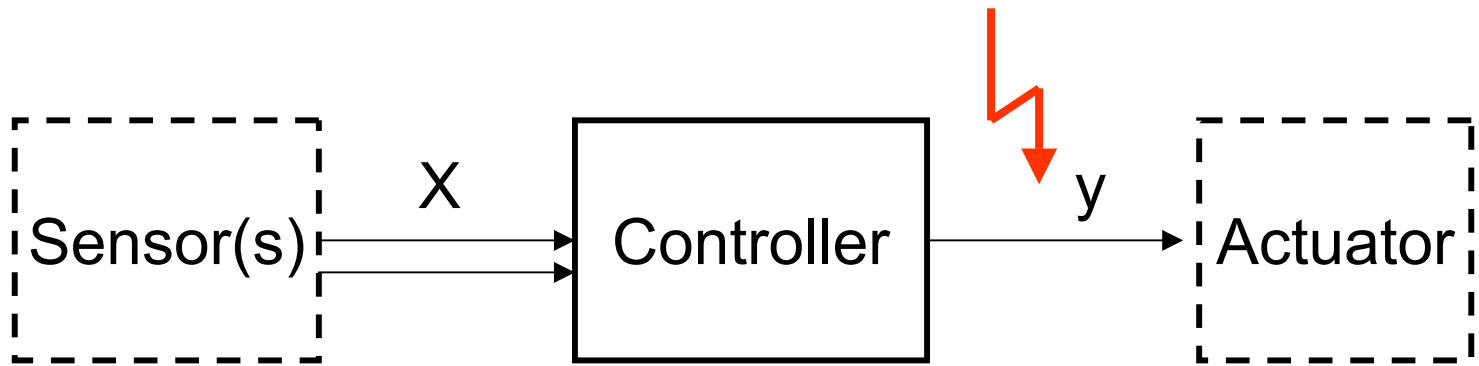


Dr. Matthias Bretschneider  
Hamburg - Dept. Safety

# Fehleranalyse mit Hilfe von Model Checkern



# Motivation: Design of safe embedded systems



## Design Phase

- Study the effect of failures on system behaviour
- Objective: fault-tolerance – no single failure may lead to catastrophic effects

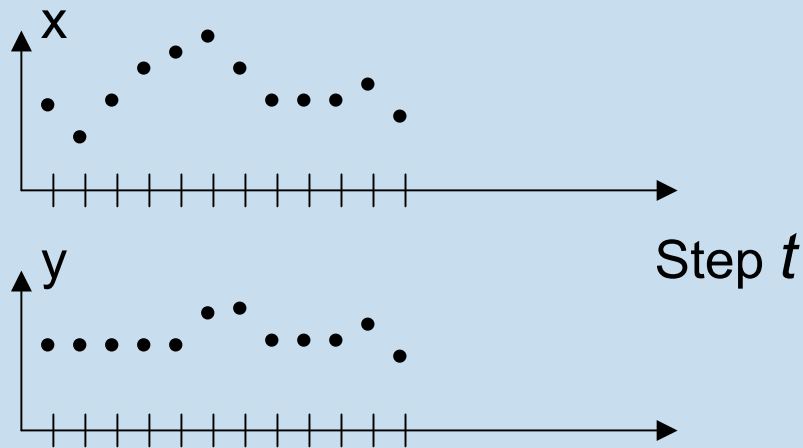
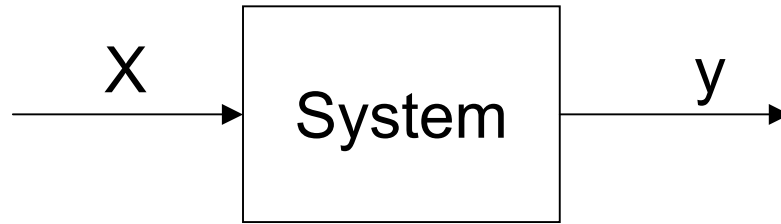
## Approach

- Use formal models
- Apply model checker (do formal verification)

1. Formal Models
2. Formal Verification
3. Failure Analysis
4. Case Study



# Formal (dynamic) Models



$x(1), x(2), \dots, x(t-1), x(t), \dots$   
 $y(1), y(2), \dots, y(t), \dots$

Output  $y(t)$  depends on previous inputs (State)

Examples:  $y(t) = x(t) - x(t-1)$

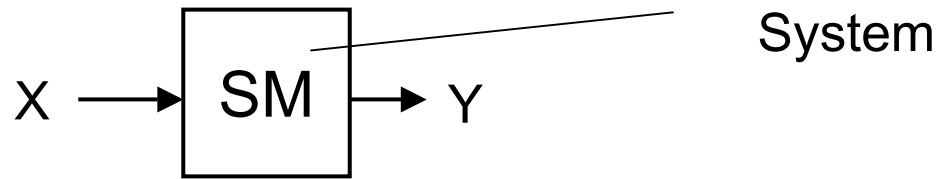
$y(t) = \text{if } x(t) \text{ then } y(t-1) \text{ else } y(t-2)$

Typically,  $X$  and  $Y$  are vectors.

Switch  
Non-linear

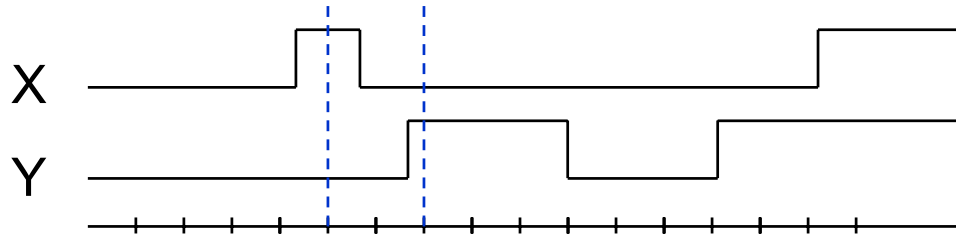


# Properties

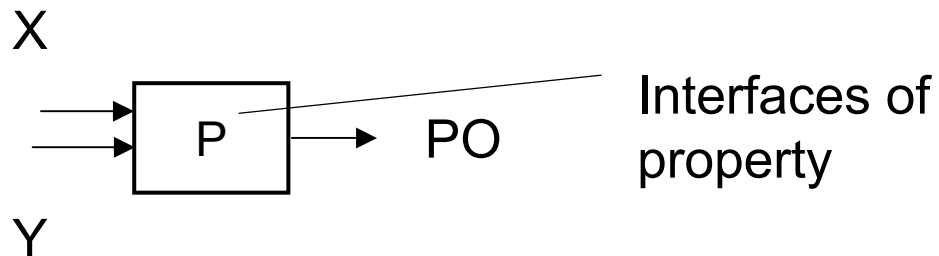


Example 1:  $Y > X$

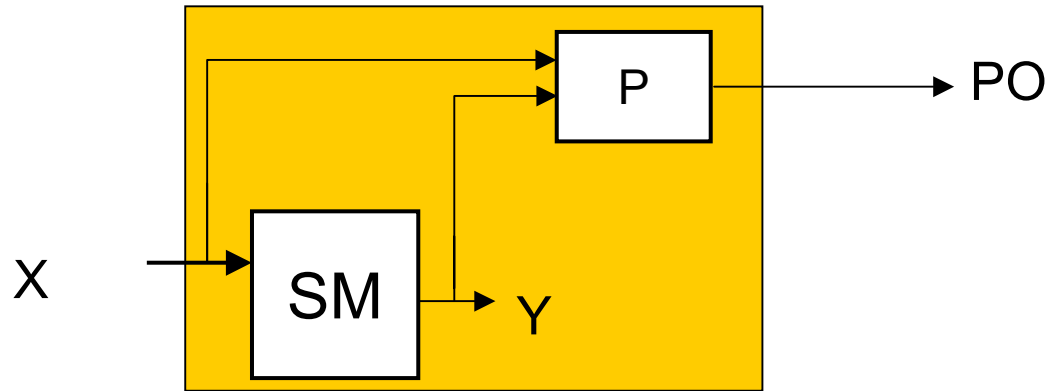
Example 2: Whenever  $X(t)=\text{true}$ , then  $Y(t+2)=\text{true}$  (delayed reaction)



For each step  $t$ , a property  $P$  is either true or false.  
Its value is denoted by  $PO$ .



# Verification of Properties



## Proof Obligation (“Claim”):

For all valuations of input sequence  $(X(t), t=1, 2, 3, \dots)$ , the output sequence yields  $PO = (\text{true}, \text{true}, \text{true}, \dots)$ .

- Universal quantification over valuations and steps is made.
- Compare “**AG PO**” in temporal logics
- Existential quantification not considered here.
- The proof obligation is either true (“**valid**”) or false (“**falsifiable**”):



# Proof Obligation: Interpretation

Step	1	2	3	4	5	...
X	0	0	0	0	0	...
PO	true	true	true	true	true	true

Step	1	2	3	4	5	...
X	0	5	0	0	0	...
PO	true	true	true	true	true	true

Step	1	2	3	4	5	...
X	*	*	*	*	*	...
PO	true	true	true	true	true	true

Arbitrary Input X

Whatever the input X, the output PO is true

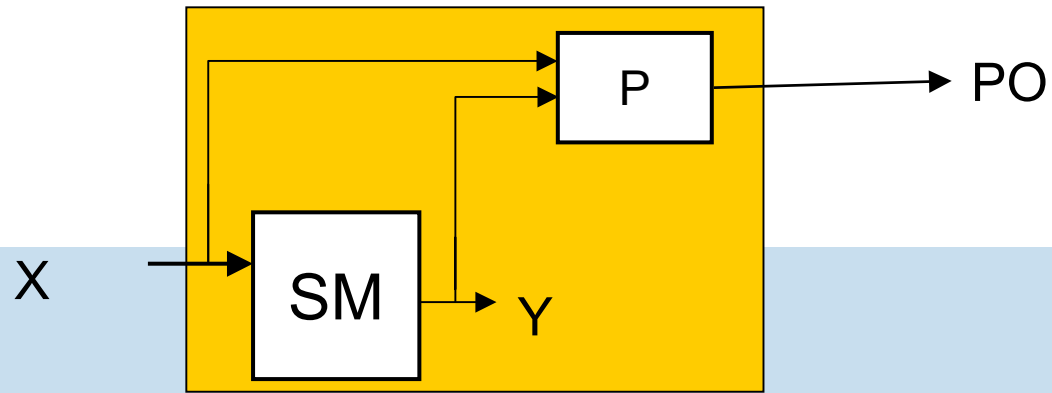
The property always holds.



# Falsification by means of Counter-Examples

$$Y = X^2$$

$$P := (Y > X)$$



The property  $P$  is falsified by the following valuation:

$$\underline{X(1) \rightarrow 0.5}$$

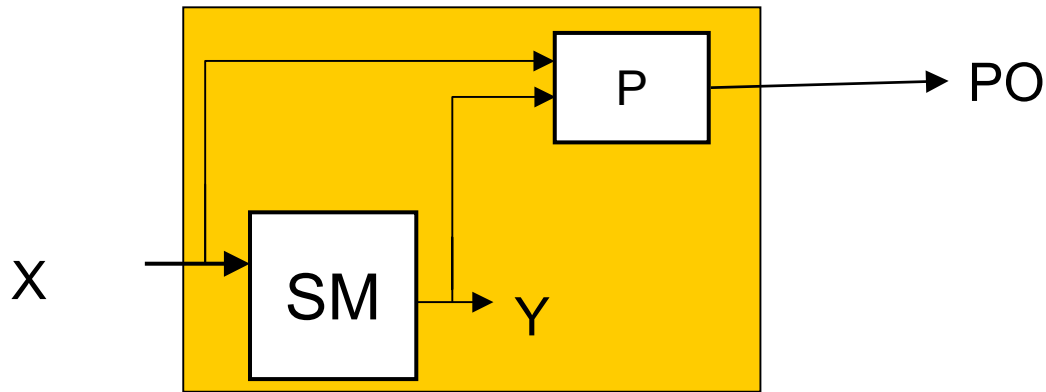
$$Y(1) \rightarrow 0.25$$

$$PO(1) \rightarrow \text{false}$$

Counter-Example has length 1.



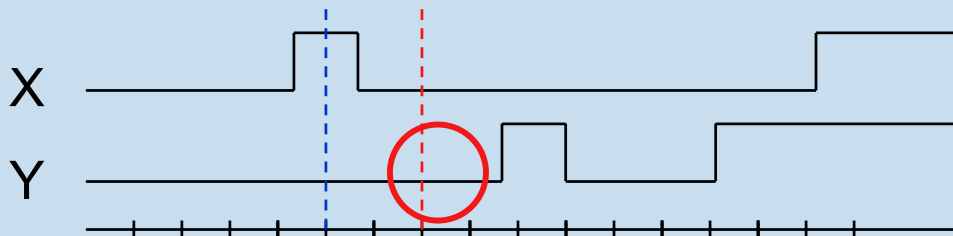
# Counter-Examples (dynamic) - principle



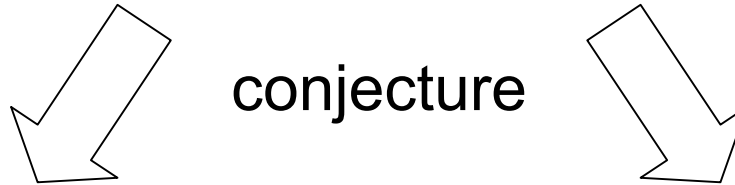
$X(1) \rightarrow v_1, X(2) \rightarrow v_2, \dots, X(t) \rightarrow v_t$  (sequence of length  $t$ )

$PO(1) = PO(2) = \dots = PO(t-1) = \text{true}$ , and  $PO(t) = \text{false}$

Example 2: Whenever  $X(t) = \text{true}$ , then  $Y(t+2) = \text{true}$  (delayed reaction)



## Proof Obligation (“Claim”):



Valid ?

- Use “**Prove Strategy**”

Falsifiable ?

- Use “**Debug Strategy**” – search for Counter-Examples of length less than N steps
- Fast algorithm



# Prove Strategy – Temporal Induction 1

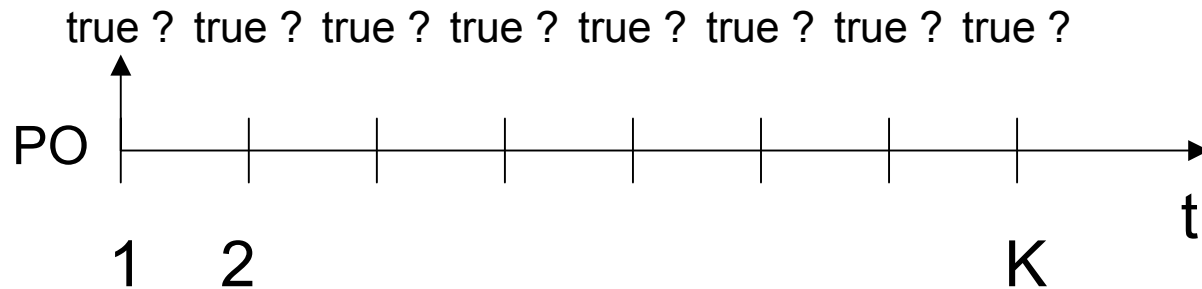
**Claim:** property  $PO(t)$  always true (all inputs, all steps)

**Idea:** Consider  $K$  consecutive steps of the automaton

## Base Case

Initial State:  $z(0) = z_0$ , Steps  $t = 1, 2, \dots, K$

Show: For all inputs  $x(t) \in I$ :  $PO(t) = \text{true}$



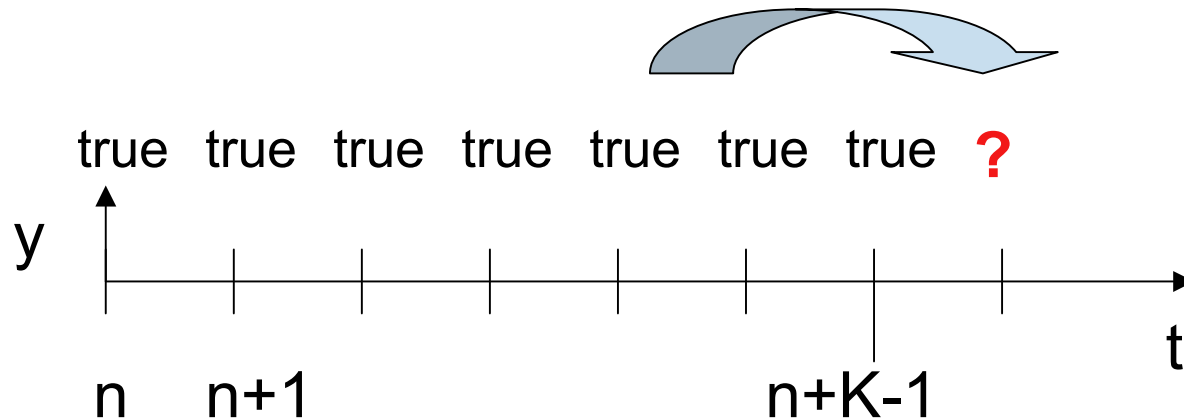
When proof of Base Case fails  $\rightarrow$  Counter-Example



# Prove Strategy – Temporal Induction 2

## Induction Step

K Steps:  $t = n, n+1, \dots, n+K-1$ ,  $n$  arbitrary



Initial State:  $z(n) \in Z$  ("free variable")

**Assumption:** For all steps  $t = n, \dots, n+K-1$ , all  $x(t) \in I : PO(t) = \text{true}$

**Prove:**  $PO(n+K) = \text{true}$

When Induction Step fails, set  $k \rightarrow k+1$



# Prove Strategy – Temporal Induction 3

When **base case** and **induction step** are proved then you know:

$PO(t) = \text{true}$  holds

- for all  $t=1,2,3,\dots$
- for all input sequences  $x(t) \in I$

Note: The analysis starts with  $K=1$

Tool: Sat-Solver (Prover Technology, Stockholm)

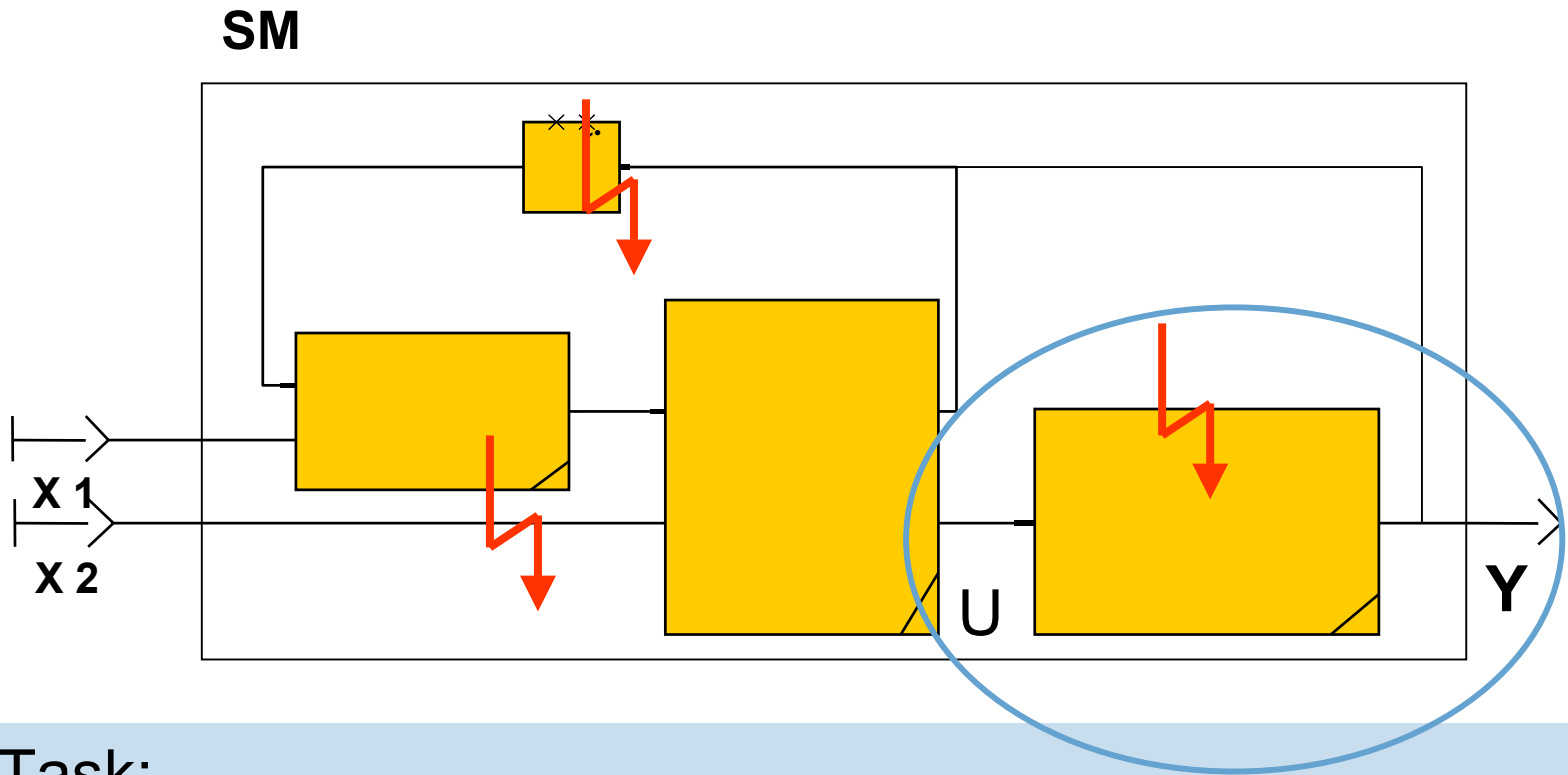
Algorithm works for **finite** state systems and some infinite state systems (Presburger Arithmetic, ...)




1. Formal Models
2. Formal Verification
3. Failure Analysis
4. Case Study



# Failure Analysis: Objectives



## Task:

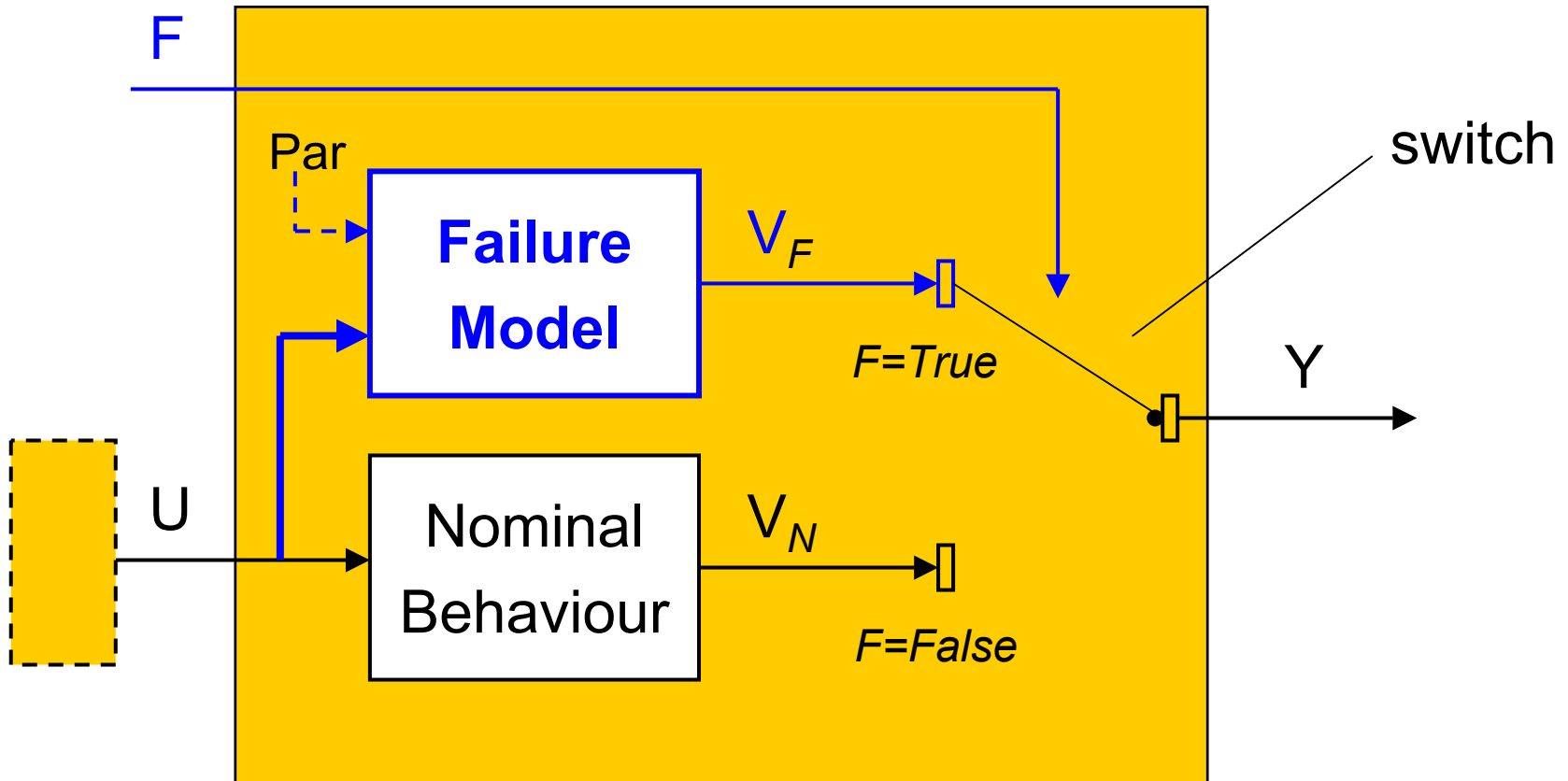
- Inject component failures 
- Compute the propagation
- Determine effect on (Safety) Property

# Injection of Failure Mode (principle)

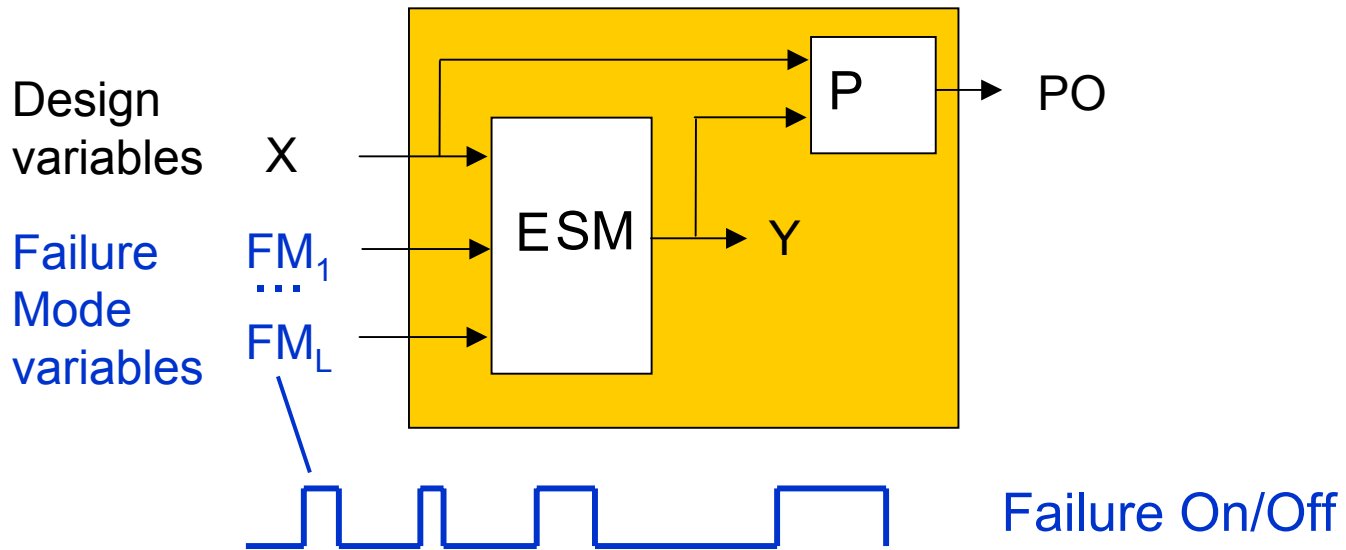
Failure Mode  
control variable



Failure On/Off



# Determine failure effect on Property



## Proof Obligation (“extended”)


For all valuations of input sequence  $(X(t), FM(t), t=1,2,3,\dots)$ ,

$PO = (\text{true}, \text{true}, \text{true}, \dots)$  holds.




Free variables: Failure ON/OFF

# Proof Obligation (“extended”) : Interpretation



Step	1	2	3	4	5	...
X	*	*	*	*	*	...
FM	false	true	false	false	false	...
PO	true	true	true	true	true	true



Step	1	2	3	4	5	...
X	*	*	*	*	*	...
FM	*	*	*	*	*	...
PO	true	true	true	true	true	true

Arbitrary values

- Whatever the design input X, and
- whenever the Failure Modes occur,  
the property PO is true



**Output:** Proof obligation is ...

valid: Each Failure Mode has no impact on Requirement

falsified: Some Failure Modes have an impact → analyse counter-example:

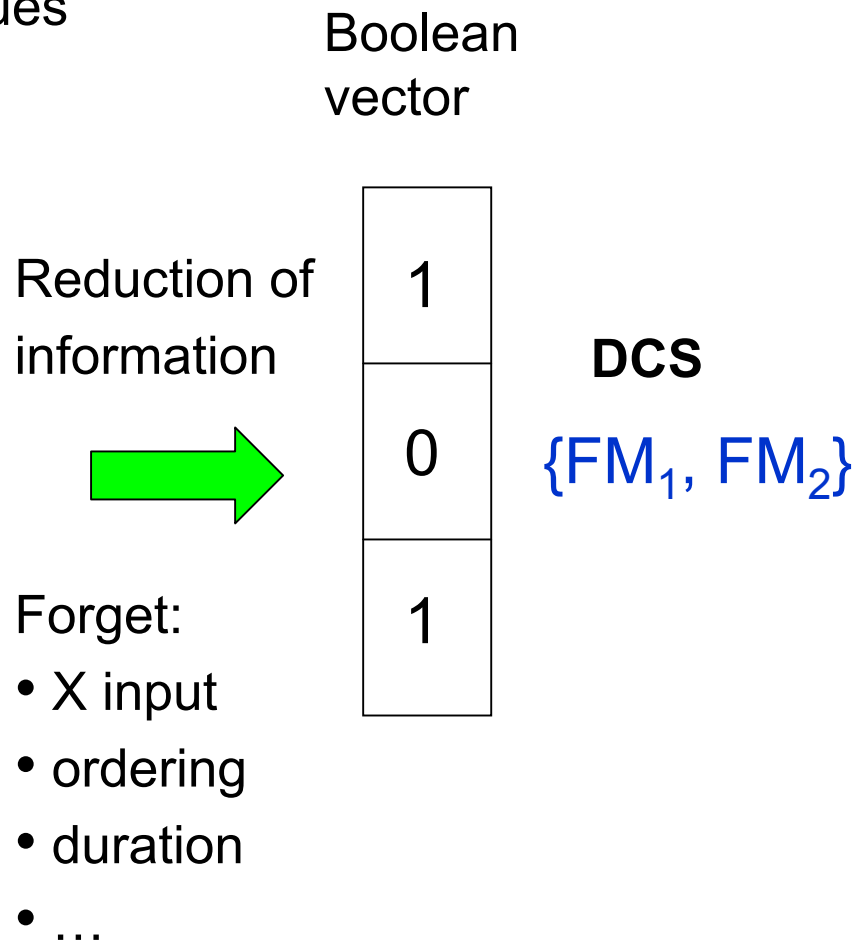
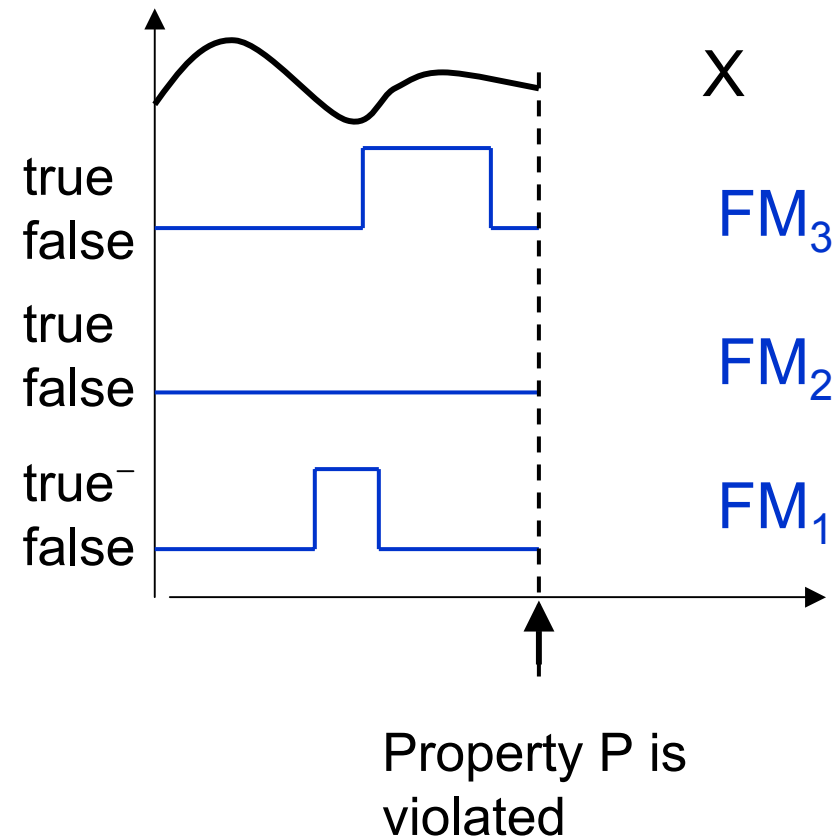
## Analysis – step by step:

- Restrict the number  $M$  of failure modes that can be triggered:
  - $M=0$  : valid → fault-free system satisfies property  $P$
  - $M=1$  : valid → system tolerates **single** failures (w.r.t.  $P$ )
  - $M=2$  : valid → system tolerates **double** failures (w.r.t.  $P$ )
- 
- New Concept “Dynamic Cut Sets”



# Analysis of Counterexample 1 – dynamic cut set

- Counter example: sequence of values



# Output of Failure Analysis

- Scenarios showing when these failure occur (ordering, dependency on system state)
- List of (all) single, double, triple failures ... (if any) that violate the (safety) property P.

From this the design engineer learns what he has to do to make his/her system **fault-tolerant** .

## Tool

- Fault Tree Manager (Prover Technology, Stockholm), developed within European research project **ISAAC**



# Failure Analysis vs. Simulation

## **Simulation**

The failure modes are triggered “by hand”. This is based on a decision

- when the failures become active (per step)
  - how long the test sequence is
- 
- No complexity problem
  - Critical scenario might be overlooked. ( N.B. failure effect depends on system state)

## **Failure Analysis based on model checking**

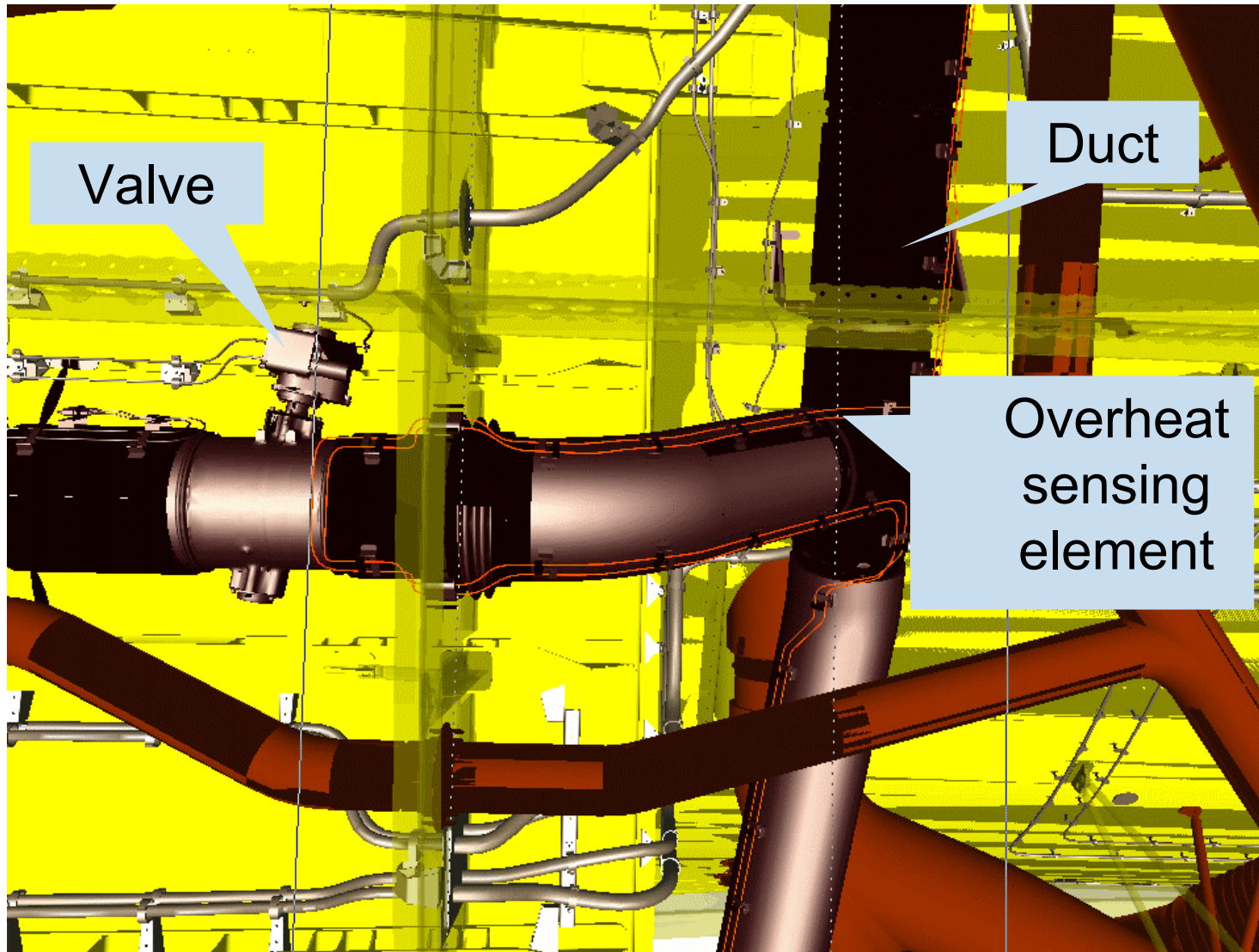
- All scenarios will be considered (with respect to user-defined set of failure modes)
- Potential complexity problem (computation does not terminate)



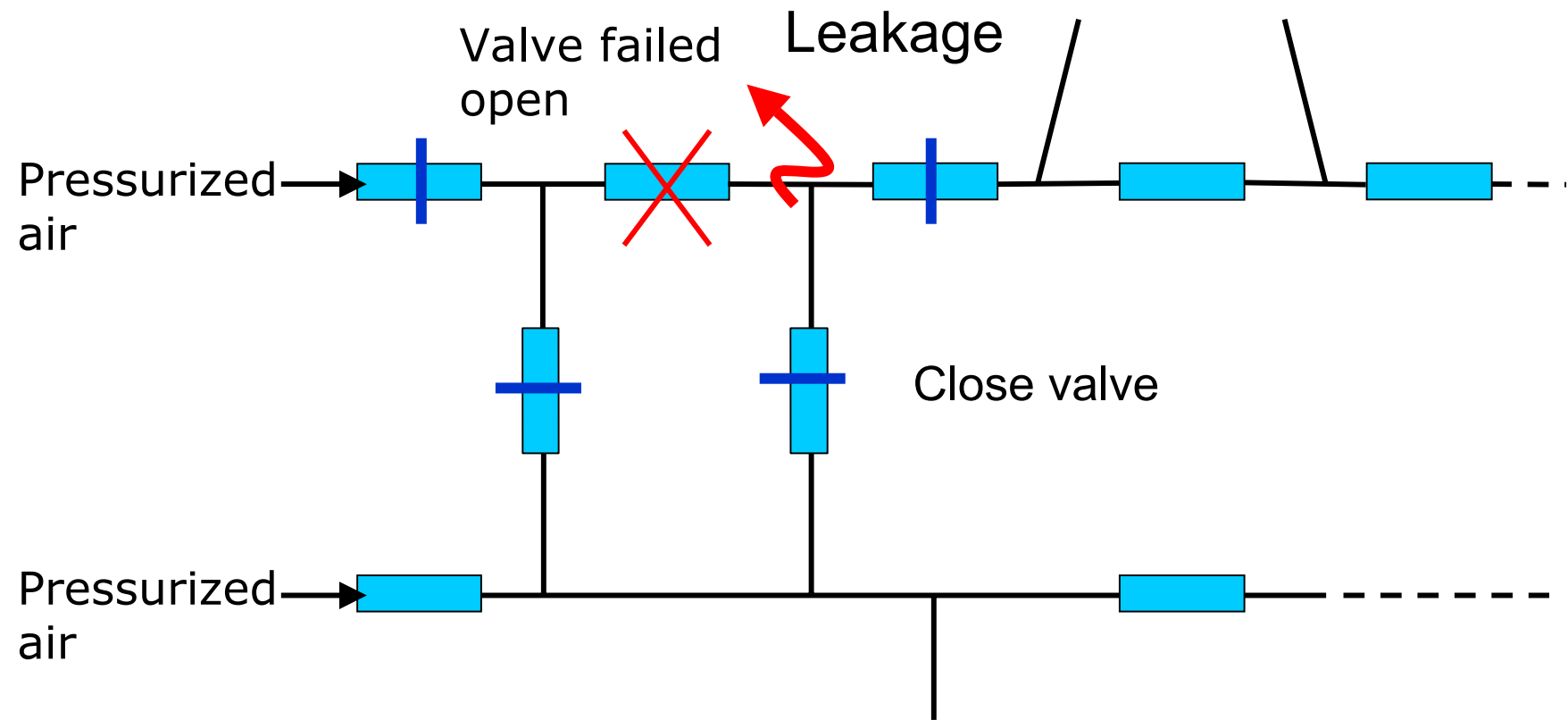
1. Formal Models
2. Formal Verification
3. Failure Analysis
4. Case Study



# Example - Pressurized Air Ducts



# Isolation Procedure in case of valve failure



Isolation Procedure is implemented by Valve Control Logic



# Safety Requirements R1 – R3

- R1 Leaking sections shall be isolated from pressurized air sources.
- R2 This shall be possible with a single valve failed open.
- R3 Isolation shall be possible within time  $T_C$ .



## **Objective:**

Utilize Design Models to support traditional safety and reliability analysis techniques (FMEA, FTA).

## **Approach.**

Use **model checkers** for exhaustive analysis (“find all critical scenarios”)

By means of simulation you will find some scenarios.

Limitation: “state explosion problem”

## **Further information:**

EU research project ISAAC:

Improvement of Safety Activities on Aeronautical Complex systems

<http://www.cert.fr/isaac/>

