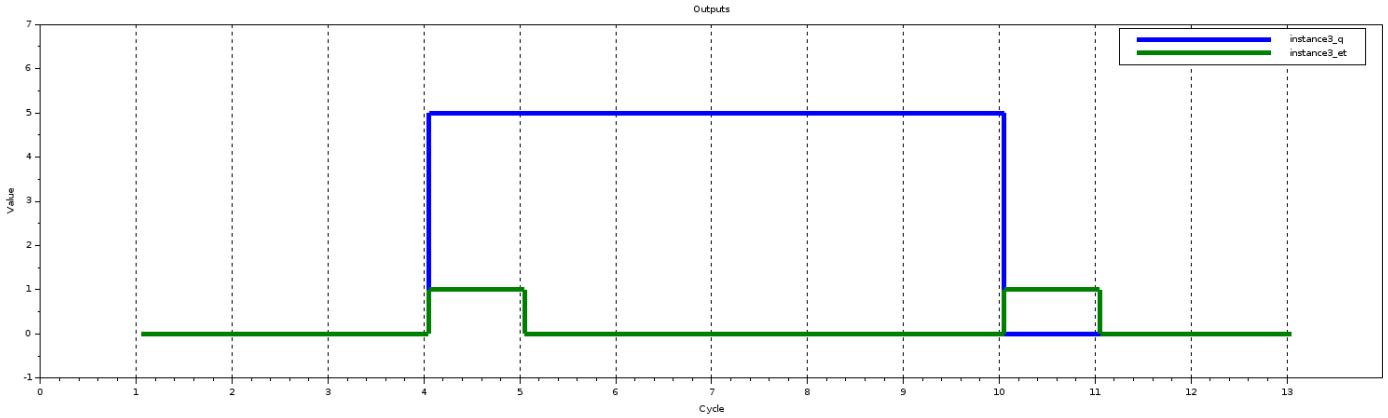
Class	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
FB_HYST	0 ms	0		0		2		2	
OnOff-Siemens	0 ms	1		0		3		4	
TON	0 ms	0		0		3		3	

instance/outonov
Exp: 0 / Real: 1
0
Exp: 1 / Real: 0
1
Exp: 1 / Real: 0
Exp: 1 / Real: 0
0
0
0
0
0
0
0
0

Visualization

- Automatic transformation to **Scilab**
- Visualize **inputs** and **outputs**
- PLC code changes → diagrams can be **redrawn easily**



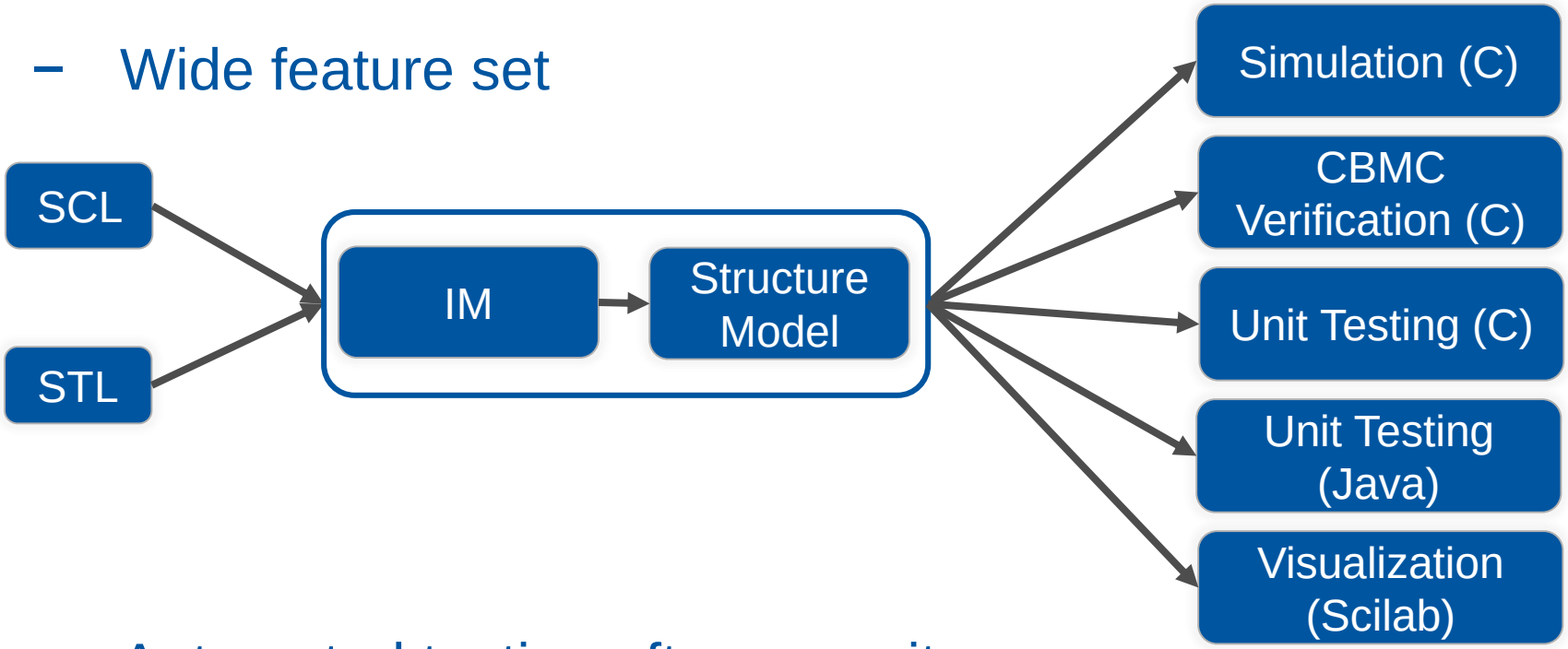


Summary

- Workflow implemented

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

- Wide feature set



- Automated testing after commits

www.cern.ch

Full Workflow