Modelling, Specification, and Verification using UPPAAL

Kim Guldstrand Larsen
Modelling using Finite State Machines
Modelling processes

- A process is the execution of a sequential program.
- Modeled as a finite state machine (LTS)
  - transits from state to state
  - by executing a sequence of *atomic* actions.

---

A light switch

\[
\begin{array}{ccc}
  & \text{on} & \\
  \text{s0} & \text{off} & \text{s1} \\
\end{array}
\]

on \rightarrow off \rightarrow on \rightarrow off \rightarrow on \rightarrow off \rightarrow \ldots \ldots \ldots

---

Kim G. Larsen
Modelling Choices

• Who or what makes the choice?
• Is there a difference between input and output actions?
Non-deterministic Choice

- Tossing a coin

  ![Diagram of a non-deterministic choice with states s2, s0, and s1, transitions for toss, tails, and heads, and a 50/50 probability for fairness.]

- Possible traces?
  - Both outcomes possible
  - Nothing said about relative frequency
  - If coin is fair, the outcome is 50/50
Non-Deterministic Choice modelling failure

How do we model an unreliable communication channel which accepts packets, and if a failure occurs produces no output, otherwise delivers the packet to the receiver?

Use non-determinism...
Internal-Actions

- Spontaneous actions
- Internal actions
- Tau-actions

Internal transitions can be taken on the initiative of a single machine without communication with others
Extended FSM

• EFSM = FSM + variables + enabling conditions + assignments
  - Transition still atomic
  - Can be translated into FSM if variables have bounded domain

• State: control location + variable values:

  (state, amount, capacity)
  - (s0, 5, 10)
Parallel Composition: interleaving

Flipper

Speaker

Lecturer = Speaker || Flipper

Kim G. Larsen
Process Interaction

- ! = Output, ? = Input
- Handshake communication
- Two-way

Coffee Machine

Lecturer

4 states

University = Coffee Machine || Lecturer

- LTS?
- How many states?
- Traces?

4 states

synchronization results in internal actions

4 states: Interaction constrain overall behavior
Adding Time
Collaborators

@UPPsala
- Wang Yi
- Paul Pettersson
- John Håkansson
- Anders Hessel
- Pavel Krcal
- Leonid Mokrushin
- Shi Xiaochun

@AALborg
- Kim G Larsen
- Gerd Behrman
- Arne Skou
- Brian Nielsen
- Alexandre David
- Jacob I. Rasmussen
- Marius Mikucionis
- Thomas Chatain

@Elsewhere
- Emmanuel Fleury, Didier Lime, Johan Bengtsson, Fredrik Larsson, Kåre J Kristoffersen, Tobias Amnell, Thomas Hune, Oliver Möller, Elena Fersman, Carsten Weise, David Griffioen, Ansgar Fehnker, Frits Vandraager, Theo Ruys, Pedro D’Argenio, J-P Katoen, Jan Tretmans, Judi Romijn, Ed Brinksma, Martijn Hendriks, Klaus Havelund, Franck Cassez, Magnus Lindahl, Francois Laroussinie, Patricia Bouyer, Augusto Burgueno, H. Bowmann, D. Latella, M. Massink, G. Faconti, Kristina Lundqvist, Lars Asplund, Justin Pearson...
Real Time Systems

Eg.: Realtime Protocols
     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!!
Real Time Model Checking

**Plant**
*Continuous*

**Controller Program**
*Discrete*

Model of tasks (automatic?)

Model of environment (user-supplied / non-determinism)

UPPAAL Model

SAT $\phi$ ??
Real Time Control Synthesis

Plant
Continuous

Controller Program
Discrete

Model of environment (user-supplied)

Partial UPPAAL Model

Synthesis of tasks/scheduler (automatic)

sensors

actuators

SAT !!
Real-time Model-Based Testing

Plant
Continuous

Controller Program
Discrete

Test generation (offline or online) wrt. Design Model

Conforms-to?

UPPAAL Model

inputs

outputs
UPPAAL

Graphical Design Tool
- timed automata =
  - state machines
  + clocks
- communication
- datatypes
- user defined functions
- cost variable

Diagram:
- States: Idle, Ready, Running, Error
- Transitions: t<=L[id], t>=E[id], t=D[id], t=0, e=id, e=id, run?
- Conditions: ax==C[id], done!, ax=0
Informationsteknologi

UPPAAL

Graphical Simulator
• visualization and recording
• inexpensive fault detection
• inspection of error traces
• Message Sequence Charts
• (Gantt Charts)
Informationsteknologi

UPPAAL

Verifier

- Exhaustive & automatic checking of requirements
- .. including validating, safety, liveness, bounded liveness and response properties
- .. generation of debugging information for visualisation in simulator.
- Optimal scheduling for cost models
"Impact

UPPAAL downloads

\[ y = 3,236x^2 - 13,841x + 582.21 \]

Google:

- UPPAAL: 134,000
- SPIN Verifier: 242,000
- nuSMV: 57,700

> 1,500

Google Scholar Citations
(Rhapsody/Esterel < 3,500)
Impact

Academic Courses @

DTU, MCI, IT-U (DK)
Chalmers,
Linköping, Lund,
Chalmers,
Mälardalarn (S)
Nijmegen, Twente, CWI (NL)
Upenn, Northumbria (US)
Braunschweig,
Oldenborg, Marktoberdorf (D)
Tsinghua, Shanghai, ISS, NUS (Asia)
Impact

Tutorials Given @

Estonian School (01)
IPA Fall Days (01)
FTRTFT (01)
CPN (02)
SFM (02)
MOVEP (02)
DISC School (03)
MOVEP (04)
PRISE (05)
PDMC (05)
ARTIST2 (05)
EMSOFT (05)
RTSS (05)
TECS week (06)
TAROT (06)
ARTS (06)
GLOBAN (06)
ARTIST ASIAN SCH (07)
Impact

Company Downloads
Mecel
Jet
Symantec
SRI
Relogic
Realwork
NASA
Verified Systems
Microsoft
ABB
Airbus
PSA
Saab
Siemens
Volvo
Lucent Technologies
Timed Automata
Alur & Dill 1989
Dumb Light Control

**WANT:** if press is issued twice quickly then the light will get brighter; otherwise the light is turned off.
Dumb Light Control \textit{Alur & Dill 1990}

Solution: Add real-valued clock $x$
Timed Automata review

Alur & Dill 1990

Clocks:

- **Guard**: Boolean combination of integer bounds on clocks
- **Reset**: Action performed on clocks

State

\[(location, x=v, y=u)\] where \(v,u\) are in \(\mathbb{R}\)

Transitions

- Discrete Trans
  - \((n, x=2.4, y=3.1415) \xrightarrow{a} (m, x=0, y=3.1415)\)
- Delay Trans
  - \((n, x=2.4, y=3.1415) \xrightarrow{e(1.1)} (n, x=3.5, y=4.2415)\)

\[x \leq 5 & y > 3\]

Action used for synchronization

\[x := 0\]
Timed Automata

Alur & Dill 1990

States:
( location , x=v) where v∈R

Transitions:
( Off, x=0 )

x: real-valued clock

Reset

Synchronizing action

Off

Light

Bright

x:=0

x>3

x≤3

Guard Conjunctions of x~n

press?

press?

press?
Timed Automata

Alur & Dill 1990

States:
( location , x=v) where \( v \in \mathbb{R} \)

Transitions:
( Off , x=0 )
delay 4.32
\rightarrow ( Off , x=4.32 )
Timed Automata

Alur & Dill 1990

States:
( location , x=v) where v∈R

Transitions:
( Off , x=0 )
delay 4.32 → ( Off , x=4.32 )
press? → ( Light , x=0 )
Timed Automata

Alur & Dill 1990

**States:**
( location , x=v ) where v∈R

**Transitions:**

- ( Off , x=0 )
- delay 4.32 → ( Off , x=4.32 )
- press? → ( Light , x=0 )
- delay 2.51 → ( Light , x=2.51 )

 resets

synchronizing actions

x=0

x≤3

x>3

x: real-valued clock

Guard Conjunctions of x~n

( Off , x=0 )

( Off , x=4.32 )

( Light , x=0 )

( Light , x=2.51 )
Timed Automata

Alur & Dill 1990

States:
( location , x=v) where v∈R

Transitions:

( Off , x=0 )
delay 4.32  →  ( Off , x=4.32 )
press?    →  ( Light , x=0 )
delay 2.51  →  ( Light , x=2.51 )
press?    →  ( Bright , x=2.51 )

x: real-valued clock

Synchronizing action

Guard Conjunctions of x~n

Reset

x:=0

x>3

x≤3

press?

press?
Intelligent Light Control

Using Invariants
Timed Automata review

Invariants

Clocks: $x, y$

Transitions

$\begin{align*}
  ( n, x=2.4, y=3.1415 ) & \xrightarrow{e(3.2)} \\
  ( n, x=2.4, y=3.1415 ) & \xrightarrow{e(1.1)} \\
  ( n, x=3.5, y=4.2415 )
\end{align*}$

Invariants ensure progress!!
Intelligent Light Control

Using Invariants

Transitions:

\[ (\text{Off}, x=0) \]
- delay 4.32 \rightarrow (\text{Off}, x=4.32)
- press? \rightarrow (\text{Light}, x=0)
- delay 4.51 \rightarrow (\text{Light}, x=4.51)
- press? \rightarrow (\text{Light}, x=0)
- delay 100 \rightarrow (\text{Light}, x=100)
- \tau \rightarrow (\text{Off}, x=0)

Note:

\[ (\text{Light}, x=0) \text{ delay 103 } \rightarrow \]

Invariants ensures progress
Example

With two clocks

Reachable?
Example

With two clocks

y := 0
L0

x := 0
b

y <= 2
a

x <= 2
c

y <= 2, x = 4

L1

Reachable?

(x = 0, y = 0, L0)
Example

With two clocks

$\text{(L0, } x=0, y=0) \rightarrow \varepsilon(1.4)$

$\text{(L0, } x=1.4, y=1.4) \rightarrow$
Example
With two clocks

Reachable?
Example

*With two clocks*

(L0, x=0, y=0) → _ε_(1.4)
(L0, x=1.4, y=1.4) → _a_
(L0, x=1.4, y=0) → _ε_(1.6)
(L0, x=3.0, y=1.6) → _a_
(L0, x=3.0, y=0)
Networks  Light Controller & User

\[ x := 0 \]

**Synchronization**

\[ y \geq 10 \]

**Transitions:**

- \((\text{Off}, \text{Rest}, x=0, y=0)\)  \(\rightarrow (\text{Off}, \text{Rest}, x=20, y=20)\)
- \((\text{Press}!\)\)  \(\rightarrow (\text{Light}, \text{Busy}, x=0, y=0)\)
- \((\text{Delay} 2)\)  \(\rightarrow (\text{Light}, \text{Busy}, x=2, y=2)\)
- \((\text{Press}!\)\)  \(\rightarrow (\text{Bright}, \text{Rest}, x=0, y=0)\)
Networks of Timed Automata
(a’la CCS)

Example transitions

$(l1, m1, \ldots, x=2, y=3.5, \ldots)$  $\xrightarrow{\tau}$  $(l2, m2, \ldots, x=0, y=3.5, \ldots)$

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

If a URGENT CHANNEL

Two-way synchronization on complementary actions.

Closed Systems!
Timed Automata

Formally
Constraints

Definition
Let $X$ be a set of clock variables. The set $\mathcal{B}(X)$ of clock constraints $\phi$ is given by the grammar:

$$\phi \ ::= \ x \leq c \mid c \leq x \mid x < c \mid c < x \mid \phi_1 \land \phi_2$$

where $c \in \mathbb{N}$ (or $\mathbb{Q}$).
Clock Valuations and Notation

Definition
The set of clock valuations, $\mathbb{R}^C$ is the set of functions $C \rightarrow \mathbb{R}_{\geq 0}$ ranged over by $u, v, w, \ldots$.

Notation
Let $u \in \mathbb{R}^C$, $r \subseteq C$, $d \in \mathbb{R}_{\geq 0}$, and $g \in \mathcal{B}(X)$ then:

- $u + d \in \mathbb{R}^C$ is defined by $(u + d)(x) = u(x) + d$ for any clock $x$

- $u[r] \in \mathbb{R}^C$ is defined by $u[r](x) = 0$ when $x \in r$ and $u[r](x) = u(x)$ for $x \not\in r$.

- $u \models g$ denotes that $g$ is satisfied by $u$. 
Timed Automata

Definition
A timed automaton $A$ over clocks $C$ and actions $Act$ is a tuple $(L, l_0, E, I)$, where:

- $L$ is a finite set of locations
- $l_0 \in L$ is the initial location
- $E \subseteq L \times \mathcal{B}(X) \times Act \times \mathcal{P}(C) \times L$ is the set of edges
- $I : L \rightarrow \mathcal{B}(X)$ assigns to each location an invariant
Semantics

Definition
The semantics of a timed automaton $A$ is a labelled transition system with state space $L \times \mathbb{R}^C$ with initial state $(l_0, u_0)^*$ and with the following transitions:

- $(l, u) \xrightarrow{\epsilon(d)} (l, u + d)$ iff $u \in I(l)$ and $u + d \in I(l)$,
- $(l, u) \xrightarrow{a} (l', u')$ iff there exists $(l, g, a, r, l') \in E$ such that
  - $u \models g$,
  - $u' = u[r]$, and
  - $u' \in I(l')$

*$u_0(x) = 0$ for all $x \in C$
Timed Automata: Example

\[ 2 \leq x \leq 3 \]
\[ \{ x \} \]
**Timed Automata: Example**

- **Location**: l
- **Action**: a
- **Guard**: $2 \leq x \leq 3$
- **Reset-set**: $\{x\}$

Graph showing the value of $x$ over time with points at 2, 4, and 6.
Timed Automata: Example

\[ x \leq 3 \]

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

Invariant
Timed Automata: Example

Informationsteknologi

UCb
Brick Sorting
LEGO Mindstorms/RCX

- **Sensors**: temperature, light, rotation, pressure.
- **Actuators**: motors, lamps,
- **Virtual machine**:
  - 10 tasks, 4 timers, 16 integers.
- **Several Programming Languages**:
  - NotQuiteC, Mindstorm, Robotics, legOS, etc.

![3 output ports](image1)
![1 infra-red port](image2)
![3 input ports](image3)
A Real Real Timed System

The Plant
Conveyor Belt & Bricks

Controller Program
LEGO MINDSTORM
First UPPAAL model

Sorting of Lego Boxes

Ken Tindell

Exercise: Design **Controller** so that **black** boxes are being pushed out
NQC programs

```c
int active;
int DELAY;
int LIGHT_LEVEL;

task MAIN{
    DELAY=75;
    LIGHT_LEVEL=35;
    active=0;
    Sensor(IN_1, IN_LIGHT);
    Fwd(OUT_A,1);
    Display(1);
    start PUSH;
    while(true){
        wait(IN_1<=LIGHT_LEVEL);
        ClearTimer(1);
        active=1;
        PlaySound(1);
        wait(IN_1>LIGHT_LEVEL);
    }
}

task PUSH{
    while(true){
        wait(Timer(1)>DELAY && active==1);
        active=0;
        Rev(OUT_C,1);
        Sleep(8);
        Fwd(OUT_C,1);
        Sleep(12);
        Off(OUT_C);
    }
}
```
A Black Brick

Diagram: State machine or control flow diagram with states and transitions labeled with conditions and values such as `pos=9`, `pos=18`, `pos=81`, `pos=90`, `on1`, `sensor`, `on2`, `piston`, `end`, `start`, `ok?`, `remove?`, and `off`.
Control Tasks & Piston

GLOBAL DECLARATIONS:
const int ctime = 75;
int[0,1] active;
clock x, time;
chan eject, ok;
urgent chan blck, red, remove, go;
From RCX to UPPAAL – and back

- Model includes Round-Robin Scheduler.
- Compilation of RCX tasks into TA models.
- Presented at ECRTS 2000 in Stockholm.

- From UPPAAL to RCX: Martijn Hendriks.
The Production Cell in LEGO

Course at DTU, Copenhagen

Rasmus Crüger Lund
Simon Tune Riemanni
Light Control Interface
Light Control Interface

User

Interface

Light

Control Program

press?

release?

touch!

starthold!

endhold!

L++/L--/L:=0

L++/L-/-L:=0
Light Control Interface

User

Control Program

Interface

- press?
- release?
- endhold!
- touch!
- press?
- release?
- x:=5
- x<=8
- x:=10
- x<=10
- starthold!
- endhold!

Switch

Dim

- L:=OL,
- x:=0,
- on:=1

- L<Max
- x:=delay
- x:=0

- on:=0
- touch?
- L:=L+1
- x:=0

BRICS
Basic Research
in Computer Science
Networks of Timed Automata
(a’la CCS)

Example transitions

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

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

If a URGENT CHANNEL

Two-way synchronization on complementary actions.

Closed Systems!
Network Semantics

\[ T_1 \parallel_x T_2 = (S_1 \times S_2, \rightarrow, s_0^1 \parallel_x s_0^2) \]

where

\[ S_1 \xrightarrow{\mu} S_1' \]

\[ S_1 \parallel_x S_2 \xrightarrow{\mu} S_1 \parallel_x S_2' \]

\[ S_2 \xrightarrow{\mu} S_2' \]

\[ S_1 \parallel_x S_2 \xrightarrow{\mu} S_1 \parallel_x S_2' \]

\[ S_1 \xrightarrow{a!} S_1' \]

\[ S_1 \parallel_x S_2 \xrightarrow{\tau} S_1 \parallel_x S_2' \]

\[ S_2 \xrightarrow{a?} S_2' \]

\[ S_1 \parallel_x S_2 \xrightarrow{e(d)} S_1 \parallel_x S_2' \]

\[ S_2 \xrightarrow{e(d)} S_2' \]

\[ S_1 \parallel_x S_2 \xrightarrow{e(d)} S_1 \parallel_x S_2' \]
Network Semantics
(URGENT synchronization)

\[ T_1 \parallel T_2 = (S_1 \times S_2, \rightarrow, s_0^1 \parallel s_0^2) \]

where

\[ S_1 \xrightarrow{\mu} S_1' \]
\[ S_1 \parallel S_2 \xrightarrow{\mu} S_1' \parallel S_2 \]

\[ S_2 \xrightarrow{\mu} S_2' \]
\[ S_1 \parallel S_2 \xrightarrow{\mu} S_1' \parallel S_2' \]

\[ S_1 \xrightarrow{a!} S_1' \]
\[ S_2 \xrightarrow{a?} S_2' \]
\[ S_1 \parallel S_2 \xrightarrow{\tau} S_1' \parallel S_2' \]

\[ S_1 \xrightarrow{e(d)} S_1' \]
\[ S_2 \xrightarrow{e(d)} S_2' \]
\[ S_1 \parallel S_2 \xrightarrow{e(d)} S_1' \parallel S_2' \]

\[ \forall d' < d, \forall u \in UAct: \]
\[ e(d') u? \quad e(d') u! \]
\[ \neg (\neg s_1 \rightarrow \rightarrow \land s_2 \rightarrow \rightarrow ) \]
Light Control Network
Validation

Light Controller
Druzba: The Shower Problem
The Druzba MUTEX Problem
The Druzba MUTEX Problem
The Druzba MUTEX Problem

Using the light as semaphor
Overview of the UPPAAL Toolkit
UPPAAL’s architecture

GUI (Java)
uppaal2k.jar

Server

Engine (C++)

Linux, Windows, Solaris, MacOS

CLI

xml

xta

ta
Train Crossing

Stopable Area

[10,20]

[7,15]

[3,5]

Queue

Gate

River

Crossing
Train Crossing

Communication via channels and shared variable.
Timed Automata in UPPAAL
/*
 * For more details about this example, see
 * "Automatic Verification of Real-Time Communicating Systems by Constraint Solving",
 * by Wang Yi, Paul Pettersson and Mats Daniels. In Proceedings of the 7th International
 */

const N 5;   // # trains + 1
int[0,N] el;
chan appr, stop, go, leave;
chan empty, notempty, hd, add, ren;

clock x;

int[0,N] list[N], len, i;

Train1:=Train(el, 1);
Train2:=Train(el, 2);
Train3:=Train(el, 3);
Train4:=Train(el, 4);

system
    Train1, Train2, Train3, Train4,
    Gate, Queue;
Declarations in UPPAAL

- The syntax used for declarations in UPPAAL is similar to the syntax used in the C programming language.

- **Clocks:**
  - **Syntax:**
    - `clock x1, ..., xn ;`
  - **Example:**
    - `clock x, y;` declares two clocks: x and y.
Declarations in UPPAAL (cont.)

- **Data variables**
  - Syntax:
    - int n1, … ;
    - int[l,u] n1, … ;
    - int n1[m], … ;

  Integer with “default” domain.
  Integer with domain “l” to “u”.
  Integer array w. elements n1[0] to n1[m-1].

  Example:
  - int a, b;
  - int[0,1] a, b[5][6];
Declarations in UPPAAL (cont.)

- **Actions** (or channels):
  - Syntax:
    - chan a, ... ;
    - urgent chan b, ... ;

  - Example:
    - chan a, b;
    - urgent chan c;

Ordinary channels. Urgent actions (see later)
Declarations UPPAAL (cont.)

- **Constants**
  - Syntax:
    - ```
      const int c1 = n1;
    ```
  - Example:
    - ```
      const int[0,1] YES = 1;
      const bool NO = false;
    ```
Timed Automata in UPPAAL

Discrete Variables

Safe
  appr!
  e:=id, x:=0

Appr
  x<=20
  x<=10, e:=id

Start
  x<=15
  x:=0
  go?
  e:=id

Stop
  x:=0
  stop?

Cross
  x<=5
  x:=0
  leave!

Safe
  x>=3

Resets

invariants

Guards

Synchronizations
Timed Automata in UPPAAL

\[ \text{inv} := x < \text{Expr} | x \leq \text{Expr} | \text{inv}, \text{inv} \]

\[
\begin{align*}
i &:= \text{Expr} \\
\text{Expr} &:= i | i[\text{Expr}] | n | \neg \text{Expr} | \\
& \quad \text{Expr} + \text{Expr} | \\
& \quad \text{Expr} - \text{Expr} | \\
& \quad \text{Expr} \ast \text{Expr} | \\
& \quad \text{Expr} / \text{Expr} | \\
& \quad (g_d ? \text{Expr} : \text{Expr})
\end{align*}
\]
Expressions

used in

- guards,
- invariants,
- assignments,
- synchronizations
- properties,
Expressions

Expression ::= ID
| NAT
| Expression '[' Expression ']
| '(' Expression ')' 
| Expression '++' | '++' Expression 
| Expression '--' | '--' Expression 
| Expression AssignOp Expression 
| UnaryOp Expression 
| Expression BinOp Expression 
| Expression '?' Expression ':' Expression 
| ID '.' ID
Operators

Unary

`'-' | '!' | 'not'`

Binary

`'<' | '<=' | '==' | '!=' | '>= | '>'
'+' | '-' | '*' | '/' | '%' | '
'| '|' | '^' | '<<' | '>>' | '&' | '&&' | '||' | 'and' | 'or' | 'imply'

Assignment

`':=' | '+=' | '-=' | '*=' | '/=' | '%='
| '=' | '&=' | '^=' | '<<=' | '>>='`
Guards, Invariants, Assignments

**Guards:**
- It is side-effect free, type correct, and evaluates to boolean
- Only clock variables, integer variables, constants are referenced (or arrays of such)
- Clocks and differences are only compared to integer expressions
- Guards over clocks are essentially conjunctions (I.e. disjunctions are only allowed over integer conditions)

**Assignments**
- It has a side effect and is type correct
- Only clock variable, integer variables and constants are referenced (or arrays of such)
- Only integer are assigned to clocks

**Invariants**
- It forms conjunctions of conditions of the form $x < e$ or $x \leq e$ where $x$ is a clock reference and $e$ evaluates to an integer
Synchronization

Binary Synchronization

- Declared like:
  ```
  chan a, b, c[3];
  ```
- If a is channel then:
  - `a!` = Emmission
  - `a?` = Reception

- Two edges in different processes can synchronize if one is emitting and the other is receiving on the same channel.

Broadcast Synchronization

- Declared like
  ```
  broadcast chan a, b, c[2];
  ```
- If a is a broadcast channel:
  - `a!` = Emmission of broadcast
  - `a?` = Reception of broadcast

- A set of edges in different processes can synchronize if one is emitting and the others are receiving on the same b.c. channel. A process can always emit. Receivers MUST synchronize if they can. No blocking.
More on Types

- Multi dimensional arrays
  - e.g. int b[4][2];

- Array initialiser:
  - e.g. int b[4] := { 1, 2, 3, 4 };

- Arrays of channels, clocks, constants.
  - e.g.
  - chan a[3];
  - clock c[3];
  - const k[3] { 1, 2, 3 };

- Broadcast channels.
  - e.g. broadcast chan a;
Templates may be parameterised:

- `int v; const min; const max`
- `int[0,N] e; const id`

Templates are instantiated to form processes:

- `P := A(i, 1, 5);`
- `Q := A(j, 0, 4);`
- `Train1 := Train(e1, 1);`
- `Train2 := Train(e1, 2);`
Extensions

Select statement

- models a non-deterministic choice
- \( x : \text{int}[0,42] \)

Types

- Record types
- Type declarations
- Meta variables:
  - not stored with state
  - \text{meta int} x;

Forall / Exists expressions

- \text{forall} (x:\text{int}[0,42]) \text{expr}
  - true if \text{expr} is true for all values in [0,42] of \( x \)
- \text{exists} (x:\text{int}[0,4]) \text{expr}
  - true if \text{expr} is true for some values in [0,42] of \( x \)

Example:
\[
\text{forall} (x:\text{int}[0,4]) \text{array}[x];
\]
Urgency & Commitment

Urgent Channels

- No delay if the synchronization edges can be taken!
- No clock guard allowed.
- Guards on data-variables.
- Declarations:
  urgent chan a, b, c[3];

Urgent Locations

- No delay – time is freezeed!
- May reduce number of clocks!

Committed Locations

- No delay.
- Next transition MUST involve edge in one of the processes in committed location
- May reduce considerably state space
Queries:
Specification Language
Logical Specifications

- **Validation Properties**
  - Possibly: \( E \leftrightarrow P \)

- **Safety Properties**
  - Invariant: \( A[] P \)
  - Pos. Inv.: \( E[] P \)

- **Liveness Properties**
  - Eventually: \( A \leftrightarrow P \)
  - Leadsto: \( P \rightarrow Q \)

- **Bounded Liveness**
  - Leads to within: \( P \rightarrow_{\leq t} Q \)

The expressions \( P \) and \( Q \) must be type safe, side effect free, and evaluate to a boolean.

Only references to integer variables, constants, clocks, and locations are allowed (and arrays of these).
Logical Specifications

- **Validation Properties**
  - Possibly: \( E<> P \)

- **Safety Properties**
  - Invariant: \( A[P] \)
  - Pos. Inv.: \( E[P] \)

- **Liveness Properties**
  - Eventually: \( A<> P \)
  - Leadsto: \( P \rightarrow Q \)

- **Bounded Liveness**
  - Leads to within: \( P \rightarrow_{\leq t} Q \)
Logical Specifications

- **Validation Properties**
  - Possibly: $E<> P$

- **Safety Properties**
  - Invariant: $A[] P$
  - Pos. Inv.: $E[] P$

- **Liveness Properties**
  - Eventually: $A<> P$
  - Leadsto: $P \rightarrow Q$

- **Bounded Liveness**
  - Leads to within: $P \rightarrow_{\leq t} Q$
Logical Specifications

- **Validation Properties**
  - Possibly: $E<> P$

- **Safety Properties**
  - Invariant: $A[] P$
  - Pos. Inv.: $E[] P$

- **Liveness Properties**
  - Eventually: $A<> P$
  - Leadsto: $P \rightarrow Q$

- **Bounded Liveness**
  - Leads to within: $P \rightarrow_{\leq t} Q$
Logical Specifications

- Validation Properties
  - Possibly: $E<> P$

- Safety Properties
  - Invariant: $A[] P$
  - Pos. Inv.: $E[] P$

- Liveness Properties
  - Eventually: $A<> P$
  - Leadsto: $P \rightarrow Q$

- Bounded Liveness
  - Leads to within: $P \rightarrow_{\leq t} Q$
Train Crossing

Communication via channels and shared variable.
Gear Controller

with MECEL AB

Lindahl, Pettersson, Yi 1998

Flowgraph
Gear Controller

with MECEL AB

Requirements

\[
\begin{align*}
\text{GearControl@Initiate} & \sim \leq 1500 \ ( \ ErrStat = 0 ) \Rightarrow \text{GearControl@GearChanged} \\
\text{GearControl@Initiate} & \sim \leq 1000 \\
\quad & ( \ ( ErrStat = 0 \land UseCase = 0 ) \Rightarrow \text{GearControl@GearChanged} ) \\
\text{Clutch@ErrorClose} & \sim \leq 200 \ & \text{GearControl@CCloseError} \\
\text{Clutch@ErrorOpen} & \sim \leq 200 \ & \text{GearControl@COpenError} \\
\text{GearBox@ErrorIdle} & \sim \leq 350 \ & \text{GearControl@GSetError} \\
\text{GearBox@ErrorNeu} & \sim \leq 200 \ & \text{GearControl@GNeuError} \\
\text{Inv} ( \ & \text{GearControl@CCloseError} \Rightarrow \text{Clutch@ErrorClose} ) \\
\text{Inv} ( \ & \text{GearControl@COpenError} \Rightarrow \text{Clutch@ErrorOpen} ) \\
\text{Inv} ( \ & \text{GearControl@GSetError} \Rightarrow \text{GearBox@ErrorIdle} ) \\
\text{Inv} ( \ & \text{GearControl@GNeuError} \Rightarrow \text{GearBox@ErrorNeu} ) \\
\text{Inv} ( \ & \text{Engine@ErrorSpeed} \Rightarrow \text{ErrStat} \neq 0 ) \\
\text{Inv} ( \ & \text{Engine@Torque} \Rightarrow \text{Clutch@Closed} ) \\
\bigwedge_{i \in \{1, N, 1, \ldots, 5\}} \text{Poss} ( \text{Gear@Gear}_i ) \\
\bigwedge_{i \in \{N, 1, \ldots, 5\}} \text{Inv} ( \ ( \text{GearControl@Gear} \land \text{Gear@Gear}_i ) \Rightarrow \text{Engine@Torque} )
\end{align*}
\]
UPPAAL 3.4

Gate Template

IntQueue

int[0,N] list[N], len, i;
UPPAAL 3.6 (3.5) with C-Code

```c
int[0,N] list[N], len;

void enqueue(int[0,N] element) {
    list[len++] = element;
}

void dequeue() {
    int i = 0;
    len -= 1;
    while (i < len) {
        list[i] = list[i + 1];
        i++;
    }
    list[i] = 0;
    i = 0;
}

bool isEmpty() {
    return len == 0;
}

int[0,N] hd() {
    return list[0];
}
```
Case-Studies: Controllers

- Gearbox Controller [TACAS’98]
- Bang & Olufsen Power Controller [RTPS’99, FTRTFT’2k]
- SIDMAR Steel Production Plant [RTCSA’99, DSVV’2k]
- Real-Time RCX Control-Programs [ECRTS’2k]
- Experimental Batch Plant (2000)
- RCX Production Cell (2000)
- Terma, Verification of Memory Management for Radar (2001)
- Scheduling Lacquer Production (2005)
- Memory Arbiter Synthesis and Verification for a Radar Memory Interface Card [NJC’05]
Case Studies: Protocols

- Philips Audio Protocol [HS’95, CAV’95, RTSS’95, CAV’96]
- Collision-Avoidance Protocol [SPIN’95]
- Bounded Retransmission Protocol [TACAS’97]
- Bang & Olufsen Audio/Video Protocol [RTSS’97]
- TDMA Protocol [PRFTS’97]
- Lip-Synchronization Protocol [FMICS’97]
- Multimedia Streams [DSVIS’98]
- ATM ABR Protocol [CAV’99]
- ABB Fieldbus Protocol [ECRTS’2k]
- Distributed Agreement Protocol [Formats05]
- Leader Election for Mobile Ad Hoc Networks [Charme05]
UPPAAL is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata, extended with data types (bounded integers, arrays, etc.).

The tool is developed in collaboration between the Department of Information Technology at Uppsala University, Sweden and the Department of Computer Science at Aalborg University in Denmark.

Download

The current official release is UPPAAL 3.4.11 (Jun 23, 2005). A release of UPPAAL 3.6 alpha 3 (Dec 20, 2005) is also available. For more information about UPPAAL version 3.4, we refer to this press release.

License

The UPPAAL tool is free for non-profit applications. For information about commercial licenses, please email sales(at)uppaal(dot)com.

To find out more about UPPAAL, read this short introduction. Further information may be found at this website in the pages About, Documentation, Download, and Examples.

Mailing Lists

UPPAAL has an open discussion forum group at Yahoo!Groups intended for users of the tool. To join or post to the forum, please refer to the information at the discussion forum page. Bugs should be reported using the bug tracking system. To email the development team directly, please use uppaal(at)list(dot)it(dot)uu(dot)se.