Model Based Testing of Embedded Systems

Brian Nielsen
Arne Skou

{bnielsen | ask}@cs.auc.dk
Agenda

- Time Requirements
- rt-ioco
- Offline test generation
  - time optimal
  - model coverage
- Online testing

Acknowledgements:
- Wang Yi
- Johan Bengtsson
- Gerd Behrmann
- Fredrik Larsson
- Alexandre David
- Emmanuel Fleury
- Patricia Bouyer
- Leonid Mokrushin
- Ansgar Fehnker
- Thomas Hune
- Judi Romijn ...
- Ed Brinksma
- Jan Tretmans
Types of Testing

Level

Aspect

system
integration
unit
robustness
performance
usability
reliability
functional behaviour
white box
black box

[tretnans]
Types of Testing

Level

Aspect

Accessiblity

system
integration
unit
robutness
performance
usability
reliability
functional behaviour
white box
black box

[tretnans]
Test of time requirements

- Probabilistic real-time requirements
  - Mean response time is less than 500ms
  - 90% of the responses are less than 890 ms
  - Jitter is less than 10%

- Hard deadlines / response times satisfied

- Estimation of
  - WCET
  - Context switch, IPC & scheduling overhead
Load/Stress Testing

Environment

Environment Emulator (event generator)

Stimuli

System under Test

Input

Output

System under Test

Response

Load Model

expected

measured
Specification Based Testing

A Specification

Timed Automaton = FSM + Clocks (dense) + Guards + Resets

- $x_1 \sim c$, $x_1 - x_2 \sim c$, where $\sim \in \{<, \leq, \geq, >\}$
- $x := c$
- Semantic state: $(l, \bar{u})$  \hspace{1cm} $(l_1, x = 1.17)$
Specification Based Testing

A Specification

Test cases

Timed Automaton = FSM + Clocks (dense) + Guards + Resets

- \( x_1 \sim c, x_1 - x_2 \sim c \), where \( \sim \in \{<,\leq,\geq,>\} \)
- \( x:=c \)
- Semantic state: \((l, \bar{u})\) \((l_1, x = 1.17)\)
Specification Based Testing

A Specification

Test cases

Timed Automaton = FSM + Clocks (dense) + Guards + Resets

- \( x_1 \sim c, x_1 - x_2 \sim c \), where \( \sim \in \{<,\leq,\geq,>\} \)
- \( x:=c \)
- Semantic state: \((l,\bar{u})\) \quad \((l_1, x = 1.17)\)
Specification Based Testing

A Specification

Timed Automaton = FSM + Clocks (dense) + Guards + Resets

- $x_1 \sim c$, $x_1 - x_2 \sim c$, where $\sim \in \{<,\leq,\geq,>\}$
- $x:=c$
- Semantic state: $(l,\bar{u})$  \quad $(l_1, x = 1.17)$

Test cases

- Click! Wait 1.5 click! DBLClick? (pass)
- Click! Wait 5 click! DBLClick? (fail)
- Click! Wait 0.1 click! DBLClick? (pass)
Exercises
Simple TA-specification

1) compute a timed trace that checks that a "c" can be produced!

2) How many tests do you think are necessary?
Exercises
Simple TA-specification

1) Compute a timed trace that checks that a "c" can be produced!

2) How many tests do you think are necessary?

Exercises
Generated test case

12 test cases are generated!
How do we cope with real-life specs?

Philips Sender with collision detection
Correctness Criteria
Implementation Relation $ioco$

\[ i_{\text{io}} c o \; s \; =_{\text{def}} \; \forall \sigma \in Straces(s) : \; \text{out}(i \; \text{after} \; \sigma) \subseteq \text{out}(s \; \text{after} \; \sigma) \]
Implementation Relation $ioco$

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

```
out(i\ after\ coin?) = \{ \delta, !coffee \}
```
Implementation Relation
ioco

\[ \text{i ioco } s =_{\text{def}} \forall \sigma \in \text{Straces}(s) : \text{out (i after } \sigma \text{)} \subseteq \text{out (s after } \sigma \text{)} \]

\[
\text{out (i after coin?)} = \{ \delta, \text{!coffee} \} \quad \text{out (s after coin?)} = \{ \text{!coffee} \}
\]
Implementation Relation

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

\[ \text{out}(i \text{ after } \text{coin?}) = \{ \delta, \text{!coffee} \} \quad \text{out}(s \text{ after } \text{coin?}) = \{ \text{!coffee} \} \]
Timed Conformance

i rt-ioco s  iff ∀σ ∈ (A_{in} ∪ A_{out} ∪ R)^*. out (i after σ) ⊆ out (s after σ)
iff TTr(I) ⊆ TTr(S)

s After σ = \{s' | s \xrightarrow{σ} s'\}, \quad S' After σ = \bigcup_{s \in S'} s After σ

Out(s) = \{a \in A_{out} \cup \mathbb{R} | s \xrightarrow{a}\}, \quad Out(S') \bigcup_{s \in S'} Out(s)

• TTr(s): the set of timed traces from s
  • eg.: σ = coin?.5.req?.2.weakCoffee!.9.coin?

• Out(s after σ) = possible outputs and delays after σ
  • eg. \{weakCoffee,0...2\}

• Intuition
  • no illegal output is produced and
  • required output is produced (at right time)
Timed Conformance??

\[
i \text{rt-ioco } s \quad \text{iff} \quad \forall \tau \in (A_{\text{in}} \cup A_{\text{out}} \cup R)^* . \quad \text{out} (i \text{ after } \tau) \subseteq \text{out} (s \text{ after } \tau)
\]

\[
\text{iff } \text{TTr}(I) \subseteq \text{TTr}(S)
\]

Example Traces

\[
\begin{align*}
I_1 & \text{ rt-ioco } S \\
I_2 & \text{ rt-}\not\text{ioco } S
\end{align*}
\]
Modeling Embedded Sys
Relativized rt-ioco

\[ s \text{ rtio}co_e \, t \iff \forall \sigma \in TTr(e). \, \text{Out}(s \text{ After } \sigma) \subseteq \text{Out}(t \text{ After } \sigma) \]

\[ s \text{ rtio}co_e \, t \iff TTr(s) \cap TTr(e) \subseteq TTr(t) \cap TTr(e) \]

\[ e \subseteq f \iff \text{rtioco}_f \subseteq \text{rtioco}_e \]

- Environment and IUT models are input-enabled timed LTS
- E.g. induced by I/O Timed Automata
Environments

- $E_O$
  - Smallest
  - Weakest

- $E_{EC}$
  - Realistic
  - Test Guiding
  - Test Selection

- $E_U$
  - Largest
  - Universal
  - Most discriminating
Example

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]

\[ I(D_S, D_W) \]
Test Generation
Touch-sensitive Light-Controller

- Patient user: Wait=∞
- Impatient: Wait=15
Test Generation Principles

- Test purpose based generation
  - “test that light can become max”
  - Formalize purpose
  - Use model to compute sequence that meets purpose (inputs and expected outputs)

- Model Coverage
  - State-coverage
  - Transition
  - Def-Use pairs
  - ...

- Manually produced/simulated scenarios (traces)
- Randomized model interpretation
- Fault-Models
- ONLINE (randomized) testing
Time Optimal Real-Time Test Generation using UPPAAL

Anders Hessel,
Paul Pettersson
Uppsala University
Sweden

Kim Larsen,
Brian Nielsen,
Arne Skou
Aalborg University
Denmark
Test Setup

Test Execution Host

tcp/ip

Test Interface

grasp
release
setLevel

grasp
release

LightControllerGUI

out(IGrasp); //@900
silence(500);
silence(1000);
in(OSetLevel,1,dfTolerance);
silence(1000);
in(OSetLevel,2,dfTolerance);
silence(1000);

LightController

JavaVM+w2k/Linux

mousePress
mouseRelease
“Scripts” for LightControl

Events: const IGrasp=0; const int IRelease=1; const int OSetLevel=0;

- **void out**(int eventNo);
  send eventNo to IUT at now();
- **void silence**(int msDelay);
  expect no outputs for msDelay: otherwise fail
- **void in**(int eventNo,int par, int msTolerance);
  expect input event(par) before now()+msTolerance otherwise fail
- **void at**(int eventNo, int par, int msTime, int msTolerance);
  expect input eventNo(par) at time msTime from start of test+/- msTolerance
Timed Tests

**EXAMPLE** test cases for **Interface**

- $0 \cdot \text{grasp!} \cdot 210 \cdot \text{release!} \cdot \text{touch?}. \text{PASS}$
- $0 \cdot \text{grasp!} \cdot 317 \cdot \text{release!} \cdot \text{touch?} \cdot 2^{\frac{1}{2}} \cdot \text{grasp!} \cdot 220 \cdot \text{release!} \cdot \text{touch?}. \text{PASS}$
- $1000 \cdot \text{grasp!} \cdot 517 \cdot \text{starthold?} \cdot 100 \cdot \text{release!} \cdot \text{endhold?}. \text{PASS}$

- Epsilon = 200ms
- Delta = 500ms
Timed Tests

**Example** test cases for Interface

0·grasp!·210·release!·touch?·PASS

0·grasp!·317·release!·touch?·2½·grasp!·220·release!·touch?·PASS

1000·grasp!·517·starthold?·100·release!·endhold?·PASS

• Epsilon=200ms
• Delta=500ms

INFINITELY MANY SEQUENCES!!!!!!
Optimal Tests
Optimal Tests

• Shortest test for max light??
Optimal Tests

- **Shortest** test for max light??
- **Fastest** test for max light??
Optimal Tests

- **Shortest** test for max light??
- **Fastest** test for max light??
- **Fastest** edge-covering test suite??
Optimal Tests

• **Shortest** test for max light??
• **Fastest** test for max light??
• **Fastest** edge-covering test suite??
• Least *power* consuming test??
**Test Purposes 1**

A specific test objective (or observation) the tester wants to make on SUT

**Environment model**

**System model**

**TP1**: Check that the light can become bright:

\[ E<> L=10 \]

• *Shortest (and fastest) Test:*

```plaintext
out(IGrasp); silence(500); in(OSetLevel, 0); silence(1000);
in(OSetLevel, 1); silence(1000); in(OSetLevel, 2); silence(1000);
in(OSetLevel, 3); silence(1000); in(OSetLevel, 4); silence(1000);
in(OSetLevel, 5); silence(1000); in(OSetLevel, 6); silence(1000);
in(OSetLevel, 7); silence(1000); in(OSetLevel, 8); silence(1000);
in(OSetLevel, 9); silence(1000); in(OSetLevel, 10);
out(IRelease);
```
Test Purposes 2

**TP2**: Check that controller can enter location ‘DnPassive’:

\[ E<> \text{Dim.DnPassive} \]

- If delay=1000
- **Shortest (and fastest)** Test:

\[
\text{out(IGrasp);} \\
\text{silence(500);} \\
\text{in(OSetLevel,0);} \\
\text{out(IRelase);} \\
\text{out(IGrasp);} \\
\text{silence(500);} \\
\]
Test Purposes 2

**TP2:** Check that controller can enter location `DnPassive`:

\[ E<> Dim.DnPassive \]

- If delay=40?
Test Purposes 2

**TP2**: Check that controller can enter location ‘DnPassive’:

\[ E<> \text{Dim.DnPassive} \]

- **If delay=40?**
- **Shortest Test:**

```plaintext
out(IGrasp);
silence(500);
in(OSetLevel,0);
out(IRelase);
out(IGrasp);
silence(500);
```
TP2: Check that controller can enter location ‘DnPassive’:
E<> Dim.DnPassive

• If delay=40?
• Shortest Test:
  out(IGrasp); silence(500); in(OSetLevel,0); out(IRelease); out(IGrasp); silence(500);

• Fastest Test:
  out(IGrasp); silence(500); in(OSetLevel,0); silence(40); in(OSetLevel,1); silence(40); in(OSetLevel,2); silence(40); in(OSetLevel,3); silence(40); in(OSetLevel,4); silence(40); in(OSetLevel,5); silence(40); in(OSetLevel,6); silence(40); in(OSetLevel,7); silence(40); in(OSetLevel,8); silence(40); in(OSetLevel,9); silence(40); in(OSetLevel,10); silence(40);
**Test Purposes 2**

**TP2:** Check that controller can enter location ‘DnPassive’:
E<> Dim.DnPassive

- If Wait=1500 and minDelay=400?
Test Purposes 2

TP2: Check that controller can enter location ‘DnP Passive’:

E<> Dim.DnP Passive

- If Wait=1500 and minDelay=400?

Ask a tool
**Test Purposes 3**

**TP3**: Check that controller re-sets light level to previous value after switch-on.

```plaintext
out(IGrasp);   //set level to 5
silence(500);
in(OSetLevel,0);
silence(1000);
in(OSetLevel,1);
silence(1000);
in(OSetLevel,2);
silence(1000);
in(OSetLevel,3);
silence(1000);
in(OSetLevel,4);
silence(1000);
in(OSetLevel,5);
out(IRelease);

out(IGrasp);   //touch To Off
silence(200);
out(IRelease);
in(OSetLevel,0);

out(IGrasp);   //touch To On
silence(200);
out(IRelease);
in(OSetLevel,5);
silence(2000);
```
Coverage Based Test Generation

- Multi purpose testing
- Cover measurement
- Examples:
  - Location coverage,
  - Edge coverage,
  - Definition/use pair coverage
Coverage Based Test Generation

- Multi purpose testing
- Cover measurement
- Examples:
  - Location coverage,
  - Edge coverage,
  - Definition/use pair coverage
Coverage Based Test Generation

- Multi purpose testing
- Cover measurement
- Examples:
  - Location coverage,
  - **Edge coverage**, 
  - Definition/use pair coverage
Coverage Based Test Generation

- Multi purpose testing
- Cover measurement
- Examples:
  - Location Coverage,
  - Edge Coverage,
  - **Definition/Use Pair Coverage**
Coverage Based Test Generation

- Multi purpose testing
- Cover measurement
- Examples:
  - Locations coverage,
  - Edge coverage,
  - Definition/use pair coverage
  - All Definition/Use pairs
- Generated by min-cost reachability analysis of annotated graph
Location Coverage

- Test sequence traversing all locations
- Encoding:
  - Enumerate locations $l_0, \ldots, l_n$
  - Add an auxiliary variable $l_i$ for each location
  - Label each ingoing edge to location $i$ $l_i := \text{true}$
  - Mark initial visited $l_0 := \text{true}$
- Check: $\text{EF}( l_0 = \text{true} \land \ldots \land l_n = \text{true} )$
Edge Coverage

- Test sequence traversing all edges
- Encoding:
  - Enumerate edges $e_0, ..., e_n$
  - Add auxiliary variable $e_i$ for each edge
  - Label each edge $e_i := true$
- Check: $EF( e_0 = true \land ... \land e_n = true )$
Fastest Edge Coverage

Cost=12600 ms

out(IGrasp); //touch: switch light on
silence(200);
out(IRelease);
in(OSetLevel,0);

out(IGrasp); //@200 //touch: switch light off
silence(200);
out(IRelease);//touch
in(OSetLevel,0);

//9
out(IGrasp); //@400 //Bring dimmer from ActiveUp
silence(500);//hold //To Passive DN (level=0)
in(OSetLevel,0);
out(IRelease);

out(IGrasp); //@900 //Bring dimmer PassiveDn->ActiveDN->
silence(500);///hold //ActiveUP+increase to level 10
silence(1000); in(OSetLevel,1);
silence(1000); in(OSetLevel,2);
silence(1000); in(OSetLevel,3);
silence(1000); in(OSetLevel,4);
silence(1000); in(OSetLevel,5);
silence(1000); in(OSetLevel,6);
silence(1000); in(OSetLevel,7);
silence(1000); in(OSetLevel,8);
silence(1000); in(OSetLevel,9);
silence(1000); in(OSetLevel,10)
silence(1000); in(OSetLevel,9); //bring dimm State to ActiveDN
out(IRelease); //check release->grasp is ignored
out(IGrasp); //@12400
out(IRelease);
silence(dfTolerance);
Mutants

• **M1 incorrectly implements switch**

```java
synchronized public void handleTouch() {
    if(lightState==lightOff) {
        setLevel(oldLevel);
        lightState=lightOn;
    }
    else { //was missing
        if(lightState==lightOn){
            oldLevel=level;
            setLevel(0);
            lightState=lightOff;
        }
    }
}
```

• **M2 incorrect additional delay in dimmer as if x:=0 was on ActiveUP ↔ActiveDN transitions**
<table>
<thead>
<tr>
<th>Description</th>
<th>Test#</th>
<th>M0</th>
<th>M1</th>
<th>M2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxLevel</td>
<td>1</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>Short ActiveDn</td>
<td>4</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>Resume</td>
<td>5</td>
<td>Pass</td>
<td>Fail</td>
<td>Pass</td>
</tr>
<tr>
<td>Edge Cov.</td>
<td>3</td>
<td>Pass</td>
<td>Fail</td>
<td>Fail</td>
</tr>
</tbody>
</table>
Online Testing Of Real-Time Systems
Online Testing

Torx tool

specification

implementation

[Jan Tretmans].
Online Testing

New menu
- ! x (x < 0)
- ! x (x >= 0)

Torx tool

explorer states transitions
primer

transition driver

abstract actions

adapter

bits

bytes

IUT

specification

? x (x < 0)

? x (x >= 0)

! √x

! -√x

implementation

? x (x < 0)

? x (x >= 0)

! √x

? x

[Jan Tretmans].
Online Testing

Torx tool

explorer \rightarrow primer \rightarrow driver \rightarrow adapter \rightarrow IUT

states \leftarrow transitions \leftarrow abstract actions \leftarrow bits \leftarrow bytes

specification

? x (x < 0)

! \sqrt{x}

? x (x \geq 0)

! -\sqrt{x}

implementation

? x (x < 0)

! \sqrt{x}

? x (x \geq 0)

? x

[Jan Tretmans].
Online Testing

Torx

Explorer -> Primer -> Driver

Primer -> Explorer

Driver -> Adapter

Adapter -> Driver

Explorer -> IUT

Primer -> IUT

Driver -> IUT

Adapter -> IUT

IUT

Abstract action

specification

implementation

? x (x < 0)

? x (x >= 0)

! \sqrt{x}

! \sqrt{x}

? x (x >= 0)

? x (x < 0)

[Jan Tretmans].
Online Testing

Torx tool

explorer → states → primer → transitions → driver → abstract actions → adapter → bits → IUT

specification

? x (x < 0)

! √x

? x (x >= 0)

! -√x

Concrete action

! 00001001

implementation

? x (x < 0)

? x (x >= 0)

! √x

? x

[Jan Tretmans].
Online Testing

Torx tool

specification

implementation

[Jan Tretmans]
Online Testing

Torx tool

explorer - states - transitions
primer - transition
driver - abstract actions
adapter - bits
IUT - bytes

Abstract action

? 3

specification

? x (x < 0)
? x (x >= 0)
! √x
! -√x

implementation

? x (x < 0)
? x (x >= 0)
! √x

[Jan Tretmans].
Online Testing

Torx tool

explorer ← states transitions → primer

Action

? 3

primer ↔ states transitions → driver

abstract actions

adapter ← bits bytes → IUT

specification

? x (x < 0)

? x (x \geq 0)

! \sqrt{x}

! -\sqrt{x}

implementation

? x (x < 0)

? x (x \geq 0)

? x

[Jan Tretmans].
Online Testing

specification

implementation

[Jan Tretmans].
Online Testing

New menu

! x (x < 0)
! x (x >= 0)

explorer :: states :: transitions :: primer :: transition :: driver :: abstract actions :: adapter :: bits :: bytes :: IUT

specification

implementation

? x (x < 0)
? x (x >= 0)

! √x
! -√x

! √x

? x

[Jan Tretmans].
Online Testing

specification

implementation

[Jan Tretmans].
Online Testing

Torx tool

explorer | states | transitions | primer | transition | driver | abstract actions | adapter | bits | IUT

specification

implementation

? x (x < 0)
! √x

? x (x ≥ 0)
! -√x

? x (x ≥ 0)
! √x

? x

[Jan Tretmans].
Online Testing

Torx tool

specification

implementation

[Jan Tretmans].
Online Testing

Torx tool

explorer → states → primer → transitions → driver → transition → abstract actions → adapter → bits → bytes → IUT

specification

implementation

?[x (x < 0)]

?[x (x >= 0)]

!√x

? x

! -√x

! √x

?[x (x >= 0)]

?[x (x > 0)]

? x

[Jan Tretmans].
Online Testing

Torx tool

explorer → states → transitions → primer

Abstract action

δ(x) (quiescence)

driver → abstract actions → adapter

bits → bytes → IUT

specification

implementation

[Jan Tretmans].
Online Testing

specification

implementation

[Jan Tretmans].
Online Testing

Torx tool

Check
?
(quiescence)

explorer
primer
driver
adapter
IUT

specification

? x (x < 0)

? x (x >= 0)

! √x

! -√x

implementation

? x (x < 0)

? x (x >= 0)

! √x

? x

[Jan Tretmans].
Online Testing

New menu

Torx tool

explorer \quad \text{states} \quad \text{transitions}

primer \quad \text{transition}

driver \quad \text{abstract} \quad \text{actions}

adapter \quad \text{bits} \quad \text{bytes}

IUT

specification

\begin{align*}
? x (x < 0) \\
? x (x \geq 0)
\end{align*}

\begin{align*}
! x (x < 0) \\
! x (x \geq 0)
\end{align*}

implementation

\begin{align*}
? x (x < 0) \\
? x (x \geq 0)
\end{align*}

\begin{align*}
! \sqrt{x} \\
! -\sqrt{x}
\end{align*}

[Jan Tretmans].
Test Setup

User Supplied Test Specification

- *Closed Network* partitioned into *Env* and *IUT*.
  - IUT model weakly input enabled
  - Model of *Environment* (not necessarily input enabled).
- Designate *observable input* and *output* actions.
- Specify amount of real time per one time-unit in model.
**Online Algorithm**

**Algorithm** TestGenExec (TestSpec) returns \{pass, fail\}

\[ Z := \{\langle l_0, 0 \rangle\}, \text{continueTesting} := \text{true} \]

**While** \( Z \neq \emptyset \) and \#iterations \( \leq T \) **do**

1. **if** \( \text{EnvOutput}(Z) \neq \emptyset \) \hspace{1cm} // Offer an input
   choose randomly \( a \in \text{EnvOutput}(Z) \)
   \text{send} \( i \) to \( SUT \)
   \[ Z := Z \text{ after } a \]

2. choose randomly \( \delta \in \text{Delays}(Z) \) \hspace{1cm} // Delay and wait for output
   \text{Wait}(\delta)
   \hspace{1cm} if \( o \) occurred after \( \delta' \leq \delta \) \hspace{1cm} then
   \hspace{2cm} \[ Z := Z \text{ after } \delta' \]
   \hspace{2cm} if \( o \notin \text{ImpOutput}(Z) \) \hspace{1cm} then return fail
   \hspace{2cm} \[ Z := Z \text{ after } o \]
   \hspace{1cm} else \hspace{1cm} // no output within \( \delta \) time
   \hspace{2cm} \[ Z := Z \text{ after } \delta \]

3. reset \( IUT \)
   \[ Z := \{\langle l_0, 0 \rangle\} \]

**if** \( Z = \emptyset \) **then** return fail **else** return pass
Online Testing

Testing–UPPAAL

Symbolic state set:
\{⟨k₀l₀, 0 ≤ x ≤ 0⟩\}

EnvOutput: \{coin\}

EnvInput: Ø

ImpOutput: Ø

Wait for output (delay) or offer input?
Online Testing

Symbolic state set:
\[ \{ \langle k_0 l_0, 0 \leq x \leq 0 \rangle \} \]

EnvOutput: \{ coin \}

EnvInput: \emptyset

ImpOutput: \emptyset

Let’s offer input choose (the only) "coin"
Online Testing

Symbolic state set:
\{<k_1l_1, 0 \leq x \leq 0>\}

EnvOutput: \{\text{req}\}

EnvInput: \emptyset

ImpOutput: \emptyset

Update the state set and other variables
Online Testing

Testing–UPPAAL

Symbolic state set:
\{\langle k_1l_1, 0 \leq x \leq 0 \rangle\}

EnvOutput: \{\text{req}\}

EnvInput: \emptyset

ImpOutput: \emptyset

Wait or offer input?
Let's wait for 5 units
Online Testing

Symbolic state set:
\{\langle k_1, l_1, 5 \leq x \leq 5 \rangle\}

EnvOutput: \{\text{req}\}

EnvInput: \emptyset

ImpOutput: \emptyset

..no output so far: update the state set..
Online Testing

Symbolic state set:
\{⟨k_1 l_1, 5 ≤ x ≤ 5⟩\}

EnvOutput: \{req\}

EnvInput: \emptyset

ImpOutput: \emptyset

Wait or offer input? let’s offer "req"
Online Testing

Testing–UPPAAL

Symbolic state set:
\[ \{ \langle k_2 l_2, 0 \leq x \leq 0 \rangle, \langle k_2 l_3, 0 \leq x \leq 0 \rangle \} \]

EnvOutput: \[ \emptyset \]

EnvInput: \{ weakCoffee, strongCoffee \}

ImpOutput: \{ weakCoffee, strongCoffee \}

Update the state set and other variables
Online Testing

Symbolic state set:
\{\langle k_2 l_2, 0 \leq x \leq 0 \rangle, \langle k_2 l_3, 0 \leq x \leq 0 \rangle\}

EnvOutput: \emptyset
EnvInput: \{weakCoffee, strongCoffee\}
ImpOutput: \{weakCoffee, strongCoffee\}

Wait or offer input? Let’s wait for 4 units
Online Testing

Symbolic state set:
\{ (k_2 | l_3, 4 \leq x \leq 4) \}

EnvOutput: \emptyset
EnvInput: \{ strongCoffee \}
ImpOutput: \{ strongCoffee \}

..no output so far: update the state set..
Online Testing

Testing—UPPAAL

Symbolic state set:
\{ (k_2 l_3, 4 \leq x \leq 4) \}

EnvOutput: \emptyset
EnvInput: \{ strongCoffee \}
ImpOutput: \{ strongCoffee \}

Wait or offer input?
Let’s wait for 2 units
Online Testing

Symbolic state set:
\{k_2l_3, 4 \leq x \leq 4\}

EnvOutput: \emptyset
EnvInput: \{strongCoffee\}
ImpOutput: \{strongCoffee\}

got output after 0 delay: update the state set
Online Testing

Testing–UPPAAL

Symbolic state set:
\{\langle k_2l_3, 4 \leq x \leq 4 \rangle\}

EnvOutput: \emptyset
EnvInput: \{strongCoffee\}
ImpOutput: \{strongCoffee\}

(what if there is a bug?)
Let’s wait for 2 units
Online Testing

Symbolic state set:
\[ \emptyset \]

EnvOutput: \[ \emptyset \]
EnvInput: \[ \emptyset \]
ImpOutput: \[ \emptyset \]

..no output so far: update the state set.. (!)
## Error Detection Capability

- **Train Xing:**
  - Environment 4 trains
  - IUT-model: 4 controllers, 1 queue

<table>
<thead>
<tr>
<th>Mutant</th>
<th>Number of verdicts</th>
<th>Input actions</th>
<th>Duration (time units)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pass</td>
<td>Fail</td>
<td>Crash</td>
</tr>
<tr>
<td>M1</td>
<td>0</td>
<td>1100</td>
<td>0</td>
</tr>
<tr>
<td>M2</td>
<td>0</td>
<td>1100</td>
<td>0</td>
</tr>
<tr>
<td>M3</td>
<td>0</td>
<td>1100</td>
<td>0</td>
</tr>
<tr>
<td>M4</td>
<td>0</td>
<td>1098</td>
<td>2</td>
</tr>
<tr>
<td>M5</td>
<td>0</td>
<td>1100</td>
<td>0</td>
</tr>
<tr>
<td>M6</td>
<td>0</td>
<td>1100</td>
<td>0</td>
</tr>
<tr>
<td>M0</td>
<td>1099</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Execution Performance

Train Xing:
- Environment 4 trains
- IUT-model: 4 controllers, 1 queue

<table>
<thead>
<tr>
<th>Mutant</th>
<th>Execution time, $\mu$s</th>
<th>State-set size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>After(delay)</td>
<td>After(action)</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>Max</td>
</tr>
<tr>
<td>M1</td>
<td>1055.1</td>
<td>2109.7</td>
</tr>
<tr>
<td>M2</td>
<td>1056.5</td>
<td>2085.4</td>
</tr>
<tr>
<td>M3</td>
<td>1055.6</td>
<td>2022.7</td>
</tr>
<tr>
<td>M4</td>
<td>1047.6</td>
<td>2024.8</td>
</tr>
<tr>
<td>M5</td>
<td>1064.4</td>
<td>1948.5</td>
</tr>
<tr>
<td>M6</td>
<td>1039.4</td>
<td>1956.5</td>
</tr>
<tr>
<td>M0</td>
<td>1045.5</td>
<td>3485.9</td>
</tr>
</tbody>
</table>
Demo Setup

T-UPPAAL
On-the-fly
Testing Host

tcp/ip

Test Interface

LightControllerGUI

setLevel

grasp

release

setLevel

grasp

release

LightController

JavaVM+w2k/Linux

mousePress

mouseRelease