Formal Conformance Testing 2006

Lecture 1

14th Sep 2006

Welcome!

- This is T-79.5304: Formal Conformance Testing
- Lectures from 10 to 12 am, no regular tutorials
- Cancellations and other notes at the web page (go to http://www.tcs.hut.fi/)

Copyright © Antti Huima 2004-06. All Rights Reserved.

Lecturer

- Antti Huima = me
- “Special teacher” = not member of HUT staff
- Work: Managing Director at Conformiq Software

Copyright © Antti Huima 2004-06. All Rights Reserved.

Practical Matters

- Website contains all important information
- The news group can be used for discussion, but I will not follow it
- Lecture notes will be printed and distributed by the lecture note print service

Copyright © Antti Huima 2004-06. All Rights Reserved.

Organization

- Lectured in Finnish
- All written material in English
- Examination material = lecture notes
- To pass the course you must pass the examination

Copyright © Antti Huima 2004-06. All Rights Reserved.

Testing

- Testing is the process of
  1. interacting with a system, and
  2. evaluating the results, in order to
  3. determine if the system conforms to its specification
- In testing setup, the system is known as the system under test (SUT)

Copyright © Antti Huima 2004-06. All Rights Reserved.

“Interacting”

- If you can’t interact with a system, the system is uninteresting
- Interacting covers anything you can do with the system

Copyright © Antti Huima 2004-06. All Rights Reserved.

“Conforms”

- Interaction does not imply judgement
- Conformance = “correspondence in form or appearance”
- Conformance to a specification = “works as specified”
- “Were the results of the interaction allowed by the specification?”

Copyright © Antti Huima 2004-06. All Rights Reserved.

“Formal”

- Formal = “according to a form”
- Here: testing is based on a mathematical, formalized foundation
- Not: testing based on a rigorous process where you need to fill lots of bureaucratic forms
- Also: “formal methods based”, but this is very vague

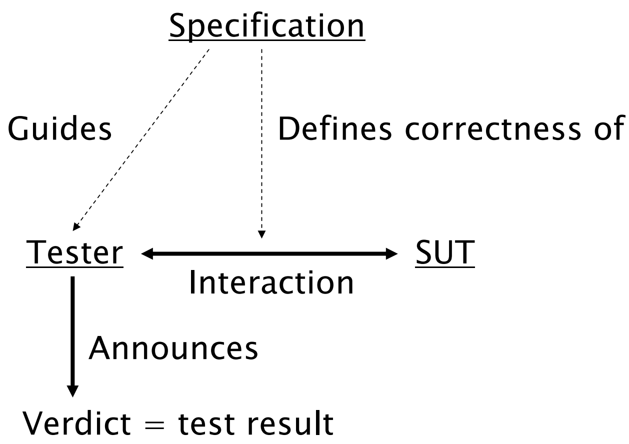
Copyright © Antti Huima 2004-06. All Rights Reserved.

Operational specification

- Specifies how a system should work
- Operational = describes behaviour, not e.g. usability scores
- Operational: “after 3 s, system must respond with X”
- Non-operational: “users must like the stuff”
- From now on just “specification” (assume operational)

Copyright © Antti Huima 2004-06. All Rights Reserved.

FCT setup



Copyright © Antti Huima 2004-06. All Rights Reserved.

Tester

- Tester has two functions:
  - Interact = generate behaviour
  - Give verdict = judge behaviour
- These two functions can be separated

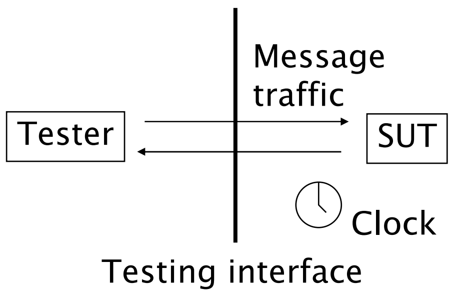
Copyright © Antti Huima 2004-06. All Rights Reserved.

Verdicts

- Typical verdicts:
  - PASS = system behaved ok
  - FAIL = system behaved badly
  - ERROR = tester messed up
  - (Often in the literature also "inconclusive verdict"—we return to this later)

Copyright © Antti Huima 2004-06. All Rights Reserved.

Testing interface



Copyright © Antti Huima 2004-06. All Rights Reserved.

Testing interface

- All interaction happens through the testing interface
- Bidirectional message passing
- All transmissions have a time stamp
- Every event has a distinct time stamp (this simplifies the theory slightly)

Copyright © Antti Huima 2004-06. All Rights Reserved.

Directions

- Input
  - input to the SUT
  - output from the tester
- Output
  - output from the SUT
  - input to the tester
- "SUT's viewpoint"

Copyright © Antti Huima 2004-06. All Rights Reserved.

Alphabets

- Σ_{in} is the set of input messages
- Σ_{out} is the set of out messages
- Σ is the union of the two
- Messages "contain" their direction
- Alphabet = traditional name for a set of potential messages

Copyright © Antti Huima 2004-06. All Rights Reserved.

Events

- Event = message + a time stamp
- Thus, event = (member of Σ) + (nonnegative real number)
- Formally, set of events is $\Sigma \times [0,\infty)$
- E.g. <"hello world"$_{in}$, 1.4 s>

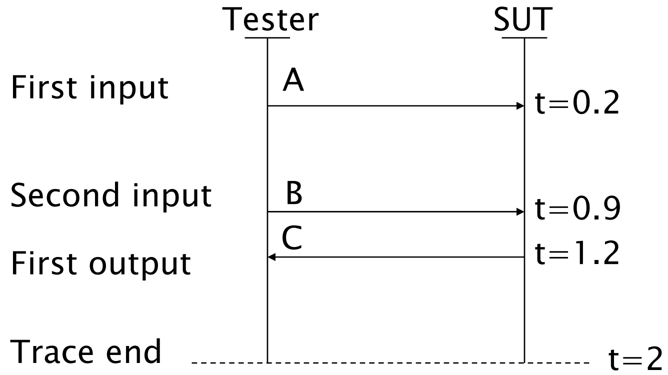
Copyright © Antti Huima 2004-06. All Rights Reserved.

Traces

- A trace denotes an observation of events for a certain time
- Trace = a finite set of events with distinct time stamps + end time stamp
- E.g. <{<"hello"$_{in,}$ 0.5>}, 0.8>

Copyright © Antti Huima 2004-06. All Rights Reserved.

Graphical sketch



<{<A,0.2>,<B,0.9>,<C,1.2>},2>

Copyright © Antti Huima 2004-06. All Rights Reserved.

lecture 1 summary

- Testing = interact + judge
- Specification, tester, SUT
- Testing interface = point of interaction
- Trace = a finite series of events observed during a finite time span

Copyright © Antti Huima 2004-06. All Rights Reserved.

Practical Matters

- 28th September there is no lecture because I am in Berlin
- Probably also no lecture on 26th October
- Consider ordering the lectures notes if not ordered yet

Copyright © Antti Huima 2004-06. All Rights Reserved.

Formal Conformance Testing 2006

Lecture 2

14th Sep 2006

Review of previous lecture

- Testing = interact + judge
- Specification, tester, SUT
- Testing interface = point of interaction
- Trace = a finite series of events observed during a finite time span

Copyright © Antti Huima 2004-06. All Rights Reserved.

Process notation

- We need a notation for "computational processes", i.e. a programming language to describe
  - implementations = SUTs
  - operational specifications as "reference implementations"
  - full testers
  - testing strategies = interaction strategies

Copyright © Antti Huima 2004-06. All Rights Reserved.

Requirements

- Support data, time, concurrency
- Familiar
- Compact
- Executable

Copyright © Antti Huima 2004-06. All Rights Reserved.

The choice

- UML statecharts and Java
- We have formal conformance testing tools for this choice, which is good

Copyright © Antti Huima 2004-06. All Rights Reserved.

Zero-time execution principle

- We assume that all UML/Java execution consumes zero time
- The only exception are timeouts and explicit waiting for asynchronous communications

Copyright © Antti Huima 2004-06. All Rights Reserved.

UML

- UML = Unified Modeling Language
- Standardized by OMG (Object Management Group)
- Current major version is UML 2

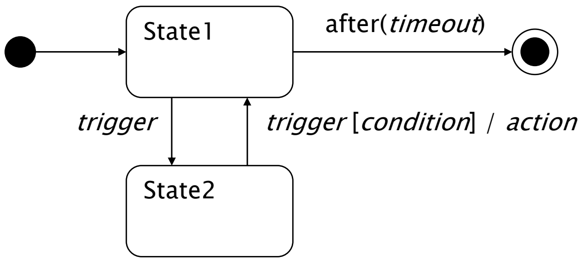
Copyright © Antti Huima 2004-06. All Rights Reserved.

UML State Charts

- Most common state chart components:
  - States
  - Transitions
  - Initial and final states

Copyright © Antti Huima 2004-06. All Rights Reserved.

Statechart Example

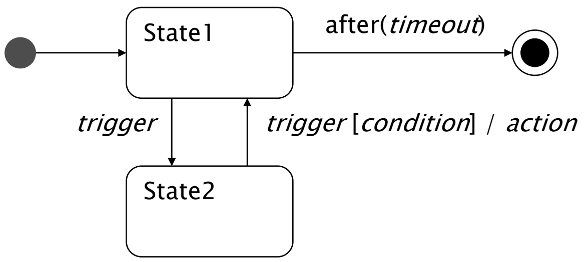


Copyright © Antti Huima 2004-06. All Rights Reserved.

Statechart Example

Initial state

This is where an object instance’s life begins.

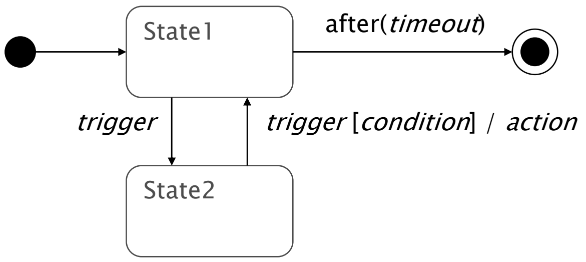


Copyright © Antti Huima 2004-06. All Rights Reserved.

Statechart Example

State

State represents a control state. The object instance can “stay” in a state waiting for an event. (This is not true of initial and final states).

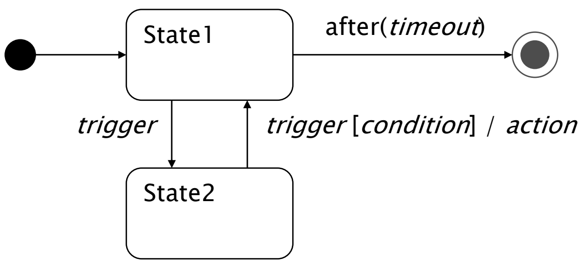


Copyright © Antti Huima 2004-06. All Rights Reserved.

Statechart Example

Final state

When an instance reaches its final state it finishes its behavior and dies.

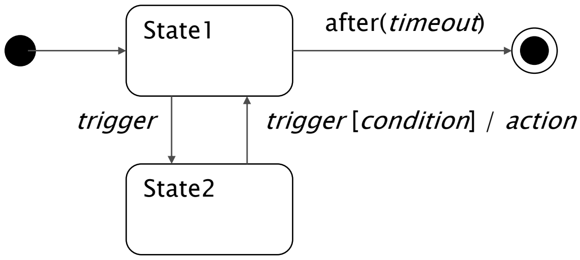


Copyright © Antti Huima 2004-06. All Rights Reserved.

Statechart Example

Transitions

Transitions can be “fired” then the instance is in the **source state**. Then the instance “moves” to the **destination state**.

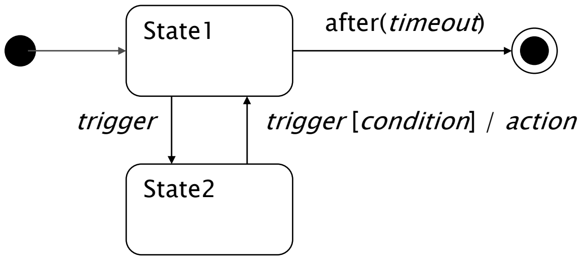


Copyright © Antti Huima 2004-06. All Rights Reserved.

Statechart Example

Spontaneous transition

Spontaneous transition is a transition without any **trigger**. It is fired “spontaneously” when the instance has arrived in the source state.

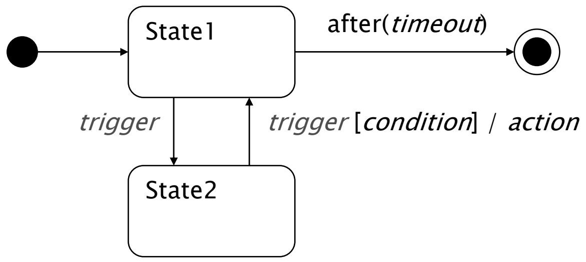


Copyright © Antti Huima 2004-06. All Rights Reserved.

Statechart Example

Trigger

A transition with a trigger is fired when the trigger "happens" and the instance is in the source state. Typically trigger is **message reception**.

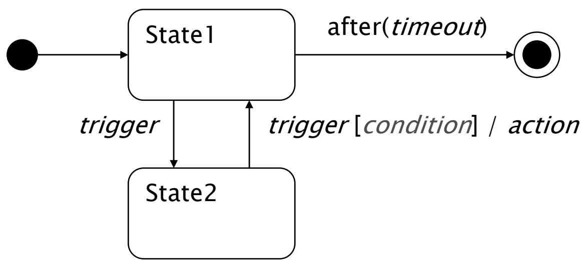


Copyright © Antti Huima 2004-06. All Rights Reserved.

Statechart Example

Condition

A transition that has a condition is fired only if the condition evaluates to true.

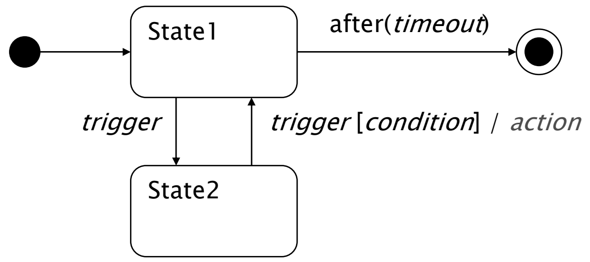


Copyright © Antti Huima 2004-06. All Rights Reserved.

Statechart Example

Action

A transition can have an action. It is code that is run when the transition is fired. In our case we write actions (and conditions) in Java.

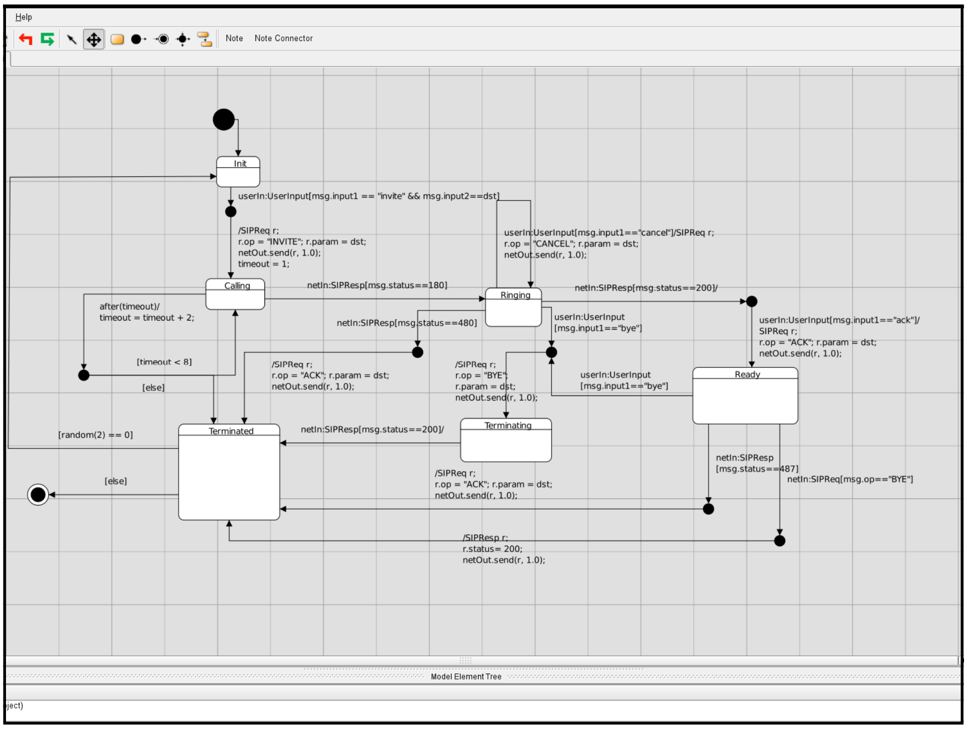


Copyright © Antti Huima 2004-06. All Rights Reserved.

Example

```
system
{
   Inbound userIn : UserInput;
   Inbound netIn : SIPResp, SIPReq;
   Outbound netOut : SIPResp, SIPReq;
}

class SIPClient extends StateMachine {
   public int timeout = 1;
   public String dst = "sip:192.168.0.1";
}

record UserInput
{
   public String input1;
}

record SIPResp
{
   public int status;
}

record SIPReq
{
   public String op;
   public String param;
}

void main()
{
   a = new SIPClient();
   t = new Thread(a);
   t.start();
}
```

Copyright © Antti Huima 2004-06. All Rights Reserved.



Formal Conformance Testing 2006

Lecture 5

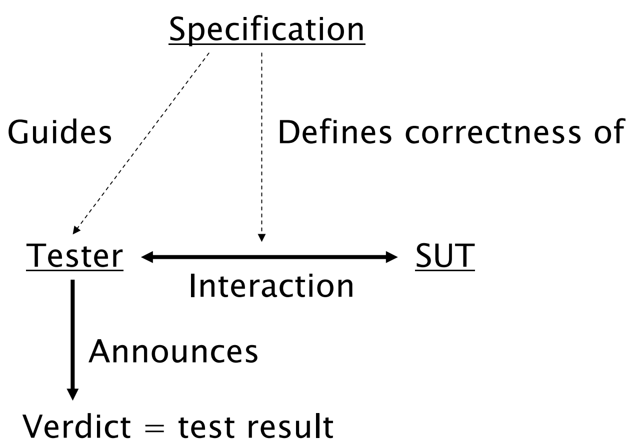
5th Oct 2006

Course this far

1	▸Introduction ▸General concepts ▸Traces
2	▸Java + UML

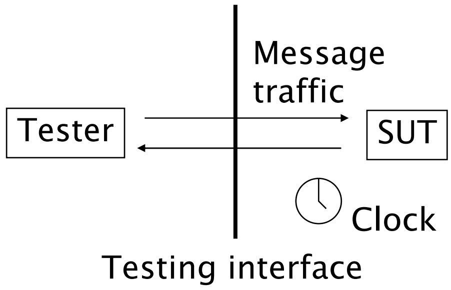
Copyright © Antti Huima 2004-06. All Rights Reserved.

FCT setup (replay)



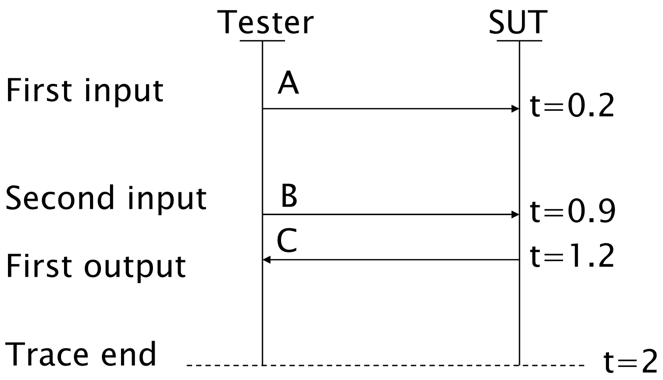
Copyright © Antti Huima 2004-06. All Rights Reserved.

Testing interface (replay)



Copyright © Antti Huima 2004-06. All Rights Reserved.

A trace (replay)



<{<A,0.2>,<B,0.9>,<C,1.2>},2>

Copyright © Antti Huima 2004-06. All Rights Reserved.

Traces

- Traces denote finitely long observations on the testing interface
- A trace contains a finite number of events and an end time stamp
- Traces are the *lingua franca* for discussing behaviours

Copyright © Antti Huima 2004-06. All Rights Reserved.

Traces

- Alphabet
- Event
- Trace
- Trace prefix
- Empty trace
- Trace extension
- Snapshot
- Difference time

Copyright © Antti Huima 2004-06. All Rights Reserved.

Alphabets

- Σ_{in} is the set of input messages
- Σ_{out} is the set of out messages
- Σ is the union of the two
- Messages "contain" their direction
- Alphabet = traditional name for a set of potential messages

Copyright © Antti Huima 2004-06. All Rights Reserved.

Events

- Event = message + a time stamp
- Thus, event = (member of Σ) + (nonnegative real number)
- Formally, set of events is $\Sigma \times [0,\infty)$
- E.g. $<$"hello world"$_{in}$, 1.4 s$>$

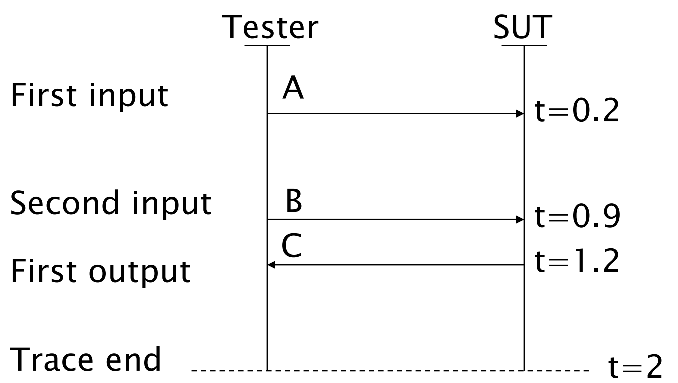
Copyright © Antti Huima 2004-06. All Rights Reserved.

Traces

- A trace denotes an observation of events for a certain time
- Trace = a finite set of events with distinct time stamps + end time stamp
- E.g. <{<“hello”$_{in}$, 0.5>}, 0.8>

Copyright © Antti Huima 2004-06. All Rights Reserved.

Graphical sketch



<{<A,0.2>,<B,0.9>,<C,1.2>},2>

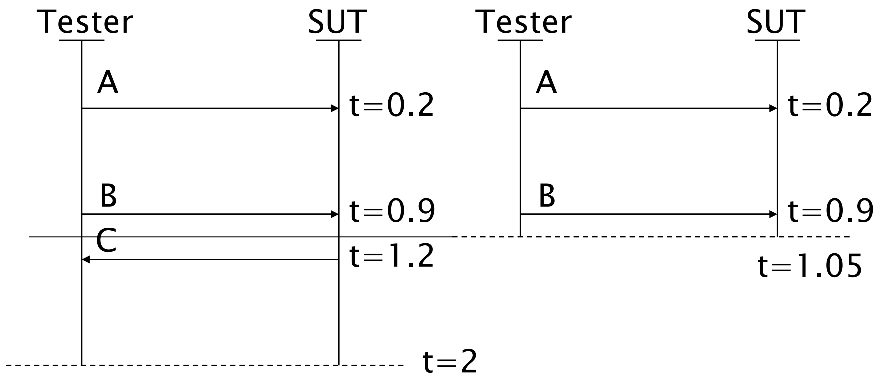
Copyright © Antti Huima 2004-06. All Rights Reserved.

Prefixes

- A trace is a prefix of another if the first trace can be extended <u>in time</u> to become the second one
- Let T and T' be traces
- T=<E,t> is a prefix of T'=<E',t'> (write $T \preccurlyeq T'$) if
  - $t \leq t'$ and
  - $E = \{ \langle \alpha, \kappa \rangle \mid \langle \alpha, \kappa \rangle \in E' \wedge \kappa < t \}$

Copyright © Antti Huima 2004-06. All Rights Reserved.

Prefix sketch



Copyright © Antti Huima 2004-06. All Rights Reserved.

Empty trace

- <∅,0> is the empty trace, denoted by ϵ
- The empty trace is a prefix of every other trace
- Empty trace has <u>no information content</u>

Copyright © Antti Huima 2004-06. All Rights Reserved.

Extensions

- A trace T is an extension of trace T' if the trace T' is a prefix of trace T
- Thus, being extension = <u>reverse</u> of being prefix

Copyright © Antti Huima 2004-06. All Rights Reserved.

Prefix set

- Pfx(T) is the set of all prefixes of T
- Pfx(T) = { T' | T' $\preccurlyeq$ T }
- Note: If T $\preccurlyeq$ T' then Pfx(T) $\subseteq$ Pfx(T')

Copyright © Antti Huima 2004-06. All Rights Reserved.

Snapshot

- Denote $\Sigma_\tau = \Sigma \cup \{\tau\}$
- Here τ is an object that does not belong to set Σ
- Let T = <E,t>
- Assume $\kappa < t$
- Denote by $T|_\kappa$ the event at time κ, or τ if no event at trace T has time stamp κ

Copyright © Antti Huima 2004-06. All Rights Reserved.

Snapshot example

- Suppose T = <{<A,1>}, 2>
- $T|_1 = A$
- $T|_{1.5} = \tau$
- $T|_2$ is not defined
- $T|_3$ is not defined

Copyright © Antti Huima 2004-06. All Rights Reserved.

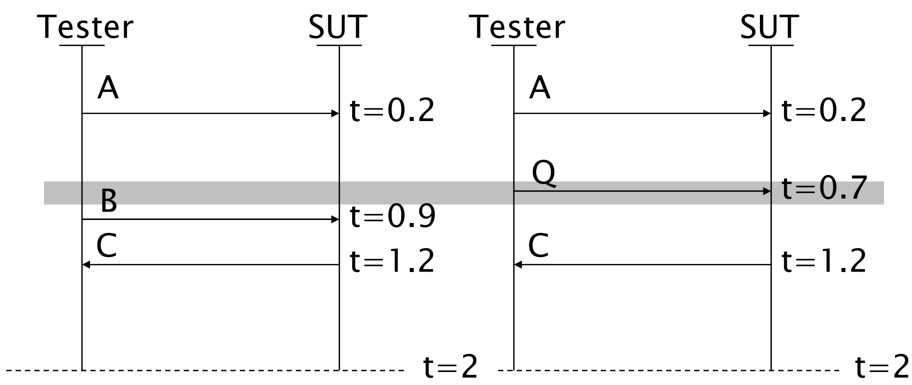
Difference time

- Suppose T and T’ are traces such that T is not a prefix T’ and T’ is not a prefix of T
- T and T’ are hence not equal
- Define

$$\Delta(T,T') = \min t^* : T|_{t^*} \neq T'|_{t^*}$$

Copyright © Antti Huima 2004-06. All Rights Reserved.

Difference time sketch



Copyright © Antti Huima 2004-06. All Rights Reserved.

Specifications

- Set of specifications
- Set of valid traces
- Prefix-completeness
- Seriality

Copyright © Antti Huima 2004-06. All Rights Reserved.

Set of specification

- $\mathbb{S}$ is a countable set of specifications
- Could be e.g.
  - Set of syntactically correct UML state charts
  - Set of valid English documents
- Structure not relevant
- Assume exists

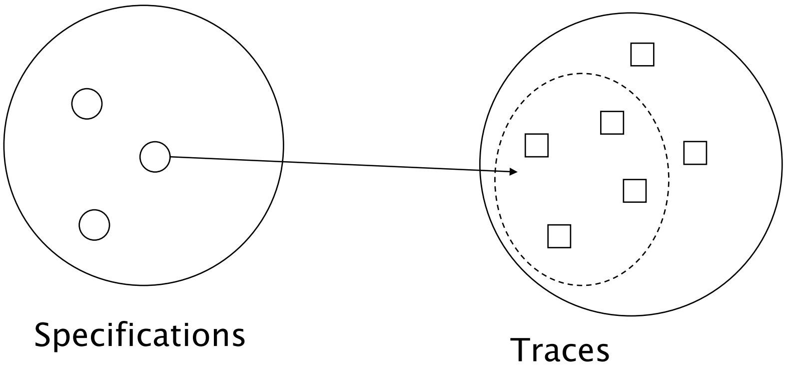
Copyright © Antti Huima 2004-06. All Rights Reserved.

Valid traces

- Every specification denotes a set of traces: the set of *valid traces*
- If S is a specification, Tr(S) is the set of valid traces for S
- Tr(S) must contain ϵ
- Tr(S) must be *prefix-complete*
- Tr(S) must be *serial*

Copyright © Antti Huima 2004-06. All Rights Reserved.

Specifications



Copyright © Antti Huima 2004-06. All Rights Reserved.

Prefix-completeness

- A set X of traces is prefix-complete if the following holds:
- If T $\in$ X and T' $\preccurlyeq$ T then also T' $\in$ X
- If a trace belongs to a prefix-complete set, then also all its prefixes belong to the set
- Why Tr(S) must be prefix-complete?

Copyright © Antti Huima 2004-06. All Rights Reserved.

Motivation for prefix-completeness

- Tr(S) denotes a set of *acceptable behaviours*
- Assume T is an acceptable behaviour
- Can you imagine a case where T', a prefix of T, would be not acceptable?

Copyright © Antti Huima 2004-06. All Rights Reserved.

Testing of Concurrent Systems 2004

Lecture 6

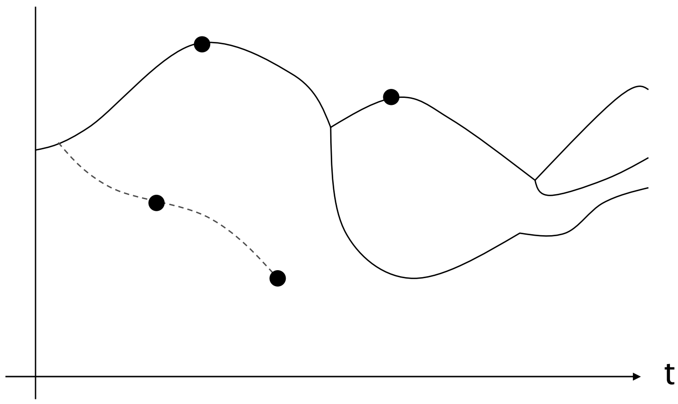
5th Oct 2004

Seriality

- A set X of traces is <u>serial</u> if for every <E, t> ∈ X and for every δ > 0 this holds:
- There exists <E', t+δ> ∈ X such that <E, t> ≼ <E', t+δ>
- Every trace of X has at least one arbitrarily long extension in X

Copyright © Antti Huima 2004-06. All Rights Reserved.

Seriality sketch



Copyright © Antti Huima 2004-06. All Rights Reserved.

Motivation for seriality

- Suppose non-serial Tr(S)
- There exists a valid trace T without an extension
- Let T' $\preccurlyeq$ T such that every far enough extension of T' contains T
- Is the behaviour T' acceptable?
- Why? And why not?

Copyright © Antti Huima 2004-06. All Rights Reserved.

Implementations

- We assume there exists a countable set of implementations, denoted by $\mathbb{I}$
- Could be e.g.
  - Set of all valid JAVA programs
  - Set of all valid C programs
  - Set of all functioning digital circuits

Copyright © Antti Huima 2004-06. All Rights Reserved.

Failure model

- Failure model links a specification to its potential implementations
- A failure model is a function

$$\mu: \mathbb{S} \to (\mathbb{I} \to [0,1])$$

- For every $s \in \mathbb{S}$, it holds that

$$\Sigma_i\ \mu(s)[i] = 1$$

- Hence $\mu(s)$ is a discrete probability distribution over implementations

Copyright © Antti Huima 2004-06. All Rights Reserved.

Use of failure models

- Failure model is a hypothesis about implementations and their potential defects
- Example: Boundary Value Pattern and the related failure model

Copyright © Antti Huima 2004-06. All Rights Reserved.

Testing strategies

- A testing strategy is a strategy on how to interact with an implementation
- Let S denote the set of all testing strategies
- What happens when a testing strategy is executed "against" an implementation?

Copyright © Antti Huima 2004-06. All Rights Reserved.

Execution

- Testing strategy + implementation yields a sequence of traces T_1, T_2, T_3,…
- Here $T_1 \preccurlyeq T_2 \preccurlyeq T_3 \preccurlyeq \ldots$
- These correspond to *test steps*
- Many different trace sequences are possible
- How do we formalize this?

Copyright © Antti Huima 2004-06. All Rights Reserved.

Semantic function ξ

- Maps implementation, testing strategy and "system state" to extensions of the currently observed trace
- Actually to a probability distribution of extensions
- System state = trace observed this far

Copyright © Antti Huima 2004-06. All Rights Reserved.

ξ function properties

- Gives a probability distribution
- Test steps are proper trace extensions
- Progressivity (non-Zenoness)
- Test steps are disjoint

Copyright © Antti Huima 2004-06. All Rights Reserved.

Signature

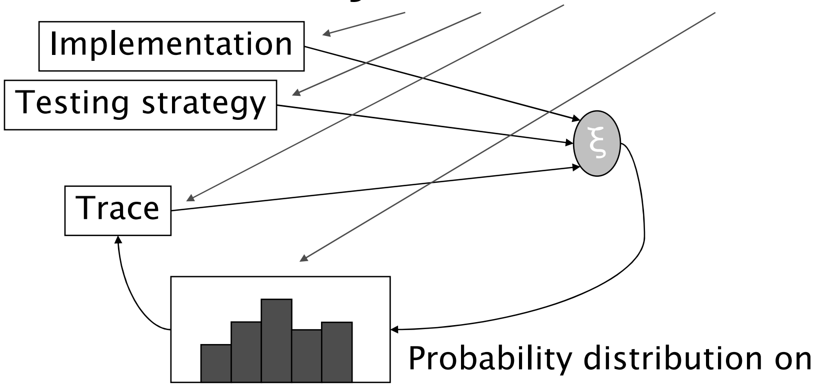
- Let $\mathcal{T}$ denote the set of all traces
- The signature is

$$\xi : \mathbb{I} \times \mathbb{S} \times \mathcal{T} \to (\mathcal{T} \to [0, 1])$$

Copyright © Antti Huima 2004-06. All Rights Reserved.

Execution

$$\xi : \mathbb{I} \times \mathbb{S} \times \mathcal{T} \to (\mathcal{T} \to [0, 1])$$



Copyright © Antti Huima 2004-06. All Rights Reserved.

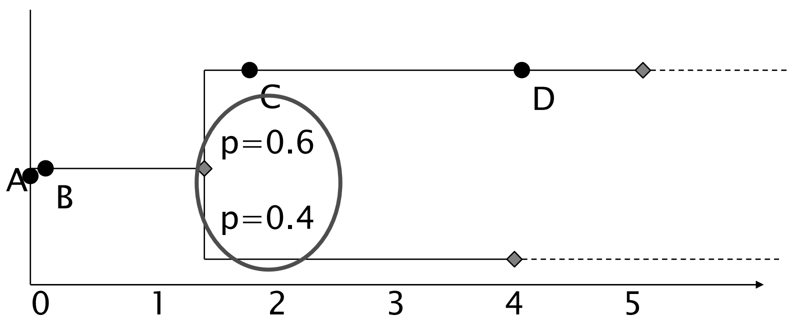
Gives probability distribution

- For all i, s and T, it must hold that
  $\Sigma_{T'} \xi(i,s,T)[T'] = 1$

Copyright © Antti Huima 2004-06. All Rights Reserved.

Gives probability distribution

- For all i, s and T, it must hold that
  $\Sigma_{T'} \xi(i,s,T)[T'] = 1$



Copyright © Antti Huima 2004-06. All Rights Reserved.

Test steps = proper trace extensions

- For all i, s, T and T' it must hold that

$$\xi(i,s,T)[T'] > 0 \Rightarrow T \prec T'$$

- Hence: every test step consumes time

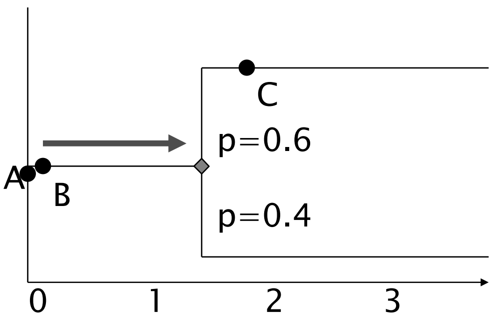
Copyright © Antti Huima 2004-06. All Rights Reserved.

Test steps = proper trace extensions

- For all i, s, T and T' it must hold that

$$\xi(i,s,T)[T'] > 0 \Rightarrow T \prec T'$$

- Hence: every test step consumes time



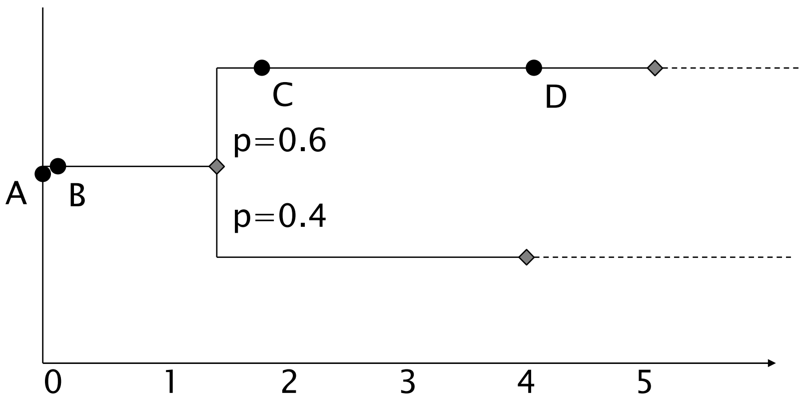
Copyright © Antti Huima 2004-06. All Rights Reserved.

Test step disjointness

- For any i, s, T, and T_1 and T_2 it must hold that if $T_1 \neq T_2$ and
- $\xi(i, s, T)[T_1] > 0$ and
- $\xi(i, s, T)[T_2] > 0$, then
- $T_1 \not\prec T_2$, and
- $T_2 \not\prec T_1$
- A technical convenience

Copyright © Antti Huima 2004-06. All Rights Reserved.

Execution sketch



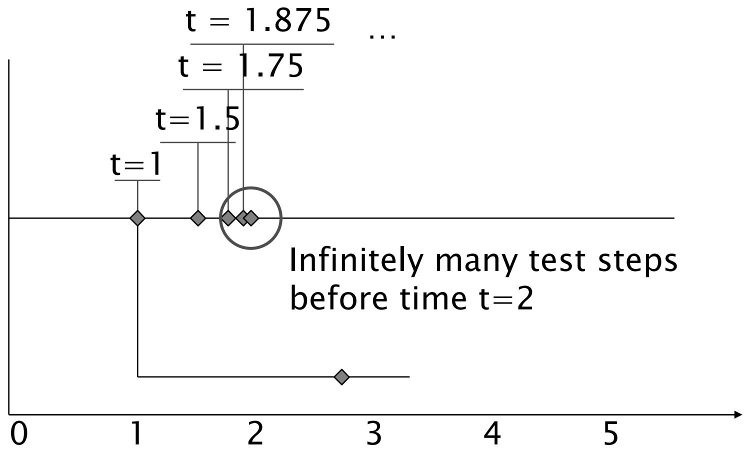
Copyright © Antti Huima 2004-06. All Rights Reserved.

Progressivity

- There does not exist an infinite sequence T_1, T_2, T_3, … and a constant $K \in \mathbb{R}$ such that
- $\xi(i, s, T_i)[T_{i+1}] > 0$ for all i, but such that for all $T_i = \langle E_i, t_i \rangle$ it holds that $t_i < K$.

Copyright © Antti Huima 2004-06. All Rights Reserved.

Non-progressivity sketch



Copyright © Antti Huima 2004-06. All Rights Reserved.