DATA 61

Modelling and Verification of Protocols for Wireless Networks

(Lecture 4)

Peter Höfner

(Lecture at University of Twente, Jan/Feb 2017)

www.data61.csiro.au

UNSW AUSTRALIA

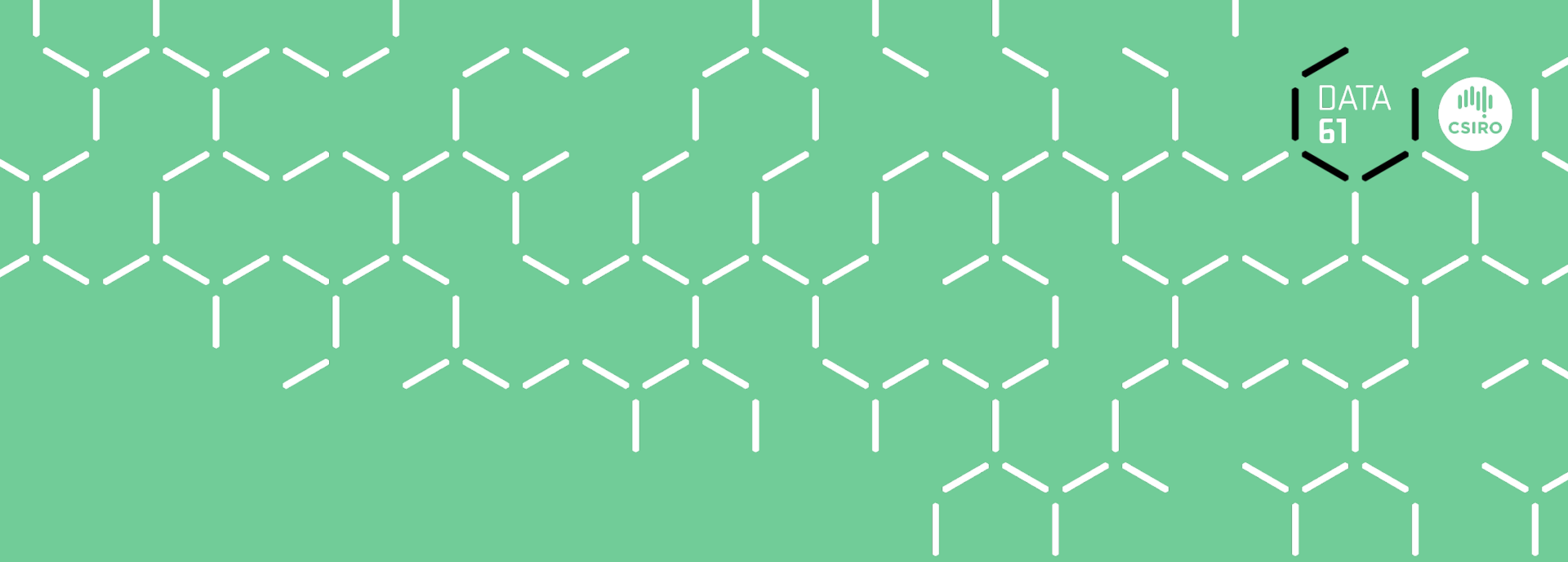
CSIRO

last update: Jan 25, 2017

Contents of this Lecture

What should you have learnt

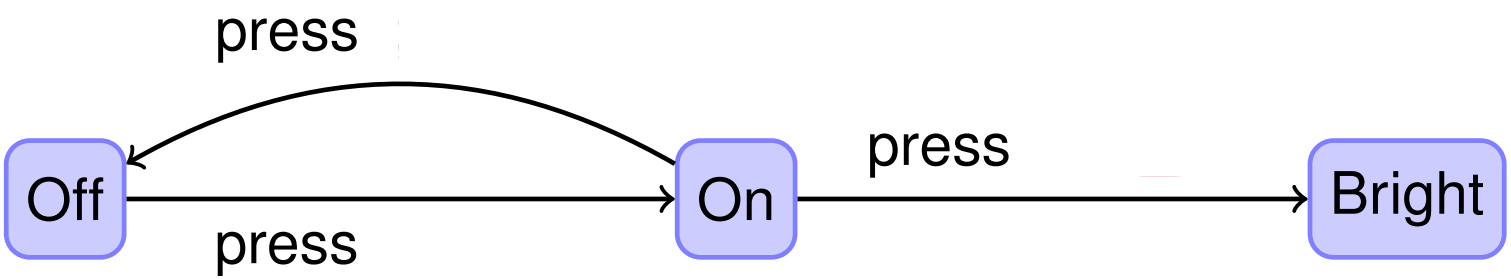
- Timed Automata
- Uppaal
- AWN 2 Uppaal
- Uppaal for Wireless Protocols



Timed Automata

Light Control System

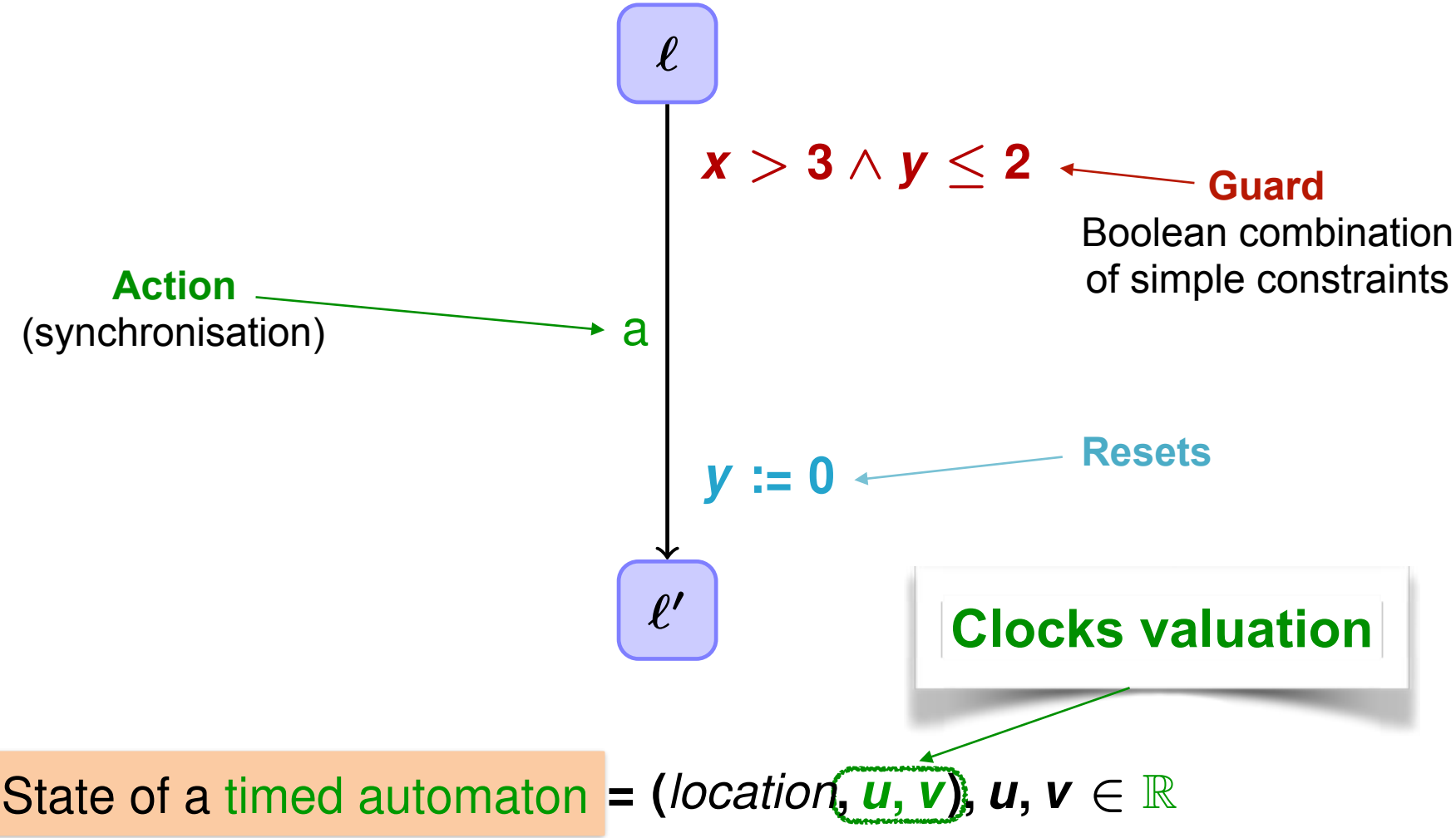
- Timed Automata = Finite Automata + (Real-Time) Clocks



if press button pressed twice quickly, the light gets brighter

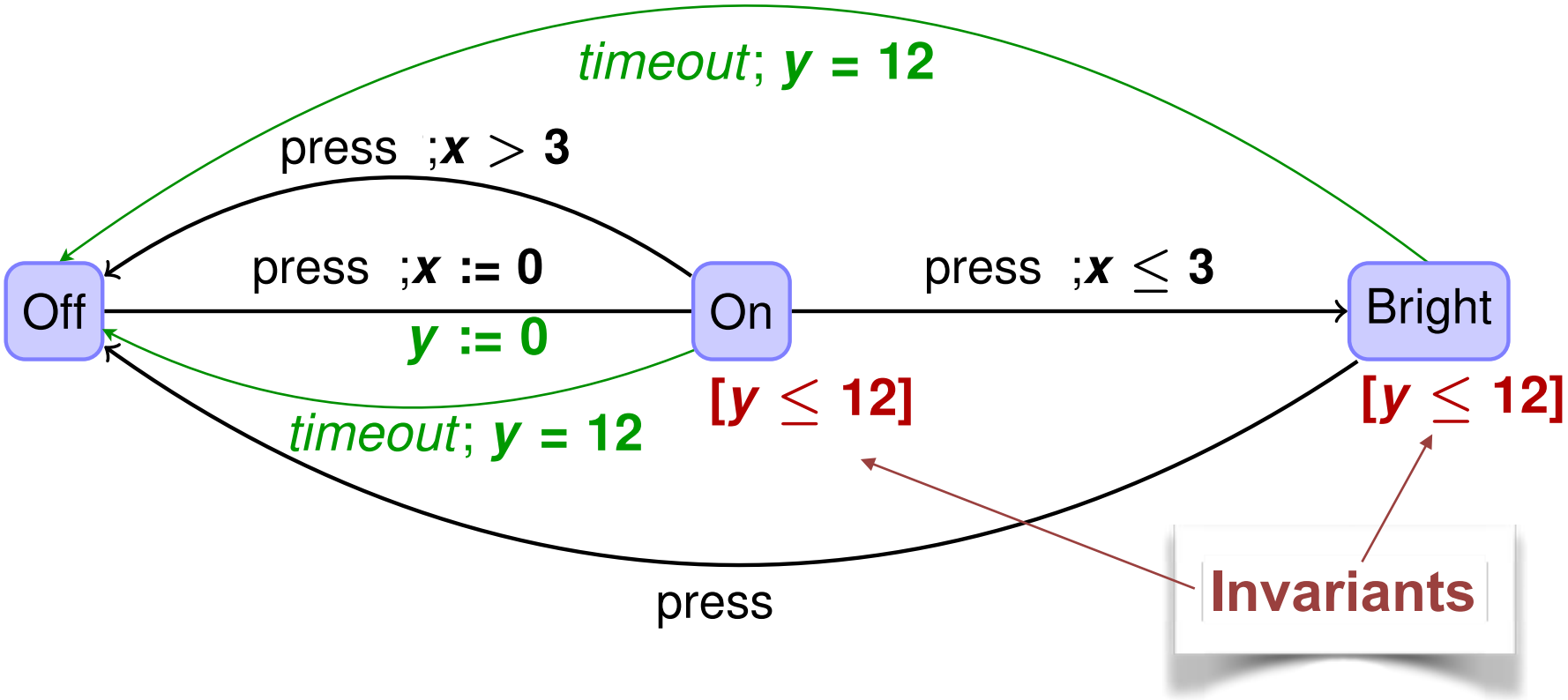
Add a real-valued clock

Guarded Transitions



State of a timed automaton = (*location*, ***u***, ***v***), ***u***, ***v*** $\in \mathbb{R}$

Light Control System – Version 2



- if press button pressed twice quickly, the light gets brighter
- Light should go off after **12** time units

Clock Constraints

Clock Constraints

$$\varphi ::= x \sim k \mid x - y \sim k \mid \varphi \wedge \varphi$$

where $x \in X$ and $k \in \mathbb{Z}$ and $\sim \in \{<, \leq, =, \geq, >\}$.

Examples

$x < 2$

$x - y = 0 \wedge z - x > 4$

$x < 2 \wedge y - x \leq 4$

$x - y = 0 \wedge z - x = -2$

Timed Automaton

Timed Automaton $\mathcal{A} = (L, \ell_0, \mathbf{Act}, X, \mathbf{Inv}, \longrightarrow)$

- L is a finite set of locations and ℓ_0 is the initial location
- X is a finite set of clocks
- **Act** is a finite set of actions
- $\longrightarrow$ is a set of edges of the form $\ell \xrightarrow{g, a, R} \ell'$ with:
  - $a \in$ **Act**
  - a guard g which is a clock constraint over X
  - a reset set R which is the set of clocks to be reset to **0**
- **Inv** associates with each location ℓ an invariant **Inv**(ℓ)

Semantics

Notations

$v : V \to \mathbb{R}^X_{\geq 0}$

$v \models g$: g is satisfied by v
$v + t$: $(v + t)(x) = v(x) + t$
$v[r := 0]$: $v[r := 0](x) = v(x)$ if $x \notin r$ and 0 otherwise

discrete step:

$$(\ell, v) \xrightarrow{a} (\ell', v') \iff \begin{cases} \exists\, \ell \xrightarrow{g,a,r} \ell' \in \mathcal{A} \\ v \models g \\ v' = v[r := 0] \\ v' \models \mathsf{Inv}(\ell') \end{cases}$$

Time Step:

$$(\ell, v) \xrightarrow{d} (\ell, v + d) \iff d \in \mathbb{R}_{\geq 0} \wedge v + d \models \mathsf{Inv}(\ell)$$

States

States: $(\ell, \boldsymbol{v}) \in \boldsymbol{L} \times \mathbb{R}^{\boldsymbol{X}}_{\geq \mathbf{0}}$

Initial state: $(\ell_0, \bar{\mathbf{0}})$

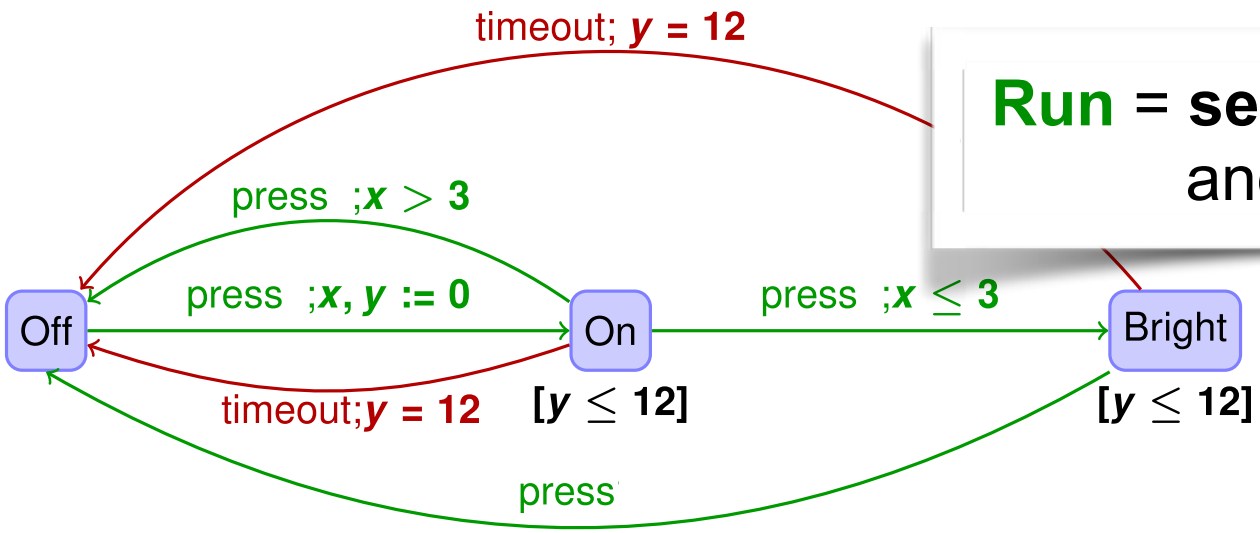
Set of Transitions = discrete + delay

Semantics of timed automaton $\mathcal{A}$

$$\boldsymbol{T} = (\boldsymbol{L} \times \mathbb{R}^{\boldsymbol{X}}_{\geq \mathbf{0}}, (\ell_0, \bar{\mathbf{0}}), \mathbf{Act} \cup \mathbb{R}_{\geq 0}, \xrightarrow{\mathbf{Act}} \cup \xrightarrow{\mathbb{R}_{\geq 0}})$$

infinite transition system

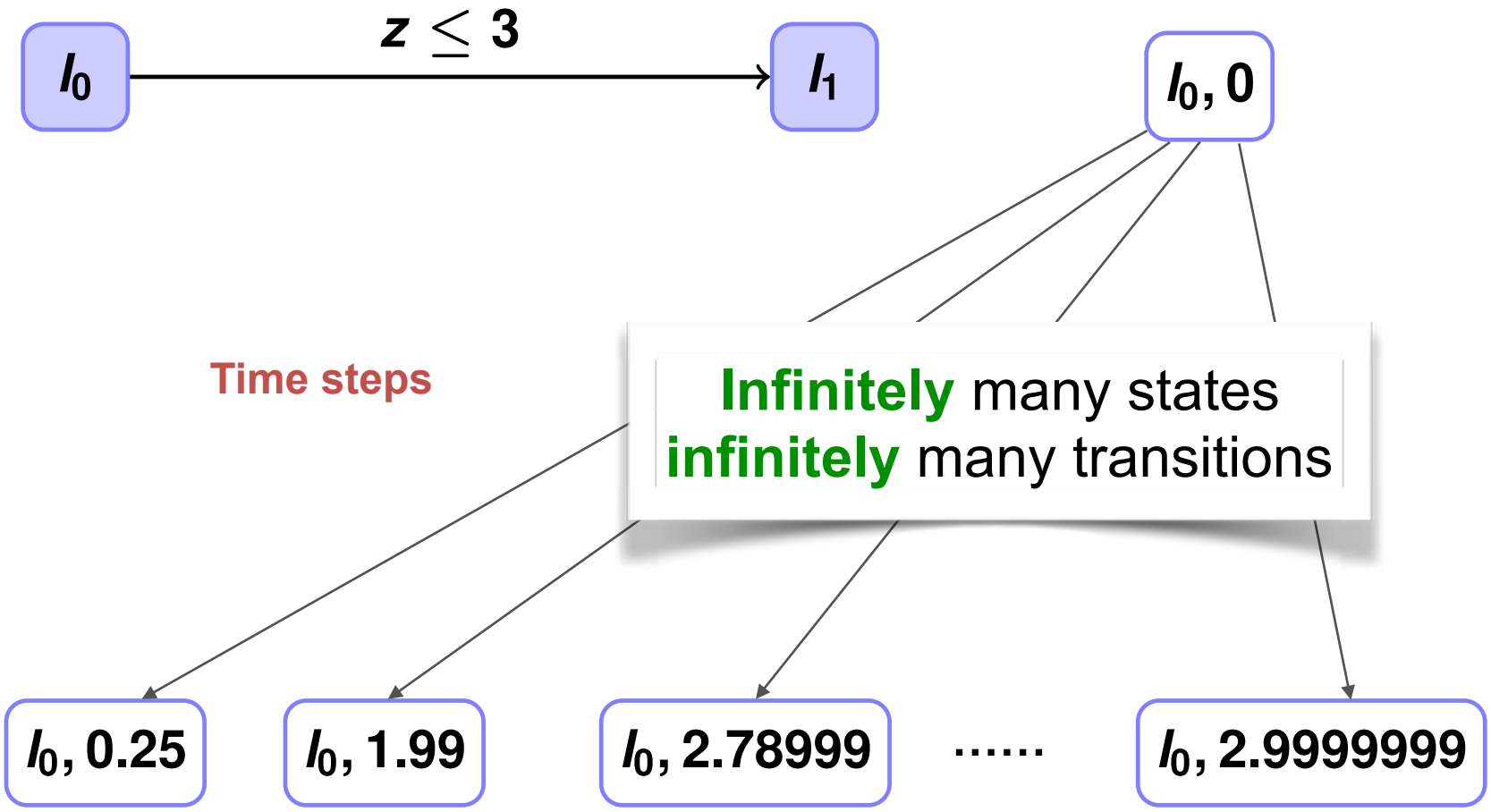
Runs



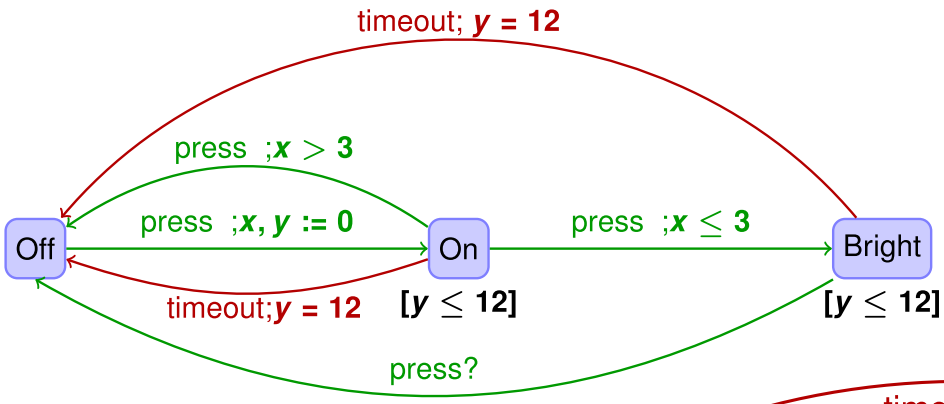
Run = **sequence** of discrete and time steps

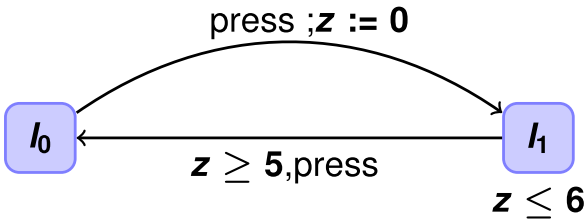
$$(\text{Off}, x = y = 0) \xrightarrow{265+\pi^2} (\text{Off}, x = y = 265 + \pi^2) \xrightarrow{press} (\text{On}, x = y = 0)$$
$$\xrightarrow{3.2} (\text{On}, x = y = 3.2) \xrightarrow{press} (\text{Off}, x = y = 3.2) \xrightarrow{27.87} (\text{Off}, x = y = 31.07) \quad \cdots$$

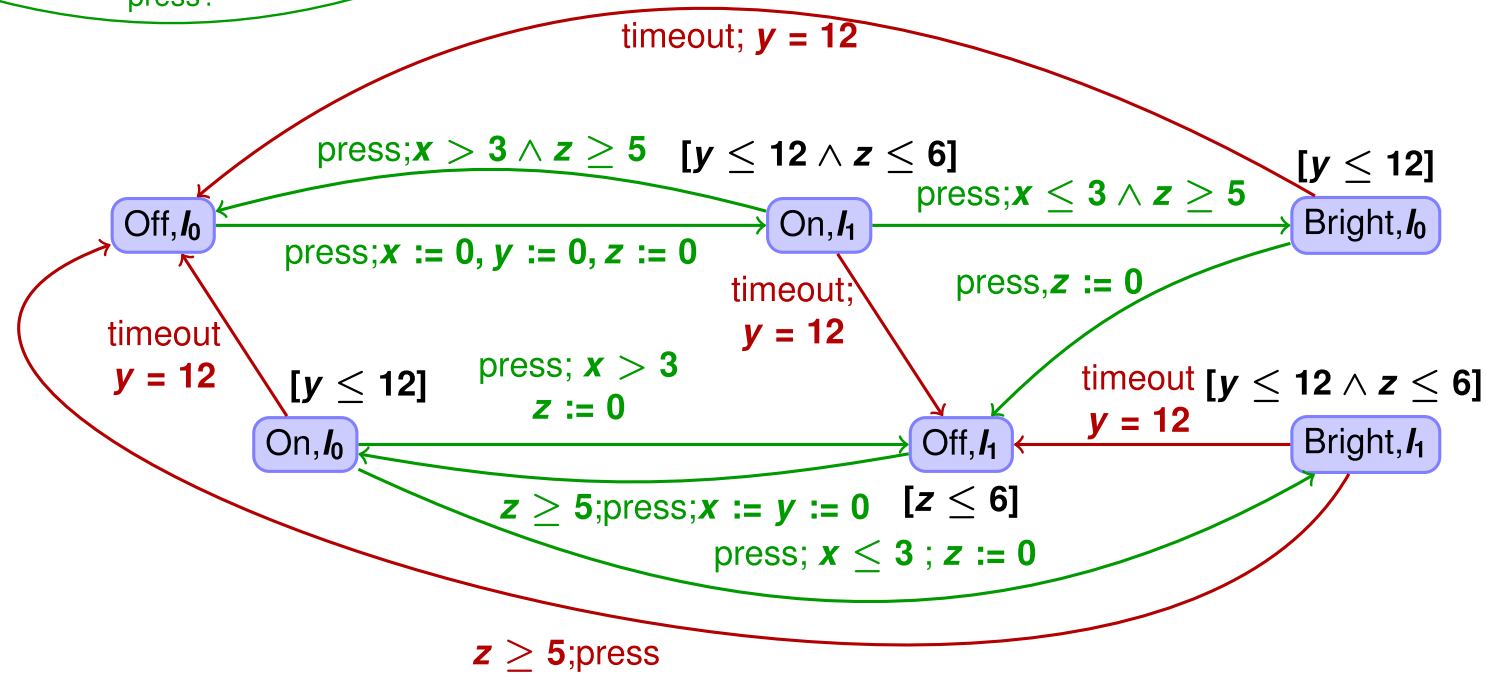
Reachability in Timed Automata



Product of Timed Automata







Product (Formal Definition)

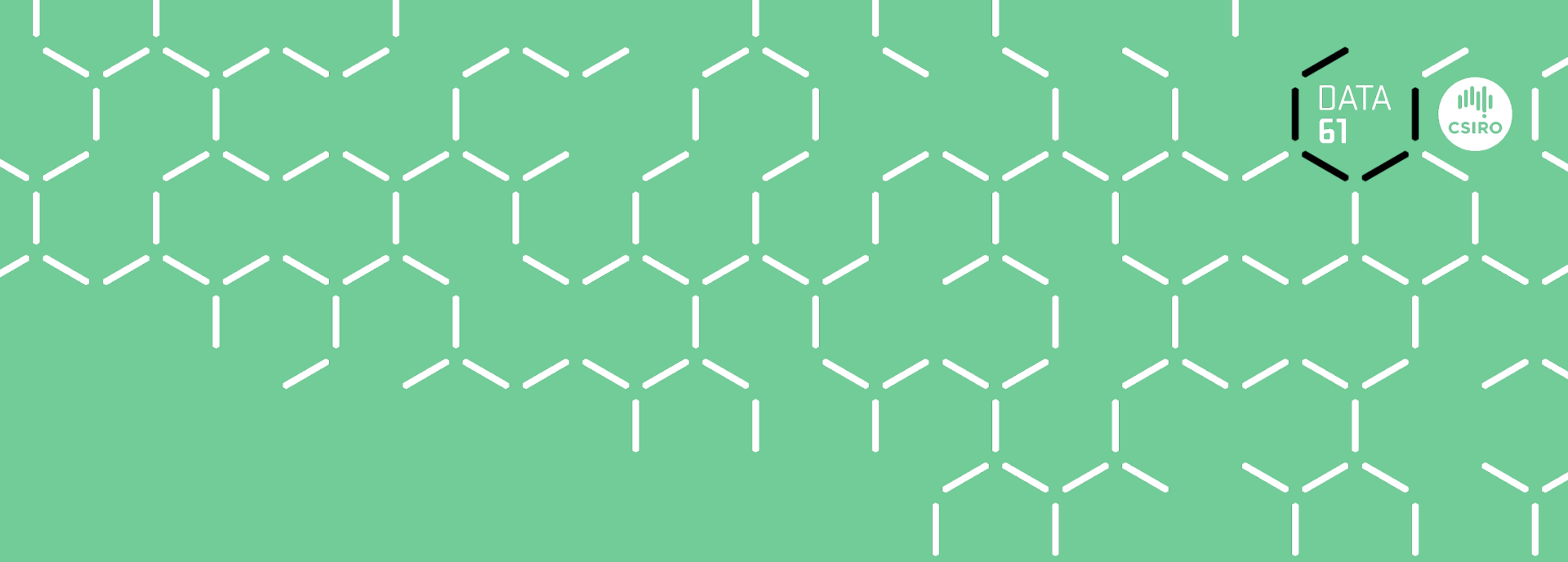
The synchronized product of A_1 and A_2 is the timed automaton $A_1 \times A_2 = (L, \ell_0, \mathbf{Act}, X, \mathbf{Inv}, \rightarrow)$ defined by:

- $L = L_1 \times L_2$ and $\ell_0 = (\ell_0^1, \ell_0^2)$,
- $\mathbf{Act} = \mathbf{Act}_1 \cup \mathbf{Act}_2$,
- $X = X_1 \cup X_2$,
- $\mathbf{Inv}((l_1, l_2)) = \mathbf{Inv}_1(l_1) \wedge \mathbf{Inv}_2(l_2)$,
- $\rightarrow$ is defined by:
  - $(l_1, l_2) \xrightarrow{g_1 \wedge g_2, a, R_1 \cup R_2} (l_1', l_2')$ iff $a \in \mathbf{Act}_1 \cap \mathbf{Act}_2$ and $l_i \xrightarrow{(g_i, a, R_i)}_i l_i'$, $i \in \{1, 2\}$
  - $(l_1, l_2) \xrightarrow{g, a, R} (l_1', l_2')$ iff $a \in \mathbf{Act}_i \setminus \mathbf{Act}_{3-i}$ and $l_i \xrightarrow{(g, a, R)} l_i'$ for some $i \in \{1, 2\}$ and $l_{3-i}' = l_{3-i}$.

Syntactic Product

Challenges in Model Checking TAs

- state-space explosion
  - as for discrete event systems
- state-space w.r.t. time
  - abstract to region of time



Uppaal
(model checker for timed automata)

Introduction

- *UPPAAL* is a toolbox for modelling, simulation and verification of timed systems
  - Uppsala Univ. + Aalborg Univ. = UPPAAL
- *UPPAAL Model*
  - network of timed automata
  - *enriched with data structure*
    – integers, structures, arrays ...
  - enriched with synchronisation
    – handshake, broadcast
  - enriched with urgency
- *UPPAAL's verification language*
  - subset of CTL (computation tree logic)

UPPAAL: Some History

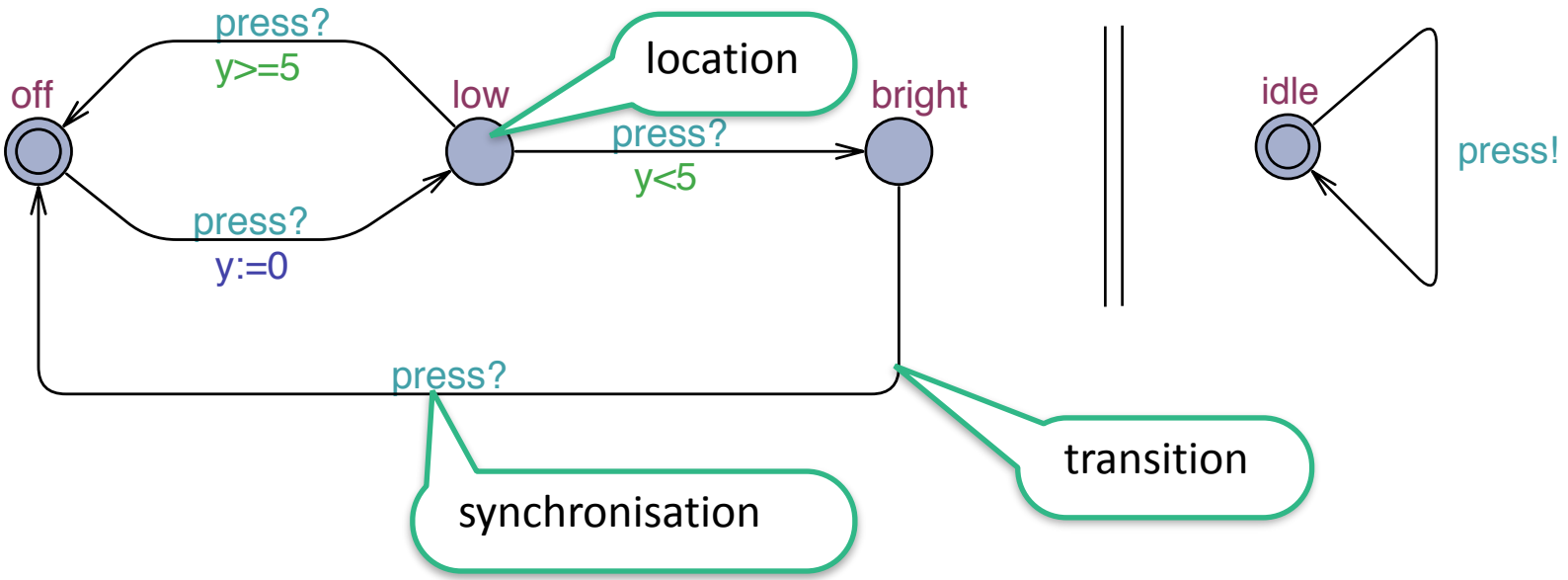
- Uppaal
  - 1995: first version
  - 2013: CAV award
    "the foremost tool suite for the automated analysis and verification of real-time systems"

- consists of
  - a description language
  - a simulator and a model checker
  - a graphical interface and command line

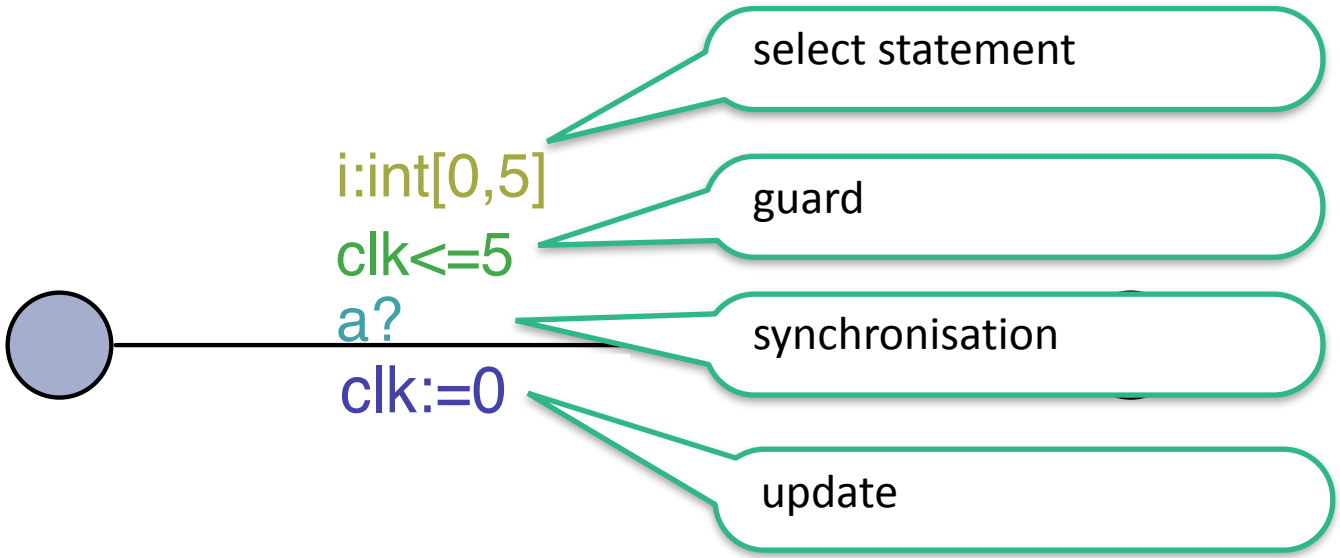
- available for academics (free) and industry
  http://www.uppaal.org

Network of Timed Automata

- demo: a switch

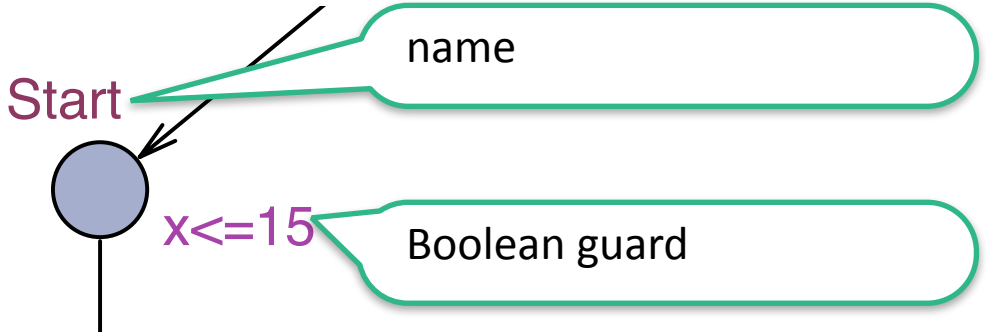


Transitions



Location Invariants

- Boolean guards are used as transitions



- only in state **Start** if valuation
- possibility of deadlocks

Data Structure

- C-like syntax (and semantics)
  - (bounded) integers, constants, arrays
  - structs
  - expressions
  - functions
  - local/global definitions
  - …
- Clocks
  - tests on clocks are not allowed
  - reset of clocks are allowed

Synchronisation

- *handshake synchronisation*
  - 1-1 communication

- *broadcast synchronisation*
  - 1-n communication

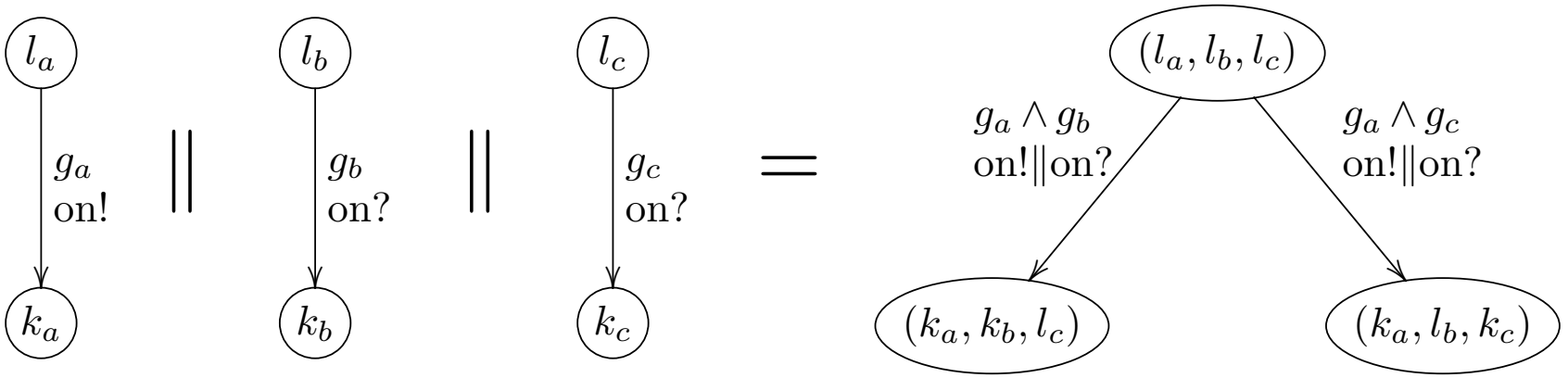
- synchronisation via channels
- exchange of data via shared variables

Handshake Synchronisation

- declaration

  `chan` a, b, c[3];

- synchronisation on **!**-**?** pairs
  - both guards have to be satisfied
  - if multiple pairs possible, choose non-deterministically

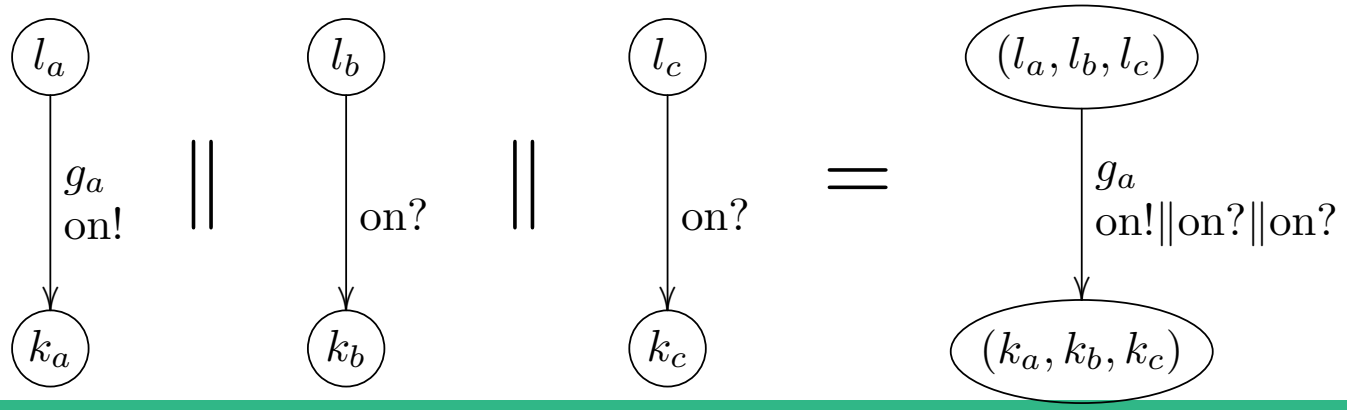


Broadcast Synchronisation

- declaration

  `broadcast chan` a, b, c[3];

- synchronisation from **!** to all **?** channels
  - guards only on transition labelled **!**
  - if multiple **!** enabled, choose non-deterministically



Simulator

- validation tool
  - enables examination of possible dynamic executions
  - provides an inexpensive mean of fault detection prior to verification
  - automatic and manual mode

Model-Checker

- check invariants and reachability properties
- explores the state-space of a system
- provides simulation path in case the property is not satisfied

More Features

- locations can be *urgent* and *committed*
  - urgent states do not have delay
  - committed states have to be left asap
  - committed states often used to create atomic sequences
- *channels can be urgent*
  - no time-delay if transition can be taken
- channel prioritisation
- probabilistic transitions
- …

Verification Properties

- E<> p there exists a path where p eventually holds
- A[] p for all paths p always holds
- E[] p there exists a path where p always holds
- A<> p for all paths will p eventually hold
- p --> q whenever p holds q will eventually hold

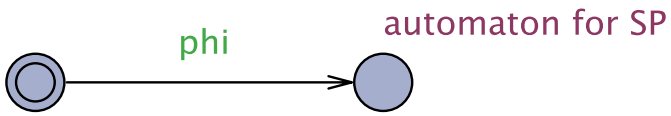
(p and q are state formulas)

AWN 2 Uppaal

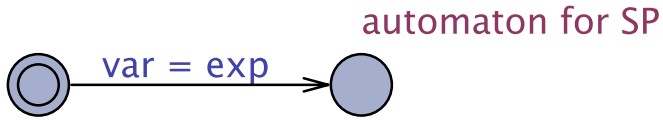
AWN 2 Uppaal

sequential processes (1)

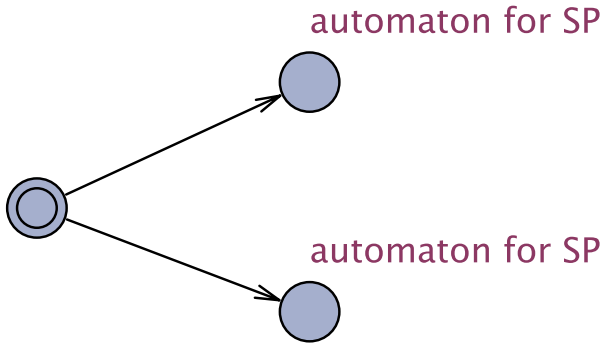
- most primitives straight forward, e.g.
  - $[\varphi]SP$



  - $[\![\texttt{var} := exp]\!]SP$



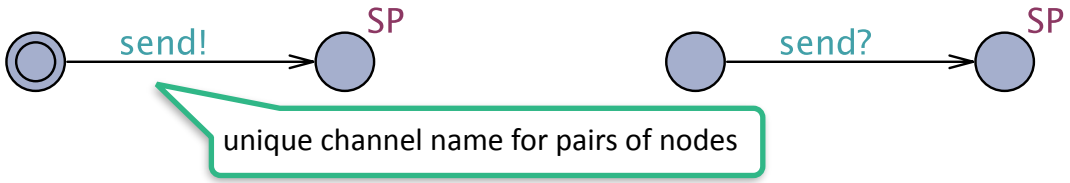
  - $SP + SP$



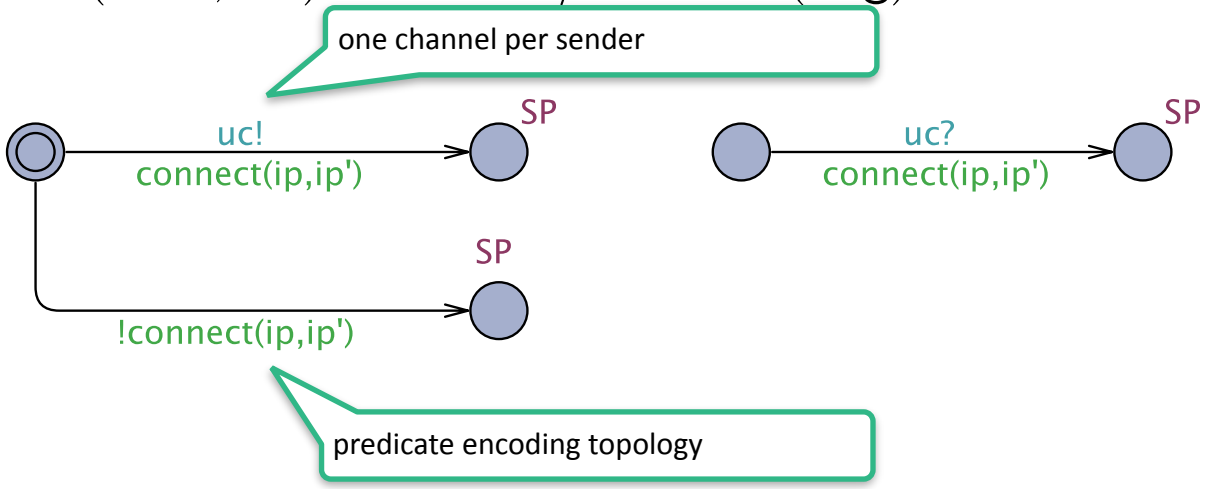
AWN 2 Uppaal

sequential processes (2)

- synchronisation (message is passed on via global variable)
  - $\mathbf{send}(ms)\ /\ \mathbf{receive}(\mathtt{msg})$:



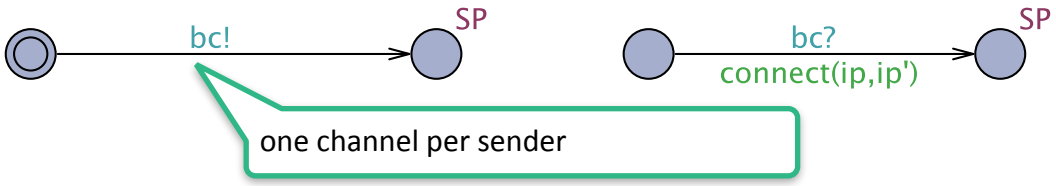
  - $\mathbf{unicast}(dest\,, ms).SP \blacktriangleright SP\ /\ \mathbf{receive}(\mathtt{msg})$:



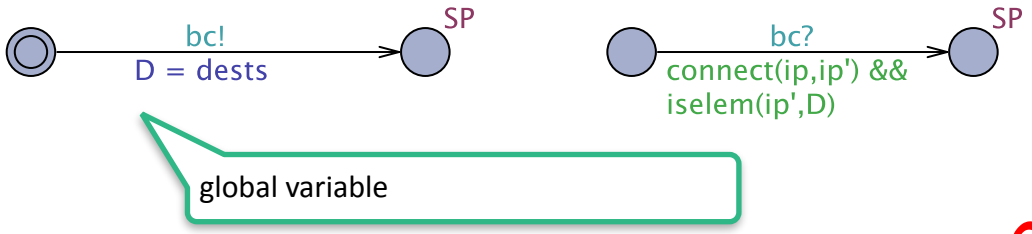
AWN 2 Uppaal

sequential processes (2)

- synchronisation (message is passed on via global variable)
  - $\mathbf{broadcast}(ms)$ / $\mathbf{receive}(\mathtt{msg})$:



  - $\mathbf{groupcast}(dests, ms)$ / $\mathbf{receive}(\mathtt{msg})$:



ONLY IF SYSTEM IS INPUT ENABLED

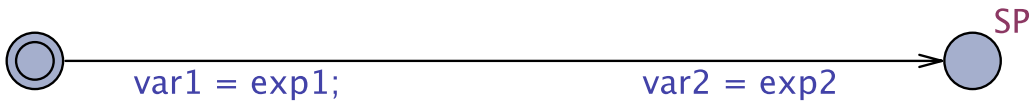
AWN 2 Uppaal

sequential processes (3)

- process calls
  - complicated if modelled as synchronisations
    - multiple copies
    - value passing
  - inline and self-reference is easier
    - only works for *guarded processes*

- optimisations
  - using committed states or “rewriting” makes state space smaller

$$[\![\texttt{var1} := exp1]\!][\![\texttt{var2} := exp2]\!]SP$$



AWN 2 Uppaal

parallel processes and networks & misc

- Uppaal offers only one parallel operator
  - different parallel operators can be encoded via channels
- injection of new packets require an additional automaton

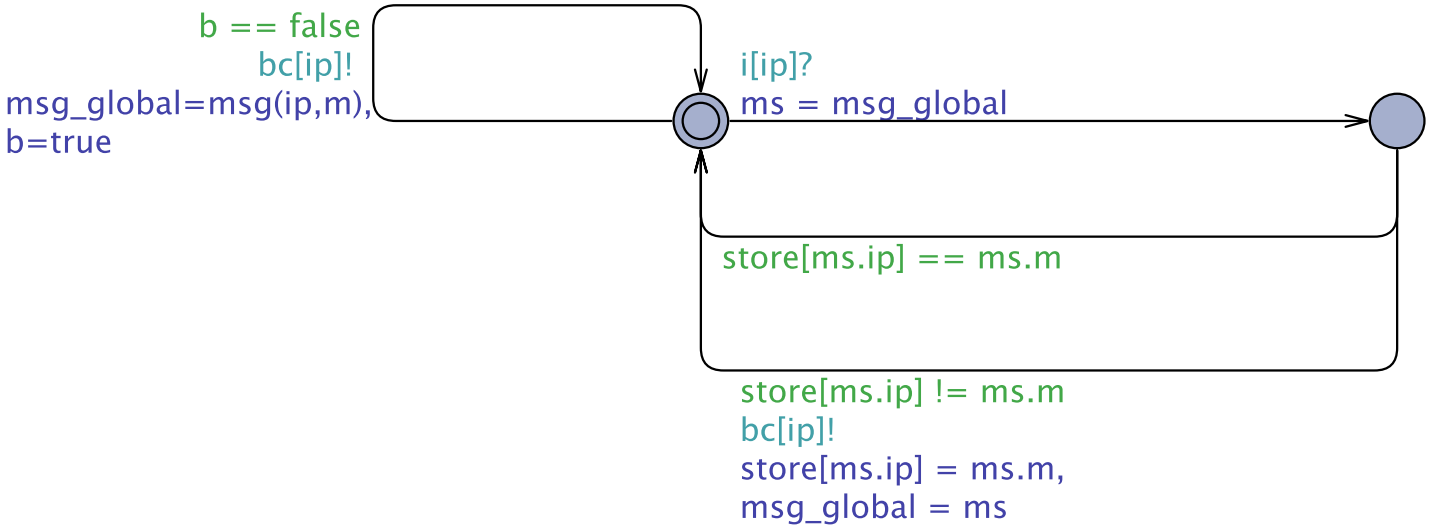
- data structure (functions etc.) needs to be given in Uppaal-syntax

- timed AWN is under consideration (no problems expected)
- only *intuitive correctness* of transformation
  - formal semantics of Uppaal not available

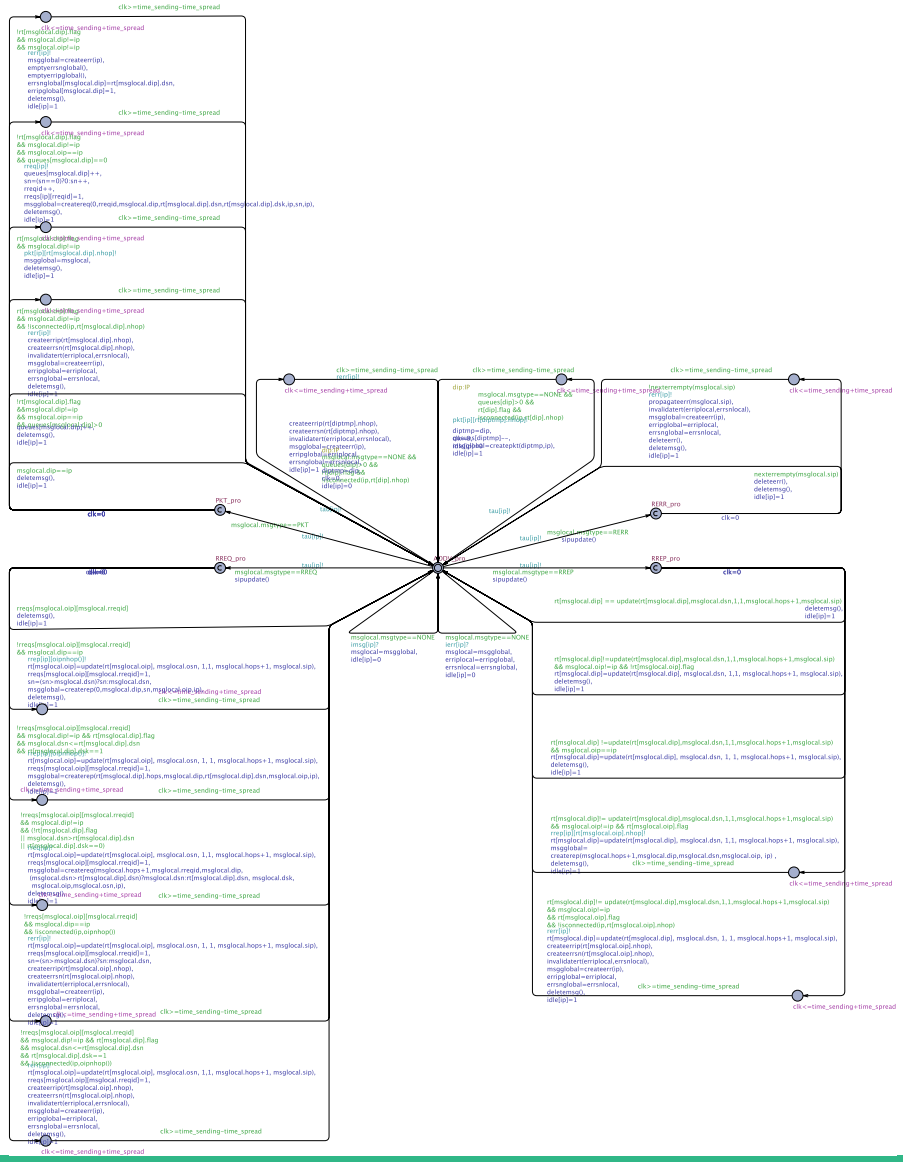
Example: Flooding

Process 1 Flooding

```
FLOOD(ip, m, b, store) =def
 0. (receive(ms) .
 1.    /* check message format and distill contents */
 2.    [ ms = msg(ip′, m′) ]
 3.       (
 4.          [ store(ip′) = m′ ]      /* message handled before */
 4.             FLOOD(ip,m,b,store)
 5.          + [ store(ip′) ≠ m′ ]     /* new message */
 6.             ⟦store(ip′) = m′⟧
 7.             broadcast(ms) .
 7.             FLOOD(ip,m,b,store)
 8.       ))
 9. + [ b = false ]      /* message not yet send */
10.    broadcast(msg(ip, m)) . FLOOD(ip,m,true,store)
```



Example: timed AODV (no queue)



References

- Uppaal homepage: www.uppaal.org
- G. Behrmann, A. David, K. Larsen:
  A Tutorial on Uppaal. In: M. Bernardo, F. Corradini (eds.) Formal Methods for the Design of Real-Time Systems, LNCS 3185, 200–236, Springer, 2004
  doi: 10.1007/978-3-540-30080-9_7, updated pdf available at
  http://www.uppaal.com/admin/anvandarfiler/filer/uppaal-tutorial.pdf