Real Time Model Checking using UPPAAL

Kim G Larsen
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 Illum Rasmussen
- Marius Mikucionis

@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

A system where correctness not only depends on the logical order of events but also on their **timing**!!

Eg.: Realtime Protocols
Pump Control
Air Bags
Robots
Cruise Control
ABS
CD Players
Production Lines
Real Time Model Checking

Plant
Continuous

Controller Program
Discrete

sensors
actuators

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

UPPAAL Model

Model of tasks (automatic?)
Real Time Control Synthesis

**Plant**
*Continuous*

**Controller Program**
*Discrete*

Model of environment (user-supplied)

Partial UPPAAL Model

Synthesis of tasks/scheduler (automatic)

sensors

actuators
Model-Checking

Model: $A$

Requirement Specification: $F$

$A \models F$

A – Model: Network of Timed Automata

F – Requirement: $\text{TCTL}$ formula, e.g.:
- $\text{Invariant}$: something bad will never happen
- $\text{Liveness}$: something good will eventually happen
- $\text{Bounded Liveness}$: something good will happen before some upper time-bound $T$. 
UPPAAL Tool

Modeling

Simulation

Verification
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

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
- \( ( \text{location}, x=v, y=u ) \) 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**: \( ( n, x=2.4, y=3.1415 ) \xrightarrow{e(1.1)} ( n, x=3.5, y=4.2415 ) \)

Action used for synchronization

\( x \leq 5 \) & \( y > 3 \)

Guard:
\[ x := 0 \]

Reset:
\[ a \]

\[ n \]

\[ m \]
Timed Automata review

Invariants

Clocks: $x, y$

Transitions

$\begin{array}{c}
(n, x=2.4, y=3.1415) \\
\xrightarrow{e(3.2)} \\
(n, x=3.5, y=4.2415)
\end{array}$

Invariants ensure progress!!
Example

Reachable?
Example

\[ \text{Reachable?} \]

\( (L_0, x=0, y=0) \)
Example

Informationsteknologi

Reachable?

Example

Reachable?

(L0, x=0, y=0)

(ε(1.4)

(L0, x=1.4, y=1.4)
Example

\[(L_0, x=0, y=0) \xrightarrow{\varepsilon(1.4)} (L_0, x=1.4, y=1.4) \xrightarrow{a} (L_0, x=1.4, y=0)\]
Example

Reachable?

\[ y := 0 \]
\[ x := 0 \]
\[ y \leq 2 \]
\[ x \leq 2 \]

\[ (L_0, x=0, y=0) \rightarrow \varepsilon(1.4) \]
\[ (L_0, x=1.4, y=1.4) \rightarrow a \]
\[ (L_0, x=1.4, y=0) \rightarrow \varepsilon(1.6) \]
\[ (L_0, x=3.0, y=1.6) \rightarrow a \]
\[ (L_0, x=3.0, y=0) \]
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

- **location**: $l$
- **action**: $a$
- **guard**: $2 \leq x \leq 3$
- **reset-set**: $\{x\}$
- **value of $x$**
  - $2$
  - $3$
  - $4$
- **Graph**
  - Horizontal axis: $2, 4, 6, 8, 10$
  - Vertical axis: $2, 3, 4$
  - Points: $a, a$
Timed Automata: Example

\[ x \leq 3 \]

\[ \{ x \} \]

\[ x \geq 2 \]

Invariant

value of \( x \)

\[ \text{time} \rightarrow \]

\[ \text{value of } x \]

\[ a \]

\[ a \]

\[ a \]
Light Control Interface
Light Control Interface

- **Press?**
- **Release?**
- **Touch!**
- **Starthold!**
- **Endhold!**

Program:
- $L = +1$ if $L < 0$
- $L = 0$ otherwise
- $x = 0$
- $x = +1$ if $x = 0$
- $x = 0$ otherwise

Interface:
- Press?
- Release?
- Touch!
- Starthold!
- Endhold!
Networks of Timed Automata  
(a’la CCS)

Two-way synchronization on complementary actions.

Closed Systems!

Example transitions

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

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

If a URGENT CHANNEL
Network Semantics

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

where

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

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

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

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

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

where

\[ \begin{align*}
S_1 \xrightarrow{\mu} & S_1' \\
S_2 \xrightarrow{\mu} & S_2'
\end{align*} \]

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

\[ S_1 \xrightarrow{! a} S_1', \quad S_2 \xrightarrow{? a} S_2' \]

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

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

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

\[ \forall d' < d, \forall u \in UAct: \]

\[ e(d') u? \rightarrow \rightarrow \quad e(d') u! \rightarrow \rightarrow \]

\[ \neg (s_1 \rightarrow \rightarrow \land s_2 \rightarrow \rightarrow) \]
Light Control Network
Overview of the UPPAAL Toolkit
UPPAAL’s architecture

GUI (Java)
uppaal2k.jar

Server

Engine (C++)
Linux, Windows, Solaris, MacOS

CLI
xml
ta
xta
GUI

Simulator

Editor

Verifier
Train Crossing

Stopable Area

[10, 20]

[7, 15]

[3, 5]

Queue

Gateway

River
Train Crossing

Communication via channels and shared variable.

Stopable Area

Queue

Gate

River

Communication via channels and shared variable.
Timed Automata in UPPAAL
Declarations

/*
 * 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, ... ;

  Ordinary channels.
  Urgent actions (see later)

- Example:
  - chan a, b;
  - urgent chan c;
Declarations UPPAAL (cont.)

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

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

\[ \text{i := Expr} \]
\[ \text{Expr ::= i} \mid \text{i[Expr]} \mid \text{n} \mid \text{Expr + Expr} \mid \text{Expr - Expr} \mid \text{Expr * Expr} \mid \text{Expr / Expr} \mid (\text{gd?Expr : Expr}) \]

invariants

\[ \text{g ::= gc|gd|g,g} \]
\[ \text{gc ::= x \otimes Expr} \mid \text{x \otimes y + Expr} \]
\[ \text{gd ::= Expr op Expr} \]
\[ \otimes \in \{<,\leq,=,\geq,>\} \]
\[ \text{op} \in \{<,\leq,=,\geq,>,!\text{=}\} \]

Guards
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**

- 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(el, 1);`
  - `Train2:=Train(el, 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
  - \texttt{meta int x;}

Forall / Exists expressions

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

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

Example:
\texttt{forall (x:\text{int}[0,4]) 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 freezed!
- 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
TCTL: Timed Computational Tree Logic
\[ TCTL = CTL + Time \]

\[
\phi ::= p \mid \alpha \mid \neg \phi \mid \phi \lor \phi \mid z \text{ in } \phi \mid E[\phi U \phi] \mid A[\phi U \phi]
\]

\( p \in AP \), \textit{atomic propositions},
\( z \in D \), \textit{formula clocks},
\( \alpha \) – constraints over formula clocks and automata clock,
\( z \text{ in } \phi \) – “freeze operator” introduces new formula clock \( z \)
\( E[\phi U \phi], A[\phi U \phi] \) - like in CTL
\( \text{No } EX \phi \)
Derived Operators

\[ A[\phi U_{\leq 7} \psi] = z \text{ in } A[(\phi \land z \leq 7) U \psi]. \]

Along any path \( \phi \) holds continuously until within 7 time units \( \psi \) becomes valid.

\[ EF_{<5} \phi = z \text{ in } EF(z < 5 \land \phi) \]

The property \( \phi \) becomes valid within 5 time units.
A path is an infinite sequence $s_0 a_0 s_1 a_1 s_2 a_2 \ldots$ of states alternated by transition labels such that $s_i \xrightarrow{a_i} s_{i+1}$ for all $i \geq 0$.

Example:

$(off, x = y = 0) \xrightarrow{3.5} (off, x = y = 3.5) \xrightarrow{push}$

$(on, x = y = 0) \xrightarrow{\pi} (on, x = y = \pi) \xrightarrow{push}$

$(on, x = 0, y = \pi) \xrightarrow{3} (on, x = 3, y = \pi + 3) \xrightarrow{9-(\pi+3)}$

$(on, x = 9-(\pi+3), y = 9) \xrightarrow{click} (off, x = 0, y = 9) \ldots$
Elapsed time in path

\[ \Delta(\sigma,0) = 0 \]
\[ \Delta(\sigma,i+1) = \Delta(\sigma,i) + \begin{cases} 0 & \text{if } a_i = * \\ a_i & \text{if } a_i \in \mathbb{R}^+ \end{cases} \]

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

\[ \Delta(\sigma,1)=3.5, \Delta(\sigma,6)=3.5+9=12.5 \]
TCTL Semantics

\[
\begin{align*}
  s, w &\models p \quad \text{iff } p \in \text{Label}(s) \\
  s, w &\models \alpha \quad \text{iff } \nu \cup w \models \alpha \\
  s, w &\models \neg \phi \quad \text{iff } \neg (s, w \models \phi) \\
  s, w &\models \phi \lor \psi \quad \text{iff } (s, w \models \phi) \lor (s, w \models \psi) \\
  s, w &\models z \text{ in } \phi \quad \text{iff } s, \text{reset } z \text{ in } w \models \phi \\
  s, w &\models E[\phi U \psi] \quad \text{iff } \exists \sigma \in P_M^\infty(s). \exists (i, d) \in \text{Pos}(\sigma). \\
  & \quad \quad \quad \quad \quad (\sigma(i, d), w+\Delta(\sigma, i) \models \psi \quad \land \\
  & \quad \quad \quad \quad \quad (\forall (j, d') \ll (i, d). \sigma(j, d'), w+\Delta(\sigma, j) \models \phi \lor \psi)) \\
  s, w &\models A[\phi U \psi] \quad \text{iff } \forall \sigma \in P_M^\infty(s). \exists (i, d) \in \text{Pos}(\sigma). \\
  & \quad \quad \quad \quad \quad (((\sigma(i, d), w+\Delta(\sigma, i)) \models \psi \quad \land \\
  & \quad \quad \quad \quad \quad (\forall (j, d') \ll (i, d). (\sigma(j, d'), w+\Delta(\sigma, j)) \models \phi \lor \psi)).
\end{align*}
\]

\[(i, d) \ll (i', d') \text{ iff } (i<j) \text{ or } ((i=j) \text{ and } (d<d'))\]
Timeliness Properties

\[ AG[send(m) \Rightarrow AF_{\leq 5} receive(r_m)] \]

receive\((m)\) occurs within 5 time units after \(send\(m)\)

\[ EG[send(m) \Rightarrow AF_{=11} receive(r_m)] \]

receive\((m)\) occurs exactly 11 time units after \(send\(m)\)

\[ AG[AF_{=25} putbox] \]

\(putbox\) occurs periodically (exactly) every 25 time units
(note: other \(putbox\)’s may occur in between)
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: \( P \rightarrow_{\leq t} Q \)

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: $P \to Q$

- **Bounded Liveness**
  - Leads to within: $P \to \leq t Q$
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

\[
\begin{align*}
\text{GearControl@Initiate} & \sim_{\leq 1500} ( \text{ErrStat} = 0 ) \Rightarrow \text{GearControl@GearChanged} & (1) \\
\text{GearControl@Initiate} & \sim_{\leq 1000} ( \text{ErrStat} = 0 \land \text{UseCase} = 0 ) \Rightarrow \text{GearControl@GearChanged} & (2) \\
\text{Clutch@ErrorClose} & \sim_{\leq 200} \text{GearControl@CCloseError} & (3) \\
\text{Clutch@ErrorOpen} & \sim_{\leq 200} \text{GearControl@COpenError} & (4) \\
\text{GearBox@ErrorIdle} & \sim_{\leq 350} \text{GearControl@GSetError} & (5) \\
\text{GearBox@ErrorNeu} & \sim_{\leq 200} \text{GearControl@GNeuError} & (6) \\
\text{Inv} ( \text{GearControl@CCloseError} \Rightarrow \text{Clutch@ErrorClose} ) & & (7) \\
\text{Inv} ( \text{GearControl@COpenError} \Rightarrow \text{Clutch@ErrorOpen} ) & & (8) \\
\text{Inv} ( \text{GearControl@GSetError} \Rightarrow \text{GearBox@ErrorIdle} ) & & (9) \\
\text{Inv} ( \text{GearControl@GNeuError} \Rightarrow \text{GearBox@ErrorNeu} ) & & (10) \\
\text{Inv} ( \text{Engine@ErrorSpeed} \Rightarrow \text{ErrStat} \neq 0 ) & & (11) \\
\text{Inv} ( \text{Engine@Torque} \Rightarrow \text{Clutch@Closed} ) & & (12) \\
\bigwedge_{i \in \{ i, N, 1, \ldots, 5 \}} \text{Poss} ( \text{Gear@Gear}_i ) & & (13) \\
\bigwedge_{i \in \{ i, 1, \ldots, 5 \}} \text{Inv} ( ( \text{GearControl@Gear} \land \text{Gear@Gear}_i ) \Rightarrow \text{Engine@Torque} ) & & (14)
\end{align*}
\]
**UPPAAL 3.4**

**Gate Template**

**IntQueue**

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

Gate Template

```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];
}
```

Gate Declaration
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.

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 web site 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.

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.