Online Testing of Real-Time Systems using Uppaal-TRON

Brian Nielsen
Kim Larsen
Marius Mikucionis
Arne Skou

{bnielsen|kgl|marius|ask}@cs.aac.dk
MB-T&V of RTS using Uppaal

- **Lecture 1**
  - Timed Automata, Uppaal, Model Checking

- **Lecture 2**
  1. Optimal Scheduling
     - Time & Cost Optimal test generation (offline)
     - Deterministic, Output Urgent TA
  2. Controller Synthesis
     - Timed Games
     - Deterministic, timing uncertainty

- **Lecture 3**
  - Online real-time testing
    - Full non-determinism, full (input enabled) Uppaal-TA
    - Tool TRON was introduced & demo’ed in mondays exercise session
Agenda

- Introduction
- Correctness Criteria
  - IOCO
  - Real-Time
  - Relatizived IOCO
- Online Testing Algorithm
  - Non-determinism,
  - State-set computation
- Testing, Monitoring, Simulation, Environment Emulation
- Danfoss Case Study
- Conclusions Research Problems
Automated Model Based Conformance Testing

Model

Test suite

Test execution tool
- Event mapping
- Driver

Implementation Relation

Implementation Under Test

Does the behavior of the (blackbox) implementation comply to that of the specification?
Online Testing

- Test generated and executed event-by-event (randomly)
- A.K.A on-the-fly testing
Tron Framework

• **UppAal-TRON**: Testing Real-Time Systems Online
• Spec = UppAal Timed Automata Network: Env || IUT

“Relativized Real-Time i/o conformance” Relation

Timed Trace: $i_1.2\frac{1}{2}.o_1.3.o_2.19.i_2.5.i_3$
Correctness Criteria
I conforms-to S ??

[Jan Tretmans].
I conforms-to S ??

[Jan Tretmans].
I conforms-to S ??

[Jan Tretmans].
I conforms-to S ??

[IoCo]

[Jan Tretmans].
Tretman’s Ioco

“The” conformance relation used for blackbox testing of (untimed) reactive systems

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

\[ p \text{ after } \sigma = \{ p' \mid p \xrightarrow{\sigma} p' \} \]

\[ p \xrightarrow{\delta} p \text{ iff } \forall o! \in L_U \cup \{\tau\} : p \xrightarrow{o!} \]

\[ \text{out } (P) = \{ o! \in L_U \mid p \xrightarrow{!o} p \in P \} \]

\[ \cup \{ \delta \mid p \xrightarrow{\delta} p, p \in P \} \]

\[ \text{Straces } (s) = \{ \sigma \in (L \cup \{\delta\})^* \mid s \xrightarrow{\sigma} \} \]

[Jan Tretmans].
**Ioco**

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

\begin{align*}
\text{out}(i \text{ after coin?}) &= \{ \text{coffee!} \} \\
\text{out}(i \text{ after token?}) &= \{ \text{tea!} \} \\
\text{out}(s \text{ after coin?}) &= \{ \text{coffee!} \} \\
\text{out}(s \text{ after token?}) &= \emptyset
\end{align*}

But token? \notin \text{Straces}(s)

[Jan Tretmans].
**Ioco**

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

\[ \text{out}(i \text{ after coin?}) = \{ \text{coffee!} \} \]
\[ \text{out}(i \text{ after token?}) = \{ \text{tea!} \} \]

\[ \text{out}(s \text{ after coin?}) = \{ \text{coffee!} \} \]
\[ \text{out}(s \text{ after token?}) = \{ \text{tea!} \} \]

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

\[ \text{out}(\text{i after token?}) = \{ \delta \} \quad \text{out}(s \text{ after token?}) = \{ \text{tea!} \} \]

[Jan Tretmans].
Ioco

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

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

[Jan Tretmans].
Timed Conformance??

Example Traces

- c?.2.r?.2.weakC
- c?.5.r?.4.strongC
- c?.5.r?.7

I1  rt-ioco  S
I2  rt/ioco  S
Real-Time conformance

- **TTr(s)**: the set of *timed traces* from s
  - eg.: \( \sigma = \text{coin?}.5.\text{req?}.2.\text{weakCoffee}!.9.\text{coin?} \)
- **Out(s after \( \sigma \))**: possible outputs and delays after \( \sigma \)
  - eg. out ({l_2,x=1}): \{weakCoffee,0...2\}

\[ \text{i rt-ioco } s = \text{def} \]
- \( \forall \sigma \in \text{TTr}(s): \text{Out}(i \text{ after } \sigma) \subseteq \text{Out}(s \text{ after } \sigma) \)
- \( \text{TTr}(i) \subseteq \text{TTr}(s) \)

- **Intuition**
  - no illegal output is produced and
  - required output is produced (at right time)
**Sample Cooling Controller**

**IUT-model**
- 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)

**Env-model**
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

- Let $P$ be a set of states
- $\mathcal{T}Tr(P)$: the set of timed traces from states in $P$
- $P$ after $\sigma$ = the set of states reachable after timed trace $\sigma$
- $\text{Out}(P)$ = possible outputs and delays in $P$

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

- $I \text{ rt-ioco}_E S$ iff $\mathcal{T}Tr(I) \cap \mathcal{T}Tr(E) \subseteq \mathcal{T}Tr(S) \cap \mathcal{T}Tr(E)$

- Intuition, for all relevant environment behaviors
  - never produces illegal output, and
  - always produces required output in time
- $\sim$ timed trace inclusion
Re-use Testing Effort

- Given I, E, S
- Assume I \text{rt-ioco}_E S

1. Given new (weaker) system specification \( S' \)

\[
\text{If } S \sqsubseteq S' \text{ then } I \text{rt-ioco}_E S' \]

2. Given new (stronger) environment specification \( E' \)

\[
\text{If } E' \sqsubseteq E \text{ then } I \text{rt-ioco}_E' S \]
An Algorithm
IDEA: State-set tracking

- Dynamically compute all potential states that the model \( M \) can reach after the timed trace \( \varepsilon_0, i_0, \varepsilon_1, o_1, \varepsilon_2, i_2, o_2, \ldots \)

- \( Z = M \) after \( (\varepsilon_0, i_0, \varepsilon_1, o_1, \varepsilon_2, i_2, o_2) \)

- If \( Z = \emptyset \) the IUT has made a computation not in model: **FAIL**

- \( i \) is a relevant input in Env iff \( i \in \text{EnvOutput}(Z) \)
(Abstract) Online Algorithm

Algorithm TestGenExe \((S, E, IUT, T)\) returns \{pass, fail\}
\[Z := \{(s_0, e_0)\}.\]

\[\text{while } Z \neq \emptyset \land \# \text{iterations} \leq T \text{ do either randomly:}
\]

1. // offer an input
   \[\text{if } \text{EnvOutput}(Z) \neq \emptyset\]
   \[\text{randomly choose } i \in \text{EnvOutput}(Z)\]
   \[\text{send } i \text{ to } IUT\]
   \[Z := Z \text{ After } i\]

2. // wait d for an output
   \[\text{randomly choose } d \in \text{Delays}(Z)\]
   \[\text{wait (for } d \text{ time units or output } o \text{ at } d' \leq d)\]
   \[\text{if } o \text{ occurred then}\]
   \[Z := Z \text{ After } d'\]
   \[Z := Z \text{ After } o \text{ // may become } \emptyset (\Rightarrow \text{fail})\]
   \[\text{else}\]
   \[Z := Z \text{ After } d \text{ // no output within } d \text{ delay}\]

3. restart:
   \[Z := \{(s_0, e_0)\}, \text{ reset } IUT \text{ //reset and restart}\]

   \[\text{if } Z = \emptyset \text{ then return fail else return pass}\]
Online Algorithm

Algorithm $TestGenExe (S, E, IUT, T)$ returns \{\text{pass, fail}\)

$Z := \{(s_0, e_0)\}$.

\textbf{while} $Z \neq \emptyset \land \#\text{iterations} \leq T$ \textbf{do either} randomly:

1. \quad \text{// offer an input}
   \begin{itemize}
   \item \quad \text{if} $EnvOutput(Z) \neq \emptyset$
   \item \quad \text{randomly choose} $i \in EnvOutput(Z)$
   \item \quad \textbf{send} $i$ to $IUT$
   \item \quad $Z := Z \setminus \{(s, e)\}$
   \end{itemize}

2. \quad \text{// wait for a delay or output}
   \begin{itemize}
   \item \quad \text{randomly choose} $d \in \text{Delays}(Z)$
   \item \quad \textbf{wait} (for $d$ time units or output $o$ at $d'$ ≤ $d$)
   \item \quad \text{if} $o$ occurred
   \item \quad \quad $Z := Z \setminus \{(s, e)\}$
   \item \quad \quad \textbf{if} $o \notin \text{ImpOutput}(Z)$ \textbf{then return fail}
   \item \quad \quad \textbf{else} $Z := Z \setminus \{(s, e)\}$ \textbf{After} $o$
   \item \quad \quad \textbf{else}$
   \item \quad \quad \quad $Z := Z \setminus \{(s, e)\}$ \textbf{After} $d$ \quad \textbf{// no output within $d$ delay}
   \end{itemize}

3. \quad \textbf{restart:}
   \begin{itemize}
   \item \quad $Z := \{(s_0, e_0)\}$, \textbf{reset} $IUT$ \textbf{//reset and restart}
   \end{itemize}

\textbf{if} $Z = \emptyset$ \textbf{then return fail} \textbf{else return pass}

\begin{itemize}
\item Sound
\item Complete (as $T \to \infty$)
\end{itemize}
(Under some technical assumptions)
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 $Z$ be a set of states

**Z after $a$:** possible states after $a$ (and $\tau^*$)

\[
\begin{align*}
{l_0} & \xrightarrow{x \geq 7, a} {l_1} \\
{l_0} & \xrightarrow{a, x := 0} {l_3} \\
{l_1} & \xrightarrow{\tau} {l_4} \\
{l_3} & \xrightarrow{a} {l_2}
\end{align*}
\]

\[
\{ \langle l_0, x=3 \rangle \} \text{ after } a = \\
\{ \langle l_2, x=3 \rangle, \langle l_4, x=3 \rangle, \langle l_3, x=0 \rangle \}
\]

**Z after $\varepsilon$:** possible states after $\tau^*$ and $\varepsilon_i$, totaling a delay of $\varepsilon$

\[
\begin{align*}
{l_0} & \xrightarrow{\tau, x := 0} {l_1} \\
{l_0} & \xrightarrow{a} {l_2} \xrightarrow{\tau} {l_4} \\
{l_3} & \xrightarrow{a} {l_2}
\end{align*}
\]

\[
\{ \langle l_0, x=0 \rangle \} \text{ after } 4 = \\
\{ \langle l_0, x=4 \rangle, \langle l_1, 0 \leq x \leq 4 \rangle \}
\]

\[
\langle l_0, x=0 \rangle \xrightarrow{1} \langle l_0, x=1 \rangle \xrightarrow{\tau} \langle l_1, x=0 \rangle \xrightarrow{3} \langle l_1, x=3 \rangle
\]
State-set Operations

$Z \text{ after } a$: possible states after action $a$ (and $\tau^*$)

$Z \text{ after } \varepsilon$: possible states after $\tau^*$ and $\varepsilon_i$, totaling a delay of $\varepsilon$

- Can be computed efficiently using the symbolic data structures and algorithms in UppAal
Symbolic Interpretation

- **Zone** is a conjunction of clock constraints of the form:
  \[ \{x_i - x_j < c_{ij}\} \cup \{a_i < x_i\} \cup \{x_j < b_j\} \text{ where } < \in \{\leq, =\} \]

- **Difference bound matrix - compact representation.**

- **Symbolic state set** \( Z = \{\langle l_1, z_1\rangle, \ldots, \langle l_n, z_n\rangle\} \)

- **Action transition:** \( \langle l, z\rangle \xrightarrow{a} \langle l', (z \land g)_{\text{r}}, I(l')\rangle \): \( l \xrightarrow{g,a,r} l' \) is an action transition and \( z \land g \neq \emptyset, (z \land g)_{\text{r}} \land I(l') \neq \emptyset \).

- **Delay transition:** \( \langle l, z\rangle \xrightarrow{\delta} \langle l, z^{+\delta} \land I(l)\rangle \) iff \( z^{+\delta} \land I(l) \neq \emptyset \).

\[
z = [(y - x \leq 4) \land (y \geq 5) \land (x \leq 3)]
\]
Tron: implementation

Graphical User Interface (java)
- editor
- simulator
- verifier

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

Zones & Reachability, Etc

State-set explorer
Online Test Generation

Online Test Generation

System Under Test

Adapter API

Physical I/O
Online Testing Example

Symbolic state set:
\[ \{ (k_0 l_0, 0 \leq x \leq 0) \} \]

EnvOutput: \{ coin \}
EnvInput: \{ \}
ImpOutput: \{ \}

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

Testing-UPPAAL

Symbolic state set:
\( \{ (k_0l_0, 0 \leq x \leq 0) \} \)

EnvOutput: \{ coin \}

EnvInput: \( \emptyset \)

ImpOutput: \( \emptyset \)

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

Testing−UPPAAL

Symbolic state set:
\( \{\langle k_1 l_1, 0 \leq x \leq 0 \rangle \} \)

EnvOutput: \( \{\text{req}\} \)

EnvInput: \( \emptyset \)

ImpOutput: \( \emptyset \)

Update the state set
and other variables
Online Testing

Symbolic state set:
\{ (k_1, l_1, 0 \leq x \leq 0) \}
EnvOutput: \{ req \}
EnvInput: \emptyset
ImpOutput: \emptyset

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

Symbolic state set:
\{ (k_1, l_1, 5 \leq x \leq 5) \}

EnvOutput: \{ req \}

EnvInput: \emptyset

ImpOutput: \emptyset

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

Testing-UPPAAL

Symbolic state set:
\{ \langle k_1l_1, 5 \leq x \leq 5 \rangle \}

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

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

Testing–UPPAAL

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

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: {weakCoffee, strongCoffee}
EnvInput: {weakCoffee, strongCoffee}
ImpOutput: {weakCoffee, strongCoffee}

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

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

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

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

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

Testing–UPPAAL

Symbolic state set:
\[ \{ (k_2l_3, 4 \leq x \leq 4) \} \]

EnvOutput: \emptyset

EnvInput: \{ strongCoffee \}

ImpOutput: \{ strongCoffee \}

Implementation

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

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

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

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

Testing–UPPAAL

Symbolic state set:
\[
\emptyset
\]
EnvOutput: \[
\emptyset
\]
EnvInput: \[
\emptyset
\]
ImpOutput: \[
\emptyset
\]

..no output so far: update the state set.. (!)
Testing, Monitoring, Simulation, Environment Emulation
Real System

Plant
Continuous

System Under Test

Physical I/O
Monitoring

Passively listen and check observed trace
Environment Emulation

TRON-Env Emulator

Environment Model

Plant
Continuous

System Under Test

Adapter

Physical I/O

Adapter API
Simulator / prototype

![Diagram of a simulator/prototype system with labels:
- Plant
- Continuous
- Adapter
- ENV=System Model
- TRON
- Physical I/O
]
Testing = Environment Emulation + Monitoring
Offline Verdict Evaluation

Only Env Emulation is required in real-time
Danfoss EKC Case
Electronic Cooling Controller

Sensor Input
- air temperature sensor
- defrost temperature sensor
- (door open sensor)

Keypad Input
- 2 buttons (~40 user settable parameters)

Output Relays
- compressor relay
- defrost relay
- alarm relay
- (fan relay)

Display Output
- alarm / error indication
- mode indication
- current calculated temperature

Optional real-time clock or LON network module
Industrial Cooling Plants
The Collaboration

- **Goals**
  - Can we model significant aspects and time constraints?
  - Can we test in real-time?
  - Is the tool fast enough?
  - How do we control and observe target?
  - Detect errors??

- **Means**
  - Existing product
  - Documentation
    - requirements specification
    - users manuals
    - equipment and software for real test execution
    - Meeting and e-mail with Danfoss Engineers

- **Continued collaboration**
  - Test of new generation controllers being developed
  - Improved test interface
  - Test Case Language & Automatic Execution System
Basic Refrigeration Control

- **highAlarm Limit**: When the temperature exceeds the high alarm limit, the compressor will start.
- **setpoint + differential**: Compressor starts when the temperature is above the setpoint plus differential.
- **differential**: Temperature difference between setpoint and limit.
- **lowAlarm Limit**: Compressor stops when the temperature drops below the low alarm limit.
- **start compressor**: Compressor starts when the temperature reaches the high alarm limit.
- **stop compressor**: Compressor stops when the temperature drops below the low alarm limit.
- **normal**: Normal operation when temperature is within the setpoint.
- **min restart time not elapsed**: Must wait before restarting.
- **min cooling time not elapsed**: Must wait before cooling.
- **alarm delay**: Delay before the start alarm is indicated.

*Time*
Refrigeration Control (2)

- Sensor errors (open/short circuit): safety mode with duty cycle based on
  - historical compressor on/off times
  - or fixed (uninterrupted #cutins >72)

- Several defrost modes
  - periodic (eg. 8 hrs)
  - manual
  - RT-clock
  - temperature diff. based
  - electrical or hot gass

- Alarms after a defrost further postponed (90 mins)

- Fan control
  - cooling mode / defrost mode

- ~40 user settable parameters
**EKC Adaptation 1**

- Read and write parameter “database”
- 47 parameters

**EKC Software Layering**

- Control Software
- Parameter DB (shared variables)
- Device drivers+kernel
- Hardware+Physical I/O

Test Interface

- LON ➔ GW ➔ RS232

- AK-Online (PC SW)
  - configuration
  - supervision
  - logging

- win32+OLE+VB
EKC Adaptation 2

TRON Engine

compressorOn

setTemp(20)

Adaptor

Old copy

New copy

“continous” readout

2 readouts/s

22.3 0 1

“par#4=20.0”

Need better test interface!

• Read-only parameters
• Delay and synchronization

win32+OLE+VB

Solaris/Linux (C++)
Main Model Components

- Significant sub-set of functionality
  - 18 concurrent timed automata
  - 14 clocks, 14 integers
Temperature Tracking

“periodic” weighted average:

\[ T_n = \frac{T_{n-1} \times 4 + T_{\text{sampled}}}{5} \]

Temperature Tracking diagram:
- EKC calculated temperature
- Model calculated temperature
- Error/uncertainty envelope
- Tolerance in value computation
- Tolerance in sampling time
- CompressorOn!
Reverse Engineering

- Unclear and incomplete specifications
- Method of Working
  1. Formulate hypothesis model
  2. Test
  3. **FAIL**-verdict ⇒ Refine model
  4. **(PASS)** ⇒ Confirm with Danfoss
- Detects differences between actual and modeled behavior
- *Indicates promising error-detection capability*
- 4 examples
Ex1: Control Period

- Control actions issued when "calculatedTemp" crosses thresholds

  "periodic" weighted average: \[ T^n = \frac{T_{n-1} \times 4 + T_{sampled}}{5} \]

- No requirements on period given
- Tested to be 1.2 seconds
Clearing the alarm do not switch off alarm state, only alarm relay
Ex2: High Alarm Monitor v2

- Add HighAlarmDisplay action
- Add location for “noSound, but alarmDisplaying”
- (Postpone alarms after defrosting)
Ex3: Defrosting and Alarms

- When defrosting the temperature rises, therefore postpone high temperature alarms during defrost
- System parameter `alarmDelayAfterDefrost`
- Several Interpretations
  - Postpone `alarmDelayAfterDefrost + alarmDelay` after defrost?
  - Postpone `alarmDelayAfterDefrost + alarmDelay` after `highTemp` detected?
  - Postpone `alarmDelayAfterDefrost` until temperature becomes low; then use `alarmDelay`
- Option 3 applies!
Ex4: Defrost Time Tolerance

- Defrost relays engaged earlier and disengaged later than expected
- Assumed 2 seconds tolerance
- Defrosting takes long time
- Implementation uses a low resolution timer (10 seconds)
Example Test Run
State-set Evolution (EKC)

Correlation between state-sets and model behavior
Cost of state-set update

$\mu S$

Number of Symbolic states

Average after Delay CPU time, microseconds

Initial state set size
Conclusions

- Real-Time Online testing from timed automata is feasible, but
  - Both theoretically and technically very challenging
  - Many open research issues
Research Problems

- Testing Theory
- Efficient data structures and algorithms for state set computation
- Diagnosis & Debugging
- Guiding and Coverage Measurement
- Probabilistic & Hybrid extensions
- Real-Time execution of TRON
  - Controller Synthesis (Observability / Implementability)
  - Scheduling
- Adaptor Abstraction
Related Work

- **Formal Testing Frameworks**
  - [Brinksma, Tretmans]

- **Real-Time Implementation Relations**
  - [Khoumsi’03, Briones’04]

- **Symbolic Reachability analysis of Timed Automata**
  - [Dill’89, Larsen’97,...]

- **Online state-set computation**
  - [Tripakis’02]

- **Online Testing**
  - [Tretmans’99, Peleska’02, Krichen’04]
Coverage Evaluation

- Convert observed concrete timed trace to a timed automata
- Replace env-model by trace automaton
- Coverage Criteria Decorated IUT-model
- Reachability Analysis

- **Possibly** Covered
  \[ \text{E}<> \text{ENV\_trace.end and } e[1]==1 \]

- **Definitely** Covered
  \[ A[] \text{ ENV\_trace.end imply } e[1]==1 \]

- **Not** Covered
  \[ A[] \text{ ENV\_trace.end imply } e[1]==0 \]