An introduction to Uppaal and Timed Automata
What is Uppaal?
(http://www.uppaal.com/)

• A simple graphical interface for drawing extended finite state machines (automatons + shared variables)
• A graphical simulator – including MSC’s
• An analyser (model checker)
• In addition, Uppaal supports the notion of timed automatons
Transitions in Uppaal

• Automata transitions are labelled with the following (optional) parts:
  A set of guards on variables
  A label (input? or output!)
  A set of variable assignments

• A transition can be taken when:
  • All guards are true
  • A synchronization is possible with another process
Transitions in Uppaal

- $i<3$ holds
- Counter and incrementer may synchronize
A Uppaal system

- Consists of a set (network) of automata
- System state = snapshot of each machine's control location + local variables + global variables
Parallel Composition: interleaving

Flipper:
- States: \( s_0, s_1 \)
- Transitions: \( s_0 \xrightarrow{\text{flip\_slide}} s_1 \)
- 2 states

Speaker:
- States: \( s_0, s_1, s_2 \)
- Transitions: \( s_0 \xrightarrow{\text{think}} s_1 \xrightarrow{\text{talk}} s_2 \)
- 3 states

Lecturer = Speaker \( || \) Flipper
- States: \( s_0, s_1 \)
- Transitions: \( (s_0,s_0) \xrightarrow{\text{think}} (s_0,s_1) \xrightarrow{\text{talk}} (s_0,s_2) \)
- 2*3 states
Home-Banking?

int accountA, accountB; //Shared global variables
//Two concurrent bank costumers

Thread costumer1 () {
    int a,b; //local tmp copy
    a=accountA;
    b=accountB;
    a=a-10; b=b+10;
    accountA=a;
    accountB=b;
}

Thread costumer2 () {
    int a,b;
    a=accountA;
    b=accountB;
    a=a-20; b=b+20;
    accountA=a;
    accountB=b;
}

• Are the accounts in balance after the transactions?
Home Banking

A[] (pc1.finished and pc2.finished) imply (accountA+accountB==200)?
Home Banking

int accountA, accountB; //Shared global variables
Semaphore A,B;       //Protected by sem A,B
//Two concurrent bank costumers

Thread customer1() {
    int a, b; //local tmp copy
    down(A);
    down(B);
    a = accountA;
    b = accountB;
    a = a - 10; b = b + 10;
    accountA = a;
    accountB = b;
    up(A);
    up(B);
}

Thread customer2() {
    int a, b;
    down(B);
    down(A);
    a = accountA;
    b = accountB;
    a = a - 20; b = b + 20;
    accountA = a;
    accountB = b;
    up(B);
    up(A);
}
Semaphore FSM Model

Binary Semaphore

- States: Open, Closed
- Transitions: Up, Down

Counting Semaphore

- States: Counting, Up
- Variables: c
- Transitions: c:=init_count, c:=c-1, c:=c+1, Up?
Semaphore Solution?

1. Race conditions?
2. Consistency? (Balance)
3. Deadlock?

1. A[] (mc1.finished and mc2.finished) imply (accountA+accountB==200)
2. E<> mc1.critical_section and mc2.critical_section
3. A[] not (mc1.finished and mc2.finished) imply not deadlock
Reachability Analysis

• Compute *all* possible execution sequences
• And consequently *all* states of the system
• *Exhaustive search* $\implies$ *proof*
• Check if each state encountered has the (un)-desired property
UPPAAL Property Specification Language

- $A[] ~ p$
- $A<> ~ p$
- $E<> ~ p$
- $E[] ~ p$
- $p ~\rightarrow~ q$

process location  data guards  clock guards

$p ::= a.l ~|~ gd ~|~ gc ~|~ p ~and~ p ~|$
$p ~or~ p ~|~ not ~p ~|~ p ~imply ~p ~|$
$(p) ~|~ \text{deadlock} \text{(only for } A[], E<>\text{)}$

$A[] ~ (\text{mc1.finished and mc2.finished}) ~\text{imply} ~ (\text{accountA}+\text{accountB}==200)$
Uppaal “Computation Tree Logic”

- **E<> p**: Possible
- **A[] p**: always
- **E[] p**: potentially always
- **A<> p**: inevitable
- **p --> q**: leads-to
Reachability Analysis

Passed:=Ø     //already seen states
Waiting:={S_0} //states not examined yet

While(waiting!=Ø) {
    Waiting:=Waiting\{s_i\}
    if s_i ∉ Passed
        whenever (s_j → s_j) then
            waiting:=waiting ∪ s_j
    }

Depth First: maintain waiting as a stack
Breadth First: maintain waiting as a queue
(shortest counter example)

Order: 0 1 3 6 7 4 8 2 5 9
Order: 0 1 2 3 4 5 6 7 8 9
Hybrid & Real Time Systems

Control Theory  

Plant
*Continuous*

Computer Science

Controller Program  
*Discrete*

**Eg.:**  
Pump Control  
Air Bags  
Robots  
Cruise Control  
ABS  
CD Players  
Production Lines

Real Time System  
A system where correctness not only depends on the logical order of events but also on their timing
**Timed Automata**

**Intelligent Light Control**

**WANT:** if press is issued twice quickly then the light will get brighter; otherwise the light is turned off.
Solution: Add real-valued clock $x$
Timed Automata

(Alur & Dill 1990)

Clocks: $x, y$

Guard
Boolean combination of comp with integer bounds

Reset
Action performed on clocks

State
$(\text{location}, x=v, y=u)$ where $v,u$ are in $\mathbb{R}$

Transitions

$(n, x=2.4, y=3.1415) \xrightarrow{a} (m, x=0, y=3.1415)$

$(n, x=2.4, y=3.1415) \xrightarrow{e(1.1)} (n, x=3.5, y=4.2415)$
Timed Safety Automata =
Timed Automata + Invariants

Clocks: \( x, y \)

Transitions

\(( n, x=2.4, y=3.1415 )\)  \(e(3.2)\)

\(( n, x=2.4, y=3.1415 )\)  \(e(1.1)\)

\(( n, x=3.5, y=4.2415 )\)

Invariants ensure progress!!

Location Invariants

\( x \leq 5 \)

\( y \leq 10 \)

(Henzinger et al, 1992)
Clock Constraints

For set $C$ of clocks with $x, y \in C$, the set of clock constraints over $C$, $\Psi(C)$, is defined by

$$\alpha ::= x < c \mid x - y < c \mid \neg \alpha \mid (\alpha \land \alpha)$$

where $c \in \mathbb{N}$ and $\prec \in \{<, \leq\}$.
Timed (Safety) Automata

A timed automaton $A$ is a tuple $(L, l_0, E, Label, C, clocks, guard, inv)$ with

- $L$, a non-empty, finite set of locations with initial location $l_0 \in L$
- $E \subseteq L \times L$, a set of edges
- $Label : L \rightarrow 2^{AP}$, a function that assigns to each location $l \in L$ a set $Label(l)$ of atomic propositions
- $C$, a finite set of clocks
- $clocks : E \rightarrow 2^C$, a function that assigns to each edge $e \in E$ a set of clocks $clocks(e)$
- $guard : E \rightarrow \Psi(C)$, a function that labels each edge $e \in E$ with a clock constraint $guard(e)$ over $C$, and
- $inv : L \rightarrow \Psi(C)$, a function that assigns to each location an invariant.
Timed Automata: Example

\[
x \geq 2
\]
\[
\{ x \}
\]
Timed Automata: Example

\[ x \geq 2 \]
\[ \{ x \} \]

Guard
Location
Reset

Value of \( x \)

Time

2 4 6 8 10
Timed Automata: Example

\[ x \leq 3 \]

\[ x \geq 2 \]

\[ \{x\} \]
Timed Automata: Example

\[ x \leq 3 \]

\[ \{ x \} \quad x \geq 2 \]

Value of \( x \):

\[ \begin{array}{c|c}
   \text{Time} & \text{Value} \\
   \hline
   2 & 2 \\
   4 & 3 \\
   6 & 4 \\
   8 & 3 \\
   10 & 2 \\
\end{array} \]
Timed Automata: Example

\[
\begin{align*}
2 & \leq x \leq 3 \\
\{x\}
\end{align*}
\]
Timed Automata: Example

\[ 2 \leq x \leq 3 \]
\[ \{ x \} \]

value of \( x \)

4
3
2
2 4 6 8 10
Light Switch

\[ \begin{align*}
push & \quad \frac{x \geq 2}{\{x, y\}} \\
\text{off} & \quad y \leq 9 & \quad \text{on} & \quad \frac{x \geq 2}{\{x\}} \\
\text{click} & \quad \frac{y = 9}{\{x\}}
\end{align*} \]
Light Switch

- Switch may be turned on whenever at least 2 time units have elapsed since last “turn off”

\[
\begin{align*}
\text{push} &\quad x \geq 2 \\
\{x, y\} &\quad y \leq 9
\end{align*}
\]

\[
\begin{align*}
\text{click} &\quad y = 9 \\
\{x\} &\quad x \geq 2
\end{align*}
\]

\[
\begin{align*}
\text{push} &\quad x \geq 2 \\
\{x\} &
\]
Light Switch

- Switch may be turned on whenever at least 2 time units has elapsed since last “turn off”
- Light automatically switches off after 9 time units.
Semantics

• **clock valuations**: \( V(C) \quad v : C \rightarrow R \geq 0 \)

• **state**: \((l, v)\) where \( l \in L \) and \( v \in V(C) \)

• Semantics of timed automata is a **labeled transition system** \((S, \rightarrow)\)
  
  where \( S = \{(l, v) \mid v \in V(C) \text{ and } l \in L\} \)

• **action transition** \( (l, v) \xrightarrow{a} (l', v') \) iff \( l \xrightarrow{g \ a \ r} l' \)
  
  \( g(v) \) and \( v' = v[r] \) and \( \text{Inv}(l')(v') \)

• **delay Transition** \( (l, v) \xrightarrow{d} (l, v + d) \) iff
  
  \( \text{Inv}(l)(v + d') \) whenever \( d' \leq d \in R \geq 0 \)
Semantics: Example

\[(\text{off}, x = y = 0) \xrightarrow{3.5} (\text{off}, x = y = 3.5) \xrightarrow{\text{push}} (\text{on}, x = y = \pi) \xrightarrow{\text{push}} (\text{on}, x = 3, y = \pi + 3) \xrightarrow{9-(\pi+3)} (\text{off}, x = 0, y = 9) \ldots\]
Uppaal

Network of timed automata

Timing requirement

No!
Debugging Information

Yes

Uppsala (6 persons), Aalborg (10 persons), 1995-
21 papers, 6 invited talks/tutorials
9 industrial case studies
http://www.docs.uu.se/docs/rtmv/uppaal/index.shtml
(or www.uppaal.com)
Timed Automata in UPPAAL

• Networks of Timed Safety Automata
  + urgent actions
  + urgent locations
    (i.e. zero-delay locations)
  + committed locations
    (i.e. zero-delay and \textbf{atomic} locations)
  + data-variables (integers with bounded domains)
  + arrays of data-variables
  + guards and assignments over data-variables and arrays...
Networks of Timed Automata

+ Integer Variables + arrays ....

Example transitions

\((l1, m1, \ldots, x=2, y=3.5, i=3, \ldots,) \xrightarrow{\text{tau}} (l2, m2, \ldots, x=0, y=3.5, i=7, \ldots,)\)

\((l1, m1, \ldots, x=2.2, y=3.7, I=3, \ldots,)\)

Two-way synchronization on complementary actions.

Closed Systems!
Timed Automata in UPPAAL

**clock assignments**

\[
\begin{align*}
x &:= n \\
i &:= \text{Expr} \\
\text{Expr} &:= i | i[\text{Expr}] | n | -\text{Expr} | \text{Expr} + \text{Expr} | \text{Expr} - \text{Expr} | \text{Expr} \times \text{Expr} | \text{Expr} / \text{Expr} | (\text{gd} ? \text{Expr} : \text{Expr})
\end{align*}
\]

**clock guards**

\[
\begin{align*}
g &:= g_c | g_d | g, g \\
g_c &:= x < n | x \leq y + n \\
g_d &:= \text{Expr} \text{ op } \text{Expr} \\
\langle &\in \{<, \leq, =, >, =, >\} \\
op &\in \{<, \leq, =, >, =, >\}
\end{align*}
\]

**data guards**

\[
\begin{align*}
\text{inv} &:= x < n | x \leq n | \text{inv}, \text{inv}
\end{align*}
\]

**location invariants**

\[
\begin{align*}
x &\leq 5 \\
\text{clock guards} \\
\text{data guards} \\
\text{and}
\end{align*}
\]
Urgent Channels

urgent chan hurry;

Informal Semantics:
• There will be no delay if transition with urgent action can be taken.

Restrictions:
• No clock guard allowed on transitions with urgent actions.
• Invariants and data-variable guards are allowed.
Urgent Locations

Click “Urgent” in State Editor.

Informal Semantics:
• No delay in urgent location.

Note: the use of urgent locations reduces the number of states in a model, and thus the complexity of the analysis.
Click “Committed” in State Editor.

Informal Semantics:
- **No delay** in committed location.
- Next transition must involve automata in committed location.

**Note:** the use of committed locations **reduces** the number of states in a model, and allows for more space and time efficient analysis.
Urgent and Committed Locations

\begin{align*}
(m \mid p, x = 0) \\
2.5 \\
(m \mid p, x = 2.5) \\
d \rightarrow (n \mid r, x = 2.5) \\
a \rightarrow (n \mid q, x = 2.5) \\
\downarrow \quad \downarrow \quad \downarrow \\
(n \mid q, x = 2.5 + d) \\
\downarrow \\
(o \mid q, x = 0) \\
\downarrow \\
(o \mid r, x = 0) \\
\downarrow \\
(o \mid q, x = 2.5 + d)
\end{align*}
Uppaal Demo
Exercise: The Coffee Machine

Machine
- takes time to brew
- time-out if coffee not taken before time-limit

Observer
- complain if more than 8 time-units between two consecutive publ.