An introduction to UPPAAL

Jeremy Sproston

sproston@di.unito.it

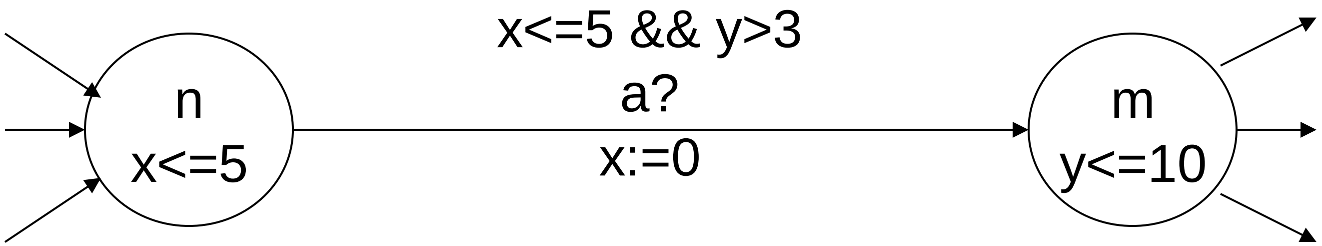
(lezione tenuta dalla prof.ssa Donatelli

UPPAAL

- Developed by the universities of Uppsala (Sweden) and Aalborg (Denmark)
  - www.uppaal.com
- Used to model check:
  - Systems expressed as networks of interacting timed automata (with discrete variables)
  - A restricted class of CTL properties (limited nesting)

Timed automata

- Recall: timed automata
  - Finite state graph equipped with a finite set of variables called clocks, which increase at the same rate as real-time



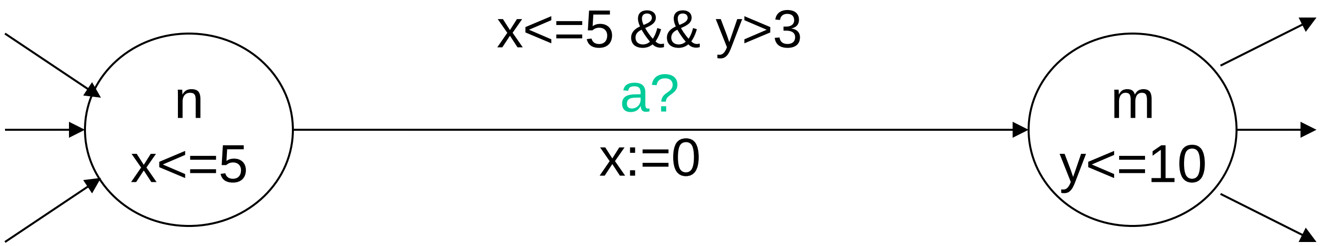
  - Semantics: timed transition systems
    - E.g. of (timed) transition:

      (n, x=2.4, y=3.1415) $\rightarrow$ (n, x=3.5, y=4.2415)
    - E.g. of (discrete) transition:

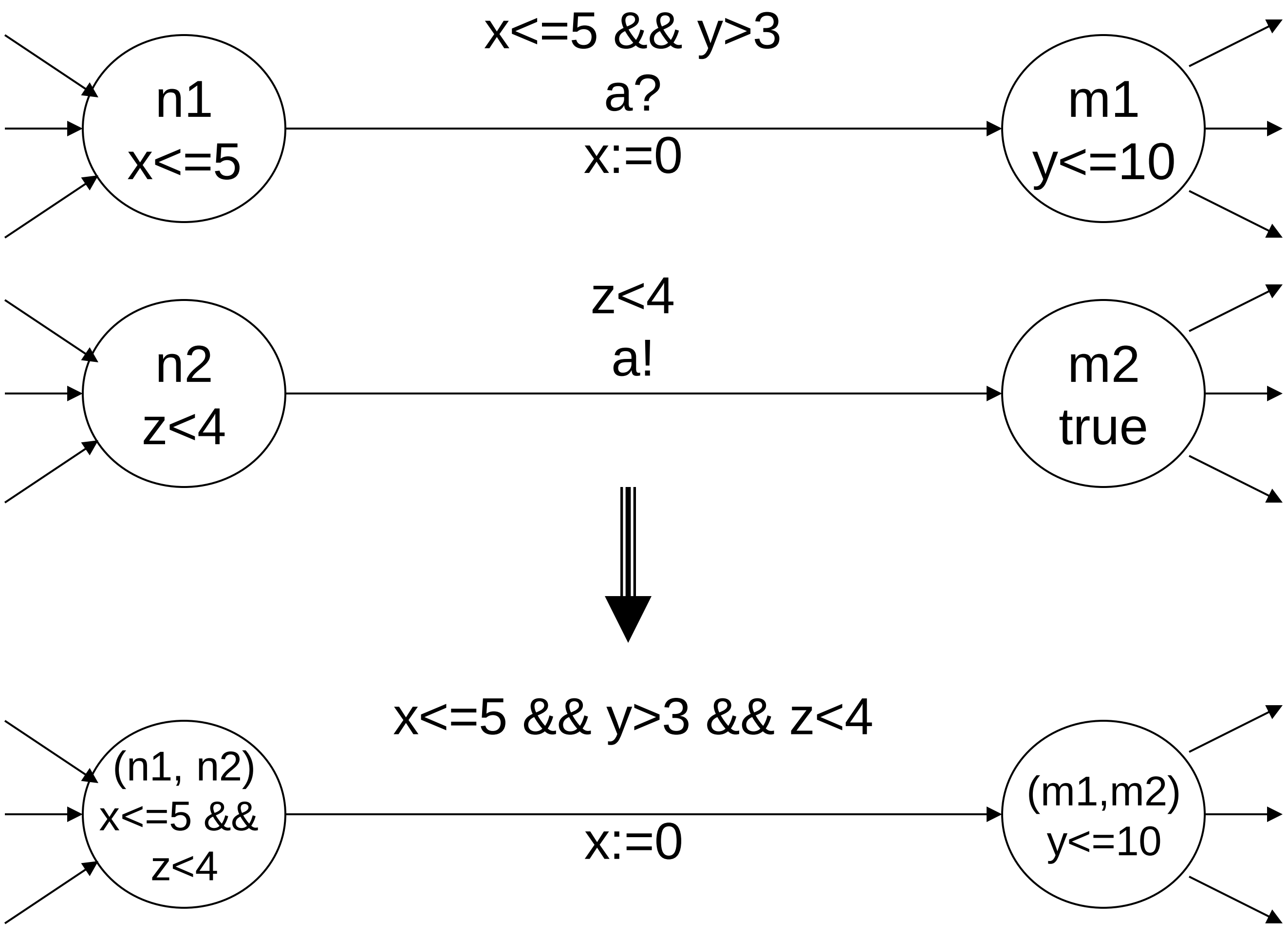
      (n, x=2.4, y=3.1415) $\rightarrow$ (m, x=0, y=3.1415)

Networks of timed automata

- Model complex systems using a set of interacting timed automata
- Edges of timed automata can be labelled with *actions*
  - Can be used to define synchronization, as in process algebra
  - UPPAAL models feature two-way synchronization on *complementary* actions
  - No action label: internal action



Networks of timed automata

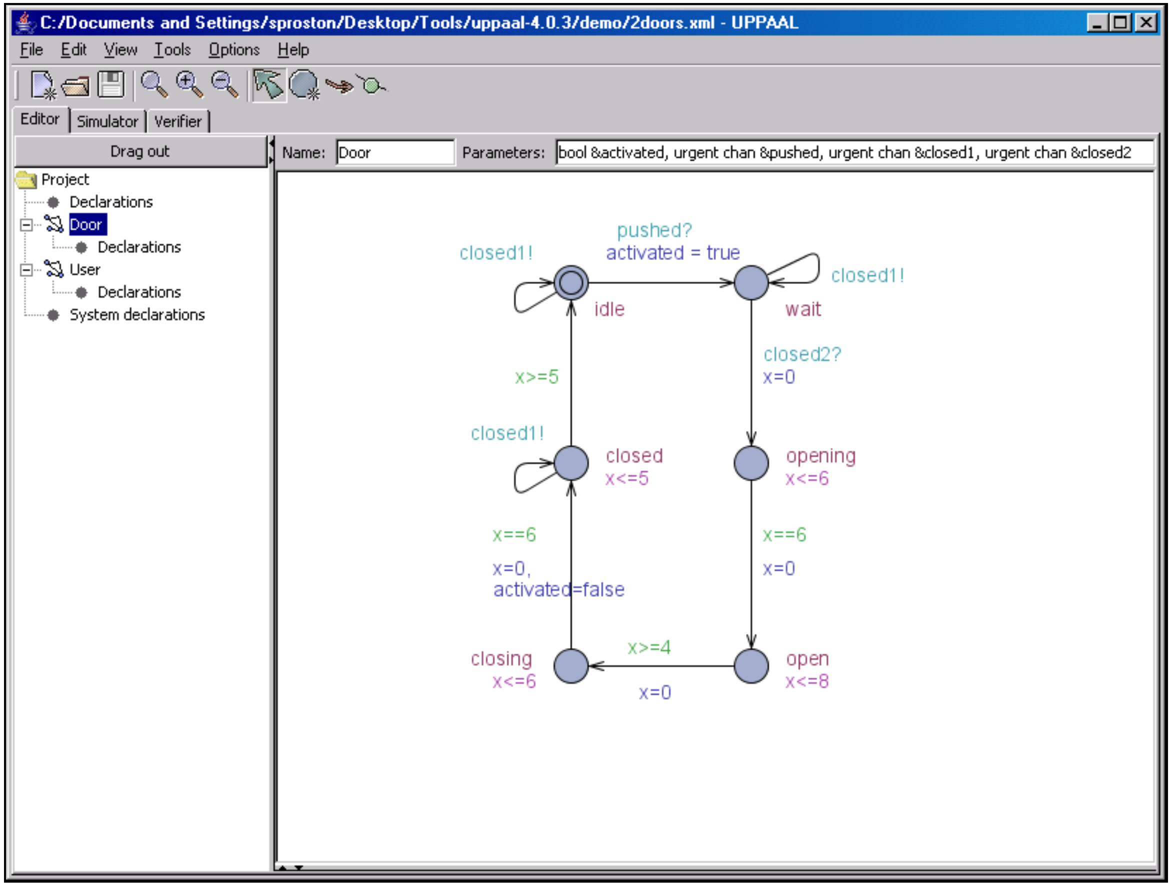


Modelling in UPPAAL

- Other key concepts in the UPPAAL modelling language:
  - Urgency (of locations, and of synchronization channels)
  - Committed locations
  - Discrete variables (with bounded domains)
  - Constants
- There are additional concepts (more recently introduced)

Modelling in UPPAAL

- System editor (to create and edit system models):



Modelling in UPPAAL

- Declaring clocks:
  - Syntax:

```
clock x1, …, x_n;
```

  - Example: (to declare clocks x and y)

```
clock x, y;
```

Modelling in UPPAAL

- Declaring discrete variables:
  - Syntax:

```
int[l,u] p1, …, p_n;
```

  - Example: (to declare two integer variables which takes values between 0 and 255 inclusive)

```
int[0,255] p, q;
```

  - Example - “default” domain: (to declare an integer variable which takes values from the “default” domain [-32768, 32767])

```
int p;
```

  - Example - initialisation: (to declare an integer variable which takes values between 1 and 100 inclusive, and which is initialised to 20)

```
int[1,100] p=20;
```

Modelling in UPPAAL

- Declaring channels (i.e. actions):
  - Syntax:

```
chan a1, …, a_n;
```

  - Example: (to declare two channels)

```
chan a, b;
```

- Declaring urgent channels: (to be explained later…):
  - Syntax:

```
urgent chan a1, …, a_n;
```

Modelling in UPPAAL

- Declaring boolean variables:
  - Syntax:

```
bool b1, …, bn;
```

  - Example:

```
bool switch=false;
```

- Declaring constants:
  - Syntax:

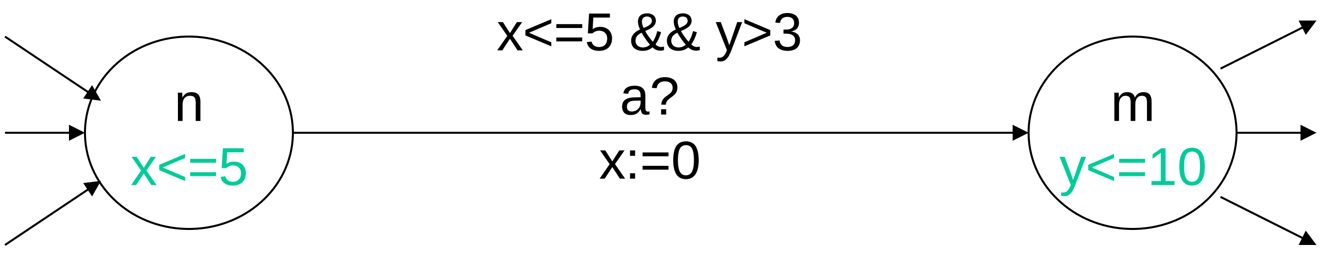
```
const int c=n;
const bool c=n;
```

  - Example:

```
const int N=1024;
```

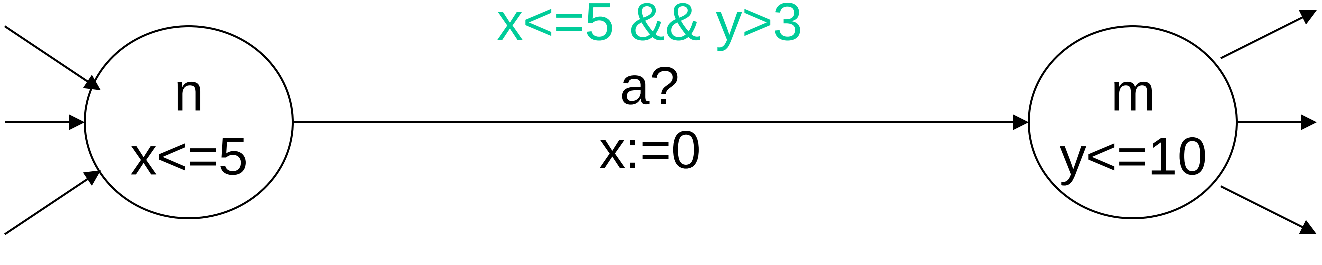
Modelling in UPPAAL

- Invariant conditions:
  - Conjunction of upper bounds on the values of clocks (the bound can be given by an expression over integers, including integer variables)
  - Example:
    - x<40 && y<=time_out*3 (where x, y are clocks, and time_out is an integer variable or integer constant)



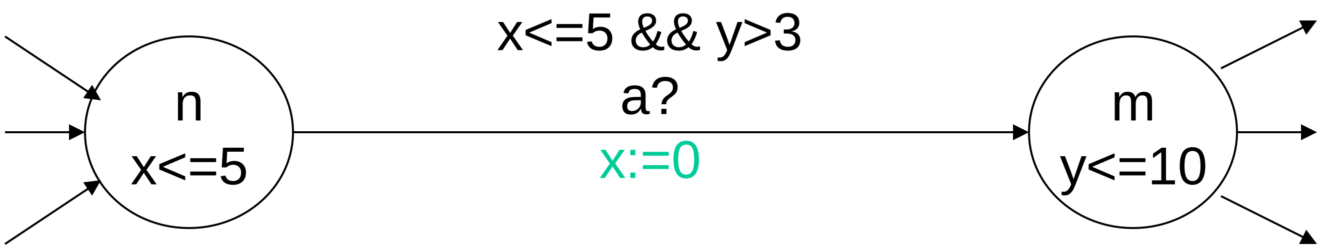
Modelling in UPPAAL

- Guards (on edges):
  - Clock guards: comparisons of values of clocks with bounds (bounds can be given as integer expressions)
  - Data guards: comparisons of values obtained by resolving integer expressions
  - For example:
    - x>backoff && backoff=bc_max (where x is a clock, backoff is an integer variable, and bc_max is an integer constant)



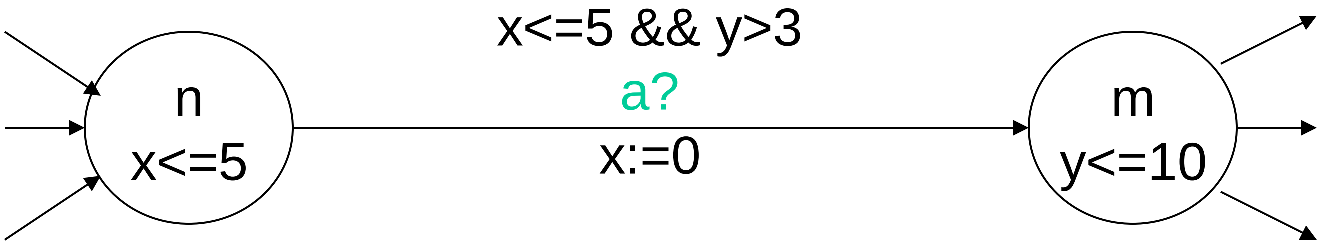
Modelling in UPPAAL

- Updates (to clocks and variables):
  - Assignment of a new value to a clock or variable (the new value may be the result of an integer expression)
  - For example:
    - x:=0 (where x is a clock)
    - x:=backoff*3 (where x is a clock and backoff is an integer variable)
    - backoff:=5 (where backoff is an integer variable)



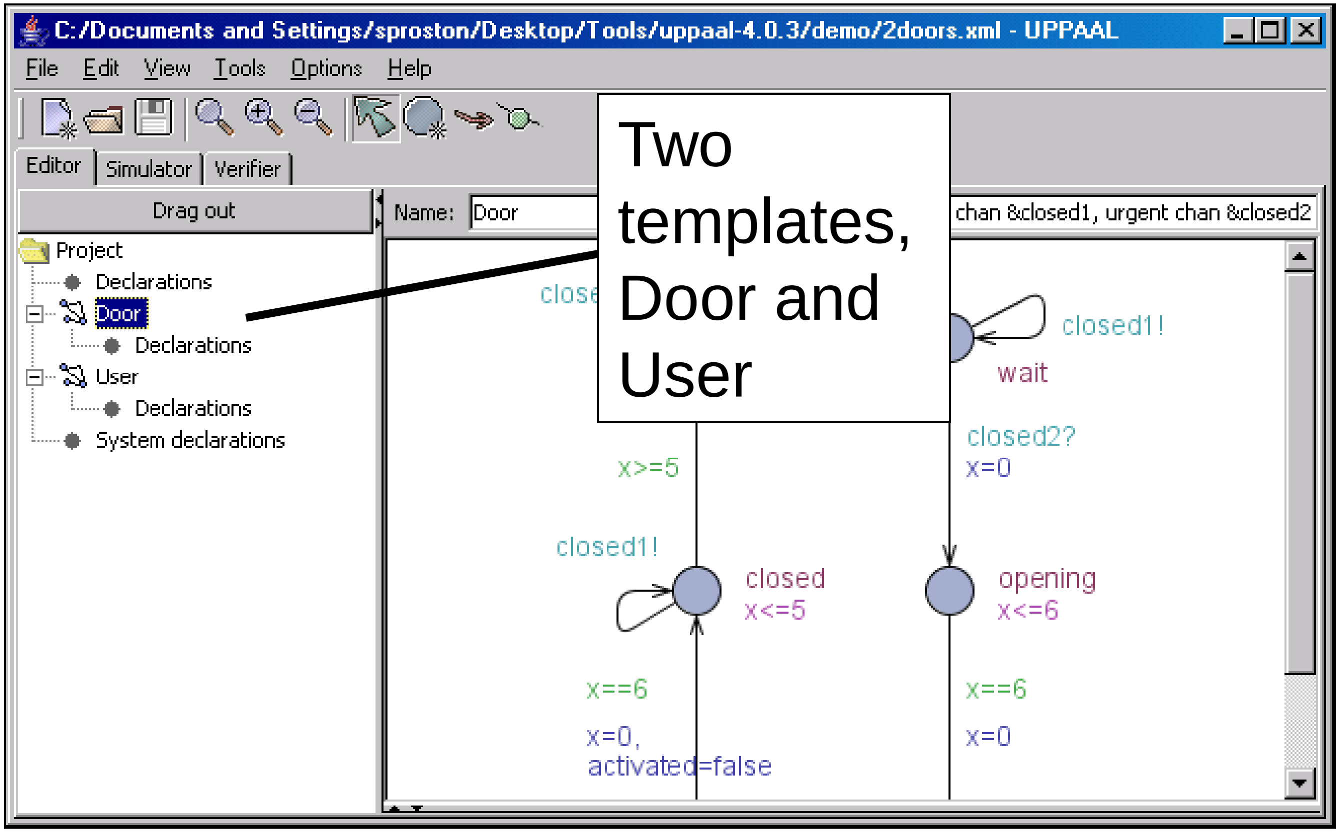
Modelling in UPPAAL

- Actions:
  - Can be of the form a!, a?, where a is the name of a channel
  - … or the edge can be unlabelled (corresponding to choice of the edge unrestricted by other automata of the system, i.e., internal action)



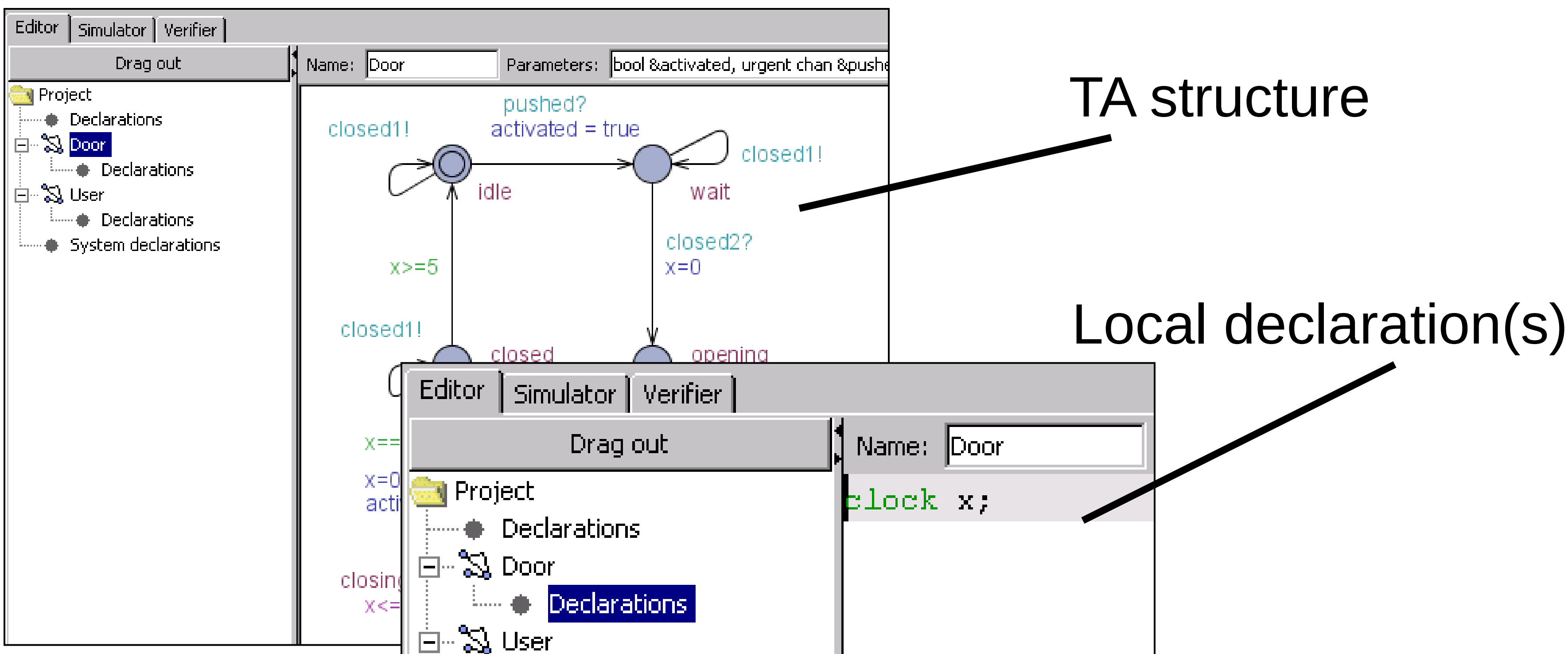
Modelling in UPPAAL

- Timed automata are modelled using *templates*
  - The list of templates are given in the left-hand bar:



Modelling in UPPAAL

- Template: the structure of a timed automaton (represented graphically), plus a set of local declarations



Modelling in UPPAAL

- Each template has a name and a set of parameters:

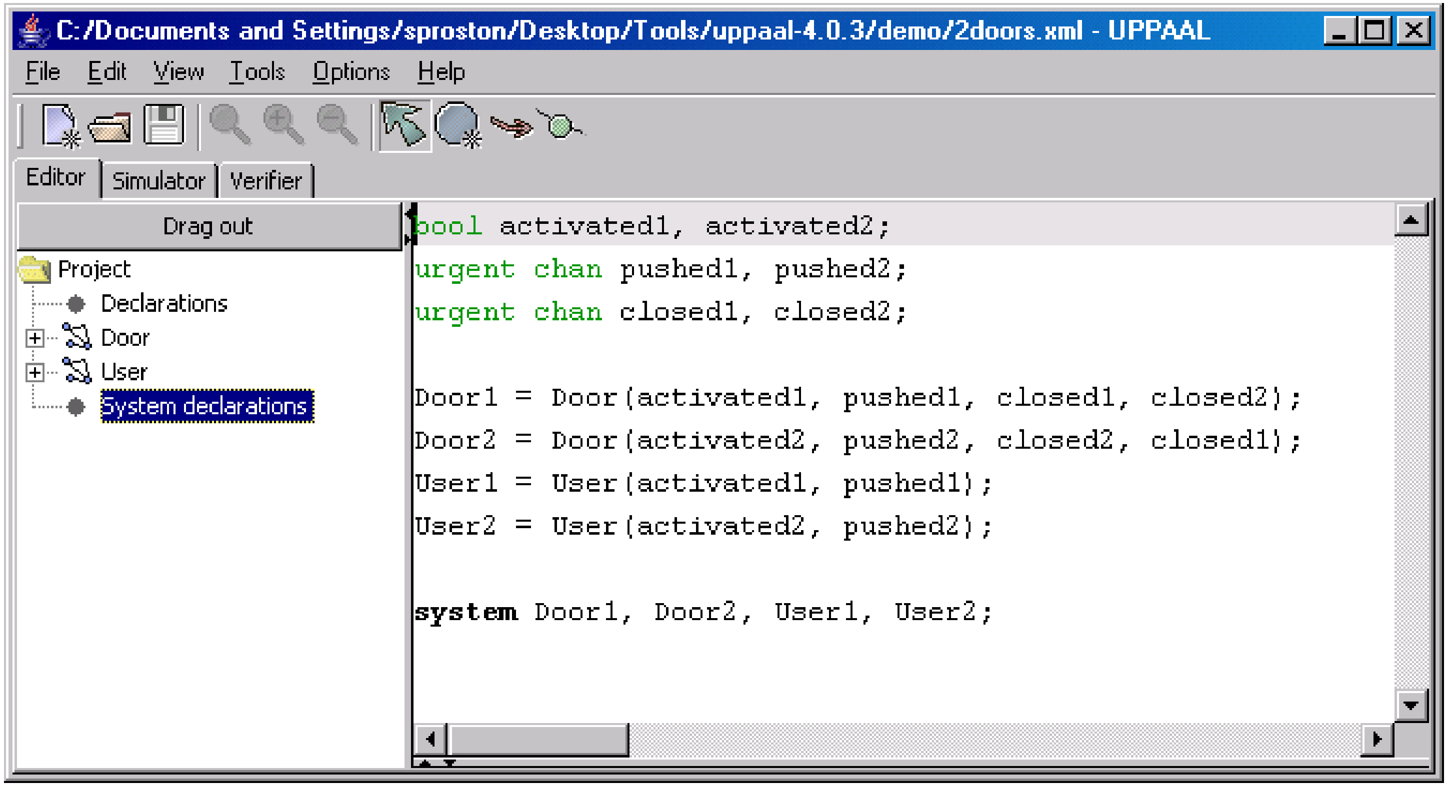
Name: Door   Parameters: bool &activated, urgent chan &pushed, urgent chan &closed1, urgent chan &closed2

- Each template can be instantiated a number of times to obtain a number of timed automata sub-components:

```
Door1 = Door(activated1, pushed1, closed1, closed2);
Door2 = Door(activated2, pushed2, closed2, closed1);
```

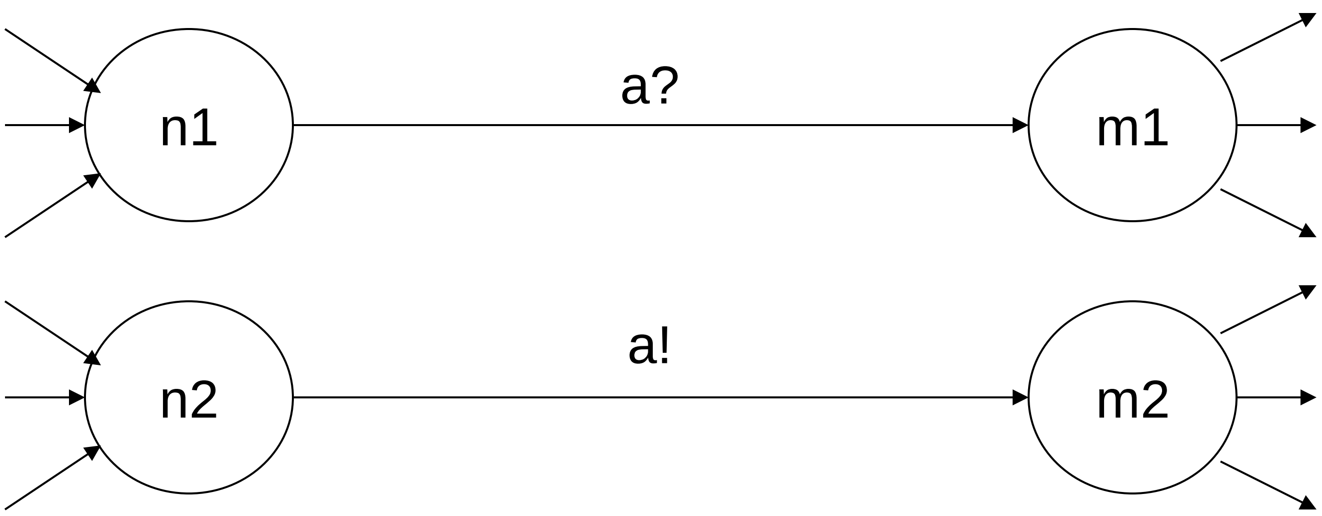
Modelling in UPPAAL

- System: corresponds to a series of instantiated templates (plus global clocks, channels, data variables, constants, which may be used in the instantiated templates)



Modelling in UPPAAL

- Urgent channels
  - Suppose that in the two timed automata, the edges from n1 to m1, and n2 to m2, should be taken as soon as possible
    - That is, when both timed automata are able to synchronise on channel a
  - Solution: declare a as an urgent channel

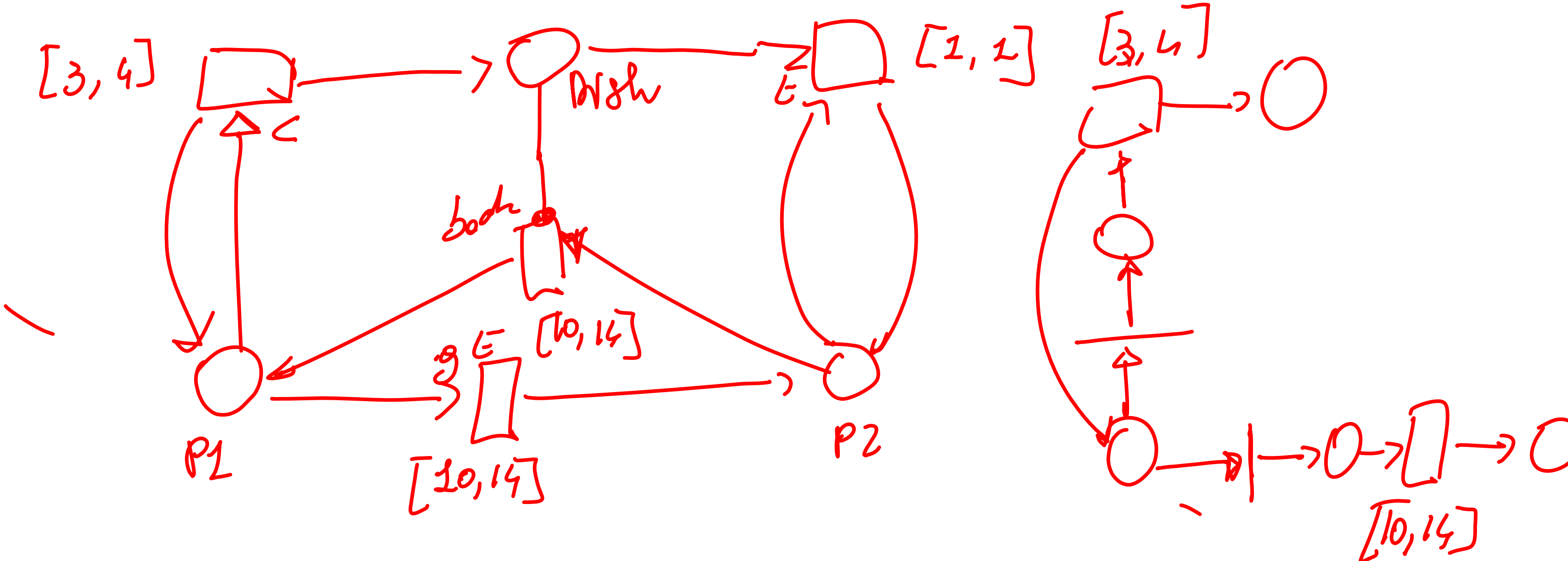


Modelling in UPPAAL

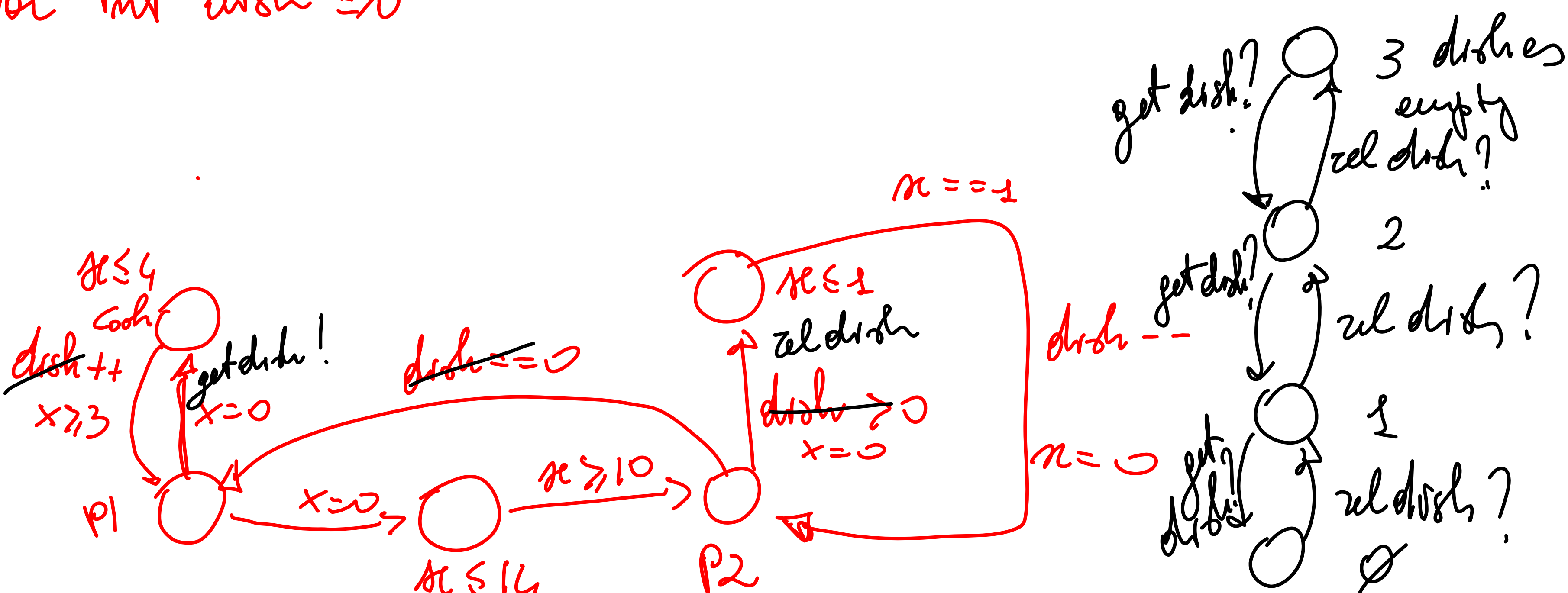
- Urgent channels
  - Recall syntax:

```
urgent chan a1, …, a_n;
```

  - Informal semantics: *no time delay is possible when an urgent action can be taken*
  - Restrictions: it is not permitted to have clock guards on transitions with urgent channels (however, invariants and data variable guards are permitted)



var int dish = 0

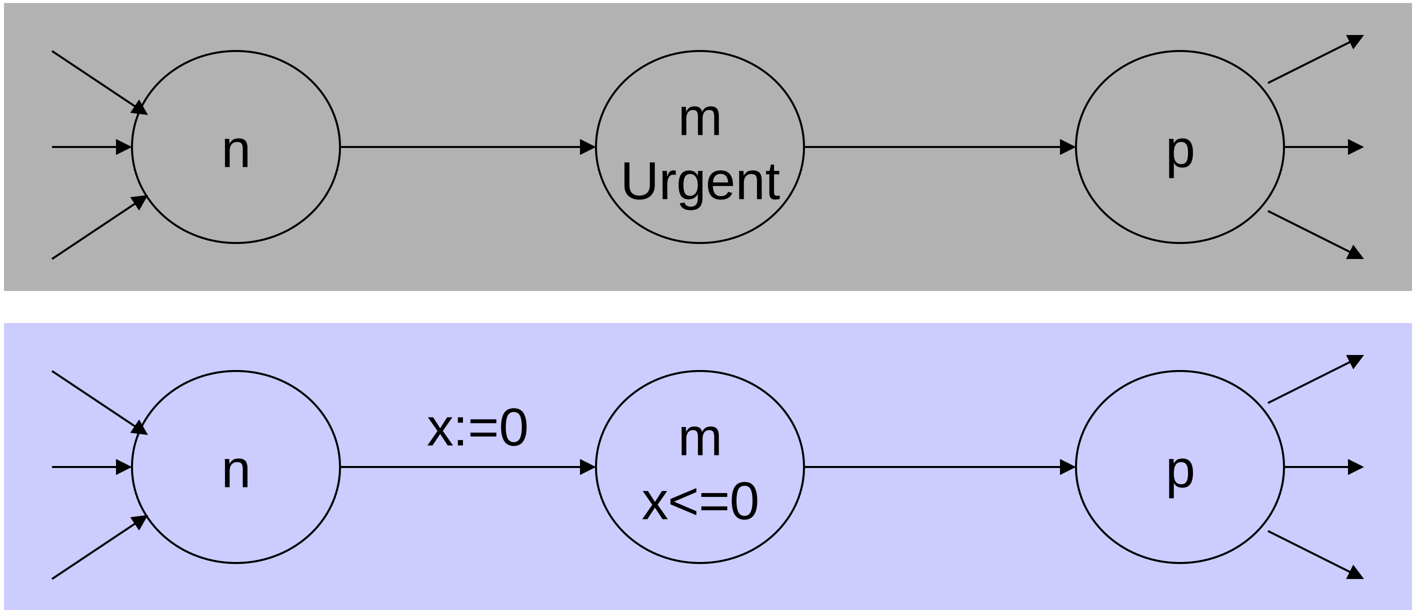


Modelling in UPPAAL

- Urgent locations
  - Informal semantics: *no time delay is possible when some timed automaton component of the system is in an urgent location*
  - Note that this places no restriction on the (enabled) discrete transitions that can be taken when an urgent location is entered
    - E.g. TA1 enters an urgent location, then the next transition of the system can be one of TA2’s enabled discrete transitions

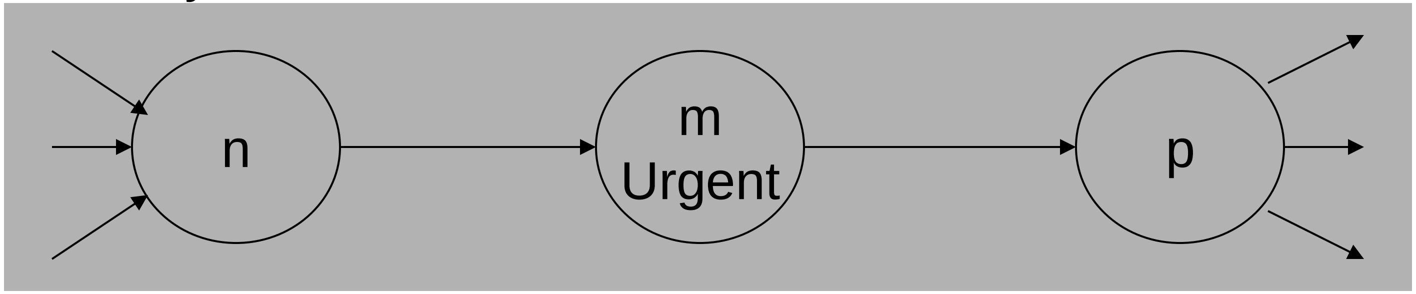
Modelling in UPPAAL

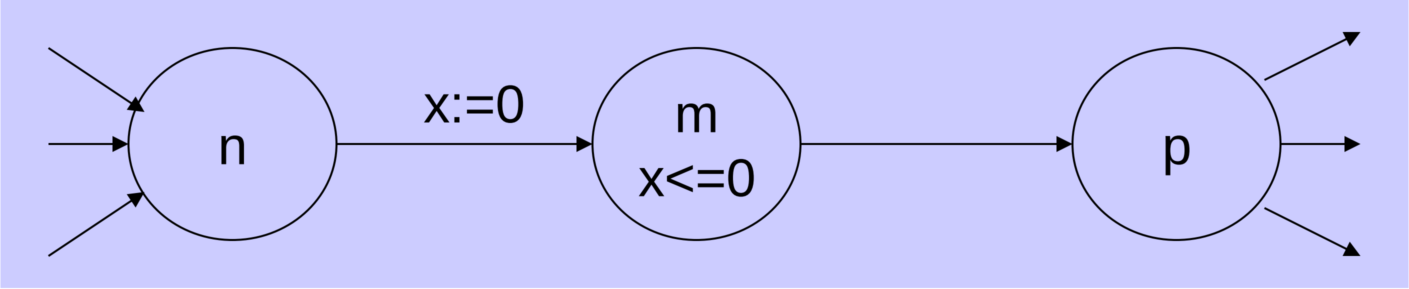
- Urgent locations
  - What is the difference between the following two situations (from the point of view of the semantics)?



Modelling in UPPAAL

- Urgent locations
  - No difference for the semantics: it’s just that we require the “extra” clock x to “simulate” urgency of location m
  - Having the extra clock is (generally) bad for modelling and analysis



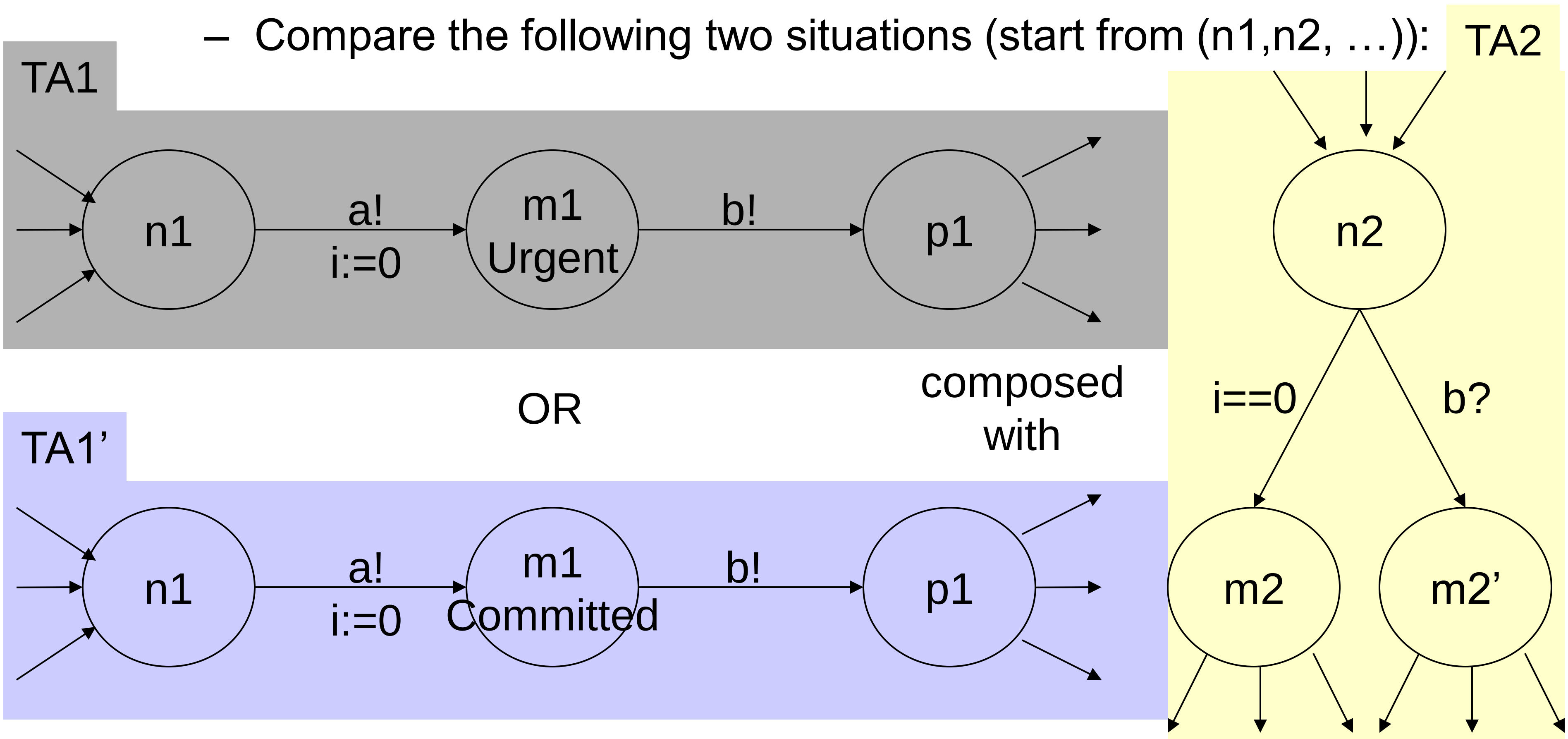


Modelling in UPPAAL

- Committed locations
  - Informal semantics:
    - *No time delay is possible when some timed automaton component of the system is in a committed location*
    - *The next transition must involve a timed automaton in a committed location*

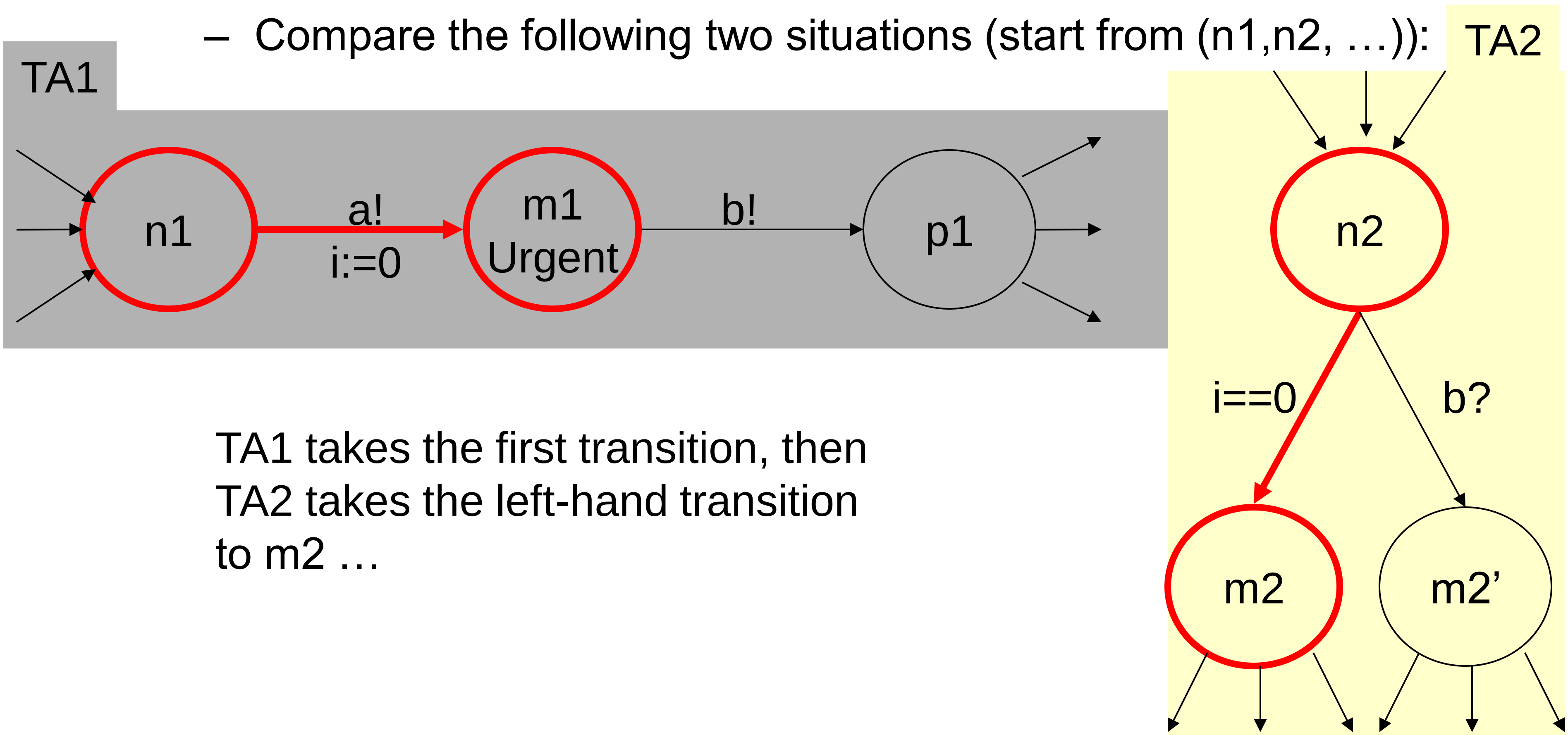
Modelling in UPPAAL

- Committed locations
  - Compare the following two situations (start from (n1,n2, …)):



Modelling in UPPAAL

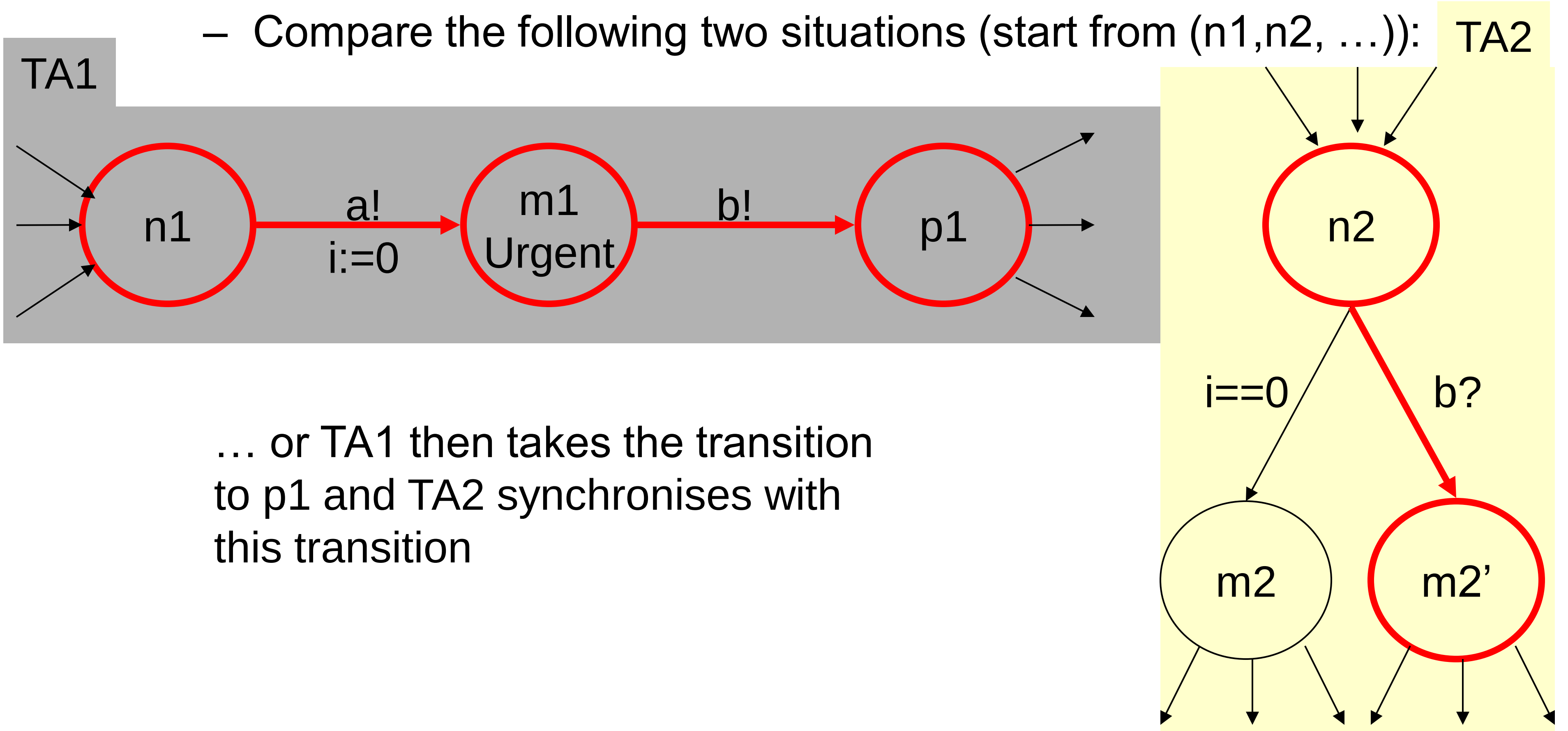
- Committed locations
  - Compare the following two situations (start from (n1,n2, …)):



TA1 takes the first transition, then TA2 takes the left-hand transition to m2 …

Modelling in UPPAAL

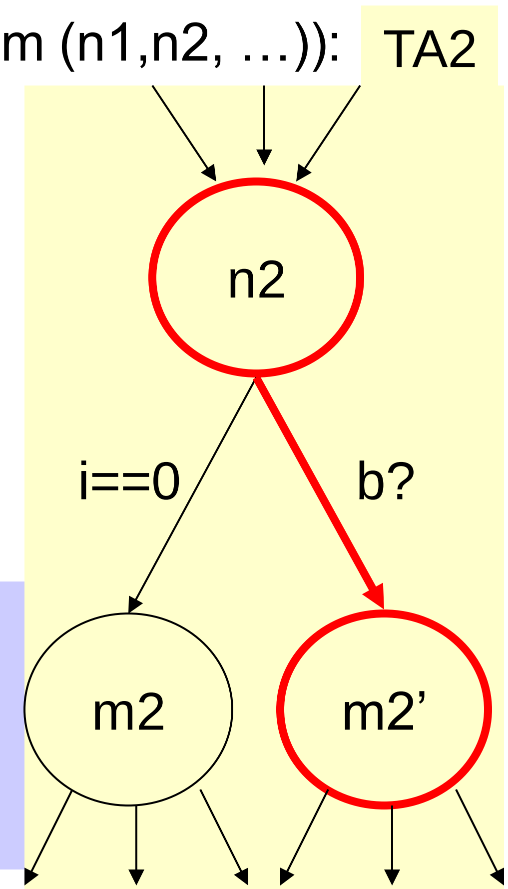
- Committed locations
  - Compare the following two situations (start from (n1,n2, …)):



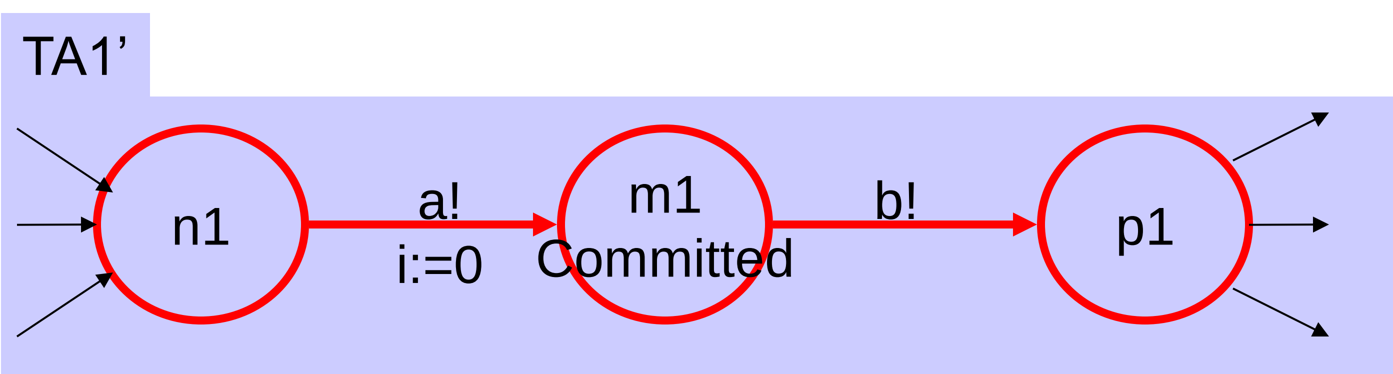
… or TA1 then takes the transition to p1 and TA2 synchronises with this transition

Modelling in UPPAAL

- Committed locations
  - Compare the following two situations (start from (n1,n2, …)):

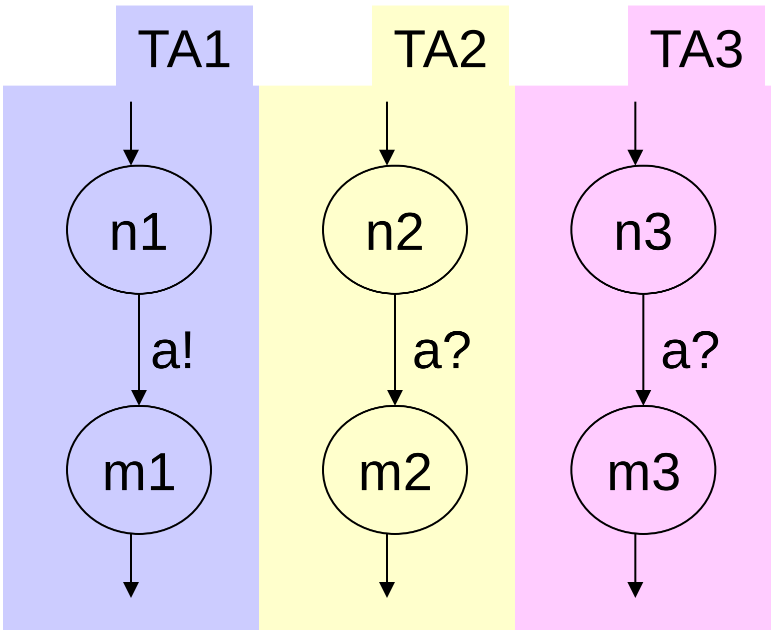


In the case the m1 is committed, TA2 does not have the opportunity to take the transition to m2: only TA1' can take a transition



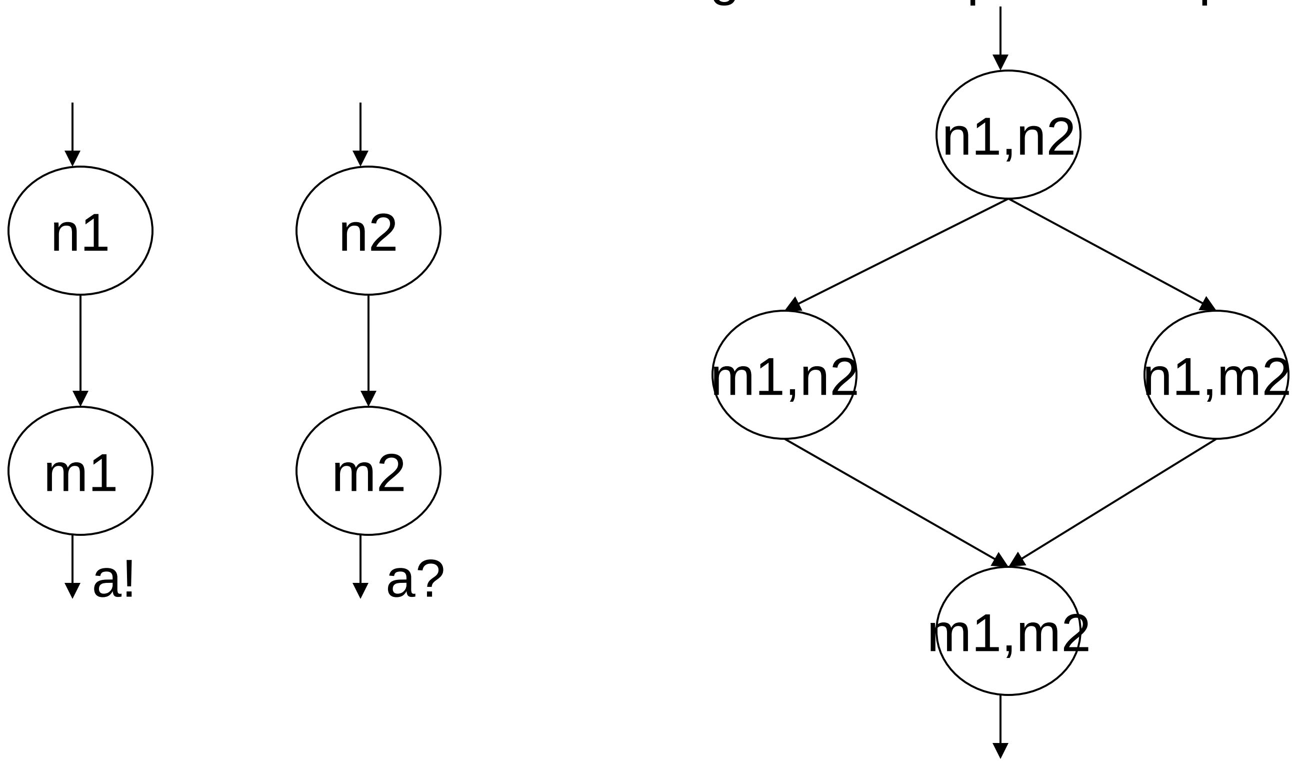
Modelling in UPPAAL

- Committed locations
  - Can aid modelling (e.g. for multi-way synchronization)
    - Example: to synchronize on a! in TA1, a? in TA2, and a? in TA3



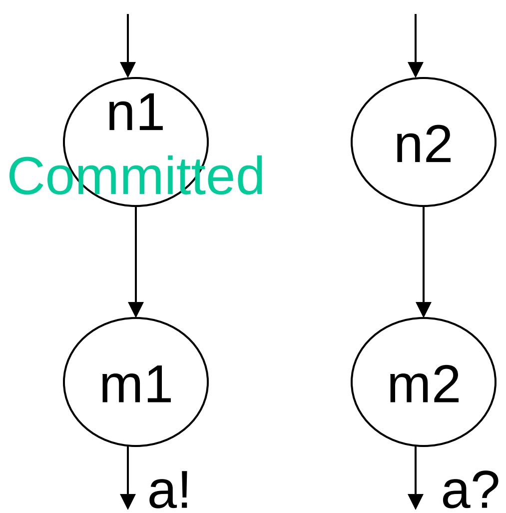
Modelling in UPPAAL

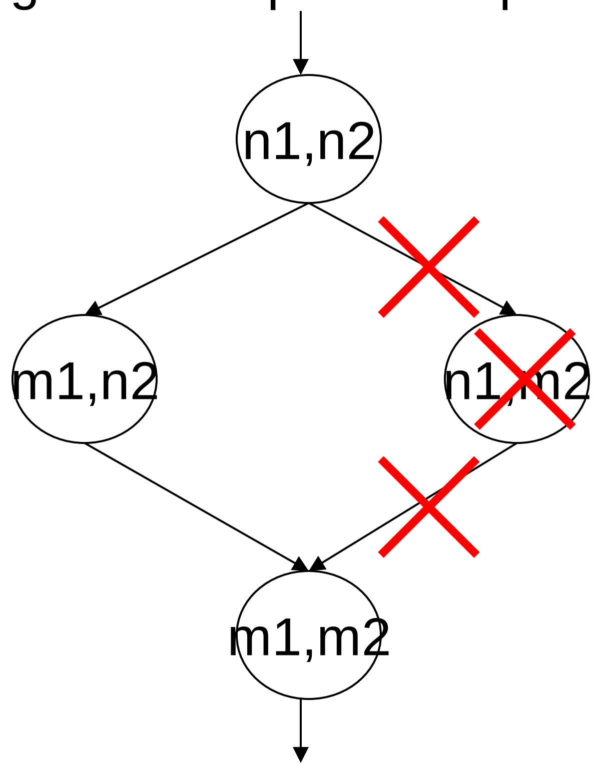
- Committed locations
  - Can aid modelling (e.g. for multi-way synchronization)
  - Can reduce the interleaving in state space computation



Modelling in UPPAAL

- Committed locations
  - Can aid modelling (e.g. for multi-way synchronization)
  - Can reduce the interleaving in state space computation





Modelling in UPPAAL

- Extensions to the UPPAAL modelling language:
  - Broadcast channels
  - Arrays of data variables (which can be referred to in guards and assignments)
  - Arrays of channels, clocks and constants
  - Further operators on data variables (e.g. i++)
  - Priorities on channels and processes
  - C-like functions
  - Others …

Verifying in UPPAAL

- Specification language: a subset of CTL
  - `A[]` p (corresponds to AG p)
  - `A<>` p (corresponds to AF p)
  - `E<>` p (corresponds to EF p)
  - `E[]` p (corresponds to EG p)
  - p `-->` q (corresponds to AG(p → AF q))

Verifying in UPPAAL

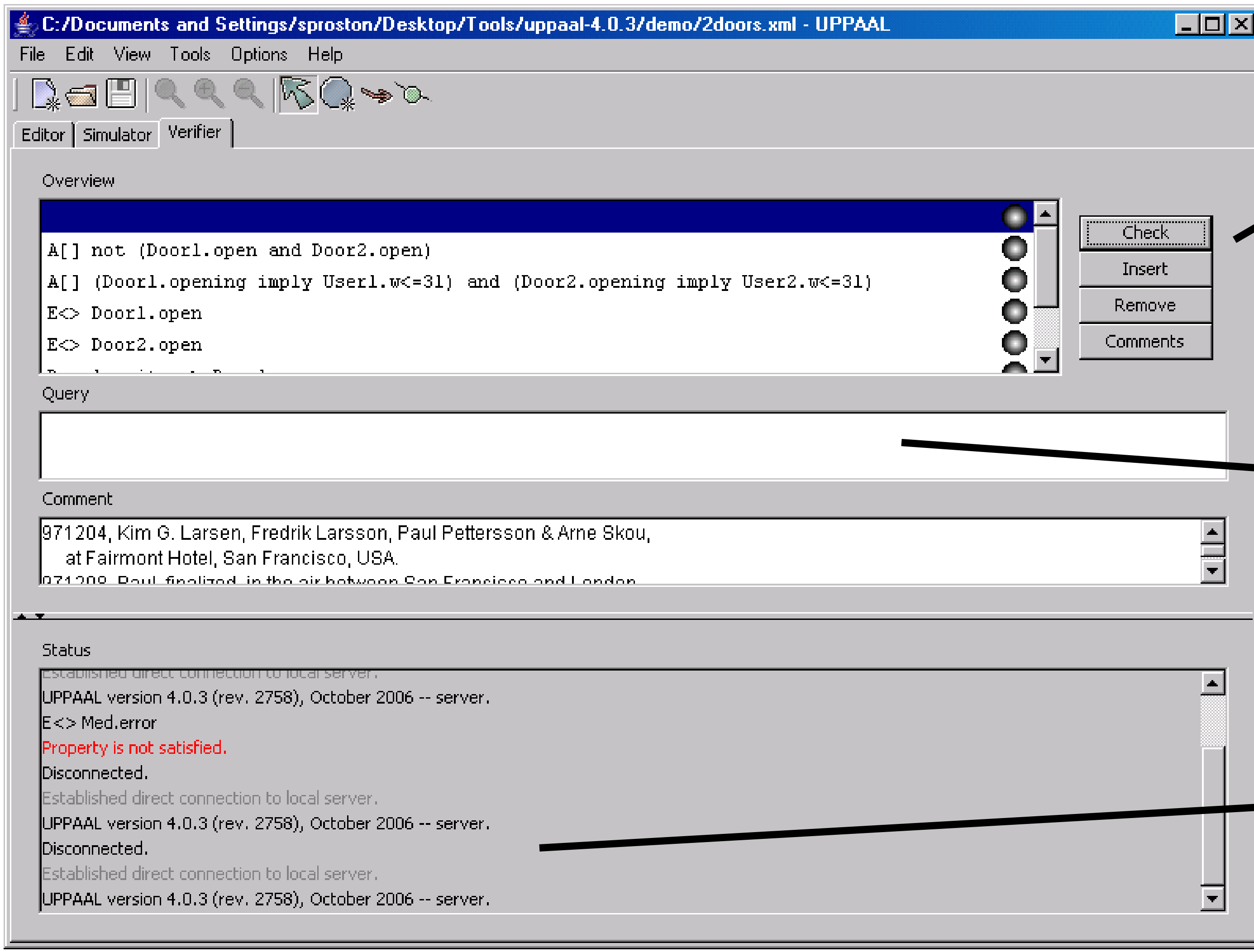
- `A[] p, A<> p, E<> p, E[] p, p --> q`

p ::= a.l | gd | gc | p `and` p | p `or` p | `not` p | p `imply` p | `(` p `)`

where:

- a is the name of a timed automaton
- l is the name of a location of a
- gd is an expression over data variables
- gc is an expression over clock variables

Verifying in UPPAAL

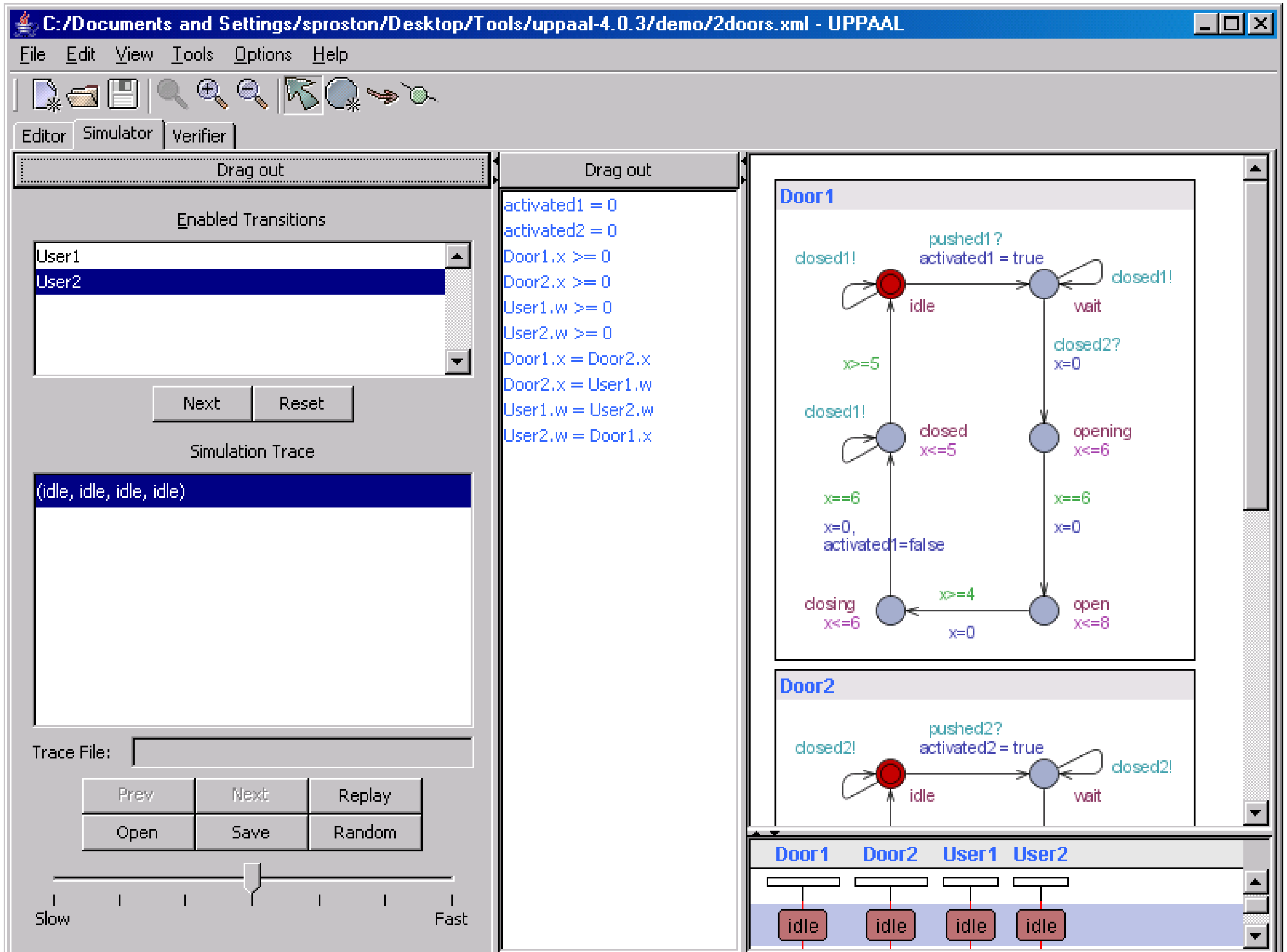


Check to verify a property, insert to add a new property

New queries can be written here

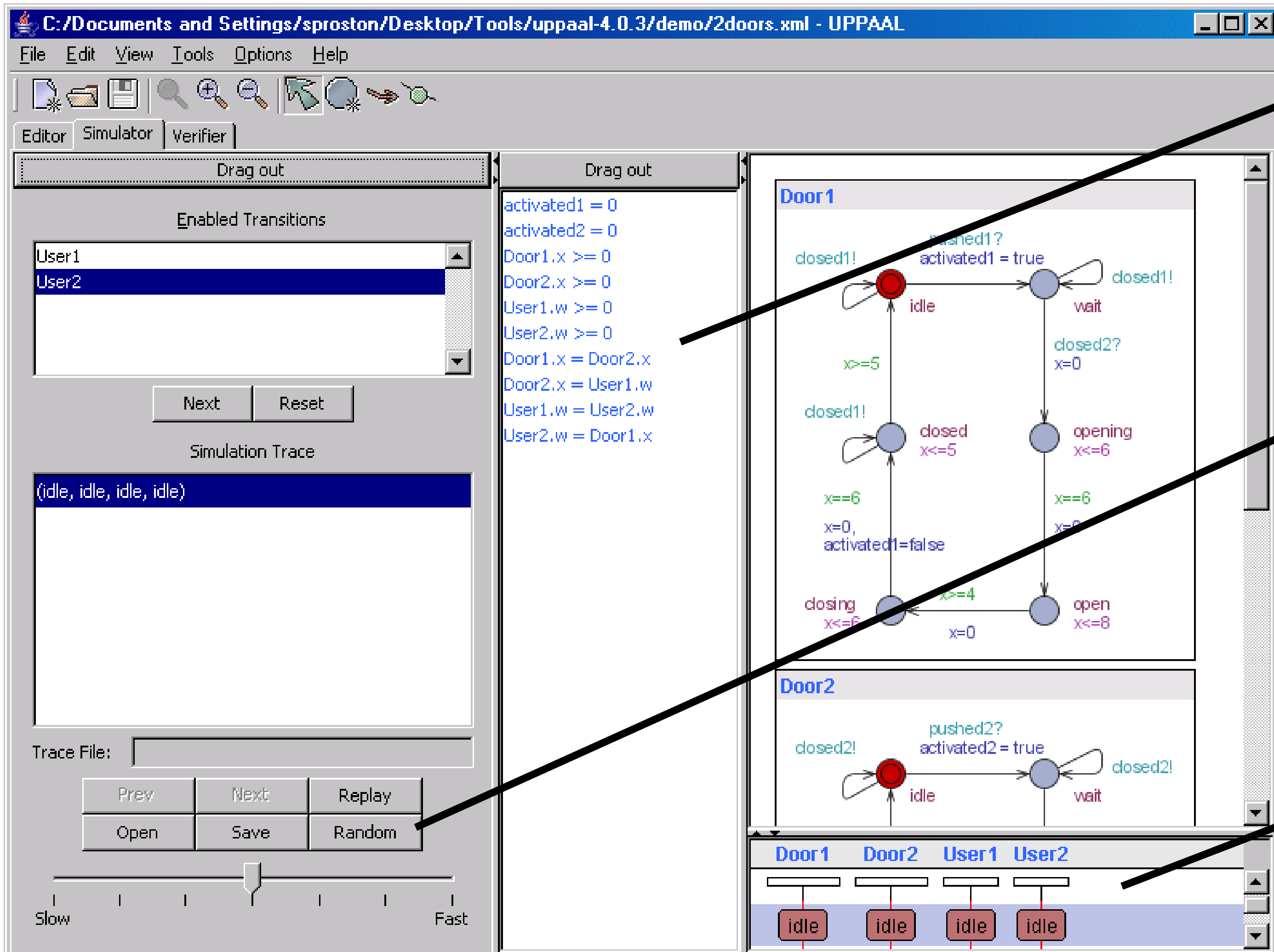
Results (property satisfied – or not - in the initial state)

UPPAAL's simulator



Permits exploration of the system following a (random or user-specified) behaviour

UPPAAL's simulator



List of variables (including possible clock values)

Random generates a random trace

Message sequence chart describing the interaction of components

UPPAAL's simulator

- The simulator can be used to visualise "error traces" generated by the verifier (choosing an option from "Diagnostic trace")
- For example:
  - If `E<>` p is satisfied, UPPAAL can return a trace which leads from the initial state to a state in which p is true
  - Dually, if `A[]` p is not satisfied, UPPAAL can return a trace which leads from the initial state to a state in which p is false
  - Similar for `E[]` p and `A<>` p, except traces containing loops are returned