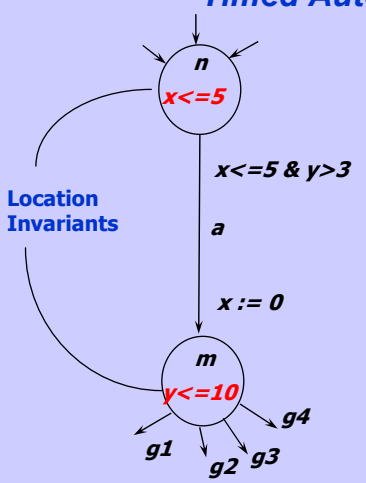
Timed Automata in UPPAAL

(Henzinger et al, 1992)

Timed Safety Automata

Timed Automata + Invariants



Clocks: *x, y*

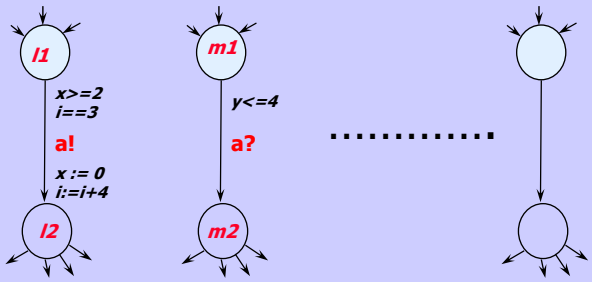
Transitions:

(*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)

Location invariants are used to force an automata to progress (i.e. leave the location) before the invariant becomes false.

Networks of Timed Automata



Two-way synchronization on **complementary** actions.

Closed Systems!

Timed Automata in UPPAAL

- Timed Automata with Invariants
  + urgent action channels,
  + broadcast channels,
  + urgent and committed locations,
  + data-variables (with bounded domains),
  + arrays of data-variables,
  + constants,
  + guards and assignments over data-variables and arrays…,
  + templates with local clocks, data-variables, and constants

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 from "l" to "u".
Integer array w. elements n1[0] to n1[m-1].

  - **Example;**
  - `int a, b;`
  - `int[0,1] a, b[5];`

Declarations in UPPAAL (cont.)

- Actions (or channels):
  - **Syntax:**

```
chan a, … ;
urgent chan b, … ;
```

Ordinary channels.
Urgent actions (described later)

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

Declarations UPPAAL (const.)

- Constants
  - **Syntax:**

```
const c1 n1, …, ck nk;
```

  - **Example:**
  - `const true 1, false 0;`

Timed Automata in UPPAAL

Clock Assignments

$x := n$

Variable Assignments

$i := Expr$

$Expr ::= i \mid i[Expr] \mid$

$n \mid -Expr \mid$

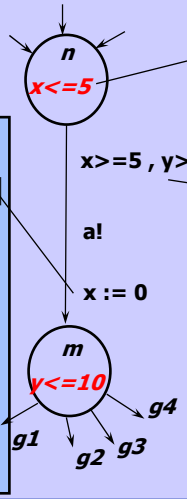
$Expr + Expr \mid$

$Expr - Expr \mid$

$Expr * Expr \mid$

$Expr / Expr \mid$

$(g_d \,?\, Expr : Expr)$



Location Invariants

$inv ::= x < n \mid x <= n \mid inv, inv$

clock natural number "and"

$g ::= g_c \mid g_d \mid g, g$

$g_c ::= x \otimes n \mid x \otimes y + n$

$g_d ::= Expr \;\; op \;\; Expr$

$\otimes \in \{<, <=, ==, >=, >\}$

$op \in \{<, <=, ==, >=, >, !=\}$

Clock guards

Data guards

Timed Automata in UPPAAL

Clock Assignments

$x := n$

Variable Assignments

$i := Expr$

$Expr ::= i \mid i[Expr] \mid$

$n \mid -Expr \mid$

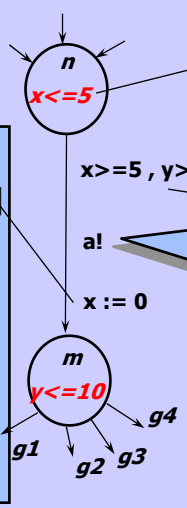
$Expr + Expr \mid$

$Expr - Expr \mid$

$Expr * Expr \mid$

$Expr / Expr \mid$

$(g_d \,?\, Expr : Expr)$



Location Invariants

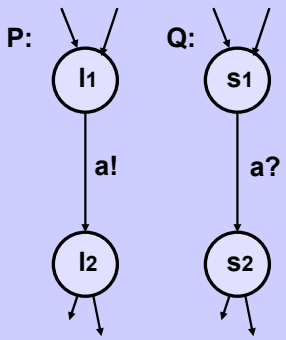
$inv ::= x < n \mid x <= n \mid inv, inv$

clock natural number "and"

Actions:

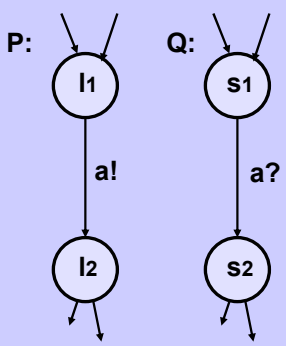
- "a" name of action
- a! or a?
- one or zero per edge

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 l_1 and s_1).
- How to model with invariants if either one may reach l_1 or s_1 first?

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 l_1 and s_1).
- How to model with invariants if either one may reach l_1 or s_1 first?
- **Solution**: declare action “a” as urgent.

Urgent Channels

```
urgent chan hurry;
```

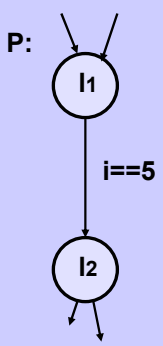
Informal Semantics:

- There will be <u>no delay</u> if transition with urgent action can be taken.

Restrictions:

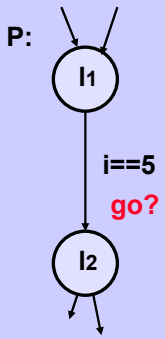
- <u>No clock guard</u> allowed on transitions with urgent actions.
- <u>Invariants</u> and <u>data-variable guards</u> are allowed.

Urgent Channel: Example 2

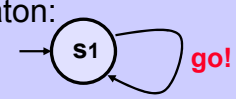


- Assume i is a data variable.
- We want P to take the transition from l1 to l2 as soon as i==5.

Urgent Channel: Example 2



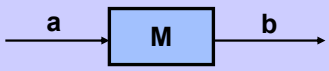
- Assume i is a data variable.
- We want P to take the transition from l1 to l2 as soon as i==5.
- **Solution**: P can be forced to take transition if we add another automaton:



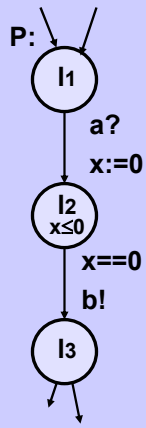
where "go" is an urgent channel, and we add "go?" to transition l1→l2 in automaton P.

Urgent Location: Example

- Assume that we model a simple media M:

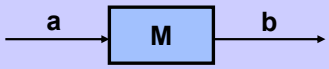


that receives packages on channel a and immediately sends them on channel b.
- P models the media using clock x.

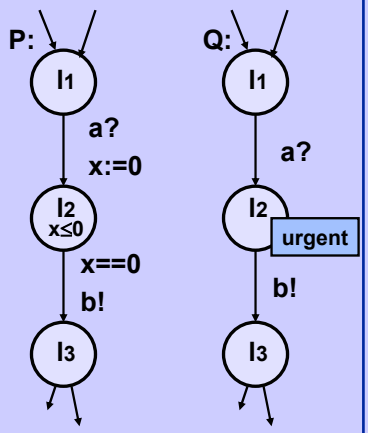


Urgent Location: Example

- Assume that we model a simple media M:



  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.



Urgent Location

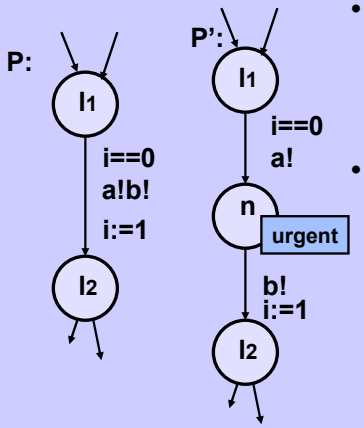
Click "Urgent" in State Editor.

Informal Semantics:

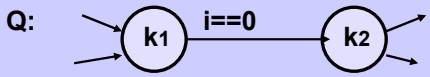
- <u>No delay</u> in urgent location.

Note: the use of urgent locations **<u>reduces</u>** the number of clocks in a model, and thus the complexity of the analysis.

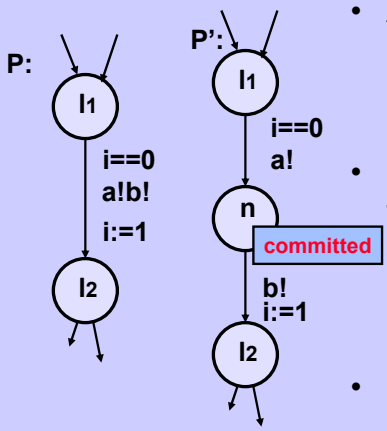
Committed Location: Example 1



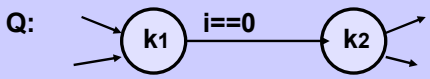
- **Assume:** we want to model a process (P) simultaneously sending message a and b to two receiving processes (when i==0).
- P' sends "a" two times at the same time instant, but in location "n" other automata, e.g. Q may interfear



Committed Location: Example 1



- **Assume:** we want to model a process (P) simultaneously sending message (a) 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 interfear:



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

Committed Location

Click "Committed" i State Editor.

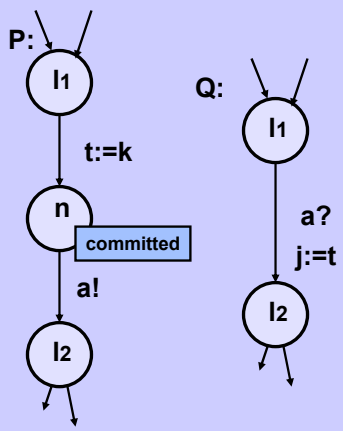
Informal Semantics:

- No delay in committed location.
- Next transition must involve automata in committed location.

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

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 is passed using a global integer variable "t".
- Location "n" is committed to ensure that no other automat can assign "t" before the assignment "j:=t".



More Operators

- New operators (not clocks):
  - Logical:
    - && (logical and), || (logical or), ! (logical negation),
  - Bitwise:
    - ^ (xor), & (bitwise and), | (bitwise or),
  - Bit shift:
    - << (left), >> (right)
  - Numerical:
    - % (modulo), <? (min), >? (max)
  - Compound Assignments:
    - +=, -=, *=, /=, ^=, <<=, >>=
  - Prefix or Postfix:
    - ++ (increment), -- (decrement)

More Declarations

- Multi dimensional arrays

  e.g. int b[2][3];
- Array initialiser:

  e.g. int b[2][3] := { {1,2,3}, {4,5,6} };
- Arrays of channels, clocks, constants.

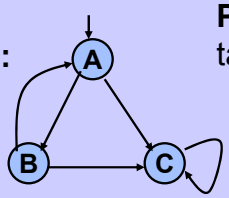
  e.g.
  - chan a[3];
  - clock c[3];
  - const k[3] { 1, 2, 3 };
- Broadcast channels.

  e.g. broadcast chan a;

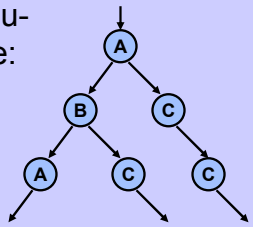
UPPAAL's Specification Language

- A subset of the logic TCTL (Timed Computation Tree Logic).
- Formula refers to the "computation tree" of an automaton.
- **Example:**

P:



P's computation tree:



Quantifiers in TCTL

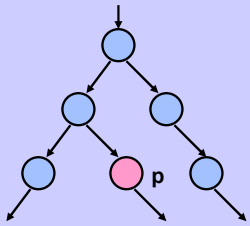
- E - exists a path ("**E**" in UPPAAL).
- A - for all paths ("**A**" in UPPAAL).
- G - all states in a path ("**[]**" in UPPAAL).
- F - some state in a path ("**<>**" in UPPAAL).

In UPPAAL the following combination are allowed:

- **A[], A<>, E<>,** och **E[].**

E<>p – “p Reachable”

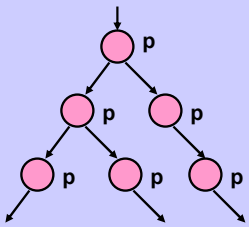
- **`E<>`** **p** – it is possible to reach a state in which p is satisfied.



- p is true in (at least) one reachable state.

A[]p – “Invariantly p”

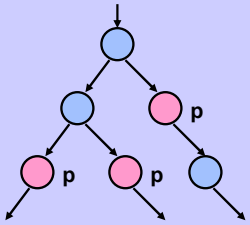
- **`A[]`** **p** – p holds invariantly.



- p is true in all reachable states.

A<>p – “Inevitable p”

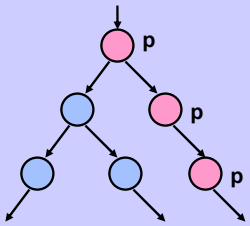
- **`A<>` p** – p will inevitable become true, the automaton is guaranteed to eventually reach a state in which p is true.



- p is true in some state of all paths.

E[ ] p – “Potentially Always p”

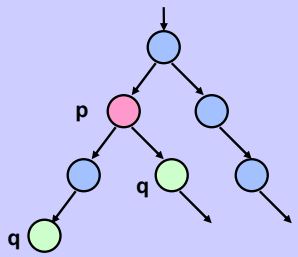
- **`E[]` p** – p is potentially always true.



- There exists a path in which p is true in all states.

p --> q– “p lead to q”

- **p** --> **q** – if p becomes true, q will inevitably become true.
  same as A[](**p** imply A<> **q**)



- In all paths, if p becomes true, q will inevitably become true.

UPPAAL – Local Properties

- **`A[]p, A<>p, E<>p, E[]p, p-->p`**
  where **`p`** is a local property

- **Syntax:**