Modeling real-time systems
---
UPPAAL modeling language

Timed Automata in UPPAAL

Clock Assignments
\[ x := n \]

Variable Assignments
\[ \begin{align*}
& x \geq 5, \ y > 3 \\
& x := 0 \\
& x \leq 5 \\
& y \leq 10 \\
& g_1, g_2, g_3, g_4, \ldots
\end{align*} \]

Location Invariants
\[ \begin{align*}
& \text{inv} := x < n \land x < n \land \text{inv}, \text{inv} \\
& \text{clock}, \text{natural number } \land \text{and}
\end{align*} \]

Clock guards
\[ g := x \oplus n \]
\[ g := x \otimes n \]
\[ g := \text{Exp} \circ \text{Exp} \]
\[ g := \text{Exp} \circ \text{Exp} \]
\[ g := \text{Exp} \]
\[ (g \Rightarrow \text{Exp} : \text{Exp}) \]

Data guards
\[ \begin{align*}
& \text{op} \\
& \text{ExpropExpr} \\
& \text{nyxnx} \\
& \text{g} \\
& \text{dc}
\end{align*} \]

Declaration in UPPAAL

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

- Clocks:
  - Syntax:
    \[ \text{clock } x_1, \ldots, x_n ; \]
  - Example:
    \[ \text{clock } x, y ; \]
    Declares two clocks: \( x \) and \( y \).

- Data variables
  - Syntax:
    \[ \begin{align*}
    & \text{int } n_1, \ldots ; \\
    & \text{int}[1, u] n_1, \ldots ; \\
    & \text{int}[0, m] a, b[5] ;
    \end{align*} \]

Networks of Timed Automata

Declarations in UPPAAL (cont.)
Declarations in UPPAAL (cont.)

- Actions (or channels):
  - Syntax:
    ```
    chan a, ... ;
    urgent chan b, ... ;
    ```
  - Ordinary channels.
  - Urgent actions (described later)
  - Example:
    - chan a, b[2];
    - urgent chan c;

Declarations UPPAAL (const.)

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

Templates in UPPAAL

- 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(e2, 2);

Urgent Channels: Example 1

- Suppose the two edges in automata P and Q should be taken as soon as possible.
- I.e. as soon as both automata are ready (simultaneously in locations l1 and s1).
- How to model with invariants if either one may reach l1 or s1 first?

Solution: declare action "a" as urgent.
Urgent Channels

```plaintext
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 Channel: Example 2

- Assume i is a data variable.
- We want P to take the transition from l1 to l2 as soon as i==5.

```plaintext
• Solution: P can be forced to take transition if we add another automaton:

where "go" is an urgent channel, and we add "go?" to transition l1→l2 in automaton P.
```

Broadcast Synchronisation

```plaintext
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.

Urgent Location

Click "Urgent" in State Editor.

Informal Semantics:
- No delay in urgent location.

Note: the use of urgent locations reduces the number of clocks in a model, and thus the complexity of the analysis.

Urgent Location: Example

- Assume that we model a simple media M:

```plaintext
that receives packages on channel a and immediately sends them on channel b.
- P models the media using clock x.
```
Urgent Location: Example

- Assume that we model a simple media $M$: 
  
  ![Diagram of media M](Diagram.png)

  that receives packages on channel a and immediately sends them on channel b.
- P models the media using clock $x$.
- Q models the media using urgent location.
- P and Q have the same behavior.

Committed Location

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 interleaving in state space exploration (and also the number of clocks in a model), and thus allows for more space and time efficient analysis.

Committed Location: Example 1

- Assume: we want to model a process (P) simultaneously sending message a and b to two receiving processes (when $i=0$).
- P' sends "a" two times at the same time instant, but in location "n" other automata, e.g. Q may interfere:

  ![Diagram of committed location example 1](Diagram.png)

  Solution: mark location n "committed" in automata P' (instead of "urgent").

Committed Location: Example 2

- Assume: we want to pass the value of integer "k" from automaton P to variable "j" in Q.
  The value of k can be passed using a global integer variable "t".
- Location "n" is committed to ensure that no other automaton can assign "t" before the assignment "j:=t".