System Integration - Timed Automata

Hugo A. López

hulo@dtu.dk

A little recap

This course so far

+ Operational
Easier to verify

+ Abstract
Harder to verify

This course so far

+ Operational
Easier to verify

+ Abstract
Harder to verify

This course so far

+ Operational
Easier to verify

+ Abstract
Harder to verify

Automata

Computation

This course so far

+ Operational
Easier to verify

+ Abstract
Harder to verify

Automata

Workflow Models

Computation

Task distribution

This course so far

+ Operational
Easier to verify

+ Abstract
Harder to verify

Automata — Computation

Workflow Models — Task distribution

Interaction Models — Handovers

This course so far

+ Operational
Easier to verify

+ Abstract
Harder to verify

Automata | Workflow Models | Interaction Models | Requirement Models

Computation | Task distribution | Handovers | Strategy

This course so far

+ Operational
Easier to verify

+ Abstract
Harder to verify

Automata	Workflow Models	Interaction Models	Requirement Models	Enterprise Architecture Models
Computation	Task distribution	Handovers	Strategy	Technology Alignment

This class

+ Operational
Easier to verify

+ Abstract
Harder to verify

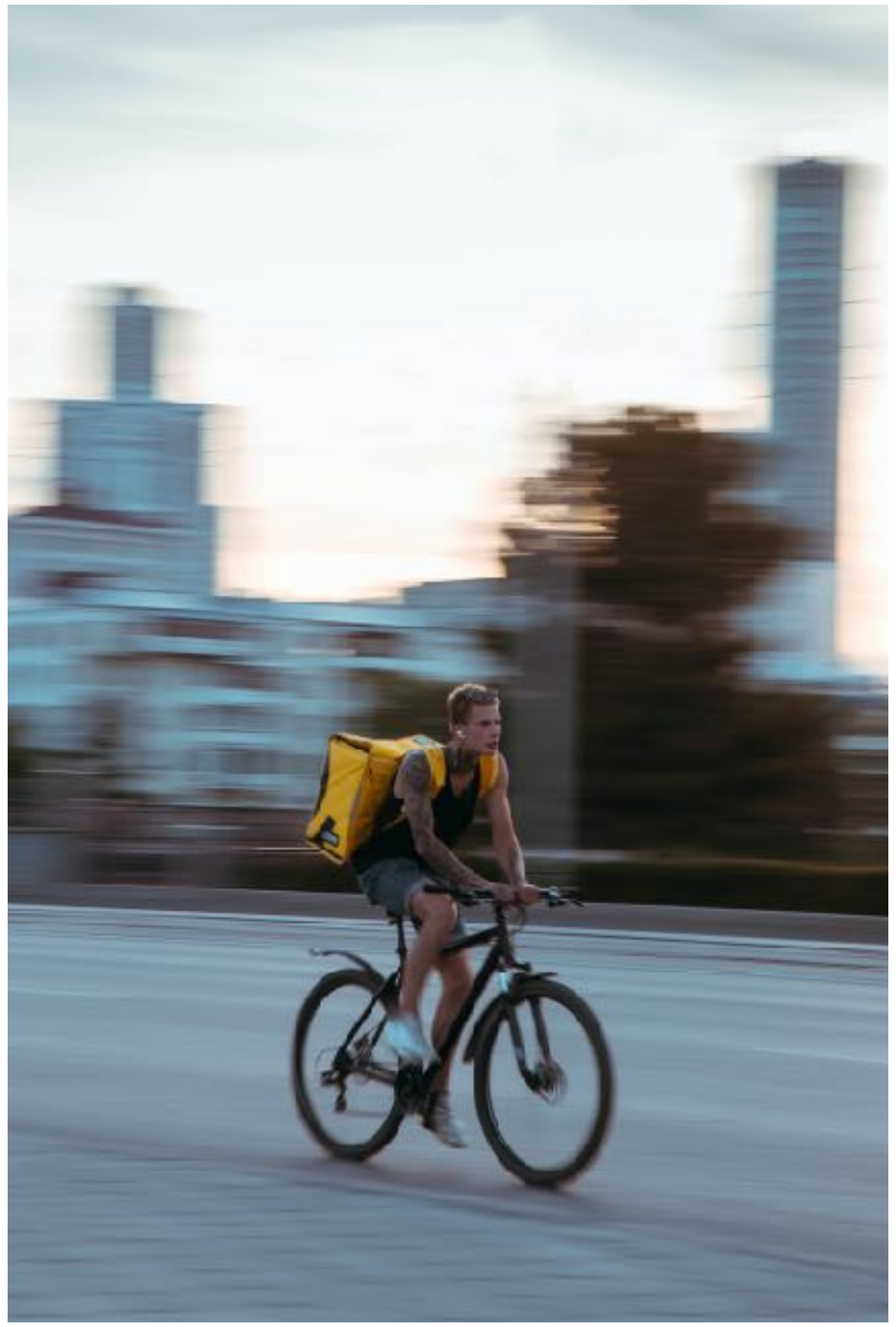
Automata	Workflow Models	Interaction Models	**Real-time models**	Requirement Models	Enterprise Architecture Models
Computation	Task distribution	Handovers	**Temporal Dependencies**	Strategy	Technology Alignment

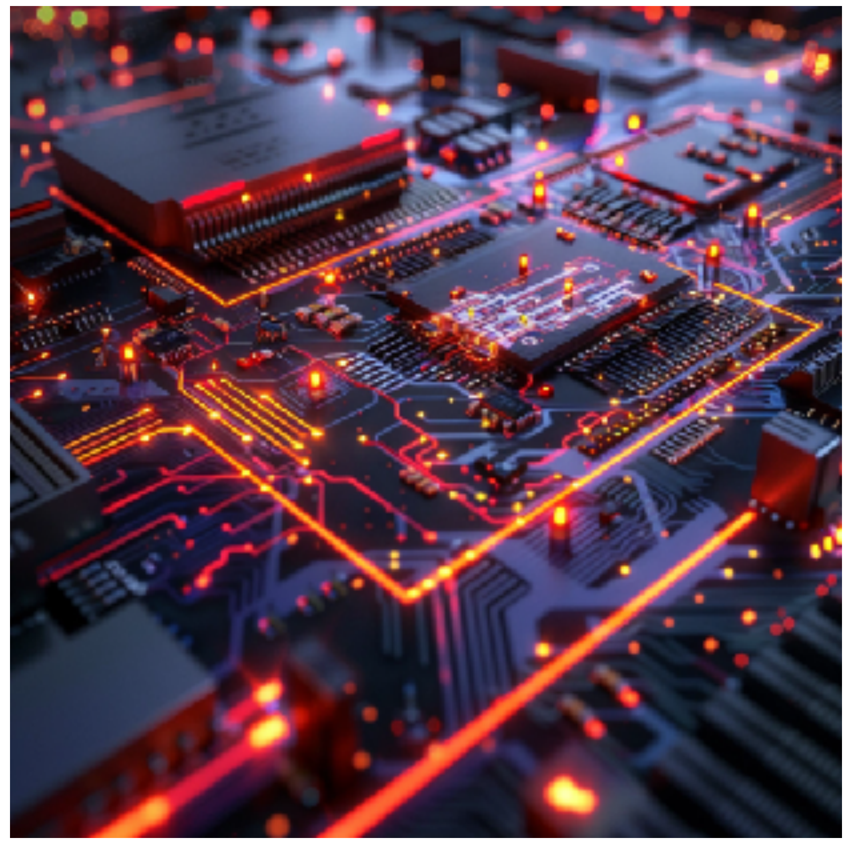
Why real-time systems

Real-time systems

- Parts of the socio-technical system are expected to finish within reliable time bounds
- Communication with microservices does not run forever, but under standard service level agreements
- Fault-tolerant systems need to model availability considerations under absences of responses (failures)

A system where correctness does not only depend on the logical order of events, but also on their **timing**



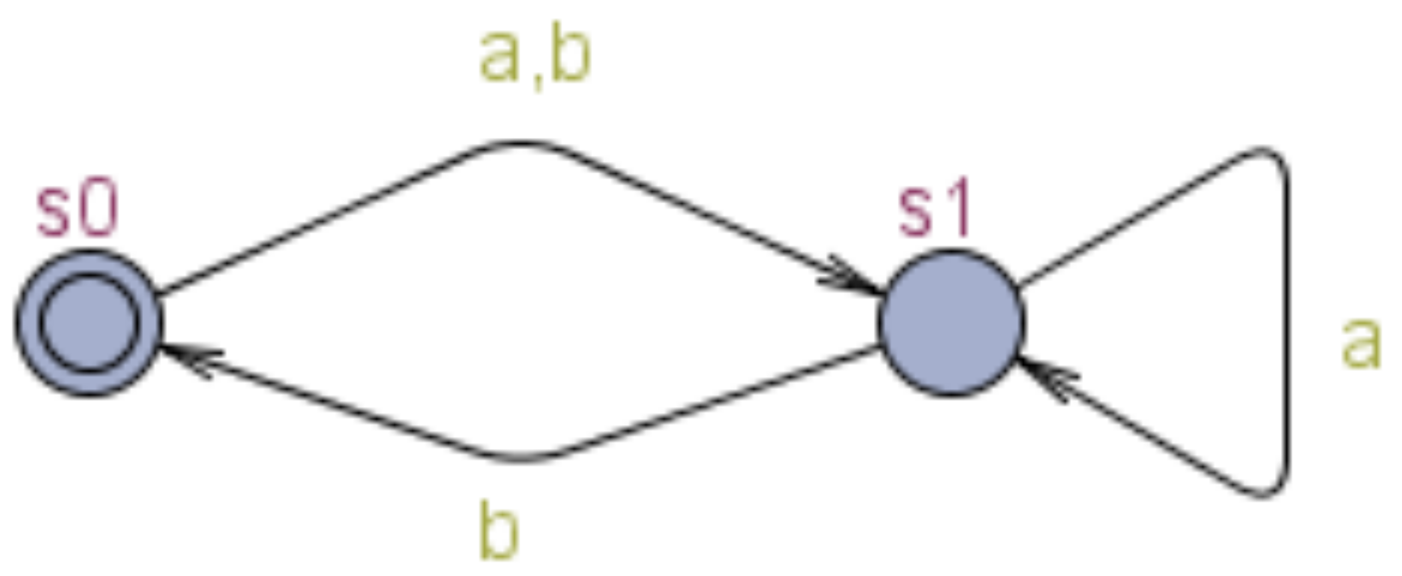


How does time influence our specification?

- Evolution

How does time influence our specification?

- Evolution

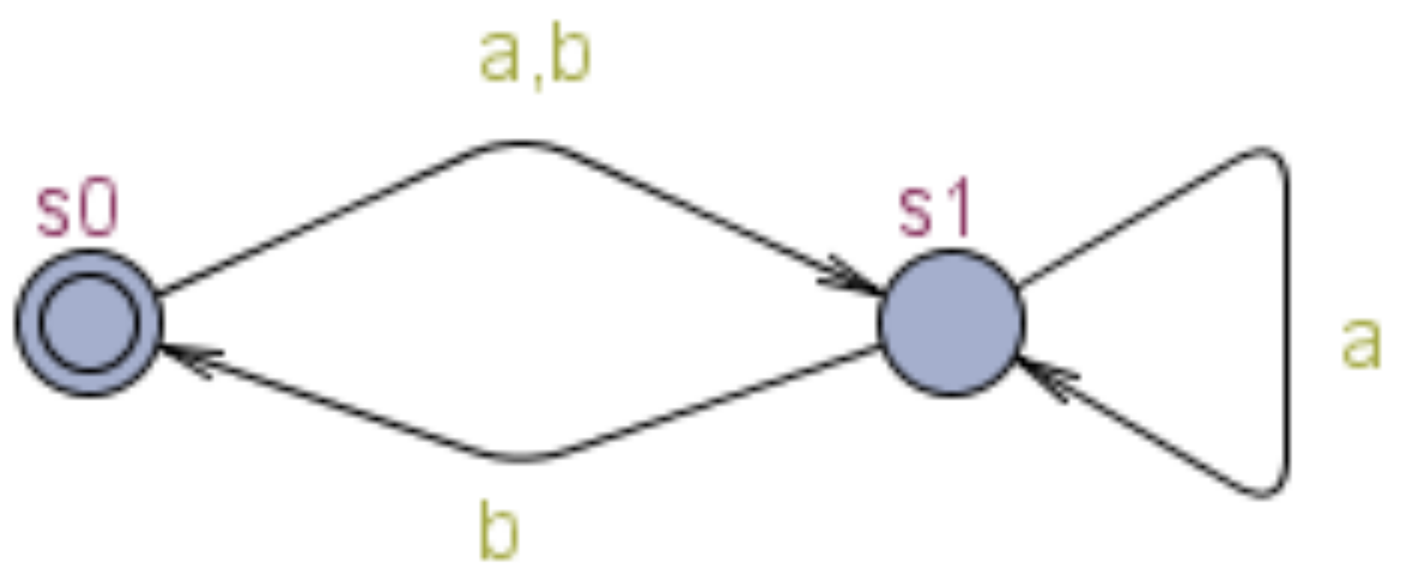


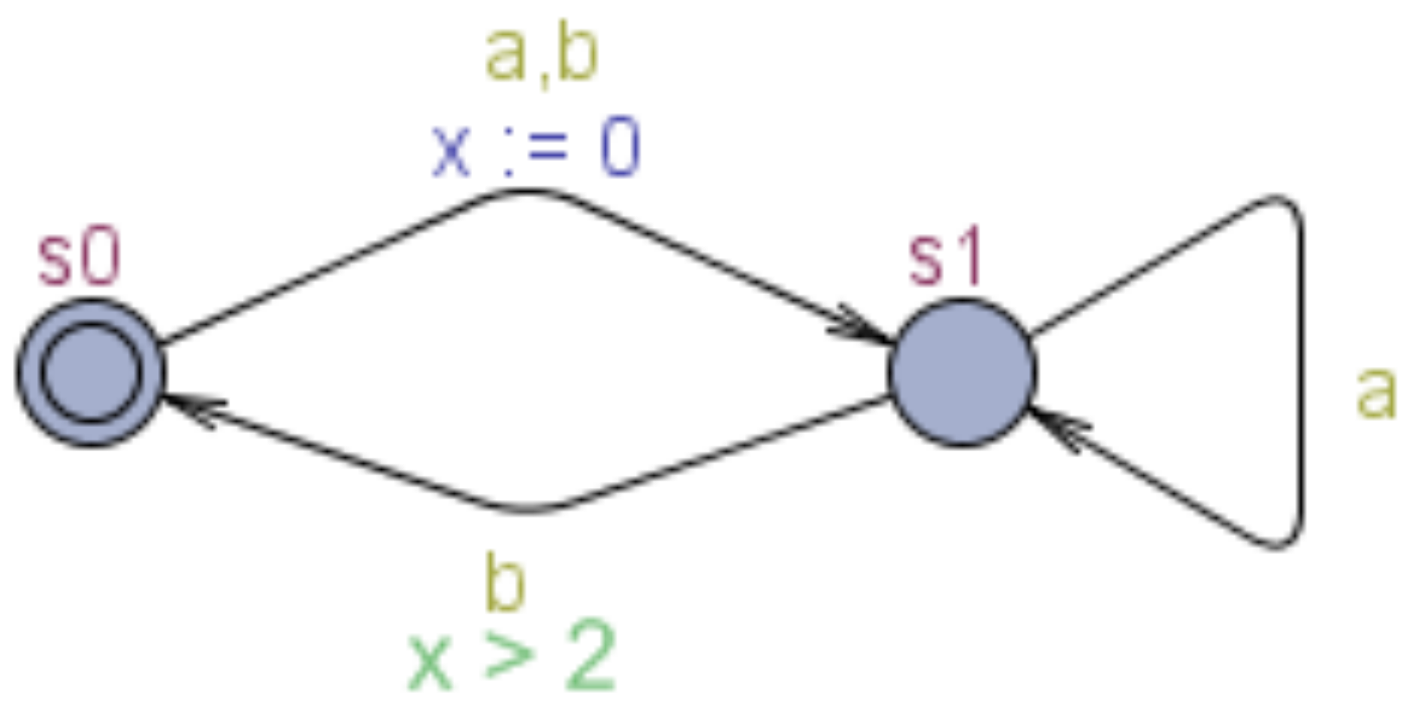
Büchi Automata

- Infinite alphabet
- Initial and accepting states
- Accept execution if pass through accepting state infinitely many times

How does time influence our specification?

- Evolution





Büchi Automata

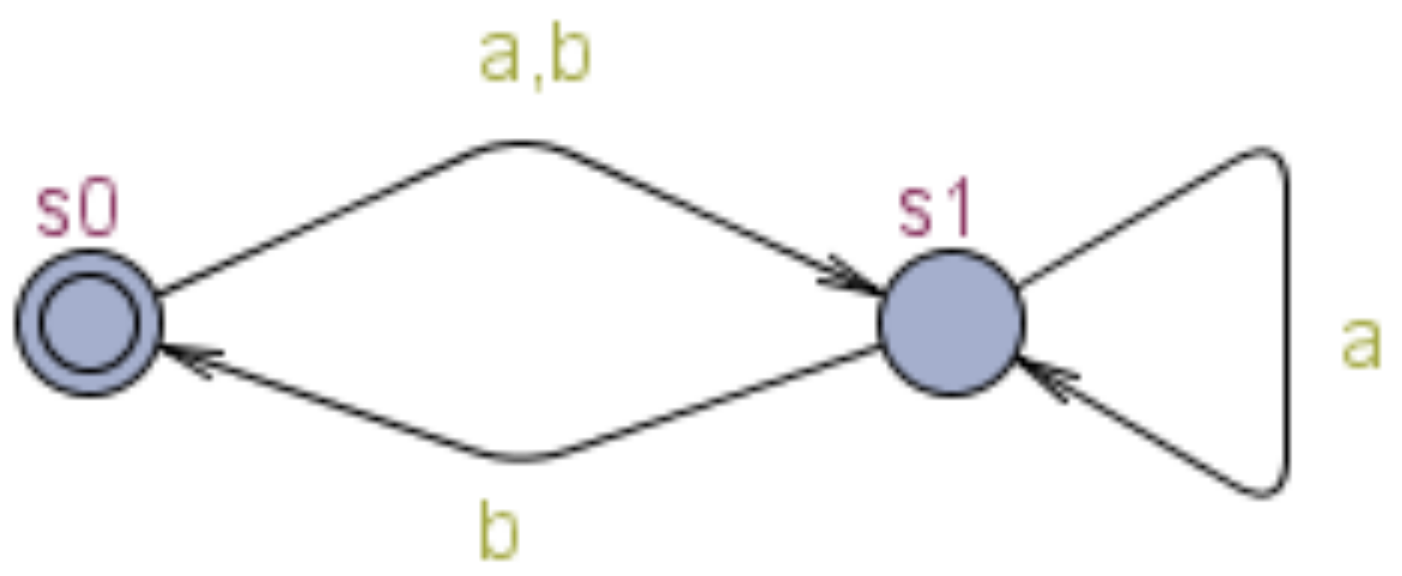
- Infinite alphabet
- Initial and accepting states
- Accept execution if pass through accepting state infinitely many times

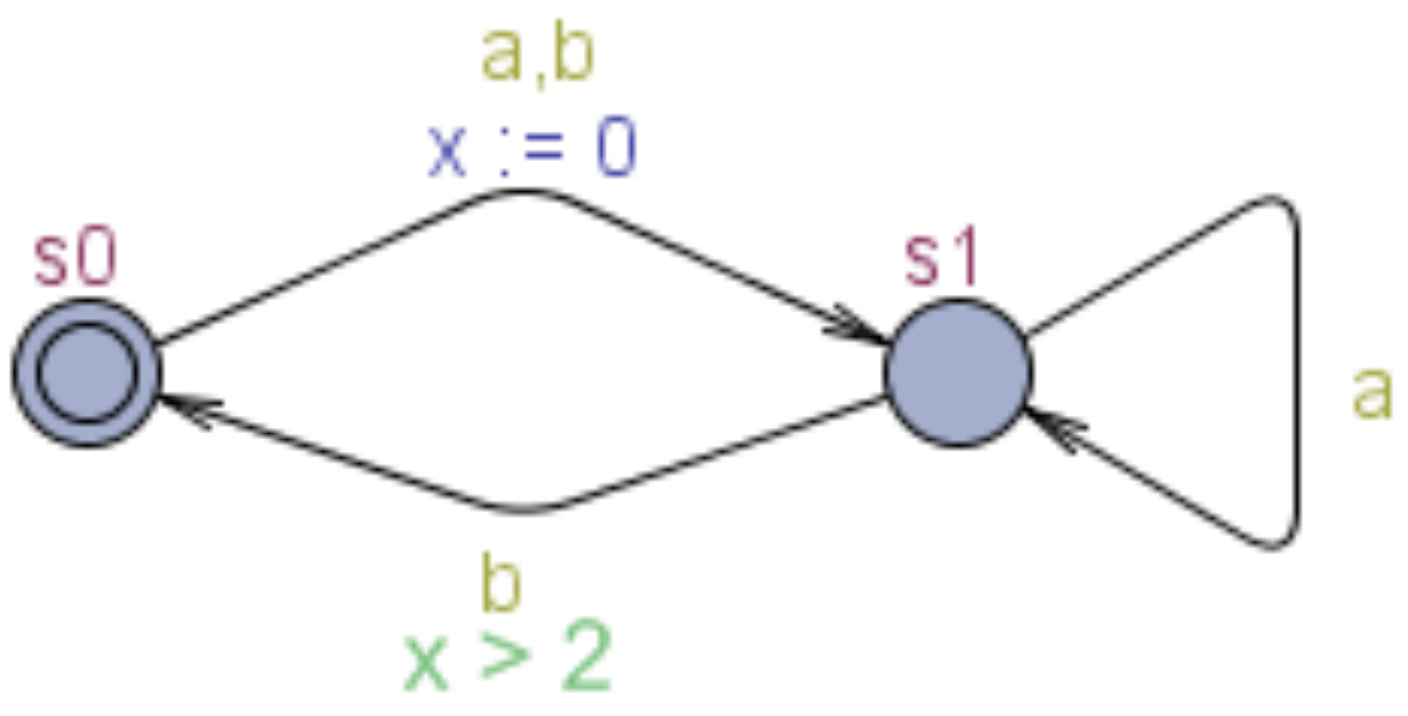
Büchi Timed Automata

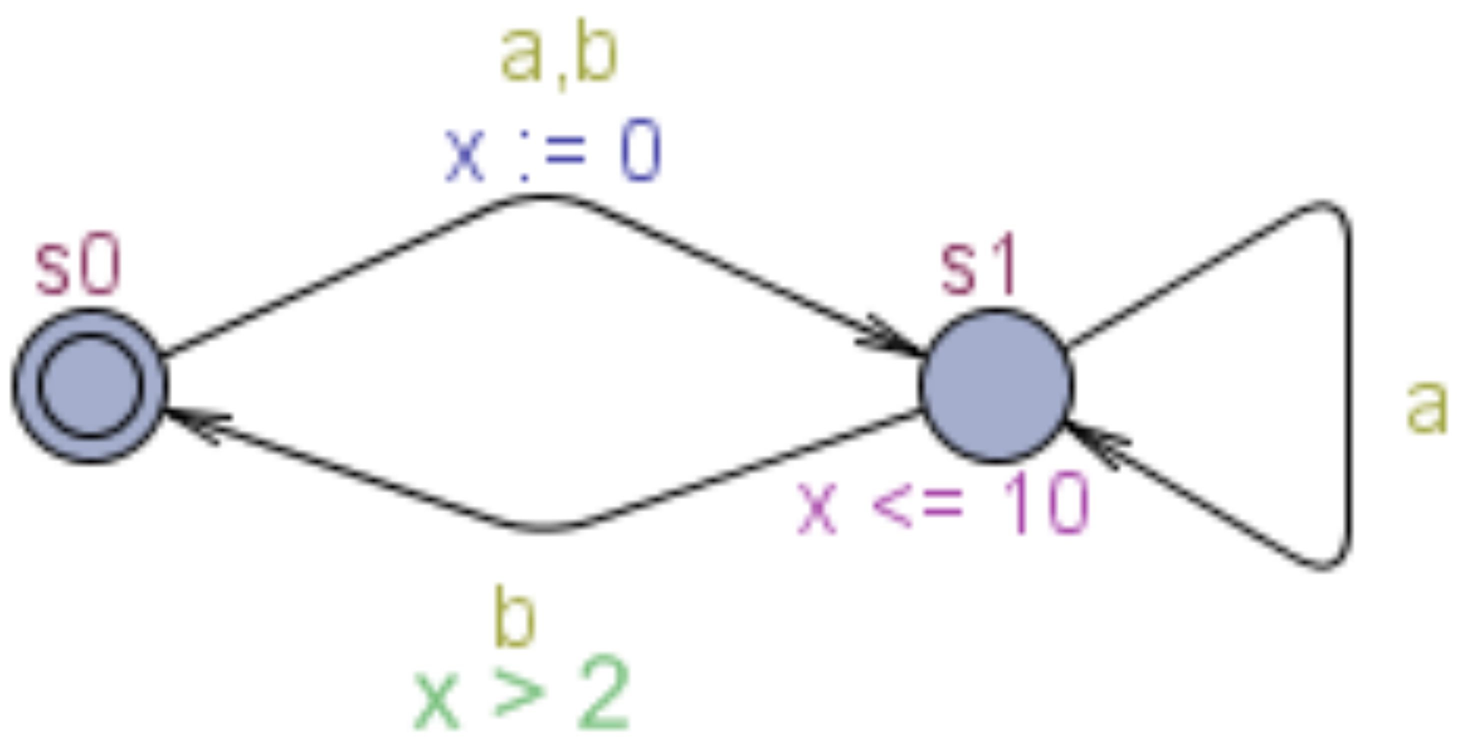
- Büchi-accepting
- Real-valued variables: modelling clock
- Constraints on clock variables and resets

How does time influence our specification?

- Evolution







Büchi Automata

- Infinite alphabet
- Initial and accepting states
- Accept execution if pass through accepting state infinitely many times

Büchi Timed Automata

- Büchi-accepting
- Real-valued variables: modelling clock
- Constraints on clock variables and resets

Timed Safety Automata

- Clock variables
- Local invariant conditions
- Accept when invariant is satisfied

Timed Automata

- A timed automaton is a tuple

$$(\mathcal{L}, l_0, C, \mathcal{A}, \mathcal{E}, \mathcal{I})$$

- where $\mathcal{L}$ is a set of locations,
- $l_0 \in \mathcal{L}$ is the initial location,
- C is the set of clocks,
- $\mathcal{A}$ is a set of actions, co-actions and the internal τ-action,
- $\mathcal{E} \subseteq \mathcal{L} \times \mathcal{A} \times \mathcal{B}(C) \times 2^C \times \mathcal{L}$ is a set of edges between locations with an action, a guard and a set of clocks to be reset, and
- $\mathcal{I} : \mathcal{L} \rightarrow \mathcal{B}(C)$ assigns invariants to locations

Semantics

The operational Semantics of a timed automaton is:

- If $u, u + d \in I(l)$ and $d \in \mathbb{R}^+$,

  then $\langle l, u \rangle \xrightarrow{d} \langle l, u + d \rangle$

Timed action

- If $l \xrightarrow{\tau,\alpha,r} l', u \in g, u' = [r \mapsto 0]u$ and $u' \in I(l)$,

  then $\langle l, u \rangle \xrightarrow{\alpha} \langle l', u' \rangle$

Labelled action

- **Notation:** $\langle l, u \rangle$ is a state

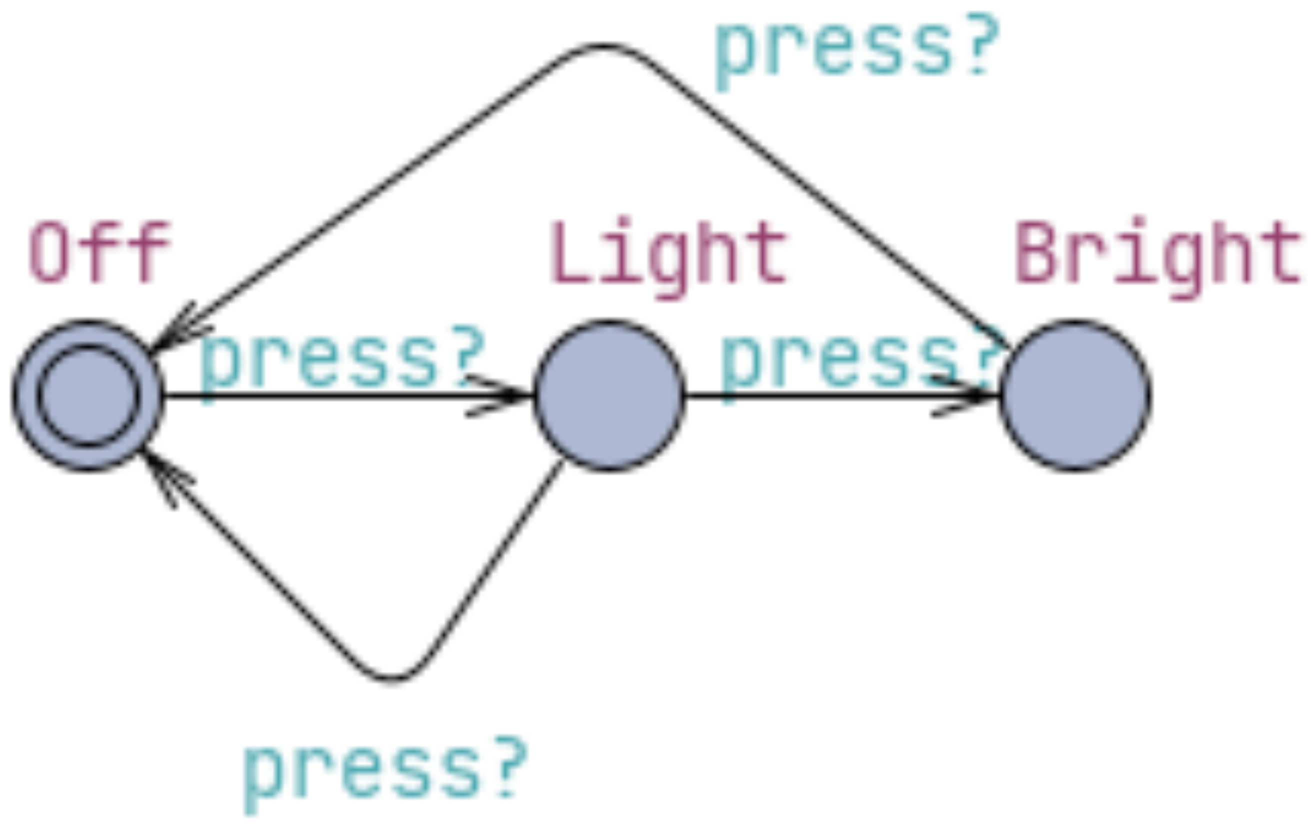
  $\langle l, u \rangle \xrightarrow{\alpha} \langle l', u' \rangle$ is a transition

Operational Semantics

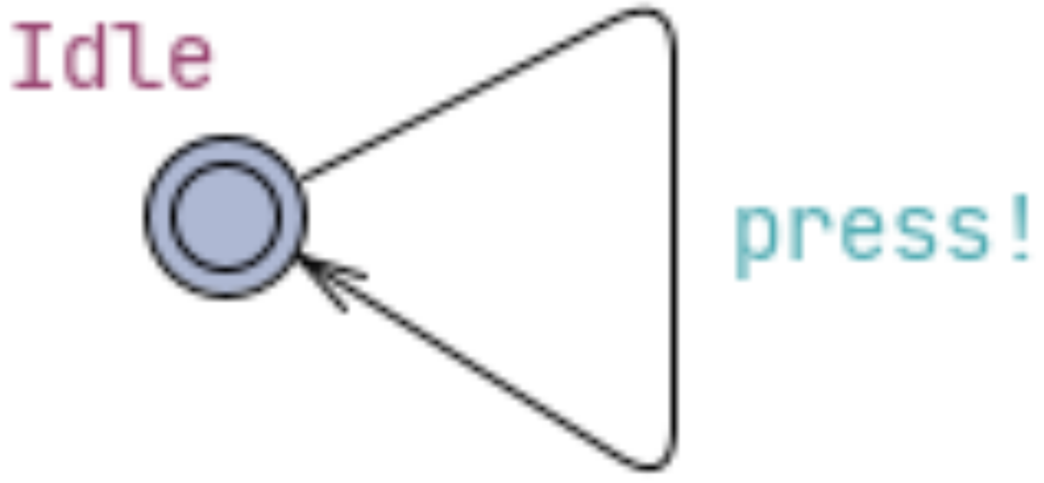
- For a TA $(\mathcal{L}, l_0, C, \mathcal{A}, \mathcal{E}, \mathcal{I})$, its semantics is given in terms of an LTS $\langle S, s_0, \rightarrow \rangle$, where
  - $S \subseteq \mathcal{L} \times \mathcal{R}^C$ is the set of states,
  - $s_0 = (l_0, u_0)$ is the initial state, and
  - $\rightarrow \subseteq S \times \{\mathcal{R}_{\geq 0} \cup \mathcal{A}\} \times S$ is the transition relation such that:
    - $(l,u) \xrightarrow{d} (l, u+d)$ if $\forall d' : 0 \leq d' \leq d \Rightarrow u+d \in \mathcal{I}(l)$, and
    - $(l, u) \xrightarrow{a} (l', u')$ if there exists $e = (l, a, g, r, l') \in \mathcal{E}$ such that
      - $u \in g$,
      - $u' = [r \mapsto 0]u$, and
      - $u' \in \mathcal{I}(l)$

The transition respect timed invariants

Our first CPS



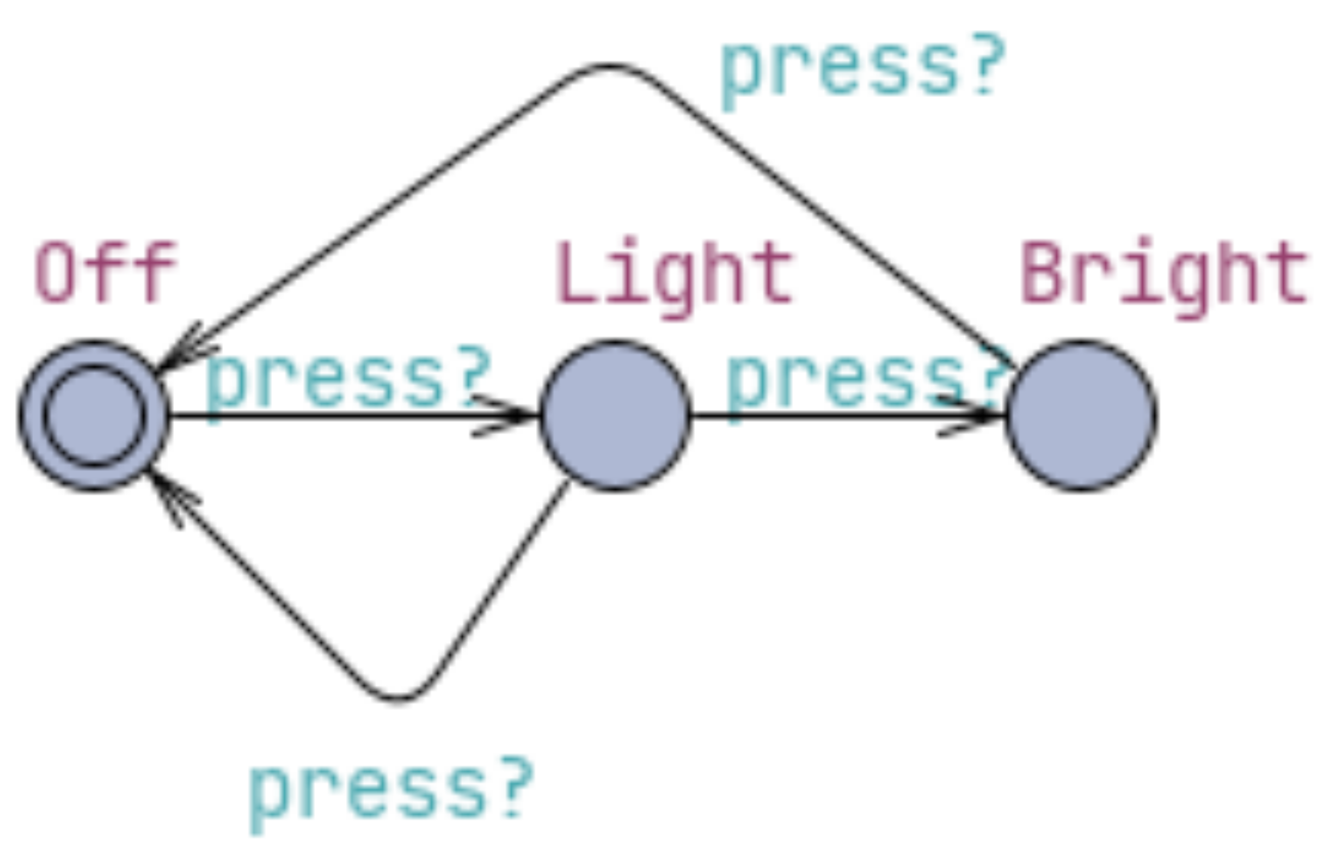
Lamp



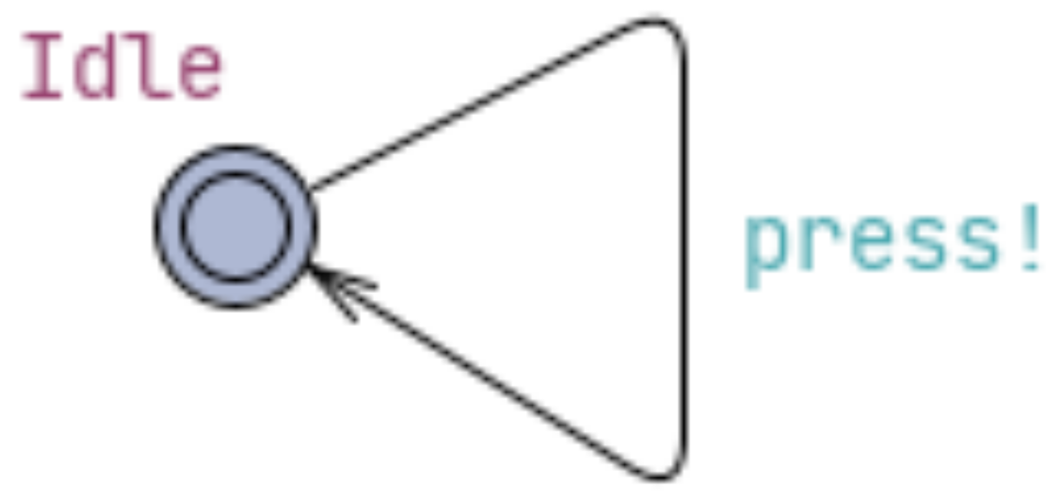
User

- System = parallel composition of lamp and user

Our first CPS



Lamp

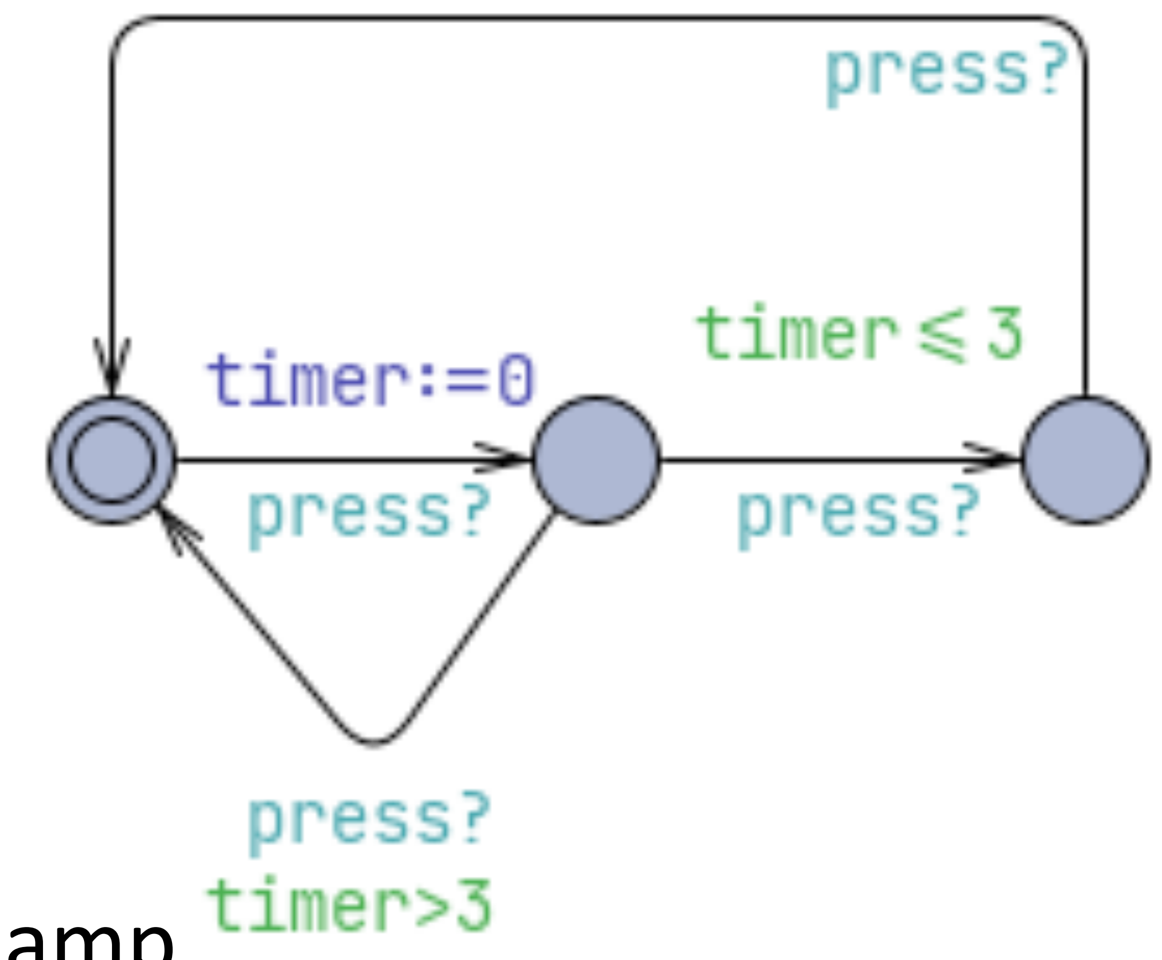


User

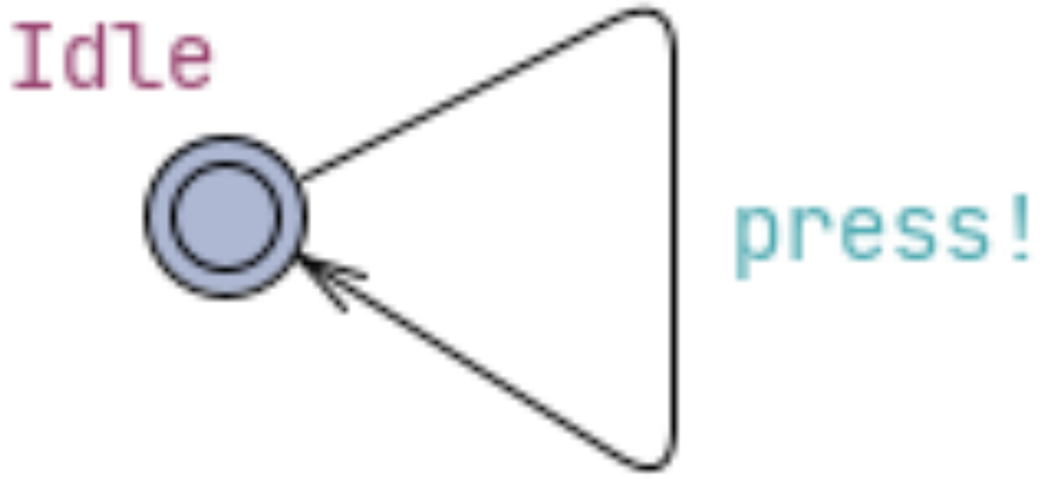
- System = parallel composition of lamp and user

Question: how can we model the policy that the light should be dimmable if the user presses **quickly**?

Adding timers & guards



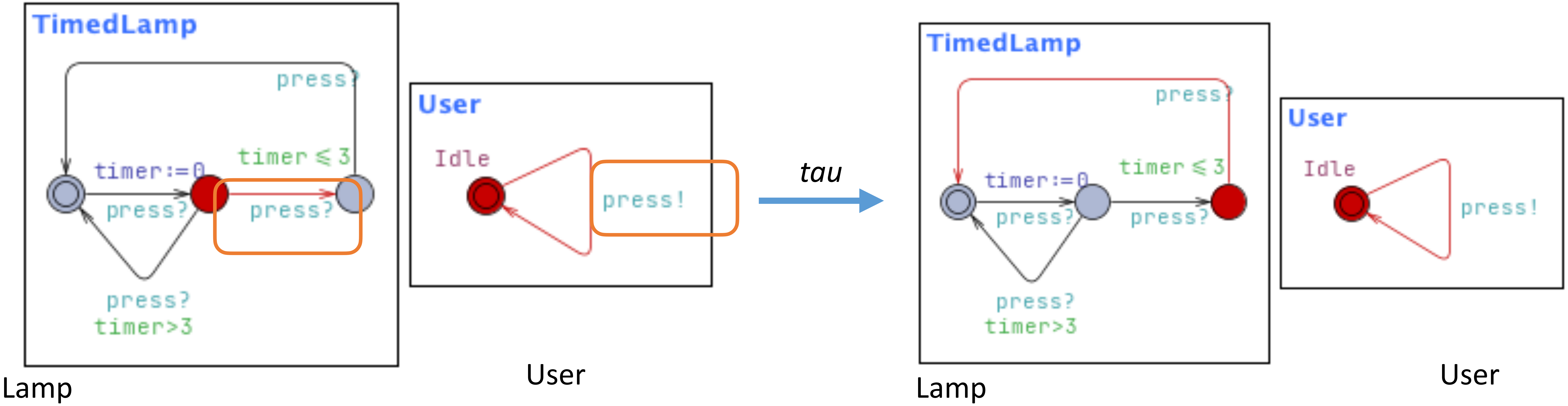
Lamp



User

- A real-value timer measures the delay between the *press* events
- Clocks can be resetted, and queried
- Multiple clocks can be instantiated

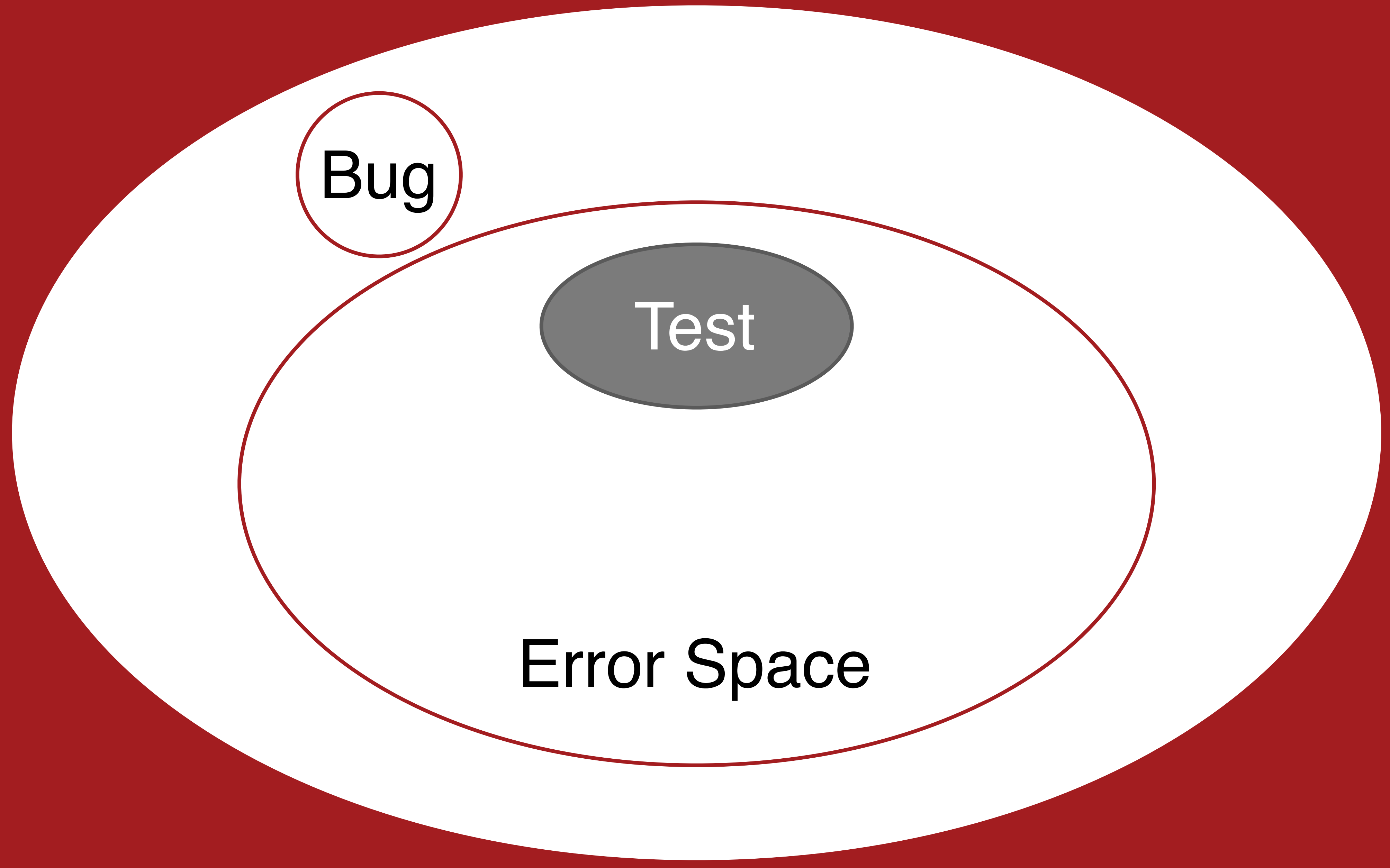
Networks of Timed Automata



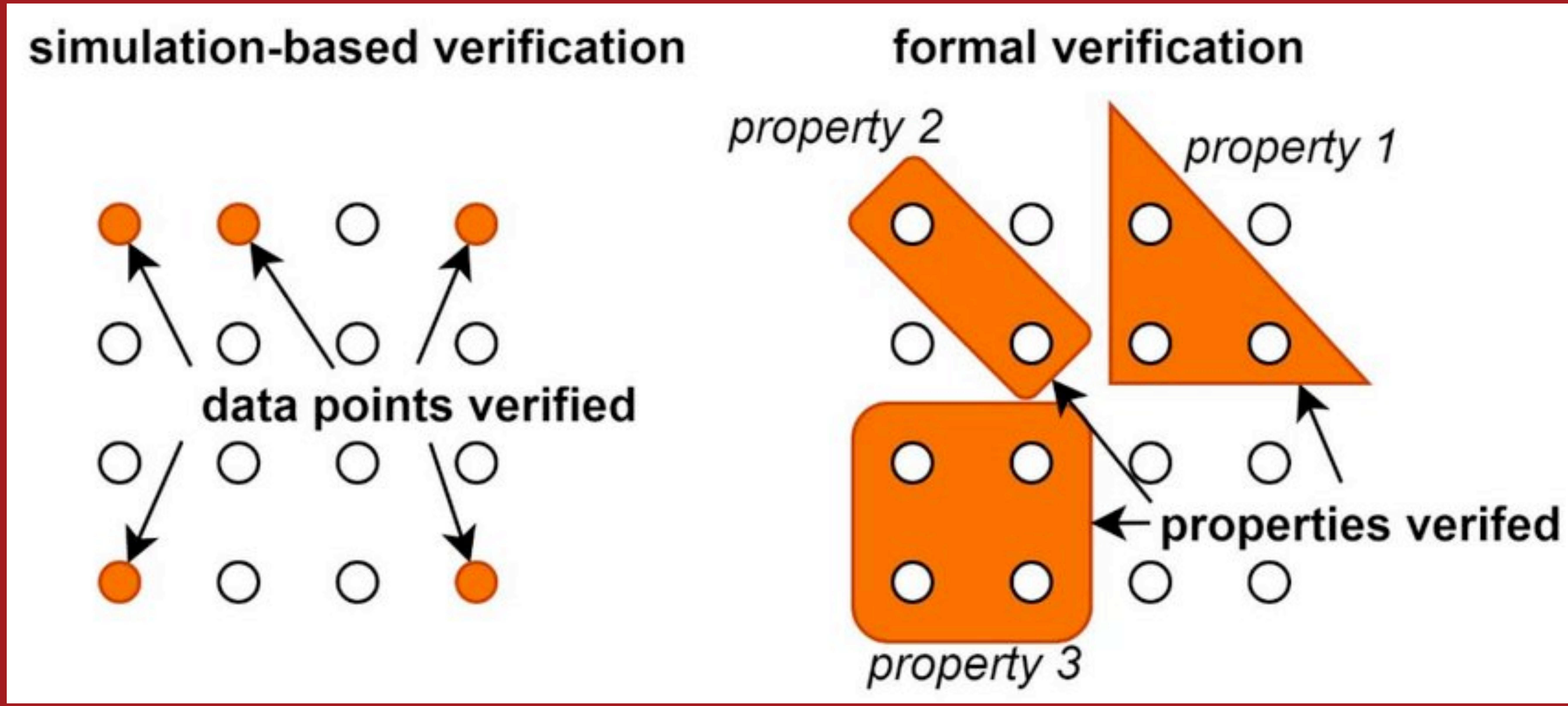
- A network of timed automata provides a two-way synchronization between co-actions
  - Akin to CCS!

Extensions

- See tutorial for more details
  - Urgent channels: no delay if transition with urgent action can be taken.
  - Committed locations: reduce the number of clocks and disallow delays between commited locations
  - Broadcast channels: one-to-many communication
  - Parameterised templates: allow to span finite copies of a model



Input Space



Testing and Verification

A testing approach does not guarantee a fail-free implementation. It guarantees that the software passes the tests you have provided.
If you need full-assurance, you need higher guarantees, for instance, formal methods

Verification of Timed Models

- Instead of running tests, we would like to apply **model-checking**
- Types of properties:
  - Safety: “**Something bad never happens**” [Lamport 1977]
    - Type 1: a broken invariant in the specification as state properties that fail
      - E.g. A mutual exclusion algorithm that allows more than 1 processors in the critical section
    - Every invariant is a safety property, but not the reverse
      - For e.g. "vending machine should get money **before** dispensing a product” is not a state property, but a temporal property

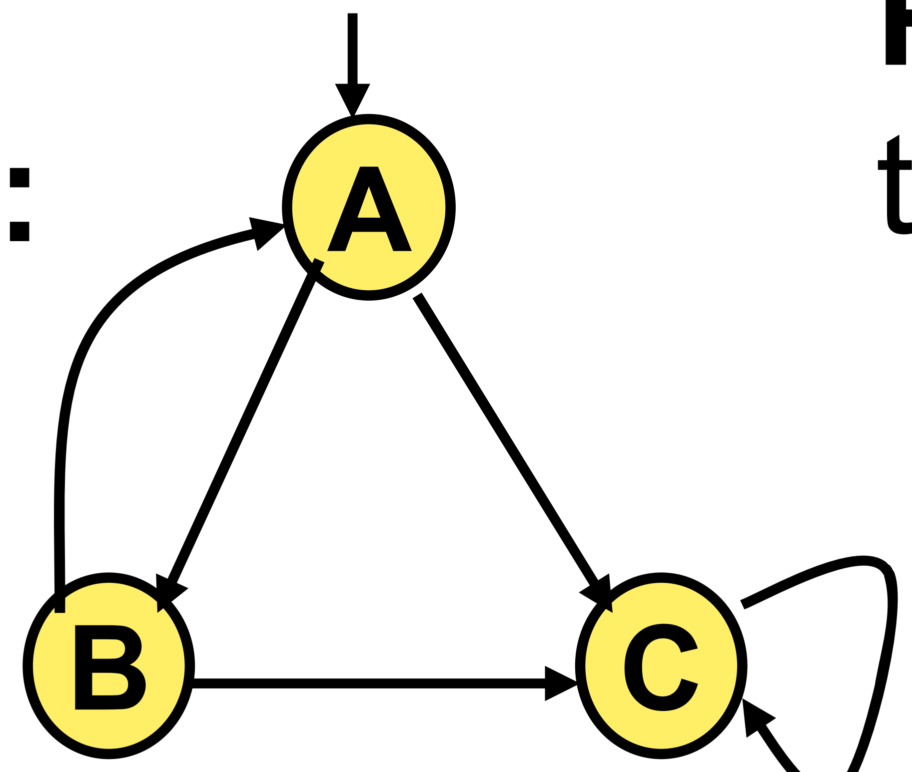
Liveness

- Doing nothing easily fulfils a safety property as this will never lead to a "bad" situation
  - For e.g. "vending machine should get money **before** dispensing a product" not buying a product satisfy the property
- Liveness properties complement safety properties and require progress
  - ``something good" will happen eventually [Lamport 1977]
- Liveness properties are violated in "infinite time"
  - whereas safety properties are violated in finite time
  - finite traces are of no use to decide whether P_{live} holds or not
  - any finite prefix can be extended such that the resulting infinite trace satisfies P_{live}
  - Examples: fairness

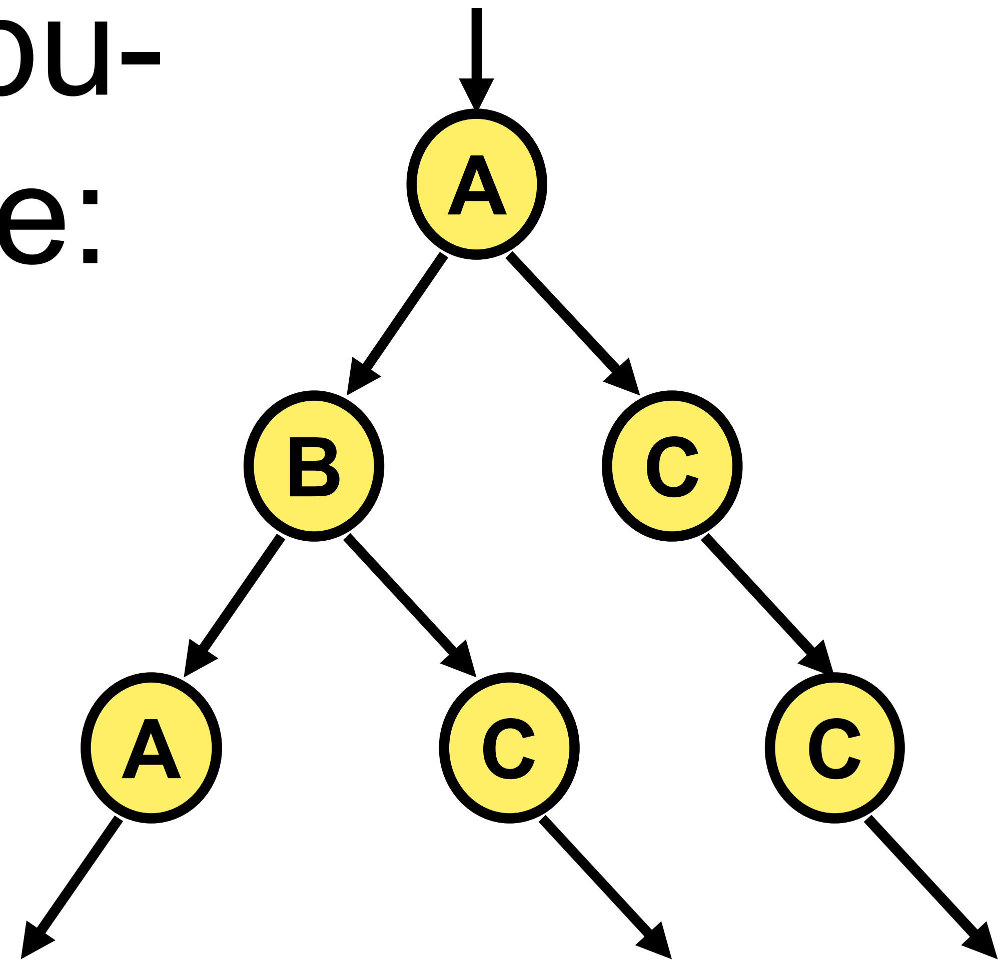
Verification in Uppaal

- A model checker verifies whether a model respects a requirement
- UPPAAL uses a simplified version of CTL [1] (temporal first-order logic)
- State formulae
- Path formulae: reachability, safety, liveness

P:



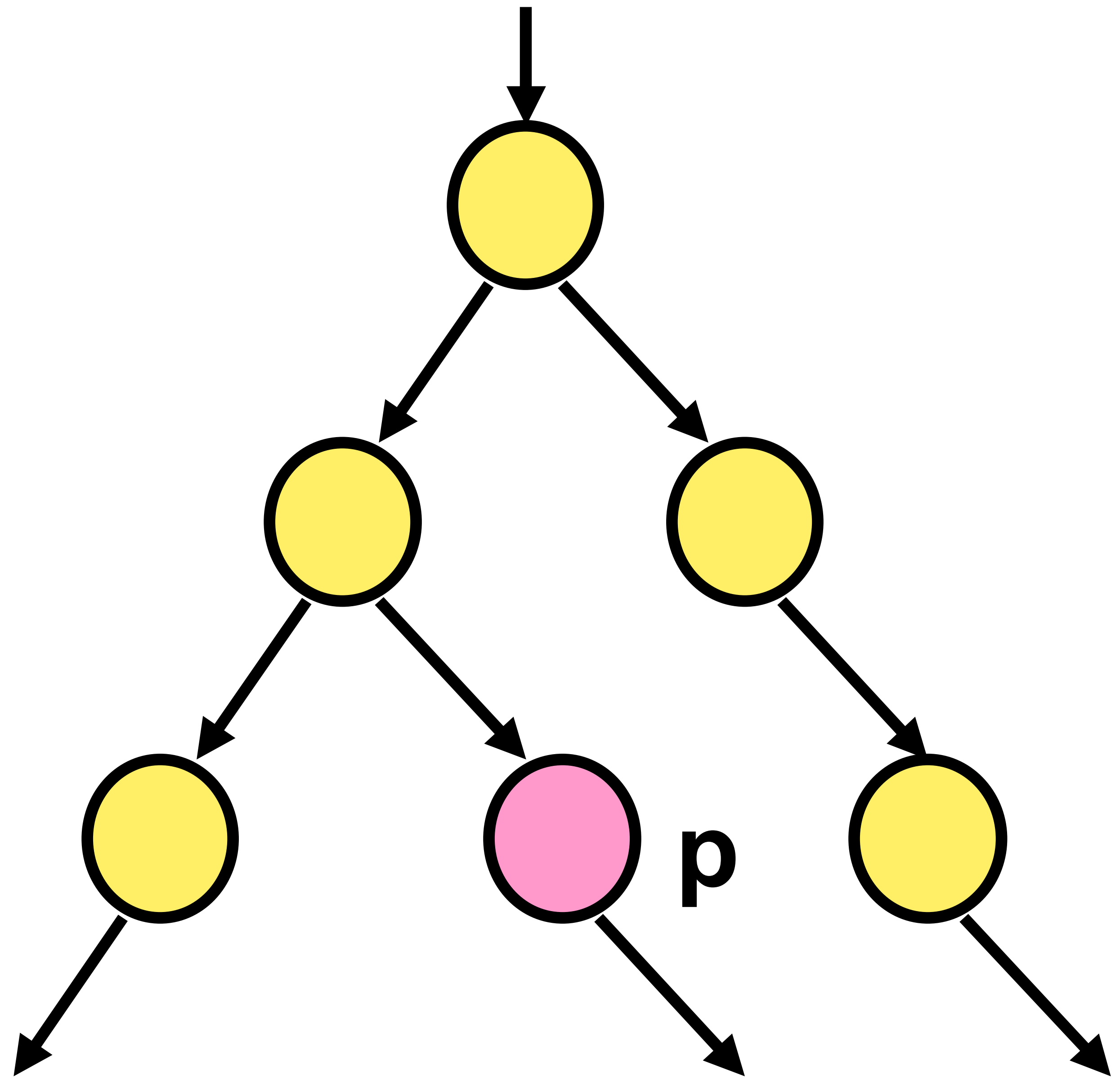
P's computation tree:



[1] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, April, 1986.

Formulae in TCTL

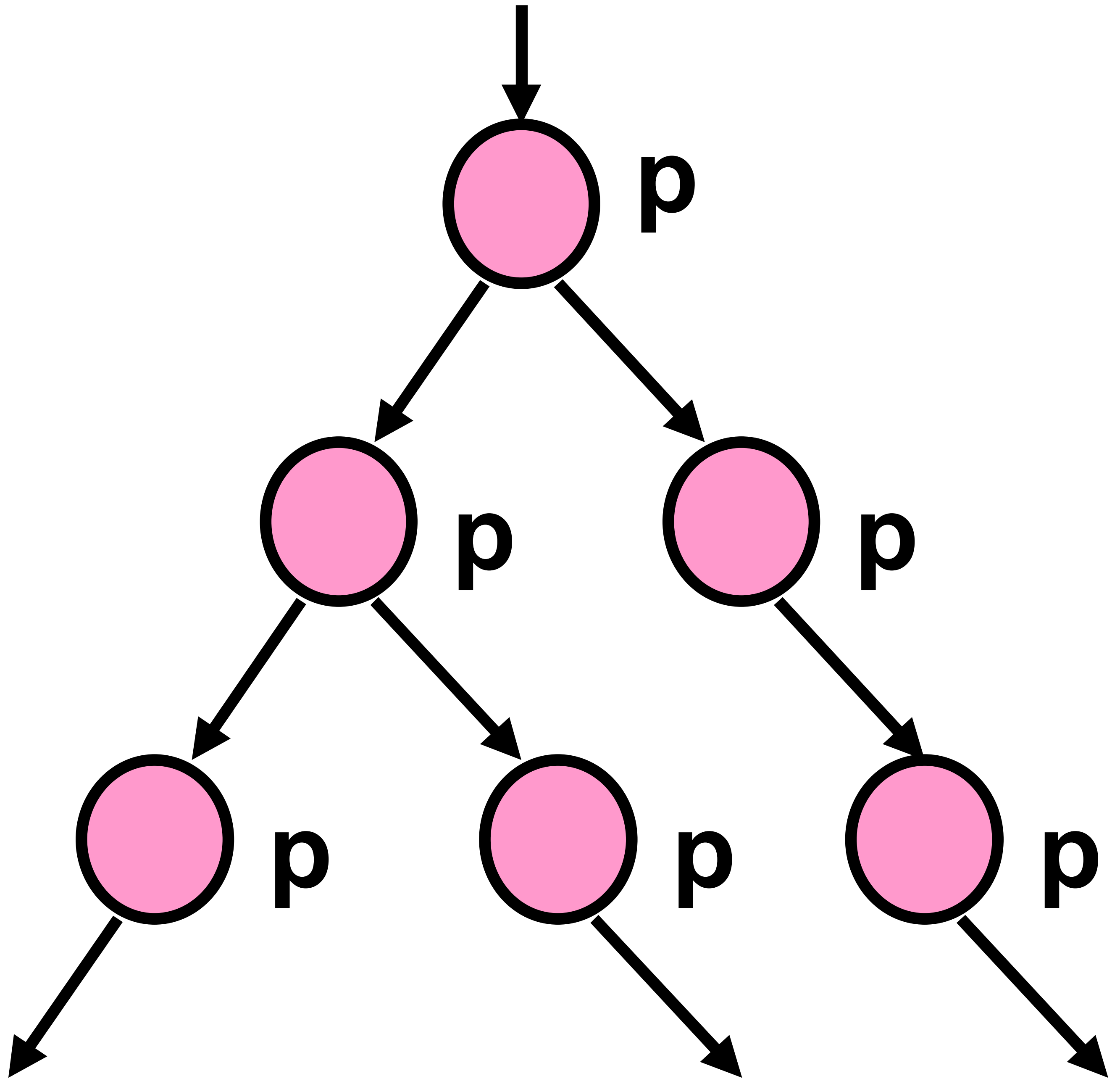
- Reachability -> E<> *P*
- Reads as “it is possible to reach a state in which *P* is satisfied



P is true in at least one reachable state

Formulae in TCTL

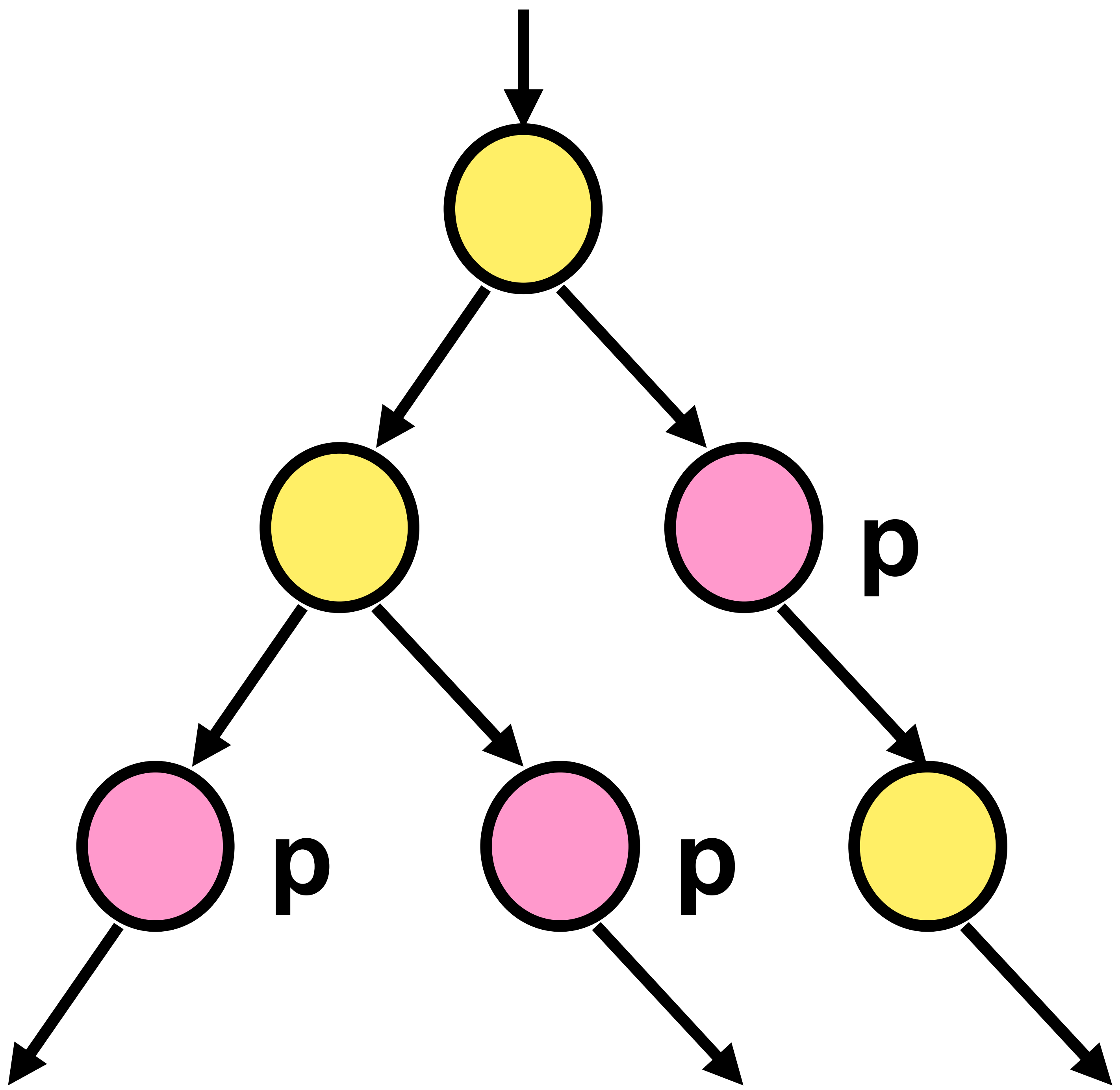
- Invariance: A[] *P*
- "Always *P*'"



P holds in each possible state

Formulae in TCTL

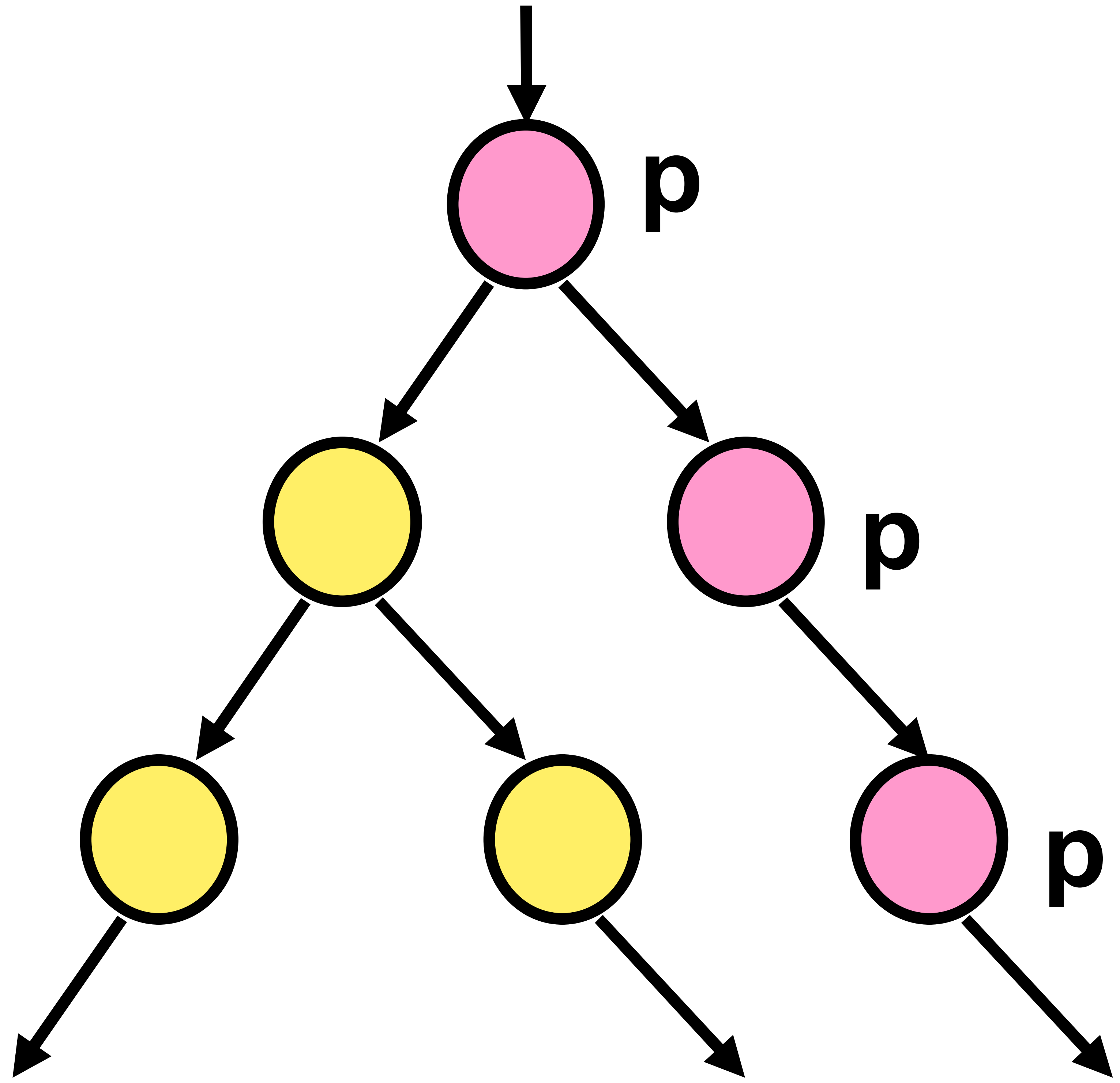
- Inevitable *P*, A<> *P*
- *P* will always eventually become true



P is true in all paths, in some state

Formulae in TCTL

- Potentially Always: E[] *P.*
- There exists a path in which P is true for each transition



Formulae in TCTL

- `A[]p, A<>p, E<>p, E[]p` - `p` is a local property

- Syntax:

no action transition going out of a state or of its delay successors

data guard

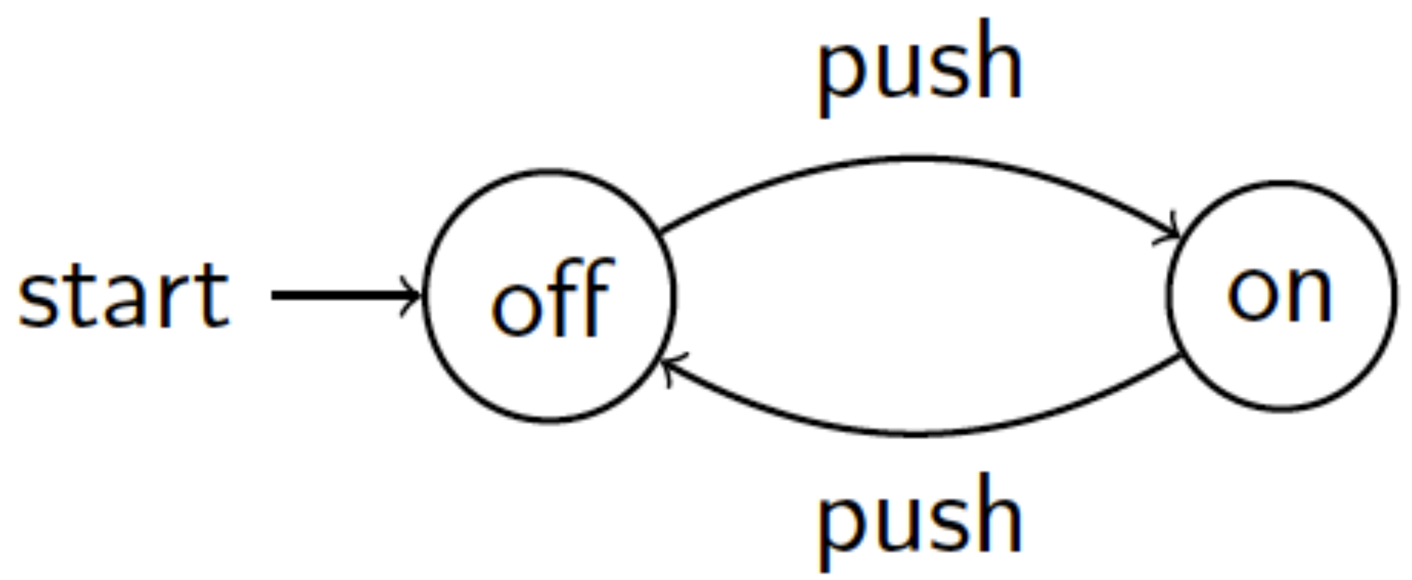
clock guard

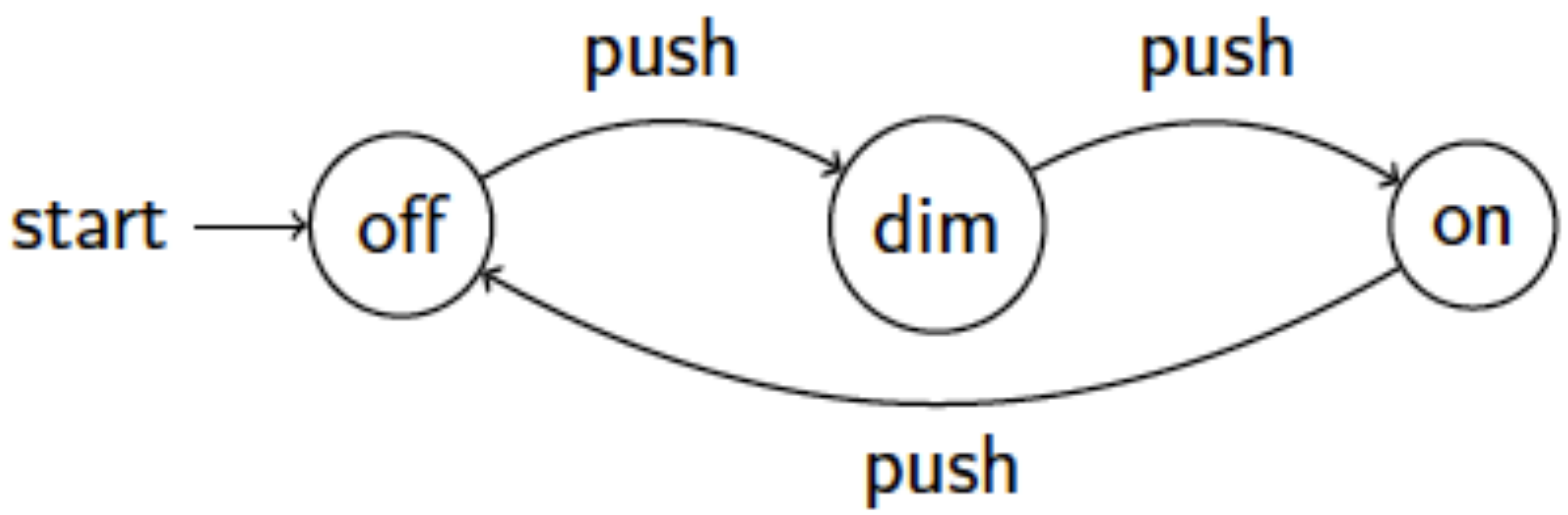
automata location

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

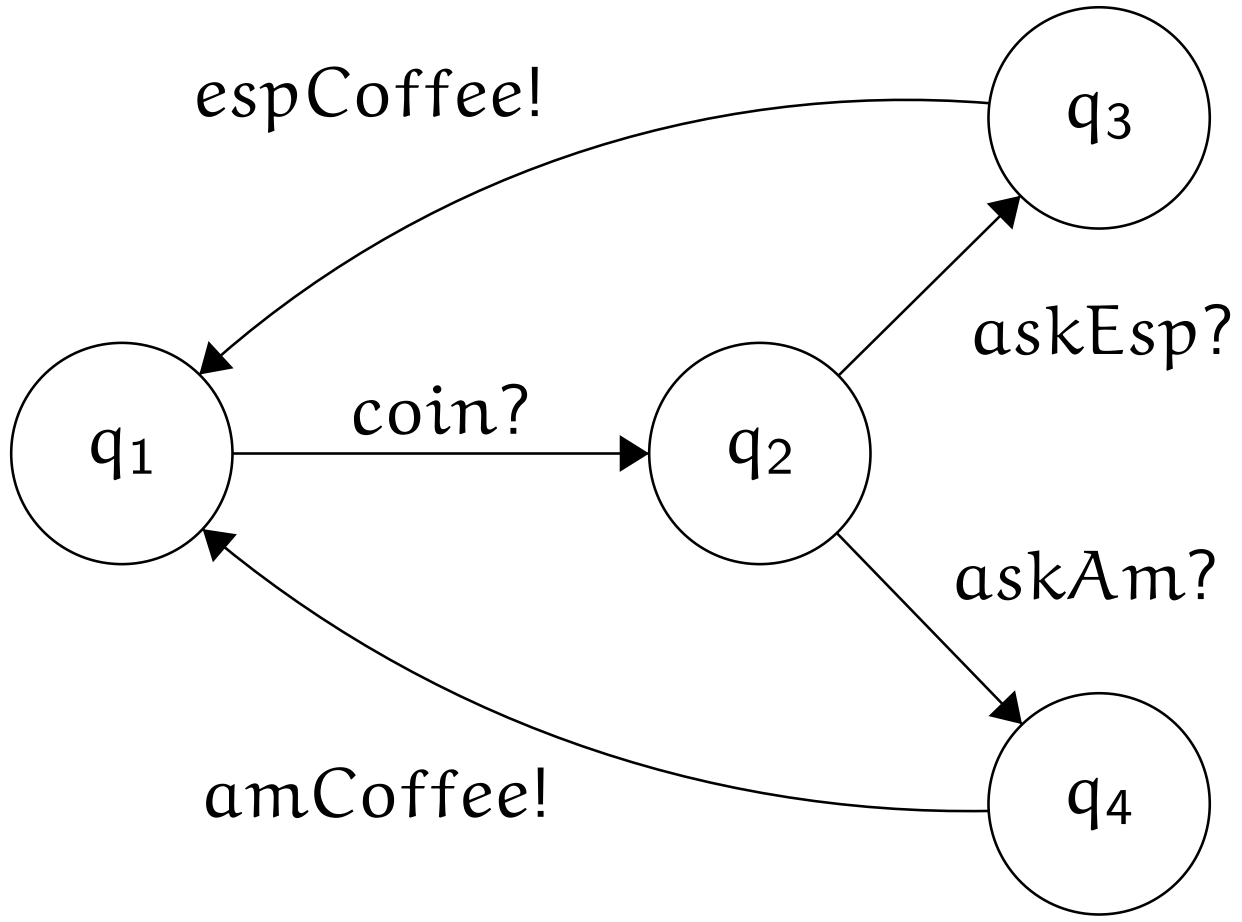
process name

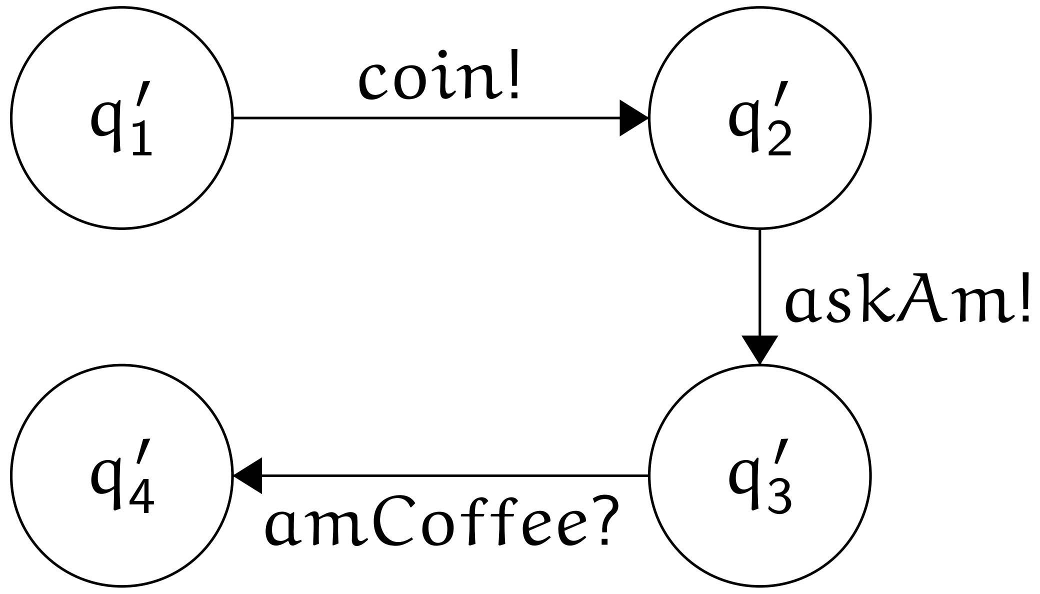
Uppaal Demo: A lamp





Second try: the coffee vending machine





Take home message

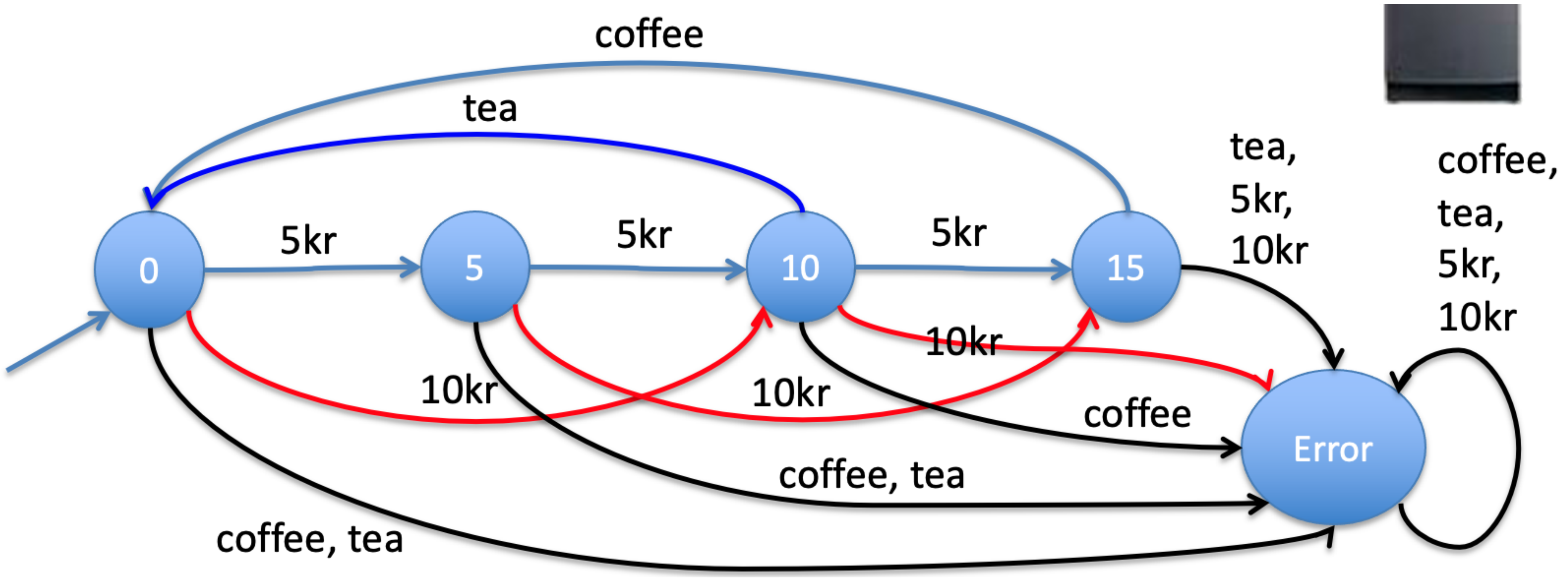
- Real time systems combine action-determinism, message passing, and time operations in to be considered correct
- While the state space becomes indeed humoungous, we can verify the correctness of the system via model checking techniques

Exercise

- Follow the tutorial "A Tutorial on Uppaal"
  Gerd Behrmann, Alexandre David, and Kim G. Larsen, and implement the models in sections 4 and 5.

Exercise

- Implement the following state machine in uppaal, together with a corresponding set of users (that like tea or coffee)
- Make sure that the vending machine returns money after a timeout of 30 seconds
- Verify that for a multiple user can eventually be served by the vending machine (fairness)



Exercise

- Consider the interaction model you handed in
  - Discuss with your group the type of timing constraints you may have in your model
    - And if you can, implement the model in Uppaal
  - Discuss with your group the type of model-checking properties you may be interested
    - And If you manage to implement the models, try to verify the new properties

Peer review

- Carried out individually
- Step 1: each student submits a copy of the group project report and of the models they already submitted in the Assignment section.
- Submission through the Peer-review part 1 module in the Peer-review section
- All students working in the same group will submit the same files
- Make sure that the report and models are anonymized
- Deadline: March 14 at 11:59 (A.M.)

Peer review

- Step 2: each student reviews 1 report of another group.
- Review through the Peer-review part 1 module in the Peer-review section
- All students working in the same group will get different reports
- Assessment is guided: a rubric of several criteria is defined
- You need to assign a score for each criterion
- You need to motivate your score, pointing out where issues are present in the models
- Feedback is anonymous to students (but not to teachers)
- Deadline: March 19 at 23:59.
- If you get more than 1 report to review, please contact the teachers.