

# Industrial-Strength Model- Based Testing - State of the Art and Current Challenges

Jan Peleska

University of Bremen

Verified Systems International GmbH

MBT Workshop, Rome, 2013-03-17



Universität Bremen

C © M P A S S



# Overview

- Model-based testing
- A reference tool
- Modelling aspects
- Requirements, test cases and strategies
- Conclusion – challenges

# ● **Model-based testing**

- A reference tool
- Modelling aspects
- Requirements, test cases and strategies
- Conclusion – challenges

# Model-Based Testing

- Model-based testing (MBT) as defined in Wikipedia
- **“*Model-based testing* is application of [Model based design](#) for designing and optionally also executing artefacts to perform [software testing](#). Models can be used to represent the desired behaviour of a System Under Test (SUT), or to represent testing strategies and a test environment.”**

# Model-Based Testing

- Model-based testing (MBT) as defined in Wikipedia

- “**Model-based testing** is the application of Model based designing and optionally also executing artefacts to perform software testing. Models can be used to represent the desired behaviour of a System Under Test (SUT), or to represent testing strategies and a test environment.”

We would say:  
*system or software testing*

# Model-Based Testing

Let's analyse this definition

- “*Apply model-based design*”: use modelling formalism to specify any test-related information
  - “*Models ...represent desired behaviour of ... SUT*”: Just specify the desired capabilities of the SUT
  - ... or, alternatively ...

# Model-Based Testing

- *“Models ... represent testing strategies and a test environment”:*
- It is explicitly modelled how test cases and associated test data should be produced and
- how these should interact with the SUT
- Here MBT helps to
  - represent test cases in a concise and intuitive way
  - transform test cases and data into executable test procedures

# Our MBT Approach

Instead of writing test procedures,

- develop a **test model** specifying expected behaviour of SUT → the first MBT variant
- use **generator** to identify “relevant” test cases from the model and calculate concrete test data
- generate **test procedures** fully automatic
- perform **tracing** requirements ↔ test cases in a fully automatic way



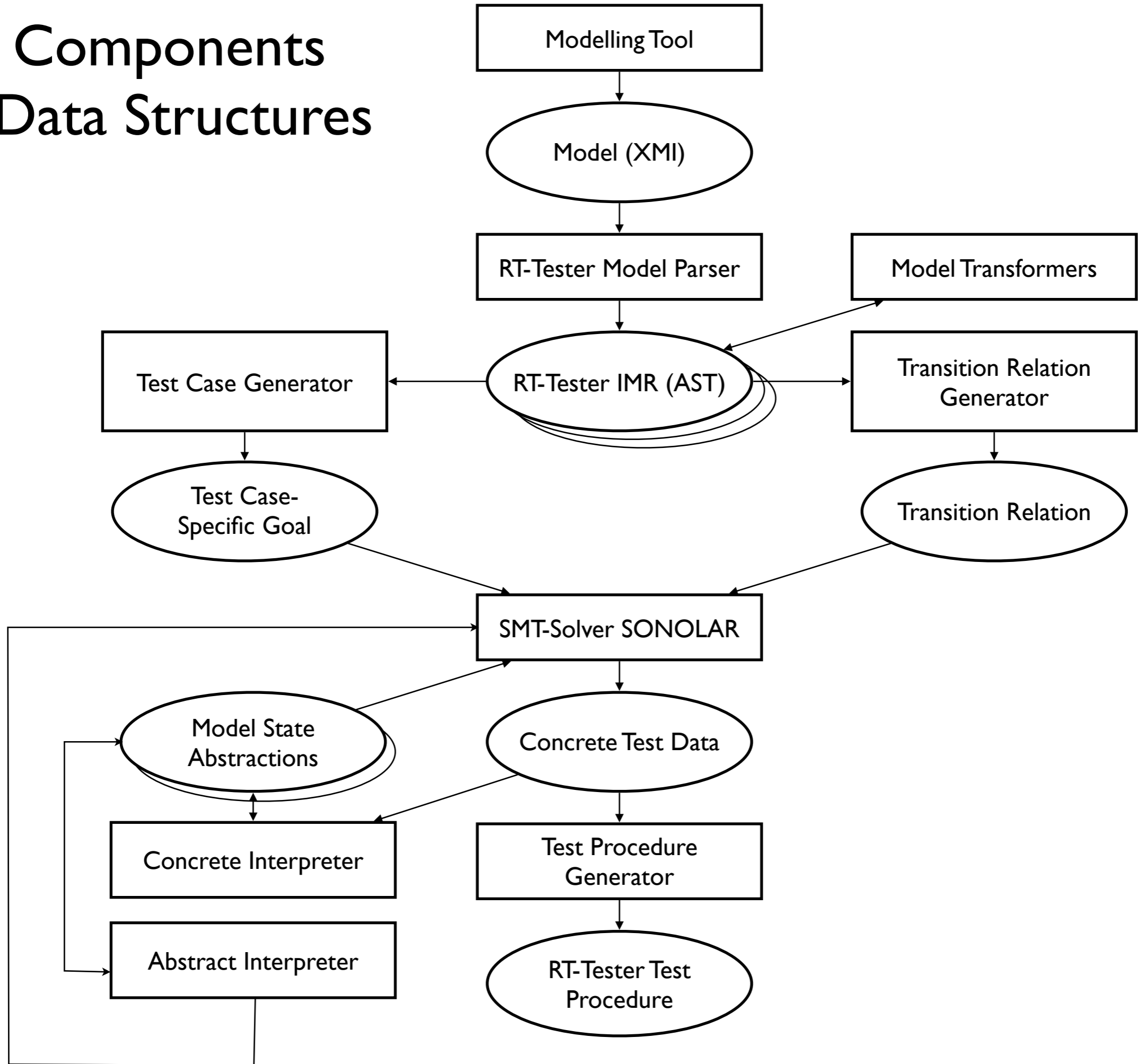
- Model-based testing
- **A reference tool**
- Modelling aspects
- Requirements, test cases and strategies
- Conclusion – challenges

# Reference Tool RT-Tester

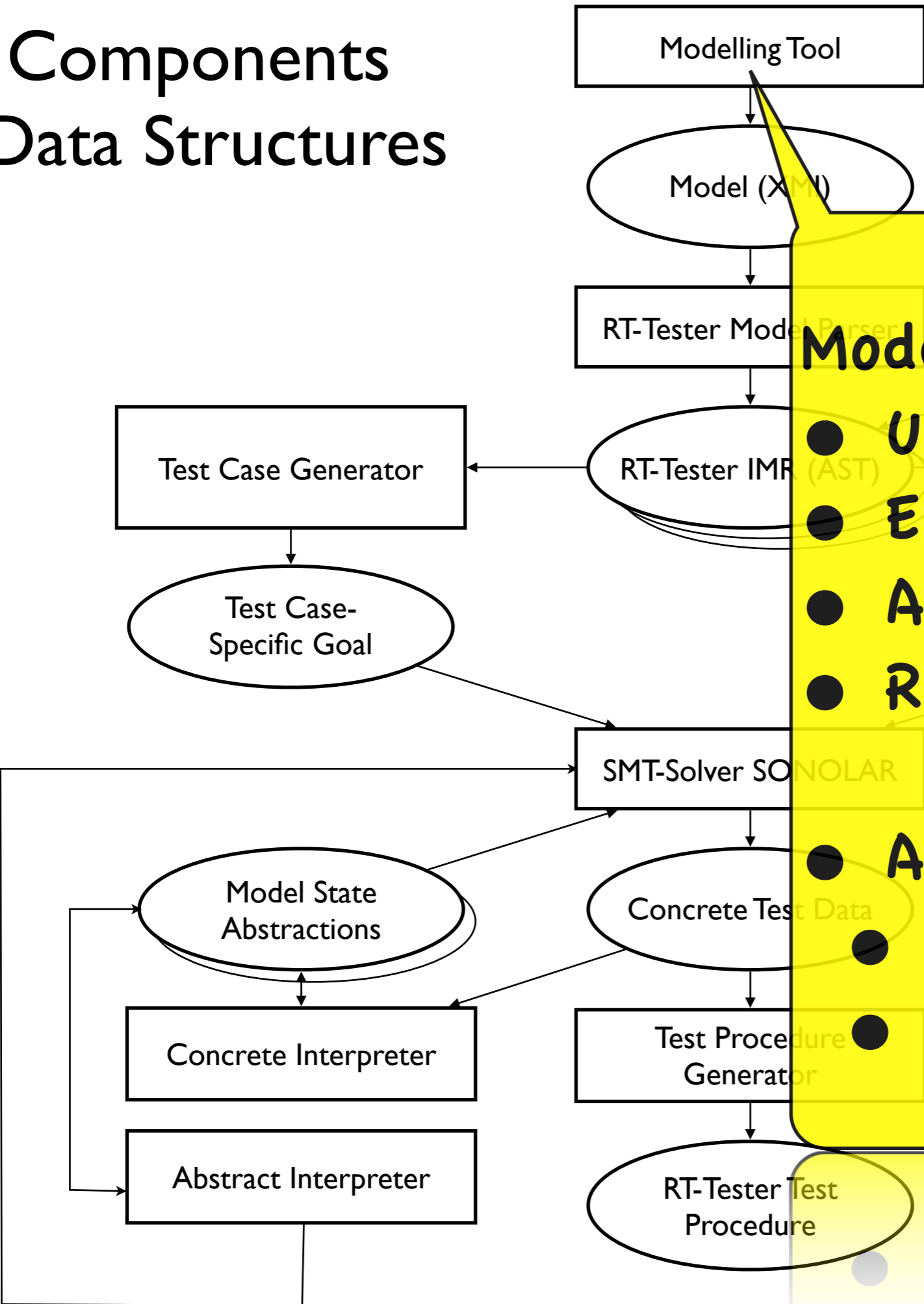
- Supports all test levels – from unit to system integration testing
- Software tests and hardware-in-the-loop tests
- Test projects may combine hand-written test procedures with automatically generated procedures

→ The tool capabilities are presented here to stimulate benchmarking activities

# Tool Components and Data Structures



# Tool Components and Data Structures

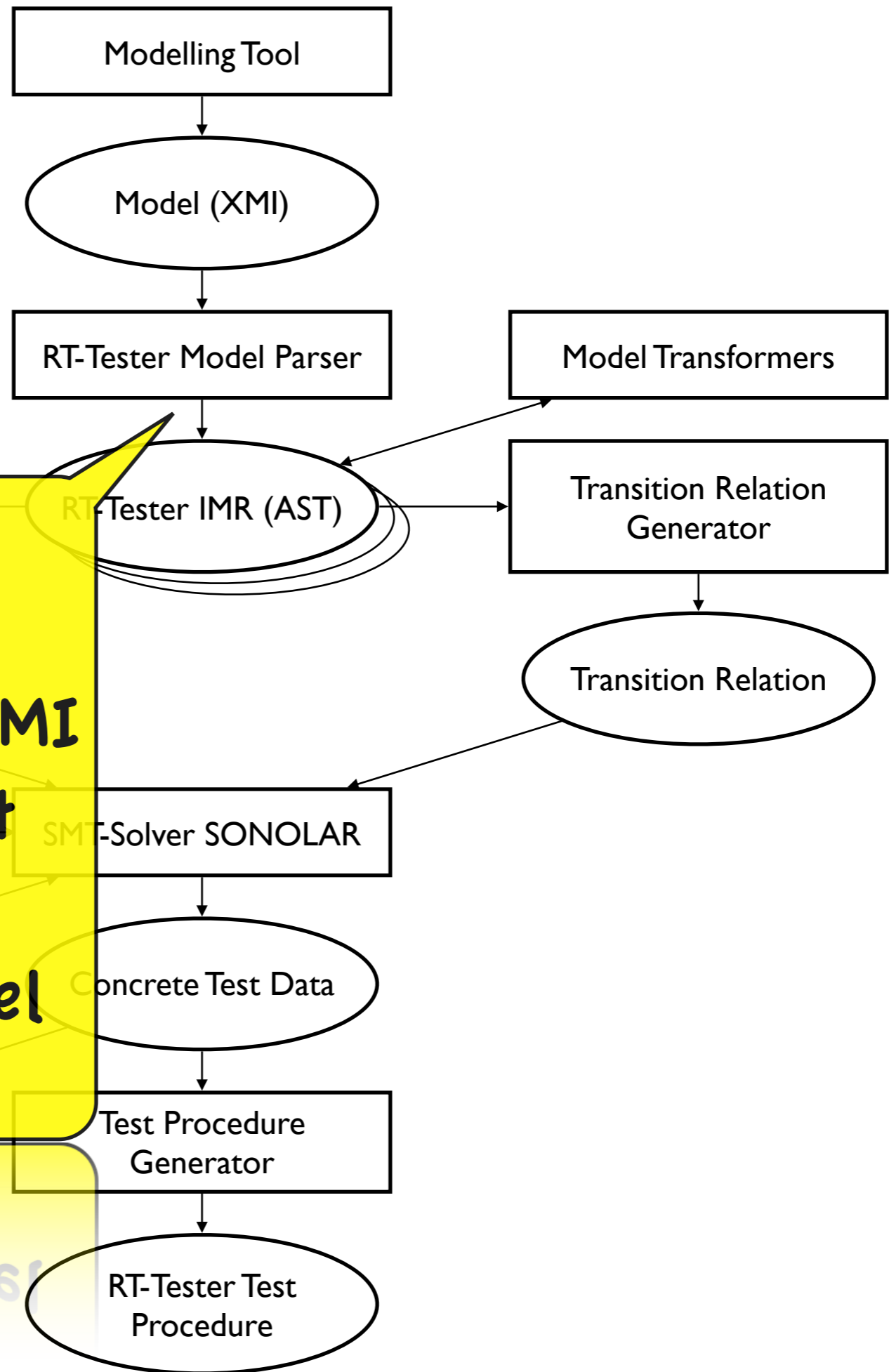


**Modelling Tool**

- UML/SysML subset
- Enterprise Architect
- Artisan Studio
- Rhapsody

**Alternatively:**

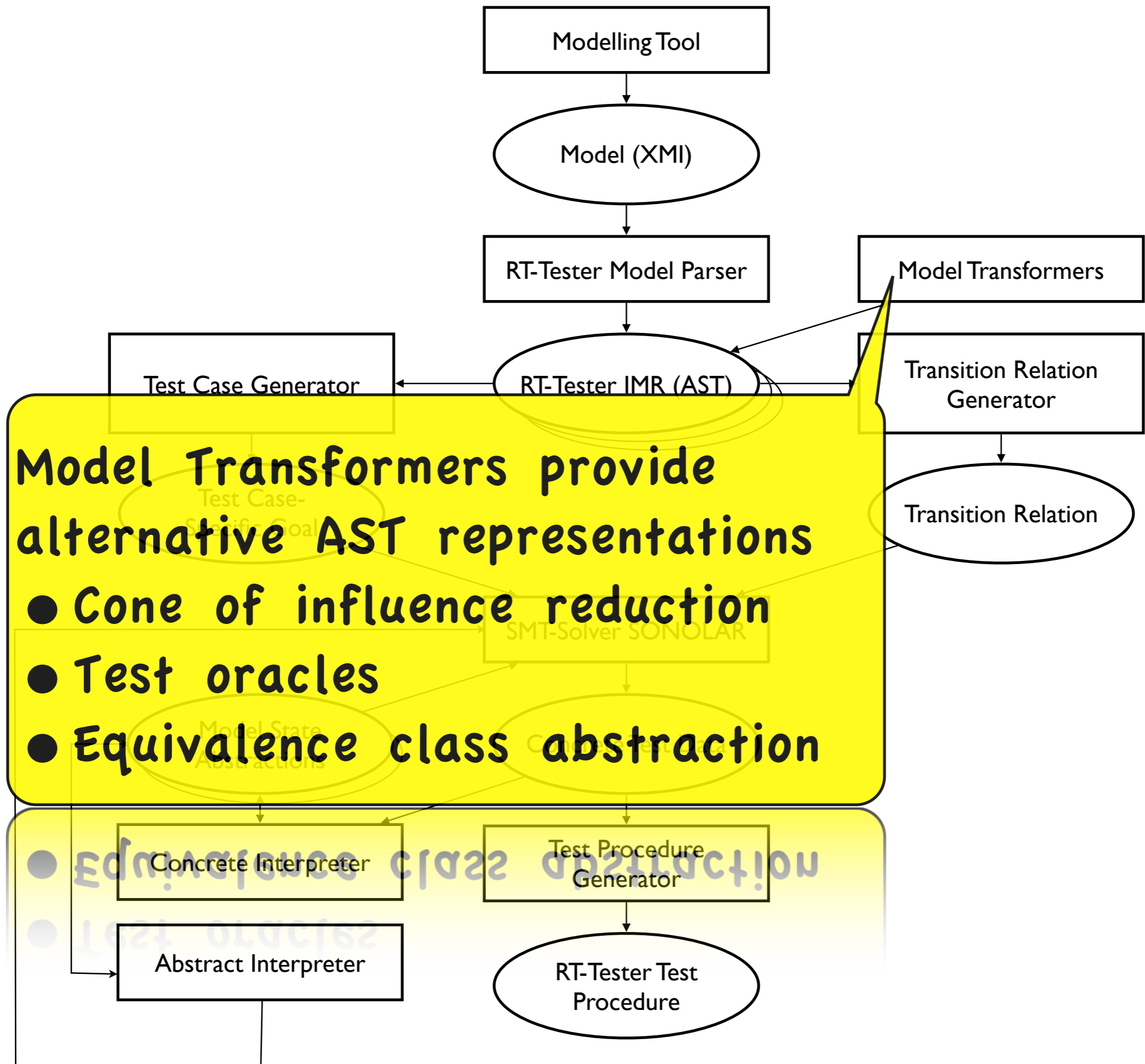
- DSL
- MetaEdit+

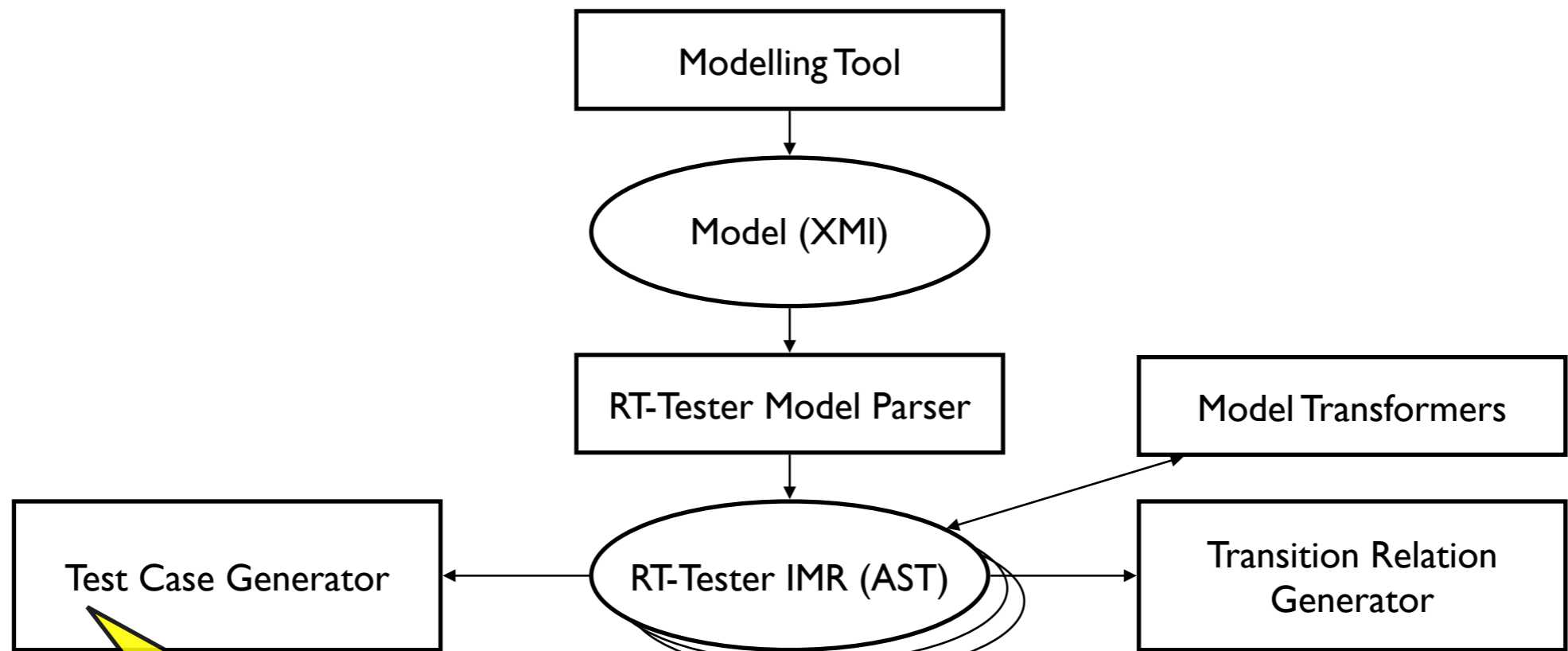


## Parser Front Ends

- transform model representations in XMI format into abstract syntax tree
- **AST = Internal Model Representation IMR**

Abstract Interpreter

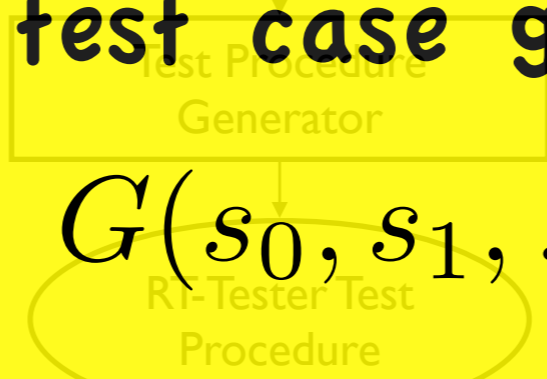
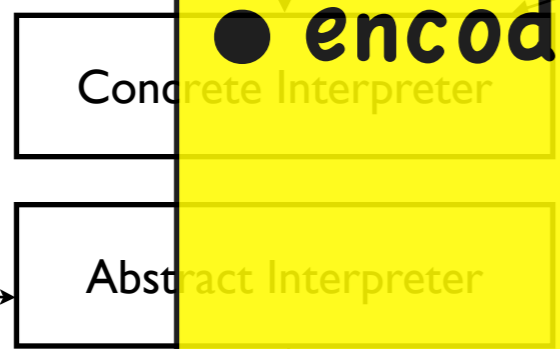




## Test Case Generator

- identifies “relevant” test cases
- uses ASTs as identification basis
- exploits traceability information from requirements to model elements
- encodes test case goals as propositions

$$G(s_0, s_1, \dots, s_c)$$



Modelling Tool

**Transition Relation Generator**

- encodes operational semantics of the model by relating pre-states to post states

$$\Phi(s, s')$$

Model Transformers

Transition Relation Generator

Transition Relation

SMT-Solver SONOLAR

Model State Abstractions

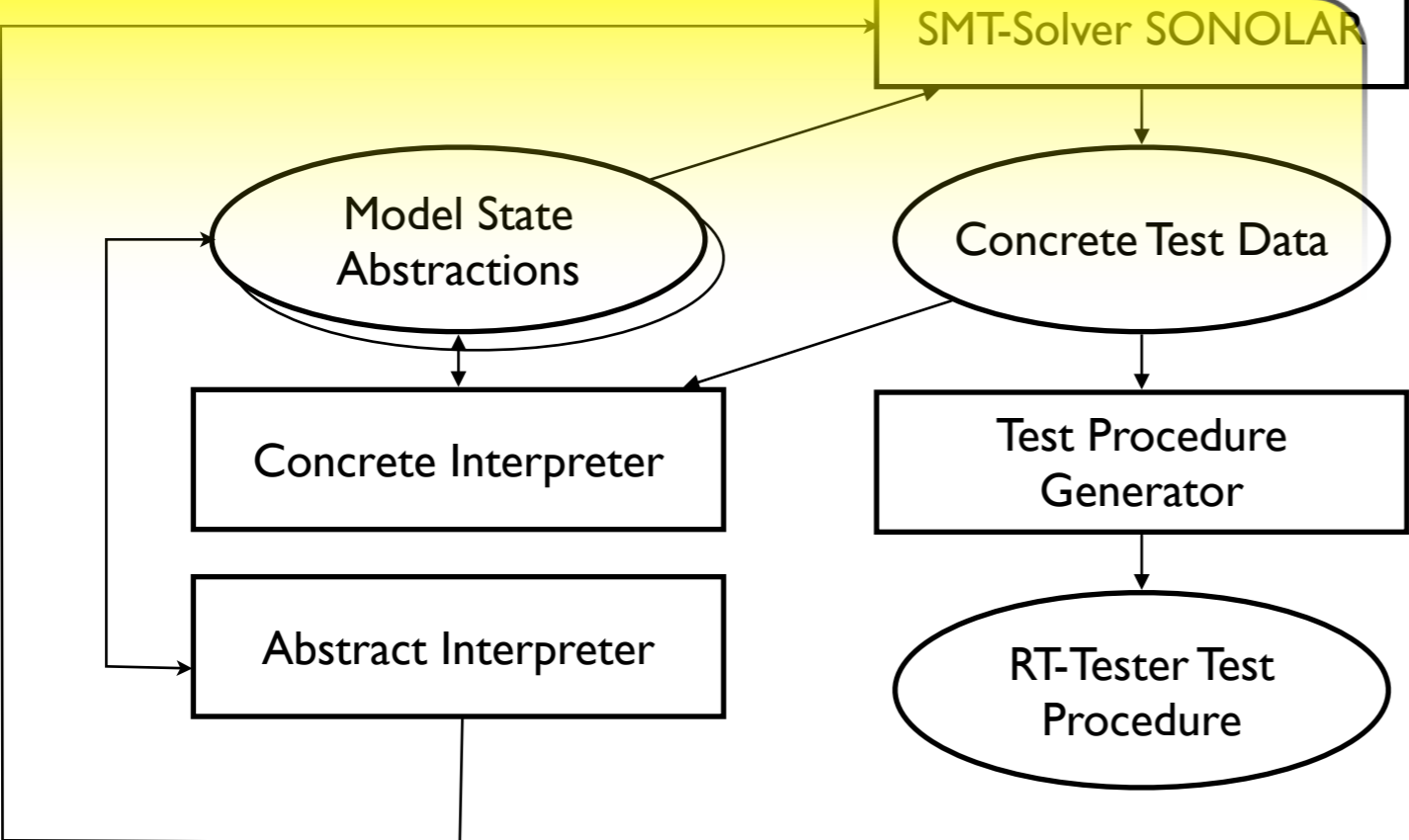
Concrete Test Data

Concrete Interpreter

Test Procedure Generator

Abstract Interpreter

RT-Tester Test Procedure





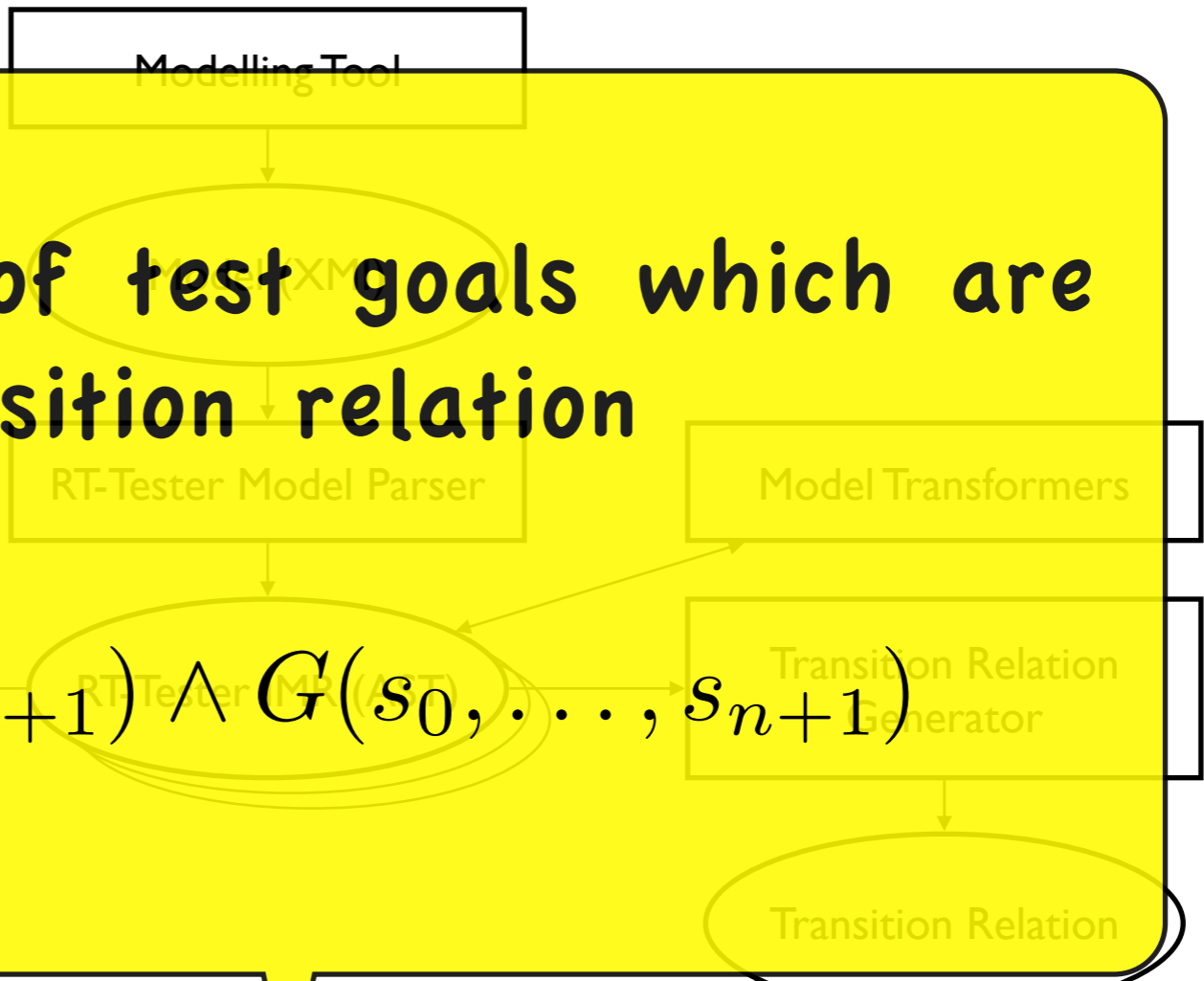
# SMT-Solver

- calculates solution of test goals which are compatible the transition relation

$$J(s_0) \wedge \bigwedge_{i=0}^n \Phi(s_i, s_{i+1}) \wedge G(s_0, \dots, s_{n+1})$$

Test Case-Specific Goal

Transition Relation



SMT-Solver SONOLAR

Model State Abstractions

Concrete Test Data

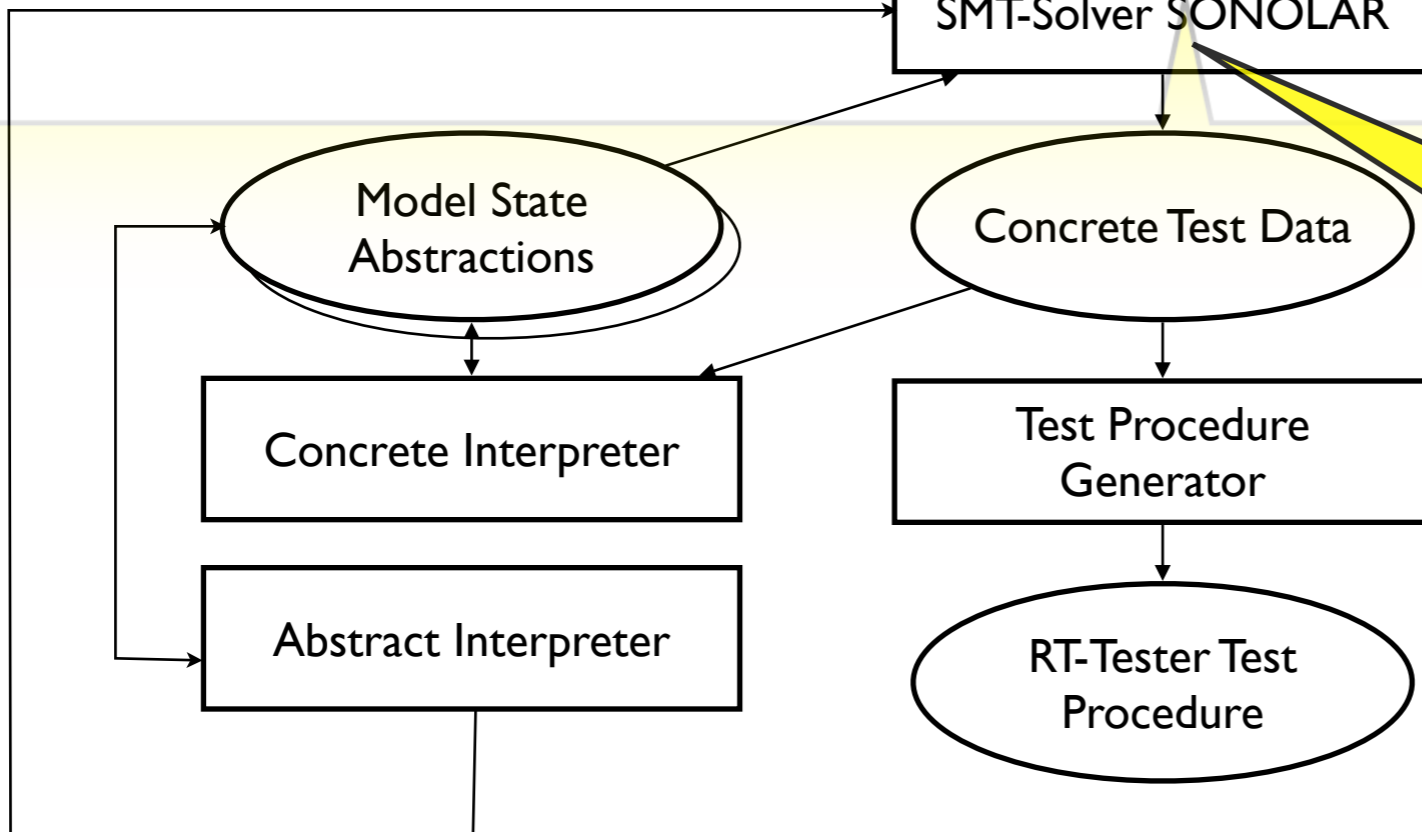
Concrete Interpreter

Test Procedure Generator

Abstract Interpreter

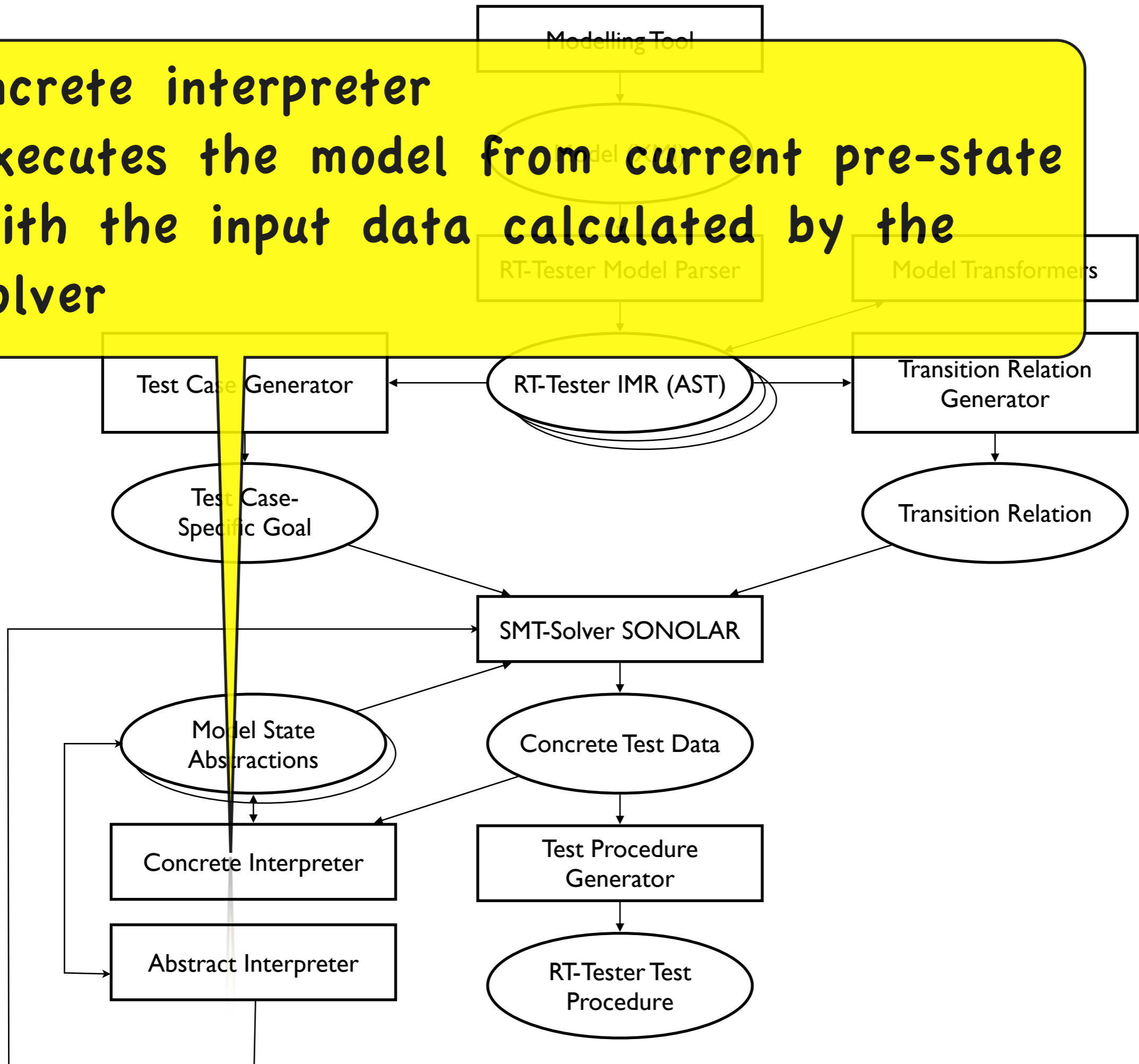
RT-Tester Test Procedure

Can handle Boolean, Integer, Float, Array data types



# Concrete interpreter

- executes the model from current pre-state with the input data calculated by the solver



Modelling Tool

## Abstract interpreter

- speeds up SMT-solver by
- calculating minimal number of steps required for finding solutions
- restricting the ranges of inputs and other model variables in traces leading to a solution of

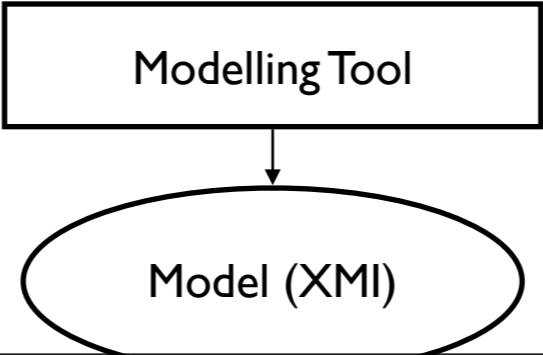
$$J(s_0) \wedge \bigwedge_{i=0}^n \Phi(s_i, s_{i+1}) \wedge G(s_0, \dots, s_{n+1})$$

Concrete Interpreter

Abstract Interpreter

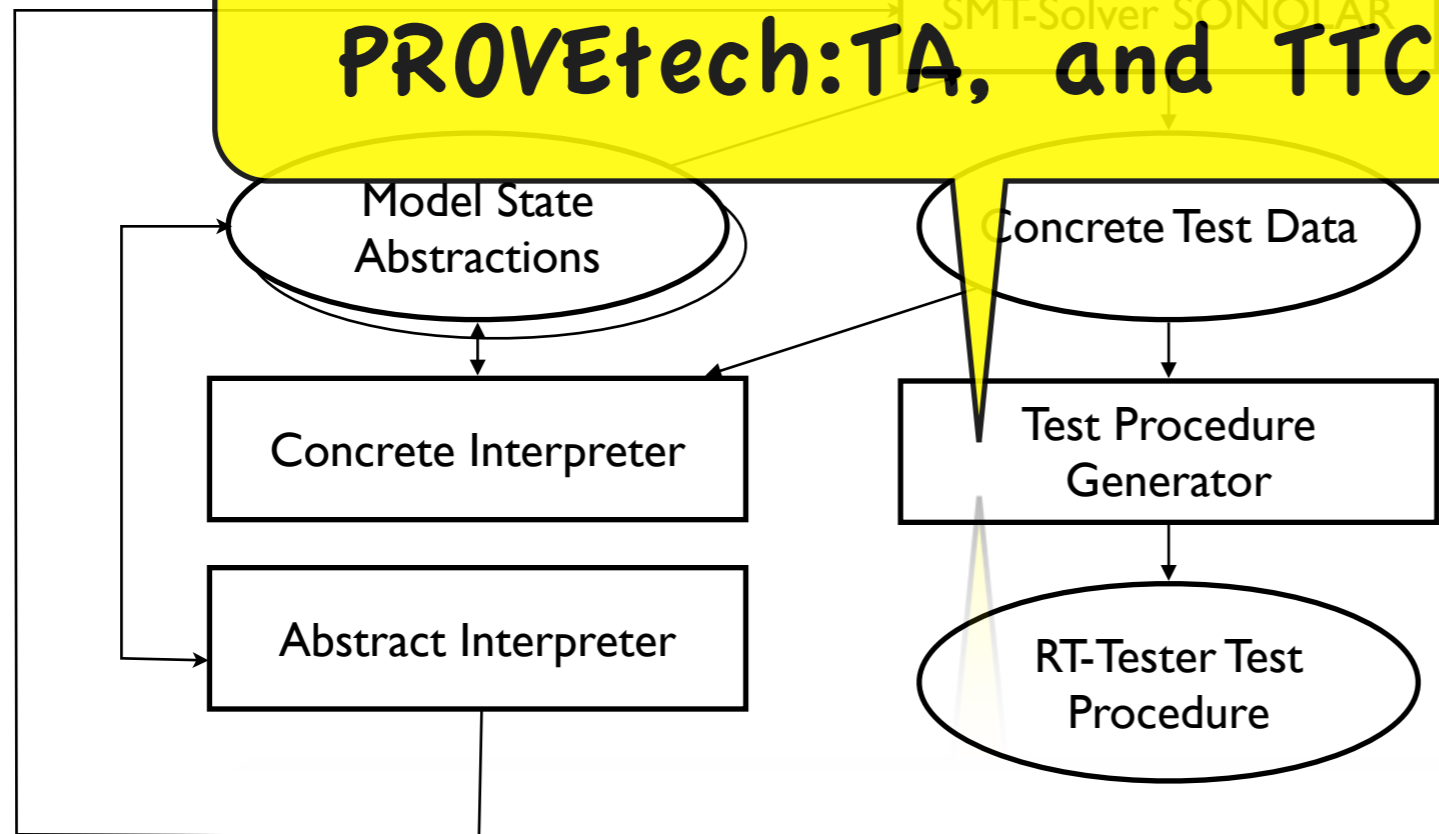
Test Procedure  
Generator

RT-Tester Test  
Procedure



## Test Procedure Generator

- is a compile back-end for transforming test case solutions to executable test procedures
- provides different compile back-ends for RT-Tester Real-Time Test Language, PROVEtech:TA, and TTCN-3



- Model-based testing
- A reference tool
- **Modelling aspects**
- Requirements, test cases and strategies
- Conclusion – challenges

# Formalisms

- The controversy about modelling formalisms is unlikely to come to an end in the foreseeable future
- Domain-specific language methodology even suggests that productivity and quality are improved, if formalisms optimised for their application domains are used

# Formalisms

## Modelling formalisms supported by RT-Tester

- Timed CSP
- CML – COMPASS Modelling Language
- Timed Moore Automata
- UML
- SysML

# Formalisms

- UML
  - Composite structure diagrams
  - Interfaces
  - Classes and operations
  - State machines with timers



# Formalisms

- SysML
  - Block definition diagrams
  - Internal block diagrams
  - Item flows
  - State machines with timers
  - Operations
  - Requirements
  - <<satisfy>> relationship between requirements and model elements

# Case Study With SysML

- Simplified version of the turn indication and emergency flashing function in Daimler vehicles
- Full model available under

`http://www.mbt-benchmarks.org`

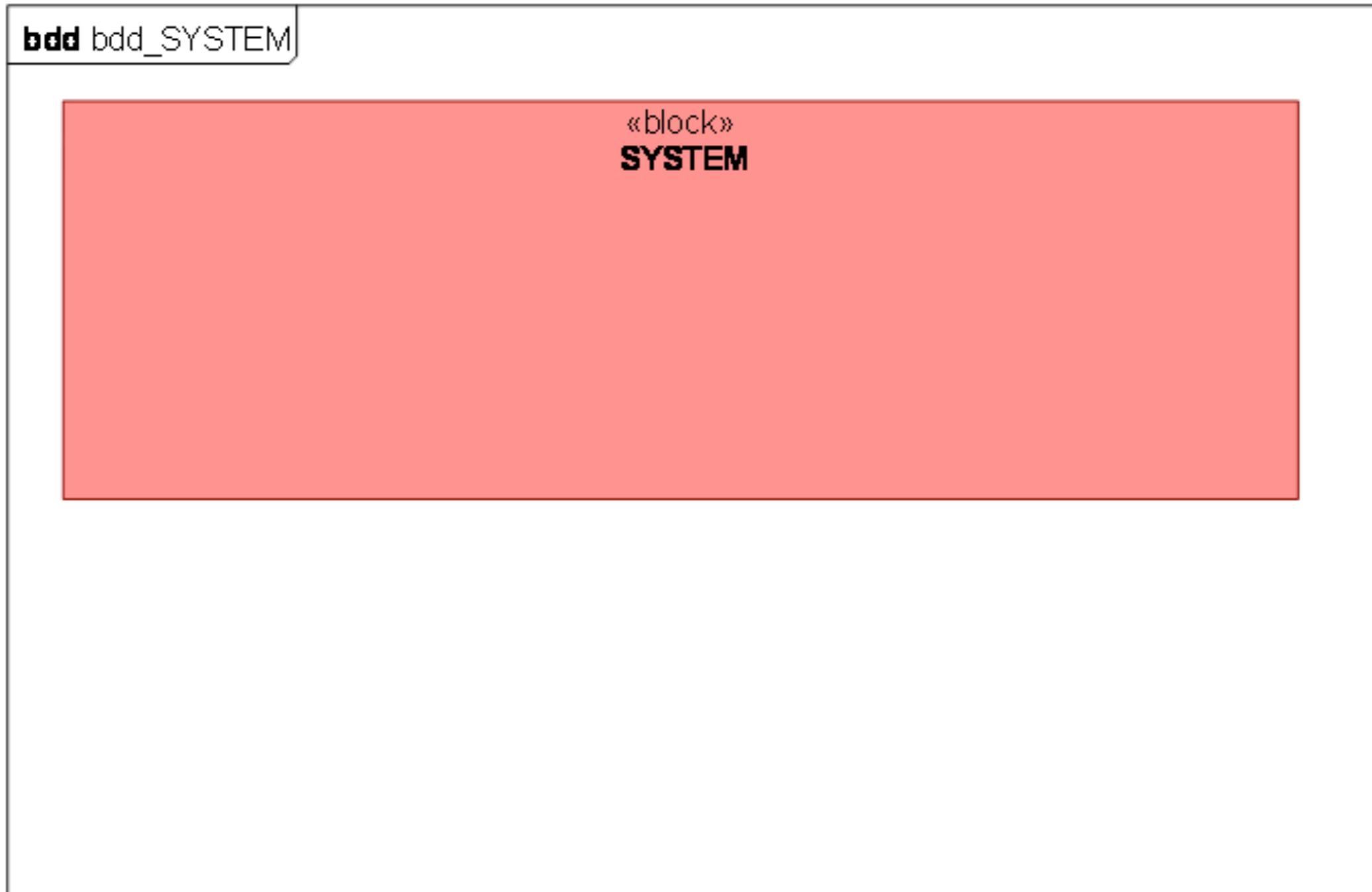
→ Benchmarks

→ Turn Indicator Model Rev. 1.4

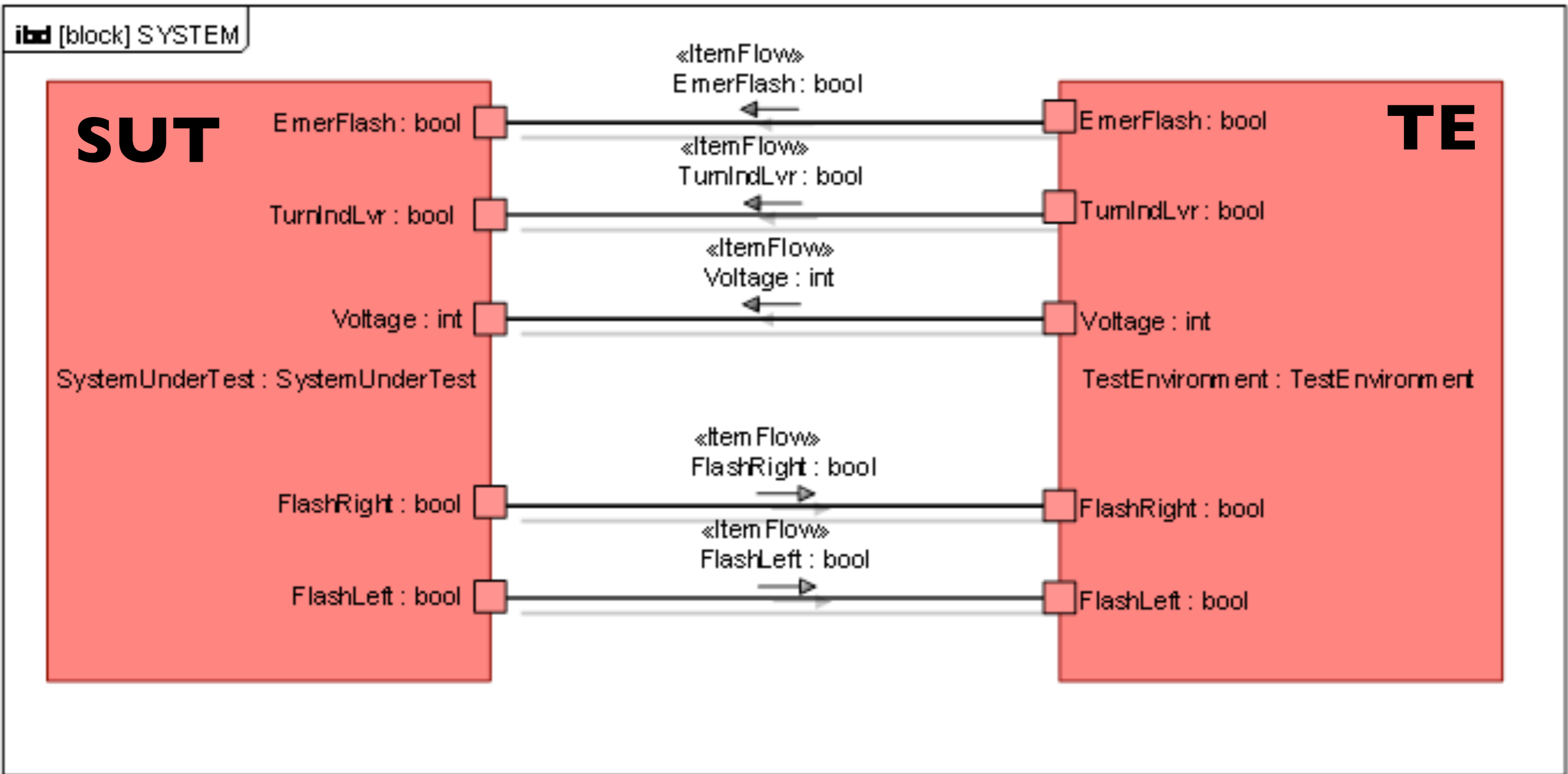
# Turn Indication Function

Requirement	Description
REQ-001	Flashing requires more than 80% of nominal input voltage
REQ-002	Flashing is performed with 340ms/320ms on-off periods
REQ-003	Turn indication lever switched to I results in left-hand side flashing
...	...

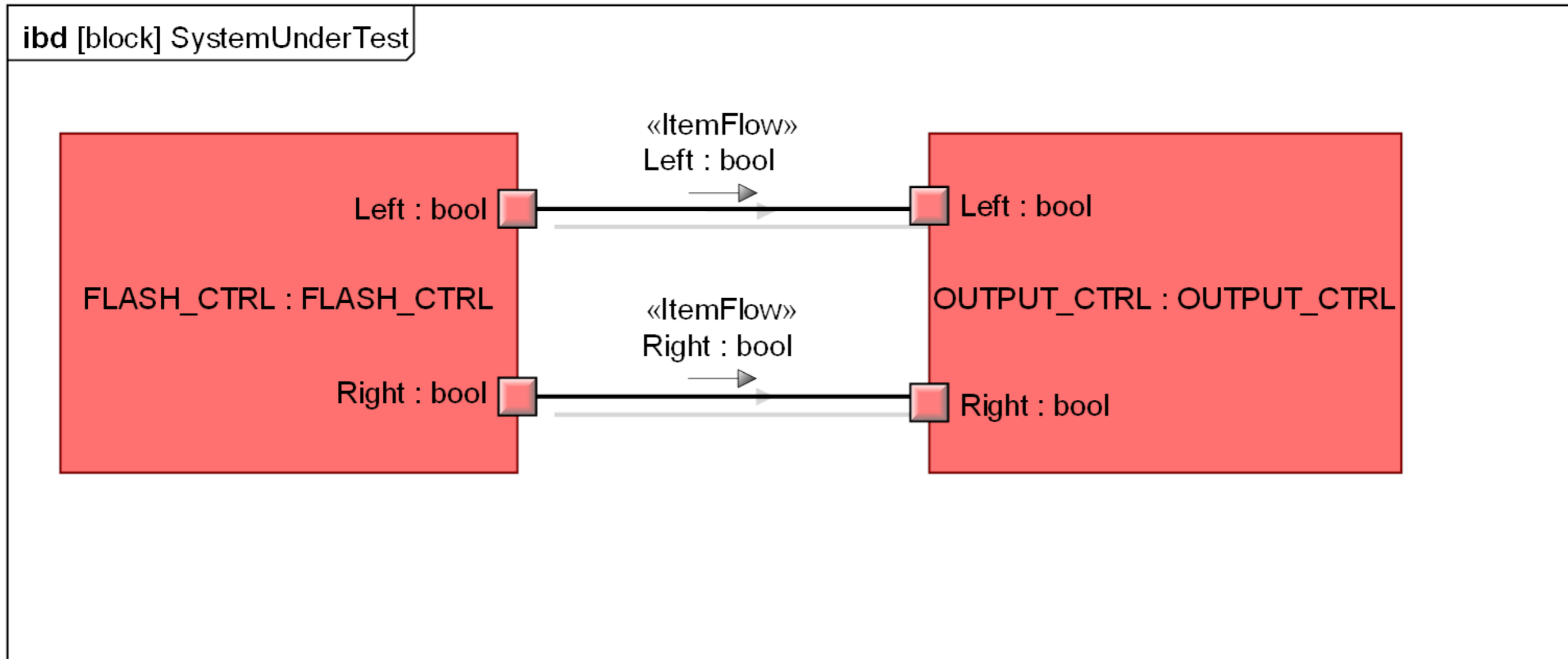
# Test Model



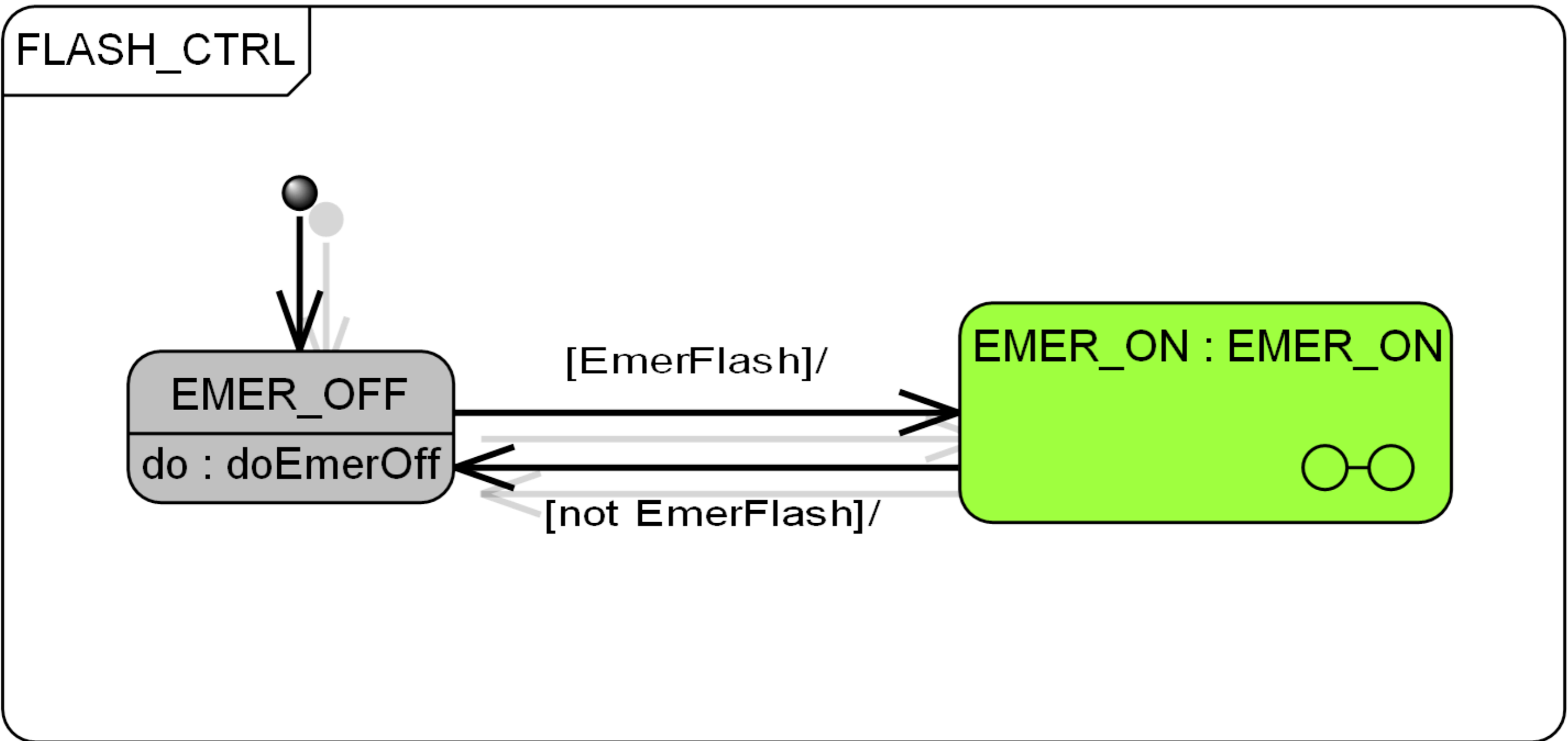
# SUT and TE

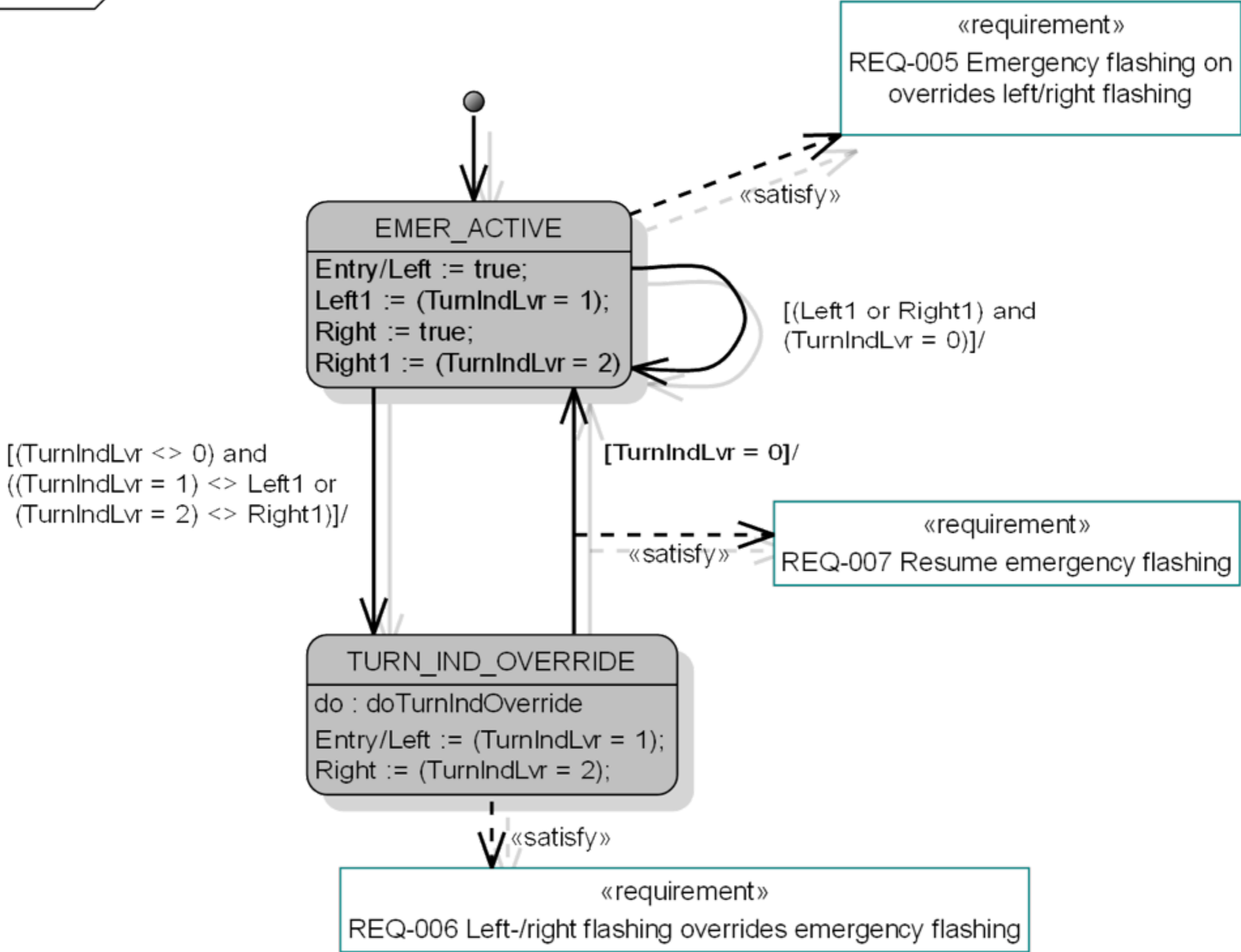


# Turn Indication Controller

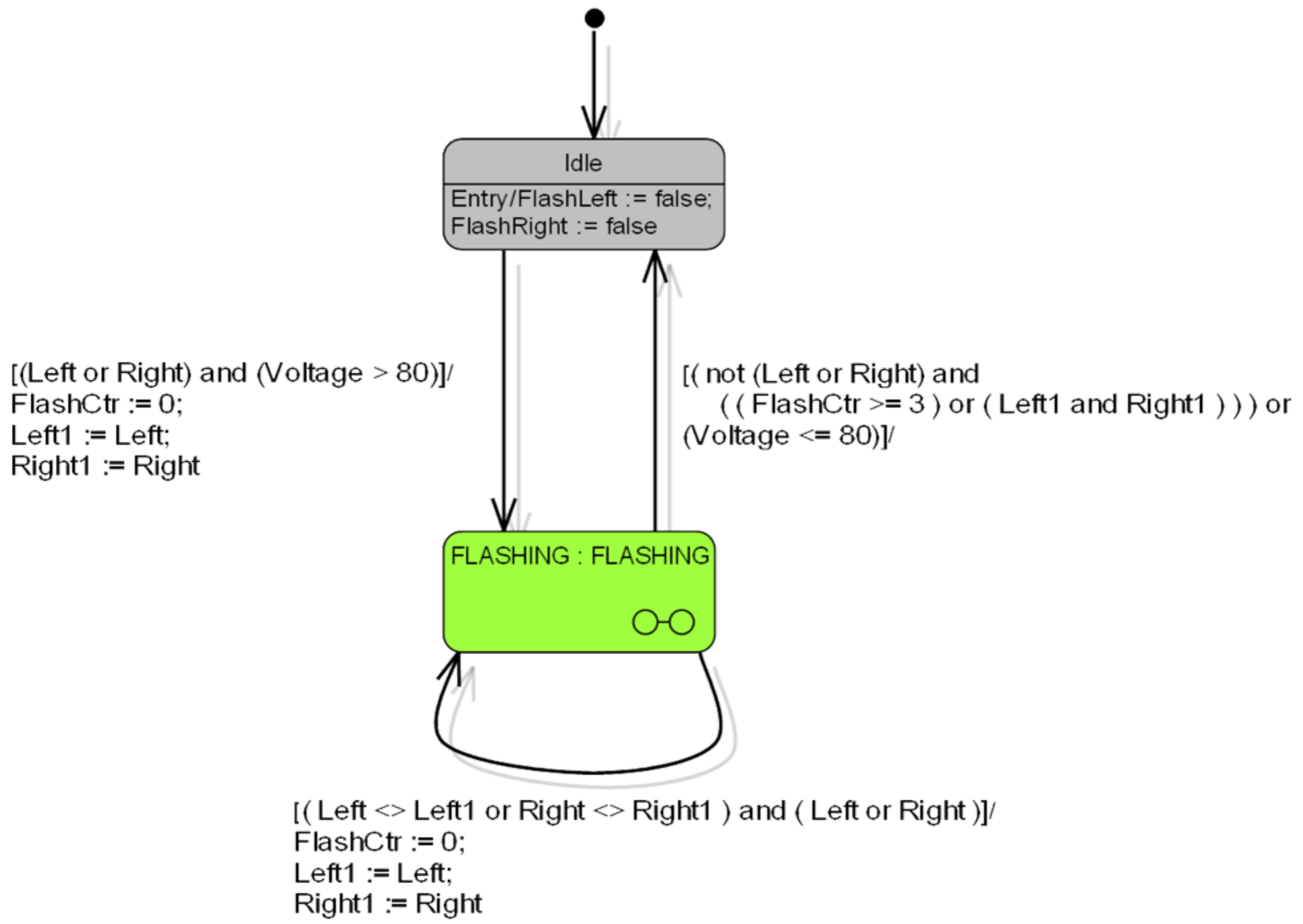


# Turn Indication Controller

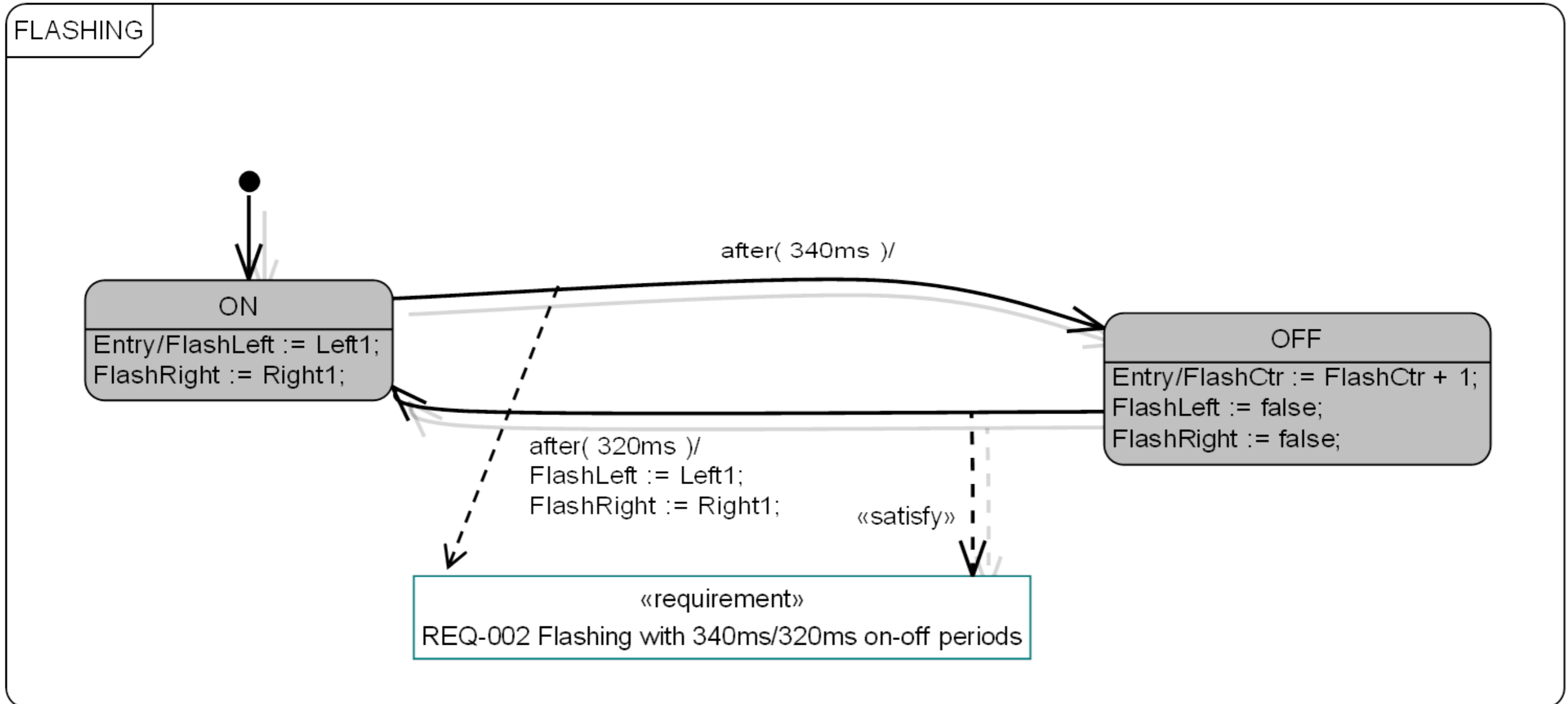








# Turn Indication Controller



# Model Semantics

- Based on Kripke Structures
- Equivalent to alternative operational semantics based on labelled transition systems

$$K = (S, S_0, R, L)$$

$S$  : State space

$S_0 \subseteq S$  : Initial states

$R \subseteq S \times S$  : Transition relation

$L : S \rightarrow 2^{AP}$  : Labelling function

$AP$  : Atomic propositions

# Conformance Relations

## Idealised conformance relation

- For any timed input trace, SUT should produce the same outputs as the model

$$\forall i \in \{0, \dots, n\} : s_i|_{IUOU\{\hat{t}\}} = s'_i|_{IUOU\{\hat{t}\}}$$

$s_0.s_1 \dots s_n$  : Model trace

$s'_0.s'_1 \dots s'_n$  : SUT trace

# Conformance Relations

Idealised conformance relation is justified when

- Interfaces are non-blocking
- Most-recent values of SUT outputs are always available
- Each sequential SUT component is deterministic
- Synchronous concurrency semantics applies
- Application in RT-Tester: testing SCADE software

# Conformance Relations

Conformance relations in presence of non-determinism – required for

- asynchronous distributed control systems
- in presence of SUT outputs behaving non-deterministically over certain periods of time – often due to under-specification

# Conformance Relations

## Non-determinism as handled in RT-Tester

- Admissible output deviations

$$|s'(y) - s(y)| \leq \varepsilon_y$$

- Admissible output latency

$$s'(\hat{t}) - s(\hat{t}) \leq \delta_y^0$$

- Admissible early changes

$$s(\hat{t}) - s'(\hat{t}) \leq \delta_y^1$$

- Time-bounded non-deterministic assignment

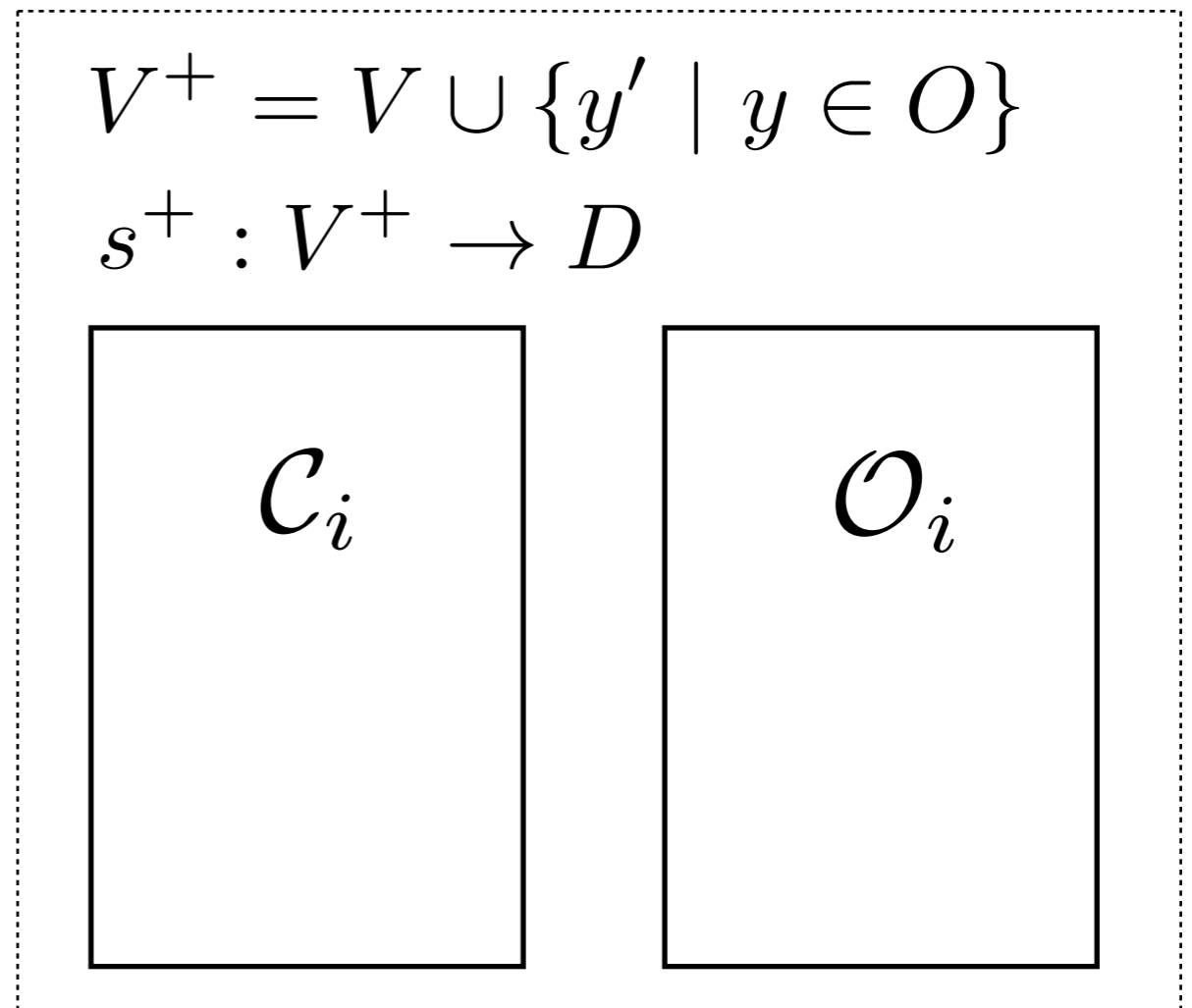
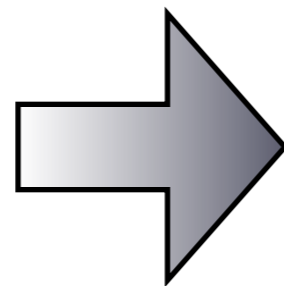
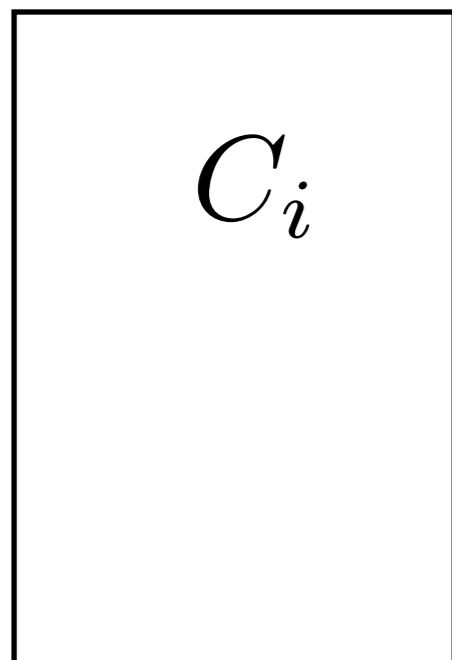
$$y = \text{UNDEF}(t, c);$$

- Model transformation  
SUT  $\rightarrow$  Test oracle

# Model Transformation for Test Oracles

Model component with associated state machine ...

Transformed into modified state machine and test oracle

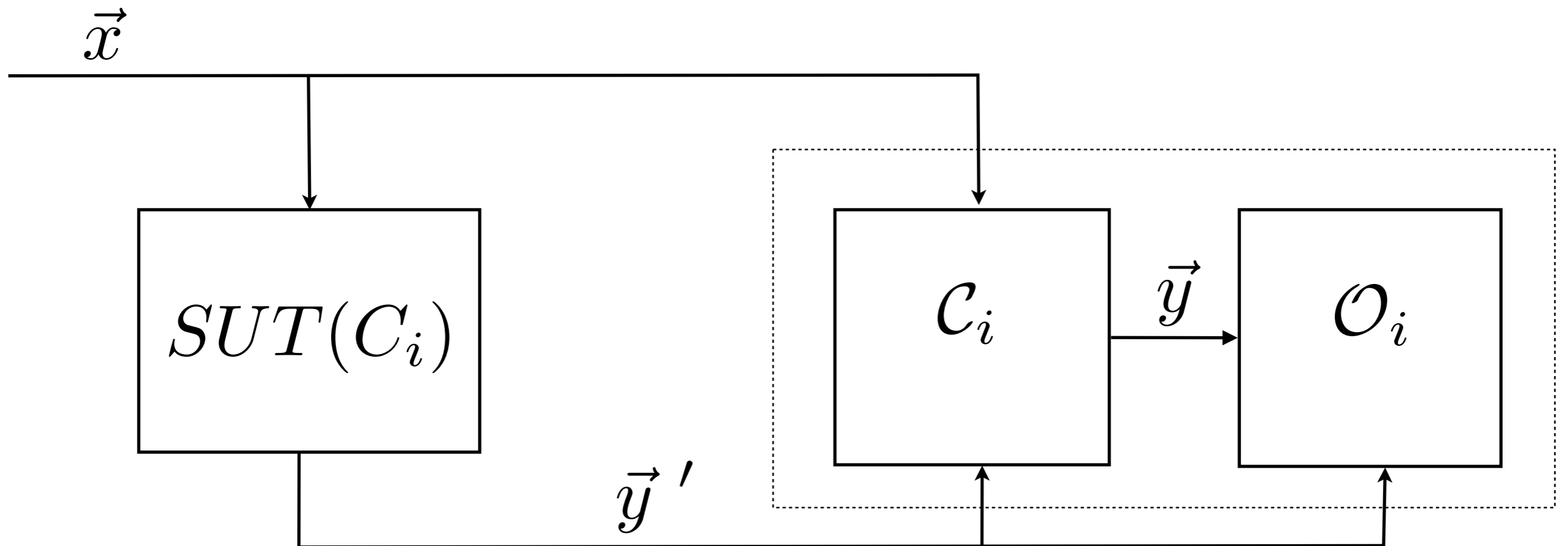




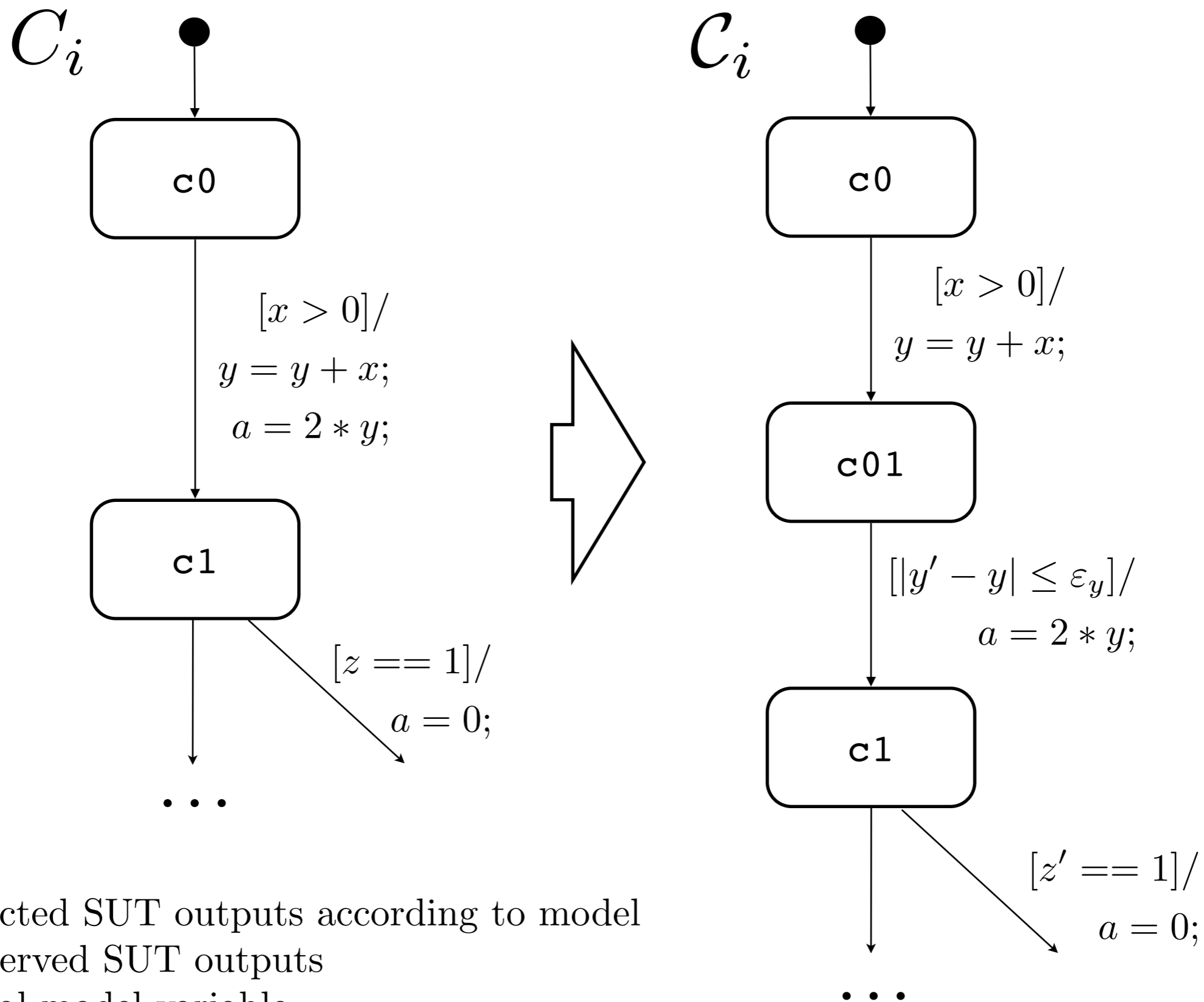
# Model Transformation for Test Oracles

SUT component with associated state machine

Transformed model consisting of transformed state machine and oracle



# Model Transformation for Test Oracles



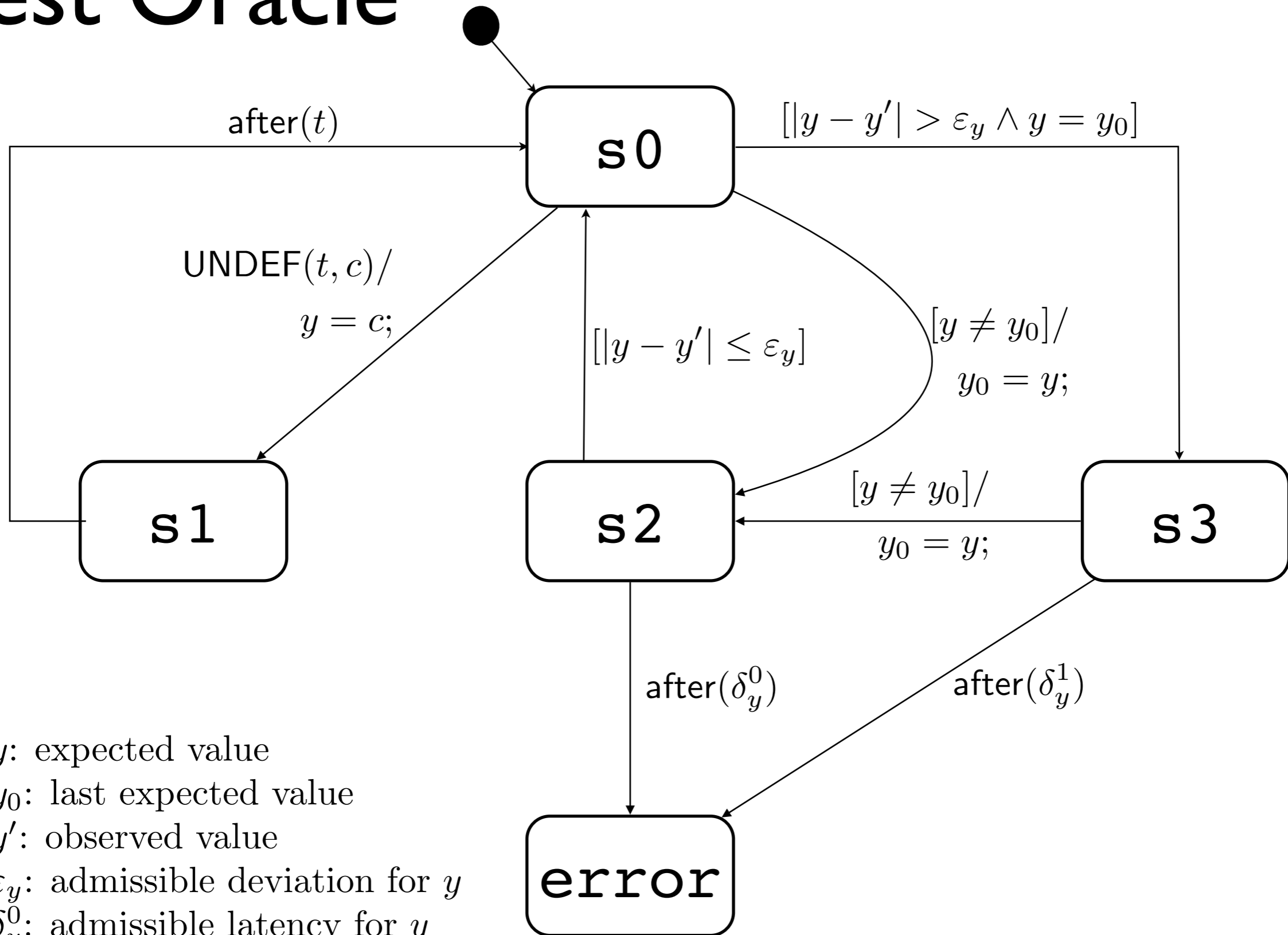
$x$  : input

$y, z$ : expected SUT outputs according to model

$y', z'$ : observed SUT outputs

$a$  : internal model variable

# Test Oracle



$y$ : expected value

$y_0$ : last expected value

$y'$ : observed value

$\varepsilon_y$ : admissible deviation for  $y$

$\delta_y^0$ : admissible latency for  $y$

$\delta_y^1$ : admissible time for early changes of  $y'$

$\delta_y^1 < \delta_y^0$

# Conformance Relation

- For a given input sequence and resulting SUT I/O trace, the transformed system should never assume an `error` state in any of its test oracles.

$$\begin{aligned} & \forall s'_0 \dots s'_n, s_0^+ \dots s_n^+ : (\forall i = 0, \dots, n, y \in O : \\ & \quad s'_i|_{I \cup \{\hat{t}\}} = s_i^+|_{I \cup \{\hat{t}\}} \wedge s'_i(y) = s_i^+(y')) \Rightarrow \\ & (\forall i = 0, \dots, n, j = 1, \dots, k : \neg s_i^+(\mathcal{O}_j.\text{error})) \end{aligned}$$

- Model-based testing
- A reference tool
- Modelling aspects
- **Requirements, test cases and strategies**
- Conclusion – challenges

# Requirements

- Each requirement is reflected by set of model computations

$$\pi = s_0.s_1.s_2 \dots$$

- Computation sets can be characterised by Linear Temporal Logic (LTL)

$\mathbf{G}\phi$  : Globally  $\phi$  holds on path  $\pi$

$\mathbf{X}\phi$  : In the next state on path  $\pi$ , formula  $\phi$  holds.

$\mathbf{F}\phi$  : Finally  $\phi$  holds on path  $\pi$

$\phi\mathbf{U}\psi$  :  $\mathbf{F}\psi$  and  $\phi$  holds on path  $\pi$  until  $\psi$  is fulfilled

# Requirements – LTL Examples

- REQ-001. Flashing requires sufficient voltage

$$\mathbf{G}(\text{Voltage} \leq 80 \Rightarrow \mathbf{X}(\neg(\text{FlashLeft} \vee \text{FlashRight}) \mathbf{U} \text{Voltage} > 80))$$

- Reduced to model computations

$$\mathbf{G}(\text{Voltage} \leq 80 \Rightarrow \mathbf{X}(\text{Idle} \mathbf{U} \text{Voltage} > 80))$$

- Finally 
$$\mathbf{F}(\text{Voltage} \leq 80)$$

# Requirements – LTL Examples

- Requirements specification is simplified by referring to internal model symbols
- REQ-002. Flashing with 340/320ms on-off-periods

$$\mathbf{F}(\mathbf{OFF} \wedge \mathbf{X} \mathbf{ON})$$



# Requirements Tracing to Model Elements

- Simple requirements tracing: every computation finally covering one model element of a given collection contributes to the requirement

$\mathbf{F}\langle\text{State Formula}\rangle$

- Simple requirements are reflected by formulas satisfying

$$\mathbf{F} \left( \bigvee_{i=0}^h \phi_i \right)$$

# Requirements Tracing – Complex Requirements

- Computations contributing to complex requirements require full LTL expressions
- Insert LTL formula in constraint
- Link constraint to requirement via `<<satisfy>>` relation

<b>Requirement</b>	<b>Constraint</b>
REQ-001 Flashing requires sufficient voltage	$\mathbf{F}(\text{Voltage} \leq 80)$
REQ-002 Flashing with 340ms/320ms on-off periods	$\mathbf{F}(\text{OFF} \wedge \mathbf{XON})$
REQ-003 Switch on turn indication left	$\mathbf{F}(\text{FlashLeft} = 1 \wedge \text{FlashRight} = 0)$
REQ-004 Switch on turn indication right	$\mathbf{F}(\text{FlashLeft} = 0 \wedge \text{FlashRight} = 1)$
REQ-005 Emergency flashing on overrides left/right flashing	$\mathbf{F}(\text{EMER\_OFF} \wedge \text{TurnIndLvr} > 0 \wedge \text{EmerFlash})$
REQ-006 Left-/right flashing overrides emergency flashing	$\mathbf{F} \text{ TURN\_IND\_OVERRIDE}$
REQ-007 Resume emergency flashing	$\mathbf{F}(\text{TURN\_IND\_OVERRIDE} \wedge \mathbf{XEMER\_ACTIVE})$
REQ-008 Resume turn indication flashing	$\mathbf{F}(\text{EMER\_ACTIVE} \wedge \neg \text{EmerFlash} \wedge \text{TurnIndLvr} > 0)$
REQ-009 Tip flashing	$\mathbf{F}(\text{Voltage} > 80 \wedge \neg(\text{Left} \vee \text{Right}) \wedge \text{Left1} + \text{Right1} = 1 \wedge \text{FlashCtr} < 3)$

# Test Cases

- Test cases are finite witnesses of model computations
- Trace = finite prefix of a computation
- If computation satisfies LTL formula associated with a requirement, trace prefixes must at least not violate this formula
- Some formulas can only be verified on an infinite computation (liveness formulas, e.g. fairness properties)
- But these properties can only be partially verified by testing

# Trace Semantics for LTL Formulas

$\langle \varphi \rangle_i^k$  states that formula  $\varphi$  holds in trace segment  $s_i \cdot s_{i+1} \dots s_k$  of a trace  $s_0 \dots s_k$

- $\langle \mathbf{G} \varphi \rangle_0^k = \bigwedge_{i=0}^k \langle \varphi \rangle_i^k$
- $\langle \mathbf{X} \varphi \rangle_i^k = \langle \varphi \rangle_{i+1}^k$
- $\langle \varphi \mathbf{U} \psi \rangle_i^k = \langle \psi \rangle_i^k \vee (\langle \varphi \rangle_i^k \wedge \langle \varphi \mathbf{U} \psi \rangle_{i+1}^k)$
- $\langle \mathbf{F} \psi \rangle_i^k = \langle \text{true} \mathbf{U} \psi \rangle_i^k$

# Test Data Computation

- LTL formulas interpreted on finite traces can be transformed into first order expressions

$$tc \equiv J(s_0) \wedge \bigwedge_{i=0}^n \Phi(s_i, s_{i+1}) \wedge G(s_0, \dots, s_{n+1})$$

- Recall. These formulas can be solved by an SMT solver

# Model Coverage Strategies

Strategies currently realised in RT-Tester

- Basic control state coverage
- Transition coverage
- MC/DC coverage
- Hierarchic transition coverage
- Equivalence class and boundary value coverage
- Basic control state pairs coverage
- Interface coverage
- Block coverage

# Model Coverage Strategies

- Example. Hierarchic transition coverage for state machine FLASH\_CTRL

$$tc_1 \equiv \mathbf{F}(\text{EMER\_OFF} \wedge \text{EmerFlash})$$

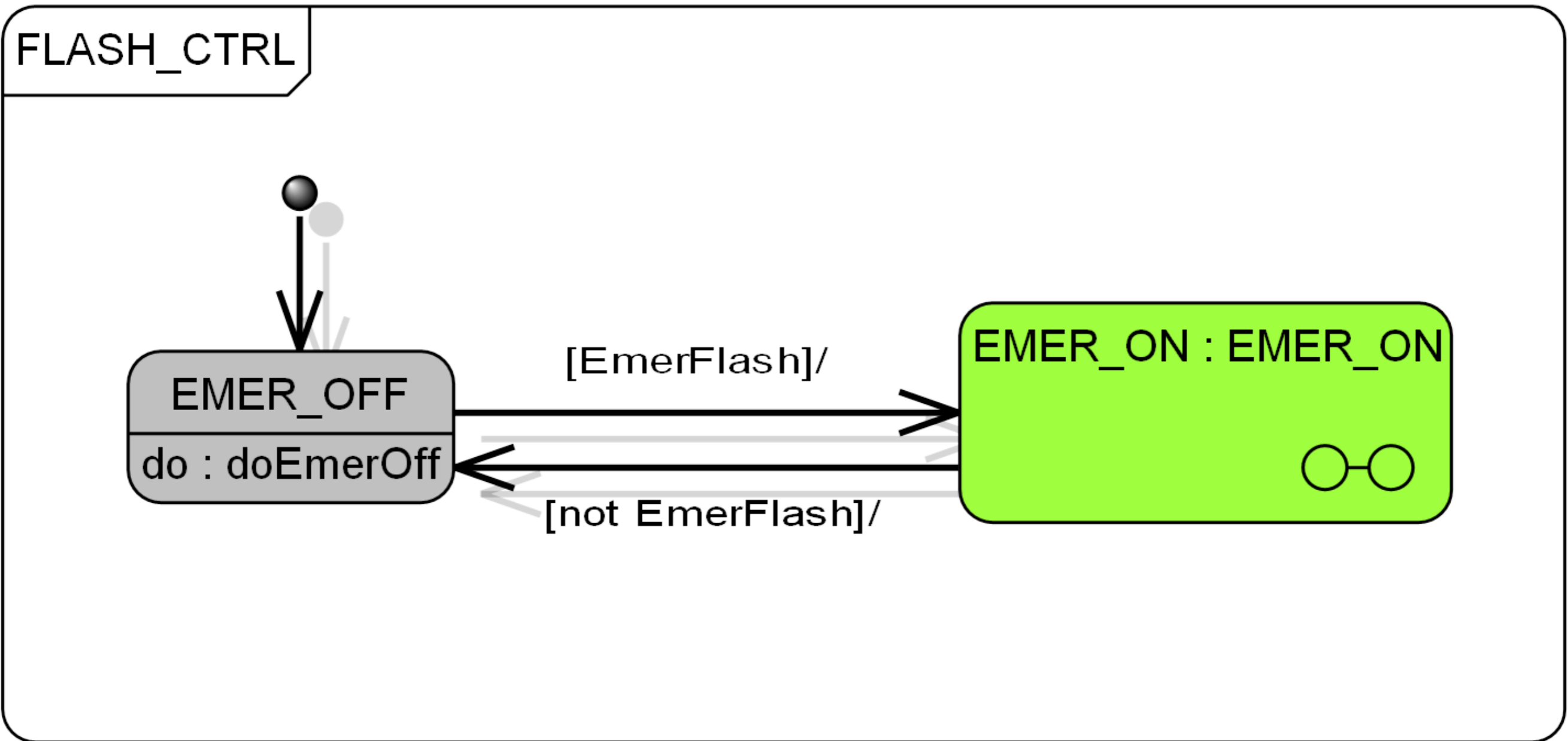
$$tc_2 \equiv \mathbf{F}(\text{EMER\_ACTIVE} \wedge \text{TurnIndLvr} \neq 0 \wedge \\ ((\text{TurnIndLvr} = 1) \neq \text{Left1} \vee \\ (\text{TurnIndLvr} = 2) \neq \text{Right1}))$$

...

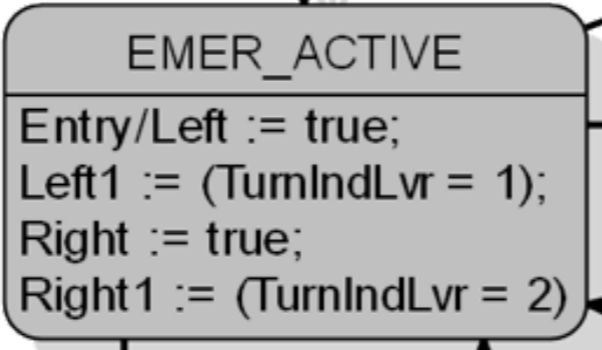
$$tc_6 \equiv \mathbf{F}(\neg \text{EmerFlash} \wedge \text{TURN\_IND\_OVERRIDE} \wedge \\ \text{TurnIndLvr} \neq 0)$$



# Turn Indication Controller



$[(\text{TurnIndLvr} \neq 0) \text{ and } ((\text{TurnIndLvr} = 1) \neq \text{Left1} \text{ or } (\text{TurnIndLvr} = 2) \neq \text{Right1})]$



«requirement»  
REQ-005 Emergency flashing on overrides left/right flashing

$[(\text{Left1} \text{ or } \text{Right1}) \text{ and } (\text{TurnIndLvr} = 0)]/$

«requirement»  
REQ-007 Resume emergency flashing

«requirement»  
REQ-006 Left-/right flashing overrides emergency flashing

# Requirements Tracing

- If some model elements are linked to requirement  $R$  via  $\ll\text{satisfy}\gg$  relationship, then model coverage test cases  $tc$  covering these elements are automatically traced to  $R$ :
- $tc \ll\text{verify}\gg R$

# Requirements Tracing

If requirement R is characterised by complex LTL formula  $\phi$ , proceed as follows

- Transform  $\phi$  into some disjunctive form  $\phi \equiv \bigvee_{i=0}^m \phi_i$
- **For each**  $\phi_i$  associate test cases separately:
  - If  $\psi \Rightarrow \phi_i$  and  $(tc \equiv \psi)$ , add  $(tc \equiv \psi) \ll \text{verify} \gg R$
  - If  $\psi \not\Rightarrow \phi_i$  and  $\phi_i \not\Rightarrow \psi$ , but  $\psi \wedge \phi_i$  has solution, add new test case  $(tc' \equiv \psi \wedge \phi_i) \ll \text{verify} \gg R$ .
  - If  $((tc_1 \equiv \mathbf{F}\psi_1) \ll \text{verify} \gg R$  or  $(tc_2 \equiv \mathbf{F}\psi_2) \ll \text{verify} \gg R)$  and  $tc' \equiv \mathbf{F}(\psi_1 \wedge \psi_2)$  has a solution, add  $tc' \ll \text{verify} \gg R$ .

# Requirements Tracing

**Example.** Refined test cases for REQ-002  
(Flashing with 340/320ms on-off period)

$$tc_7 \equiv \mathbf{F}(\text{OFF} \wedge (\mathbf{XON}))$$

$$tc_8 \equiv \mathbf{F}(\text{OFF} \wedge (\mathbf{XON}) \wedge \text{TurnIndLvr} = 1)$$

$$tc_9 \equiv \mathbf{F}(\text{OFF} \wedge (\mathbf{XON}) \geq 320 \wedge \text{TurnIndLvr} = 2)$$

$$tc_{10} \equiv \mathbf{F}(\text{OFF} \wedge (\mathbf{XON}) \geq 320 \wedge \text{EMER\_ACTIVE})$$

$$tc_{11} \equiv \mathbf{F}(\text{OFF} \wedge (\mathbf{XON}) \wedge \text{TURN\_IND\_OVERRIDE})$$

...




**Combinatorial explosion problem**

# Test Case Reduction

- Reduction is inevitable for real-world systems
- Reduction should be justified
- Justification should conform to V&V standards, such as
  - RTCA DO-178C
  - CENELEC EN 50128:2011
  - ISO 26262

# Test Case Reduction

## **Option 1.** No further test cases when

- all requirements have been covered by at least one test case
- code coverage required by the standard has been achieved
-  This option is appropriate for RTCA DO-178C, if code coverage measurement is possible

# Test Case Reduction




**Option 2.** Test case selection according to assurance level (= criticality)

- Level 3: interface tests, basic control state coverage
- Level 2: + transition coverage
- Level 1: + basic control state pairs coverage, hierarchic transition coverage, MC/DC coverage, first-level test case refinements as introduced above, second-level refinements if new conjuncts have impact on the requirement



- Model-based testing
- A reference tool
- Modelling aspects
- Requirements, test cases and strategies
- **Conclusion – Challenges**

# Challenges – Modelling


- Testing must not be delayed by modelling
  -  Incremental modelling and learning from concrete executions
- Complexity
  -  Abstraction, equivalence class partitioning
- Test model development requires higher skills than test script programming
  -  Management issue: need fewer engineers with higher competence

# Challenges – Test Cases / Strategies

Coping with state space complexity in Systems of Systems (SoS)

- Associate mission threads of constituent systems with equivalence classes
- On SoS level, identify “relevant” class combinations by means of impact analysis

# Challenges – SoS-Specific

- Dynamic changes of system configuration  run-time acceptance testing required
- Under-specification and non-determinism due to abstractions in contracts
- Justification of test strategies by proof of exhaustiveness: still possible on this level?

# Contributors ...



# Contributors ...



**Acknowledgements.** *This presentation has been elaborated in the context of the EU FP7 COMPASS project under grant agreement no.287829.*