STANCE : un outil d'analyse de contrexemples inspiré du test

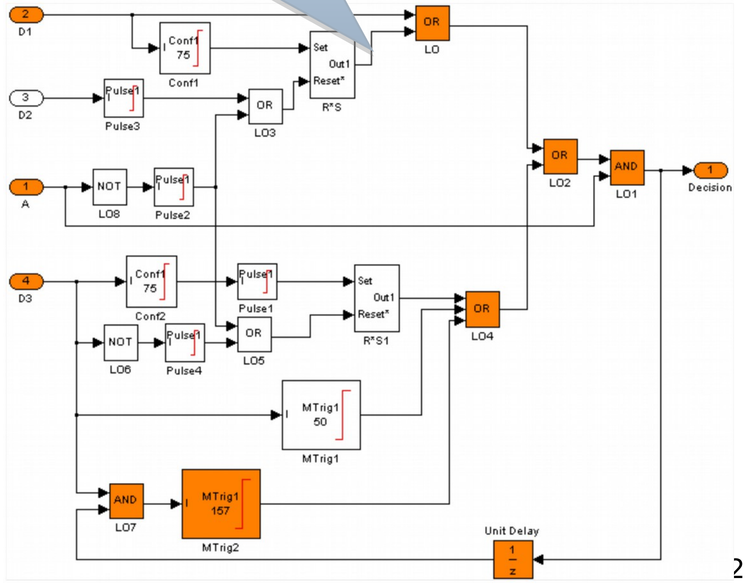
Kalou Cabrera Castillos, Hélène Waeselynck, Virginie Wiels

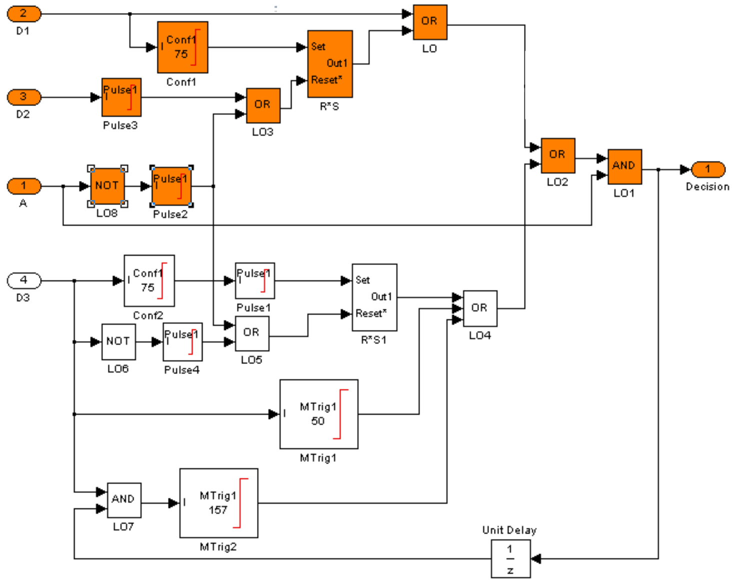
In a nutshell…

- Lightweight verification: counterexamples used to debug Simulink models
- STANCE (Structural Analysis of Counterexamples) → visualizes counterexamples + searches for multiple counterexamples

1. Synthesize activation constraints
2. Challenge the model checker to violate the property under the constraints

Paths via this arc are inactive in the current counterexample





Overview

- Background: structural analysis of synchronous data-flow models
- Search for new counterexamples
- Application to a case study from the automotive domain
- Conclusion and future work

Simulink model and property

Compound operator

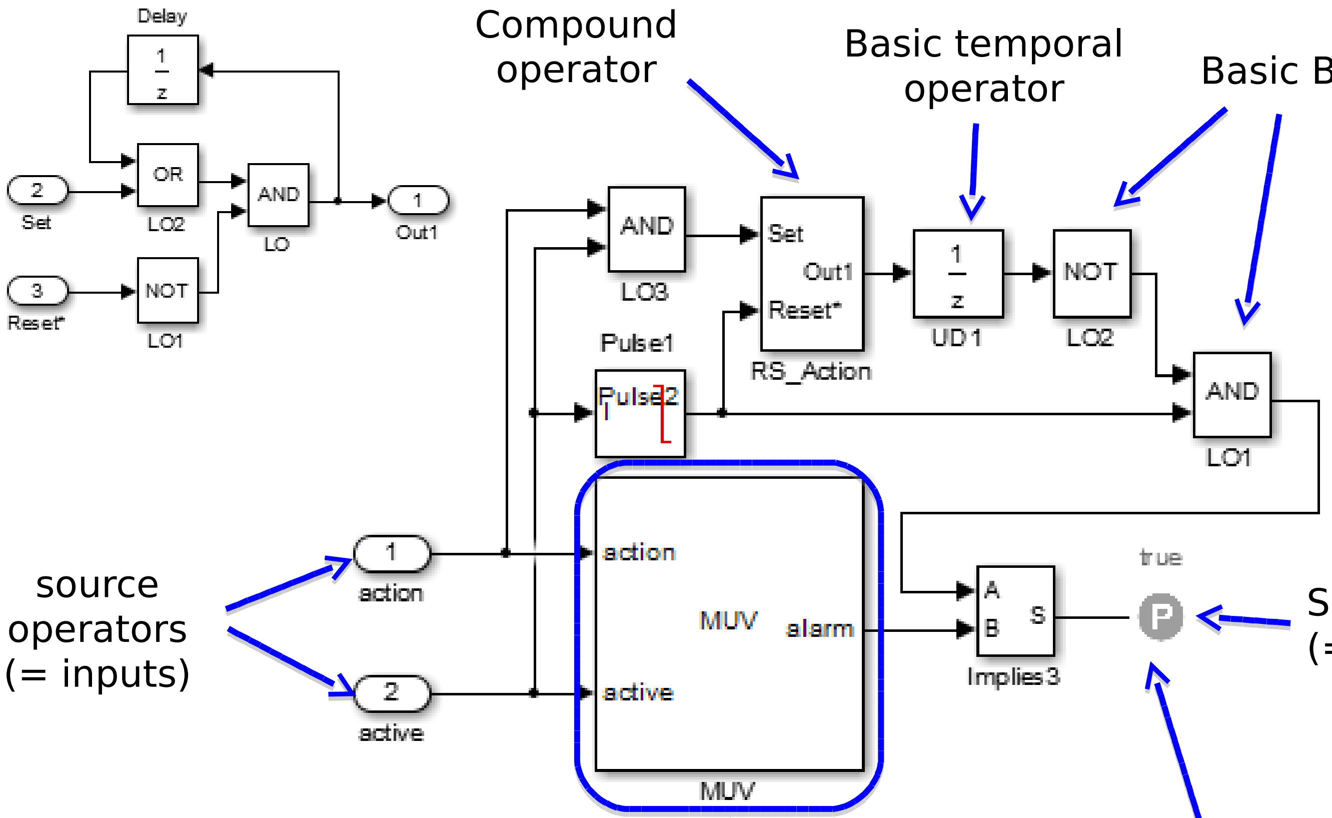
Basic temporal operator

Basic Boolean operators

+ basic numerical & relational operators

source operators (= inputs)

Sink Property operator (= output)



	Property P
action(1) active(1)	1
action(2) active(2)	1
…	1
action(n) active(n)	**0**

Model Under Verification (MUV) monitored by a **property**

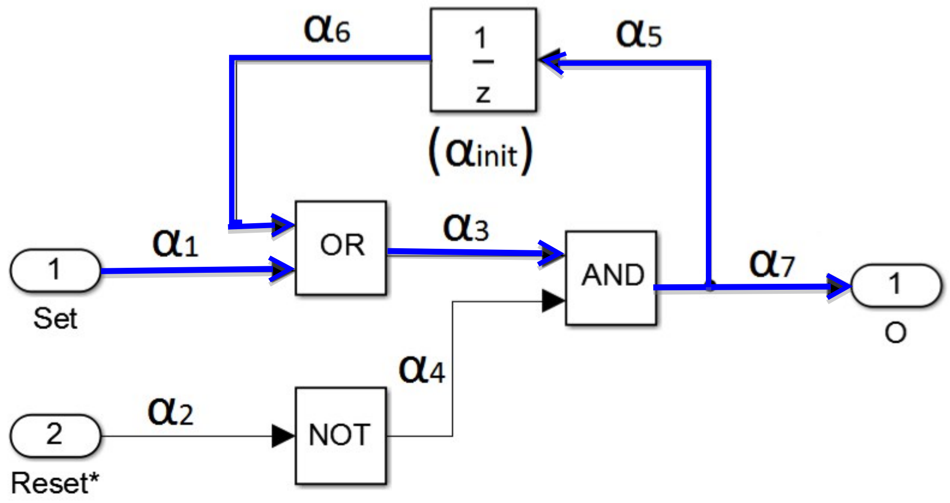
→ Same language to express the model and the property

Counterexample = sequence of n inputs that falsifies the property at the end

Paths

- Interpretation: execution is triggered by clock ticks
  - ✓ n ticks = n execution cycles
  - ✓ At each cycle, all Simulink operators are simultaneously executed
  - ✓ Path = potential data propagation channel

- Example: zoom on the RS-latch compound operator



$\alpha_1 \rightarrow \alpha_3 \rightarrow \alpha_7$

Output at cycle n may depend on the Set input at cycle n

$\alpha_1 \rightarrow \alpha_3 \rightarrow (\alpha_5 \rightarrow \alpha_6 \rightarrow \alpha_3)^k \rightarrow \alpha_7$

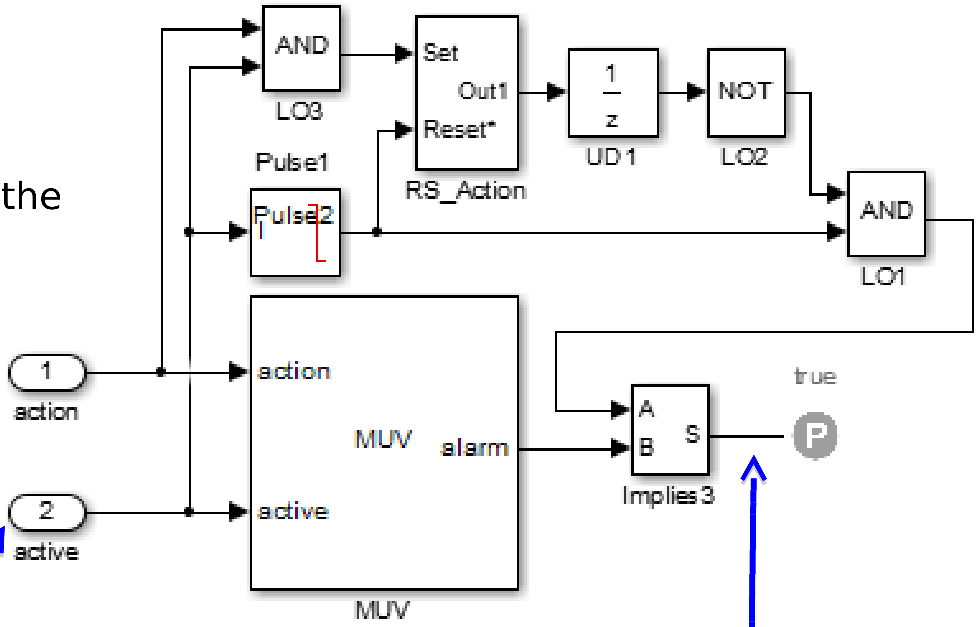
Output at cycle **n** may depend on the Set input at cycle **n-k**

(inactive during the first k cycles)

- Visualization of counterexamples: focus on paths active at the last cycle

Application to the running example

Boolean input indicating the occurrence of the action



Boolean input indicating a temporal interval in which an action is expected
E.g. active = 0**11**0
→ an action is expected at cycles 2 or 3

MUV shall issue an alarm if there is an active interval and no action

Active	0**11**0
Action	0**00**0
Alarm	000**1**

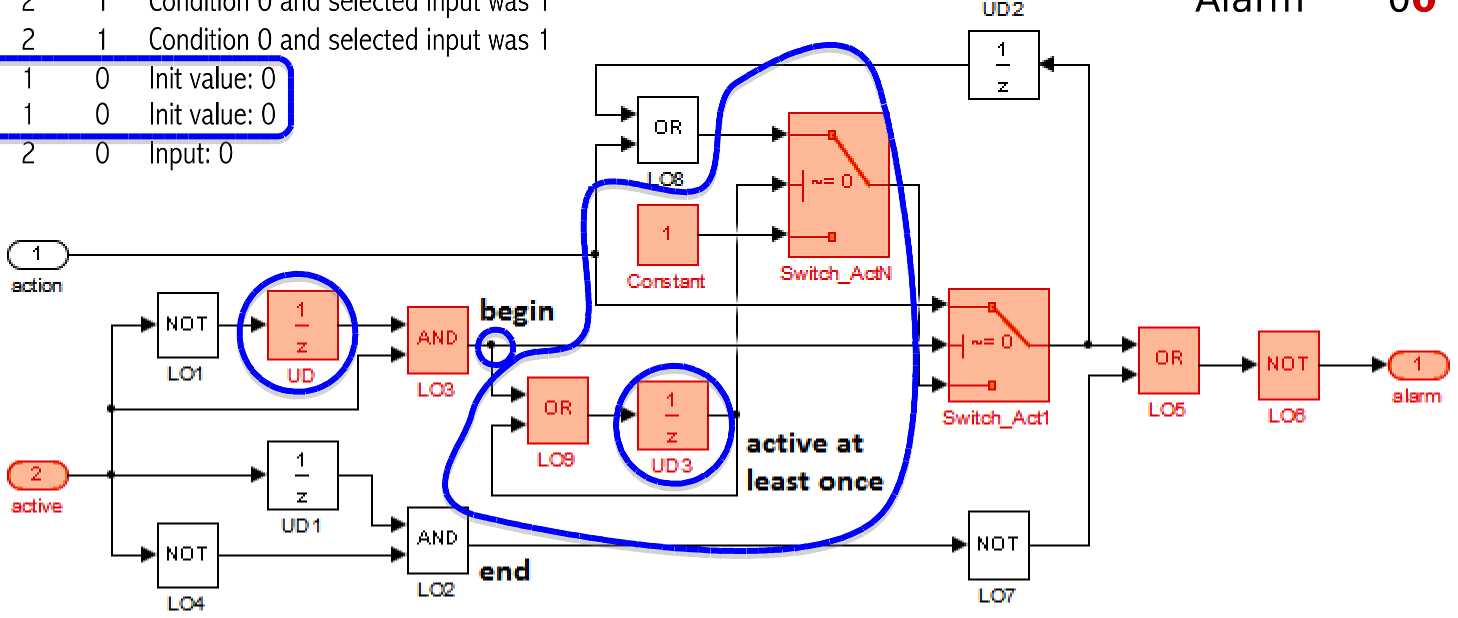
A counterexample is found by the model checker

Visualization of the counterexample

MUV Key Events

Bloc Names	Time	Value	Comment
MUV/alarm	2	0	MUV output on port alarm
MUV/Constant	2	1	Constant value: 1
MUV/Switch_Act1	2	1	Condition 0 and selected input was 1
MUV/Switch_ActN	2	1	Condition 0 and selected input was 1
MUV/UD	1	0	Init value: 0
MUV/UD3	1	0	Init value: 0
MUV/active	2	0	Input: 0

Active	**1**0
Action	**0**0
Alarm	0**0**



- The alarm does not depend on the action!
- Rather, it depends on the initial values of delays
  - ✓ Initialization problem: the beginning of the active interval is not detected
  - ✓ Convoluted structure to process the initial inactive cycles

More feedback?

- Planned revision of the design
  - ✓ Fix the initialization flaw
  - ✓ Simplify the processing of inactive cycles

- Other initialization flaws? Other problems with the processing of inactive cycles?

- Would be useful to know before attempting a revision!

⇨ Automated search for new counterexamples

Overview

- Background: structural analysis of synchronous data-flow models
- **Search for new counterexamples**
- Application to a case study from the automotive domain
- Conclusion and future work

Principle

- Force the model checker to consider violation paths via new arcs

$$MUV \models P \Rightarrow MUV \models \wedge_i C_i \Rightarrow P$$

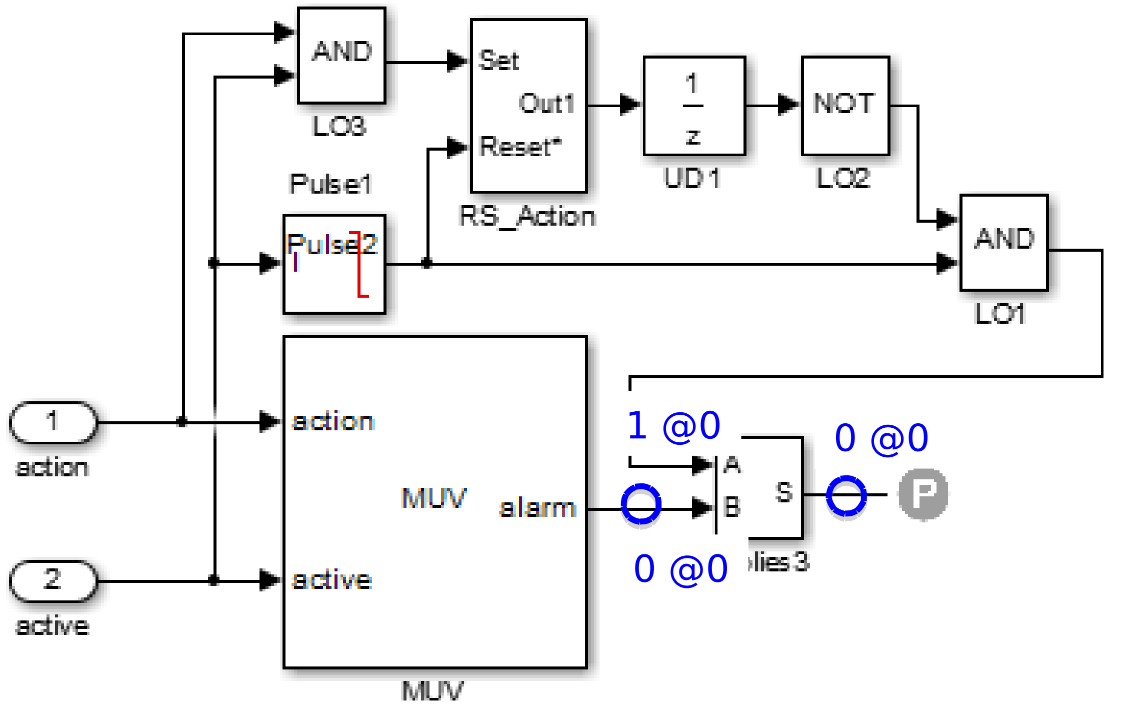
Constraints added by instrumentation of MUV + P

- Two classes of constraints in an instrumentation
  - ✓ Primary constraints: local constraints targeting the new arc
  - ✓ Secondary constraints: ensure data propagation to the property output and its falsification

Backward analysis

inputs ← property output
cycle –n+1 ← cycle 0 (*relative* time)

Primary = Ø
Secondary = ant. is 1 @0



Explore backward the active paths:

ɔn,

•New ways to produce 1 @0 at the antecedent?
Secondary: force 0 @0 at the consequent

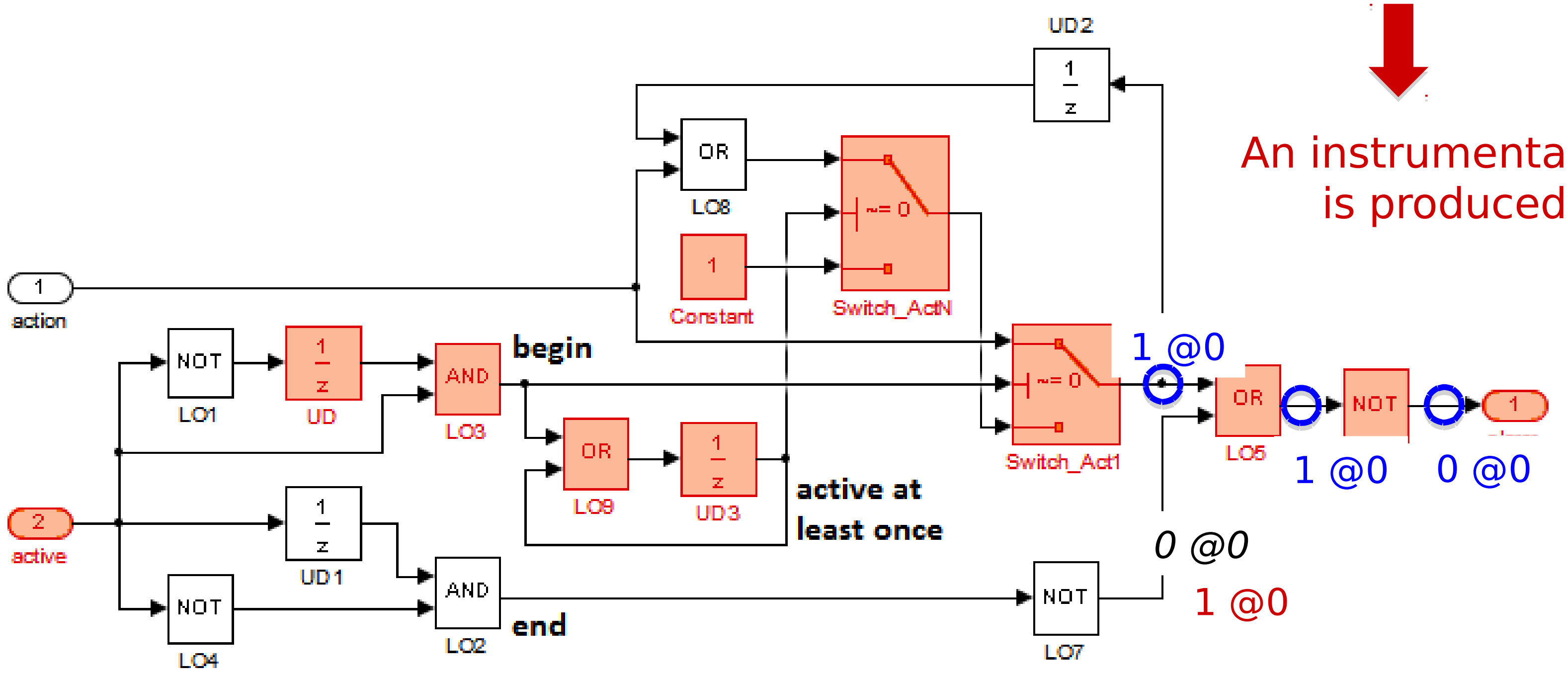
•New ways to produce 0 @0 at the consequent?
Secondary: force 1 @0 at the antecedentent

Backward analysis

inputs ← property output
cycle –n+1 ← cycle 0 (*relative* time)

Primary = OR_{i2} is 1@0
Secondary = ant. is 1 @0

An instrumentation is produced

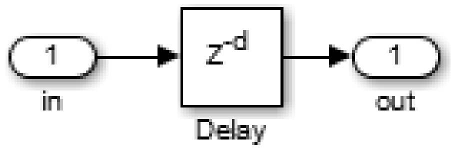


Explore backward the active paths the OR,
to produce output 1 @0?

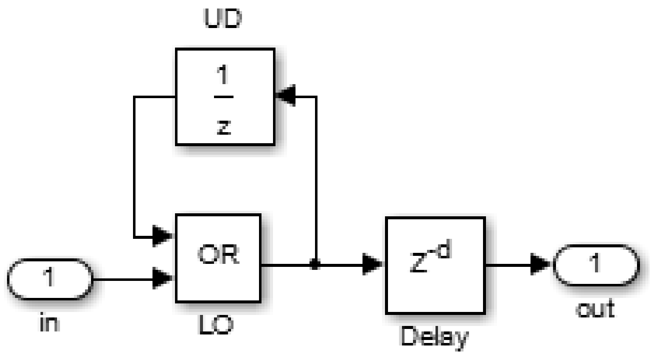
Yes! Force 1@0 at the other input of the OR!

Instrumentation blocks

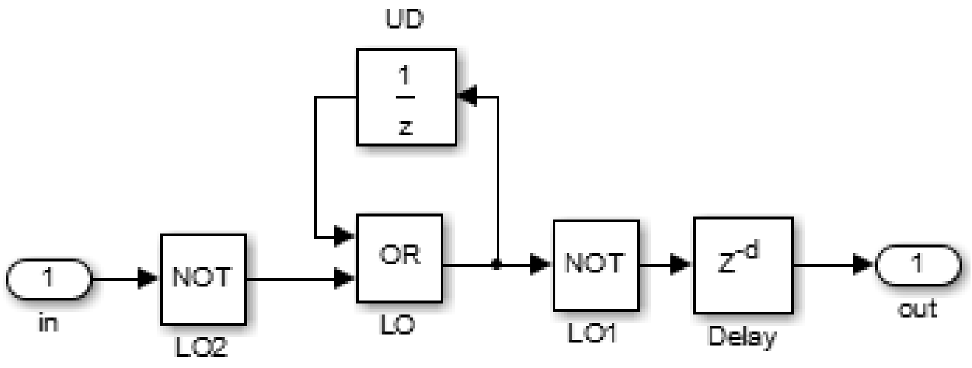
Basic value @-d:



A least once @-d or earlier:



Always until @-d:



Iterative search

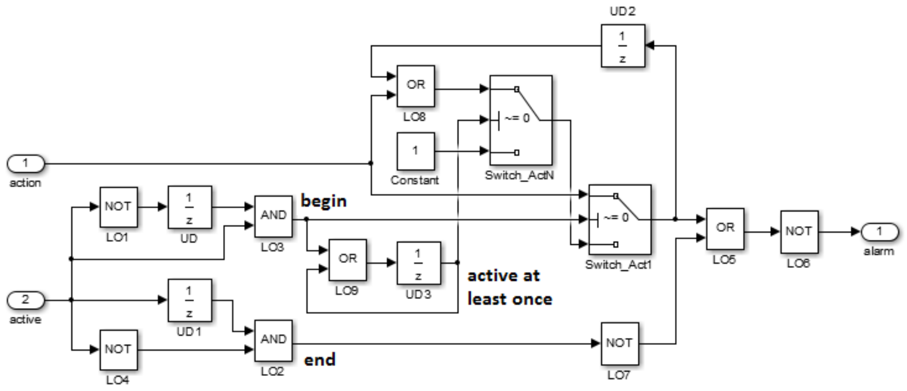
```
Intial step to process the first counterexample:
    Extract its active paths
    Produce its instrumentations

For each instrumentation
    If a counterexample is found
        Replay it on the un-instrumented model
        Extract its active paths
        If new set of paths
            Produce its instrumentations
            Retain instrumentations that target new arcs
        Endif
    Endif
Enfor
```

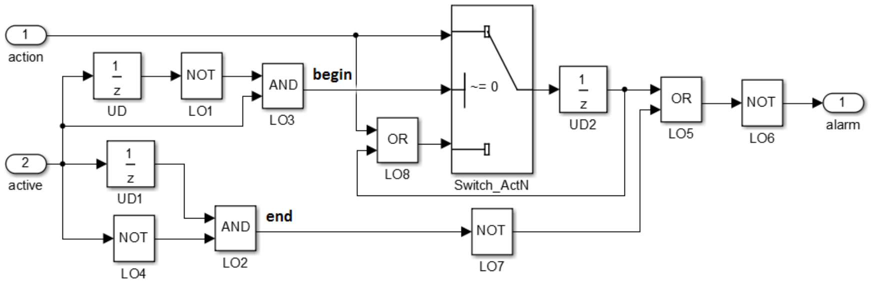
Avoids endless iteration!
(temporal loops)

Application to the running example

- Initial step → first counterexample
  - ✓ Initialization problem to detect the beginning of an active period
  - ✓ Convoluted design to process the inactive cycles
- Five iterations, one new counterexample found
  - ✓ No alarm if action arrives exactly one cycle too late (i.e., at the first inactive cycle after an active interval)



Original (flawed) design



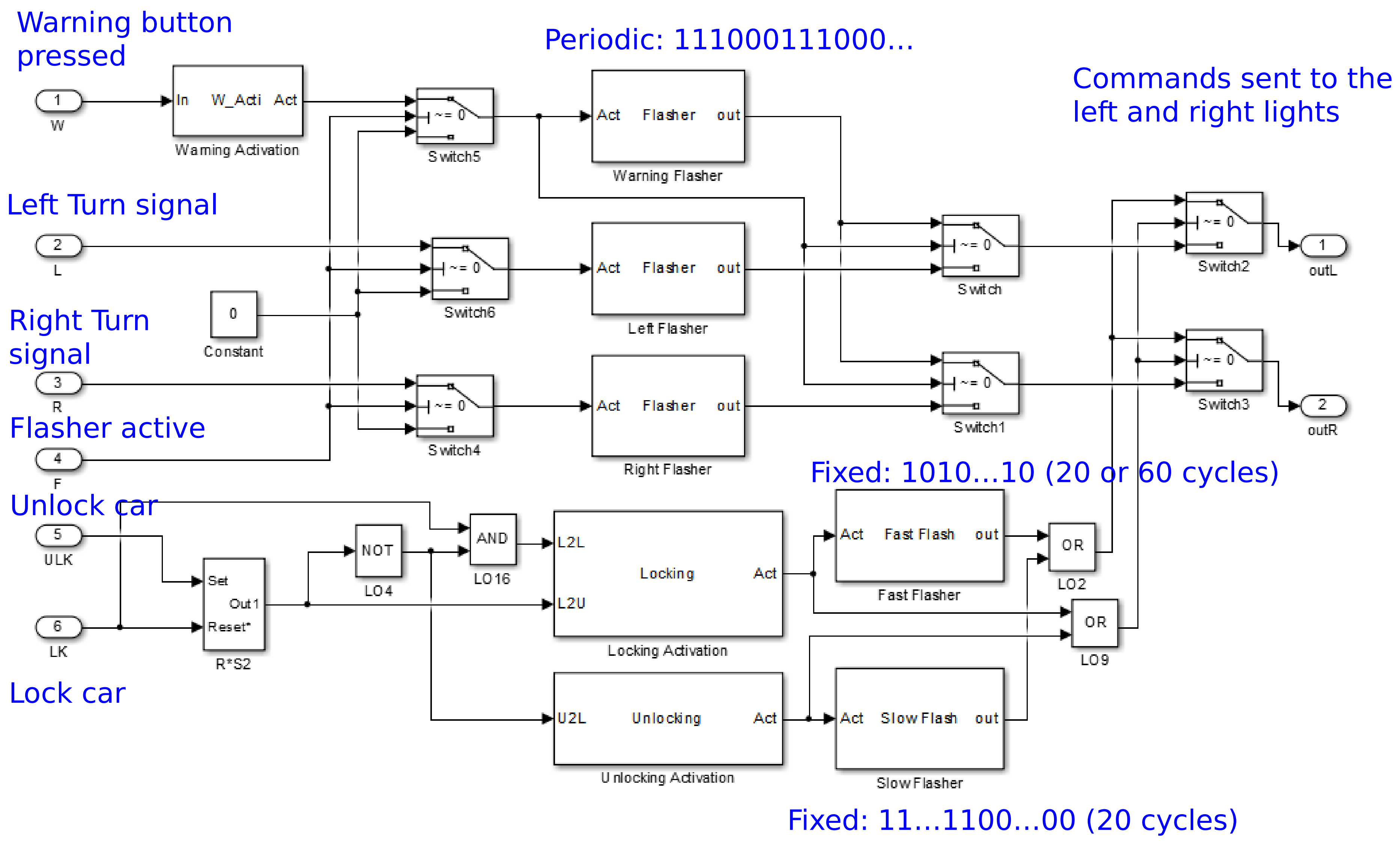
Revised (correct) design

- ✓ Fixes the initialization problem (Cex1)
- ✓ Simplifies and fixes the processing of inactive cycles (Cex1 & Cex2)

Overview

- Background: structural analysis of synchronous data-flow models
- Search for new counterexamples
- **Application to a case study from the automotive domain**
- Conclusion and future work

A flasher manager (Geensoft/Dassault Systems)



Checked property

- "Lights should never remain lit infinitely"
- Not checkable → bounded version
  "Lights should never remain lit during X cycles"
- Falsified for X from 10 to 1,600 [Collavizza 2014]
  - ✓ Authors did not explain why (it was not their point!)

Let's take a small X (=10) and explore the violation patterns

Counterexamples (9 found)

- User can lit a light for 10 cycles by:
  - ✓ Unlocking the car and then locking it back (Cex1) Would not work for X>10
  - ✓ Repeatedly acting on the warning button and the direction change lever (Cex2, Cex5, Cex7, Cex8)
  - ✓ Repeatedly acting on the warning button and the unlock/lock buttons (Cex3, Cex4, Cex6, Cex9)

Would work for any X!

- User has to keep acting forever (and be fast!)
  - ✓ No self-sustained scenario observed
  - ✓ Interval of 3 cycles to perform the next user action

Feedback from the counterexamples

Option 1: Introduce user assumptions

E.g, slow user (4 cycles) → successfully model checked with X = 20

Option 2: Revise the design

E.g., add logic to ignore user actions or delay response to actions

Overview

- Background: structural analysis of synchronous data-flow models
- Search for new counterexamples
- Application to a case study from the automotive domain
- **Conclusion and future work**

Conclusion and perspective

- Search for new counterexamples
  - ✓ Is a functionality of our STANCE tool (Structural Analysis of Counterexamples)
  - ✓ Works in integration with the Simulink environment
  - ✓ Is driven by structural coverage criteria

- Provides feedback about the different violation patterns
  - ✓ Application to an academic example + industrial case study

- Future work:
  - ✓ If numerous counterexamples found, extract the most “insightful” ones – the most distant? the least complex?
  - ✓ Investigate connection with fault localization approaches