UPPAAL

Evelin Halling

06.02.2013

Introduction

- UPPAAL is a tool for modeling, validation and verification of real-time systems.
- Appropriate for systems that can be modeled as a collection of non-deterministic processes with finite control structure and real-valued clocks (i.e. timed automata)
- UPPAAL = UPP (Uppsala University) + AAL (Aalborg University).
- http://www.uppaal.org

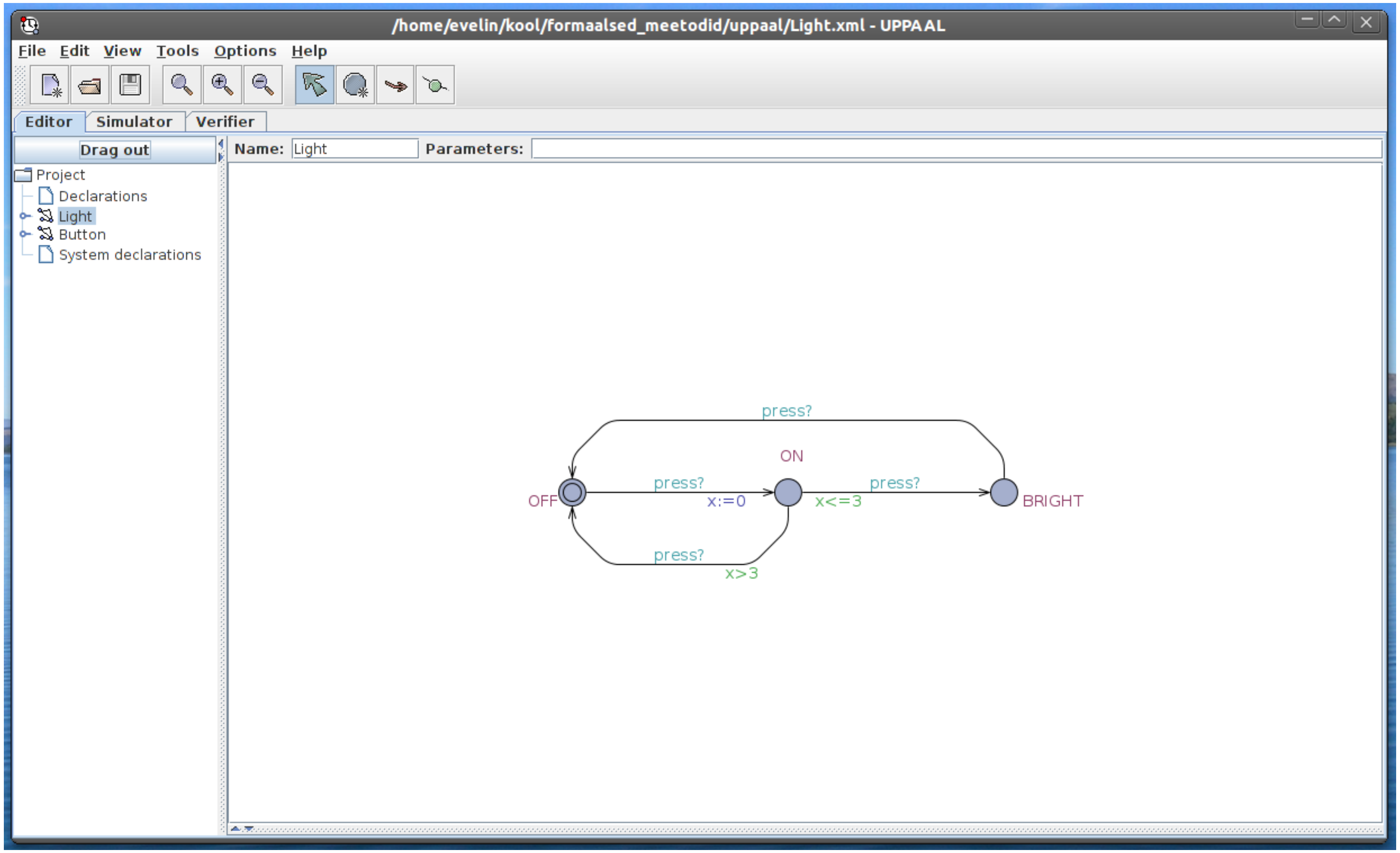
Introduction

- It serves as a modeling or design language to describe system behavior as networks of automata extended with clock and data variables.
- It used to make a simulation of the system and check if there is an error in the system.
- Communication is through channels and (or) shared data structures.
- Typical application areas:
  - Real-time controllers
  - Communication protocols
  - Other systems in which timing aspects are critical

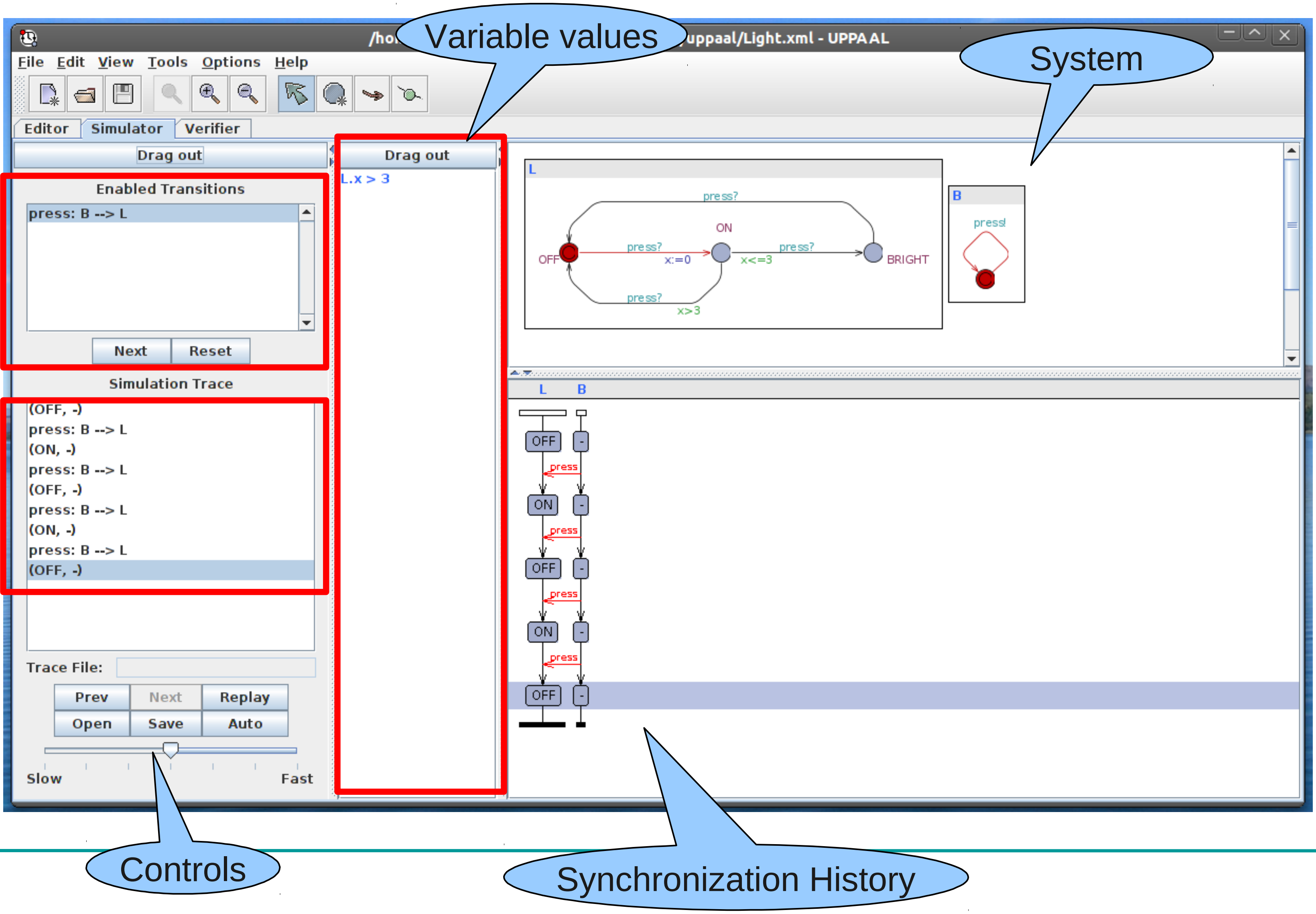
UPPAAL Tool Parts

- Graphical user interface (GUI)
  - Used for modeling, simulation, and verification. Uses the verification server for simulation and verification.
- Verification server
  - Used for simulation and verification. In simulation, it is used to compute successor states.
- A command line tool
  - A stand-alone verifier, appropriate for e.g. batch verifications.

Editor


/home/evelin/kool/formaalsed_meetodid/uppaal/Light.xml - UPPAAL
File Edit View Tools Options Help
Editor Simulator Verifier
Drag out
Project
Declarations
Light
Button
System declarations
Name: Light
Parameters:
press?
ON
OFF
press?
x:=0
press?
x<=3
BRIGHT
press?
x>3

Simulator



press: B --> L

(OFF, -)
press: B --> L
(ON, -)
press: B --> L
(OFF, -)
press: B --> L
(ON, -)
press: B --> L
(OFF, -)

L.x > 3

Simulation

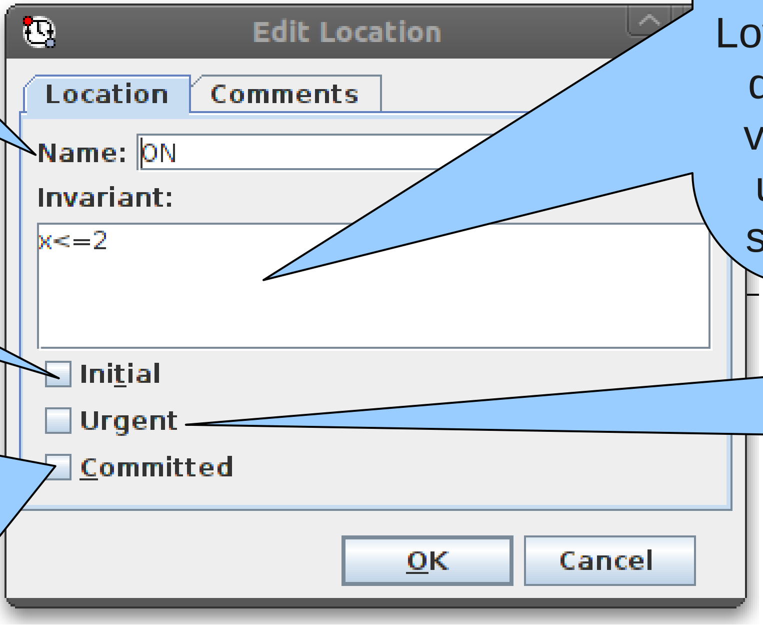
- Step-by-step simulation
  - Good for observations of variable values at each step
  - Manually selecting transitions (when many are enabled)
  - Good for tracing errors
- Automatic simulation
  - Good for observing overall system behavior
- Saving/Opening Simulation Traces

Locations

Locations can have an optional name. The name must be a valid identifier.

Exactly one per Template

Like urgent locations, committed locations freeze time. Furthermore, if any process is in a committed location, the next transition must involve an edge from one of the committed locations.



Name: ON

Invariant:

x<=2

Initial [ ]

Urgent [ ]

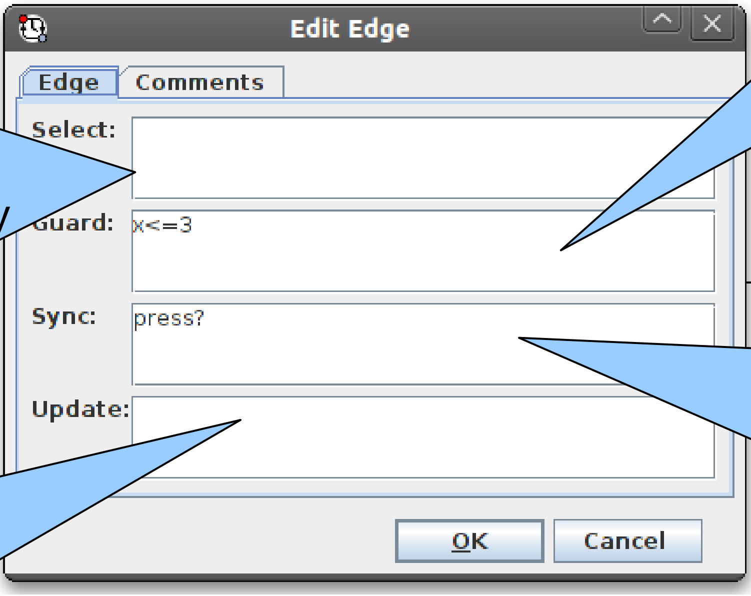
Committed [ ]

Conjunction of simple conditions on clocks, differences between clocks, and boolean expressions not involving clocks. The bound must be given by an integer expression. Lower bounds on clocks are disallowed. States which violate the invariants are undefined; by definition, such states do not exist.

Freeze time; *i.e.* time is not allowed to pass when a process is in an urgent location.

Edges

Selections are randomized initialization of some variable in a range whenever an edge is executed. The other three labels of an edge are within the scope of this binding. E.g., “i: int[3,5]” – randomly set i to be between 3 to 5, inclusively.



Select:

Guard: x<=3

Sync: press?

Update:

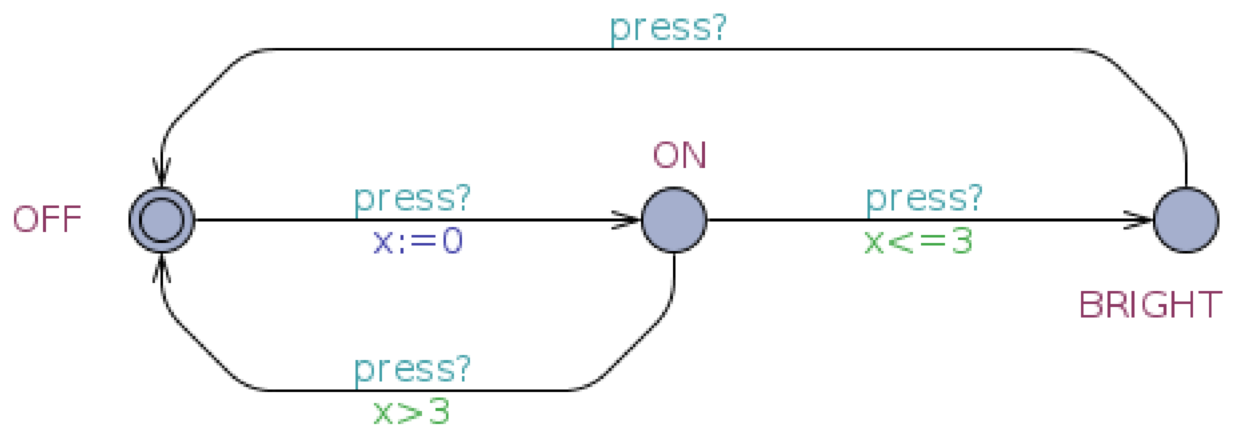
An edge is enabled in a state if and only if the guard evaluates to true.

Processes can synchronize over channels. Edges labeled with complementary actions over a common channel synchronize (press! press?).

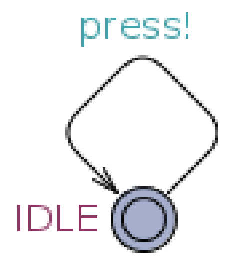
When executed, the update expression of the edge is evaluated. The side effect of this expression changes the state of the system.

The Light Controller Example

Figure (a) shows a timed automaton modelling a simple lamp. The lamp has three locations: OFF, ON, and BRIGHT. If the user presses a button, i.e., synchronises with press? , then the lamp is turned on. If the user presses the button again, the lamp is turned off. However, if the user is fast and rapidly presses the button twice, the lamp is turned on and becomes bright.
The user model is shown in figure (b). The user can press the button randomly at any time or even not press the button at all. The clock x of the lamp is used to detect if the user was fast (y <= 3) or slow (y > 3).



(a) Lamp.



(b) User.

Variables

- Integer – int (range -32768 … 32767)
  int n1, n2;
  int[0,300] n1; --- variable in range 0-300
  int n[2][3];
  int[0,5] n1 = 0;
- Boolean – bool (True (1), False (0))
  bool y = true;
  bool b[4]; --- array of 4 elements
- Constant
  const int a=5; const bool b = fase;

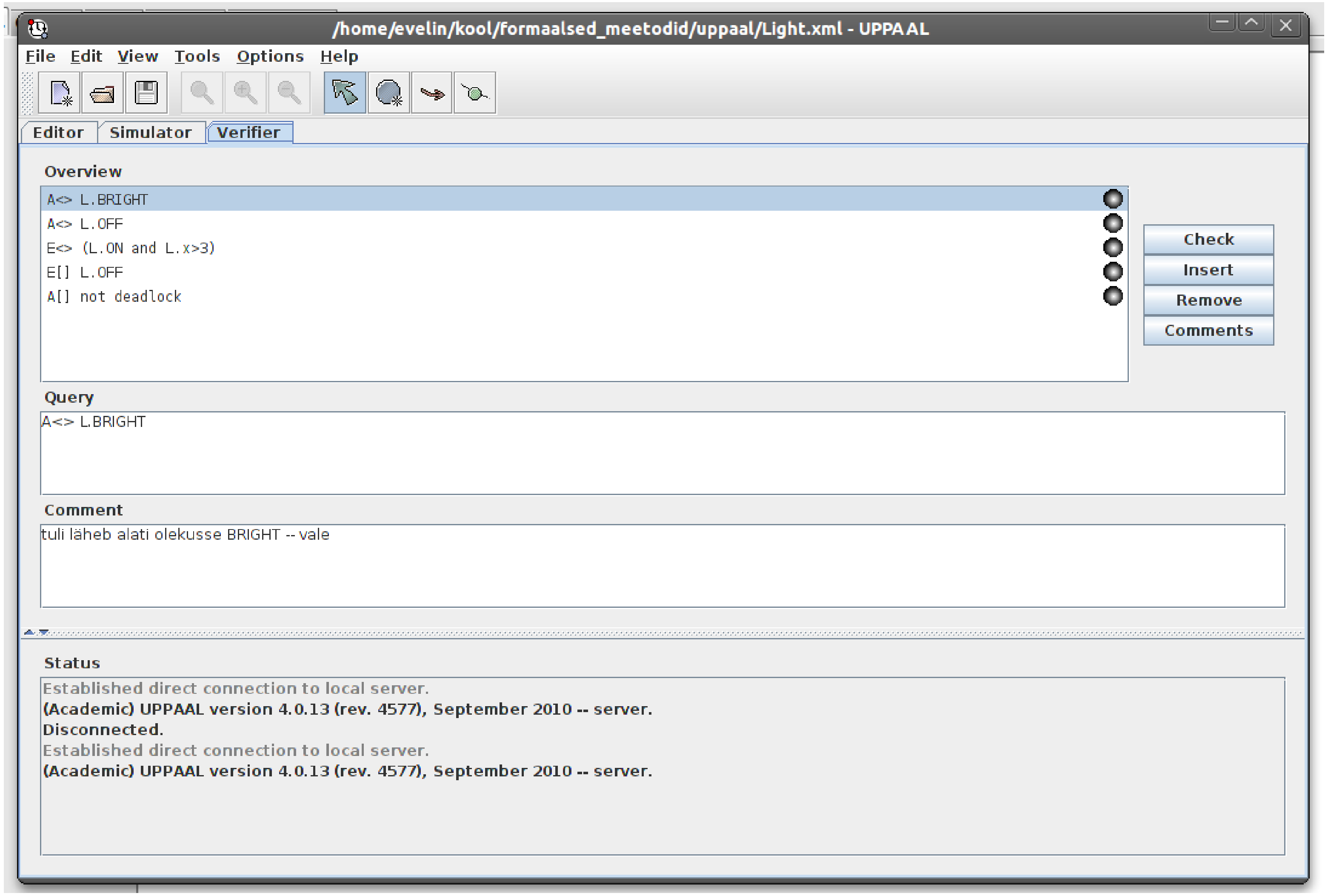
Variables

- Clock - clock

  clock cl1, cl2;
- Channel – chan (used to synchronize two processes)
  chan ch;

  ch! – sending
  ch? – receiving
- Broadcast channels
- Urgent channels – higher priority, clock guard not allowed.

Verifier



/home/evelin/kool/formaalsed_meetodid/uppaal/Light.xml - UPPAAL
File Edit View Tools Options Help
Editor Simulator Verifier
Overview
Check
Insert
Remove
Comments
Query
Comment
Status

```
A<> L.BRIGHT
A<> L.OFF
E<> (L.ON and L.x>3)
E[] L.OFF
A[] not deadlock
```

Query:

```
A<> L.BRIGHT
```

Comment:

tuli läheb alati olekusse BRIGHT -- vale

Status:

```
Established direct connection to local server.
(Academic) UPPAAL version 4.0.13 (rev. 4577), September 2010 -- server.
Disconnected.
Established direct connection to local server.
(Academic) UPPAAL version 4.0.13 (rev. 4577), September 2010 -- server.
```

Query in Uppaal

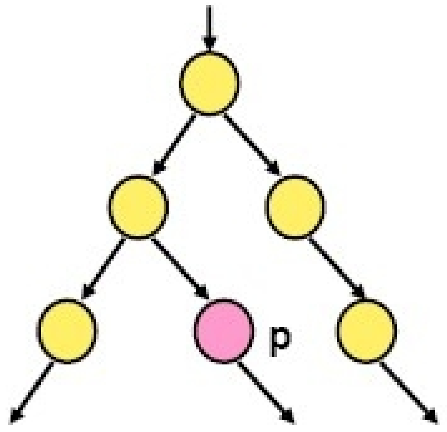
- E - exists a path (“**E**” in UPPAAL).
- A - for all paths (“**A**” in UPPAAL).
- [ ] – all states in a path
- <> - some states in a path

The following combination are supported:

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

E<> p – “p Reachable”

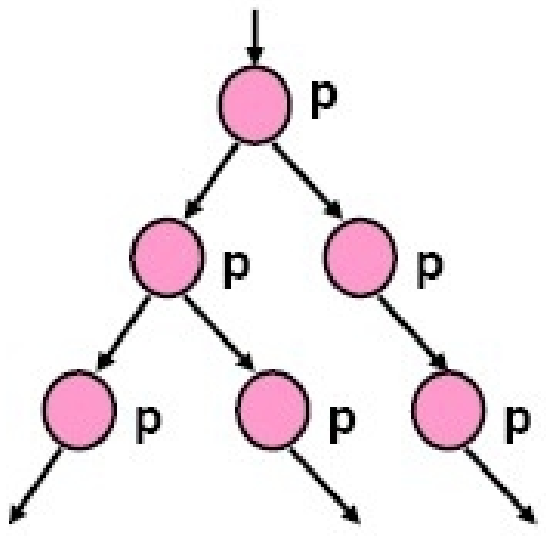
E<> p – it is possible to reach a state in which p is satisfied.



P is true in (at least) one reachable state.

A[ ] p – “Invariantly p”

A[ ] p – p holds invariantly.

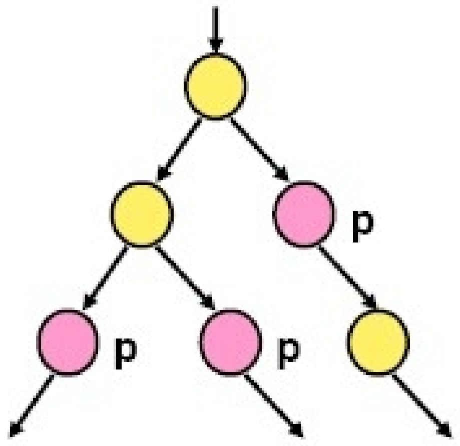


P is true in all reachable states.

A<> p – “Inevitable p”

A<> p – p will inevitable become true

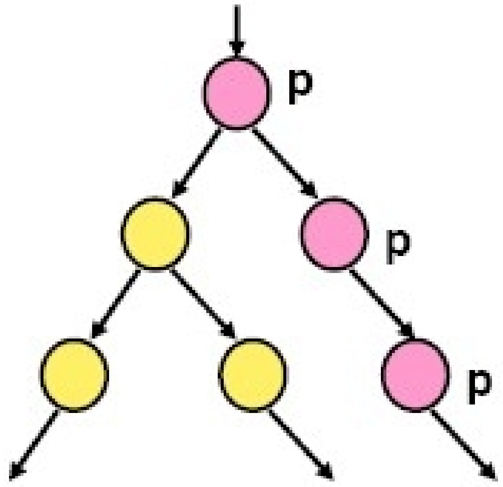
The automaton is guaranteed to eventually reach a state in which p is true.



P is true in some state of all paths.

E[ ] p – “Potentially Always p”

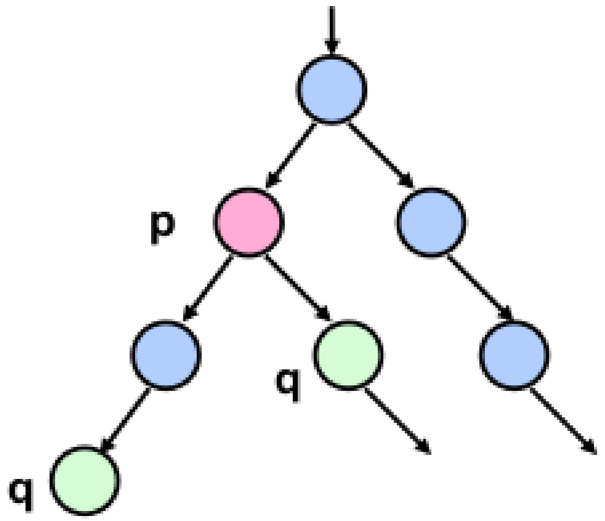
E[ ] p – p is potentially always true.



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)

Specifying Properties

- A[ ] not deadlock

  - no deadlocks
  - true

- E[ ] L.OFF

  - is it possible that the the light is always OFF
  - true

- E<> (L.ON and L.x >3)

  - it is possible that the light isn't pressed a second time within 3 seconds after it’s turned on
  - true

- A<> L.OFF

  - no matter how your operate the light, it will go to OFF
  - true

- A<> L.BRIGHT

  - no matter how your operate the light, it will go to BRIGHT
  - false