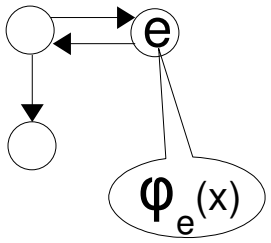
Verifying SystemC with Scenario

Nicolas Ayache
Loïc Correnson
Franck Védrine

Different techniques for System Verification

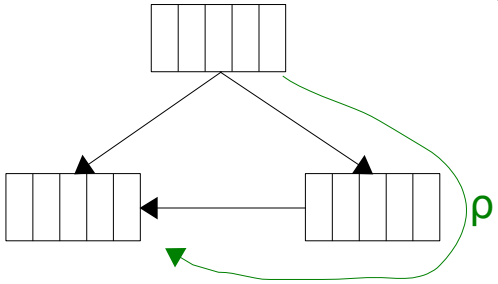
Abstract Interpretation:

1 process



Model Checking: N processes

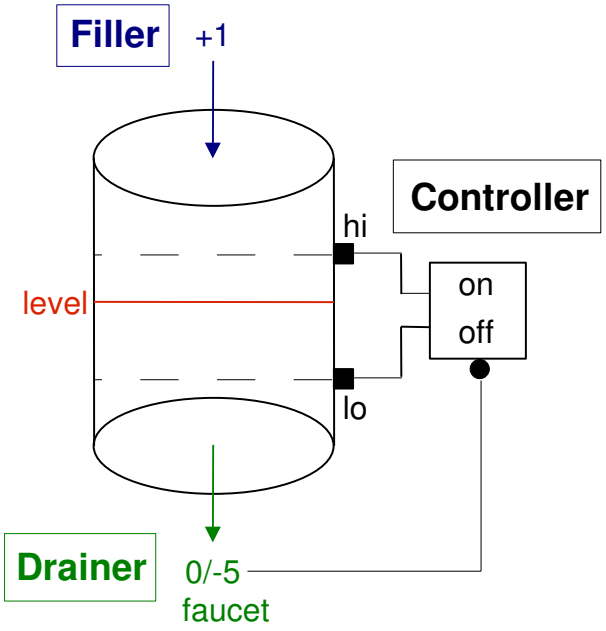
$\forall \rho, \varphi(\rho)$



$\varphi(\rho)$

Simulation: N processes

SystemC Example: Tank



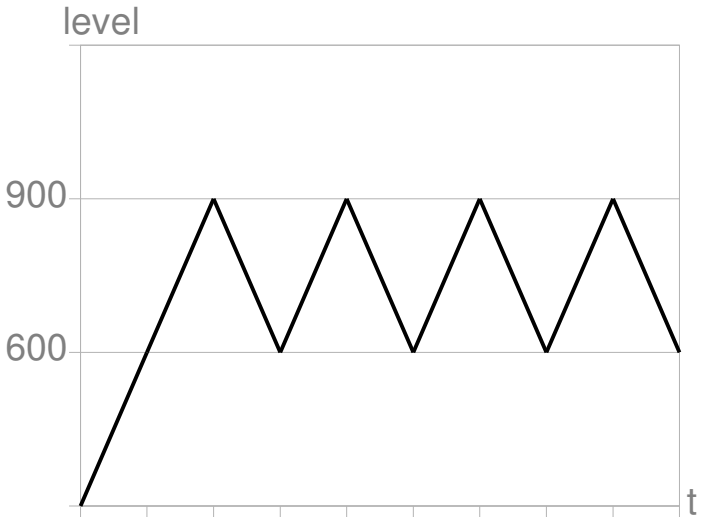
3 processes:
Filler, Drainer, Controller

4 shared variables:
level, hi, lo, faucet

Interesting variable : *level*

What is happening?

Eventually, Periodic Behavior



Goal: Eventually, level ($\pm\ \delta$) $\in$ [600 ; 900]

Our Approach for Verification

« Abstract Interpretation for N processes »

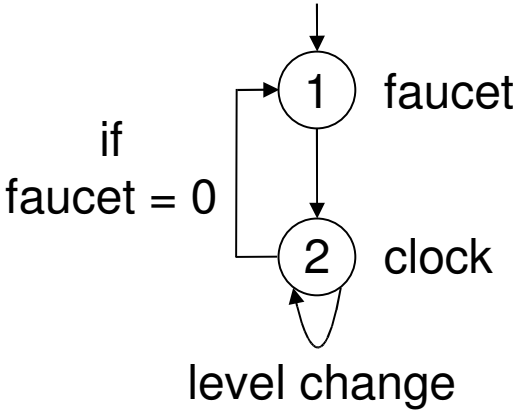
« Scenario: Abstract Interpretation and Model Checking cooperating »

« Scenario: the Engineering point of view »

Abstract Interpretation of N processes

Processes:

translated into LTS of waiting points



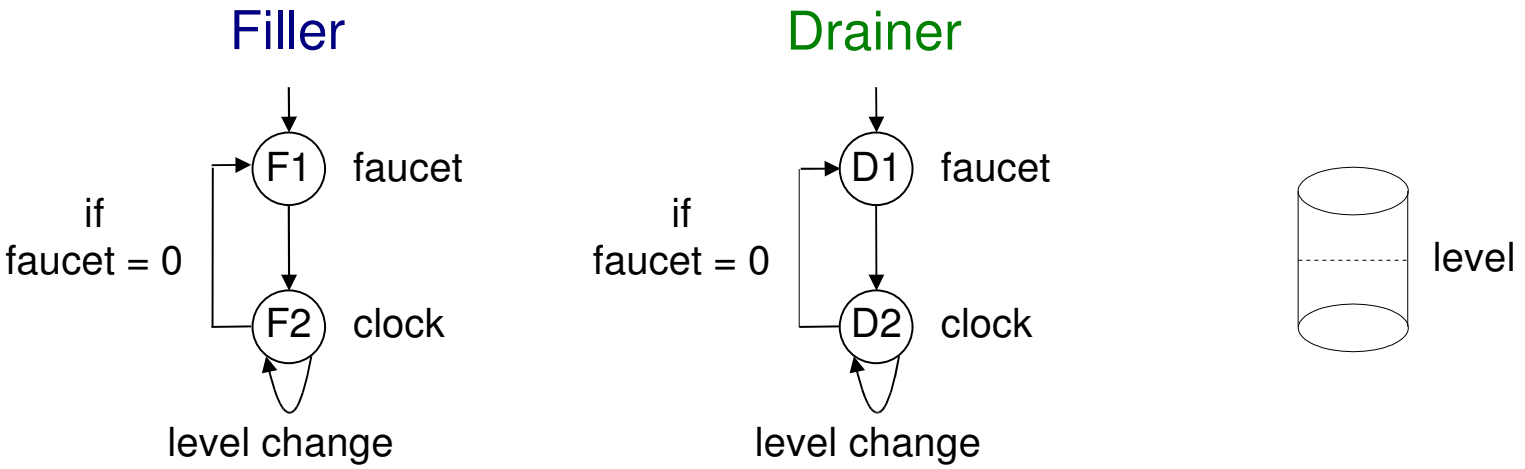
D1/F1: **Drainer**/**Filler** waits faucet
D2/F2: **Drainer**/**Filler** waits clock

System States:

variable values for each waiting points tuple

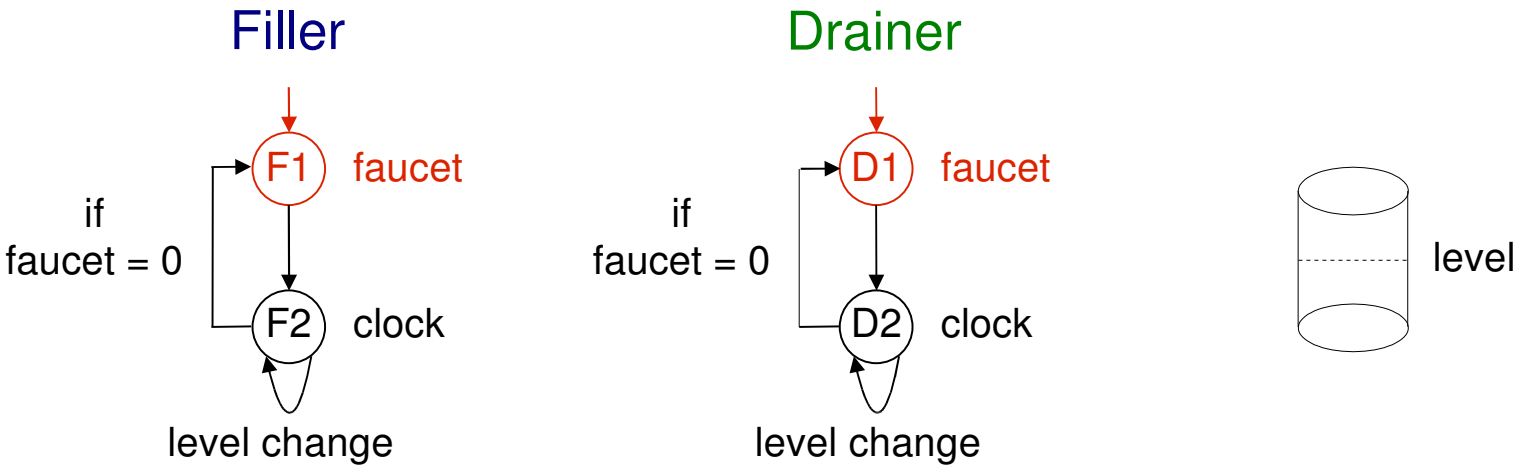
config	name	level
F1 D1	init	min? .. max?
F2 D1	fill	min? .. max?
F1 D2	no filler	min? .. max?
F2 D2	drain	min? .. max?

Abstract Interpretation of N processes



config	name	level
F1 D1	init	?
F2 D1	fill	?
F1 D2	no filler	?
F2 D2	drain	?

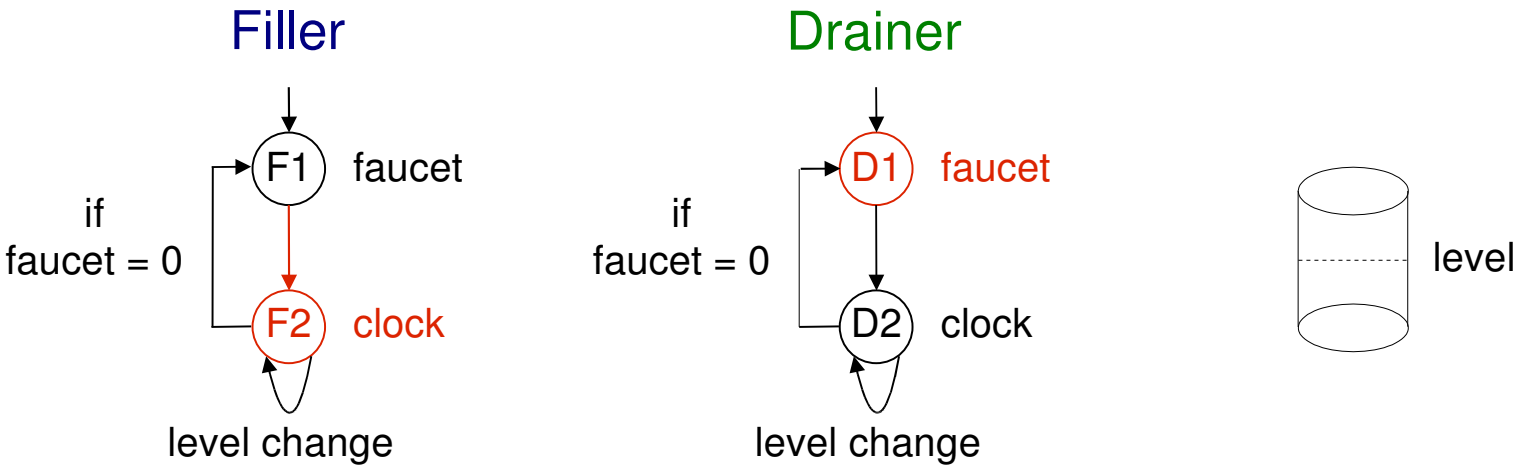
Abstract Interpretation of N processes



config	name	level
F1 D1	init	0 .. 0
F2 D1	fill	?
F1 D2	no filler	?
F2 D2	drain	?

F1 D1	0 .. 0

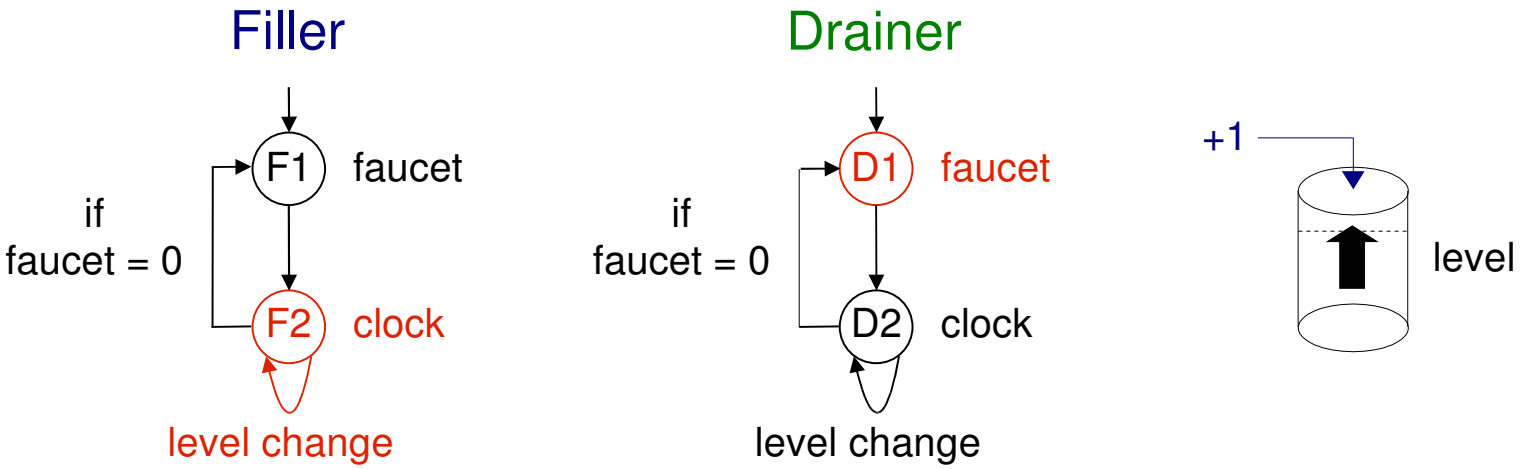
Abstract Interpretation of N processes



config	name	level
F1 D1	init	0 .. 0
F2 D1	fill	?
F1 D2	no filler	?
F2 D2	drain	?

F2 D1	0 .. 0

Abstract Interpretation of N processes

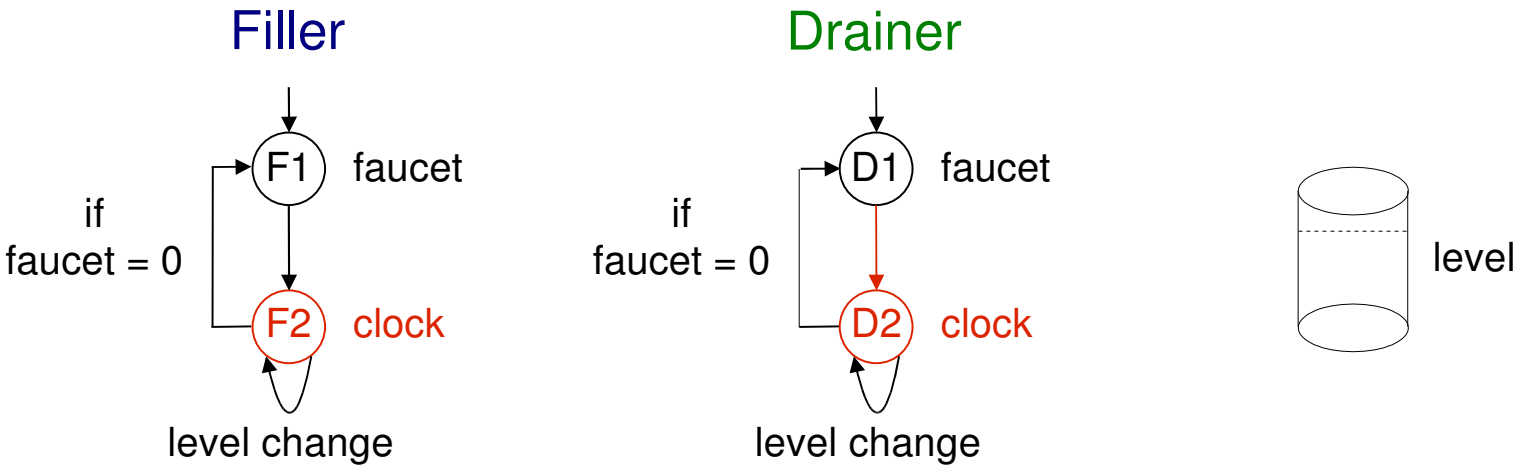


config	name	level
F1 D1	init	0 .. 0
F2 D1	fill	0 .. 902
F1 D2	no filler	?
F2 D2	drain	?

F2 D1
0 .. 0
0 .. 1
...
0 .. 902

widening
narrowing

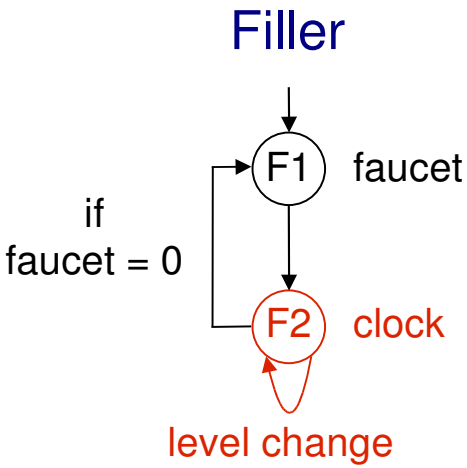
Abstract Interpretation of N processes

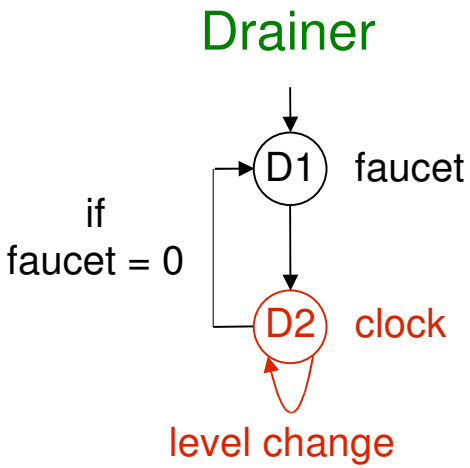


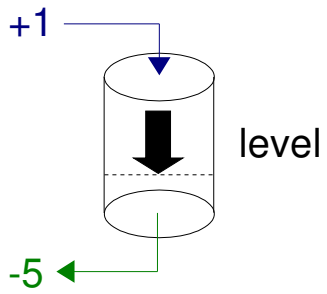
config	name	level
F1 D1	init	0 .. 0
F2 D1	fill	0 .. 902
F1 D2	no filler	?
F2 D2	drain	?

F2 D2	901 .. 902

Abstract Interpretation of N processes





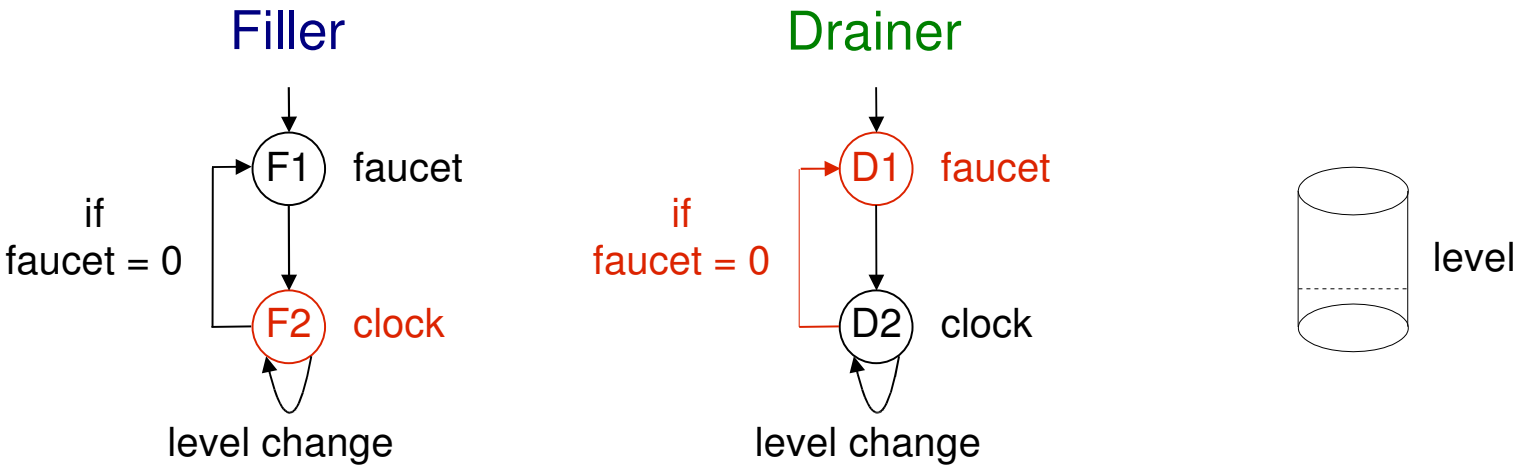


config	name	level
F1 D1	init	0 .. 0
F2 D1	fill	0 .. 902
F1 D2	no filler	?
F2 D2	drain	593 .. 902

F2 D2	901 .. 902
	897 .. 902
	...
	593 .. 902

widening
narrowing

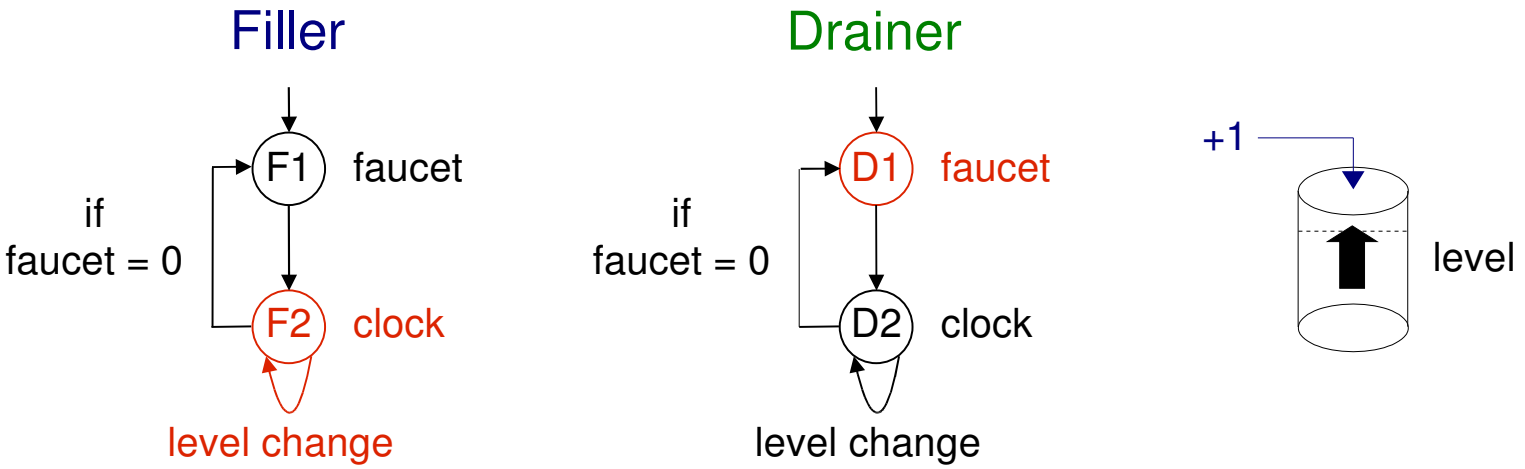
Abstract Interpretation of N processes



config	name	level
F1 D1	init	0 .. 0
F2 D1	fill	0 .. 902
F1 D2	no filler	?
F2 D2	drain	593 .. 902

F2 D1	593 .. 599

Abstract Interpretation of N processes

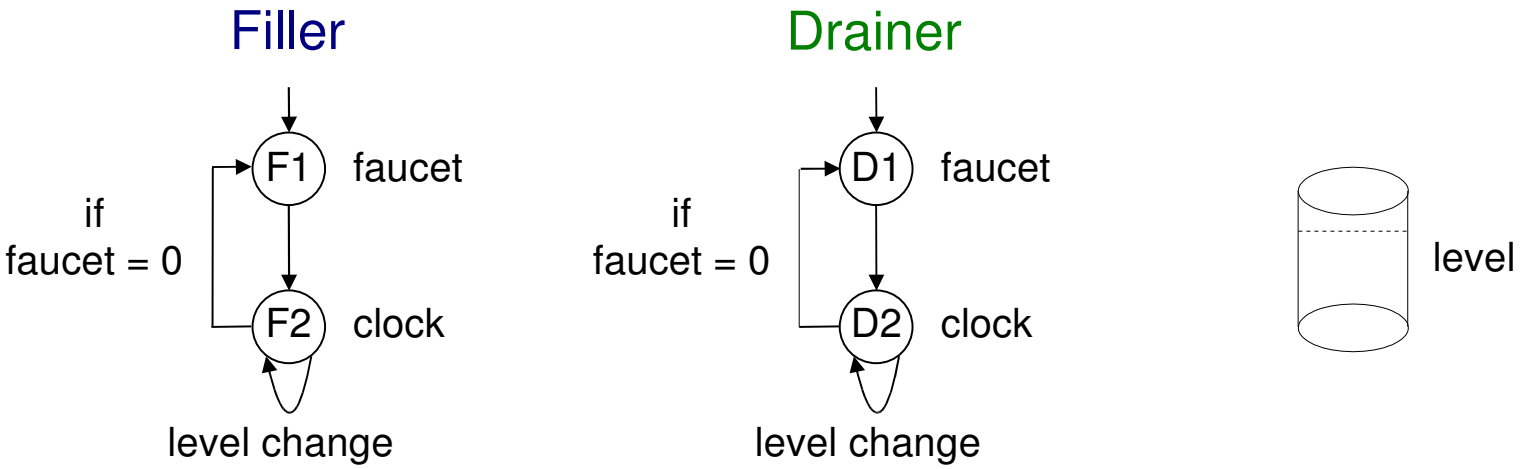


config	name	level
F1 D1	init	0 .. 0
F2 D1	fill	0 .. 902
F1 D2	no filler	?
F2 D2	drain	593 .. 902

⊇

F2 D1	593 .. 599

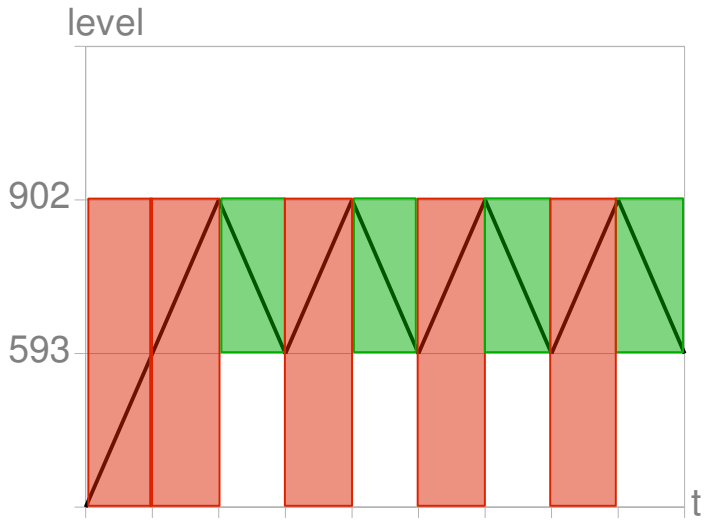
Abstract Interpretation of N processes



config	name	level
F1 D1	init	0 .. 0
F2 D1	fill	0 .. 902
F1 D2	no filler	?
F2 D2	drain	593 .. 902

Fixpoint!

Abstract Interpretation: Results



2 phases

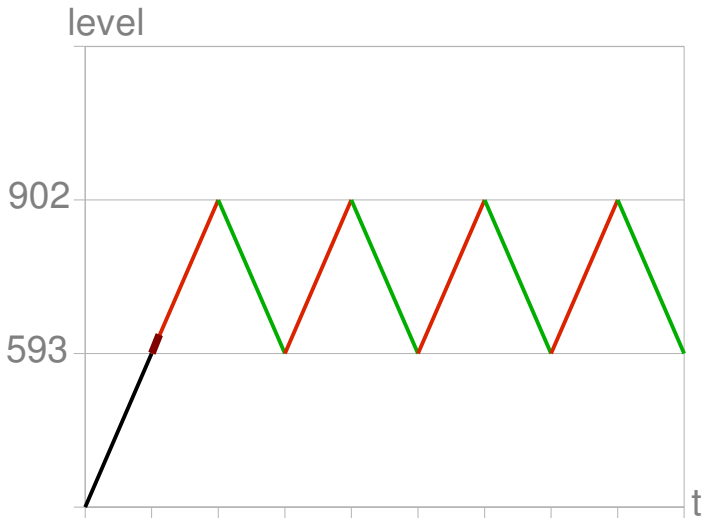
Fill: level ∈ 0 .. 902

Drain: level ∈ 593 .. 902

Alternating Behavior ✓

Eventually, level (± δ) ∈ [600 ; 900] ✖

What is really happening



3 phases:

Fill$_0$ level (± δ) ∈ 0 .. 600 ↗

↓

Fill level (± δ) ∈ 600 .. 900 ↗

↓ ↑

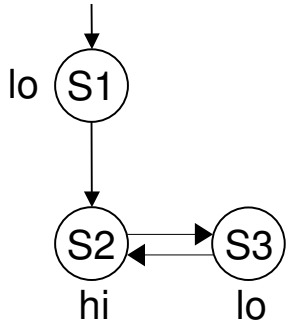
Drain level (± δ) ∈ 600 .. 900 ↘

Scenarios

(Meta-Processes as Observer Automata)

Scenario

```
wait lo;
loop {
      wait hi;
      wait lo;
}
```



System w/o Scenario

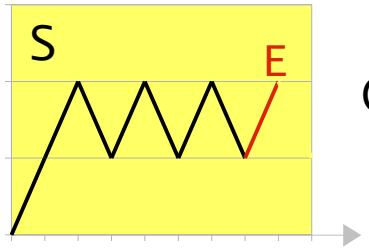
config	name	level
F2 D1	fill	0 .. 902
F2 D2	drain	593 .. 902

Synchronized Product *(for free)*

config	name	level
S1 F2 D1	$fill_0$	0 .. 602
S2 F2 D1	fill	593 .. 902
S3 F2 D2	drain	593 .. 902

Scenario: Engineering point of view

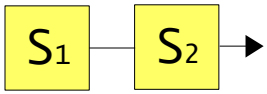
Scenario ≡ (a set of) System traces



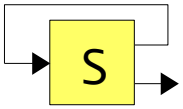
Observation:
`wait E`

Imperative Programming Language

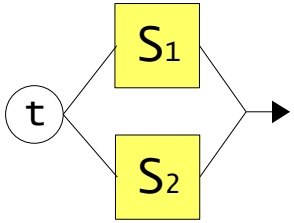
Sequence:
$S_1 ; S_2$



Loop:
`Loop S`



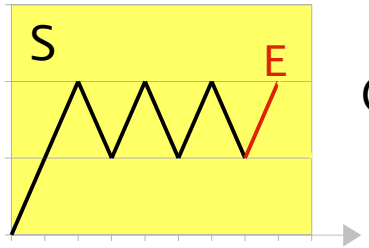
Test:
`t? S1:S2`



Assignment:
`x := exp`

Scenario: Engineering point of view

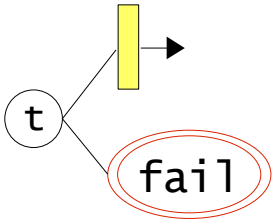
Scenario ≡ (a set of) System traces



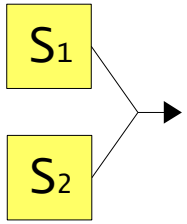
Observation:
`wait E`

Behavior Specification

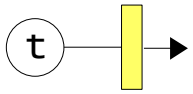
Assertion:
`assert t`



Union:
$S_1 | S_2$



Cut:
`assume t`



Non-determinism:
`x := random()`

More specification

Eventually, level ($\pm\ \delta$) $\in$ [600 ; 900] ✓

```
wait(lo);
loop {
        wait clock;
        assert 590 ≤ level ≤ 910;
}
```

The Specification?

Monotony

```
while (!lo) {
        int mark = level;
        wait clock;
        assert mark ≤ level;
}
```

Computed Properties

Conclusion

«Abstract Interpretation and Model Checking cooperating »

« Methodology and Formalism for Software Engineering »