Formal Methods in Software Engineering

Lecture 03 – Organizational Issues

Prof. Dr. Joel Greenyer

October 28, 2014

Organizational Issues

- Tutorial dates:
  - Tuesdays 15:00-16:00 in A310 (before the lecture, same room)
  - Wednesdays 9:00-10:00 in G325
    - I could not find a suitable room to move to Thursday afternoon
- I will be recording/streaming the tutorial on Tuesdays
  - If you want to present something, please *send me your slides at least one hour before the tutorial*, so I can open them on my laptop from where the stream happens
    - example: Subject “[FMSE] presentation of project results”
      - “Hi, I would like to present the results of our project in the tutorial next Tuesday...”

SOFTWARE SE ENGINEERING

Live Hangout Policy

- I will send an invitation to the live hangout of a lecture or tutorial to the whole FMSE community
  - I'm happy if you join – the more the merrier
- When you join, please set your microphone to mute
- Unmute your microphone when you want to say something
  - I set up speakers, so everyone can hear you
  - I will setup the hangout so that my screen stays in focus. So, nobody will see you. If you want to share something on your screen/camera, let me know, then I'll release the focus from my screen.
  - You can answer to questions I ask
  - I'll try to have a short pause after every slide where you can jump in and ask if you did not understand something
    - It's not impolite to interrupt me! – Don't worry, if it is too much, I'll let you know :)

Feedback to the Lecture/Tutorials

- There is always something about a lecture that **can be improved**
  - I'm happy to change things and try out something different

To improve is to change, so to be perfect is to have changed often.
Winston Churchill

SOFTWARE SE ENGINEERING

Feedback to the Lecture/Tutorials

- There is always something about a lecture that **can be improved**
  - I'm happy to change things and try out something different
- There is always something that **somebody will not like**
  - some would like more interactive lectures, others don't
  - some enjoy working in groups, others don't
- There are various ways that you can **provide feedback** to the lecture and **help improve** it
  - personally: talk to me, write me an email
  - anonymously: drop a letter in the mailbox on the G3 floor
  - I will provide ways for you to give feedback
  - There will be an evaluation in the middle of the semester

Feedback to the Lecture/Tutorials

- Participate in the current poll: (it's anoymous)

Privat 57 Mitglieder

Community suchen

Alle Beiträge

Announcements

General discussion

Forming groups

Programming

Veranstaltungen

Fotos

LUH Formal Methods in Software Engineering TV

General discussion - 09:27

#Poll

Share your opinion on the first mini-project!

Vote on the short poll below

Mehr anzeigen

Your opinion on the first mini-project!

How do you like the first mini-project?

(programming an LTS model, implementing a parallel composition operator, etc.)

- 5 (Exciting)
- 4 (Interesting)
- 3 (Okay)
- 2 (boring)
- 1 (can't stand it)

SOFTWARE SE ENGINEERING

If you give feedback... (Feedback Policy)

- Describe your opinion/feelings based on a **concrete situation**
- Formulate your feedback **subjectively**
  - If you generalize, you project your opinion on everyone else – others may feel very differently than you
  - **good (subjective):** "*I find* … too hard/easy", "*I have difficulties* understanding ...", "*I wish* you could ...", "*It would help me* if ..."
  - **not good (generalizing):** "... is too hard/easy", "nobody can understand ...", "you should ...", "it would be better if ..."
- Very helpful: give concrete suggestions for improvement
- If something bothers you, give your feedback **right away**
  - anonymously is fine, **personally is better**: We can think of a possible improvement together or I can explain why I am doing things a certain way

Formal Methods in Software Engineering

Lecture 03 – Modeling

Prof. Dr. Joel Greenyer

October 28, 2014

SOFTWARE SE ENGINEERING

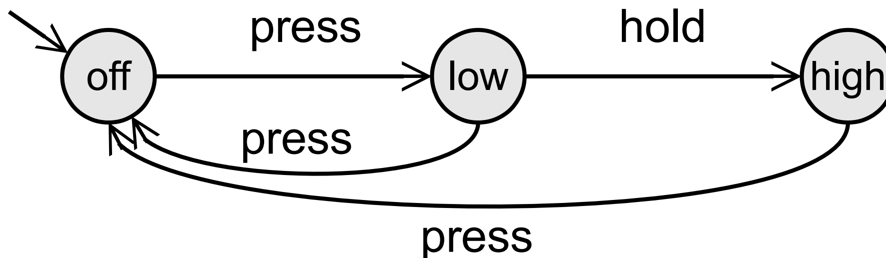
Learning Objective

- Understand fundamental modeling concepts and languages for concurrent reactive systems
- Get to know some tools that support modeling and analysis based on these fundamental modeling languages
- Understand that higher-level modeling languages can be mapped to fundamental modeling languages
  - unfolding, composition

Labeled Transition Systems (LTSs)

in the last lecture...

- A **(Labeled) Transition System** is a tuple $TS = (S, \Sigma, T, I)$
  - S is a set of **states**
  - $I \in S$ is the set of **initial states**
  - Σ is an alphabet, an element in Σ is called a **symbol** (a symbol is also called an **input**, **event**, or **action**)
  - $T \subseteq S \times \Sigma \times S$ is a **transition relation**
    - an element of T is a **transition**
  - A TS is called **finite** if S and Σ are finite



model of a flashlight

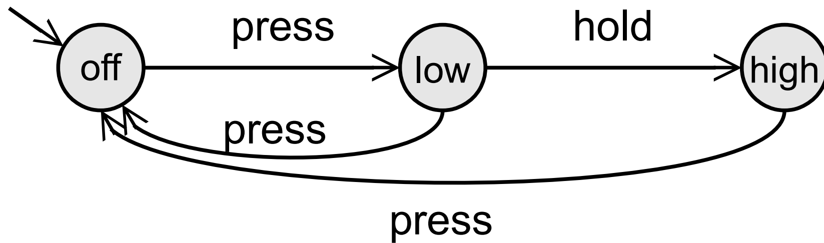
$S = \{off, low, high\}$
$I = \{off\}$
$\Sigma = \{press, hold\}$
$T = \{(off, press, low),$
$(low, hold, high),$
$(low, press, off)$
$(high, press, off)\}$

Parallel Composition via Handshaking

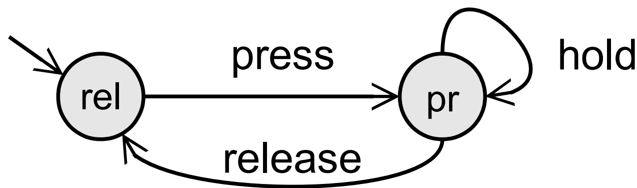
in the last lecture...

- Processes synchronize on certain common events
  - transitions with these events are executed simultaneously
  - only if both processes are “ready” execute a common event
  - the other transitions are interleaved

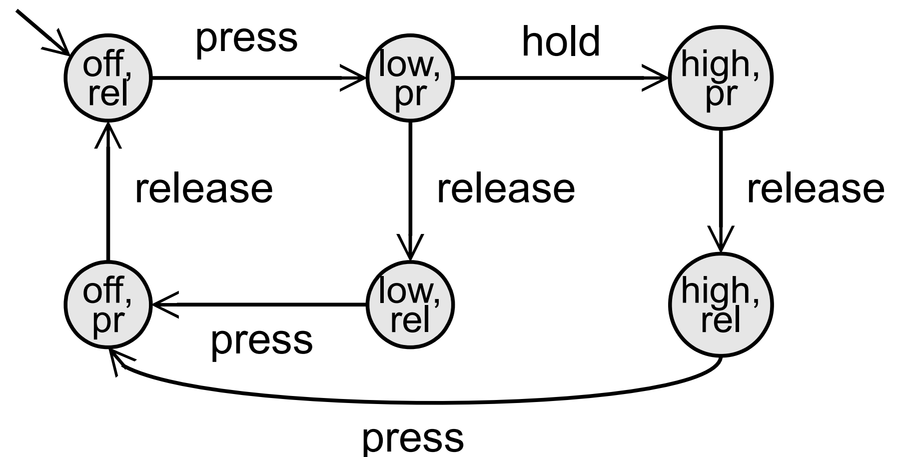
Light



Switch



$Light \, ||_{\{press,\ hold\}} \, Switch$



Handshaking

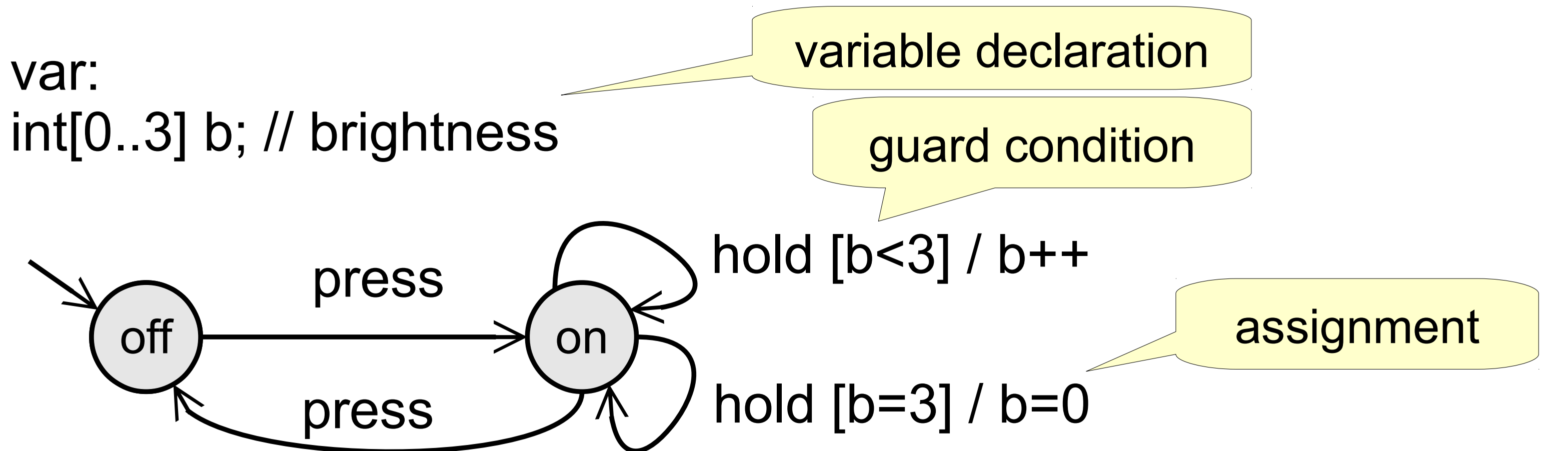
in the last lecture...

- Let $TS_1 = (S_1, \Sigma_1, T_1, I_1)$ and $TS_2 = (S_2, \Sigma_2, T_2, I_2)$ be two transition systems and $H \subseteq \Sigma_1 \cap \Sigma_2$ then $TS_1 \,||_H\, TS_2$ is defined as follows
  - $TS_1 \,||_H\, TS_2 = (S_1 \times S_2, \Sigma_1 \cup \Sigma_2, T, I_1 \times I_2)$
  - where T is defined by the following rules
    - if $a \in H$ and $(s_1, a, s'_1) \in T_1$ and $(s_2, a, s'_2) \in T_2$ then $(<s_1, s_2>, a, <s'_1, s'_2>) \in T$
    - if $a \in \Sigma_1 \setminus H$ and $(s_1, a, s'_1) \in T_1$ then $(<s_1, s_2>, a, <s'_1, s_2>) \in T$
    - if $a \in \Sigma_2 \setminus H$ and $(s_2, a, s'_2) \in T_2$ then $(<s_1, s_2>, a, <s_1, s'_2>) \in T$
- If $H = \Sigma_1 \cap \Sigma_2$ we just write $TS_1 \,||\, TS_2$ instead of $TS_1 \,||_H\, TS_2$
- If $H = \{\}$ then $TS_1 \,||_H\, TS_2$ is equivalent to $TS_1 \,|||\, TS_2$
- Handshaking is also called *synchronous message passing*

Variables, Conditions, and Assignments

in the last lecture...

- When modeling real-life systems, it is often convenient to
  - consider variables
  - consider guarded transitions
  - and side-effects of transitions on variables (assignments)
- Consider the following extended LTS. (We introduce these concepts in a by-example fashion.)
  - (we can also represent sequential programs this way)

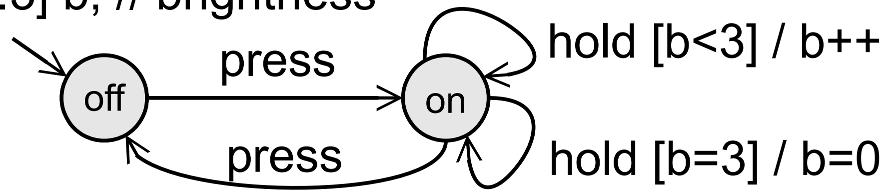


Unfolding

in the last lecture...

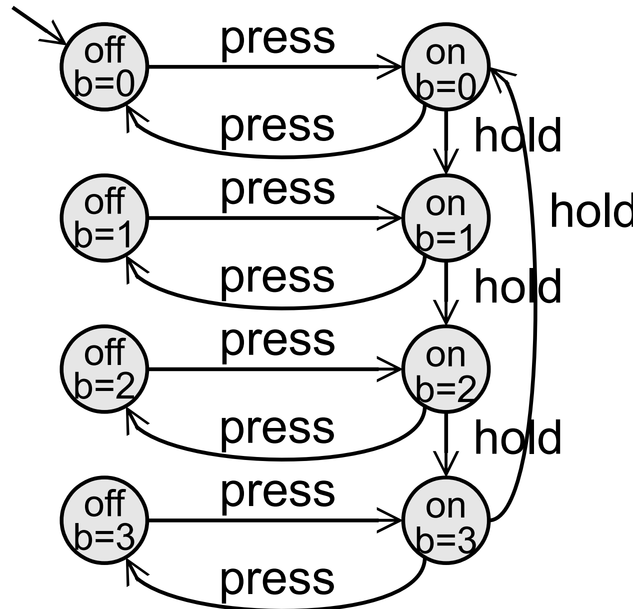
- An LTS with variables, guard conditions, and assignments can be transformed into a regular LTS via *unfolding*

var:
int[0..3] b; // brightness



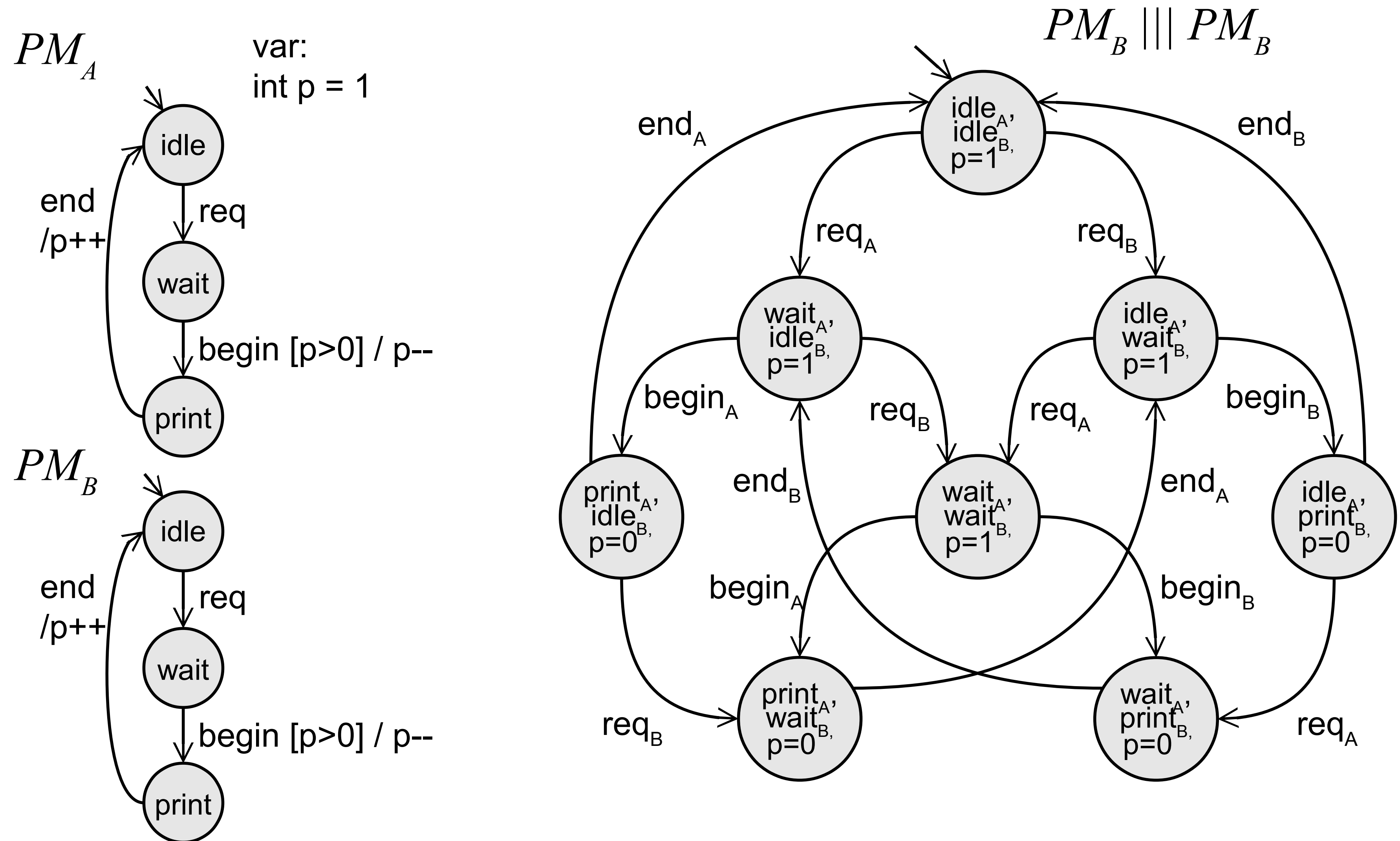
unfolded LTS without variables, guards and assignments:

("off b=2" is just the name of a state---not a variable!---to illustrate the correspondence)



Communication via Shared Variables

- Example: Printer Manager



SOFTWARE ENGINEERING

The LTSA Tool with Animator (for Simulation)

in the last lecture...

File Edit Check Build Window Help Options

LIGHTSWITCH

Edit Output Draw

LIGHT
SWITCH
||LIGHTSWITCH

LIGHTSWITCH

0 1 2 3 4

press
release
hold
press
release
release

Animator

press
hold
release
press
release
press
release

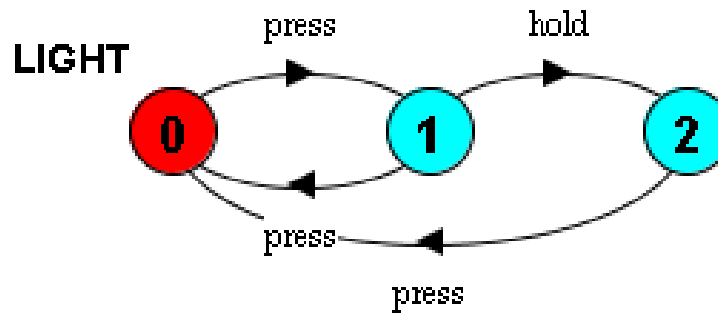
Run Step

- [x] press
- [ ] hold
- [ ] release

Labelled Transition System Analyser (LTSA)

in the last lecture...

- Modeling and verification tool for concurrent systems
- Modeling with *Finite State Processes (FSP)*
  - they can be transformed into Labeled Transition Systems (LTS)

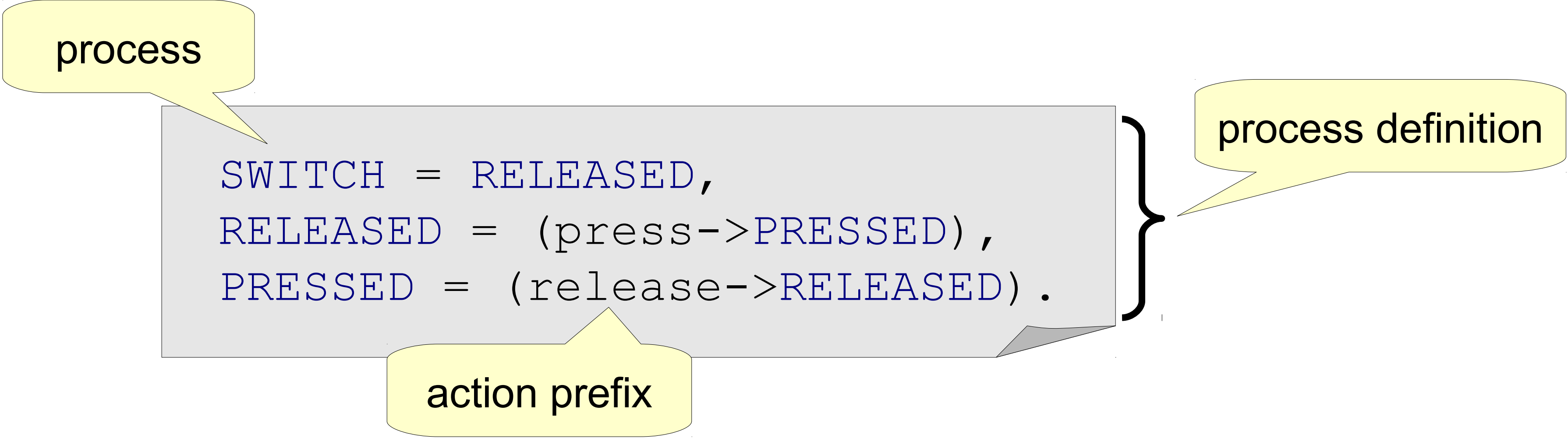


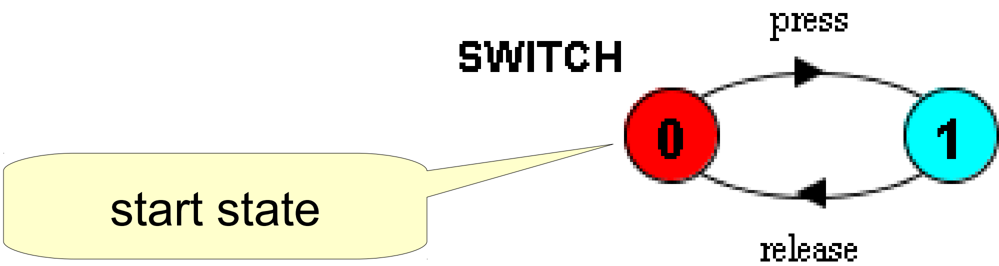
- Supports some rich language features
  - composition via handshaking “||”
  - shared variables
  - multiple “instances” of processes (process labeling)
- Supports deadlock detection and automated verification

Finite State Processes (FSP)

in the last lecture...

- The graphical notation for LTSs becomes unmanageable for big processes / transition systems
- Alternative: Textual, algebraic notation





Finite State Processes (FSP)

in the last lecture...

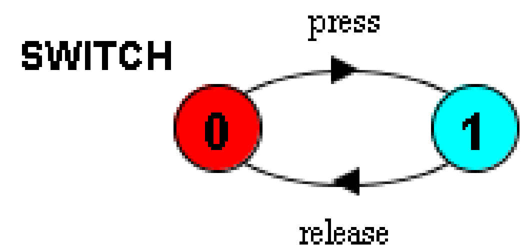
- The following process definitions are equivalent:

```
SWITCH = RELEASED,
RELEASED = (press->PRESSED),
PRESSED = (release->RELEASED).

SWITCH = RELEASED,
RELEASED = (press->(release->RELEASED)).

SWITCH = (press->(release->SWITCH)).
```

again the corresponding LTS:

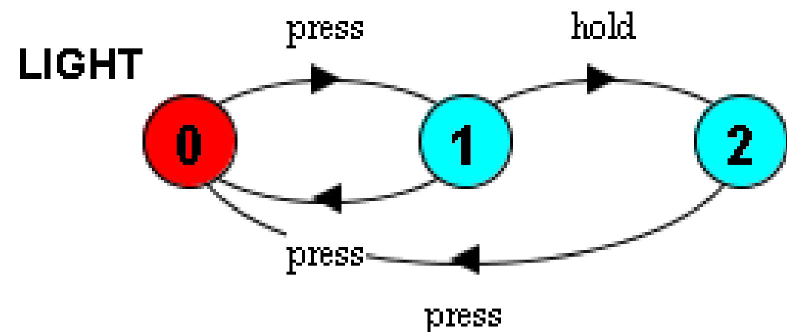


Choice

- "LIGHT_LOW engages either in the actions hold or press. Then behaves as LIGHT_HIGH or LIGHT_OFF, respectively"

```
LIGHT_LOW  = (hold->LIGHT_HIGH|press->LIGHT_OFF)
```

- Full Light example:

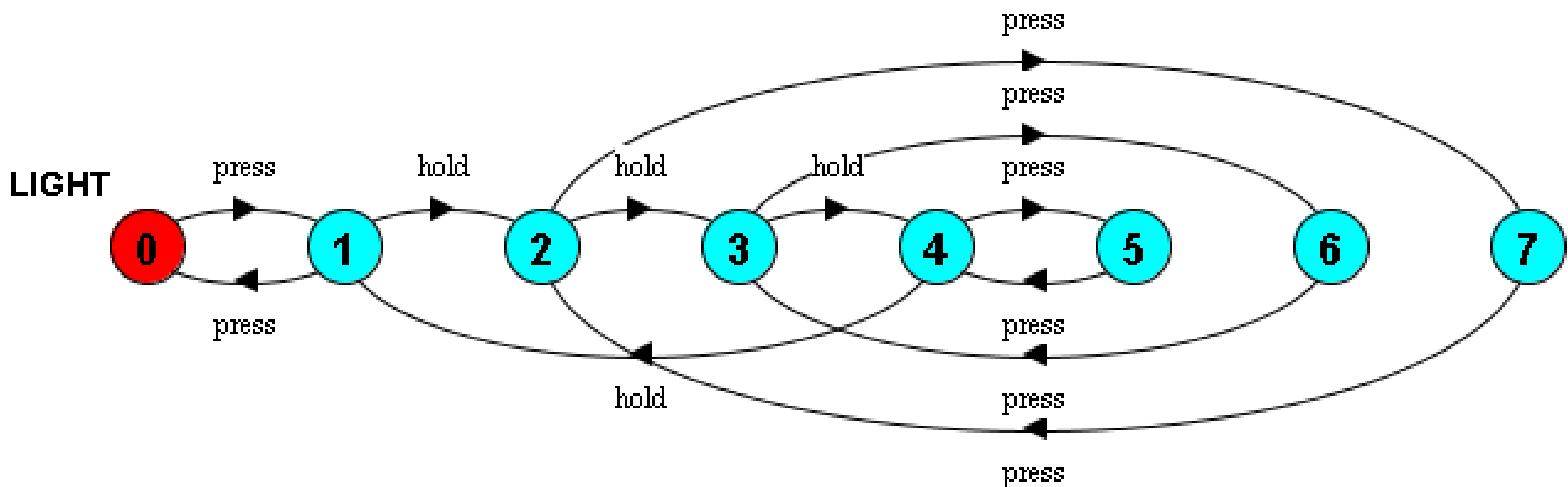



```
LIGHT  = LIGHT_OFF,
LIGHT_OFF  = (press->LIGHT_LOW),
LIGHT_LOW  = (hold->LIGHT_HIGH|press->LIGHT_OFF),
LIGHT_HIGH = (press->LIGHT_OFF).
```

Variables and Conditions

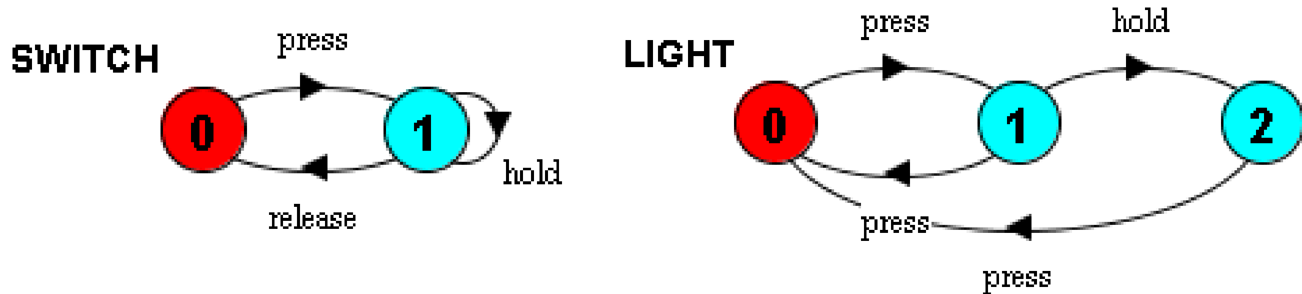
```
const N = 3
range Brightness = 0..N

LIGHT = OFF[0],
OFF[b:Brightness] = (press->ON[b]),
ON[b:Brightness] = (press->OFF[b]
        |when (b<N) hold->ON[b+1]
        |when (b==N) hold->ON[0]).
```

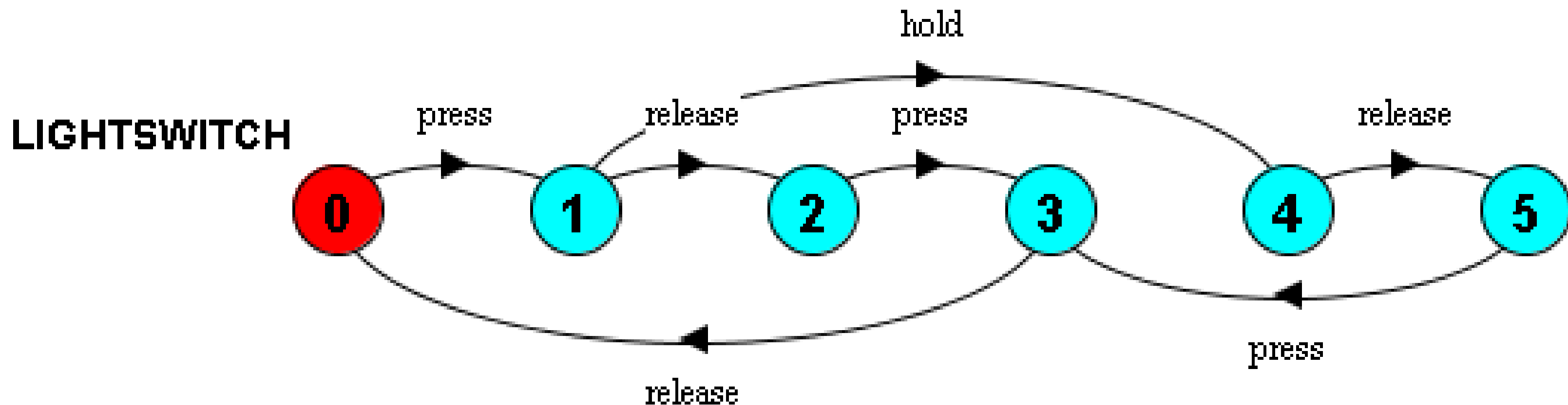


Parallel Composition

- Composite process definitions are preceded by “||”
- Shared actions must be executed at the same time by all processes that share the action
- Example:




```
||LIGHTSWITCH = (LIGHT || SWITCH).
```



The LTSA Tool with Animator (for Simulation)

Handshaking

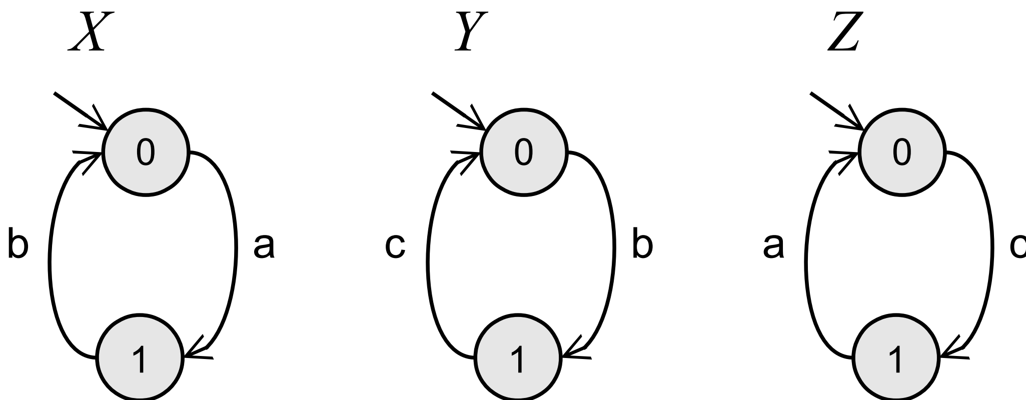
- Handshake is **associative** if transition systems synchronize over their common actions
  - $(TS_1 \,||\, TS_2) \,||\, TS_3 = TS_1 \,||\, (TS_2 \,||\, TS_3)$

- But handshaking is not generally associative
  - $(TS_1 \,||_H\, TS_2) \,||_{H'}\, TS_3 \neq TS_1 \,||_H\, (TS_2 \,||_{H'}\, TS_3)$ *for* $H' \neq H$

Handshaking

- Handshake is **associative** if transition systems synchronize over their common actions
  - $(TS_1 \,||\, TS_2) \,||\, TS_3 = TS_1 \,||\, (TS_2 \,||\, TS_3)$

- But handshaking is not generally associative
  - $(TS_1 \,||_H\, TS_2) \,||_{H'}\, TS_3 \neq TS_1 \,||_H\, (TS_2 \,||_{H'}\, TS_3)$ *for* $H' \neq H$

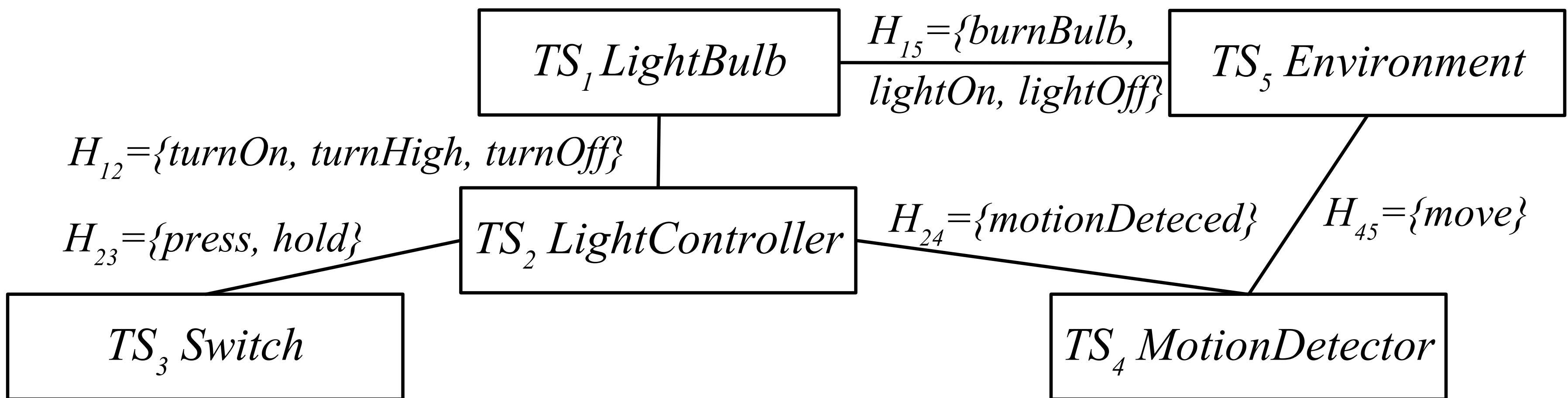


$$(X \,||_{\{b\}}\, Y) \,||_{\{c\}}\, Z \neq X \,||_{\{b\}}\, (Y \,||_{\{c\}}\, Z)$$

Pairwise Handshaking

- Sometimes we want to model processes that communicate in a pairwise fashion (have an "n-ary" composition operator)
  - $TS = TS_1 \,||\, TS_2 \,||\, \ldots \,||\, TS_n$
- TS_i and TS_j $(0 < i \neq j \leq n)$ must perform any transition labeled with an event from $H_{i,j} = \Sigma_i \cap \Sigma_j$ together
  - associative if $H_{i,j} \cap \Sigma_k = \{\}$ *for any* $k \notin \{i, j\}$

an event is shared by at most two LTSs

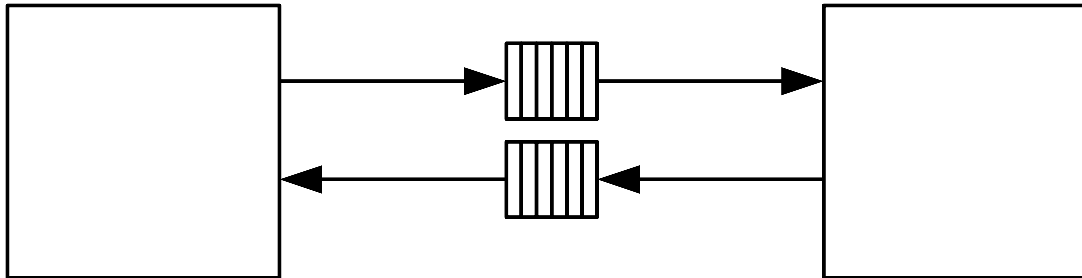


Switch || MotionDetector || LightController || LightBulb || Environment

SOFTWARE
SE
ENGINEERING

Messages and Channels

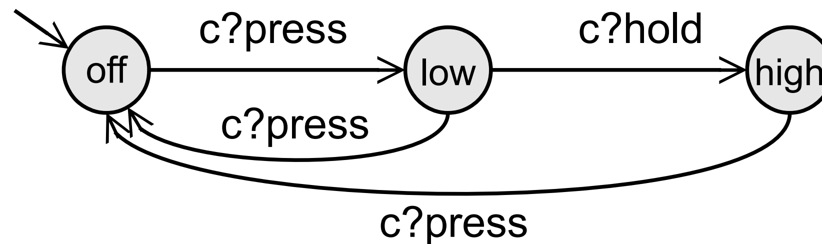
- Processes communicate by sending messages over channels
  - asynchronous: messages can be stored in a FIFO buffer (we will get back to these later)

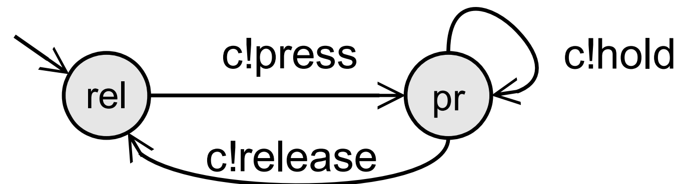


  - synchronous: messages are sent and received at the same time
    - similar to handshaking, but introduces a direction
    - buffer of size "zero"

Messages and Channels

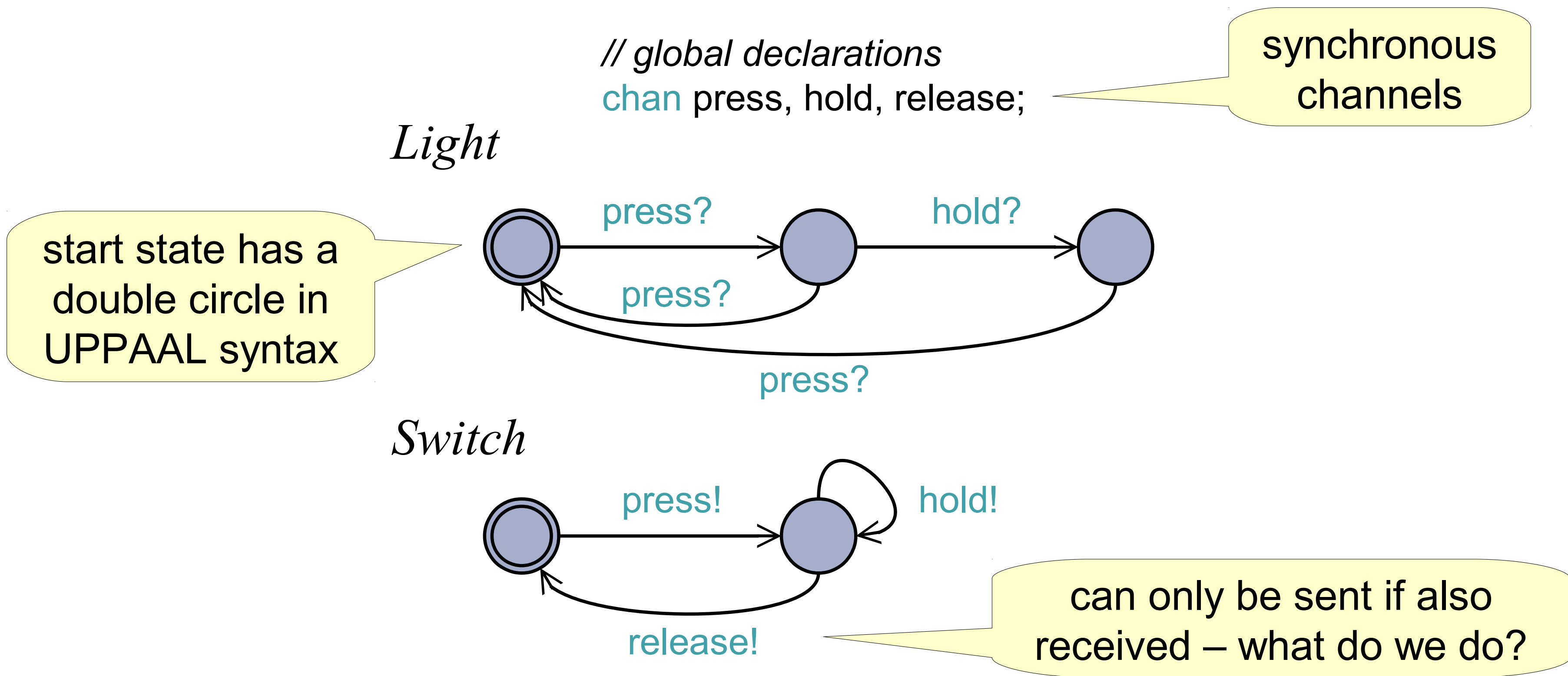
- “c!press” means that “press” is sent over channel “c”
- “c?press” means that “press” is received over channel “c”





Messages and Channels

- Some tools have simplified concepts of channels and events
- In UPPAAL, the channel name is the event name
  - Example: The light switch modeled in UPPAAL



SOFTWARE
SE
ENGINEERING

UPPAAL Tool

UPPAAL website: http://www.uppaal.org/

C:/e362/ws-tex/Formal-Methods-in-Software-Engineering/WS14-15/lecture/03/examples/light-switch.xml - UPPAAL

File Edit View Tools Options Help

Editor Simulator Verifier

Drag out

Enabled Transitions

Switch1

Next Reset

Simulation Trace

(-, -)
press: Switch1 --> Light1
(-, -)
hold: Switch1 --> Light1
(-, -)
Switch1
(-, -)
press: Switch1 --> Light1
(-, -)

Trace File:

Prev Next Replay
Open Save Auto

Slow Fast

Drag out

Light1
press?
hold?
press?
press?

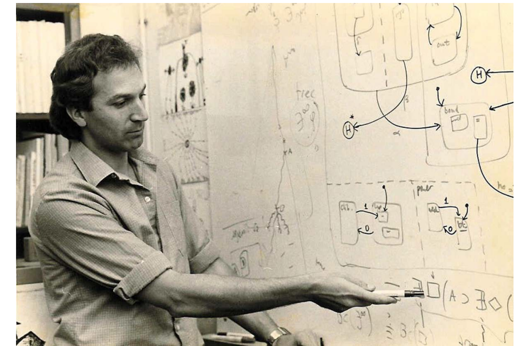
Switch1
press!
hold!

Light1 Switch1
press
hold
press

SOFTWARE
SE
ENGINEERING

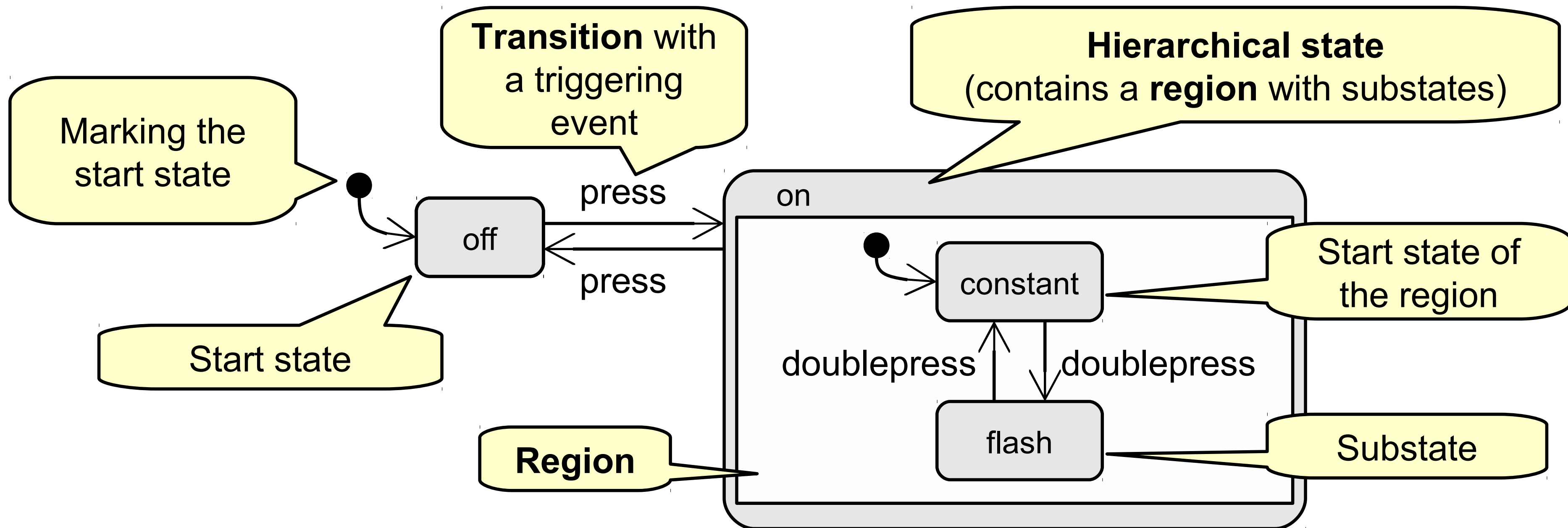
Statecharts

- Statecharts are an extension of finite state machines
  - Extended with variables, conditions, assignments, hierarchical and orthogonal states, ...
  - invented by David Harel in 1984
    - Harel, D. Statecharts: a visual formalism for complex systems, *Science of Computer Programming*, 1987, 8, 231-274 http://www.wisdom.weizmann.ac.il/~harel/papers/Statecharts.pdf
    - see also http://dl.acm.org/citation.cfm?id=1238849



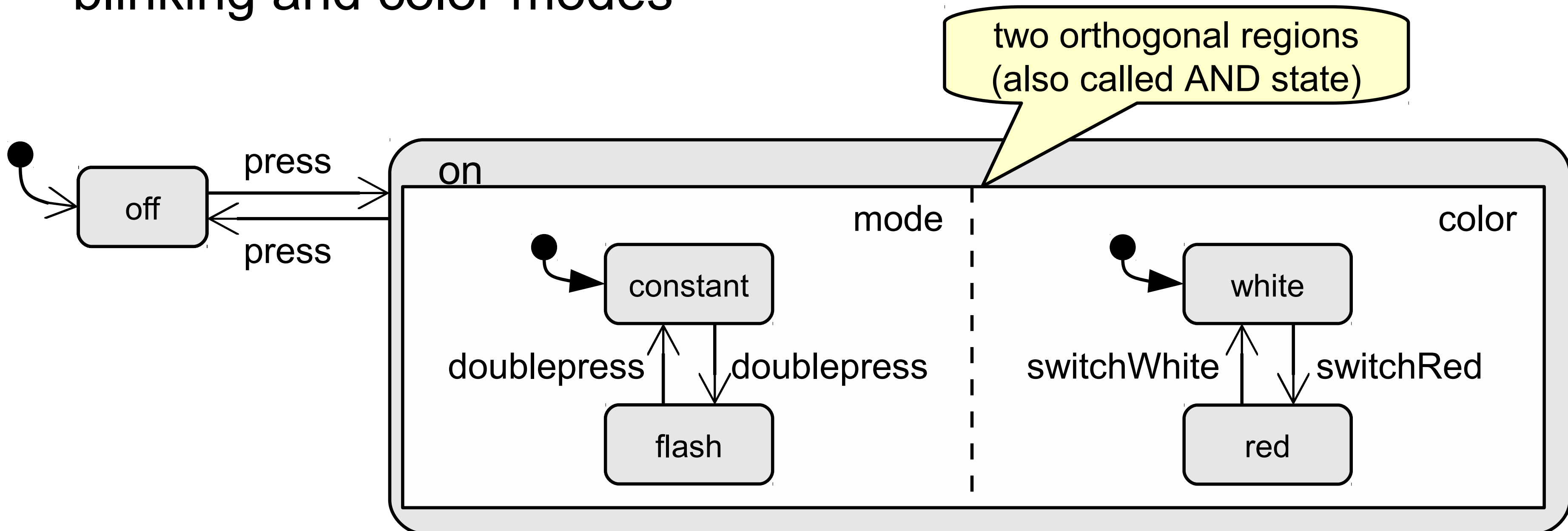
- Statecharts have later been adopted by the UML

Another Flashlight Example



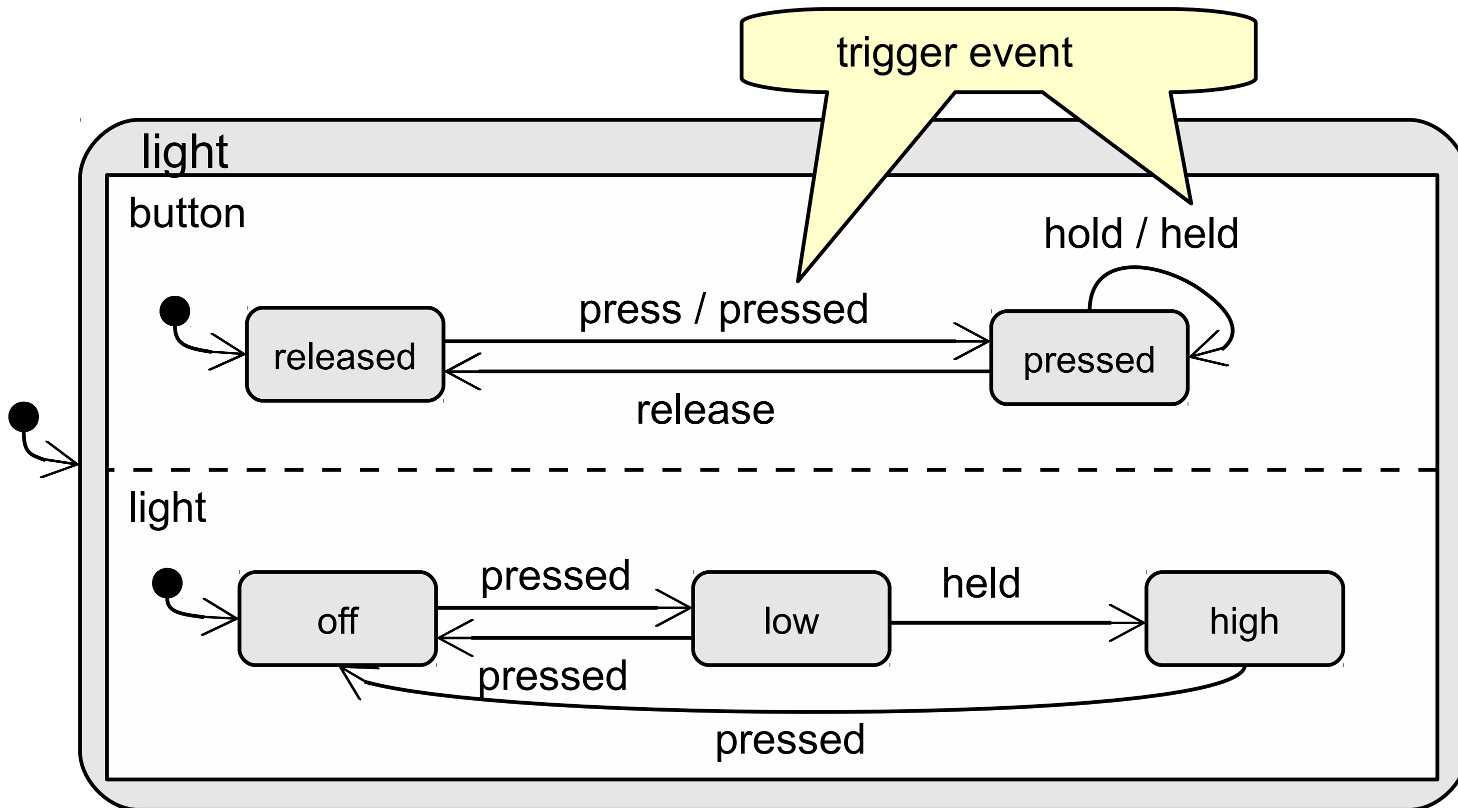
A Statechart with Orthogonal Regions

- Orthogonal regions progress concurrently
  - a system in the superstate is in one state of each contained orthogonal region
  - used to model independent active behavior in a superstate
- Simple example: Lamp that is on can independently switch blinking and color modes



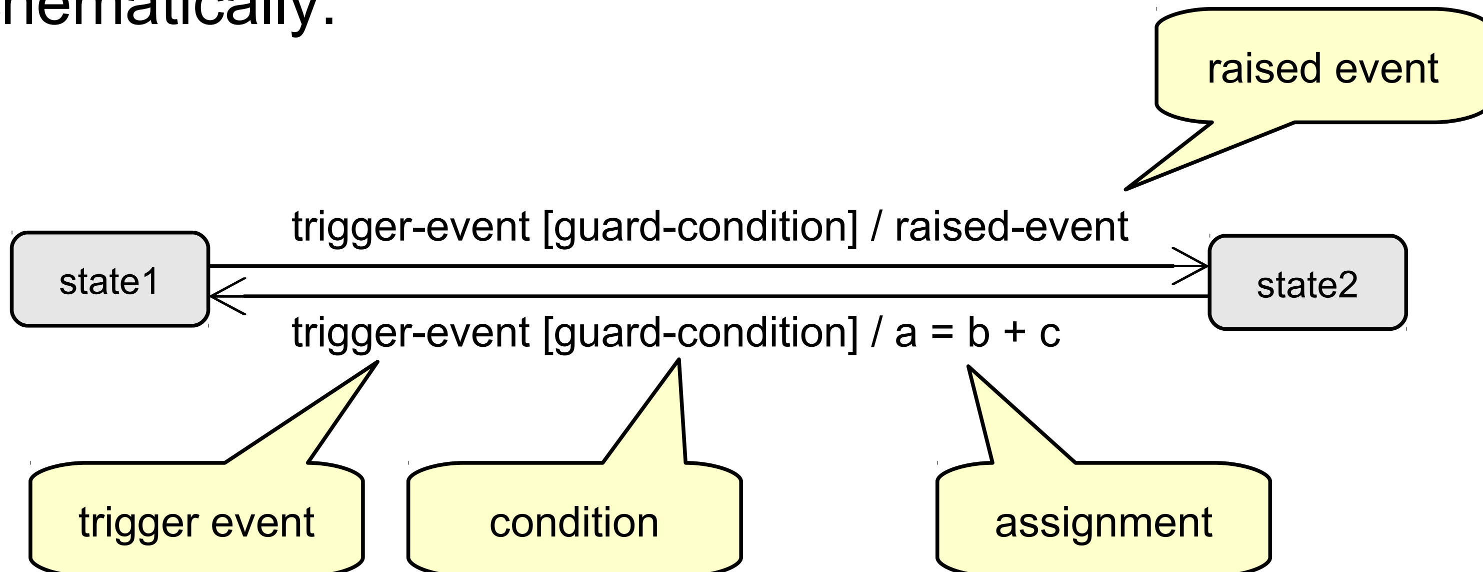
A Statechart with Orthogonal Regions

- Transitions can raise events
- Events can trigger transitions in orthogonal regions
  - Enabled transitions triggered by an event fire synchronously with enabled transitions raising the same event



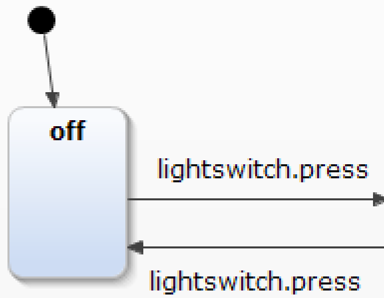
Transition Labels – Summary

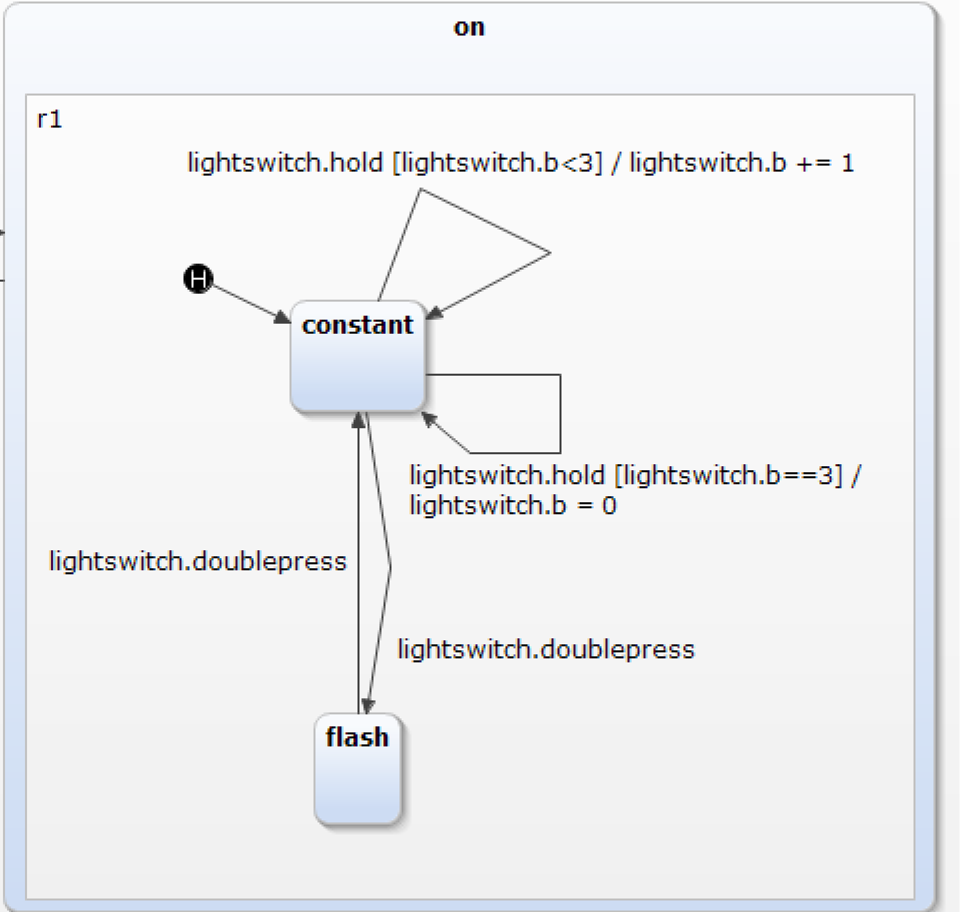
- Transitions in Statecharts
  - can have a trigger event
  - can have a guard condition
  - can raise a event
  - can have further side-effects
    - e.g. assign a new value to a variable, call a method
- Schematically:



Try it out: Yakindu Statechart Tools

```
lightswitch
interface lightswitch:
in event press
in event hold
in event doublepress
var b: integer
```





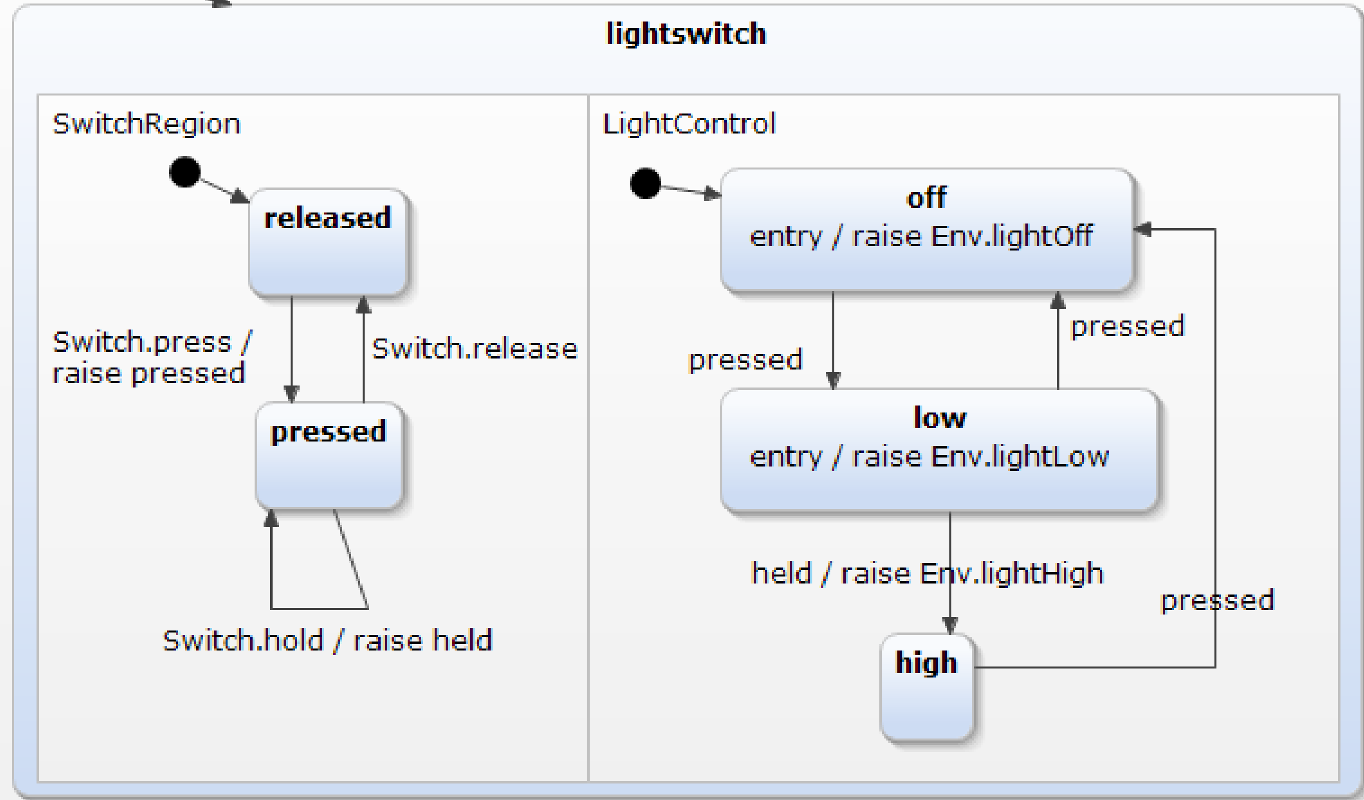
(Screenshot from Yakindu Statechart Tools http://statecharts.org/)

Another example with orthogonal states

```
light
interface Switch:
in event press
in event release
in event hold

interface Env:
out event lightOff
out event lightLow
out event lightHigh

internal:
event pressed
event held
```



Another example with orthogonal states

Interfaces: Events that can be received from or sent to external objects

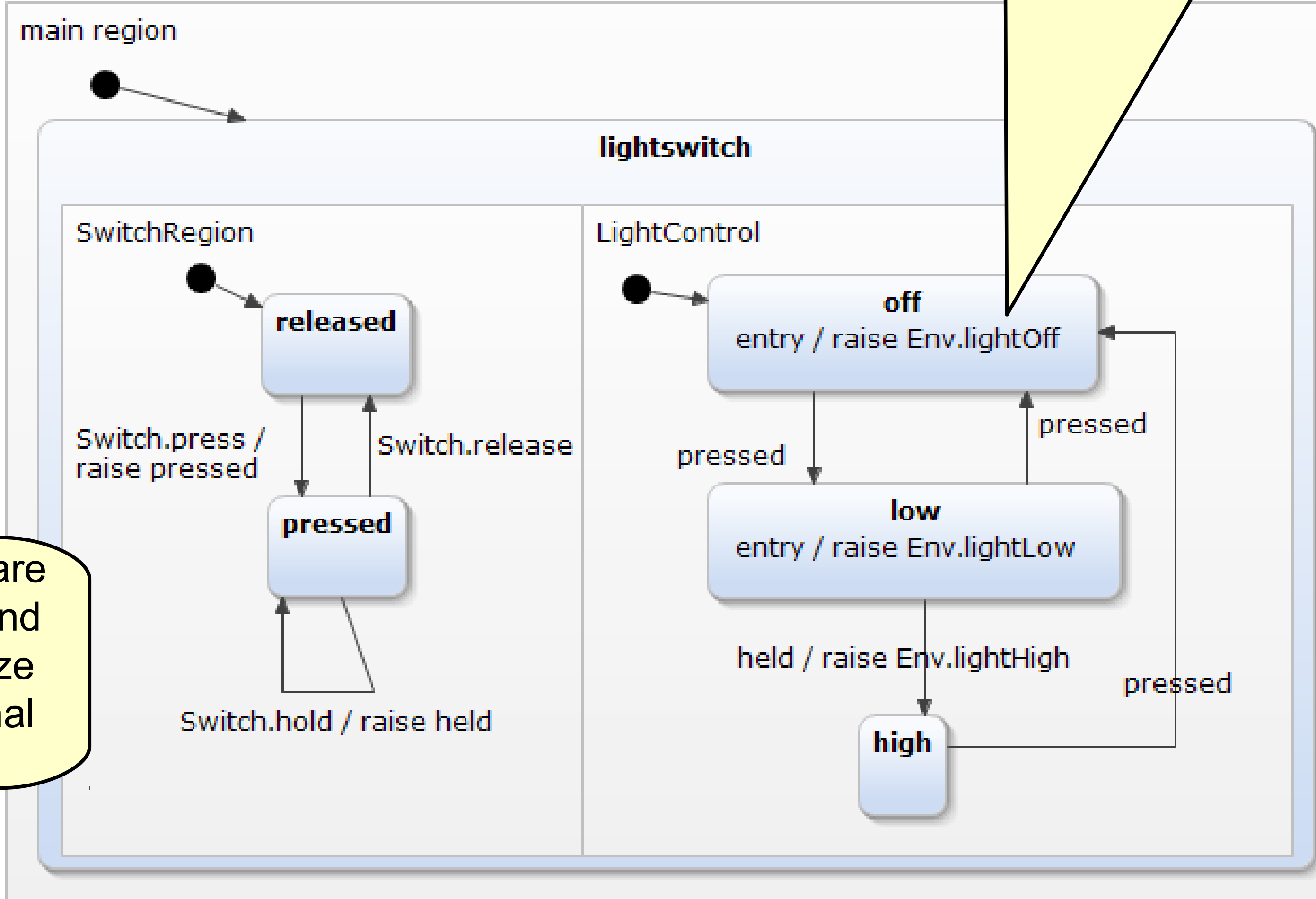
entry-action: events can be raised or assignments/methods can be executed when entering a state.

light
interface Switch:
in event press
in event release
in event hold

interface Env:
out event lightOff
out event lightLow
out event lightHigh

internal:
event pressed
event held

internal events: they are not externally visible and are used to synchronize transitions in orthogonal states



SOFTWARE ENGINEERING

Learning Objective

- Understand fundamental modeling concepts and languages for concurrent reactive systems
- Get to know some tools that support modeling and analysis based on these fundamental modeling languages
- Understand that higher-level modeling languages can be mapped to fundamental modeling languages
  - unfolding, composition

- Next time: Modeling Methodology Basics and

Assignment

- Model the light and switch example with the three brightness levels (using variables, conditions, and assignments)
  1. with LTSA (see the example in the slides above)
  2. with UPPAAL
  3. with Yakindu Statechart Tools
  - Model the light and switch using concurrent LTSs (LTSA) resp. automata (UPPAAL) resp. orthogonal states (Statecharts)
  - 2 and 3 can be presented in the tutorials
- Read:
  - Zave, P. & Jackson, M. Four dark corners of requirements engineering, ACM Trans. Softw. Eng. Methodol., ACM, 1997, 6, 1-30 (http://dl.acm.org/citation.cfm?id=237434)
    - and try to understand the relationship of requirements, (software) specification, and domain knowledge