

# Testing of Reactive Systems

## The IOCO testing theory

Henrik Bohnenkamp

Lehrstuhl Informatik 2 (MOVES)  
RWTH Aachen

Summer Semester 2008

# First part of the lecture

## Processes

- Labelled Transition Systems
- Processes
- Structural operational semantics

# First part of the lecture

## Implementation Relations

- preorders, implementation relations
- trace preorder, bisimulation
- linear-time/branching time spectrum
- the collapse of LTBT for deterministic processes

# First part of the lecture

## Observing behaviour

- Observers, observations, implementation relations
- completed, failures, failure trace preorder
- $\tau$ -steps and divergence

# First part of the lecture

## Conclusions

- Equivalences of LTBT spectrum result of years of research in **Concurrency Theory**
- Another aspect, completely ignored here: **Compositionality**.  
For  $p, q, r \in \mathbb{P}$ ,  $A \subseteq \text{Act}$ :  
$$\text{does } p \leq q \text{ imply } p \parallel_A r \leq q \parallel_A r?$$
- Basic knowledge when working with formal models

# First part of the lecture

## Conclusions

- Equivalences of LTBT spectrum result of years of research in **Concurrency Theory**
- Another aspect, completely ignored here: **Compositionality**.  
For  $p, q, r \in \mathbb{P}$ ,  $A \subseteq \text{Act}$ :

does  $p \leq q$  imply  $p \parallel_A r \leq q \parallel_A r$ ?

- Basic knowledge when working with formal models

Not suitable for practical testing

# IOCO: a testing theory used in practice

## Before:

- Before: testing understood as experimenting with processes
- processes related according to the observations made

## Now: conformance testing

- Given: **specification**
- specification describes correct behaviour of **implementation under test** (IUT)
- **Testing** as a means to (dis-)prove conformance to spec

# IOCO: a testing theory used in practice

## Before:

- Before: testing understood as experimenting with processes
- processes related according to the observations made

## Now: conformance testing

- Given: **specification**
- specification describes correct behaviour of **implementation under test** (IUT)
- **Testing** as a means to (dis-)prove conformance to spec

# IOCO: Ingredients

## Conformance relation

- **SPEC**: set of specifications
- **IMPL**: set of potential implementations
- **Conformance** described as **implementation relation**:

$$\mathbf{conf} \subseteq \text{IMPL} \times \text{SPEC}$$

- $i \mathbf{conf} s$ , if  $i$  is an implementation conforming to  $s$
- testing **theory**, so SPEC and IMPL will contain mathematical objects: LTS with **input** and **output** actions

# IOCO: Ingredients

## Test-Cases, test execution

- IOCO defines test-case generation algorithm
- Algorithm derives test-cases from specification  $s \in SPEC$
- Test execution: how to run a test-case against a implementation
- **on-the-fly** testing algorithm: test-case generation and execution **simultaneously**

# IOCO: Ingredients

## Soundness, Exhaustiveness

- test-cases derived from  $s \in SPEC$  that are **sound** and **exhaustive**
- **Sound**: whenever  $i \in IMPL$  fails a test case, then  $i \not\text{conf } s$
- **Exhaustive**: whenever  $i \in IMPL$  passes all test cases, then  $i \text{ conf } s$

# IOCO: Ingredients

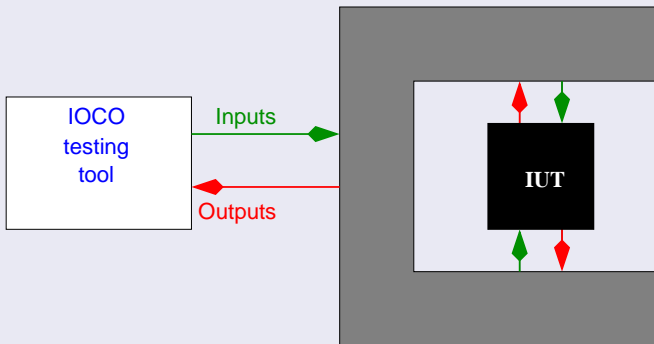
## Testing hypothesis

- Formulates the conditions under which the **theoretical results** hold for **real** implementations

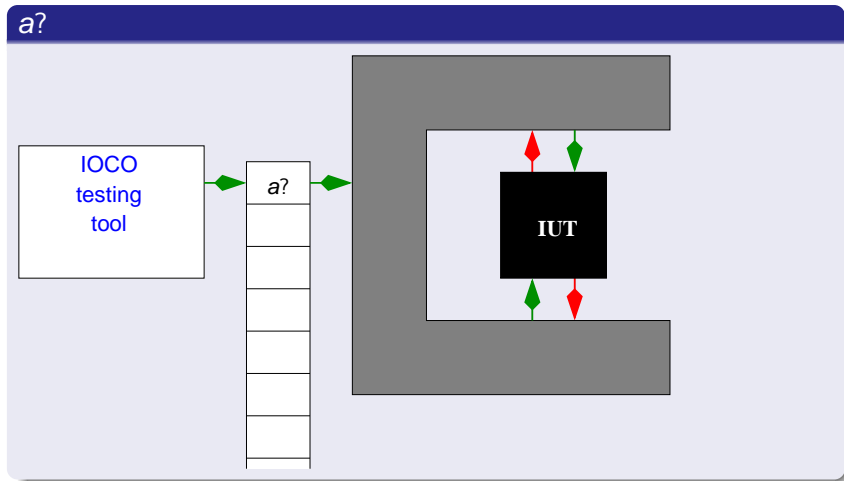
# IOCO in a nutshell

In a nutshell: it is all about **traces**

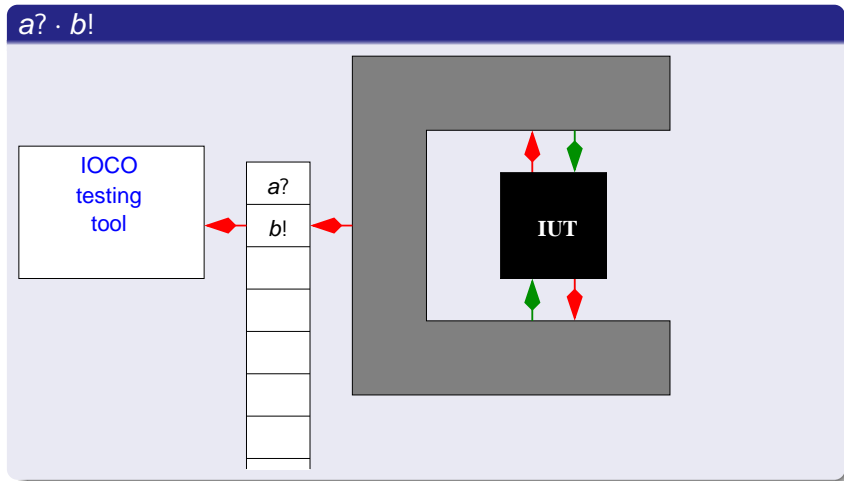
## The scenario



# Tester and IUT synthesize a trace



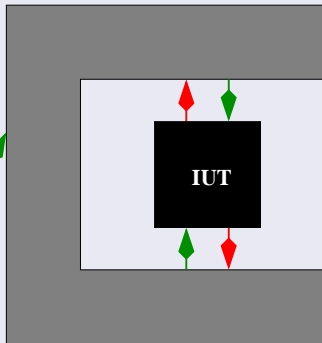
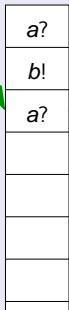
# Tester and IUT synthesize a trace



# Tester and IUT synthesize a trace

$a? \cdot b! \cdot a?$

IOCO  
testing  
tool

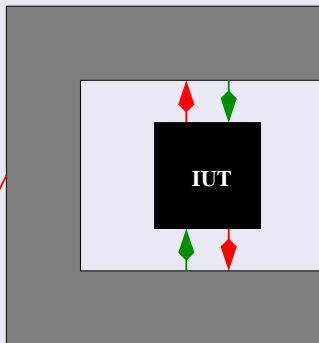


# Tester and IUT synthesize a trace

$a? \cdot b! \cdot a? \cdot b!$

IOCO  
testing  
tool

$a?$
$b!$
$a?$
$b!$

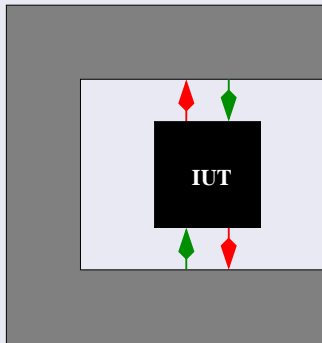


# Tester and IUT synthesize a trace

$a? \cdot b! \cdot a? \cdot b!$

IOCO  
testing  
tool

$a?$
$b!$
$a?$
$b!$



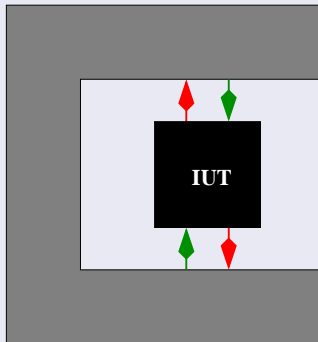
# Tester and IUT synthesize a trace

 $a? \cdot b! \cdot a? \cdot b! \cdot \delta$ 

IOCO  
testing  
tool

No output observed  
after certain time:  
Tester generates  
synthetic symbol  $\delta$   
(quiescence symbol)

$a?$
$b!$
$a?$
$b!$
$\delta$

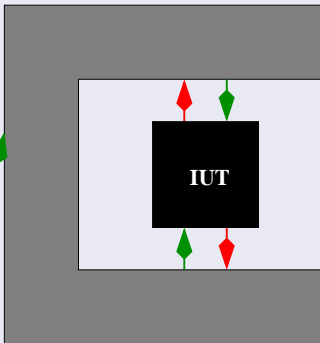


# Tester and IUT synthesize a trace

$a? \cdot b! \cdot a? \cdot b! \cdot \delta \cdot a?$

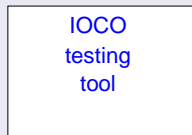
IOCO  
testing  
tool

$a?$
$b!$
$a?$
$b!$
$\delta$
$a?$



# Tester and IUT synthesize a trace

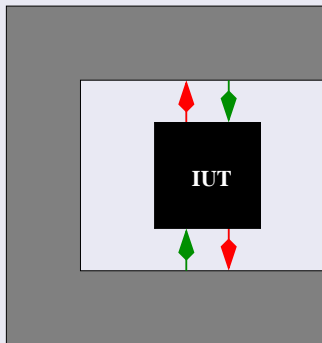
## Specification: input to testing tool



Specification  
(a set of traces)

A white rounded rectangular box containing the text "Specification (a set of traces)".

$a?$
$b!$
$a?$
$b!$
$\delta$
$a?$



# Tester and IUT synthesize a trace

## Correctness Criterion

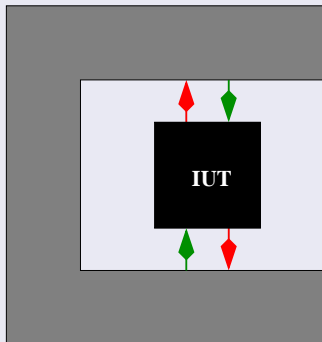


Specification  
(a set of traces)



test run

a?
b!
a?
b!
$\delta$
a?



# How does it work

## Basic ingredients

$Act_I$ : a set of input actions

$Act_U$ : a set of output actions

$\delta$ : The quiescence symbol

$Act, Act_\delta$ :  $Act = Act_I \cup Act_U$ ,  $Act_\delta = Act \cup \{\delta\}$

$S$ : The specification set of traces,  $S \subseteq Act_\delta^*$

# How does it work

## The correctness criterion

$\sigma$  is test-run synthesized by IOCO testing tool and IUT  $I$ :

if  $\sigma \notin S$ , then  $I$  does not conform to specification  $S$

# How does it work

## The testing tool

Let  $\sigma$  be test-run so far and  $\sigma \in S$

The testing tool has the choice:

- to apply an input, or
- to wait for an output, or
- to stop

# How does it work

## The testing tool

Let  $\sigma$  be test-run so far and  $\sigma \in S$

The testing tool has the choice:

- to apply an input, or
- to wait for an output, or
- to stop

## Applying an Input

only  $i? \in Act_I$  is chosen such that  $\sigma \cdot i? \in S$

# How does it work

## The testing tool

Let  $\sigma$  be test-run so far and  $\sigma \in S$

The testing tool has the choice:

- to apply an input, or
- to wait for an output, or
- to stop

## Waiting for output

only If IUT sends  $o! \in Act_U$ , then  $\sigma \cdot o! \notin S$  causes **test failure**

# How does it work

## The testing tool

Let  $\sigma$  be test-run so far and  $\sigma \in S$

The testing tool has the choice:

- to apply an input, or
- to wait for an output, or
- to stop

## Quiescence

If no output received for some time then  $\sigma \cdot \delta \notin S$  causes test failure

# How does it work

## The testing tool

Let  $\sigma$  be test-run so far and  $\sigma \in S$

The testing tool has the choice:

- to apply an input, or
- to wait for an output, or
- to stop

**Observe:** Only outputs can fail a test.

# How does it work

## The testing tool

Let  $\sigma$  be test-run so far and  $\sigma \in S$

The testing tool has the choice:

- to apply an input, or
- to wait for an output, or
- to stop

## Stop

Since  $\sigma \in S$ , testing stops with **PASS verdict**

# Questions to be answered.

## Specification

- What properties has a specification trace set?
- How to specify it?

## Correctness criterion

How to formalise the correctness criterion?

## Under which assumptions holds the theory for my real IUT?

Testing hypothesis.

## Test cases

How can we derive test cases automatically?