Testing PLC Software with Symbolic Execution

Sergey Grebenshchikov, TUM

Here: IEC 61131-3 ST

Testing PLC Software with Symbolic Execution

Sergey Grebenshchikov, TUM

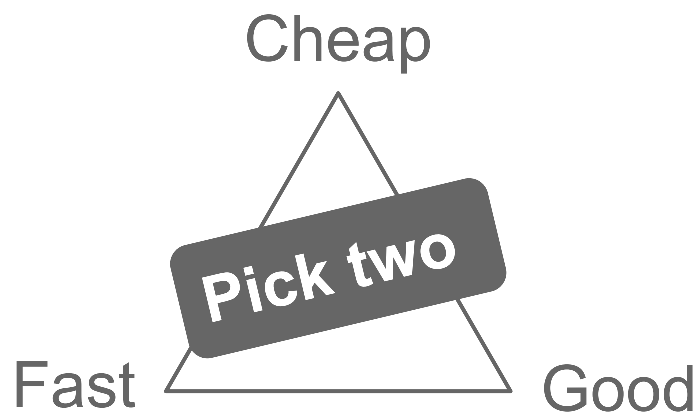
Motivation

Testing Software Is Easy.

Testing Software Well Is Hard.

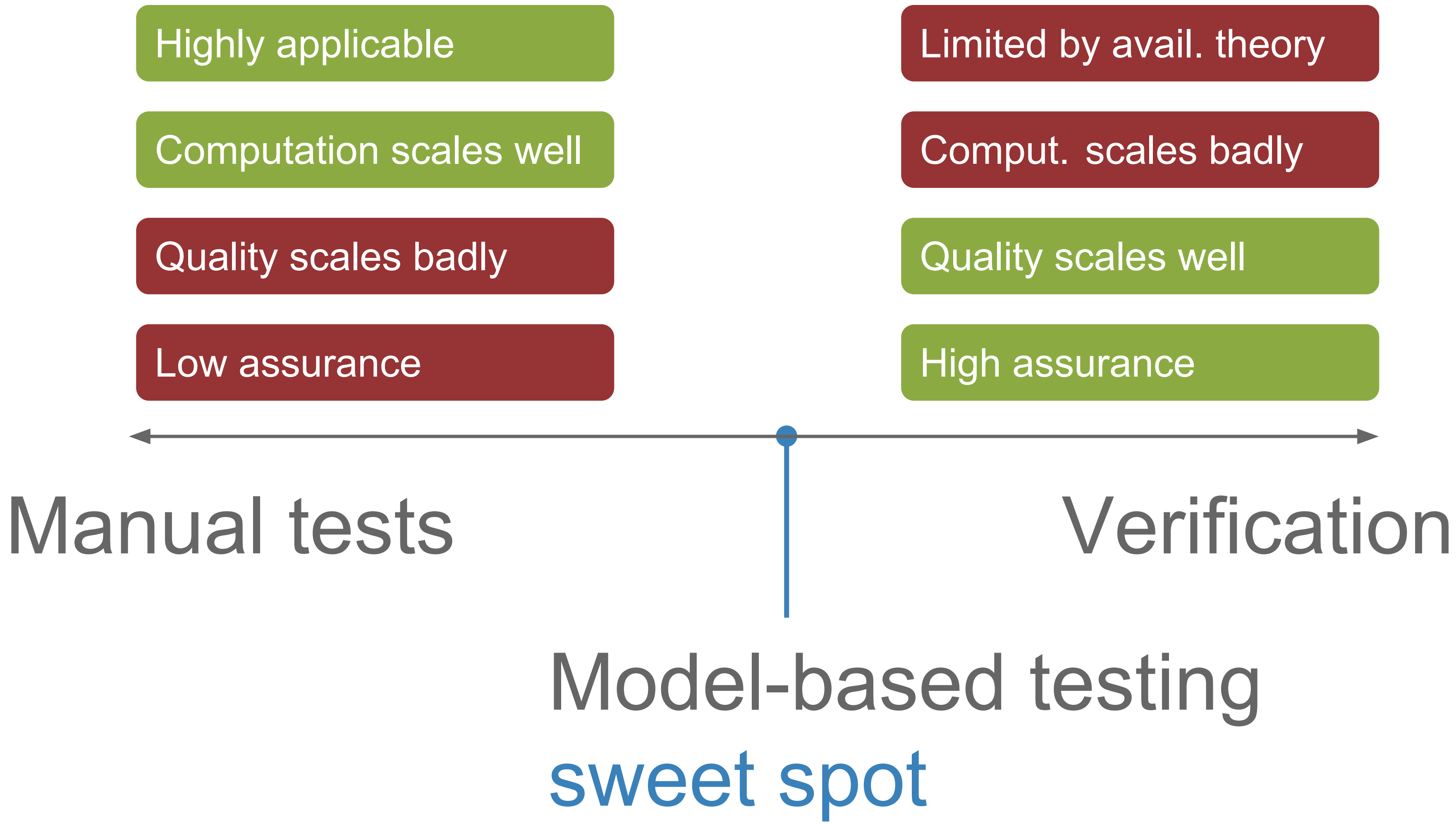
Real-Time

Testing Software Well Is Harder.


Cheap
Pick two
Fast
Good

Manual tests Verification

Manual tests	Verification
Highly applicable	Limited by avail. theory
Computation scales well	Comput. scales badly
Quality scales badly	Quality scales well
Low assurance	High assurance


Highly applicable
Computation scales well
Quality scales badly
Low assurance
Limited by avail. theory
Comput. scales badly
Quality scales well
High assurance
Manual tests
Verification
Model-based testing
sweet spot

What if there is no model?

Legacy code

Partial models

What if there is no model?

Supplement MBT with symbolic execution!

Symbolic execution recap.

```
f(int x, int y) {
    int z = 0;
    assume(x,y>=0)
    if(x>y) {
        z = x+y;
    } else {
        z = x-y;
    }
    assert(...)
}
```

x=0 y=0 z=0	...	x=2147483647 y=2147483647 z=-1	Explicit-state model checking
x=0 y=0 z=0	x=1 y=2 z=-1	x=3 y=2 z=1	Manual testing
x=*u* y=v z=u-v | $v \geq u$		x=*u* y=v z=u+v | $u > v$	Symbolic execution

Problem statement

Adapt SymEx methods to PLC software.
Focus today: Error handling.

Problem statement

Adapt SymEx methods to PLC software.
Focus today: Error handling.

There are challenges!

Challenges

```
FUNCTION_BLOCK Example
VAR
  Step1 : TON;
  Step2 : TON;
END_VAR


IF Run THEN
  Step1(IN := (
        PT := t
  IF (Step1.Q OR              2) THEN
    Error := TRUE;
  END_IF
  B := SensorB > 0.55;
  IF SensorA THEN
    Step2(IN := B;
          PT := t#50ms);
  END_IF
END_IF
END_FUNCTION_BLOCK
```

```
VAR_GLOBAL
  SensorA : BOOL;
  SensorB : REAL;
  B : BOOL;
  Run : BOOL
  Error : BOOL;
  Example : Example;
END_V

      ();
END_PROGRAM

// Specification:
// When in Run mode, SensorB
// must reach > 0.55 (B = TRUE)
// within at most 100ms.
// If this is not the case,
// an Error must be signaled
// within 50ms
```

```
FUNCTION_BLOCK Example
VAR
  Step1 : TON;
  Step2 : TON;
END_VAR


IF Run THEN
  Step1(IN := (NOT SensorA);
        PT := t#100ms);
  IF (Step1.Q OR Step2.Q) THEN
    Error := TRUE;
  END_IF
  B := SensorB > 0.55;
  IF SensorA THEN
    Step2(IN := B;
          PT := t#50ms);
  END_IF
END_IF
END_FUNCTION_BLOCK
```

```
VAR_GLOBAL
  SensorA : BOOL;
  SensorB : REAL;
  B: BOOL;
  Run : BOOL
  Error : BOOL;
  Example : Example;
END_VAR

BEGIN_PROGRAM
    RUN := TRUE;
    Ex();
END_PROGRAM

// Specification:
// When in Run mode, SensorB
// must reach > 0.55 (B = TRUE)
// within at most 100ms.
// If this is not the case,
// an Error must be signaled
// within 50ms
```

```
FUNCTION_BLOCK Example
VAR
  Step1 : TON;
  Step2 : TON;
END_VAR

IF Run THEN
  Step1(IN := (NOT SensorA);
        PT := t#100ms);
  IF (Step1.Q OR Step2.Q) THEN
    Error := TRUE;
  END_IF
  B := SensorB > 0.55;
  IF SensorA THEN
    Step2(IN := B;
          PT := t#50ms);
  END_IF
END_IF
END_FUNCTION_BLOCK
```

Time-dependent data and control flow

```
VAR_GLOBAL
  SensorA : BOOL;
  SensorB : REAL;
  B : BOOL;
  Run : BOOL
  Error : BOOL;
  Example : Example;
END_VAR

BEGIN_PROGRAM
    RUN := TRUE;
    Ex();
END_PROGRAM

// Specification:
// When in Run mode, Se
// must
// w                    ms.
// I   is is not the case,
// an Error must be signaled
// within 50ms
```

Sensor data

Informal specifications

Time dependency

Symbolic timestamps, Last-modified constraints

Temporal symbolic monitoring

Timed sequence diagrams (and MTL) over symbolic traces

Requirements and solution

By example

Specification | Analysis | Insight

R_1: Simple specification language that makes sense to domain experts.

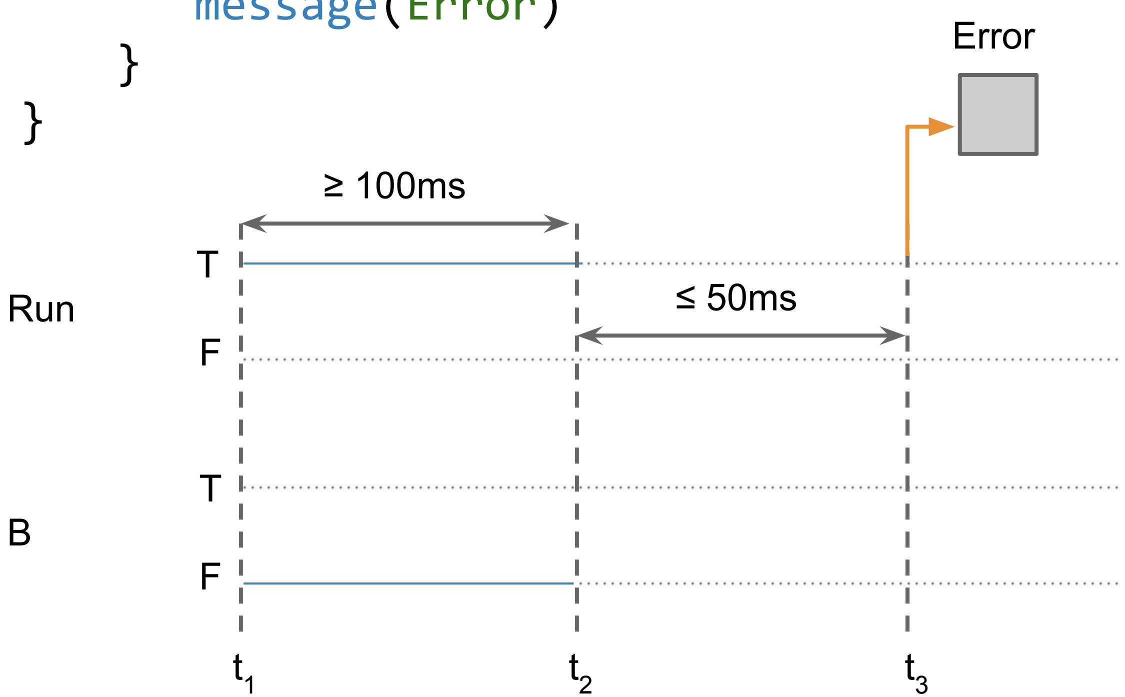
```
FUNCTION_BLOCK Example
VAR
  Step1 : TON;
  Step2 : TON;
END_VAR


IF Run THEN
  Step1(IN := (NOT SensorA);
        PT := t#100ms);
  IF (Step1.Q OR Step2.Q) THEN
    Error := TRUE;
  END_IF
  B := SensorB > 0.55;
  IF SensorA THEN
    Step2(IN := B;
          PT := t#50ms);
  END_IF
END_IF
END_FUNCTION_BLOCK
```

```
VAR_GLOBAL
  SensorA : BOOL;
  SensorB : REAL;
  B: BOOL;
  Run : BOOL
  Error : BOOL;
  Example : Example;
END_VAR

BEGIN_PROGRAM
    RUN := TRUE;
    Ex();
END_PROGRAM

// Specification:
// When in Run mode, SensorB
// must reach > 0.55 (B = TRUE)
// after at most 100ms.
// If this is not the case,
// an Error must be signaled
// within 50ms
```

```
time ::= int ms
timespec ::= ≥ time | ≤ time
tsd ::=   may { tsd }
       |  must { tsd }
       |  within(timespec) { tsd }
       |  delay(timespec) { tsd }
       |  constraint(constr) { tsd }
       |  message(global-var)
       |  tsd; tsd
       |  {}

constr ::= Boolean+LA formulae over program variables
global-var ::= Global variables
```

```
may {
    delay(>=0ms){
        constraint(Run & !B) {
          delay(>=100ms){}
        }
    }
}
must {
    within(<=50ms){
       message(Error)
    }
}
```

```
// When in Run mode, SensorB
// must reach > 0.55 (B = TRUE)
// after at most 100ms.
// If this is not the case,
// an Error must be signaled
// within 50ms
```



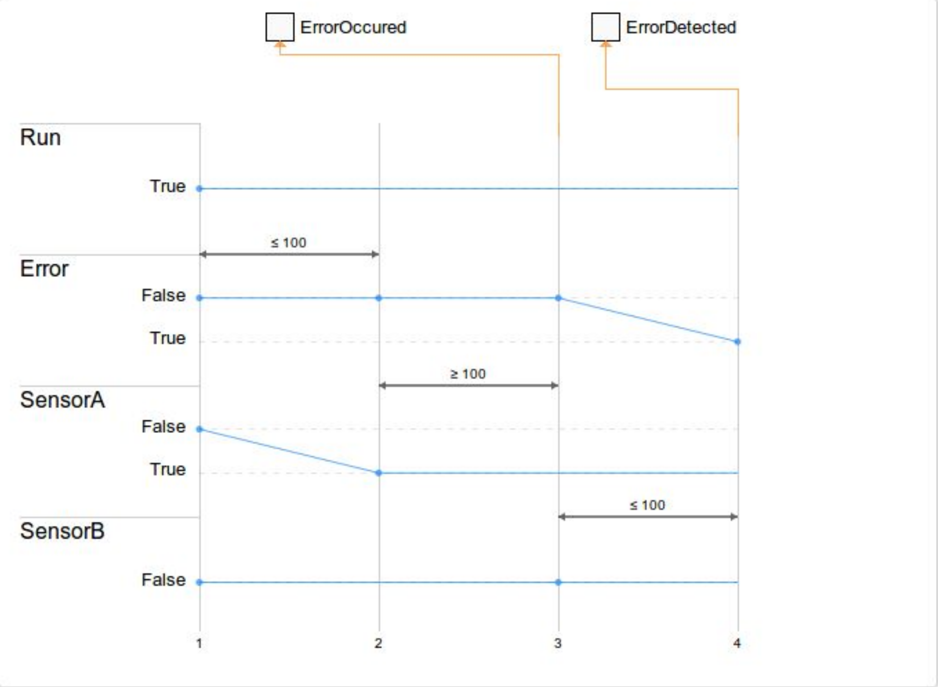
Scenario based specification

Scenario Builder | Coverage Analysis | Path Analysis

Scenario name | Save

- Scenario1 — Open | Delete
- F3 — Open | Delete
- F2 — Open | Delete
- F4 — Open | Delete
- Scenario1 (alt) — Open | Delete
- Default Example — Open | Delete
- F5 — Open | Delete
- F1 — Open | Delete
- F1.dev — Open | Delete
- F1.dev.1 — Open | Delete

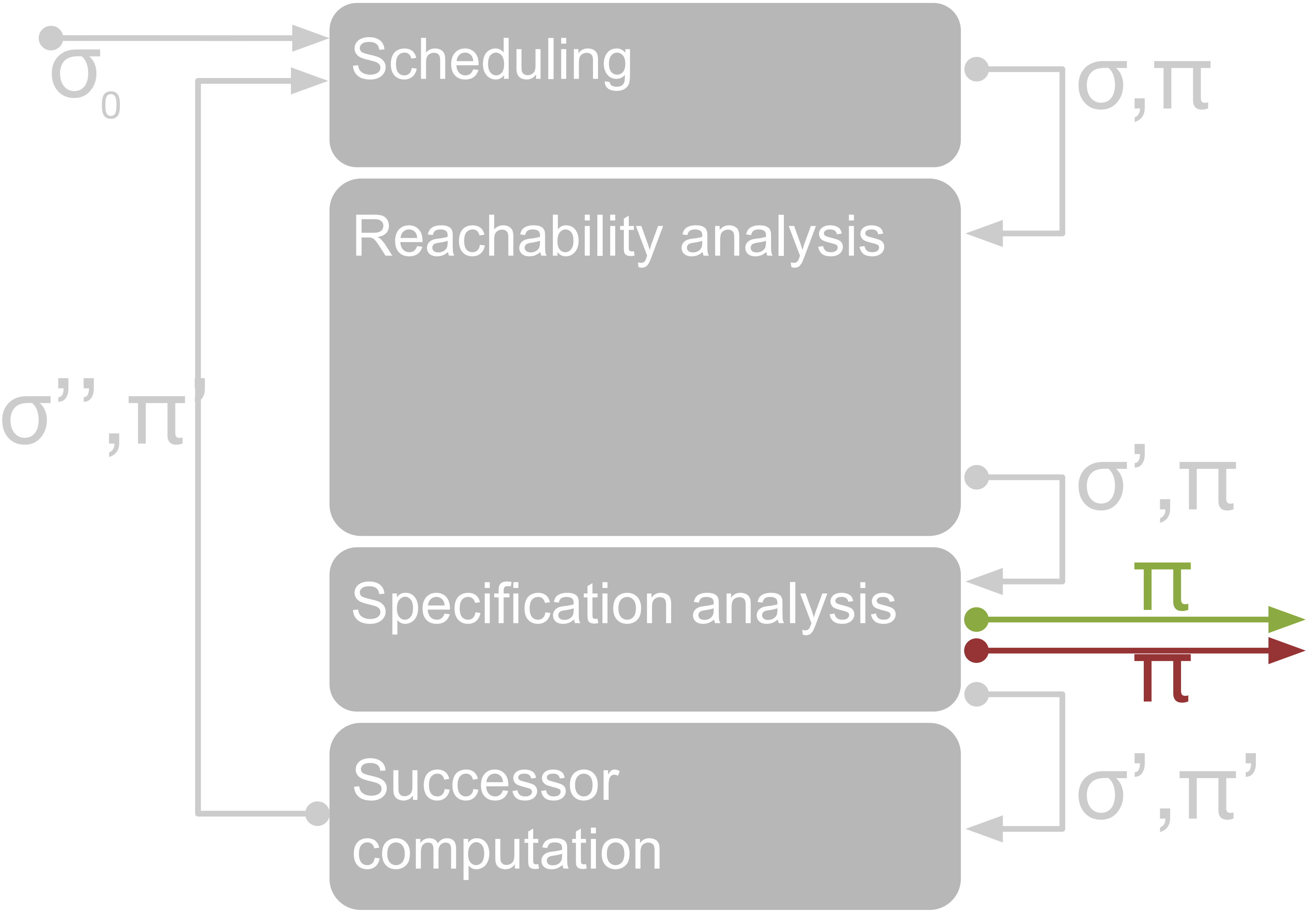
```
((constraint (Run True)
             (Error False)
             (SensorA False)
             (SensorB False))
  (later (<=100)
         (constraint (SensorA True)
                     (Error False)))
  (later (>=100)
         (constraint (SensorB False)
                     (Error False)))
  (send ErrorOccured)
  (later (<=100) (constraint (Error True))
           (send ErrorDetected))
 )

```



Your input parsed just fine :)

R_2: Automatic search for test cases that the software passes/fails w.r.t the given specification.

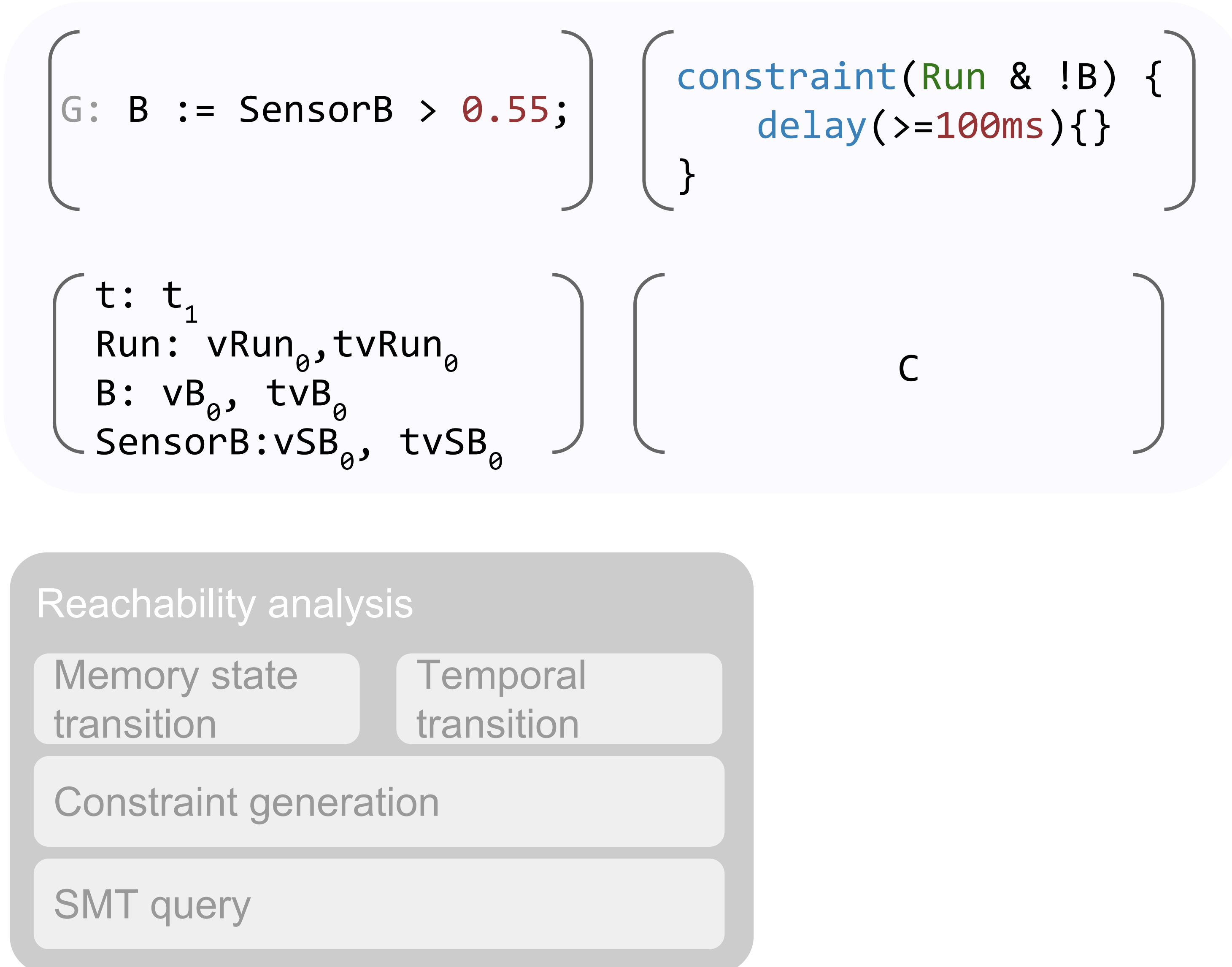
Specification → **Analysis** → Insight

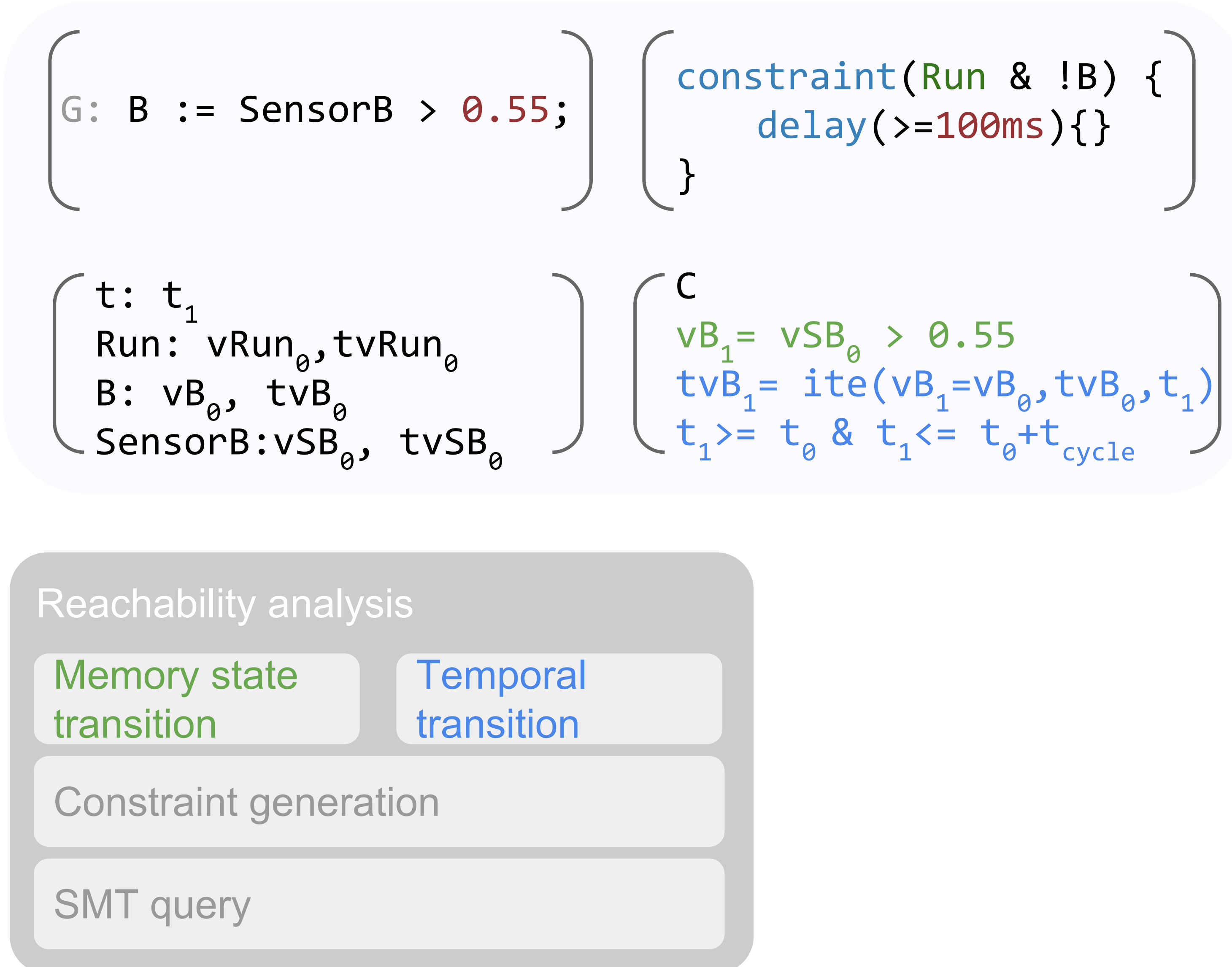


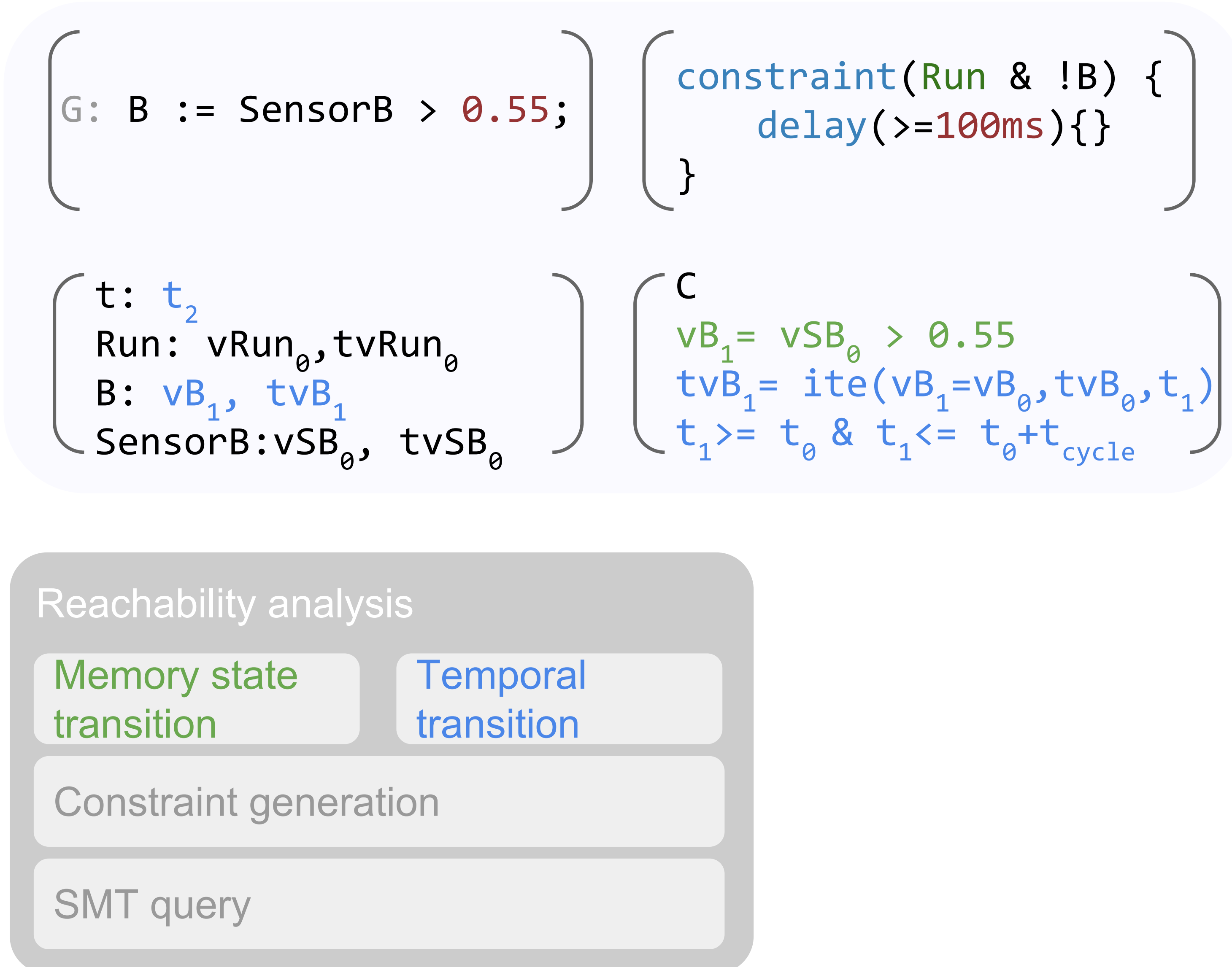
Scheduling
Reachability analysis
Specification analysis
Successor
computation
σ0
σ,π
σ',π
π
π
σ',π'
σ'',π'

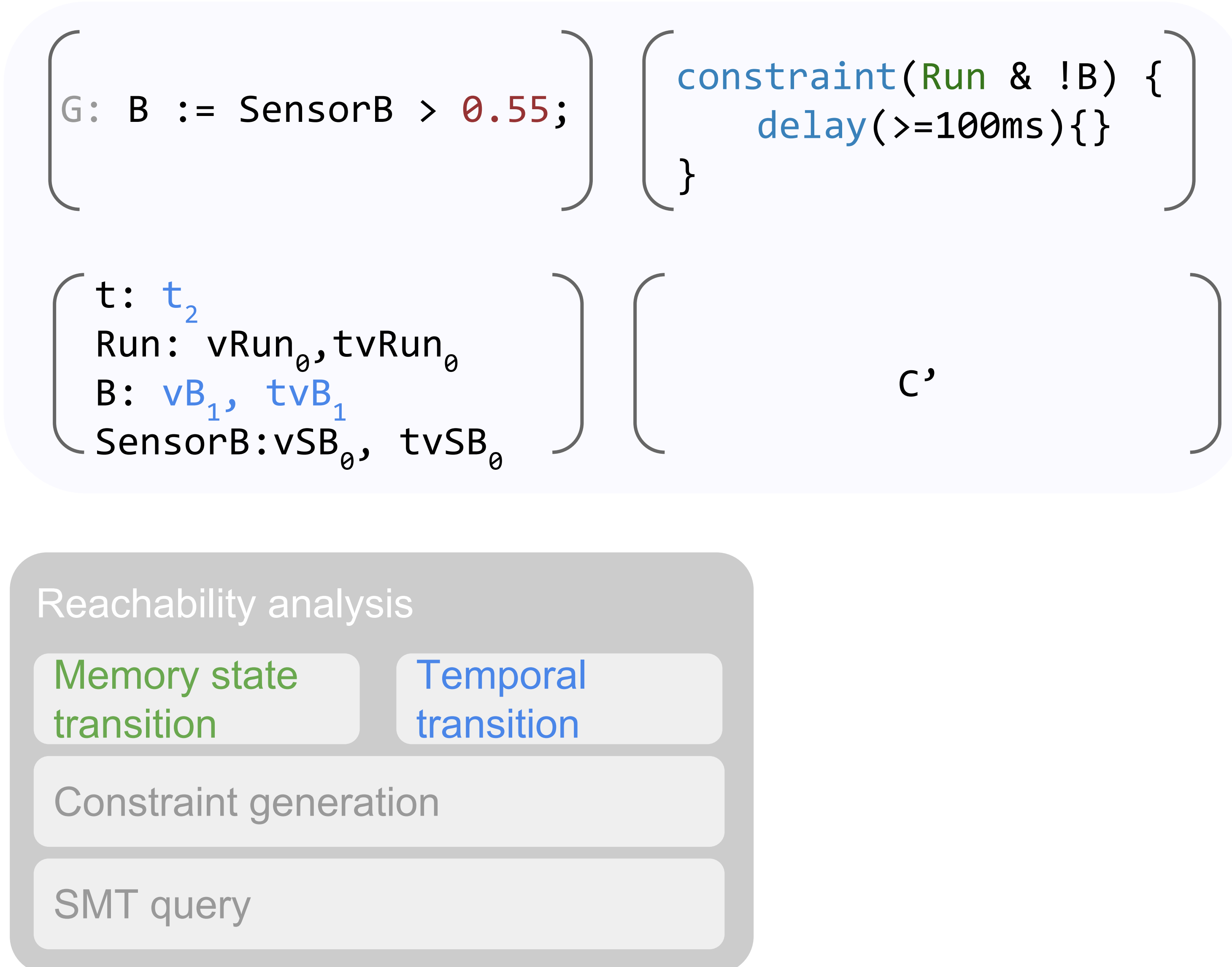
```
A: IF Run THEN
B:   Step1(IN := (NOT SensorA);
C:         PT := t#100ms);
D:   IF (Step1.Q OR Step2.Q) THEN
E:     Error := TRUE;
F:   END_IF
G:   B := SensorB > 0.55;
H:   IF SensorA THEN
I:     Step2(IN :=B;
J:           PT := t#50ms);
K:   END_IF
K: END_IF
```

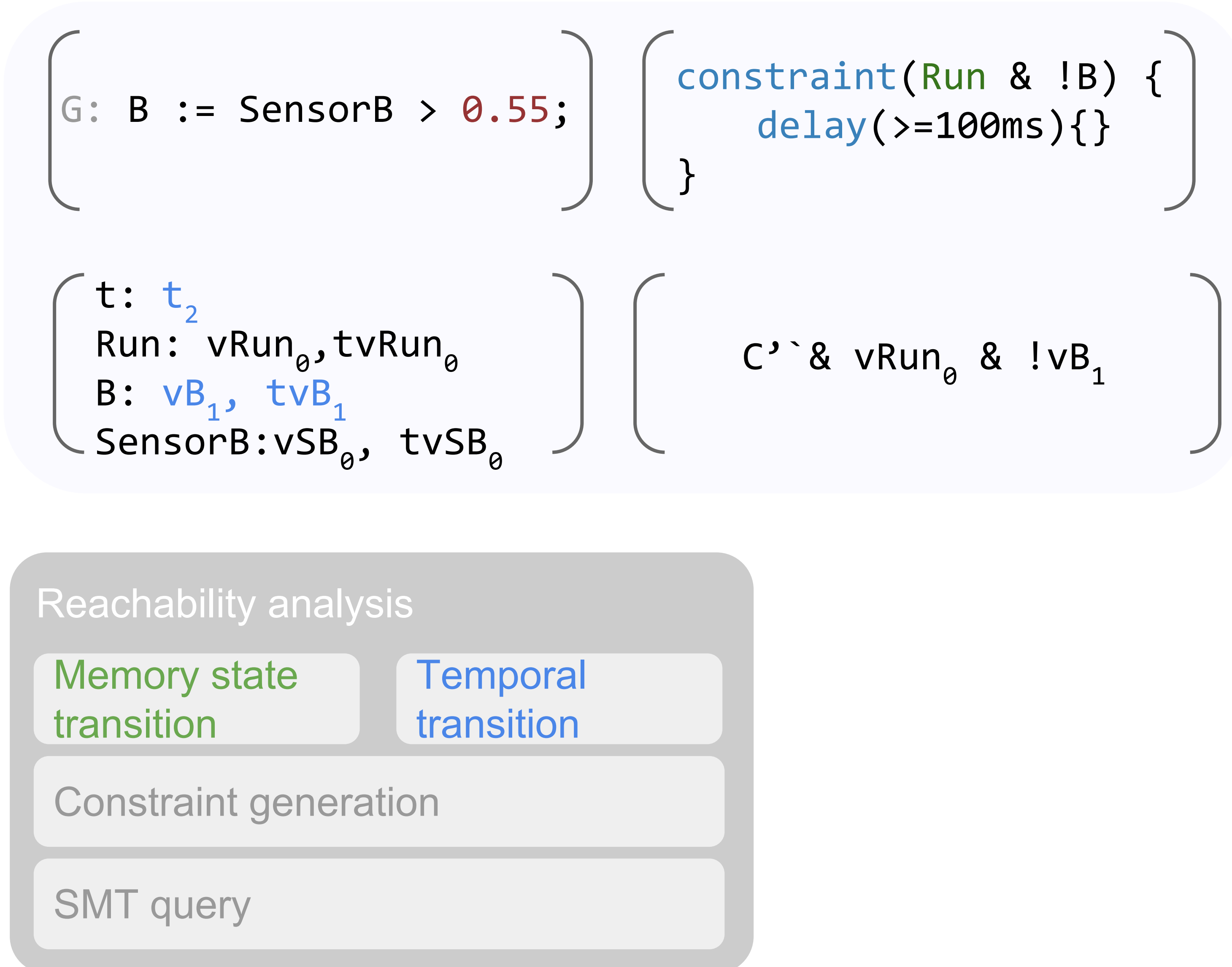
```
may {
    delay(>=0ms){
        constraint(Run & !B) {
          delay(>=100ms){}
        }
    }
}
must {
    within(<=50ms){
       message(Error)
    }
}
```

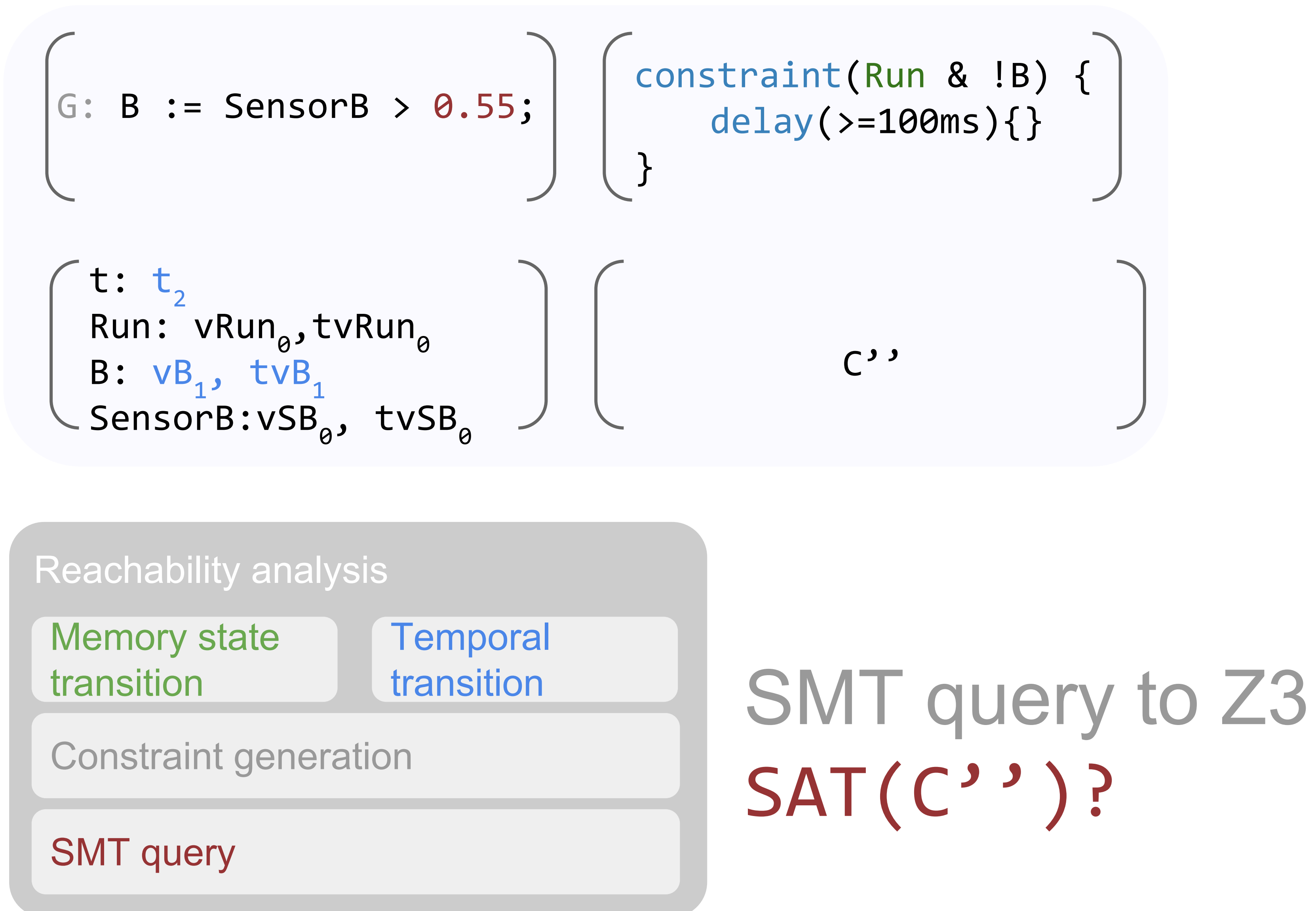

G: B := SensorB > 0.55;
constraint(Run & !B) {
delay(>=100ms){}
}
t: t1
Run: vRun0,tvRun0
B: vB0, tvB0
SensorB:vSB0, tvSB0
C
Reachability analysis
Memory state transition
Temporal transition
Constraint generation
SMT query


G: B := SensorB > 0.55;
constraint(Run & !B) {
delay(>=100ms){}
}
t: t_1
Run: $vRun_0$,$tvRun_0$
B: vB_0, tvB_0
SensorB:vSB_0, $tvSB_0$
C
vB_1= vSB_0 > 0.55
tvB_1= ite(vB_1=vB_0,tvB_0,t_1)
t_1>= t_0 & t_1<= t_0+t_{cycle}
Reachability analysis
Memory state transition
Temporal transition
Constraint generation
SMT query


G: B := SensorB > 0.55;
constraint(Run & !B) {
delay(>=100ms){}
}
t: t_2
Run: $vRun_0$,$tvRun_0$
B: vB_1, tvB_1
SensorB:vSB_0, $tvSB_0$
C
vB_1= vSB_0 > 0.55
tvB_1= ite(vB_1=vB_0,tvB_0,t_1)
t_1>= t_0 & t_1<= t_0+t_{cycle}
Reachability analysis
Memory state transition
Temporal transition
Constraint generation
SMT query


G: B := SensorB > 0.55;
constraint(Run & !B) {
delay(>=100ms){}
}
t: t2
Run: vRun0,tvRun0
B: vB1, tvB1
SensorB:vSB0, tvSB0
C’
Reachability analysis
Memory state transition
Temporal transition
Constraint generation
SMT query


G: B := SensorB > 0.55;
constraint(Run & !B) {
delay(>=100ms){}
}
t: t_2
Run: $vRun_0$,$tvRun_0$
B: vB_1, tvB_1
SensorB:vSB_0, $tvSB_0$
C’`& $vRun_0$ & !vB_1
Reachability analysis
Memory state transition
Temporal transition
Constraint generation
SMT query


G: B := SensorB > 0.55;
constraint(Run & !B) {
delay(>=100ms){}
}
t: t_2
Run: vRun_0,tvRun_0
B: vB_1, tvB_1
SensorB:vSB_0, tvSB_0
C''
Reachability analysis
Memory state transition
Temporal transition
Constraint generation
SMT query
SMT query to Z3:
SAT(C'')?

```
A: IF Run THEN
B:   Step1(IN := (NOT SensorA);
C:         PT := t#100ms);
D:   IF (Step1.Q OR Step2.Q) THEN
E:     Error := TRUE;
F:   END_IF
G:   B := SensorB > 0.55;
H:   IF SensorA THEN
I:     Step2(IN :=B;
J:           PT := t#50ms);
K:   END_IF
K: END_IF
```

```
may {
    delay(>=0ms){
        constraint(Run & !B) {
          delay(>=100ms){}
        }
    }
}
must {
    within(<=50ms){
       message(Error)
    }
}
```

```
H:  IF SensorA THEN ..
```

```
constraint(Run & !B) {
    delay(>=100ms){}
}
```

t: t_2
Run: $vRun_0$, $tvRun_0$
B: vB_1, tvB_1
SensorB: vSB_0, $tvSB_0$

C’’

Successor computation

Control flow automaton x Specification automaton

H: IF SensorA THEN ..

delay(>=100ms){}

t: t_2
Run: $vRun_0$, $tvRun_0$
B: vB_1, tvB_1
SensorB: vSB_0, $tvSB_0$

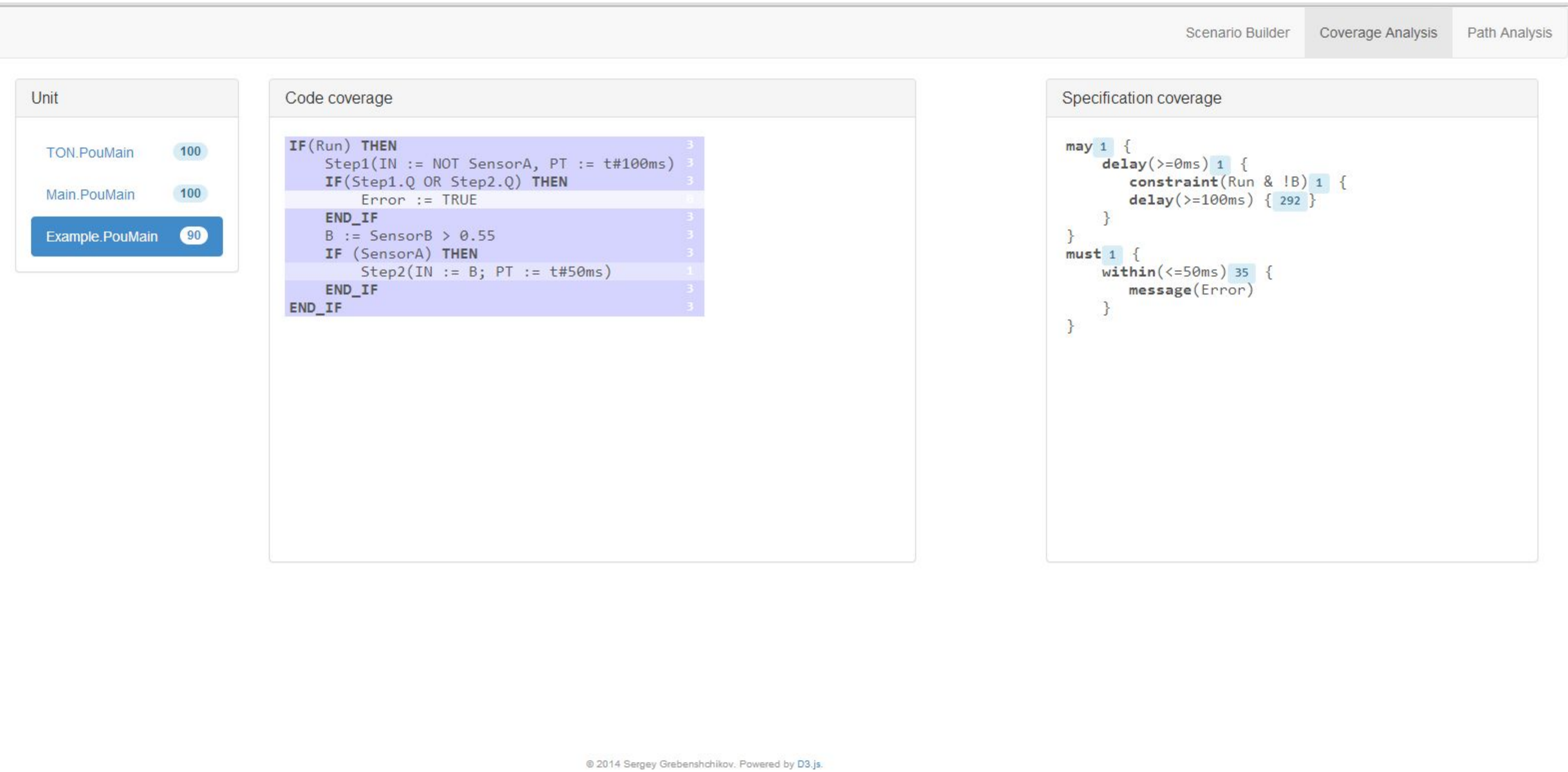
C’’

Successor computation

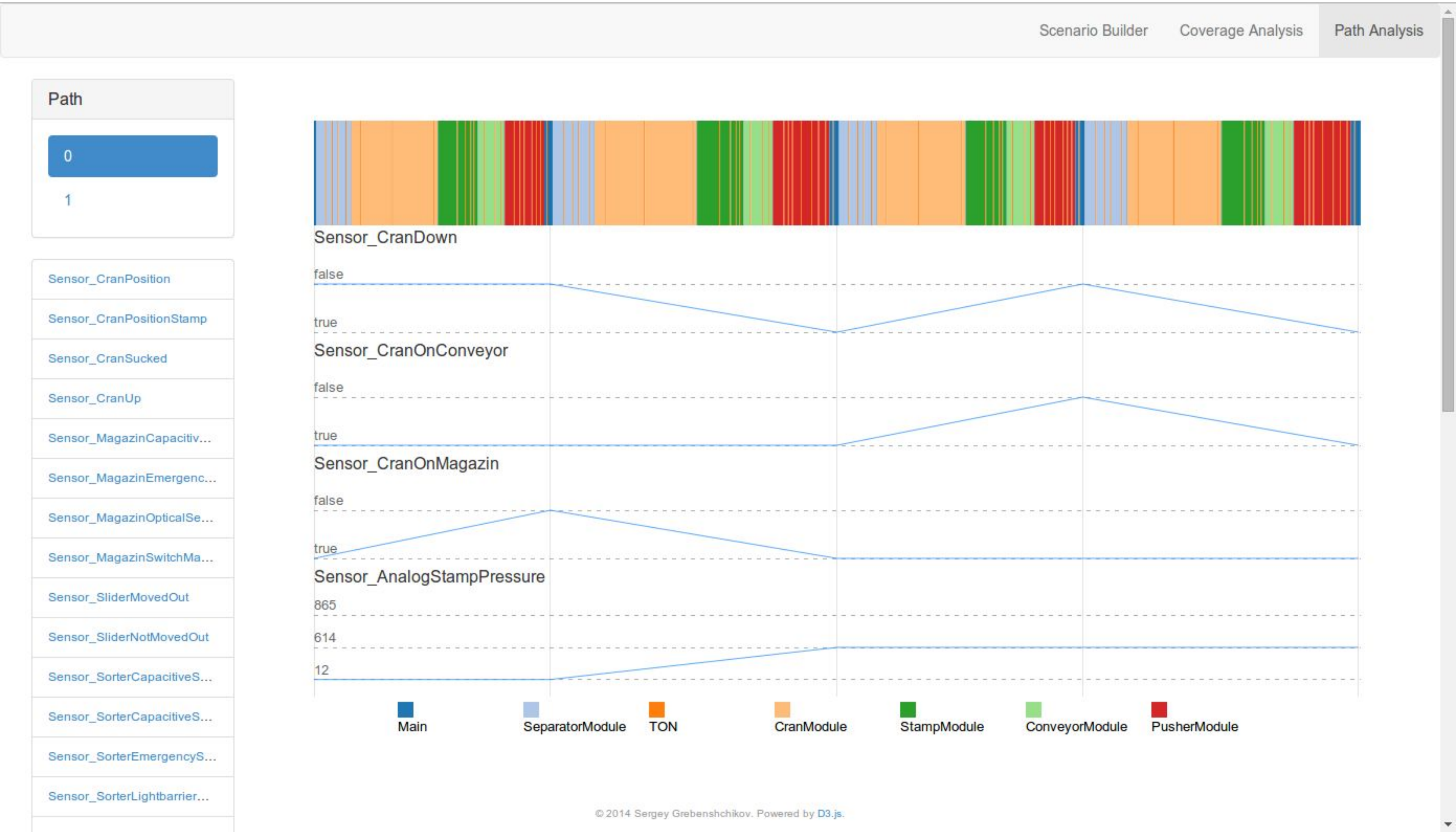
Control flow automaton x Specification automaton

R_3: Analysis results and test cases are **clearly presented** and their meaning **quantified**.

Coverage analysis



Path visualisation

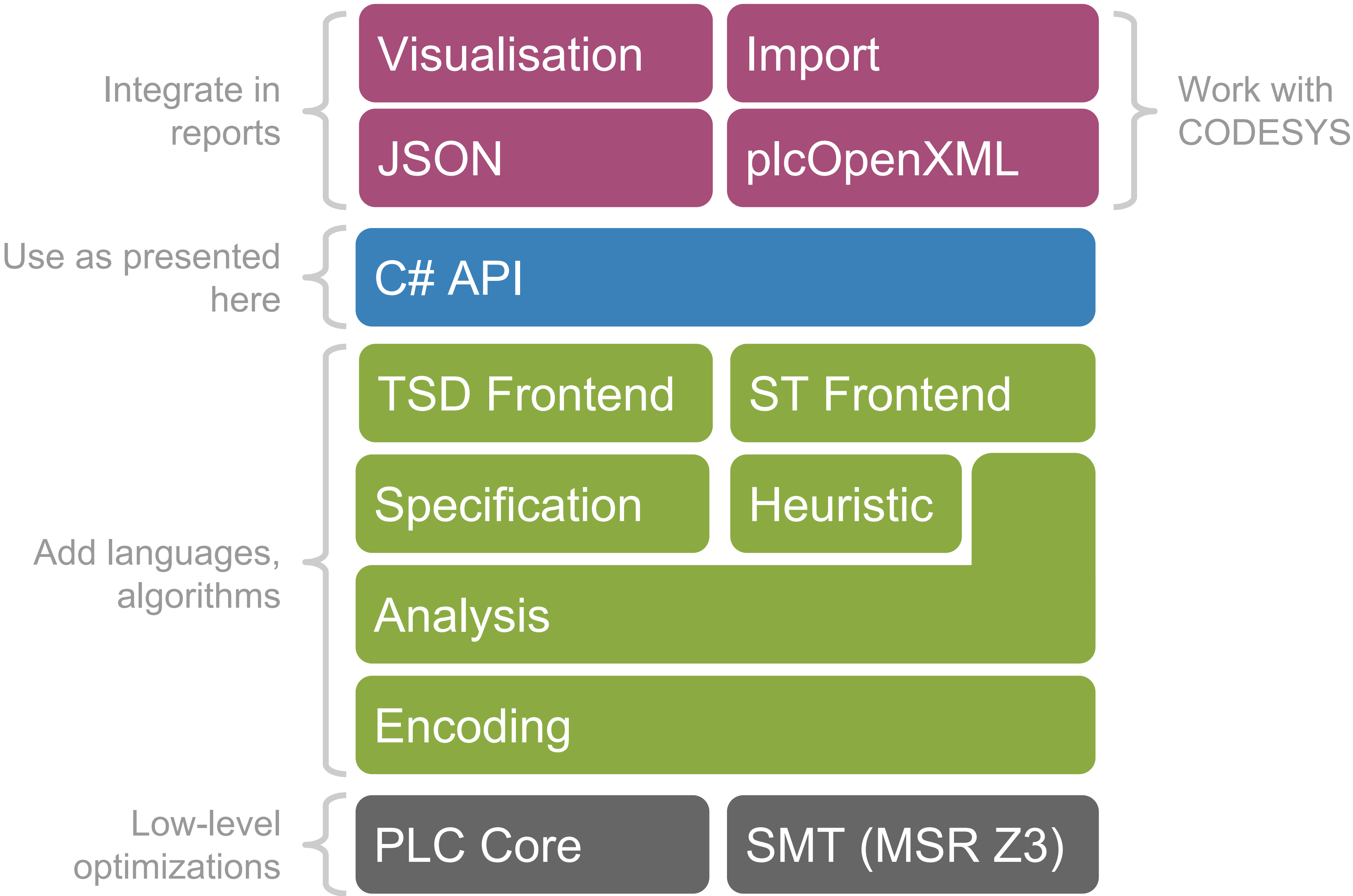


R_4: An extensible library that can be used in other tools and customized.

Specification

Analysis

Insight


Integrate in reports
Visualisation
Import
JSON
plcOpenXML
Work with CODESYS
Use as presented here
C# API
TSD Frontend
ST Frontend
Specification
Heuristic
Analysis
Encoding
Add languages, algorithms
Low-level optimizations
PLC Core
SMT (MSR Z3)

Summary

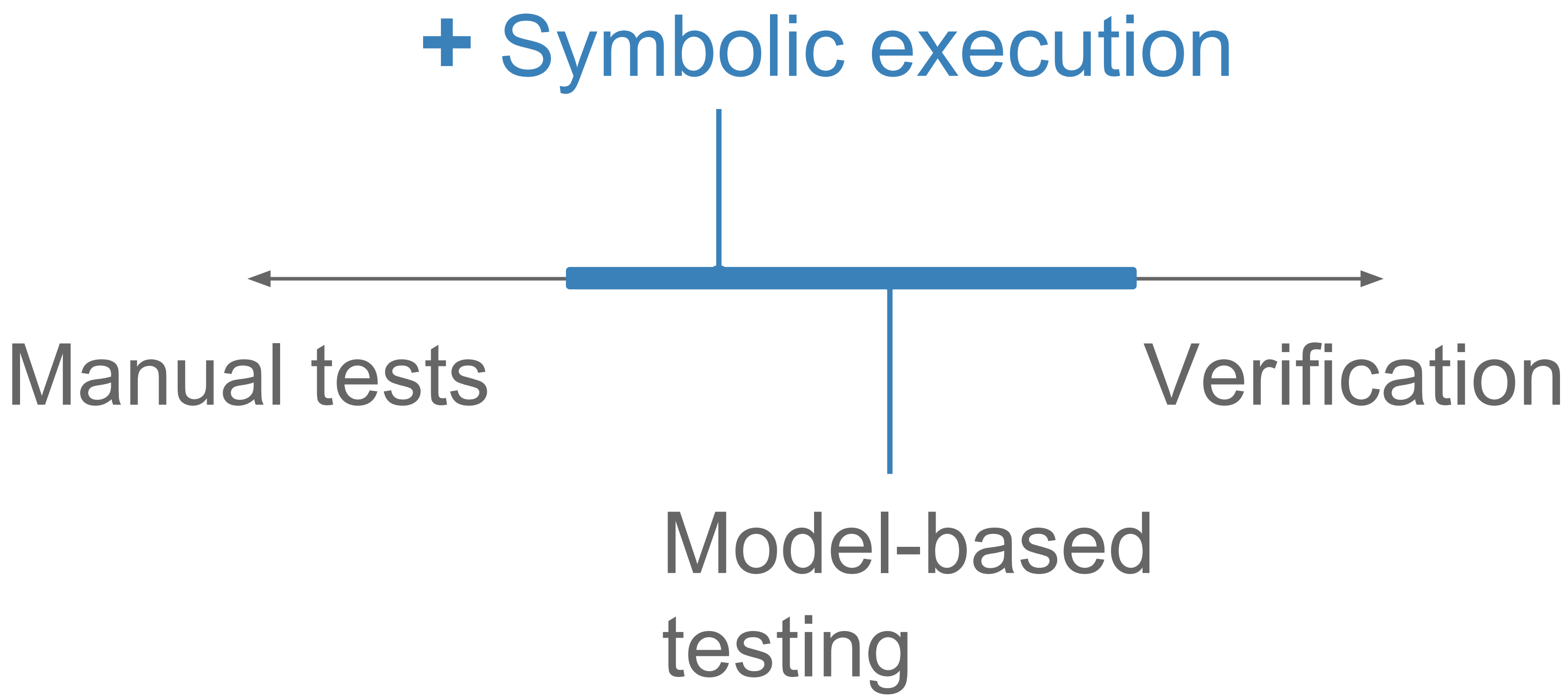
SymEx is a success story in desktop software, not yet adopted for PLC.

SymEx is a valuable companion technique to MBT.

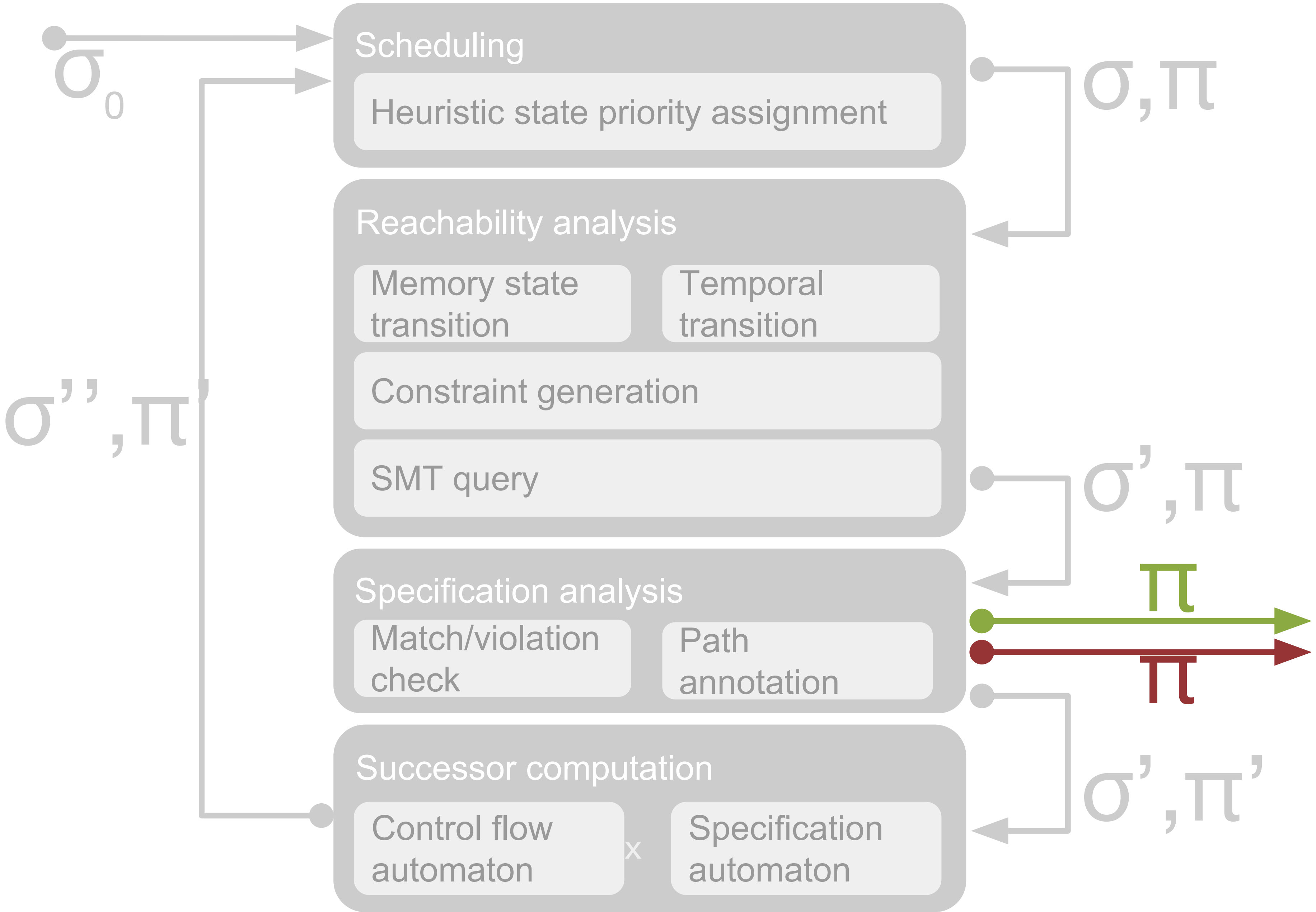
We adapted SymEx to the requirements of real-time PLC software.

We built the first ever SymEx engine for the IEC 61131-3.

You can use it as a library in your own tools today.


+ Symbolic execution
Manual tests
Verification
Model-based
testing

Backup Slides



σ_0
Scheduling
Heuristic state priority assignment
σ,π
Reachability analysis
Memory state transition
Temporal transition
Constraint generation
SMT query
σ'',π'
σ',π
Specification analysis
Match/violation check
Path annotation
π
π
Successor computation
Control flow automaton
x
Specification automaton
σ',π'