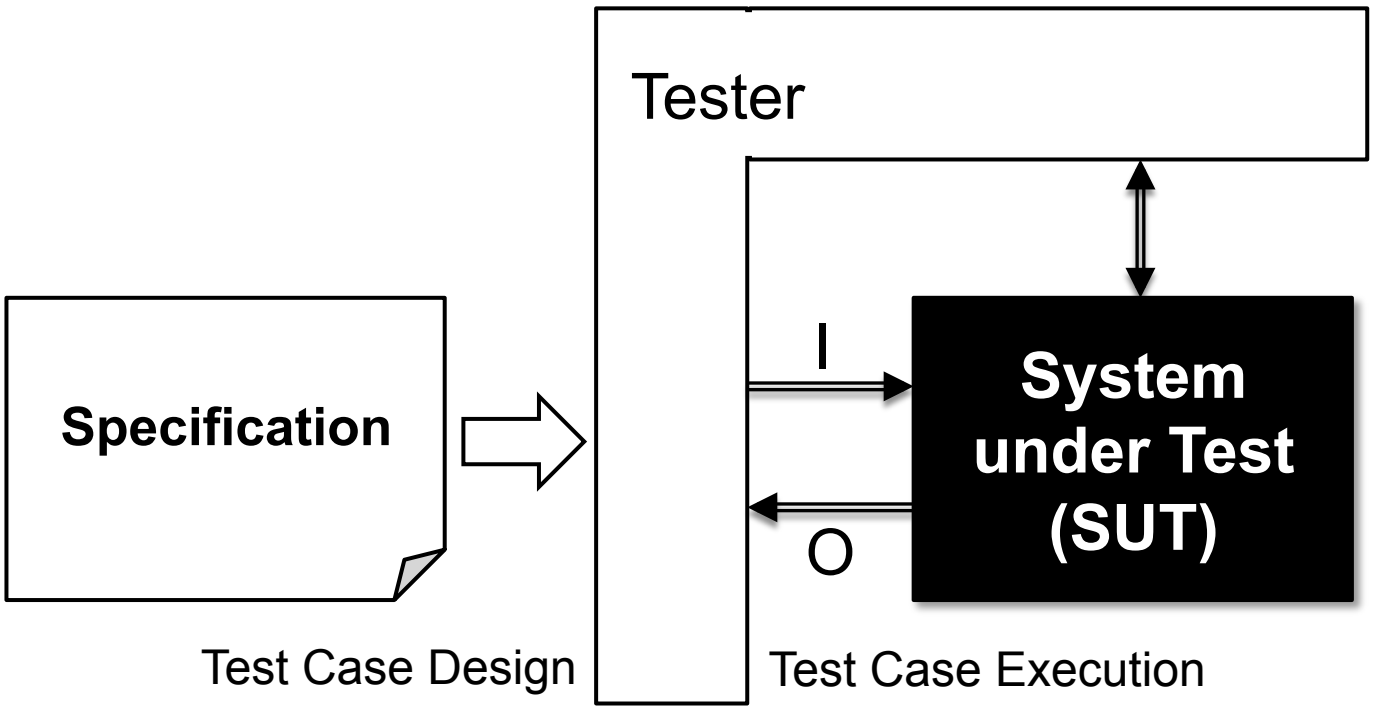
Technische Universität Braunschweig

ISF

Model-based Testing of Software Product Lines – Part I

Prof. Ina Schaefer, 09.06.2015, Halmstad Summer School on Testing

Dynamic Black-Box Software Testing



Roadmap

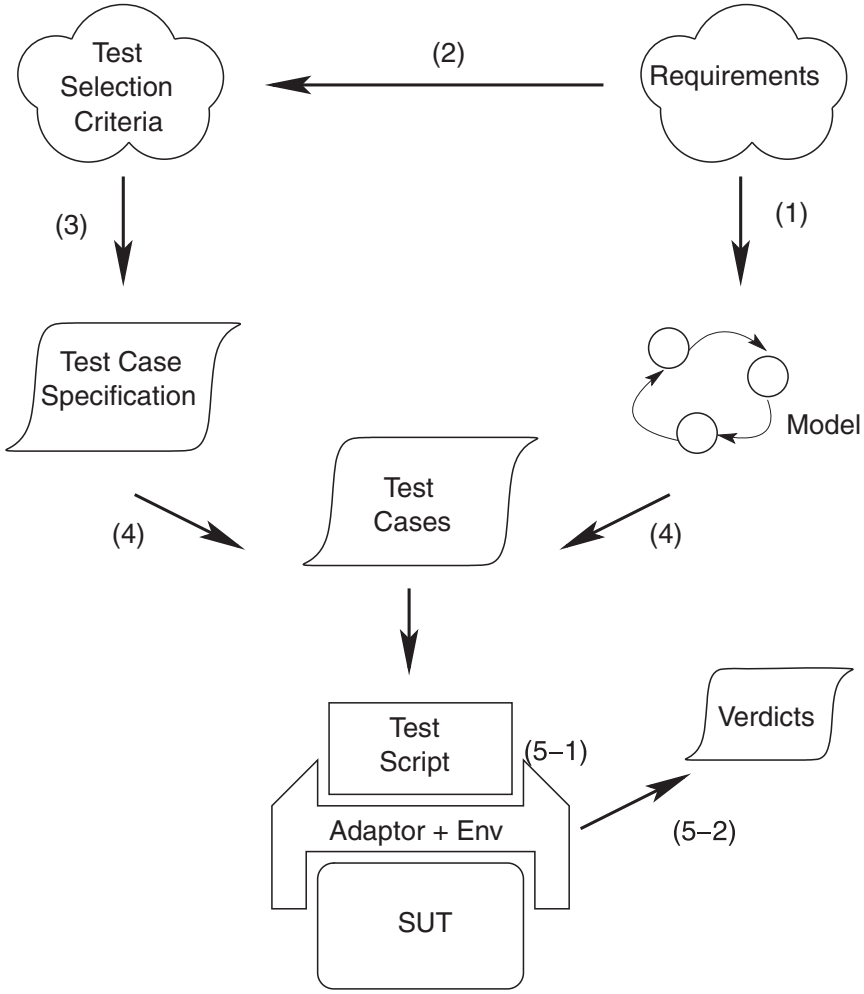
- Model-based Testing - Overall Concept
- Model-based Testing - Some Theory

MBT – Overall Idea

„MBT encompasses the processes and techniques for the automatic derivation of abstract test cases from abstract models, the generation of concrete tests from abstract tests, and the manual or automated execution of the resulting concrete test cases.“

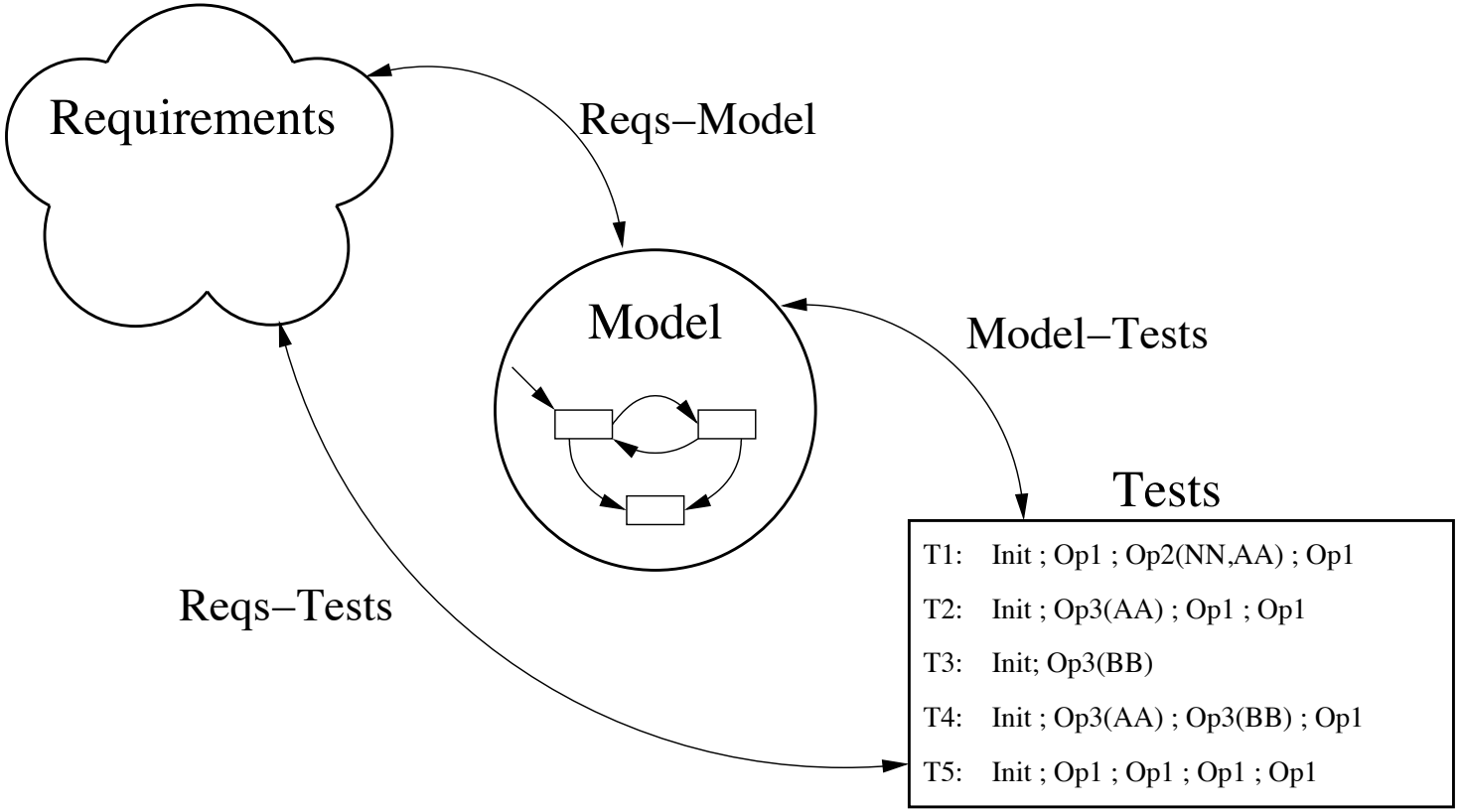
[Mark Utting, Alexander Pretschner, Bruno Legeard: A taxonomy of model-based testing approaches. Softw. Test., Verif. Reliab. 22(5): 297-312 (2012)]

Process of MBT



[Mark Utting, Alexander Pretschner, Bruno Legeard: A taxonomy of model-based testing approaches. Softw. Test., Verif. Reliab. 22(5): 297-312 (2012)]

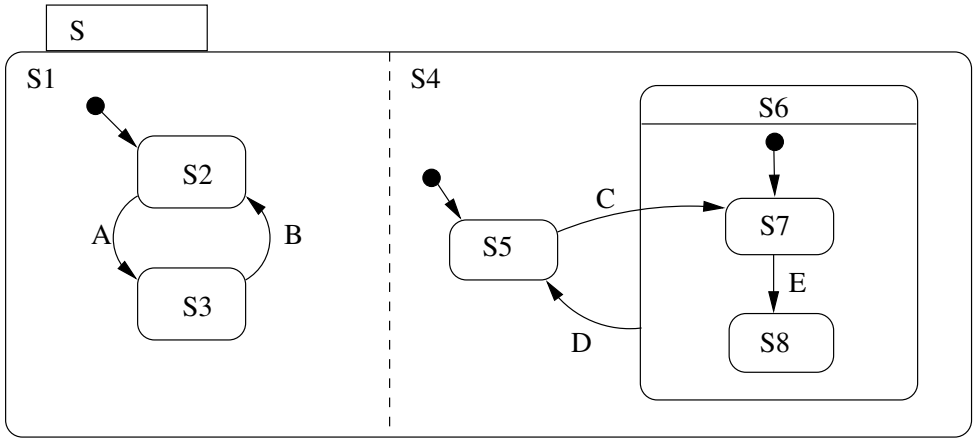
Traceability



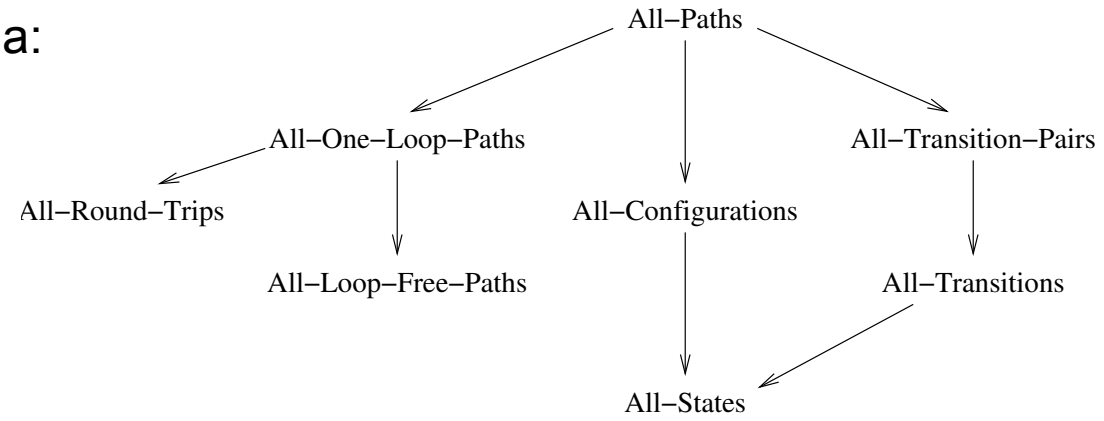
From [Utting/ Legeard, Practical MBT, 2007]

MBT – Example: State Charts

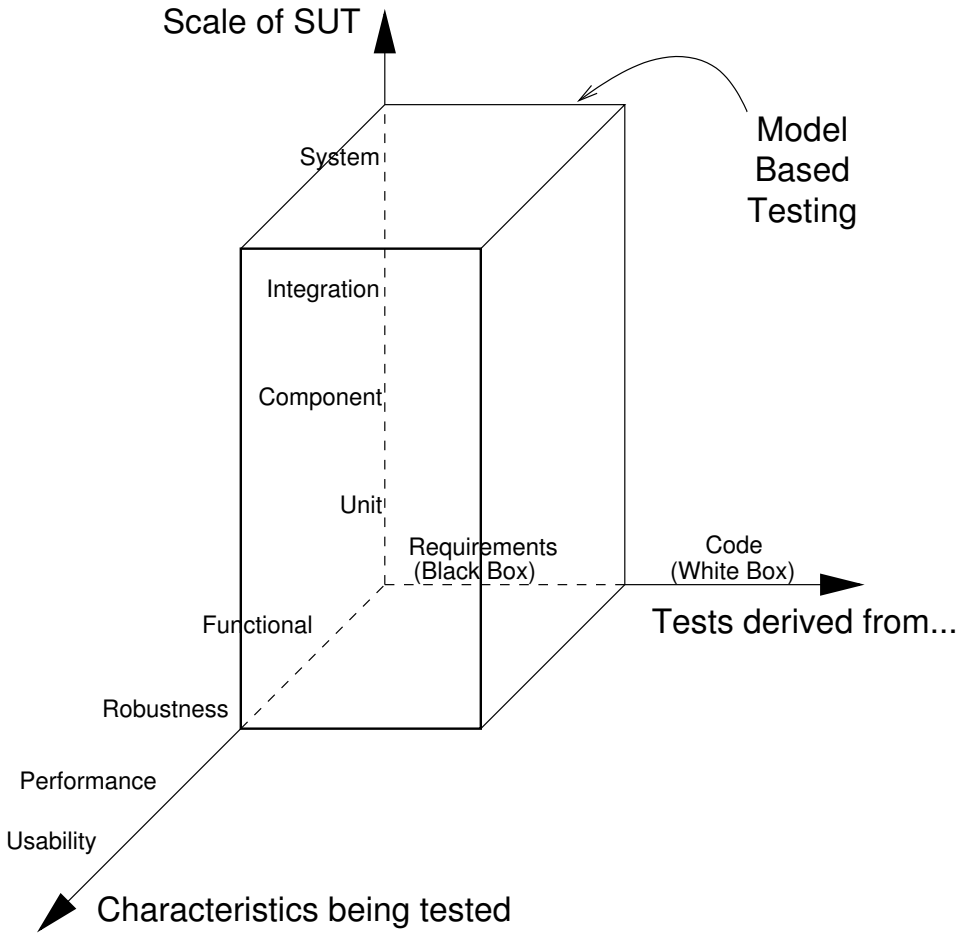
Example:



Coverage Criteria:

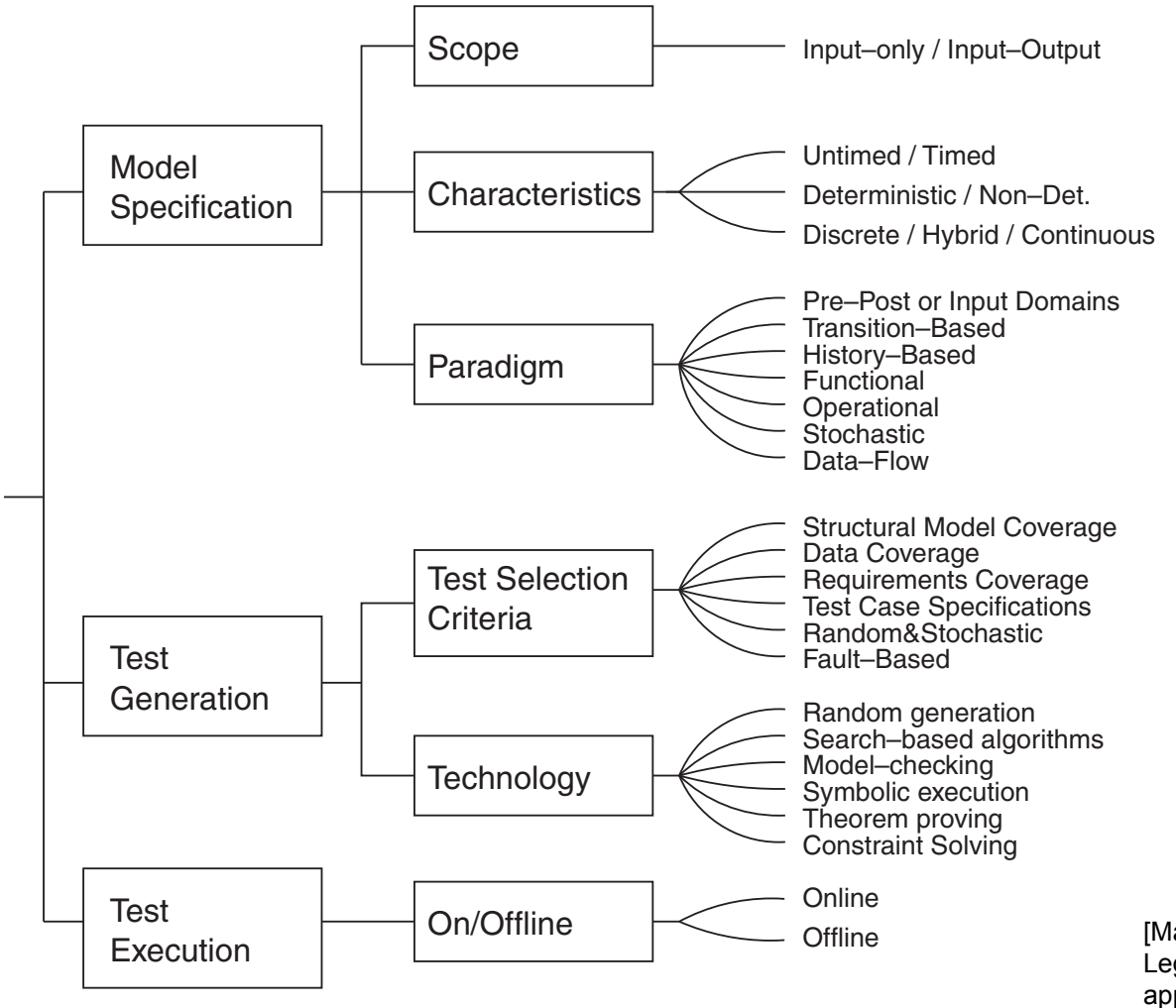


Scope of MBT



From [Utting/ Legeard, Practical MBT, 2007]

Taxonomy of MBT

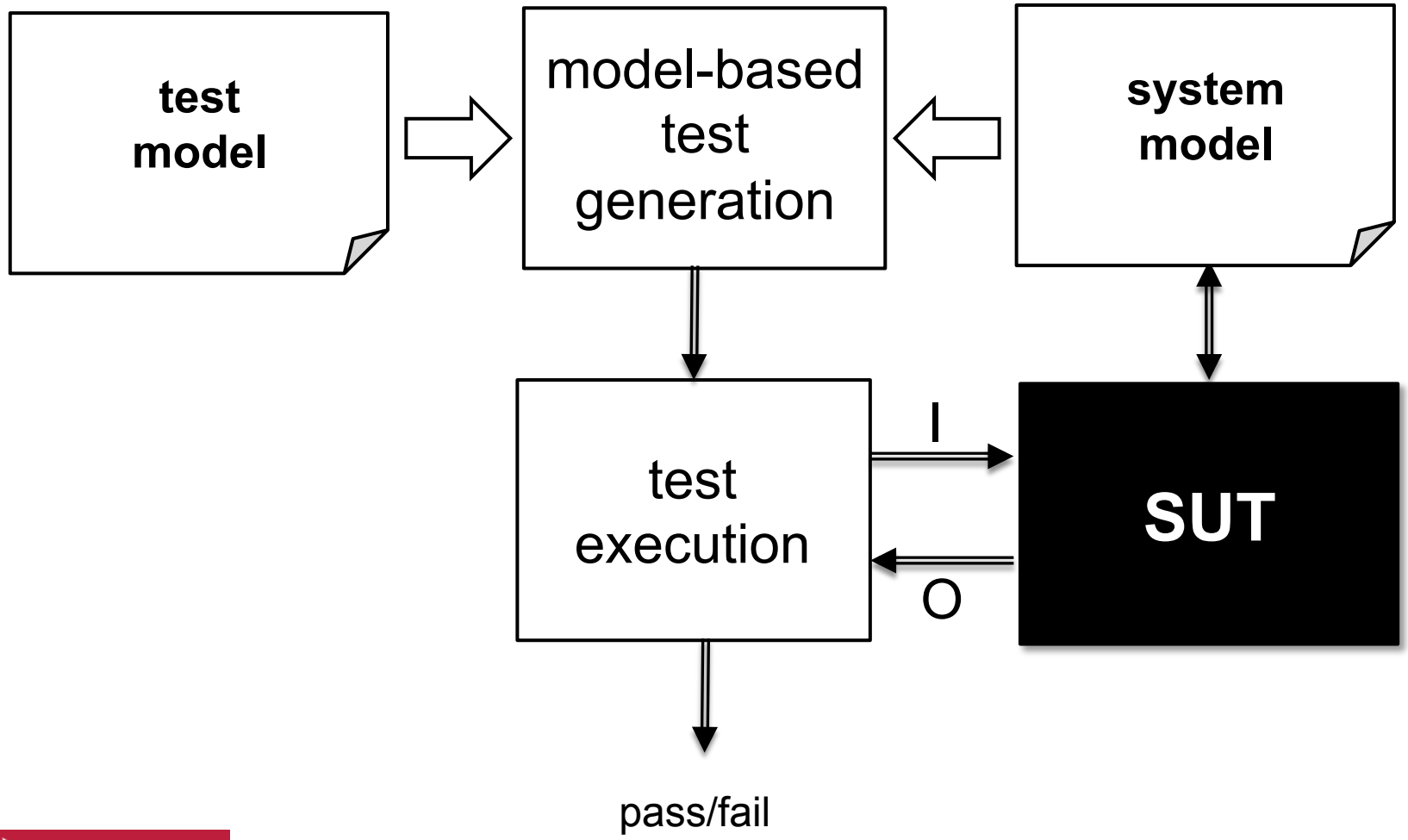


[Mark Utting, Alexander Pretschner, Bruno Legeard: A taxonomy of model-based testing approaches. Softw. Test., Verif. Reliab. 22(5): 297-312 (2012)]

Advantages of MBT

- Systematic and automatic test case generation
- Useable in early development phases (Model-in-the-Loop)
- Automatization of test case execution
- Regression test planning by analysis of model changes

Model-Based Testing – A more theoretical perspective



Conformance Relations

implementation relation: $i \simeq s$ with *implemetation i* and *formal behavioral specification s*

preorder relation: $i \sqsubseteq s$ implementation shows at most the behaviors of the specification

intentional conformance: $i\ conforms\ s :\Leftrightarrow [\![i]\!] \subseteq [\![s]\!]$

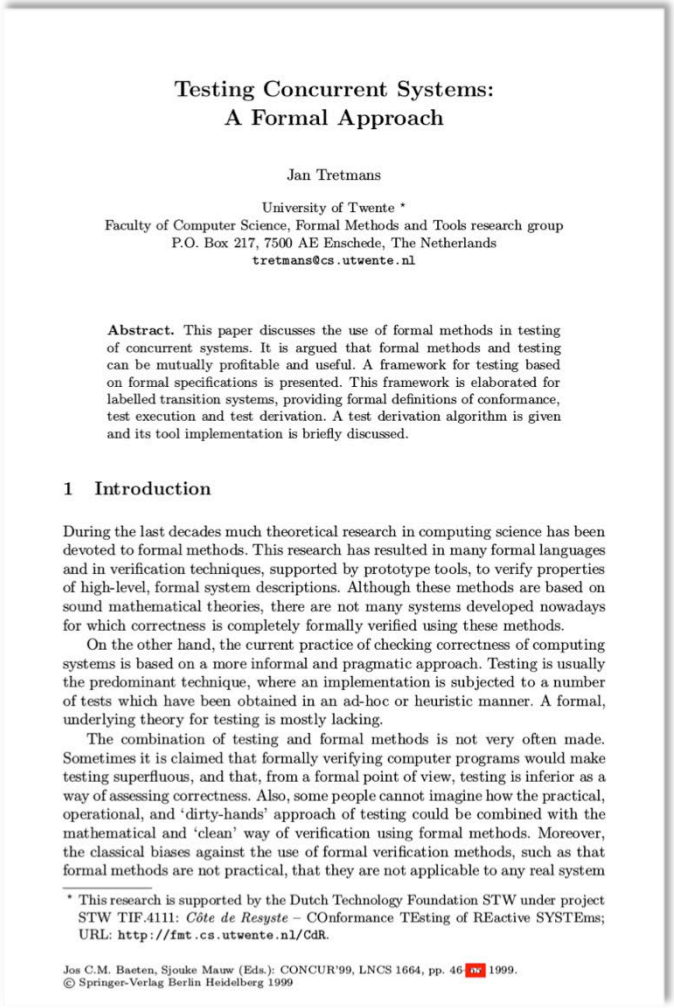
where $[\![\cdot]\!]$ defines sets of all observable behaviors

extensional conformance: $i\ conforms\ s :\Leftrightarrow \forall u \in \mathcal{U} : obs(u, i) \approx obs(u, s)$

where $\mathcal{U}$ defines sets of all observers

Model-based I/O Conformance Testing

- Proposed by Jan Tretman in the 90's
- Model-based functional conformance testing of systems with reactive, non-deterministic behaviors
- Input, output, and quiescence based testing theory
- Based on I/O labeled transition systems as test models AND implementation models
- Proven sound and exhaustive
- Rich tool support
- Formal basis for many advanced testing frameworks



Testing Concurrent Systems:
A Formal Approach

Jan Tretmans

University of Twente *
Faculty of Computer Science, Formal Methods and Tools research group
P.O. Box 217, 7500 AE Enschede, The Netherlands
tretmans@cs.utwente.nl

Abstract. This paper discusses the use of formal methods in testing of concurrent systems. It is argued that formal methods and testing can be mutually profitable and useful. A framework for testing based on formal specifications is presented. This framework is elaborated for labelled transition systems, providing formal definitions of conformance, test execution and test derivation. A test derivation algorithm is given and its tool implementation is briefly discussed.

1 Introduction

During the last decades much theoretical research in computing science has been devoted to formal methods. This research has resulted in many formal languages and in verification techniques, supported by prototype tools, to verify properties of high-level, formal system descriptions. Although these methods are based on sound mathematical theories, there are not many systems developed nowadays for which correctness is completely formally verified using these methods.

On the other hand, the current practice of checking correctness of computing systems is based on a more informal and pragmatic approach. Testing is usually the predominant technique, where an implementation is subjected to a number of tests which have been obtained in an ad-hoc or heuristic manner. A formal, underlying theory for testing is mostly lacking.

The combination of testing and formal methods is not very often made. Sometimes it is claimed that formally verifying computer programs would make testing superfluous, and that, from a formal point of view, testing is inferior as a way of assessing correctness. Also, some people cannot imagine how the practical, operational, and 'dirty-hands' approach of testing could be combined with the mathematical and 'clean' way of verification using formal methods. Moreover, the classical biases against the use of formal verification methods, such as that formal methods are not practical, that they are not applicable to any real system

* This research is supported by the Dutch Technology Foundation STW under project STW TIF.4111: *Côte de Resyste* – COnformance TEsting of REactive SYSTEms; URL: http://fmt.cs.utwente.nl/CdR.

Jos C.M. Baeten, Sjouke Mauw (Eds.): CONCUR'99, LNCS 1664, pp. 46-[illegible] 1999.

Running Example

Beverage vending machine

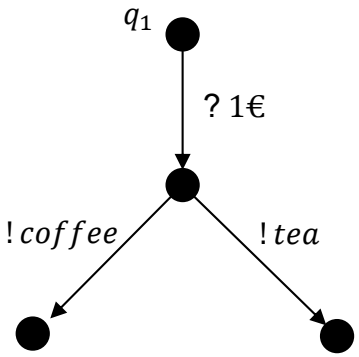
- Input actions
  - $I = \{1€, 2€\}$
  - Transitions labels prefixed with “?”
- Output actions
  - $U = \{coffee, tea\}$
  - Transition labels prefixed with “!”

I/O-Labeled Transition Systems

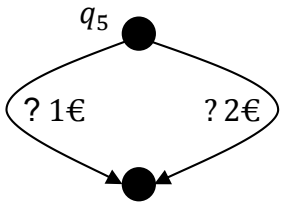
I/O Labeled Transitionsystem: $(\boldsymbol{Q}, \boldsymbol{q_0}, \boldsymbol{I}, \boldsymbol{U}, \rightarrow)$, where

- $\boldsymbol{Q}$ is a countable set of states,
- $\boldsymbol{q_0} \in \boldsymbol{Q}$ is the initial state,
- $\boldsymbol{I}$ and $\boldsymbol{U}$ are disjoint sets of input actions and output actions, and
- $\rightarrow \subseteq \boldsymbol{Q} \times \boldsymbol{act} \times \boldsymbol{Q}$ is a labeled transition relation.

LTS - Examples



$$Tr(q_1) = \{?\,1€, ?\,1€ \cdot !\,coffee, ?\,1€ \cdot !\,tea\}$$



$$Tr(q_5) = \{?\,1€, ?\,2€\}$$

LTS Trace Semantics

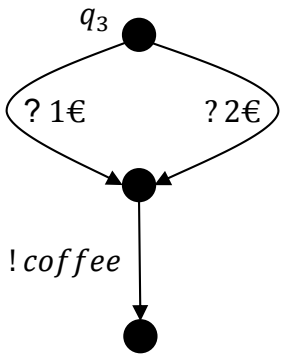
Each compution refers to some path

$$q_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} s_2 \xrightarrow{\mu_3} \cdots \xrightarrow{\mu_{n-1}} s_{n-1} \xrightarrow{\mu_n} s_n$$

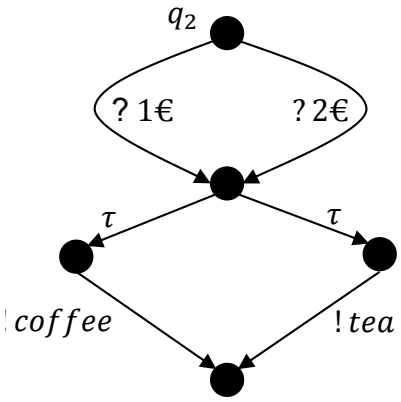
The behavior of a computation is defined by a trace

$$trace\ \sigma = \mu_1 \mu_2 \cdots \mu_n \in act^*$$

LTS - Examples



$$Tr(q_3) = \{?\,1€, ?\,2€, ?\,1€ \cdot !\,coffee, ?\,2€ \cdot !\,coffee\}$$



$$Tr(q_2) = \{?\,1€, ?\,2€, ?\,1€ \cdot !\,coffee, ?\,1€ \cdot !\,tea, ?\,2€ \cdot !\,coffee, ?\,2€ \cdot !\,tea\}$$

LTS Trace Notations

Let **s** be an **I/O** $\mathcal{LTS}$, $\mu_i \in I \cup U \cup \{\tau\}$ and $a_i \in I \cup U$

$$s \xrightarrow{\mu_1 \cdots \mu_n} s' := \exists s_0, \ldots, s_n : s = s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_n} s_n = s'$$

$$s \xrightarrow{\mu_1 \cdots \mu_n} := \exists s' : s = s \xrightarrow{\mu_1 \cdots \mu_n} s'$$

$$\neg s \xrightarrow{\mu_1 \cdots \mu_n} := \nexists s' : s \xrightarrow{\mu_1 \cdots \mu_n} s'$$

LTS Trace Notations

Let **s** be an **I/O** $\mathcal{LTS}$, $\mu_i \in I \cup U \cup \{\tau\}$ and $a_i \in I \cup U$

$$s \overset{\epsilon}{\Rightarrow} s' := s = s' or\ s \overset{\tau \cdots \tau}{\longrightarrow} s'$$

$$s \overset{a}{\Rightarrow} s' := \exists s_1, s_2 : s \overset{\epsilon}{\Rightarrow} s_1 \overset{a}{\rightarrow} s_2 \overset{\epsilon}{\Rightarrow} s'$$

$$s \overset{a_1 \cdots a_n}{\Longrightarrow} s' := \exists s_0, \ldots, s_n : s = s_0 \overset{a_1}{\Rightarrow} s_1 \overset{a_2}{\Rightarrow} \ldots \overset{a_n}{\Rightarrow} s_n = s'$$

LTS Trace Notations

Let **s** be an **I/O** $\mathcal{LTS}$, $\sigma \in (I \cup U)^*$

$$s \overset{\sigma}{\Rightarrow} := \exists s' : = \exists s' :\ s \overset{\sigma}{\Rightarrow} s'$$

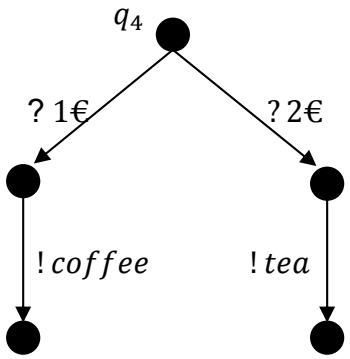
$$\neg s \overset{\sigma}{\Rightarrow} :=\ \nexists s' : s \overset{\sigma}{\Rightarrow} s'$$

LTS Trace Notations

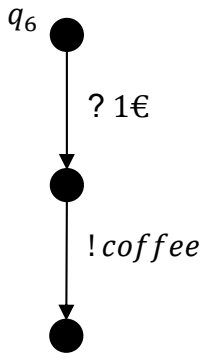
The set of traces in an $\mathcal{LTS}$ is defined as

$$Tr(s) \coloneqq \{\sigma \in (I \cup U)^* | \exists s' \in Q : q_0 \overset{\sigma}{\Rightarrow} s'\}.$$

LTS - Examples

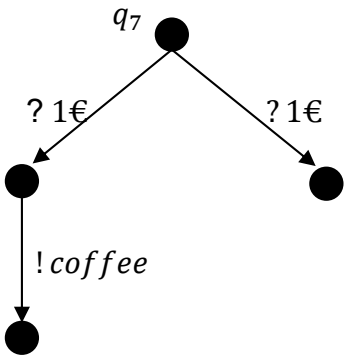


$$Tr(q_4) = \{?\,1€, ?\,2€, ?\,1€ \cdot !\,coffee, ?\,2€ \cdot !\,tea\}$$



$$Tr(q_6) = \{?\,1€, ?\,1€ \cdot !\,coffee\}$$

LTS - Examples



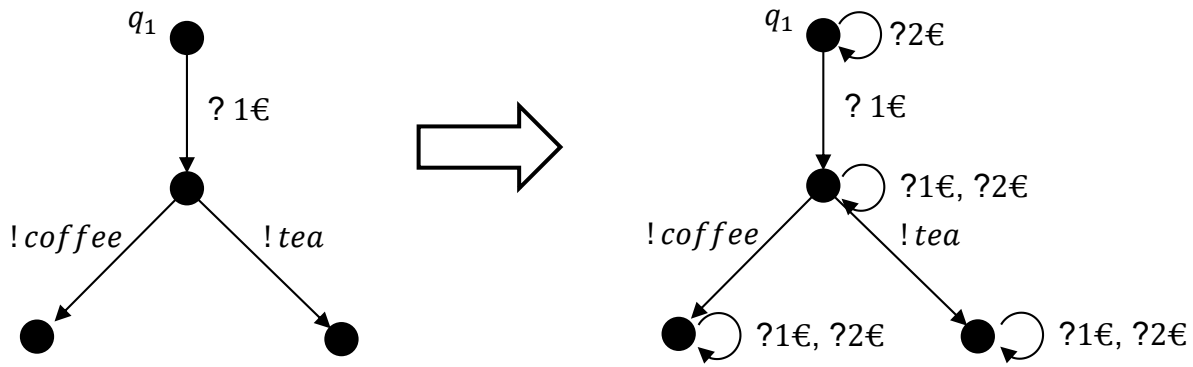
$$Tr(q_7) = \{?\,1€, ?\,1€ \cdot !\,coffee\}$$

$$Tr(q_8) = \{\}$$

Input-Enabled Transition Systems

An $\mathcal{LTS}$ is (weak) input-enabled iff for every state $s \in Q$ with $q_0 \Rightarrow^* s$ and for all $a \in \mathrm{I}$ it holds that $s \overset{a}{\Rightarrow}$.

Input Completion - Example



Not input-enabled LTS

Input-enabled LTS

A First Attempt: Conformance as Trace Inclusion

$$i \; \boldsymbol{conforms} \; s :\Leftrightarrow Tr(i) \subseteq Tr(s)$$

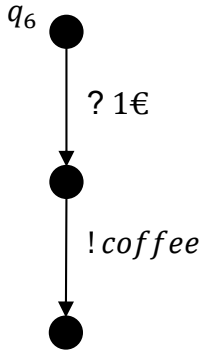
Solution: explicit notion of quiescent behavior

- Fails to refuse trivial implementations

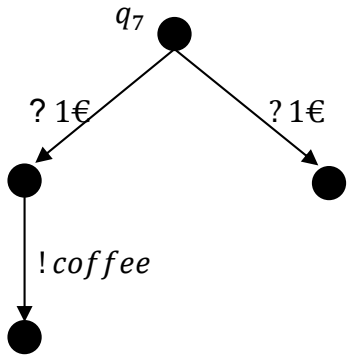
- Fails to take the asymetric nature of $\mathcal{LTS}$ traces with I/O actions into account

Solution: distinguish input and output behaviors in traces

What is the difference?



$$Tr(q_6) = \{?\,1€, ?\,1€ \cdot !\,coffee\}$$

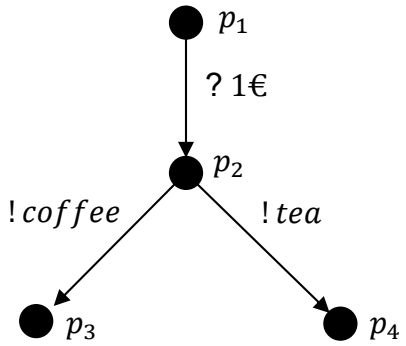


$$Tr(q_7) = \{?\,1€, ?\,1€ \cdot !\,coffee\}$$

Some Auxiliary Definitions: Init Sets

$$Let\ s\ be\ an\ \mathcal{LTS}, p \in Q, P \subseteq Q\ and\ \sigma \in (I \cup U)^*.$$

$$init(p) := \{\mu \in (I \cup U) | p \xrightarrow{u}\}$$

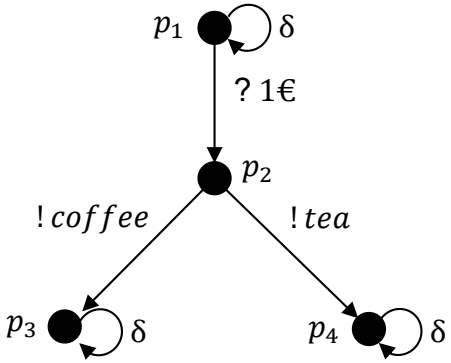


$$init(p_1) := \{?\,1€\}$$
$$init(p_2) := \{!\,coffee, !\,tea\}$$
$$init(p_3) := \{\ \ \}$$
$$init(p_4) := \{\ \ \}$$

Some Auxiliary Definitions: Quiescent States

$$Let\ s\ be\ an\ \mathcal{LTS}, p \in Q, P \subseteq Q\ and\ \sigma \in (I \cup U)^*.$$

$$p\ is\ \boldsymbol{quiescent}, denoted\ \delta(p), iff\ init(p) \subseteq I$$



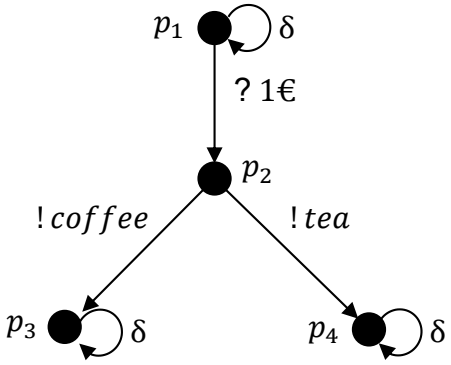
$$init(p_1) := \{?\,1€\}$$
$$init(p_2) := \{!\,coffee, !\,tea\}$$
$$init(p_3) := \{\ \}$$
$$init(p_4) := \{\ \}$$

$$I = \{?\,1€, ?\,2€\}$$

Some Auxiliary Definitions: Suspension Traces

$$Let\ s\ be\ an\ \mathcal{LTS}, p \in Q, P \subseteq Q\ and\ \sigma \in (I \cup U)^*.$$

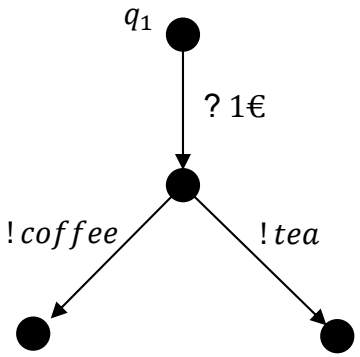
$$Straces(p) := \{\sigma' \in (I_S \cup U_S \cup \{\delta\})^* \mid p \overset{\sigma'}{\Rightarrow}\}\ where\ q \overset{\delta}{\rightarrow} q\ iff\ \delta(p)$$



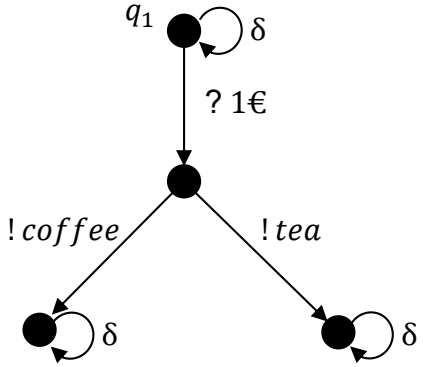
$$Straces(p_1) = \{\delta, ?\,1€, ?\,1€ \cdot !\,coffee, ?\,1€ \cdot !\,tea, ?\,1€ \cdot !\,coffee \cdot \delta, ?\,1€ \cdot !\,tea \cdot \delta\}$$
$$Straces(p_2) = \{!\,coffee, !\,tea, !\,coffee \cdot \delta, tea \cdot \delta\}$$
$$Straces(p_3) = \{\delta\}$$
$$Straces(p_4) = \{\delta\}$$

Quiescent Behaviors

Trace $Tr(q_1)$



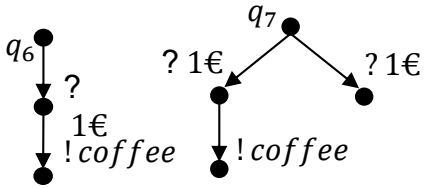
add delta transitions



$$Straces(q_1) = \{\delta, ?\,1€, \delta \cdot ?\,1€, ?\,1€ \cdot !\,coffee, ?\,1€ \cdot !\,\text{coffee} \cdot \delta, \ldots\}$$

Allows to discriminate (non-)behaviors

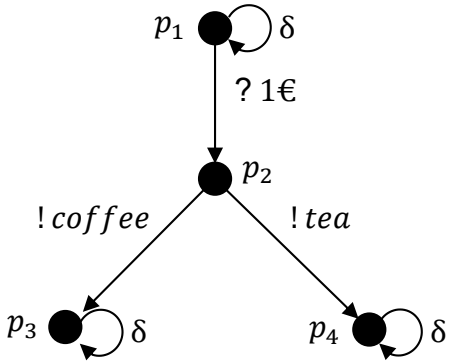
- $?\,1€ \cdot \delta \notin Straces(q_6), whereas\ ?\,1€ \cdot \delta \in Straces(q_7)$
-



Some Auxiliary Definitions: After Sets

$$Let\ s\ be\ an\ \mathcal{LTS}, p \in Q, P \subseteq Q\ and\ \sigma \in (I \cup U)^*.$$

$$p\ \boldsymbol{after}\ \sigma := \{q \in U \mid p \overset{\sigma}{\Rightarrow} q\}$$



$$U = \{!\,coffee, !\,tea\}$$

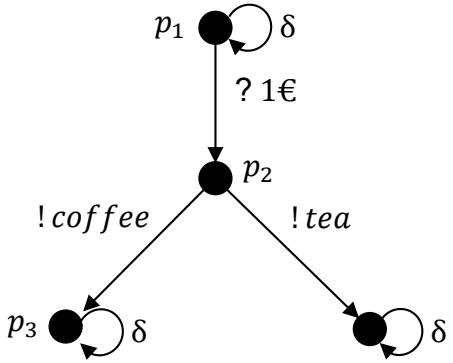
$$p_2\ \boldsymbol{after}\ !\,coffee = \{p_3\}$$
$$p_2\ \boldsymbol{after}\ !\,tea = \{p_4\}$$

$$p_1\ \boldsymbol{after}\ ?\,2€ = \{\ \ \}$$
$$p_4\ \boldsymbol{after}\ !\,tea = \{\ \ \}$$

Some Auxiliary Definitions: Out Sets

$$Let\ s\ be\ an\ \mathcal{LTS}, p \in Q, P \subseteq Q\ and\ \sigma \in (I \cup U)^*.$$

$$out(P) \coloneqq \{\mu \in U \mid \exists p \in P : p \xrightarrow{\mu}\} \cup \{\delta \mid \exists p \in P : \delta(p)\}$$



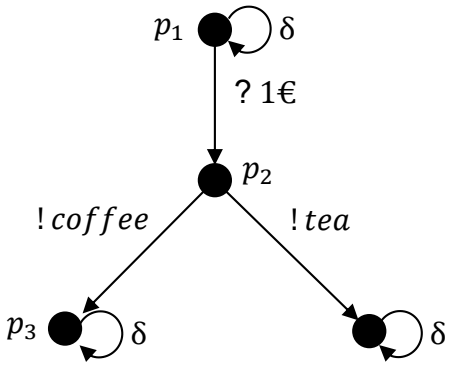
$$P = p\ \boldsymbol{after}\ \sigma$$

$$P_1 = \{p_1\ \boldsymbol{after}\ \delta\} = \{p_1\}$$
$$P_2 = \{p_2\ \boldsymbol{after}\ !tea, p_2\ \boldsymbol{after}\ !coffee\} = \{p_3, p_4\}$$
$$P_3 = \{p_3\ \boldsymbol{after}\ \delta\} = \{p_3\}$$
$$P_4 = \{p_4\ \boldsymbol{after}\ \delta\} = \{p_4\}$$

Some Auxiliary Definitions: After-Out Sets

$$Let\ s\ be\ an\ \mathcal{LTS}, p \in Q, P \subseteq Q\ and\ \sigma \in (I \cup U)^*.$$

$$out(P) \coloneqq \{\mu \in U \mid \exists p \in P : p \xrightarrow{\mu}\} \cup \{\delta \mid \exists p \in P : \delta(p)\}$$



$$P_1 = \{p_1\ \textbf{after}\ \delta\} = \{p_1\}$$
$$P_2 = \{p_2\ \textbf{after}\ !tea, p_2\ \textbf{after}\ !coffee\} = \{p_3, p_4\}$$
$$P_3 = \{p_3\ \textbf{after}\ \delta\} = \{p_3\}$$
$$P_4 = \{p_4\ \textbf{after}\ \delta\} = \{p_4\}$$

$$Out(P_1) = \{\delta\}$$
$$Out(P_2) = \{!tea, !coffee\}$$
$$Out(P_3) = \{\delta\}$$
$$Out(P_4) = \{\delta\}$$

Second Attempt: I/O Conformance (IOR)

Class of I/O LTS labeled over I and U

Subclass of input-enabled I/O LTS

$$Let\ s \in \mathcal{LTS}(I \cup \mathrm{U})\ and\ i \in \mathcal{IOTS}(I, U).$$

$$i\ \boldsymbol{ior}\ s \iff \forall \sigma \in act_{\delta}^{*} : out(i\ \boldsymbol{after}\ \sigma) \subseteq out(s\ \boldsymbol{after}\ \sigma)$$

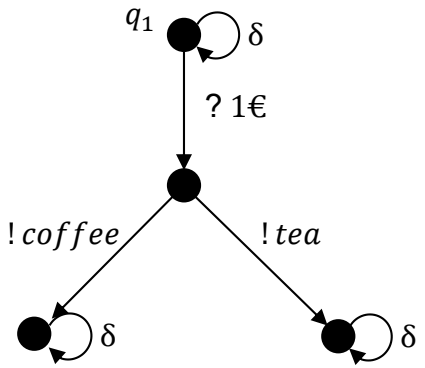
$$i\ \boldsymbol{ior}\ s \iff Straces(i) \subseteq Straces(s)$$

Example

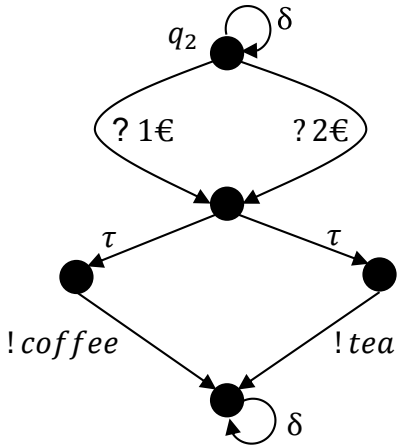
- Assume δ-transitions in the $\mathcal{LTS}$ example
- Possible environmental stimulations are $\sigma = ?1€$ and $\sigma' = ?2€$
- Investigate the observable behavior

Example

$\text{out}(\text{q}_1 \text{ after } \sigma) = \{coffee, tea\}, out(q_1 after\ \sigma') = \{\}$

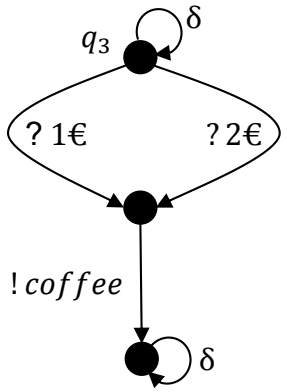


$out(q_2\ after\ \sigma) = \{coffee, tea\}, out(q_2 after\ \sigma') = \{coffee, tea\}$

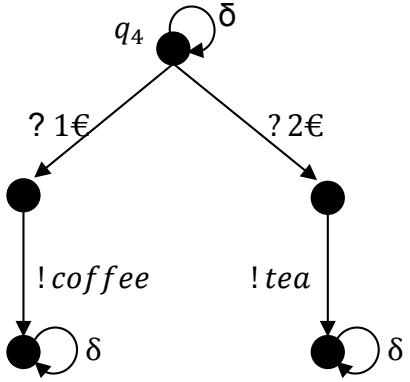


Example

$out(q_3\ after\ \sigma) = \{coffee\}, out(q_3\ after\ \sigma') = \{coffee\}$

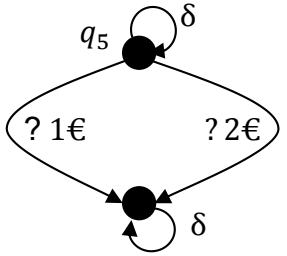


$out(q_4\ after\ \sigma) = \{coffee\}, out(q_4\ after\ \sigma') = \{tea\}$

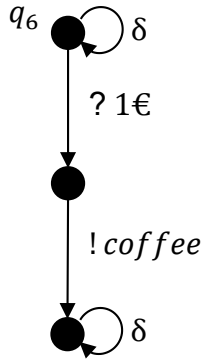


Example

$$out(q_5\ after\ \sigma) = \{\delta\}, out(q_5\ after\ \sigma') = \{\delta\}$$



$$out(q_6\ after\ \sigma) = \{coffee\}, out(q_6\ after\ \sigma') = \{\}$$



Second Attempt: I/O Conformance (IOR)

$$Let\ s \in \mathcal{LTS}(I \cup \mathrm{U})\ and\ i \in \mathcal{JOTS}(I, U).$$

Problem: this is quite a lot!

$$i\ \mathbf{ior}\ s \iff \forall \sigma \in act_{\delta}^{*} : out(i\ \mathbf{after}\ \sigma) \subseteq out(s\ \mathbf{after}\ \sigma)$$

$$i\ \mathbf{ior}\ s \iff Straces(i) \subseteq Straces(s)$$

Third Attempt: IOCO

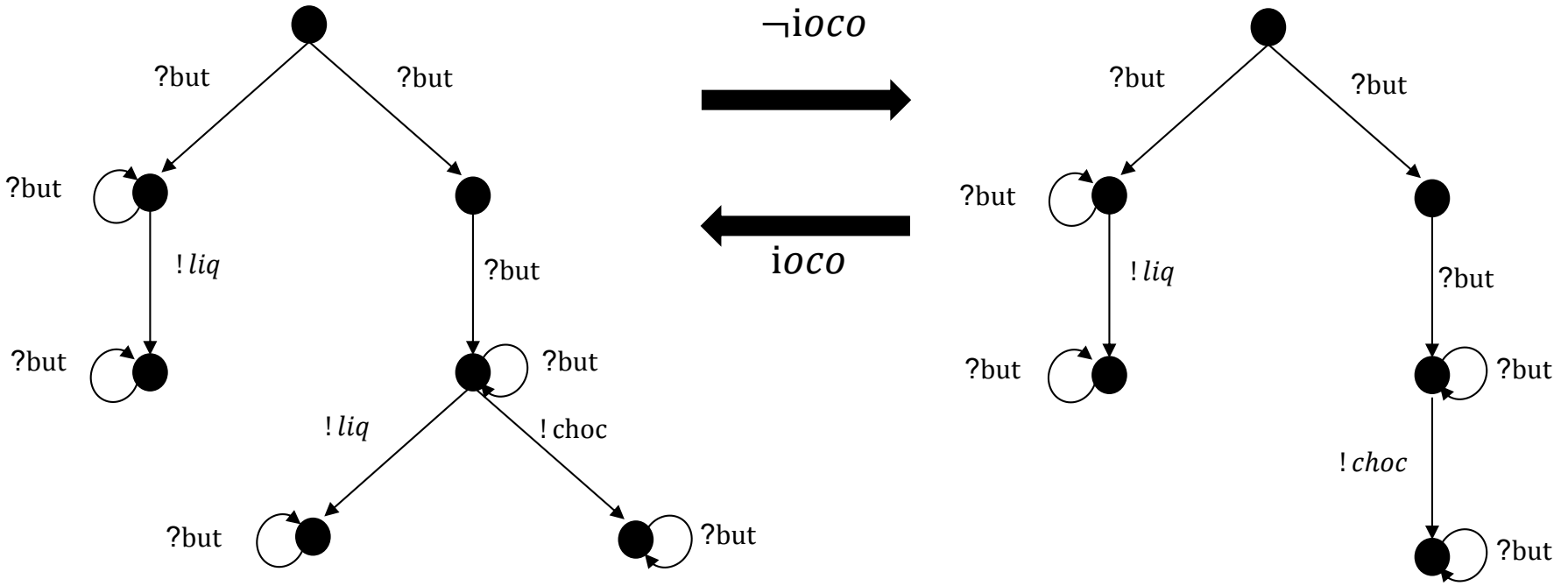
$$Let\ s \in \mathcal{LTS}(I \cup \mathrm{U})\ and\ i \in \mathcal{IOTS}(I, U).$$

Focus on specified behaviors only

$$i\ \boldsymbol{ioco}\ s \Leftrightarrow \forall \sigma \in Straces(s) : out(i\ \boldsymbol{after}\ \sigma) \subseteq out(s\ \boldsymbol{after}\ \sigma)$$

$$\boldsymbol{ior} \subset \boldsymbol{ioco}$$

Example [Tretmans, 1999]



Third Attempt: IOCO

$$Let\ s \in \mathcal{LTS}(I \cup \mathrm{U})\ and\ i \in \mathcal{IOTS}(I, U).$$

Still infinite in case of loops

$$i\ \boldsymbol{ioco}\ s \iff \forall \sigma \in Straces(s) : out(i\ \boldsymbol{after}\ \sigma) \subseteq out(s\ \boldsymbol{after}\ \sigma)$$

$$\boldsymbol{ior} \subset \boldsymbol{ioco}$$

IOCO wrt. F

- The set of suspension traces under consideration is restricted to sub sets $\mathcal{F} \subseteq act_\delta^*$
- The restricted ioco relation is denoted as

$$i \; \boldsymbol{ioco}_{\mathcal{F}} \; s :\Leftrightarrow \forall \sigma \in \mathcal{F} : out(i \; \boldsymbol{after} \; \sigma) \subseteq out(s \; \boldsymbol{after} \; \sigma) \; ,$$

where $\boldsymbol{ior} = \boldsymbol{ioco}_{act_\delta^*}$ and $\boldsymbol{ior} = \boldsymbol{ioco}_{Straces(s)}$ holds.

This is still an intentional characterization of conformance. How to prove this by testing?

Extensional IOCO

$$i \; \textbf{passes} \; t :\Leftrightarrow obs(i,t) \approx obs(s,t)$$

Observers (testers) are characterized by a finite sets of test cases they perform on an SUT

Test Cases

A test case **t** is an I/O labeled $\mathcal{LTS}$ such that

- **t** is deterministic and has a finite set of traces,
- **Q** contains terminal states **pass** and **fail** with $init(\boldsymbol{pass}) = init(\boldsymbol{fail}) = \emptyset$,
- for each non-terminal state $q \in Q$ either
  1. $init(q) = \{a\}$ for $a \in I$ or
  2. $init(q) = U \cup \{\theta\}$

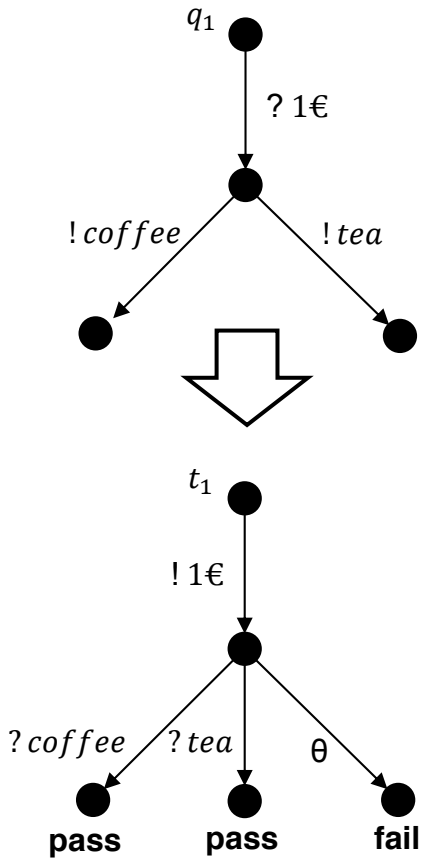
  holds.

denotes observation of quiescence

By $\mathcal{TEST}$ we denote the subclass of I/O labeled $\mathcal{LTS}$ representing valid test cases **t**

Test Case Generatation – Example

Test case for specification q_1



Stimulated input

- *!1€*

Expected output

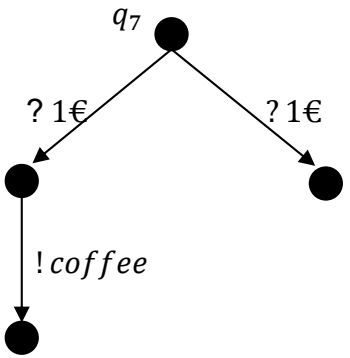
- *either coffee*
- *or tea*

Observable errors

- No output occurs: Θ

Test Case Generation – Example 2

Test case for specification q_7



Stimulated input

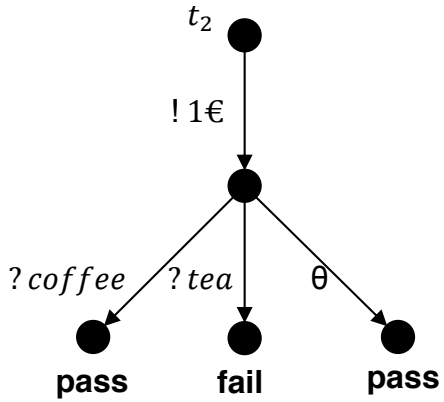
- *!1€*

Expected output

- *coffee*
- no output: Θ

Observable errors

- *tea*



IOCO is correct = sound + exhaustive

Let $s \in \mathcal{LTS}(I \cup U), i \in \mathcal{IOTS}(I \cup \mathrm{U})$ and $\mathcal{F} \subseteq Straces(s)$

Then it holds that

1. the set $\mathcal{TEST}$ of all derivable test cases is **sound** and
2. the set $\mathcal{TEST}$ of all derivable test cases is **exhaustive**.

Conclusion

Model-based testing

- automates black-box test case generation and exectution.
- requires a formal model of the system specificatic
- can be based on a formal notion of conformance.

Some Further Readings

- Malte Lochau, Sven Peldszus, Matthias Kowal, Ina Schaefer: Model-Based Testing. SFM 2014: 310-342

- Mark Utting and Bruno Legeard, Practical Model-Based Testing: A Tools Approach, Morgan-Kaufmann 2007.

- Jan Tretmans: Model Based Testing with Labelled Transition Systems. Formal Methods and Testing 2008: 1-38