Model-based *Testing of Real-time Systems*

Brian Nielsen
bnielsen@cs.aau.dk

With
Kim Larsen, Marius Mikucionis, Arne Skou
... and many others
Automated Model Based Conformance Testing

Does the **behavior** of the (blackbox) implementation **comply** to that of the specification?
Agenda

1. Introduction
2. Modeling and Conformance
   - Timed Automata
   - Real-Time-i/o Conformance, Environments
3. Offline Test generation
   - Using model-checkers a la Uppaal
   - Optimal test cases & Scheduling
   - Testing using Timed Games
4. Online Testing
   - Randomized online algorithm
   - Monitoring and Emulation
   - EKC-case study
5. Conclusions
Conformance Relation
Timed Coffee Machine

Possible users-model

Machine Model

x<=3

x<=5
Conformance Relation

- Timed Automata with Timed-LTS semantics
- **Input** actions (?) are controlled by the environment
- **Output** actions (!) are controlled by the implementation
- Implementations are *input enabled*
- **Testing hypothesis:** IUT can be modeled by some (unknown) TA
Does $I_n$ conform-to $S_1$?
Timed Conformance

• Derived from Tretman’s IOCO

• Let $\mathbf{I}$, $\mathbf{S}$ be timed I/O LTS, P a set of states
• $\text{TTr}(P)$: the set of timed traces from P
  • eg.: $\sigma = \text{coin}? \cdot 5 \cdot \text{req}? \cdot 2 \cdot \text{thinCoffee}! \cdot 9 \cdot \text{coin}?$
  • $\text{Out}(P \text{ after } \sigma) = \text{possible outputs and delays after } \sigma$
  • eg. out ($\{l2, x=1\}$): $\{\text{thinCoffee}, 0\ldots2\}$

$I \text{ rt-ioco } S = \text{def}$

• $\forall \sigma \in \text{TTr}(S): \text{Out}(I \text{ after } \sigma) \subseteq \text{Out}(S \text{ after } \sigma)$
• $\text{TTr}(I) \subseteq \text{TTr}(s)$ if $s$ and $I$ are input enabled

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

See also [Krichen&Tripakis, Khoumsi]
Does $I_n$ conform-to $S_1$?

$\sigma = \text{coin.give.10} \in \text{Tr}(I_1), \sigma \notin \text{Tr}(S_1)$

$\text{out}(I_1 \text{ after coin.give.3}) = \{0 \ldots \infty\}$

$\notin$

$\text{out}(S_1 \text{ after coin.give.3}) = \{\text{coffee}, 0 \ldots 2\}$
Does $I_n$ conform-to $S_1$?

$S_1$

1. coin?
2. give? $x=0$
3. $x \leq 5$
4. $x \geq 3$
5. coffee!

$I_3$

1. coin?
2. give? $x=0$
3. $x \leq 7$
4. $x \geq 3$
5. coffee!

$I_4$

1. coin?
2. give? $x=0$
3. $x \leq 5$
4. $x \geq 1$
5. coffee!

$\sigma = \text{coin.give.7.coffee}$
$\sigma \in TTr(I_3), \sigma \notin TTr(S_1)$

$\text{out}(I_3 \text{ after coin.give.7}) = \{\text{coffee,0}\}$
$\not\subset$

$\text{out}(I_4 \text{ after coin.give.1}) = \{\text{coffee,0...4}\}$
$\not\subset$

$\text{out}(S_1 \text{ after coin.give.7}) = \{\}$

$\text{out}(S_1 \text{ after coin.give.1}) = \{0...4\}$
Does $I_n$ conform-to $S_1$?

$\sigma = \text{coin.give.5.tea}$
$\sigma \in \text{Tr}(I7), \sigma \notin \text{Tr}(S1)$

But $\sigma$ was not specified

$\text{out}(I7 \text{ after coin.give.5}) = \{\text{tea, coffee,0}\}$
$\not\subset$
$\text{out}(S1 \text{ after coin.give.5}) = \{\text{coffee,0}\}$
• When T is high (low) switch on (off) cooling within r secs.
• When T is medium cooling may be either on or off (impl freedom)
Environment Modeling

- $E_M$: Any action possible at any time
- $E_1$: Only realistic temperature variations
- $E_2$: Temperature never increases when cooling
- $E_L$: No inputs (completely passive)

\[ E_L \subseteq E_2 \subseteq E_1 \subseteq E_M \]
Implementation relation

Relativized real-time io-conformance

- **E, S, I** are input enabled Timed LTS
- **Let \( P \) be a set of states**
- **\( TTr(P) \):** the set of timed traces from states in \( P \)
- **\( P \text{ after } \sigma = \)** the set of states reachable after timed trace \( \sigma \)
- **\( \text{Out}(P) \) =** possible outputs and delays from states in \( P \)

\[
\text{I rt-ioco}_E S \stackrel{\text{def}}{=} \forall \sigma \in TTr(E): \text{Out}((E,I) \text{ after } \sigma) \subseteq \text{Out}((E,S) \text{ after } \sigma)
\]

\[
\text{I rt-ioco}_E S \text{ iff } TTr(I) \cap TTr(E) \subseteq TTr(S) \cap TTr(E) // \text{input enabled}
\]

- **Intuition,** for all assumed environment behaviors, the IUT
  - **never produces illegal output,** and
  - **always produces required output in time**
Re-use Testing Effort

- Given I, E, S
- Assume I rt-ioco_E S

1. Given new (weaker) system specification S’
   
   \[ \text{If } S \subseteq S' \text{ then } I \text{ rt-ioco}_E S' \]

2. Given new (stronger) environment specification E’
   
   \[ \text{If } E' \subseteq E \text{ then } I \text{ rt-ioco}_{E'} S \]
Explicit Environments

- Realism and guiding
- Separation of concerns
- Modularity
- Creative tool uses
- Theoretical properties
Model-based Testing of Real-time Systems
Offline Test Generation

Brian Nielsen
bnielsen@cs.aau.dk

With
Kim Larsen, Marius Mikucionis, Arne Skou
... and many others
Time Optimal Real-Time Test Generation using UPPAAL

Anders Hessel, Paul Pettersson
Uppsala University Sweden

Kim Larsen, Brian Nielsen, Arne Skou
Aalborg University Denmark
Touch-sensitive Light-Controller

Interface

Switch

Dim

User
Timed Tests

**EXAMPLE** test cases for Interface

\[
\begin{align*}
0 \cdot \text{grasp!} \cdot 210 \cdot \text{release!} \cdot \text{touch}? \cdot & \text{PASS} \\
0 \cdot \text{grasp!} \cdot 317 \cdot \text{release!} \cdot \text{touch}? \cdot 2\frac{1}{2} \cdot \text{grasp!} \cdot 220 \cdot \text{release!} \cdot \text{touch}? \cdot & \text{PASS} \\
1000 \cdot \text{grasp!} \cdot 517 \cdot \text{starthold}? \cdot 100 \cdot \text{release!} \cdot \text{endhold}? \cdot & \text{PASS}
\end{align*}
\]

INFINITELY MANY SEQUENCES!!!!!!
Test generation using model-checking

System model
myProtocol.xml

Test purpose
Property
E<> connection.Established

Uppaal Model-Checker

Trace (witness)

Some
Random
Shortest
Fastest
testConnectionEstEst.trc

Use trace scenario as test case??!!
Controllable Timed Automata

- “DOUTA”-Model
  - **Determinism**: two transitions with same input/output leads to the same state
  - **Output Urgent**: enabled outputs will occur immediately
  - **Isolated Outputs**: if an output is enabled, no other output is enabled
  - **Input Enabled**: all inputs can always be accepted
Test Purposes 1

Test Purpose: 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(IRlease);
```
Test Purposes 2

**TP2:** Check that controller can enter location ‘DnPassive’: $E <> \text{Dim.DnPassive}$

**Shortest (and fastest) Test:**

If delay = 1000

```plaintext
out(IGrasp); silence(500); in(OSetLevel,0); out(IRelease); out(IGrasp); silence(500);
```
Test Purposes 2

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);
- 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’:

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

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. 

```
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);
```
Other Environments

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

\[\begin{align*}
T_{\text{react}} &= 2 \\
T_{\text{sw}} &= 4 \\
T_{\text{pause}} &= 5 \\
T_{\text{idle}} &= 20
\end{align*}\]

**TP1:**

*Fastest Test:*

$0 \cdot \text{touch!} \cdot 0 \cdot \text{dim?} \cdot 2 \cdot \text{touch!} \cdot 0 \cdot \text{bright?} \cdot \text{PASS}$

**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

![Diagram of a control flow graph with nodes labeled $l_1$ to $l_4$ and edges labeled $x=0$, $x \geq 2$, $a?$, $b!$, and $c!$.]
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
Location Coverage

- Test sequence traversing all locations
- Encoding:
  - Enumerate locations $1_0, \ldots, 1_n$
  - Add an auxiliary variable $1_i$ for each location
  - Label each ingoing edge to location $i$ $1_i := \text{true}$
  - Mark initial visited $1_0 := \text{true}$
- Check: $E<> (1_0 = \text{true} \land \ldots \land 1_n = \text{true})$
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 $e_i := \text{true}$
- Check: $E<> ( e_0 = \text{true} \land \ldots \land e_n = \text{true} )$
Fastest Edge Coverage

Cost = 12600 ms

Out(IGrasp);  // touch: switch light on
silence(200);
out(IGRasp);
silence(500); // hold
in(OSetLevel,0);
out(IGrasp); // touch: switch light off
out(IRelease);
silence(200);
in(OSetLevel,0);
out(IGrasp); // bring dimmer from ActiveUp
silence(500); // hold
in(OSetLevel,0);
out(IGrasp);
out(IRelease); // check release -> grasp is ignored
out(IGRasp); // @12400
out(IRelease);
silence(dfTolerance);
Definition/Use Pair Coverage

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

![Definition/use pair diagram]

- **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 $du[v_d,e_j] := \text{true}$

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

![Diagram](image)

- Same encodings and min-cost reachability
- Test sequence $\sigma = \varepsilon_0, i_0, ..., \varepsilon_1, i_1, \text{reset } \varepsilon_2, i_2, ..., \varepsilon_0, i_0, \text{reset}, \varepsilon_1, i_1, \varepsilon_2, i_2, ...$
- Test suite $T = \{\sigma_1, ..., \sigma_n\}$ with minimum cost
“Controllable” Timed I/O Automata

- Inputs (?) are controllable
- Outputs (!) are uncontrollable

Test case is a **preset sequence** of timed I/O actions
- Time and resource **optimal** tests can be generated
Offline Testing of Non-Deterministic TA

1. Compute "preset" timed input-sequence $\sigma_i$
2. Blindly Execute input sequence and log i/o sequence $\sigma_{io}$
3. Post mortem verdict evaluation by model-checking trace inclusion $\sigma_{io} \in TTr(M)$

**FAIL:** $\sigma_{io} \not\in TTr(M)$
**PASS:** INCONC $\sigma_{io} \in TTr(M)$ and goal-state possible reached
**INCONC:** $\sigma_{io} \in TTr(M)$ but goal state not reachable
Can be answered using Uppaal reachability analysis of $\sigma_{io} \parallel M$
Offline Testing of Non-Deterministic TA

1. Compute “preset” timed input-sequence $\sigma_i$
2. Blindly Execute input sequence and log i/o sequence $\sigma_{io}$
3. Post mortem verdict evaluation by model-checking trace inclusion $\sigma_{io} \in TTr(M)$

**FAIL**: $\sigma_{io} \notin TTr(M)$

**PASS**: INCONC $\sigma_{io} \in TTr(M)$ and goal-state possible reached

**INCONC**: $\sigma_{io} \in TTr(M)$ but goal state not reachable

Can be answered using Uppaal reachability analysis of $\sigma_{io} \; || \; M$
Optimal Tests

**Shortest** test for max light??

**Fastest** test for max light??

**Fastest** edge-covering test suite??

Least *power* consuming test??
Time-optimal test suites

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

- Other resources
  - Power
  - Mechanical wear
  - Manual operations
Test generation using Planning and Scheduling

Efficient algorithms and guiding for
Linearly Priced Timed Automata
Timed Automata + costs on transitions and locations

Cost of performing transition: transition cost

Cost of performing delay $\varepsilon : (\varepsilon \times$ location cost)

Trace:

$\langle a, x=y=0 \rangle \xrightarrow{4} \langle b, x=y=0 \rangle \xrightarrow{\varepsilon(2.5)} \langle b, x=y=2 \rangle \xrightarrow{0} \langle a, x=0, y=2 \rangle$

Cost of Execution Trace:

$\text{Sum of costs: } 4 + 5 + 0 = 9$
Verification vs. Optimization

- **Verification Algorithms:**
  - Checks a logical property of the entire state-space of a model.
  - Efficient Blind search.

- **Optimization Algorithms:**
  - Finds (near) optimal solutions.
  - Uses techniques to avoid non-optimal parts of the state-space (e.g. Branch and Bound).

- **Objective:**
  - Bridge gap between the two.
  - New techniques and applications in UPPAAL.
Cooperative Testing of Uncontrollable Timed Systems

Alexandre David, Kim G. Larsen, Shuhao Li and Brian Nielsen

bnielsen@cs.aau.dk

Center for Embedded Software Systems (CISS)
Aalborg University

Center for Indlejrede Software Systemer
BRICS
Basic Research in Computer Science
Controller Synthesis and Timed Games

Production Cell

**GIVEN** System moves $S$, Controller moves $C$, and property $\phi$

**FIND** strategy $s_C$ such that $s_C || S$ sat $\phi$

→ A Two-Player Game
“Controllable” Timed I/O Automata

Inputs (?) are controllable
Outputs (!) are uncontrollable

- Test case is a **preset sequence** of timed I/O actions
- Time and resource **optimal** tests can be generated
TA with Uncertainty

Inputs (?) are controllable
Outputs (!) are uncontrollable

Tidle = 20
Tsw = 4

Timing uncertainty of outputs
Multiple enabled outputs
The controller continuously observes all delays & moves

**Move:**

- *controllable edge:* $c$
- *delay:* $\lambda$

**Winning strategy:** a function that tells the controller how to move in any given state to win the game:

**Memoryless strategy:**

$$F : \text{State} \rightarrow E_c \cup \lambda$$

**Reachability Games:** Reach Goal

**Safety Games:** Avoid loose
Timed Games

a winning strategy:

L0:
\[
\begin{align*}
  x < 1 & : \lambda \\
  x = 1 & : c
\end{align*}
\]

L1:
\[
\begin{align*}
  x < 2 & : \lambda \\
  x \geq 2 & : c
\end{align*}
\]

L2:
\[
\begin{align*}
  x \leq 1 & : c
\end{align*}
\]

L3:
\[
\begin{align*}
  x < 1 & : \lambda \\
  x = 1 & : c
\end{align*}
\]
Test generation using TGA

- System Model
- Uppaal-Tiga Game Solver
- Strategy
- Control or Test Purpose Property
- Efficient zone-based on-the fly algorithms forward algorithm with a backward fix-point computation of the winning/losing sets.

Production Cell
Brick Sorting
Stable climate Control
Timed Games and Test Generation

- **Observable Timed Automata**
  - **Determinism:** two transitions with same input/output leads to the same state
  - **Time Uncertainty of outputs:** timing of outputs uncontrollable by tester
  - **Multiple Uncontrollable output:** IUT controls which enabled output will occur in what order
  - **Input Enabled:** all inputs can always be accepted
Timed I/O Games & Testing

Inputs (?) are controllable
Outputs (!) are uncontrollable

Off-line test-case generation =
Compute winning strategy for reaching **Bright**
Assign verdicts st. lost game means IUT not conforming
A trick light control

Tidle=20
Tsw=4

How to test for Bright?

E<> (control: A<> Bright)

or

<<c,u>> ◊ (<<c>> ◊ Bright)
Cooperative Strategies

Uppaal-Tiga extended to compute this partitioning motivated by testing applications.
Generate test case

- Choose & prune sub-tree of cooperative states
- Convert to suitable test notation (e.g., Timed automata) with verdicts according to RT-IOCO.

[Diagram showing model statespace with initial state, possibly winning states, winning states, loosing states, and INCONC states.]
Executing Test Strategies

winning strategy ≠ test case (lacks test verdicts)

- At each state \( s \)
  - The tester monitors outputs and delays
  - If a disallowed output or delay occurs (RT-IOCO), declare "FAIL";
    1. If \( s \) is cooperative, then according to \( F_c(s) \) either
       - offer a random enabled inputs to IUT or
       - delay random
    2. If \( s \) is winning, then deterministically according to \( F_w(s) \)
       - offer input to IUT or
       - delay
    3. If \( s \) is a goal-state, declare "PASS".
    4. If \( s \) is loosing, declare "INCONC"
- Untill verdict, or max test duration elapses
Online execution of Testing Games

Cooperative or Definitely (Winning) Strategy

UppAal-TRON
Strategy

Imp under test
Case Study

- The Leader Election Protocol [lamport05]
- To elect the node with the lowest id
- Time sensitive:
  \[ \text{timeout} = \text{INIT\_TO} + \text{leaderDist} \times \text{PropagationDelay} \]

(initial state)

\[
\begin{align*}
0 & \quad (0, 0) \\
1 & \quad (1, 0) \\
2 & \quad (2, 0)
\end{align*}
\]

\[
\begin{align*}
0 & \quad (0, 0) \\
1 & \quad (0, 1, 0, 0) \\
2 & \quad (0, 2, 0, 0)
\end{align*}
\]

\[
\begin{align*}
0 & \quad (0, 0) \\
1 & \quad (1, 0) \\
2 & \quad (2, 0, 1)
\end{align*}
\]

\[
\begin{align*}
0 & \quad (0, 0) \\
1 & \quad (0, 1) \\
2 & \quad (2, 1, 0, 1)
\end{align*}
\]

\[
\begin{align*}
0 & \quad (1, 2, 0, 1)
\end{align*}
\]

\[
\begin{align*}
\text{msg} & = (\text{source id, destination id, believed leader id, hops})
\end{align*}
\]

(node id)

(believed leader id, hops)
Model of a Node

idleClock<=Timeout+Timeout_Delay

deliverMsg?
  rMsg=envMsg

idleClock>Timeout
  timeout
  believedLeader=myId,
  leaderDistance=0,
  idleClock=0,
  Timeout=INIT_Timeout

gotMsg

((rMsg.leader==believedLeader) ||
 (rMsg.leader==believedLeader &&
  rMsg.distance>leaderDistance))
 || (rMsg.distance+1>MaxDistance)

betterMsg

idleClock=0
  Timeout=INIT_Timeout+LeaderDistance*M_Delay

forward

rMsg=nullMsg,
  betterInfo=0

forwardInfo()
Experiments of Winning Strategy Generation

**TP1**: control: $A \nneq (\text{IUT.betterInfo} == 1)$ and $\text{IUT.forward}$

**TP2**: control: $A \nneq \forall (i: \text{BufferId}) \ (\text{inUse}[i] == 1)$

**TP3**: control: $A \nneq \forall (i: \text{BufferId}) \ (\text{inUse}[i] == 1)$ and $\text{IUT.idle}$

<table>
<thead>
<tr>
<th></th>
<th>Time (s)</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n=3 4 5 6 7 8</td>
<td>n=3 4 5 6 7 8</td>
</tr>
<tr>
<td>TP1</td>
<td>0.03 0.14 0.7 3.1 11.1 33.5</td>
<td>0.1 4 9 28 85 242</td>
</tr>
<tr>
<td>TP2</td>
<td>0.81 2.13 8.4 67.1 452.0 /</td>
<td>11.2 33 88 462 2977 /</td>
</tr>
<tr>
<td>TP3</td>
<td>0.89 2.79 25.9 73.2 453.8 /</td>
<td>11.9 40 289 578 3015 /</td>
</tr>
</tbody>
</table>

Platform:
Dual-core 2.4GHz CPU, 4096MB RAM, Suse Linux Enterprise Desktop, UPPAAL-TIGA 0.9

"out of memory"

Protocol parameters:
- number of protocol nodes : $n$
- size of message buffer : $n$
- maximum distance between any two nodes : $n - 1$
- MSG_DELAY : 3 time units
- TIMEOUT_DELAY : 5 time units
- INIT_Timeout : 10 time units
Experiments of Cooperative Strategy Generation with Lossy Channels

These purposes do not have a winning strategy, but a cooperative one:

- TP1: control: A \( \exists \) (i:BufferId) (inUse[i]==1)
- TP2: control: A \( \forall \) (IUT.bufferInfo==1) and IUT.forward
- TP3: control: A \( \forall \) forall (i:BufferId) (inUse[i]==1)

<table>
<thead>
<tr>
<th></th>
<th>Time (s)</th>
<th></th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n=3 4 5 6 7 8</td>
<td>n=3 4 5 6 7 8</td>
<td></td>
</tr>
<tr>
<td>TP1</td>
<td>0.04 0.17 0.81 3.21 10.57 30.65</td>
<td>0.1 4.2 7.9 18.9 48.6 119.5</td>
<td></td>
</tr>
<tr>
<td>TP2</td>
<td>0.11 1.32 11.74 85.14 558.67 /</td>
<td>4.3 13.0 80.3 517.0 2958.9 /</td>
<td></td>
</tr>
<tr>
<td>TP3</td>
<td>3.22 75.56 / / / /</td>
<td>24.3 493.5 / / / /</td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

- Game-based approach to \textit{timed} conformance testing
  - “uncontrollable” timed systems: observable timed game automata
  - winning game strategy as test case
  - test execution algorithm
- Case study with the Leader Election Protocol shows promising performance
Future Work

- Partial observability
- Integration with TRON for cooperative test execution
- Practical evaluation of "success-rate" of cooperative strategies
END