

Model-Based Testing

Ed Brinksmas

University of Twente
Dept. of Computer Science
Formal Methods & Tools group
Enschede
The Netherlands



ARTIST2 Summer School
Nässlingen



Contents

- introduction & background
- testing pre-orders
- input/output & quiescence
- ioco implementation relation
- test generation
- TorX
- test case study
- real-time testing

October 1, 2005

ARTIST2 Summer School 2



Contents

- introduction & background
- testing pre-orders
- input/output & quiescence
- ioco implementation relation
- test generation
- TorX
- test case study
- real-time testing

October 1, 2005

ARTIST2 Summer School 3



Practical problems of testing

Testing is:

- important
- much practiced
- 30% - 50% of project effort
- expensive
- time critical
- not constructive (but sadistic?)

But also:

- ad-hoc, manual, error-prone
- hardly theory / research
- no attention in curricula
- not cool:
"if you're a bad programmer
you might be a tester"

Attitude is changing:

- more awareness
- more professional

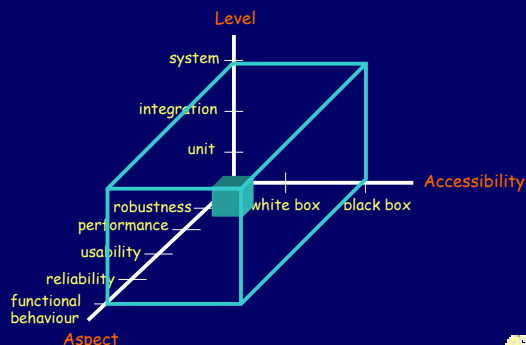
Improvements possible
with formal methods! ?

October 1, 2005

ARTIST2 Summer School 4



Types of Testing



October 1, 2005

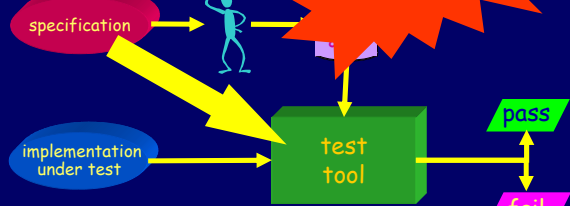
ARTIST2 Summer School 5



Test Automation

Traditional test automation
= tools to execute and manage

Why not generate
test automatically!



October 1, 2005

ARTIST2 Summer School 6





Verification and Testing

Verification :

- formal manipulation
- prove properties
- performed on model

Testing :

- experimentation
- show error
- concrete system

formal world



concrete world

Verification is only as good as the validity of the model on which it is based.

Testing can only show the presence of errors, not their absence.

October 1, 2005

ARTIST2 Summer School 7



Testing with Formal Methods

- Testing with respect to a formal specification
- Precise, formal definition of correctness : good and unambiguous basis for testing
- Formal validation of tests
- Algorithmic derivation of tests : tools for automatic test generation
- Allows to define measures expressing coverage and quality of testing

October 1, 2005

ARTIST2 Summer School 8



Challenges of Testing Theory

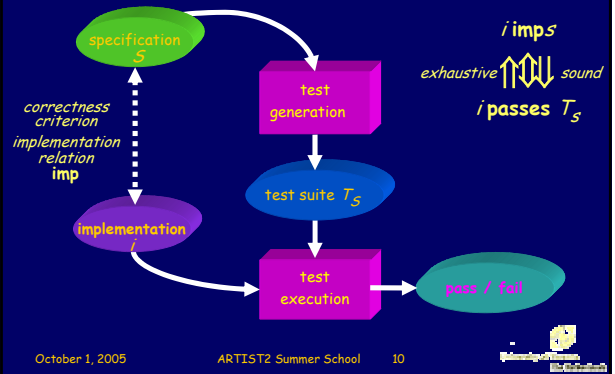
- Infinity of testing:
 - too many possible input combinations -- infinite breadth
 - too many possible input sequences -- infinite depth
 - too many invalid and unexpected inputs
- Exhaustive testing never possible:
 - when to stop testing ?
 - how to invent effective and efficient test cases with high probability of detecting errors ?
- Optimization problem of testing yield and invested effort
 - usually stop when time is over

October 1, 2005

ARTIST2 Summer School 9



Formal Testing

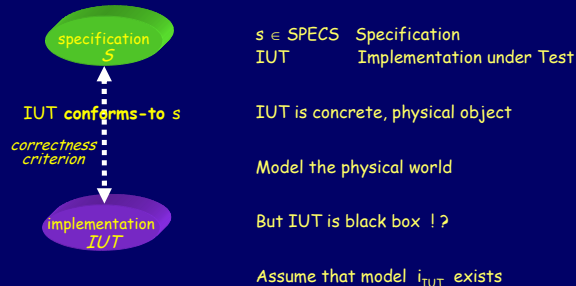


October 1, 2005

ARTIST2 Summer School 10



Formal Testing : Conformance



October 1, 2005

ARTIST2 Summer School 11



Contents

- introduction & background
- testing pre-orders
- input/output & quiescence
- ioco implementation relation
- test generation
- TorX
- test case study
- real-time testing

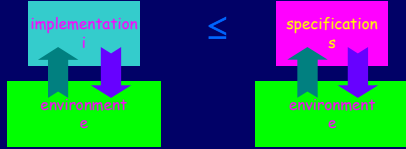
October 1, 2005

ARTIST2 Summer School 12





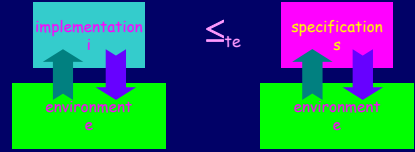
Testing Preorders on Transition Systems



For all environments e
 $i \leq s \iff \forall e \in Env. \text{obs}(e, i) \subseteq \text{obs}(e, s)$
 all observations of an implementation i in e should be explained by observations of the specification s in e .



Classical Testing Preorder

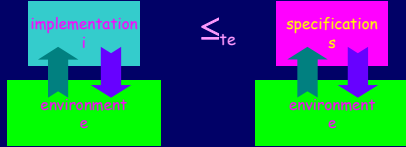


$$i \leq_{te} s \iff \forall e \in E. \text{obs}(e, i) \subseteq \text{obs}(e, s)$$

\downarrow \downarrow
 LTS(L) Deadlocks(e||s)



Classical Testing Preorder



$$i \leq_{te} s \iff \forall e \in LTS(L). \forall \sigma \in L^* .$$

$\{ \sigma \mid \text{deadlocks after } \sigma \} \subseteq \{ \sigma \mid e \parallel s \text{ deadlocks after } \sigma \}$
 $\iff FP(i) \subseteq FP(s)$
 $FP(p) = \{ \langle \sigma, A \rangle \mid p \text{ after } \sigma \text{ refuses } A \}$



Quirky Coffee Machine [Langerak]

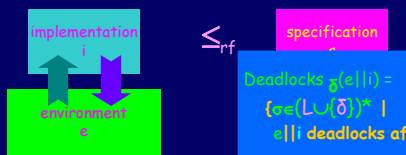
Can we distinguish between these machines?



They are testing equivalent!



Refusal Preorder



$$i \leq_{rf} s \iff \forall e \in E. \text{obs}(e, i) \subseteq \text{obs}(e, s)$$

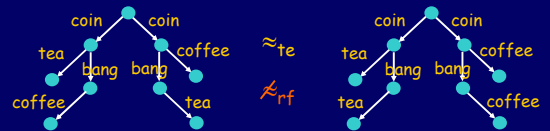
\downarrow \downarrow
 LTS(L ∪ {δ}) Deadlocks_{δ}(e||i)

e observes with δ deadlock on all alternative actions

$\text{Deadlocks}_{\delta}(e||i) = \{ \sigma \mid \text{deadlocks after } \sigma \}$
 $\{ \sigma \mid e \parallel i \text{ deadlocks after } \sigma \}$



Quirky Coffee Machine Revisited



δ only enabled if coffee is not





Contents

- introduction & background
- testing pre-orders
- **input/output & quiescence**
- ioco implementation relation
- test generation
- TorX
- test case study
- real-time testing



I/O Transition Systems

- testing actions are usually directed, i.e. there are inputs and outputs
 $L=L_{in} \cup L_{out}$ with $L_{in} \cap L_{out} = \emptyset$
- systems can always accept all inputs (input enabledness)
for all states s , for all $a \in L_{in}$ $s \xrightarrow{a}$
- testers are I/O systems
 - output (stimulus) is input for the SUT
 - input (response) is output of the SUT



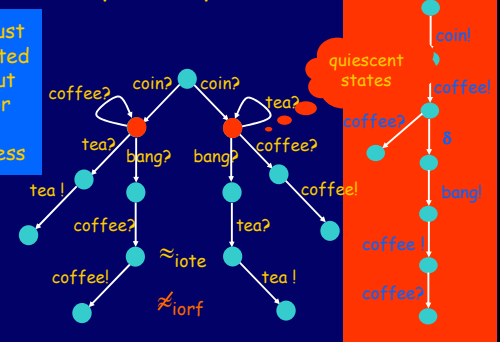
Quiescence

- Because of input enabledness $S \parallel T$ deadlocks iff T produces no stimuli and S no responses. This is known as quiescence
- Observing quiescence leads to two implementation relations for I/O systems I and S :
 1. $I \leq_{iote} S$ iff for all I/O testers T :
 - $Deadlocks(I \parallel T) \subseteq Deadlocks(S \parallel T)$ (quiescence)
 2. $I \leq_{iorf} S$ iff for all I/O testers T :
 - $Deadlocks_{\delta}(I \parallel T) \subseteq Deadlocks_{\delta}(S \parallel T)$ (repetitive quiescence)



Input-Output QCM

states must be saturated with input loops for input enabledness



Contents

- introduction & background
- testing pre-orders
- input/output & quiescence
- **ioco implementation relation**
- test generation
- TorX
- test case study
- real-time testing



Implementation Relation ioco

By adding a transition $p \xrightarrow{\delta} p$ to every quiescent state of a system we treat quiescence as an observable (synchronizable) action:

$$i \leq_{iorf} s \Leftrightarrow \forall \text{ I/O tests } T: Deadlocks_{\delta}(i \parallel T) \subseteq Deadlocks_{\delta}(s \parallel T)$$

$$\Leftrightarrow \forall \sigma \in (L \cup \{\delta\})^*: out(i \text{ after } \sigma) \subseteq out(s \text{ after } \sigma)$$

To allow under-specification we restrict the set of traces:

$$i \text{ ioco } s \Leftrightarrow \forall \sigma \in Traces_{\delta}(s): out(i \text{ after } \sigma) \subseteq out(s \text{ after } \sigma)$$





Implementation Relation ioco

Correctness expressed by implementation relation ioco:

$$i \text{ ioco } s \stackrel{\text{def}}{=} \forall \sigma \in \text{Traces}_g(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)$$

Intuition:

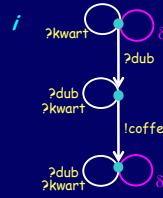
i ioco-conforms to s , iff

- if i produces output x after trace σ , then s can produce x after σ
- if i cannot produce any output after trace σ , then s cannot produce any output after σ (quiescence δ)



Implementation Relation ioco

$$i \text{ ioco } s \stackrel{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)$$

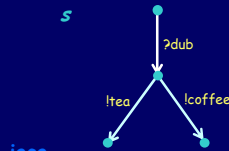
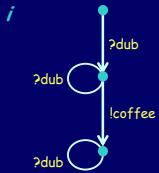


$\text{out}(i \text{ after } \varepsilon)$	$= \{ \delta \}$
$\text{out}(i \text{ after } ?dub)$	$= \{ lcoffee \}$
$\text{out}(i \text{ after } ?dub.?dub)$	$= \{ lcoffee \}$
$\text{out}(i \text{ after } ?dub.lcoffee)$	$= \{ \delta \}$
$\text{out}(i \text{ after } ?kwart)$	$= \{ \delta \}$
$\text{out}(i \text{ after } lcoffee)$	$= \emptyset$
$\text{out}(i \text{ after } ?dub.ltea)$	$= \emptyset$
$\text{out}(i \text{ after } \delta)$	$= \{ \delta \}$



Implementation Relation ioco

$$i \text{ ioco } s \stackrel{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)$$



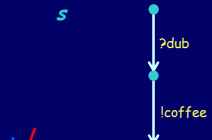
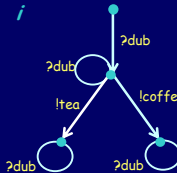
ioco

$$\text{out}(i \text{ after } ?dub) = \{ lcoffee \} \quad \text{out}(s \text{ after } ?dub) = \{ lcoffee, ltea \}$$



Implementation Relation ioco

$$i \text{ ioco } s \stackrel{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)$$



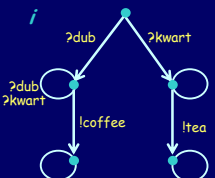
~~ioco~~

$$\text{out}(i \text{ after } ?dub) = \{ lcoffee, ltea \} \not\subseteq \text{out}(s \text{ after } ?dub) = \{ lcoffee \}$$



Implementation Relation ioco

$$i \text{ ioco } s \stackrel{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)$$



ioco

$$\begin{aligned} \text{out}(i \text{ after } ?dub) &= \{ lcoffee \} & \text{out}(s \text{ after } ?dub) &= \{ lcoffee \} \\ \text{out}(i \text{ after } ?kwart) &= \{ ltea \} & \text{out}(s \text{ after } ?kwart) &= \emptyset \end{aligned}$$

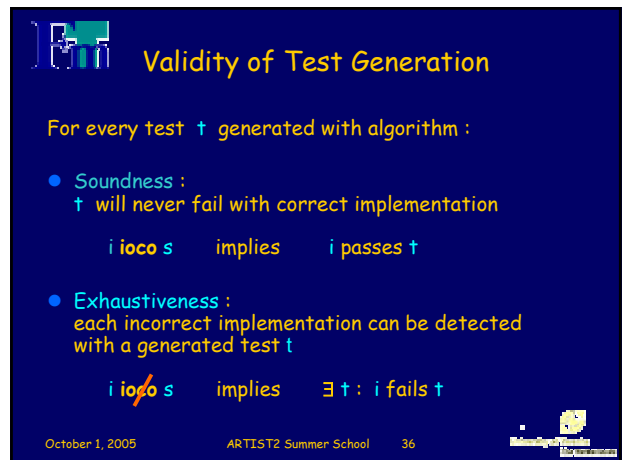
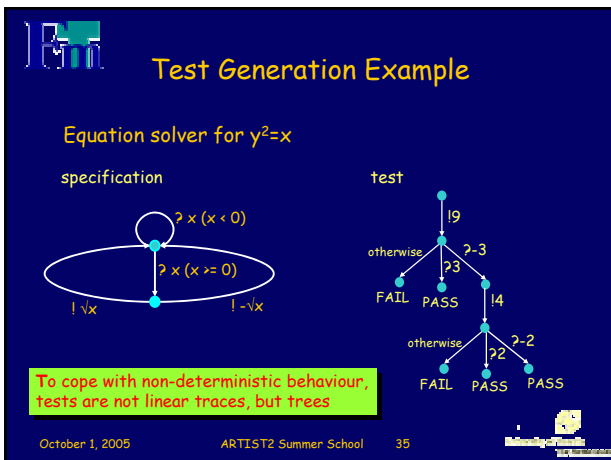
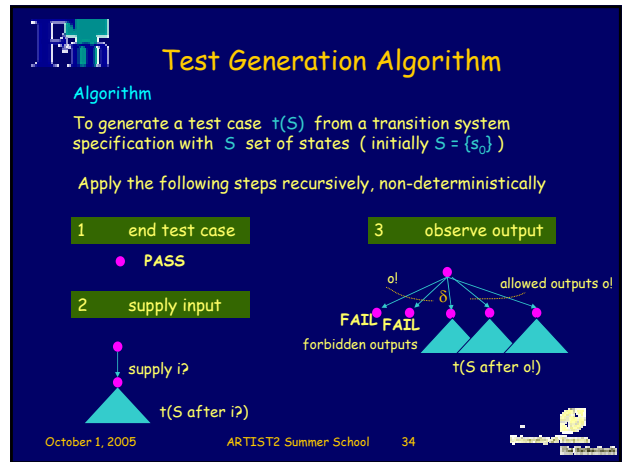
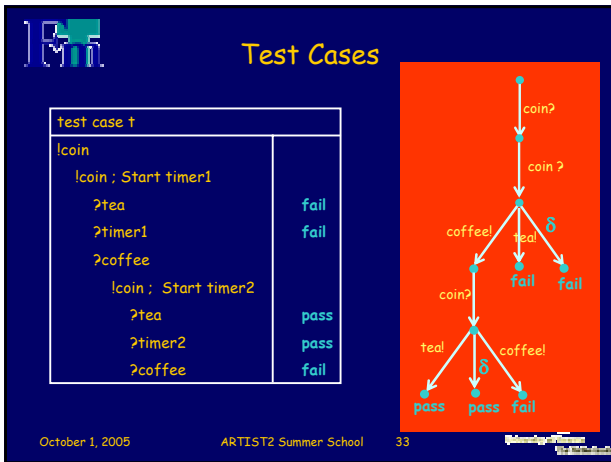
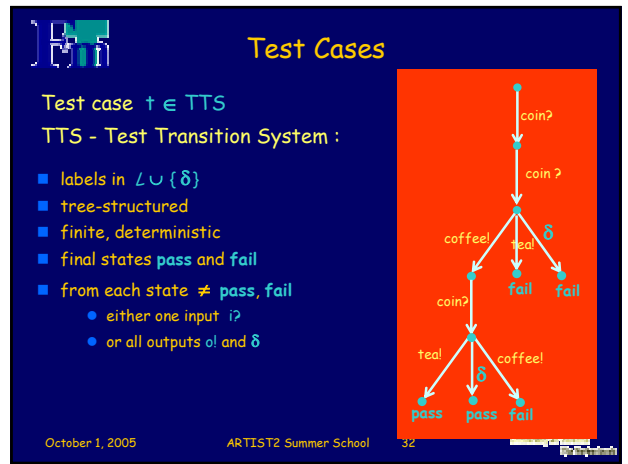
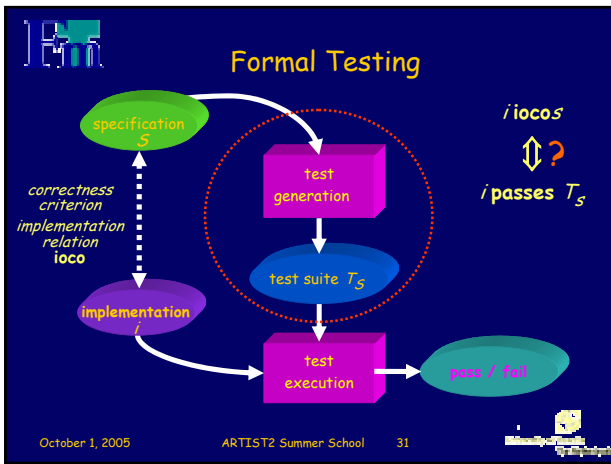
But $?kwart \notin \text{Traces}_g(s)$



Contents

- introduction & background
- testing pre-orders
- input/output & quiescence
- ioco implementation relation
- **test generation**
- TorX
- test case study
- real-time testing





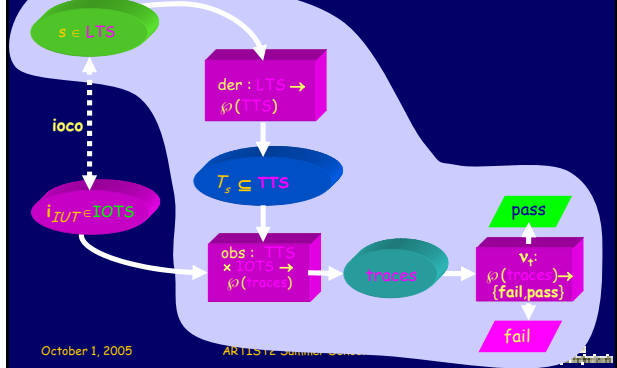


Contents

- introduction & background
- testing pre-orders
- input/output & quiescence
- ioco implementation relation
- test generation
- **TorX**
- test case study
- real-time testing



Formal Testing with Transition Systems



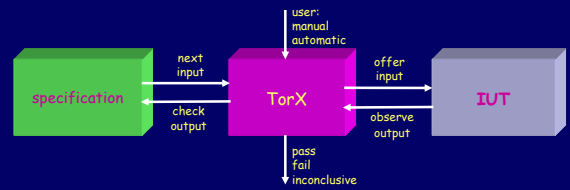
Test Generation Tools for ioco

- **TVEDA** (CNET - France Telecom)
 - derives TTCN tests from single process SDL specification
 - developed from practical experiences
 - implementation relation $R1 \approx ioco$
- **TGV** (IRISA - Rennes)
 - derives tests in TTCN from LOTOS or SDL
 - uses test purposes to guide test derivation
 - implementation relation: unfair extension of $ioco$
- **TestComposer**
 - Combination of TVEDA and TGV in ObjectGeode
- **TestGen** (Stirling)
 - Test generation for hardware validation
- **TorX** (Côte de Resysste)

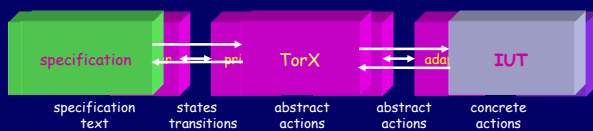


A Test Tool : TorX

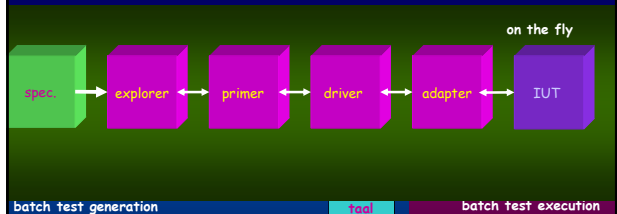
- On-the-fly test generation and test execution
- Implementation relation: **ioco**
- Specification languages: LOTOS, Promela, FSP, Automata



TorX Tool Architecture



On-the-Fly ↔ Batch Testing



On-the-Fly Testing

specification

implementation

October 1, 2005 ARTIST2 Summer School 43

TorX

October 1, 2005 ARTIST2 Summer School 44

Contents

- introduction & background
- testing pre-orders
- input/output & quiescence
- ioco implementation relation
- test generation
- TorX
- test case study
- real-time testing

October 1, 2005 ARTIST2 Summer School 45

TorX Case Studies

- Conference Protocol academic
- EasyLink TV-VCR protocol Philips
- Cell Broadcast Centre component CMG
- Road Toll Payment Box protocol Interpay
- V5.1 Access Network protocol Lucent
- Easy Mail Melder CMG
- FTP Client academic
- "Oosterschelde" storm surge barrier-control CMG
- TANGRAM: testing VLSI lithography machine ASML

October 1, 2005 ARTIST2 Summer School 46

Interpay Highway Tolling System

October 1, 2005 ARTIST2 Summer School 47

Highway Tolling Protocol

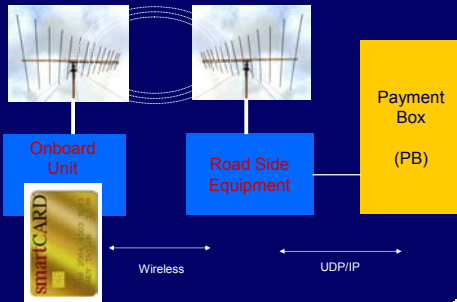
Characteristics :

- Simple protocol
- Parallellism : many cars at the same time
- Encryption
- System passed traditional testing phase

October 1, 2005 ARTIST2 Summer School 48



Highway Tolling System

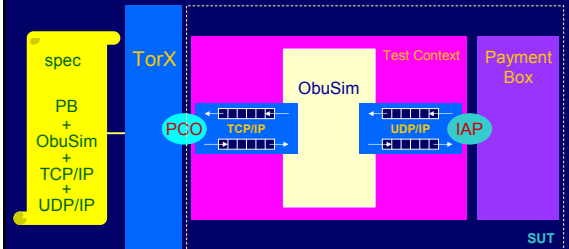


October 1, 2005

ARTIST2 Summer School 49



Highway Tolling: Test Architecture



October 1, 2005

ARTIST2 Summer School 50



Highway Tolling: Results

- Test results :
 - 1 error during validation (design error)
 - 1 error during testing (coding error)
- Automated testing :
 - beneficial: high volume and reliability
 - many and long tests executed (> 50,000 test events)
 - very flexible: adaptation and many configurations
- Real-time :
 - interference computation time-on-the-fly testing
 - interference quiescence and time-outs
- Step ahead in formal testing of realistic systems

October 1, 2005

ARTIST2 Summer School 51



Contents

- introduction & background
- testing pre-orders
- input/output & quiescence
- ioco implementation relation
- test generation
- TorX
- test case study
- real-time testing

October 1, 2005

ARTIST2 Summer School 52



RT TorX Hacking Approaches

1. Ignore RT functionality:
 - test pure functional behaviour
 - analyse timing requirements using TorX log files & assumed timing constraints
2. Add timestamps to observations
 - adapter adds timestamps to observations when they are made and passed on to the driver
 - timestamps are used to analyse TorX log files
3. Add timestamps to stimuli & observations
 - adapter add timestamps to observations when they are made and passed on to the driver
 - adapter adds timestamps to stimuli when they are applied and returned to the driver
 - analysis:
 - a. timing error logging: observed errors are written to TorX log file
 - b. timing error failure: observed errors cause fail verdict of test case

October 1, 2005

ARTIST2 Summer School 53



Real-time Testing and I/O Systems

- can the notion of repetitive quiescence be combined with real-time testing?
- is there a well-defined and useful conformance relation that allows sound and (relative) complete test derivation?
- can the TorX test tool be adapted to support Real-timed conformance testing?

October 1, 2005

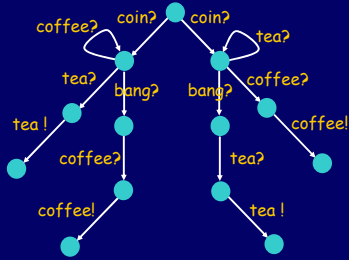
ARTIST2 Summer School 54





Do We Still Need Quiescence?

Yes!
the example processes should also be distinct in a real-time context



Real-Time and Quiescence

- s is quiescent iff: for no output action a and delay d : $s \xrightarrow{a(d)}$
- special transitions: $s \xrightarrow{\delta} s$ for every quiescent system state s
- testers observing quiescence take time: $Test_M$: set of test processes having only $\delta(M)$ -actions to observe quiescence
- assume that implementations are M -quiescent: for all reachable states s and s' : if $s \xrightarrow{\delta(M)} s'$ then s' is quiescent



Real-Time and Quiescence

$$i \leq_{\text{tiortf}}^M s \iff \forall T \in \text{Test}_M: \text{Deadlocks}_\delta(i || T) \subseteq \text{Deadlocks}_\delta(s || T)$$

$$\iff \forall \sigma \in (L \cup \{\delta(M)\})^*: \text{out}_M(i \text{ after } \sigma) \subseteq \text{out}_M(s \text{ after } \sigma)$$

$$i \text{ tiocom } s \iff \forall \sigma \in \text{Traces}_{\delta(M)}(s): \text{out}_M(i \text{ after } \sigma) \subseteq \text{out}_M(s \text{ after } \sigma)$$



Properties

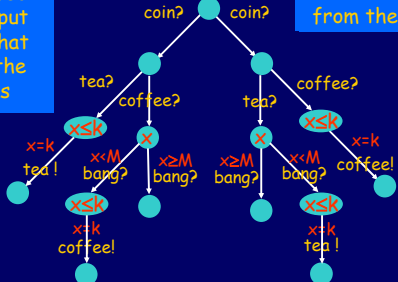
1. for all $M_1 \leq M_2$: $i \leq_{\text{tiortf}}^{M_2} s$ implies $i \leq_{\text{tiortf}}^{M_1} s$
2. for all time-independent i, s and $M_1, M_2 \geq 0$: $i \leq_{\text{tiortf}}^{M_1} s$ iff $i \leq_{\text{tiortf}}^{M_2} s$ iff $i \leq_{\text{iorf}} s$



A limitation

states are saturated with input loops that reset the clocks

this process cannot be distinguished from the previous

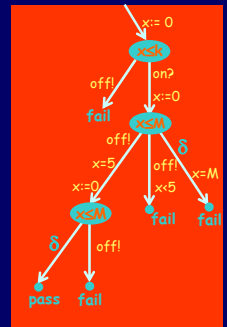


Test Cases

Test case $\dagger \in TTA$

TTA - Test Timed Automata :

- labels in $L \cup \{\delta\}, \mathcal{G}(d)$
- tree-structured
- finite, deterministic
- final states **pass** and **fail**
- from each state \neq **pass**, **fail**
 - choose an input $i?$ and a time k and wait for the time k accepting all outputs $o!$ and after k time unit provide input $i?$
 - or wait for time M accepting all outputs $o!$ and δ





Timed test Gen

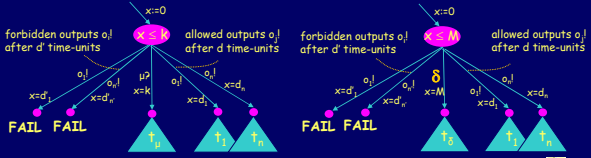
To generate a test case $t(S)$ from specification with S set of states apply the following steps recursive

can be calculated effectively only for subclasses of timed transition systems!

1. end test case ● PASS

2. choose $k \in (0, M)$ and input μ

3. wait for observing possible output



October 1, 2005

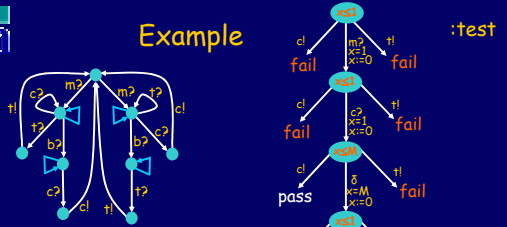
ARTIST2 Summer School

61

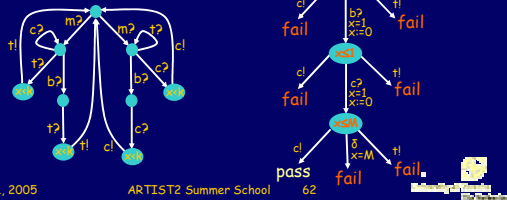


Example

spec:
 δ



impl:
 $M=k$



October 1, 2005

ARTIST2 Summer School

62



Soundness & Completeness

- the non-timed generation algorithm can be shown to generate **sound** real-time test cases
- test generation is **complete** for every erroneous trace it can generate a test that exposes it
- test generation is **not limit complete** because of continuous time there are uncountably many traces and only countably many test are generated by repeated runs
- test generation is **almost limit complete** repeated test generation runs will eventually generate a test case that will expose **one of the non-spurious errors** of a non-conforming implementation

non-spurious errors = errors with a positive probability of occurring

October 1, 2005

ARTIST2 Summer School

63



Current Work

- Extension of the framework
 - M as a function of the specification state/output channel
 - integration with symbolic data generation
 - test action refinement
 - robustness & tolerance in real-time testing
- Extending TorX environment using CORBA IDL
 - generate abstract TorX actions
 - generate TTCN-3 signatures
 - generate adapter code
- Practical application
 - TANGRAM project: testing control software for VLSI lithography machines (ASML)
 - smooth transition between timed & untimed testing

October 1, 2005

ARTIST2 Summer School

64



Future Work

- stochastic systems
- quality of service
- hybrid systems
- coverage measures
- integration white/black box spectrum
- ...

October 1, 2005

ARTIST2 Summer School

65



For more information

fmt.cs.utwente.nl/research/testing

October 1, 2005

ARTIST2 Summer School

66

