Real-Time Test Generation using UPPAAL

Brian Nielsen
bnielsen@cs.auc.dk
Agenda

- **Real-time Testing**
  - Anders Hessel
  - Kim Larsen
  - Brian Nielsen
  - Arne Skou
  - Paul Pettersson

- **Time Optimal Test Generation**
  - Marius Mikucionis
  - Egle Sasnauskaitė
  - Kim Larsen

- **Online On-the-fly testing**
  - 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

Acknowledgements:
Testing

- to check the quality (functionality, reliability, performance, ...) of a software system
  - by executing it
  - in a controlled way
- **Primary verification technique used in industry**
  - In general avg. 10-20 errors per 1000 LOC
  - 30-50 % of development time and cost in embedded software
- To find errors
- To determine risk of release
Automated Model Based Conformance Testing

Does the behavior of the (blackbox) implementation comply to that of the specification?
Time Optimal Test Generation

[FATES’03] / Work-in-progress
This work

- Test case generation from timed automata
  - by reachability analysis
  - implementation in UPPAAL

- Testing Criteria:
  - single test purpose
  - coverage criteria: location, branching, definition/use pairs, etc.

- Optimality:
  - Test Cases: \( \sigma_{fp} = \varepsilon_0, i_0, \varepsilon_1, i_1, \varepsilon_2, i_2, \ldots \) with minimum cost
    e.g. \( \min(\varepsilon_0 + \varepsilon_1 + \ldots + \varepsilon_n) \)
  - Test Suites: \( T = \{ \sigma_1, \ldots, \sigma_n \} \) with minimum cost
Time-optimal test suites

- Product instance testing
- Test more behavior in less time
- Some operations (e.g. SUT resets) are very time-consuming
- Stressful for SUT??
Testing Real-Time Systems

- FSM model of system and environment
- Test purpose $P \approx$ reachability property $\phi_P$
- Test-case generation $\approx$ witness generation
- Test input sequence $\sigma_{\phi_P} = \varepsilon_0, i_0, \varepsilon_1, i_1, \varepsilon_2, i_2, \ldots$
- Test in/output $\delta_{\phi_P} = \varepsilon_0, i_0, \varepsilon_1, o_0, \varepsilon_1, i_1, o_1, \ldots$
- Test Verdict:
  - OK, if $\delta_{\phi_P} = \varepsilon_0, i_0, \varepsilon_1, o_0, \varepsilon_1, i_1, o_1, \ldots$ run of system model
  - NOK, otherwise
- Timed Automata?
Controllable Timed Automata

- **Input Enabled**: all inputs can always be accepted
- **Output Urgent**: enabled outputs will occur immediately
- **Determinism**: two transitions with same input/output leads to the same state
- **Isolated Outputs**: if an output is enabled, no other output is enabled
Simple Light Controller

Environment model

System model

\[ T_{\text{react}} = 0 \]
\[ T_{\text{sw}} = 4 \]
\[ T_{\text{pause}} = 5 \]
\[ T_{\text{idle}} = 20 \]
Test Purposes

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:

\[ E \leftrightarrow \text{LightController.bright} \]

- **Shortest Test**: 20·touch!·0·bright?
- **Fastest Test**: 0·touch!·0·dim?·0·touch!·0·bright?
Test Purposes 2

TP2: Check that the light switches off after three successive touches

Use restricted environment and $E<> tpEnv.goal$

- The fastest test sequence is
  0·touch!·0·dim?·0·touch!·0·bright?·0·touch!·0·off?
Other Environments

**Slow User:** $T_{\text{react}}=2$

$T_{\text{react}}=2$
$T_{\text{sw}}=4$
$T_{\text{pause}}=5$
$T_{\text{idle}}=20$

**TP1:**
*Fastest Test:*
0·touch!·0·dim?·2·touch!·0·bright?

**Pausing User:** max 2 successive quick touches
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 $e_0, \ldots, e_n$
  - Add auxiliary variable $e_i$ for each edge
  - Label each edge $e_i := true$
- Check: $\text{EF}( e_0 = true \land \ldots \land e_n = true )$
**Edge Coverage**

**EC**: \( T_{\text{react}} = 0 \)

\[
0 \cdot \text{touch}! \cdot 0 \cdot \text{dim}? \cdot 0 \cdot \text{touch}! \cdot 0 \cdot \text{bright}? \cdot 0 \cdot \text{touch}! \cdot 0 \cdot \text{off}? \\
20 \cdot \text{touch}! \cdot 0 \cdot \text{bright}? \cdot 4 \cdot \text{touch}! \cdot 0 \cdot \text{dim}? \cdot 4 \cdot \text{touch}! \cdot 0 \cdot \text{off}?
\]

\[\text{Cost}=28\]

**EC'**: \( T_{\text{react}} = 2 \)

\[
0 \cdot \text{touch}! \cdot 0 \cdot \text{dim}? \cdot 4 \cdot \text{touch}! \cdot 0 \cdot \text{off}? \\
20 \cdot \text{touch}! \cdot 0 \cdot \text{bright}? \cdot 4 \cdot \text{touch}! \cdot 0 \cdot \text{dim}? \cdot 2 \cdot \text{touch}! \cdot 0 \cdot \text{off}?
\]

\[\text{Cost}=32\]

**EC'':** *pausing user* \( T_{\text{react}} = 2, \ T_{\text{pause}} = 5 \)

\[
0 \cdot \text{touch}! \cdot 0 \cdot \text{dim}? \cdot 2 \cdot \text{touch}! \cdot 0 \cdot \text{bright}? \cdot 5 \cdot \text{touch}! \cdot 0 \cdot \text{dim}? \\
4 \cdot \text{touch}! \cdot 0 \cdot \text{off}? \cdot 20 \cdot \text{touch}! \cdot 0 \cdot \text{bright}? \cdot 2 \cdot \text{touch}! \cdot 0 \cdot \text{off}?
\]

\[\text{Cost}=33\]
Definition/Use Pair Coverage

- Dataflow coverage technique
- Def/use pair of variable $x$:

$\begin{align*}
\text{Encoding:} \\
\quad v_d \in \{\text{false}\} \cup \{e_0, \ldots, e_n\}, \text{ initially false} \\
\quad \text{Boolean array } du \text{ of size } |E| \times |E| \\
\quad \text{At definition on edge } i: \quad v_d := e_i \\
\quad \text{At use on edge } j: \quad \text{if} (v_d) \text{ then } du[v_d, e_j] := \text{true}
\end{align*}$
**Definition/Use Pair Coverage**

- Dataflow coverage technique
- Def/use pair of variable $x$:
  
  ![Diagram](image.png)
  
  - Definition: $x:=0$
  - No defs
  - Use: $x \geq 4$

- **Encoding:**
  - $v_d \in \{\text{false}\} \cup \{e_0, \ldots, e_n\}$, initially false
  - Boolean array $du$ of size $|E| \times |E|
  - At definition on edge $i$: $v_d:=e_i$
  - At use on edge $j$: if($v_d$) then $du[v_d,e_j]:=true$

- **Check:**
  - $\text{EF( all } du[i,j] = true )$

  $v_d \in \{\text{false}\} \cup \{e_0, \ldots, e_n\}$, initially false
  
  Boolean array $du$ of size $|E| \times |E|
  
  At definition on edge $i$: $v_d:=e_i$
  
  At use on edge $j$: if($v_d$) then $du[v_d,e_j]:=true$

  $\text{EF( all } du[i,j] = true )$
Test Suite Generation

- In general a set of test cases is needed to cover a test criteria
- Add global reset of SUT and environment model and associate a cost (of system reset)

Test sequence $\sigma = \epsilon_0,i_0, ..., \epsilon_1, i_1, \text{reset} \epsilon_2,i_2, ..., \epsilon_0,i_0, \text{reset} \epsilon_1, i_1, \epsilon_2,i_2, ...$

Test suite $T = \{\sigma_1, ..., \sigma_n\}$ with minimum cost

- Same encodings and min-cost reachability
The Philips Audio Protocol

- A bus based protocol for exchanging control messages between audio components
  - Collisions
  - Tolerance on timing events

<table>
<thead>
<tr>
<th>Bit stream</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manchester encoding</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram showing TX and RX with signals up, dn, in0, empty, coll, isUP, out0, out1, end.
Philips Audio Protocol
Benchmark Example

- Philips Audio Protocol

<table>
<thead>
<tr>
<th>Coverage Criterion</th>
<th>Execution time (µs)</th>
<th>Generation time (s)</th>
<th>Memory usage (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Edge}_{\text{Sender}}$</td>
<td>212350</td>
<td>2.2</td>
<td>9416</td>
</tr>
<tr>
<td>$\text{Edge}_{\text{Receiver}}$</td>
<td>18981</td>
<td>1.2</td>
<td>4984</td>
</tr>
<tr>
<td>$\text{Edge}_{\text{Sender,Bus,Receiver}}$</td>
<td>114227</td>
<td>129.0</td>
<td>331408</td>
</tr>
</tbody>
</table>
Touch-sensitive Light-Controller

- Patient user: Wait=∞
- Impatient: Wait=15
## Benchmark

### Coverage Criteria

<table>
<thead>
<tr>
<th>Coverage Criterion</th>
<th>Impatient</th>
<th>Patient</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Exec-time</td>
<td>Suite length</td>
</tr>
<tr>
<td>Location\textit{Dimmer}</td>
<td>20</td>
<td>12</td>
</tr>
<tr>
<td>Location\textit{Dimmer, Switch, Interface}</td>
<td>25</td>
<td>17</td>
</tr>
<tr>
<td>Edge\textit{Dimmer}</td>
<td>253</td>
<td>176</td>
</tr>
<tr>
<td>Edge\textit{Interface}</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>Edge\textit{Dimmer, Switch, Interface}</td>
<td>263</td>
<td>188</td>
</tr>
<tr>
<td>Edge\textit{Interface} + Location\textit{Dimmer}</td>
<td>25</td>
<td>19</td>
</tr>
<tr>
<td>Def-Use\textit{on}</td>
<td>40</td>
<td>34</td>
</tr>
<tr>
<td>Def-Use\textit{OL}</td>
<td>45</td>
<td>34</td>
</tr>
</tbody>
</table>
Benchmark

- Increasing model size (light levels)
  Edge Coverage

<table>
<thead>
<tr>
<th>Levels</th>
<th>Exec-time</th>
<th>Gen-time (s)</th>
<th>Mem (MB)</th>
<th>Exec-time</th>
<th>Gen-time (s)</th>
<th>Mem (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>263</td>
<td>2.06</td>
<td>9.1</td>
<td>63</td>
<td>3.19</td>
<td>10.1</td>
</tr>
<tr>
<td>20</td>
<td>493</td>
<td>3.68</td>
<td>11.4</td>
<td>93</td>
<td>12.40</td>
<td>20.1</td>
</tr>
<tr>
<td>30</td>
<td>723</td>
<td>5.29</td>
<td>12.6</td>
<td>123</td>
<td>28.17</td>
<td>40.4</td>
</tr>
<tr>
<td>50</td>
<td>1183</td>
<td>8.59</td>
<td>17.4</td>
<td>183</td>
<td>78.30</td>
<td>86.9</td>
</tr>
<tr>
<td>100</td>
<td>2333</td>
<td>16.76</td>
<td>28.0</td>
<td>333</td>
<td>339.52</td>
<td>314.9</td>
</tr>
<tr>
<td>200</td>
<td>4633</td>
<td>34.45</td>
<td>44.3</td>
<td>633</td>
<td>1494.35</td>
<td>1233.8</td>
</tr>
<tr>
<td>400</td>
<td>9233</td>
<td>66.03</td>
<td>77.1</td>
<td>N/A</td>
<td>&gt;7000</td>
<td>&gt;4180.6</td>
</tr>
</tbody>
</table>
Benchmark

Search-order (edge coverage)

<table>
<thead>
<tr>
<th>Search order</th>
<th>First Sol.</th>
<th>Optimal Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Exec-time</td>
<td>Gen-time (s)</td>
</tr>
<tr>
<td>BF</td>
<td>123</td>
<td>27.91</td>
</tr>
<tr>
<td>DF</td>
<td>791</td>
<td>0.15</td>
</tr>
<tr>
<td>C_BF</td>
<td>123</td>
<td>30.44</td>
</tr>
<tr>
<td>C_DF</td>
<td>791</td>
<td>0.15</td>
</tr>
<tr>
<td>C_BF_R</td>
<td>123</td>
<td>30.70</td>
</tr>
<tr>
<td>C_DF_R</td>
<td>791</td>
<td>0.15</td>
</tr>
<tr>
<td>C_MC</td>
<td>123</td>
<td>25.87</td>
</tr>
<tr>
<td>C_MC_R</td>
<td>123</td>
<td>3.23</td>
</tr>
</tbody>
</table>

- BF = Breadth First
- DF = Depth First
- C_BF, C_DF = Time optimal reachability in BF or DF order
- C_MC = Minimal accumulated cost first
- R = Remaining cost estimate
  \[ R = \begin{cases} (\text{Max-L}) \times \text{delay}, & \text{if Max not reached} \\ 0, & \text{otherwise} \end{cases} \]
Implementation

- Prototype implementation in UPPAAL supporting:
  - Uniformly Priced Timed Automata
  - Minimum Cost Search
  - Random
  - Heuristics

- Release UPPAAL version 3.4 beta 1
  - Generation of Fastest Trace (rate=1)
  - Integrated with GUI etc.
  - Available at www.uppaal.com
New Implementation

- UPPAAL verifier extension
- On-the-fly annotation of symbolic state space

- Monotonic variables:
  - Auxiliary variables are monotonically growing
  - Symbolic state: \((l,m,D) \subseteq (l,m',D')\) if
    - \(D \subseteq D'\) and
    - \(m \leq m'\)

- Up to 67% reduction of run-time
- Up to 45% reduction of space
Conclusions

- We generate test cases as sequences of input/output with timing
- Implemented in UPPAAL:
  - On-the-fly annotation resets and of test coverage criteria.
  - Min-cost test cases
  - Min-cost test suite
  - Monotonic variables
- Performance is encouraging.
Future

- Further exploit monotonic variables
- Reduce determinism in controllable timed automata, e.g. output persistent DIOP:

\[ x \geq 2, o! \]

- Case Studies
- Generalize with Cost (LPTA)
Online On-the-fly Testing

Work-in-progress
On-The-Fly Testing

specification

implementation

[Jan Tretmans].
Test-setup

- **Test specification** supplied by user:
  - Closed Timed Automata Network
  - TA components partitioned into Environment and System model
  - System model assumed (weakly) Input Enabled
  - Designate observable input and output actions
  - Associate data-variables to observable actions for value passing
State-set computation

- Compute all potential states the model can occupy after the timed trace $\varepsilon_0, i_0, \varepsilon_1, o_1, \varepsilon_2, i_2, o_2, ...$

- Let $S$ be a set of states
  
  - $O$ is a legal output from SUT iff $a$ in $\text{ImpOutput}(S)$
  
  - $I$ is a relevant input in Env iff $I$ in $\text{EnvOutput}(S)$
  
  - $O$ is a expected by ENV iff $o$ in $\text{EnvInput}(S)$
State-set computation

- Compute all potential states the model can occupy after the timed trace $\varepsilon_0, i_0, \varepsilon_1, o_1, \varepsilon_2, i_2, o_2, \ldots$

- Let $S$ be a set of states
  - $S$ after $a$: possible states after executing $a$ (and $\tau^*$)
  - $S$ after $\varepsilon$: possible states after $\tau^*$ and $\varepsilon_i$, totaling a delay of $\varepsilon$

\begin{itemize}
  \item $\{ \langle l_0, x=3 \rangle \}$ after $a =$
  \{ $\langle l_2, x=3 \rangle$, $\langle l_4, x=3 \rangle$, $\langle l_3, x=0 \rangle$ \}

  \item $\{ \langle l_3, x=4 \rangle, \langle l_0, 0 \leq x \leq 4 \rangle \}$ after $4 =$
  \{ $\langle l_0, x=0 \rangle$ \}
\end{itemize}

- Implemented using UppAal’s symbolic reachability engine
Real-time Online On-the-Fly

Specification

TA-network

State-set explorer:

maintain and analyse a set of symbolic states in real time!
T-UppAal: implementation

Graphical User Interface (java)
editor 
simulator 
verifier

Uppaal Engine Server (C++)
- Parsing
- Communication
- Control

Zones & Reachability, Etc

State-set explorer
Online Test Generation

System Under Test

Adapter

Driver
Online Algorithm

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

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

**While** continueTesting **do either**

1. \( i := \text{ChooseAction}(\text{EnvOutput}(S)) \) // Offer an input
   send \( i \) to SUT
   \( Z := \text{After}(S, i) \)

2. \( \delta = \text{chooseDelay}(S) \) // Delay and wait for output
   \( \text{Wait}(\delta) \)
   if \( o \) occurred after \( \delta' \leq \delta \) then
      \( S := \text{After}(S, \delta') \)
      1. if \( o \notin \text{ImpOutput}(S) \) then return fail
      2. if \( o \notin \text{EnvInput}(S) \) then return inconclusive
      \( S := \text{After}(S, o) \)
   else // no output within \( \delta \) time
      \( S := \text{After}(S, \delta) \)
      if \( S = \emptyset \) then return fail
   3. continueTesting := false // terminate

return pass
Train-Gate Demo

- **Gate** implemented as C++ thread communicating directly with driver
- 1 time unit = 0.1 sec.
- Faulty implementation randomly forgets to stop approaching train
Future Work

- Mature prototype
  - Implement value passing
  - Environment invariants and “max delay” choice
  - State-set explosion found in one case
  - GUI component
  - Performance evaluation and optimization

- Time and clock synchronization

- Currently randomized action, and delay choices
  - Guiding based on state space partitioning and extreme values a la [Nielsen, Skou] in Tacas’01
  - Coverage

- More applications and case study

- Other applications of state set computation
  - Failure diagnosis
  - Monitoring
Design space for RT-testing ??

• Completely deterministic
• Guarantees
  • Cost optimal
  • Coverage
• Precomputed strategies & verdict assignment
• Existing Tool&Gui

• Time and action non-deterministic
• Randomized search
• Heuristic Guiding
• Dynamic interpretation
• Extensions to Uppaal Engine
END