Specification and Test of Real-Time Systems

Ph.D. Lecture

by

Brian Nielsen
What is a Real-Time System?

- Correctness depends on **absolute timing** of events

  "when the temperature reaches 90°, the valve to the cooling water must open within 1 sec"

- Communication with external physical environment
- Inherently Complex
- How to develop real-time systems correctly?
  - Requirements must be clear and concise (formal)
  - Tools to deal with complexity
Why Improve Testing?

- *Costly* (30% of development time)
- Significant errors not found
- *The validation activity used by industry today*
What is Testing?

- **Testing**: execution of an implementation with the purpose of finding errors
- **Conformance testing**: does the behavior of a system comply to the specification
- **Black-box testing**: only external behavior is visible/relevant

A Specification

Test cases

- Click!
  - Wait 1.5 click!
  - DBLClick?
  
  (pass)

- Click!
  - Wait 5 click!
  - DBLClick?
  
  (fail)

- Click!
  - Wait 0.1 click!
  - DBLClick?
  
  (pass)
How do we cope with real-life specs?

Philips Sender with collision detection

config
system sender;
tint sent;
observable in1, in0, up, dn, empty, isUp, coll;
Automated Testing

- Automatic generation of test cases from (formal) specifications
- Automatic execution of test cases
- Commercial tools becoming available
- Cases:
  - International Space Station [Peleska et al. ’99]
  - SSCOP protocol [Jard et al. ’00]
  - Satellite power and thermal control system [Schlingloff ’97]
- Little support for real-time constraints
Analysis of Real-Time Systems

- Model Checking aims at *automatically proving* real-time properties of an abstract model of a system

- *Efficient symbolic techniques for reachability analysis* [Dill’89, Larsen et al. ’96, Larsen et al. ’97]
  - Constraint solving (Zones, DBM, CDD)

- UppAal

- Numerous successful industrial applications
  - Philips audio protocol [Larsen et al. ’96],
  - B&O power controller [Skou et al. ’99],
  - ...
My Thesis

- The symbolic reachability techniques developed for model checking can also be used for automatic generation of tests.
- Can systematically generate tests
- Can handle cases of realistic size
Contributions

- Basic algorithms for test generation from CSM specifications
- Framework for selection of real-time test cases
- Timed Test generation algorithms
  - Direct interpretation of TA specifications
  - From a restricted class of TA, a guaranteed covering test suite wrt. a selection criterion
- Tool implementation RTCAT
- Applications and evaluation
- Language support for modular and reusable specifications
  - Functional components
  - Real-time and synchronization constraint patterns
Outline

1. Introduction
2. Overview of untimed techniques
3. Timed
   1. Timed testing setup
   2. Systematic algorithm (ERA)
4. Evaluation
5. Conclusions + Future work
Ingredients in Automated Testing

Testing Theory
- Correctness criterion
- Model of Tests
- Relevant tests

Specifications
- Required behavior
- Environment assumptions

Tools
- Data structures
- Algorithms

Test Suite
- Abstract tests
- Executable tests

Selection Strategy
- Test purposes
- Impl/spec coverage
- Fault models

- Verdicts
- Measured Coverage
- Diagnostic info

Test Execution
Work On Untimed Testing

Specification
-Com. State Machines
-(LTS)

Tool
-TestGen
-Succes Graph

Selection Strategy
-(Coverage of SG)

Test Suite

Testing Theory
-Hennessy
Communicating State Machines

- Shared variables, enabling conditions, assignments
- CCS-style parallel composition with synchronous communication
- Semantics Given as Labeled Transition System

LTS for Non-deterministic Coffee Machine

![Diagram of LTS for Non-deterministic Coffee Machine]

- s0
- tea!
- cof!
- coin?

s1 -> s0
s0 <- s2
Tests and Test Execution

- **Test**: LTS with state based verdicts
  - Pass, fail, (Inconclusive)
- **Test Execution**:
  - Parallel composition of tester and implementation
  - Synchronous communication on complementary actions
Implementation Relations

- **Testing Preorder** [Hennessy '84]
  1. \( S \) **must** \( T \) iff \( \forall \Sigma \in \text{Comp}(T \parallel S). \Sigma \) is successful.
  2. \( S \) **may** \( T \) iff \( \exists \Sigma \in \text{Comp}(T \parallel S). \Sigma \) is successful.
  3. \( S \sqsubseteq_{\text{must}} I \) iff \( \forall T \in L_{\text{tlts}}. S \) **must** \( T \) implies \( I \) **must** \( T \)
  4. \( S \sqsubseteq_{\text{may}} I \) iff \( \forall T \in L_{\text{tlts}}. S \) **may** \( T \) implies \( I \) **may** \( T \)
  5. \( S \sqsubseteq_{\text{te}} I \) iff \( S \sqsubseteq_{\text{must}} I \) and \( S \sqsubseteq_{\text{may}} I \)

- **Conf** [Brinksma '88], *Ioconf* [Tretmans '96]

  \( S \sqsubseteq_{\text{te}} I \) iff \( I \text{ conf } S \) \( \land \) \( Tr(I) \subseteq Tr(S) \) \( \land \) \( Tr(S) \subseteq Tr(I) \)
Hennessy Testers

1. Find a trace $\sigma$ in spec
2. Compute $B=S$ after $\sigma$
3. Find a set of actions $A$ st. $B$ must $A$

Generate "all" such test passed by the specification
Direct Interpretation

- Scales well (no state explosion)
- Systematic?
Success Graph

- A success graph is an LTS
  - $\tau$-reduced, Determinized, Trace Equivalent
  - States labeled with: Must, Can, Refusal sets

Eases systematic generation
- Mark tests already generated
- Measuring coverage
- (Is a good selection criterion)

State explosion Problem!
About Time ...

Testing Theory
- Timed Hennessy Tests

Specification
- Timed Automata
- ERA

Test Generation Tool
- RTCAT

Test Suite
- Abstract tests (TA)

Selection Strategy
- Equivalence classes
- Guaranteed Coverage
Timed Automata

Specifications

Permitted Guards: \( x_i - x_j \sim c_{ij} \) and \( x \sim c, \quad \sim \in \{<, \leq, \geq, >\} \)
Our Timed Hennessy Tests

Tester and implementation communicates using urgent synchronous actions

\[
\begin{align*}
\text{after } &\varepsilon(d_1)b_1, \ldots, \varepsilon(d_n)b_n, \varepsilon(d_{n+1}) \text{ must } \{a_1, \ldots, a_n\} \\
\text{can } &\varepsilon(d_1)b_1, \ldots, \varepsilon(d_n)b_n \\
\text{cannot } &\varepsilon(d_1)b_1, \ldots, \varepsilon(d_n)b_n
\end{align*}
\]
A Direct Interpretation

• Similar to the untimed case, except for *choice of delay*
• Which delay to choose?
  - All
  - Random
  - Periodic
• The sets after $\varepsilon(d)$ and after $a$ may be infinite
  - Must be computed symbolically
Zones and Timed Automata

- A **zone** is a conjunction of linear in-equations of the form:
  \[ x_i - x_j \sim c_{ij} \text{ and } x \sim c, \quad \sim \in \{<, \leq, \geq, >\} \]

- Forms **convex polyhedra**

- **Semantic state:** \((l, \overline{u})\)
  - \(l\) is a location
  - \(u\) is a vector of current clock values

- **Symbolic State:** \([l, Z]\)
  - \(l\) is a location
  - \(Z\) is a zone (solution set of clock vectors)
Symbolic Execution

Forward step

Backward step

Algorithms
A Timed Success Graph

• Goal
  - Make it easy to generate timed Hennessy tests
    • Finding traces
    • Deadlock properties
    • Verdicts
  - Ease *systematic* generation of tests wrt a coverage criterion
Bizarre Results from TA-theory

In contrast to finite state automata:

- **Non-deterministic** TAs are more expressive than deterministic ones [Alur et al. ´94]
  - A non-deterministic TA cannot be determinized
  - Language inclusion is undecidable

- TAs with *internal* (τ-actions) are more expressive than TAs without [Diekert et al. ´97]
  - τ-edges with clock resets lying on a directed cycle cannot be removed
  - other τ-edges can be removed, but may cause a blow-up in the number of locations

Simple clean subset of TA wanted!
Event Recording Automata (ERA)  

[Alur, Fix, Henzinger ’94]

• For each action \( a \), \( X_a \) is the event clock for \( a \)
  - \( X_a \) is reset when \( a \) occurs
  - \( X_a \) records the amount of time since last occurrence of \( a \)
  - No other assignments permitted
  - No internal actions/synchronization

• Determinizable, but strict subclass of TA

• Clean model: event clocks
Coffee Vending Machine ERA

Determinized ERA

Specifications
Domain Testing

Selection Strategy

int abs(int x);

\[
\begin{align*}
0, & \quad \text{if } x < 0 \\
x, & \quad \text{otherwise}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Input</th>
<th>expected output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>-5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

- Partition input space into equivalence classes
- Apply \textit{uniformity hypothesis}
- Use extreme values to support uniformity hypothesis
When enabledness of edges changes, implementation should match this change.

Specification is used as implementation guide => implementation can be expected to be stable.

Equivalent wrt. Must, Can, and Refusal sets.
Selection Criterion

- **A complete Covering Test Suite**
  1. For each (reachable) equivalence class $Q$
  2. For one point in $Q$ via some trace $\sigma$
  3. Makes all observation $M$, $C$, $R$ in $S$ after $\sigma$
Equivalence Class Graph (CofM)

\[
\begin{align*}
\text{[s1], tt} & \quad \to \quad \text{[s2], Xcoin=4} \\
\text{[s2], 2<=Xcoin<4} & \quad \to \quad \text{[s2], Xcoin>4} \\
\text{[s2], Xcoin<2} & \quad \to \quad \text{[s3], Xgive>=2} \\
\text{[s3], Xgive<2} & \quad \to \quad \text{[s4], Xgive>=1} \\
\text{[s4], Xgive<1} & \quad \to \quad \text{[s5], tt} \\
\text{[s5], tt} & \quad \to \quad \text{[s6], tt}
\end{align*}
\]
Reachability Graph

- \([s1], tt\)
- \([s2], X\text{coin}=4\)
- \([s2], 2<=X\text{coin}<4\)
- \([s2], X\text{coin}>4\)
- \([s2], X\text{coin}<2\)
- \([s3,s4], X\text{give}>=2\)
- \([s3,s4], 1<=X\text{give}<2\)
- \([s3,s4], 1<=X\text{give}<1\)
- \([s4], X\text{give}>=1\)
- \([s4], X\text{give}<1\)
- \([s3], X\text{give}>=2\)
- \([s3], X\text{give}<2\)

Transitions:
- Give
- Coin
- CoF
- ThinCoF
Back-Propagate
Select Delays

- $\{s1\}, \text{tt}$
- $\{s2\}, \text{Xcoin=} 4$
- $\{s2\}, 2 \leq \text{Xcoin} < 4$
- $\{s2\}, \text{Xcoin} > 4$
- $\{s2\}, \text{Xcoin} < 2$
- $\{s3, s4\}, \text{Xgive} \geq 2$
- $\{s3, s4\}, 1 \leq \text{Xgive} < 2$
- $\{s3, s4\}, 1 \leq \text{Xgive} < 1$
- $\{s4\}, \text{Xgive} \geq 1$
- $\{s4\}, \text{Xgive} < 1$
- $\{s3\}, \text{Xgive} \geq 2$
- $\{s3\}, \text{Xgive} < 2$
- $\{s6\}, \text{tt}$
- $\{s5\}, \text{tt}$
Generated Tests for CofM

- 16 test cases
Evaluation

• Specifications
  - Philips Audio Protocol
  - Pedestrian Crossing
  - Token-Bus

• Event Recording Automata as Specification Language

• Testing Preorder, test language

• Implementability

• Tool Performance
  - Number of tests
  - Traversal order
  - Memory+time usage
Tool Developed

- Networks of Event Recording Automata sharing clocks and discrete variables
- Traversal order
  - depth- and breadth-first construction of EQC-graph and reachability graph
- Composed or Individually generated Hennessey Tests
- Partial search methods
  - limit trace depth
  - use bit-state hashing on equivalence class and reachability graph
- 22 Kloc in C++
The Philips Audio Protocol

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

Bit stream: 1 0 0 0 1 1 0
Manchester encoding: 

- TX RX
- TX RX
- TX RX
Philips

Sender

with

collision
detection

config
system sender;
int sent;
observable in1, in0, up, dn, empty, isUp, coll;
config
  system receiver;
  int odd;
  observable VUP, out1, out0, end;
Token Passing

- A station may send only when it holds a token
- It may hold the token for at most 100 time units
- Token must be passed on to next station in the ring
Token Passing3

config
system station0, station1, station2;
observable GT0, RT0, send0, GT1, RT1, send1, GT2, RT2, send2;
int th;

station0

th=0 GT0?

th=0 GT0?

th:=1 GT0?

XGT0=100 RT0!

th:=1

RT0!

s3

s2

s1

send0!

XGT0<100

send0!

XGT0<100

station1

th=1 GT1?

th:=2 GT1?

XGT1=100 RT1!

th:=2

RT1!

s3

s2

s1

send1!

XGT1<100

send1!

XGT1<100

station2

th=2 GT2?

th:=0 GT2?

XGT2=100 RT2!

th:=0

RT2!

s3

s2

s1

send2!

XGT2<100

send2!

XGT2<100
# Experimental results

<table>
<thead>
<tr>
<th>Equivalence Classes</th>
<th>CofM</th>
<th>PedX</th>
<th>Philips (R)</th>
<th>Philips (S)</th>
<th>Token(^7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth First</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symbolic States</td>
<td>17</td>
<td>77</td>
<td>71</td>
<td>120</td>
<td>15427</td>
</tr>
<tr>
<td>Time (s)</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>541</td>
</tr>
<tr>
<td>Memory (MB)</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>C-Number of Tests</td>
<td>16</td>
<td>30</td>
<td>97</td>
<td>99</td>
<td>71</td>
</tr>
<tr>
<td>C-Total Length</td>
<td>45</td>
<td>151</td>
<td>527</td>
<td>548</td>
<td>574</td>
</tr>
<tr>
<td>Depth First</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symbolic States</td>
<td>17</td>
<td>247</td>
<td>85</td>
<td>119</td>
<td>7283</td>
</tr>
<tr>
<td>Time (s)</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>158</td>
</tr>
<tr>
<td>Memory (MB)</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>24</td>
</tr>
<tr>
<td>C-Number of Tests</td>
<td>16</td>
<td>26</td>
<td>86</td>
<td>95</td>
<td>60</td>
</tr>
<tr>
<td>C-Total Length</td>
<td>45</td>
<td>595</td>
<td>1619</td>
<td>1154</td>
<td>5290</td>
</tr>
</tbody>
</table>

Note: Token\(^7\) is not specified in the table.
Number of Generated Tests

• Findings
  - The number of generated tests in all configurations is very reasonable
  - The number of equivalence classes is very reasonable
  - Test composition (only) reduces test suite size by 10-25%

• Conclusions
  - Selection strategy feasible for specs of similar size
  - Plenty room for more tests
    • Extreme value selection
    • Longer traces
    • Larger specifications
Traversal Order

• Findings
  - Depth-first: fewer but much larger test suites
  - Breadth-first: more tests, but much shorter test suites
  - (Depth-first some times more efficient than BF)

• Conclusions
  - Use BF when an economic covering test suite is the goal
  - Use DF when check of behavior after long sequences is the goal
Space and Time usage

• Findings
  - Except for Token Passing\(n\) the memory and time usage is insignificant
  - Token Passing is problematic
  - A huge number of symbolic states needed to terminate reachability analysis
  - Number of clocks has severe impact

• Conclusions
  - Symbolic reachability analysis (and EQC graph construction) is a bottleneck
Possible explanations

• Specification language related
  - More active clocks compared to TA spec
  - More locations in ERAs compared to TA

• Implementation related
  - Reachability performed on EQC-graph
  - Representation of concave sets as several zones
  - Inefficient zone implementation compared to uppaal2k (>100*)
Conclusions

• Yes, the symbolic methods can be used for test generation
  - Proposed a systematic technique
  - Guarantees coverage wrt. coverage criterion

• Reasonable sized, practically relevant systems could be handled
  - Applicable “as is” for strictly timed controllers

• Current tools can be improved

• Scalability to very large specifications could not be demonstrated.
Short Term Future Work

• **Further Case Studies + test execution**
• Add Timing Uncertainty + symbolic test cases
• Distinguish Input and Output actions
• Add modeling of environment assumptions + test purposes
• Loosen ERA restrictions
• New Engine for constraint solving
• **Moderate effort will enable an even larger application domain**
Long Term Future Work

- Hybrid Systems
- Performance Testing
A Continuous Time Testing-Preorder?

- Timed Hennessy tests does not fully characterize testing power of arbitrary timed automata
- Proposal for discrete Timed Process Language
  [Hennessy & Regan '95]
  - "extending the theory of tests for a language where time is not discrete will be a major challenge."

- Conjecture:
Pedestrian crossing

- Green time is at least 30 sec for cars and 20 for pedestrians
- 5 seconds red time in each direction at each switch
- Pedestrians requests passage
- Pedestrians may extend their green time with 20 sec at most 3 times in succession
Pedestrian Crossing Controller

```c
config
    observable req, car_gr, car_rd, ped_rd, ped_gr;
    int rq;
    system pedXing;
```
Selection Strategy: \textit{Comparison of partitioning}

- **Region**: Two (super) states are equivalent if they enable the same (set of) regions.
- **Stable Edge set**: Two (super) states are equivalent if they enable the same (set of) locations and edges. \textit{Equivalent wrt. single action Hennessy tests}.
- **Edges**: Two (super) states are equivalent if they enable the same edge.
- **Action**: Two (super) states are equivalent if they enable the same action.