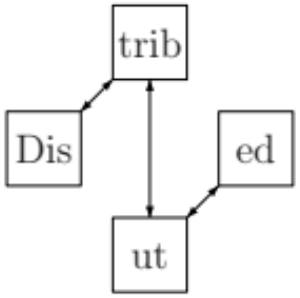
UPPAAL Tutorial

Modeling Real-Time Systems

01.02.2018

Deepak Pal

About me

- Ph.D.
  - Model based testing of 

systems.

http://research-deepak.com

What is a model?

- A model is a description of a system's behavior.
- Behavior can be described in terms of input sequences, actions, conditions, output and flow of data from input to output.
- It should be practically understandable and can be reusable; shareable must have precise description of the system under test.

What is a model?

- A model is a description of a system's behavior.
- Behavior can be described in terms of input sequences, actions, conditions, output and flow of data from input to output.
- It should be practically understandable and can be reusable; shareable must have precise description of the system under test.
- Can you name some model formalisms to describe different aspects of system behavior ?

What is a model?

- A model is a description of a system's behavior.
- Behavior can be described in terms of input sequences, actions, conditions, output and flow of data from input to output.
- It should be practically understandable and can be reusable; shareable must have precise description of the system under test.
- Can you name some model formalisms to describe different aspects of system behavior ?
  - FSM, State chart, UML, Decision Tables, Timed Automata .. etc

UPPAAL Introduction

- UPPAAL is a tool for modeling, validation (via graphical simulation) and verification (via automatic model-checking) 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)

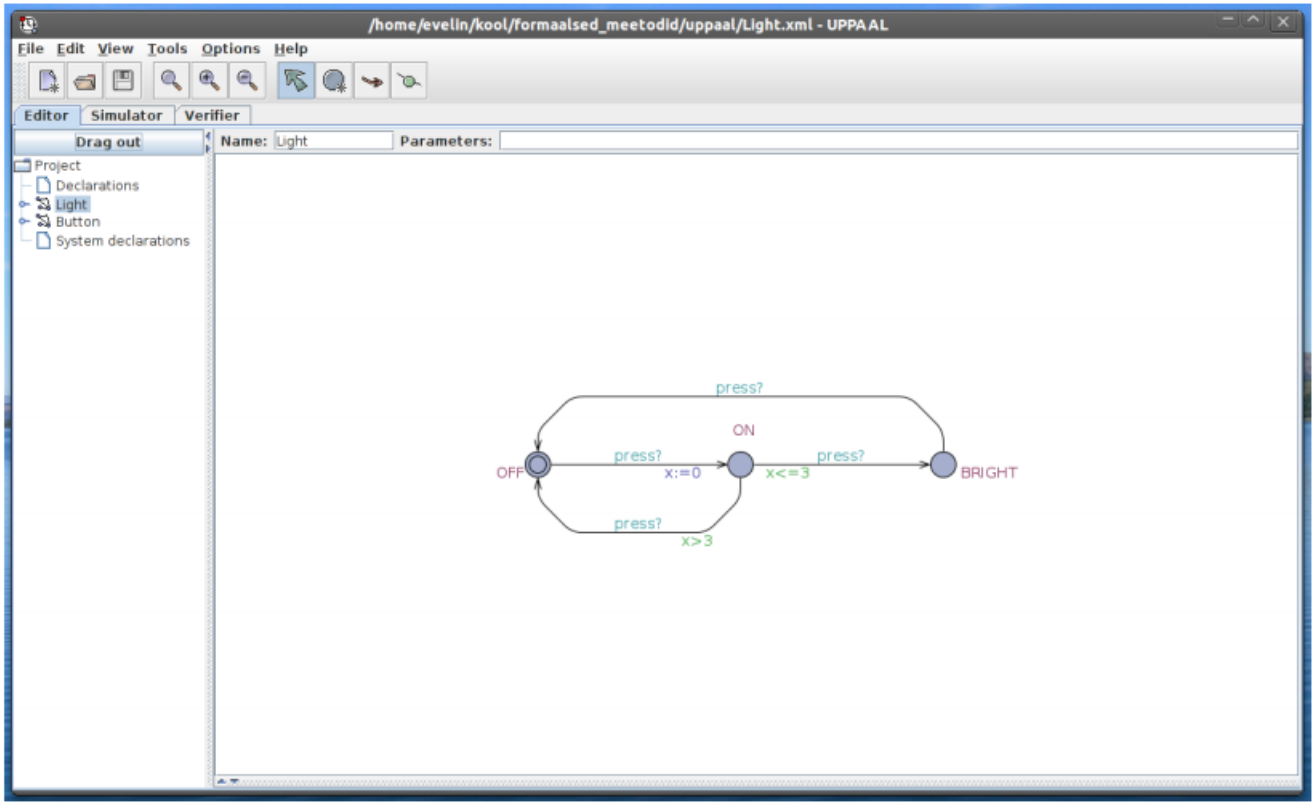
- UPPAAL = UPP (Uppsala University) + AAL (Aalborg University).

- http://www.uppaal.org

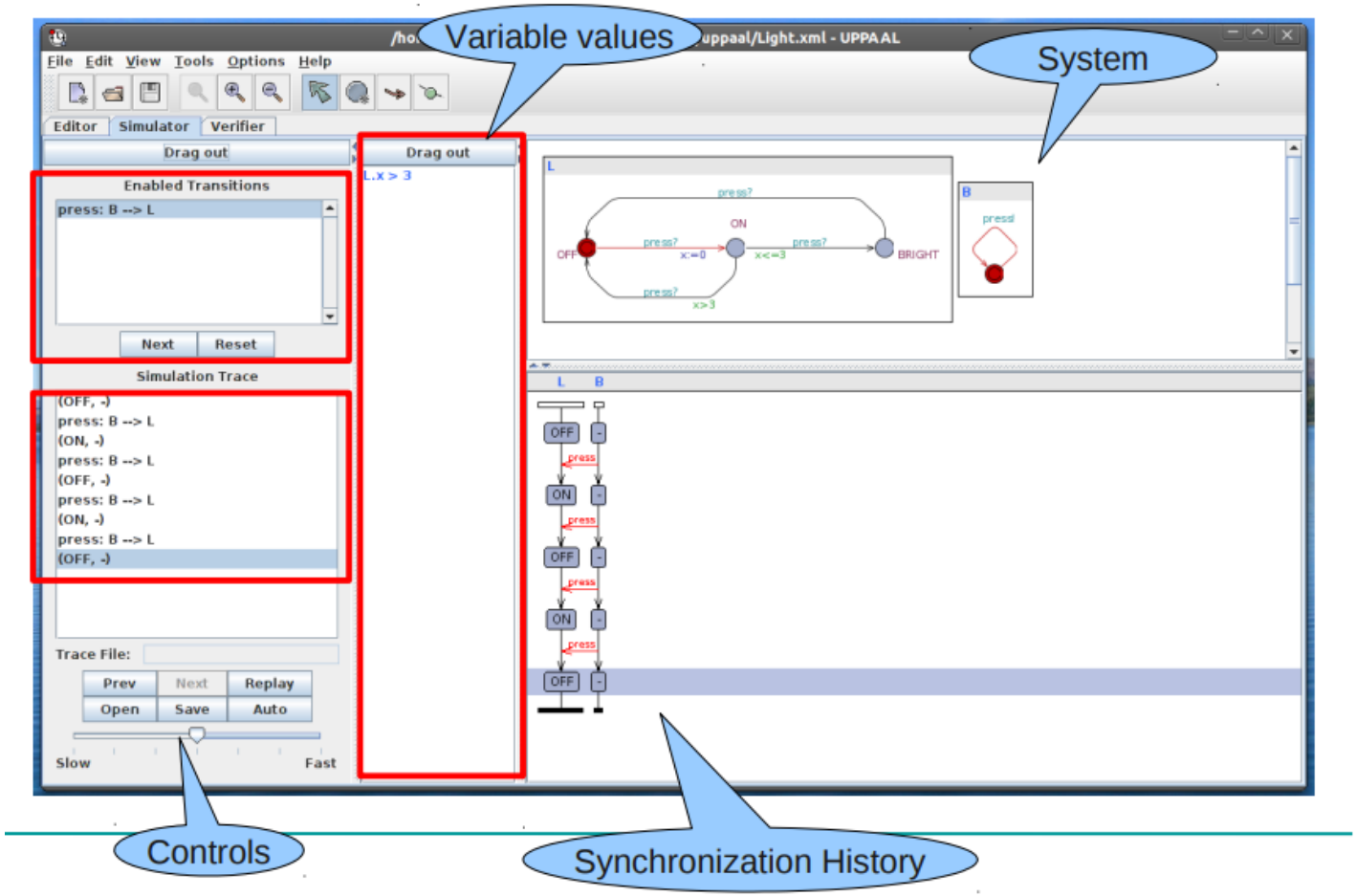
UPPAAL Introduction

- UPPAAL is a tool for modeling, validation (via graphical simulation) and verification (via automatic model-checking) 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

Editor


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

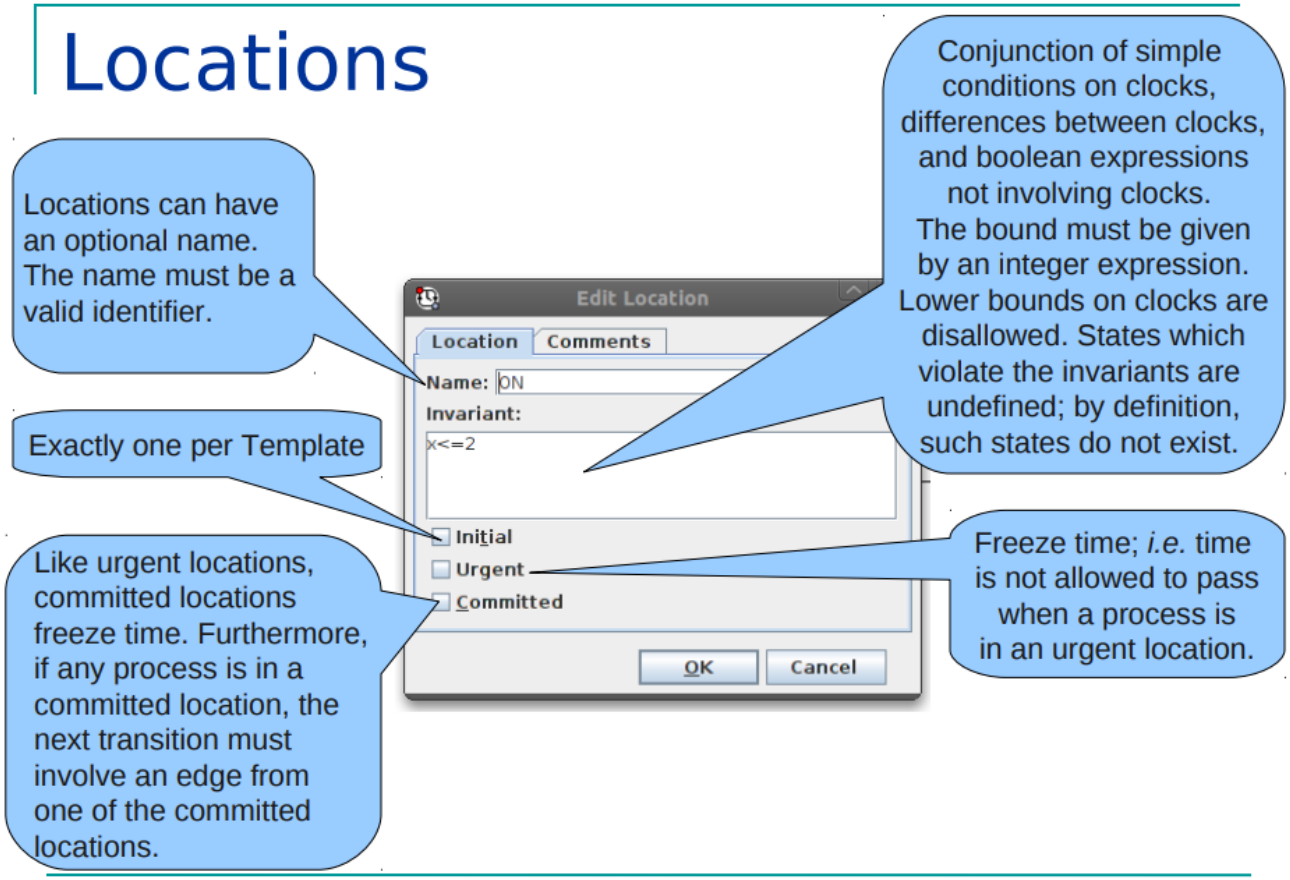
Simulator


Variable values
System
File Edit View Tools Options Help
Editor Simulator Verifier
Drag out
Enabled Transitions
press: B --> L
Next Reset
Simulation Trace
(OFF, -)
press: B --> L
(ON, -)
press: B --> L
(OFF, -)
press: B --> L
(ON, -)
press: B --> L
(OFF, -)
Trace File:
Prev Next Replay
Open Save Auto
Slow Fast
Drag out
L.x > 3
L
B
OFF
ON
BRIGHT
press?
x:=0
x<=3
x>3
press!
Controls
Synchronization History

Simulation

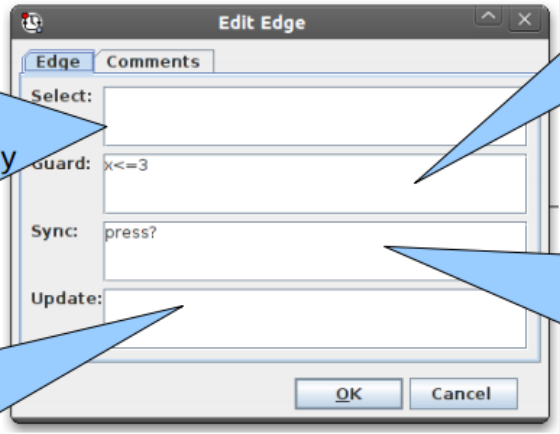
- Step-by-step simulation –
  - Good for observations of variable values at each step - Manually selecting transitions (when many are enabled)
  - Good for tracing errors
- Automatic simulation - Good for observing overall system behavior
- Saving/Opening Simulation Traces

Locations


Locations
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.
Edit Location
Location
Comments
Name: ON
Invariant:
x<=2
Initial
Urgent
Committed
OK
Cancel
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.



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

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

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

Declaration

- Declarations are either global or local (to a template) and can contain declarations of clocks, bounded integers, channels (although local channels are useless), arrays, records, and types.

- `chan d;`

  a channel.

- `urgent chan e;`

  an urgent channel.

- `struct { int a; bool b; } s1 = { 2, true };`

  an instantiation of the structure from above where the members `a` and `b` are set to `2` and `true`.

- `const int a = 1;`

  constant `a` with value 1 of type integer.

- `bool b[8], c[4];`

  two boolean arrays `b` and c, with 8 and 4 elements respectively.

- `int[0,100] a=5;`

  an integer variable with the range [0, 100] initialised to 5.

- `int a[2][3] = { { 1, 2, 3 }, { 4, 5, 6} };`

  a multidimensional integer array with default range and an initialiser.

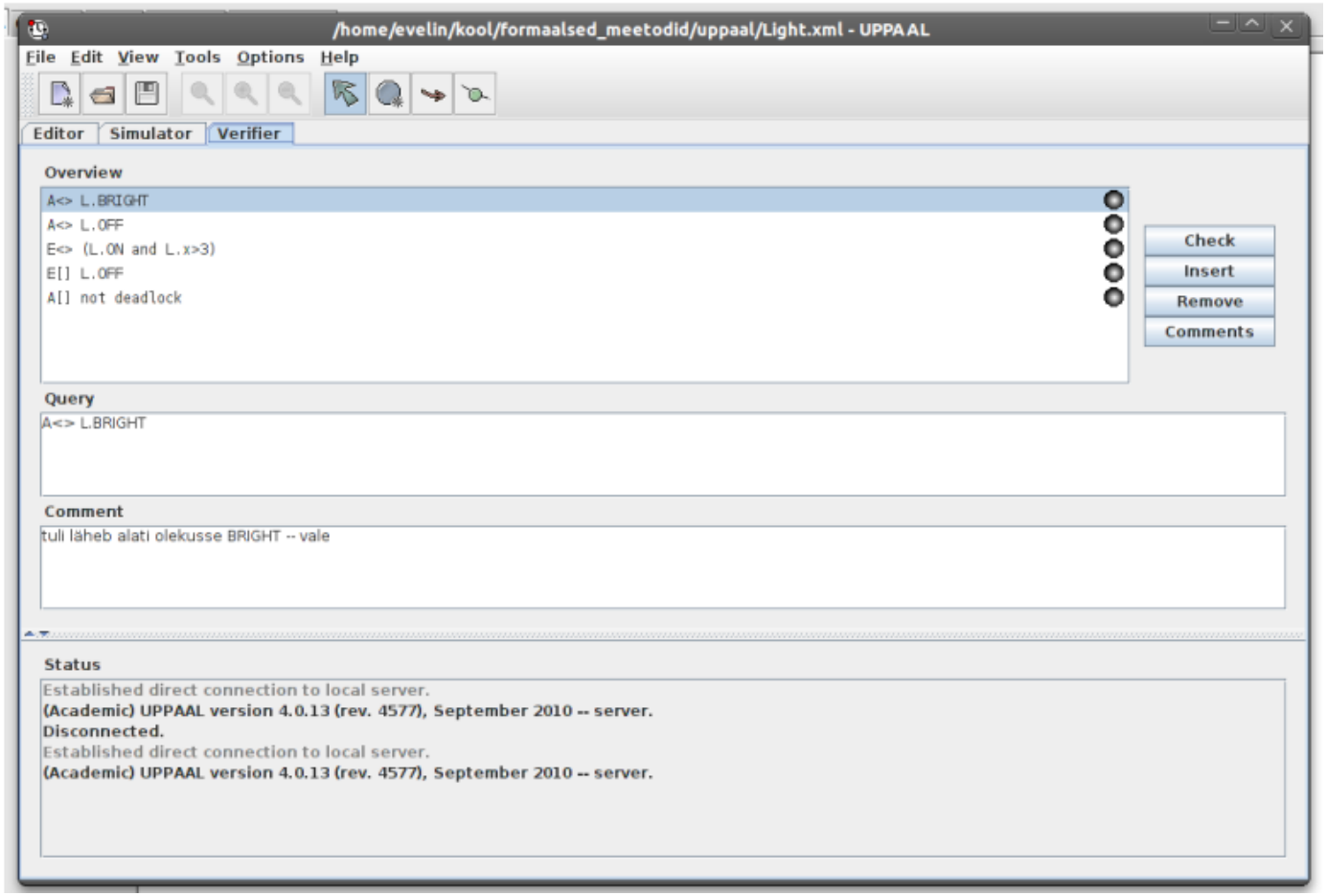
- `clock x, y;`

  two clocks `x` and `y`.

More information:

http://www.it.uu.se/research/group/darts/uppaal/help.php?file=Introduction.shtml

VERIFIER



The query language of Uppaal, used to specify properties to be checked, is a subset of TCTL (timed computation tree logic)

Query in Uppaal

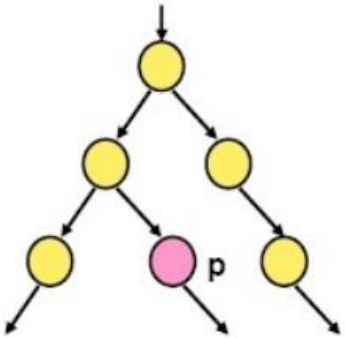
- E - exists a path (“**E**” in UPPAAL).
- A - for all paths (“**A**” in UPPAAL).
- [ ] – all states in a path
- <> - some states in a path

The following combination are supported:

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

E<> p – "p Reachable"

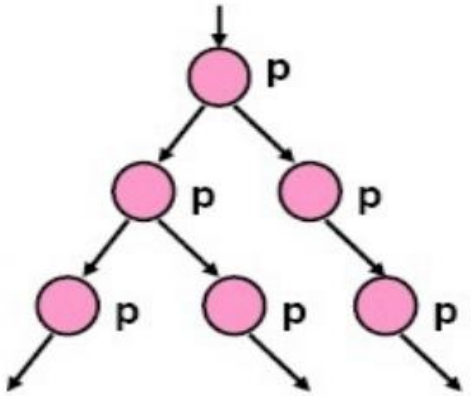
E<> p – it is possible to reach a state in which p is satisfied.



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

A[ ] p – "Invariantly p"

A[ ] p – p holds invariantly.

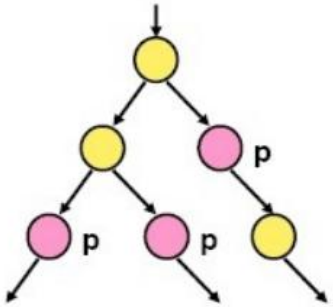


P is true in all reachable states.

A<> p – "Inevitable p"

A<> p – p will inevitable become true

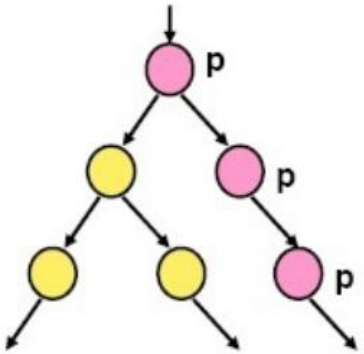
The automaton is guaranteed to eventually reach a state in which p is true.



P is true in some state of all paths.

E[ ] p – "Potentially Always p"

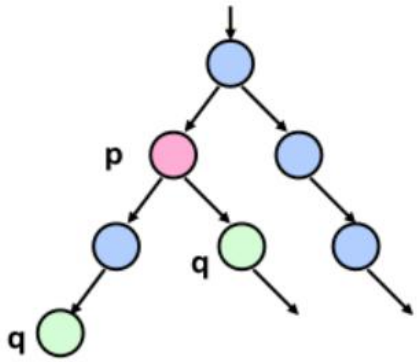
E[ ] p – p is potentially always true.



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

Light Controller Example

• Model the abstract behavior of a simple light controller switch shown in fig below.

Requirements

- The lamp has three locations: OFF, ON, and BRIGHT.
- If the user presses a button, i.e., synchronizes 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 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 (x <= 3) or slow (x > 3).

Next Lab:

- Urgent and Committed Locations
- Urgent Channels
- Broadcast Synchronization .
- Networks of Timed Automata
- Model Checking