Advanced Human-Machine Interaction
Interaction Data Analysis
Automata / State Machine

Alexandre Pauchet
BO.B.RC.18, alexandre.pauchet@insa-rouen.fr

INSA Rouen Normandie - ISIA departement

Finite-state automata (1/6)

An automatic door controller

Scenario

An automatic sliding door:

- starts closed;
- opens when it detects a person;
- remains open while the passage is occupied;
- closes when the passage becomes clear.

Your task

In pairs, draw a diagram that specifies the behaviour of the door controller.

Show:

- the situations that must be distinguished;
- what causes the door to change from one situation to another.

3 minutes

Safety requirement

If a person is detected while the door is closing, it must reopen immediately.

Does your diagram still specify the correct behaviour?

Finite-state automata (2/6)
Formal definition

Finite-state automata

A *finite-state automaton* (or a *finite state machine*) is a 5-tuple $A = (S, E, T, S_0, F)$ with

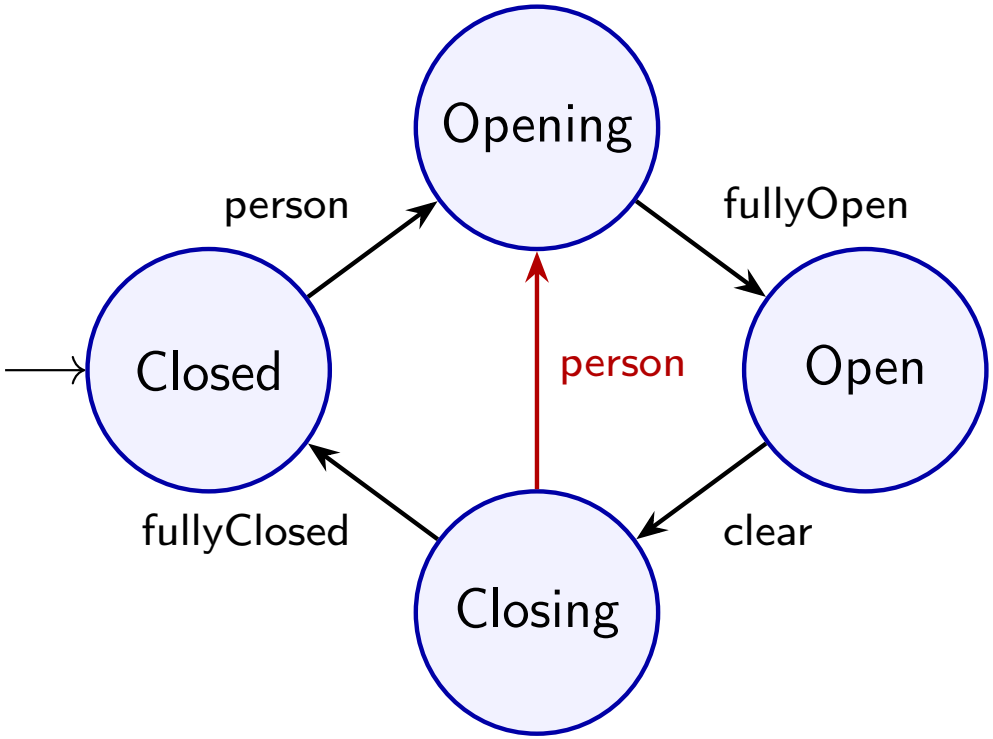
- S a finite set of states,
- E an alphabet,
- T a state-transition function $T : SxE \rightarrow S$,
- $S_0 \subseteq S$ the set of initial states,
- $F \subseteq S$ the set of final states.

Representation

A finite-state automaton can be represented as an oriented graph whose vertices symbolise states and annotated edges describe transitions.

Finite-state automata (3/6)

Formalizing the door controller



Safety transition: the door reopens if it detects a person while closing.

Reactive system

The controller has no final state: it is intended to run continuously.

The 5-tuple

$$S = \{Closed, Opening, Open, Closing\}$$

$$E = \{person, fullyOpen, clear, fullyClosed\}$$

$$S_0 = \{Closed\}$$

$$F = \varnothing$$

$$T \subseteq S \times E \times S$$

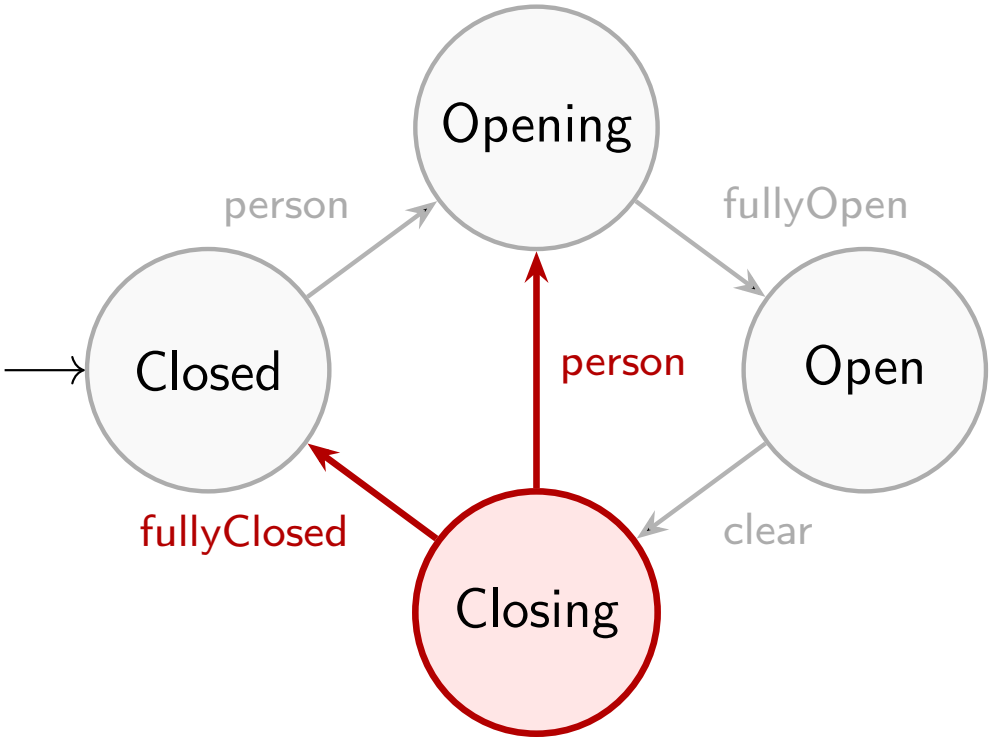
Reading a transition

$$(Closing, person, Opening) \in T$$

When the controller receives *person* while closing, its new state is *Opening*.

Finite-state automata (4/6)

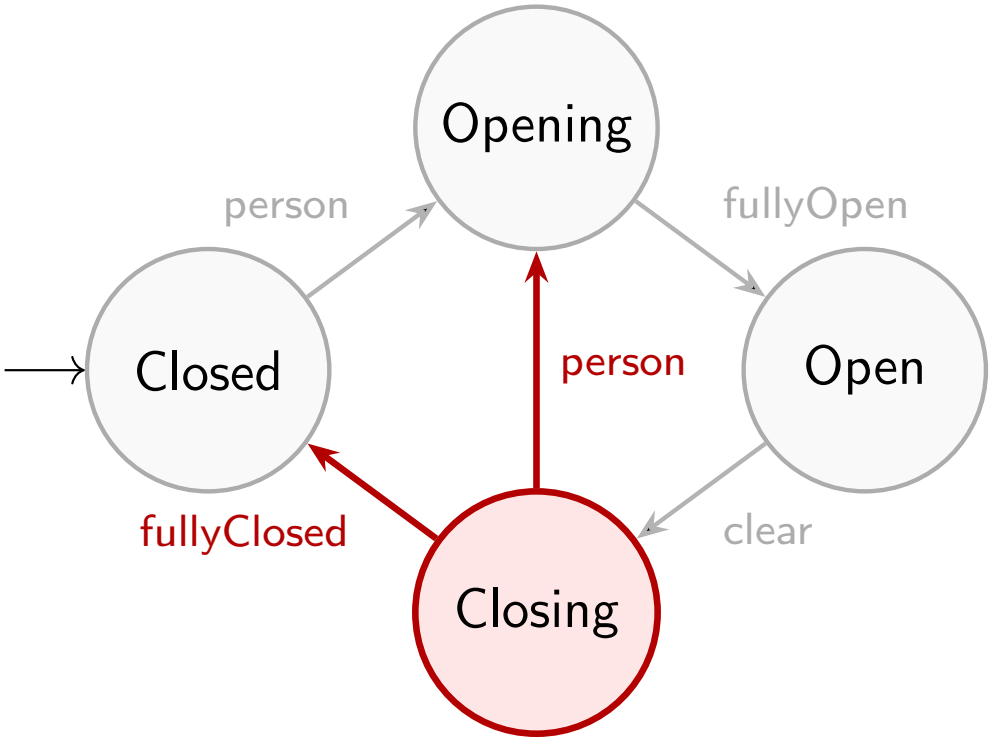
Receptiveness (of the door controller)



Current state: *Closing*

Finite-state automata (4/6)

Receptiveness (of the door controller)



Current state: *Closing*

Absent transition

The controller is not receptive to *fullyOpen* while it is closing: no transition specifies a reaction.

Definition

In a state s, an automaton is *receptive* to an event e if an outgoing transition labelled e exists from s.

$$\text{Enabled}(s) = \left\{ e \in E \mid \exists s' \in S,\ s \xrightarrow{e} s' \right\}$$

Examples

$$\text{Enabled}(\textit{Closing}) = \{\textit{person}, \textit{fullyClosed}\}$$

$$\text{Enabled}(\textit{Closed}) = \{\textit{person}\}$$

Finite-state automata (5/6)
Determinism and execution

Deterministic automaton

A finite-state automaton is *deterministic* if:

- it has exactly one initial state:
$$|S_0| = 1;$$
- a state cannot have 2 transitions with the same event leading to different states:
$$s \xrightarrow{e} s' \quad \text{and} \quad s \xrightarrow{e} s'' \quad \Longrightarrow \quad s' = s''.$$

Execution

An *execution* is a path through the automaton, starting from an initial state:

$$s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} \cdots \qquad \text{with } s_0 \in S_0.$$

Example

One possible execution of the door controller is:

$$\mathit{Closed} \xrightarrow{\mathit{person}} \mathit{Opening} \xrightarrow{\mathit{fullyOpen}} \mathit{Open}$$

Finite state automata (6/6)

Interacting with the environment

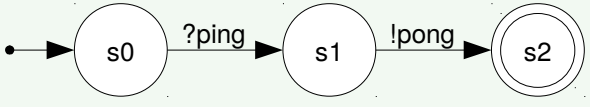
Input/output notation

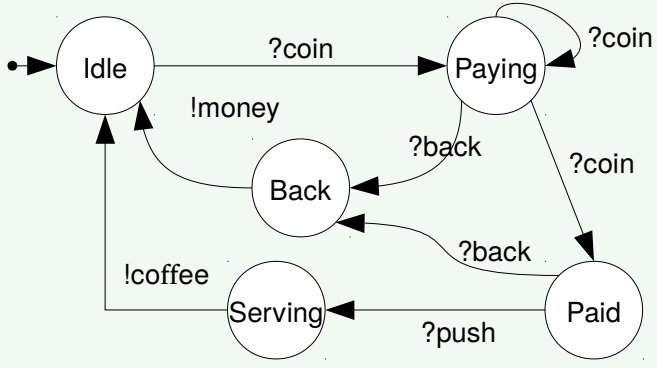
When a state machine (SM) interacts with its environment, transition labels may distinguish:

- ?e an *input event* or message received by the SM;
- !e an *output event* or message emitted by the SM.

The notation is always interpreted from the viewpoint of the SM.

Exemples

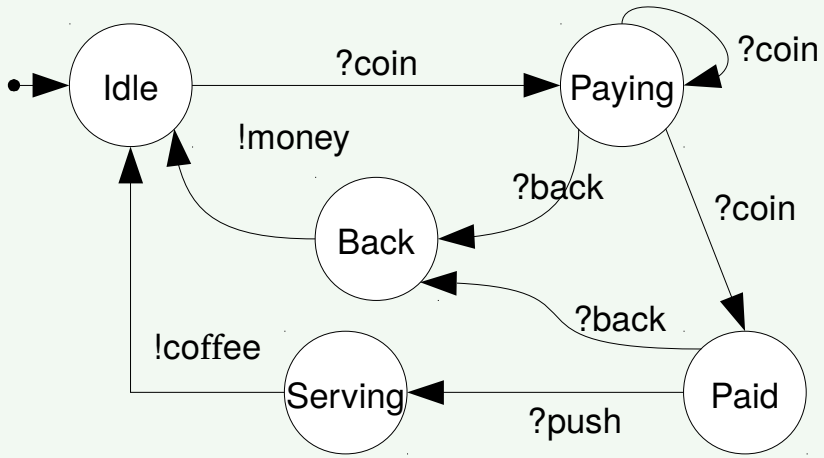




Extended finite-state machines (1/5)

Why states are not always enough

Coffee machine



Questions

Consider the state *Paying*:

- Why is the automaton not deterministic?
- What information is missing?
- What if the machine accepts up to ten coins?

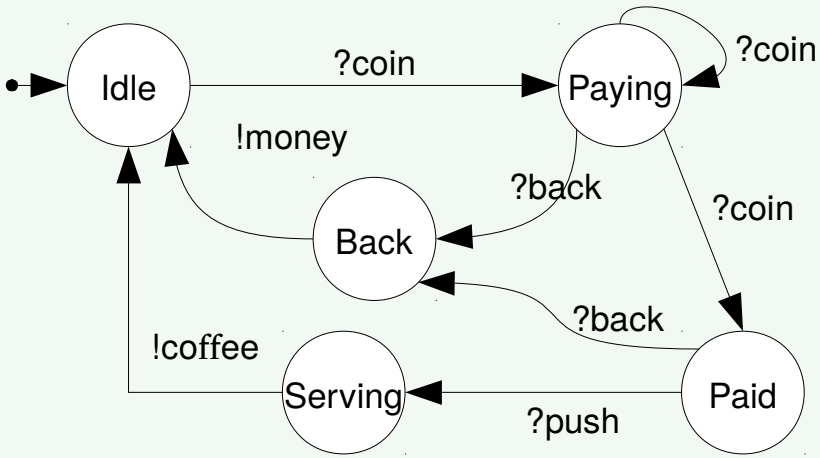
Required behaviour

A coffee costs two coins.
The controller must remember how many coins the user has already inserted.

Extended finite-state machines (1/5)

Why states are not always enough

Coffee machine



Questions

Consider the state *Paying*:

- Why is the automaton not deterministic?
- What information is missing?
- What if the machine accepts up to ten coins?

Required behaviour

A coffee costs two coins.
The controller must remember how many coins the user has already inserted.

Diagnosis

The state *Paying* does not record the current credit.
Encoding every possible credit as a different state would make the automaton grow.

Extended finite-state machines (2/5)

Adding memory to a state machine

Extended finite-state machine

An *EFSM* adds variables to a finite-state machine.

Current configuration:

- a *control state*;
- the current values of its variables.

Extended transition

A transition may contain:

$$event \quad [guard] \quad / \quad action$$

- *event* triggers the transition;
- *guard* must be true;
- *action* updates variables.

Coffee machine

$$credit = 0$$

The state can remain *Paying* while *credit* records the sold.

Examples

Inserting a coin:

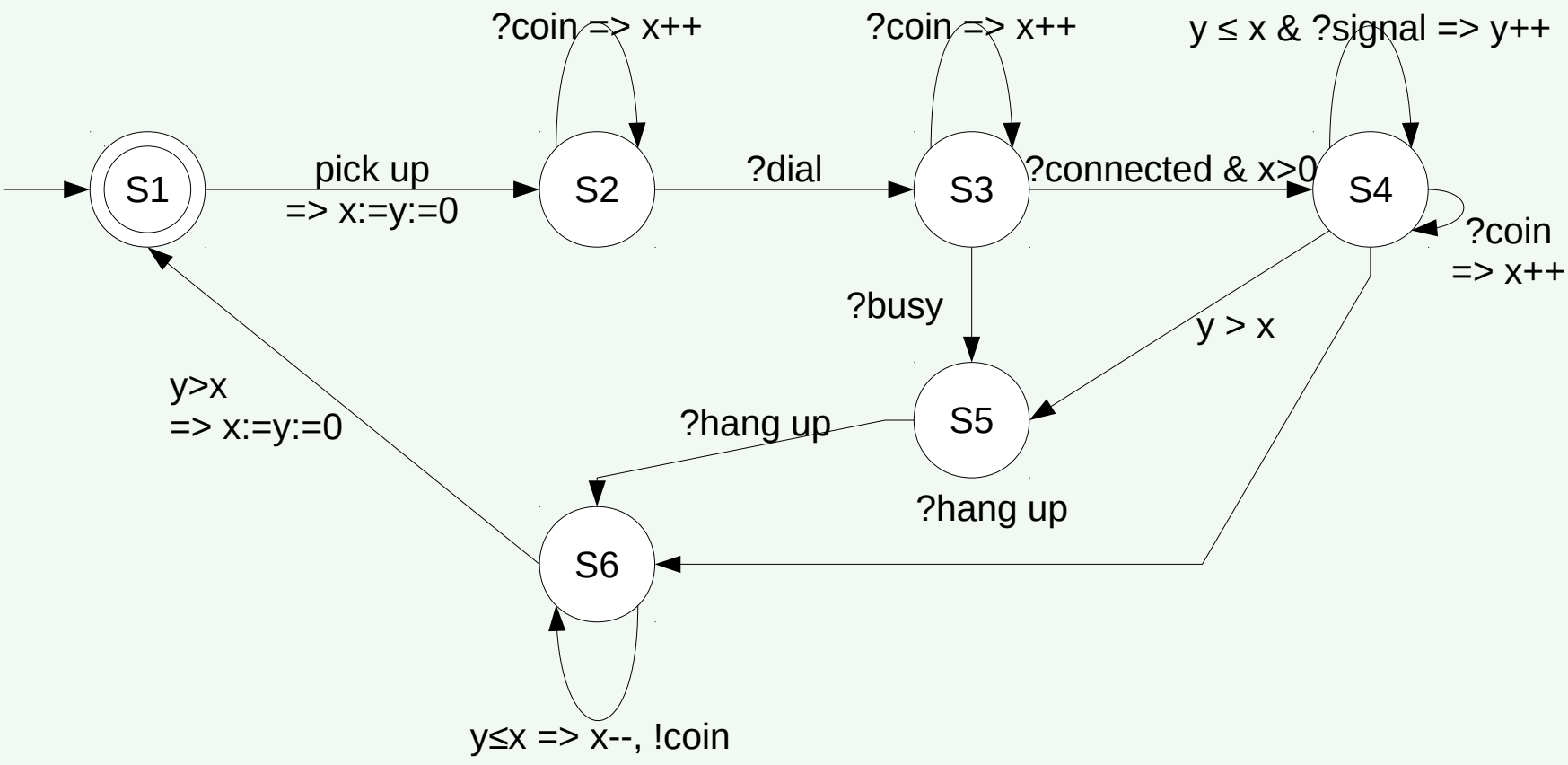
$$?coin \;/\; credit := credit + 1$$

Requesting coffee:

Extended finite-state machines (3/5))

Worked example: a payphone controller

Exemple



Extended finite-state machines (4/5))

Activity: a PIN-controlled door

System requirements

A door is controlled by a PIN code.

- The controller starts in the *Locked* state.
- A correct PIN unlocks the door and resets the number of failed attempts.
- An incorrect PIN increases the number of failed attempts.
- After 3 consecutive incorrect attempts, the controller blocks the door.
- The event `?lock` locks an unlocked door.

Model elements

States: $\{Locked, Unlocked, Blocked\}$
Events: $\{?correctPIN, ?wrongPIN, ?lock\}$
Variable: $attempts := 0$

A blocked controller requires manual intervention, which is not modelled here.

Your task

Draw the extended finite-state machine.

Label its transitions using:

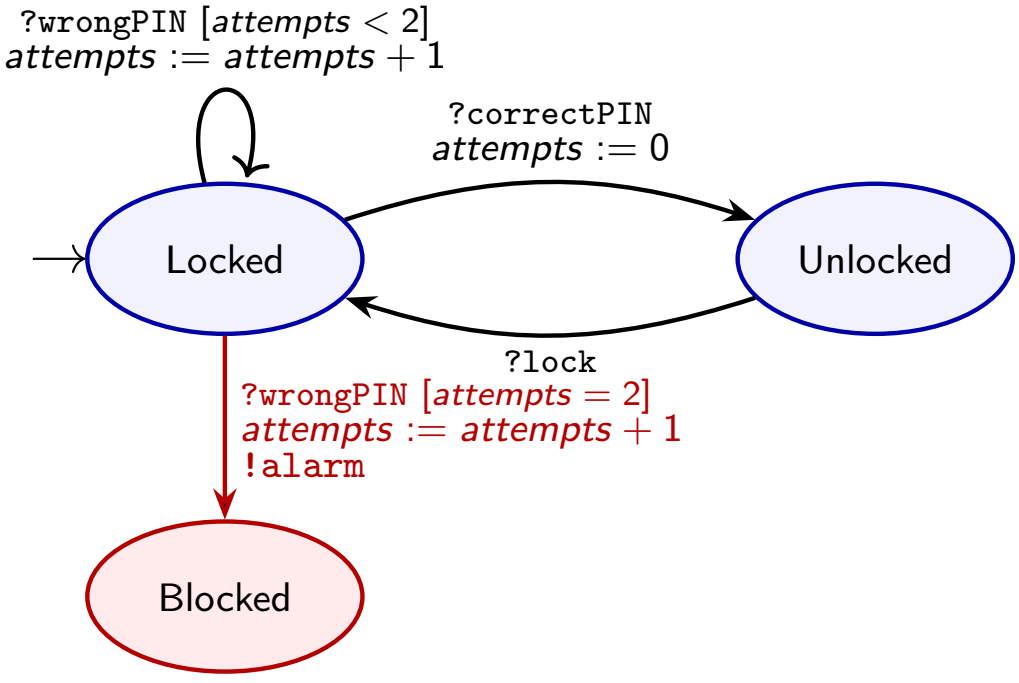
$$event \quad [guard] \quad / \quad action$$

The EFSM should be deterministic.

4 minutes, in pairs

Extended finite-state machines (5/5))

Correction: the PIN-controlled door



Determinism

The two transitions triggered by `?wrongPIN` have mutually exclusive guards:

$$attempts < 2 \quad \text{or} \quad attempts = 2.$$

Therefore, only one of them can be enabled at a time.

Timed automata (1/3)
Why do we need clocks?

Automatic light

An automatic light controller must satisfy the following requirements:

- motion turns the light on;
- the light turns off after 30 seconds without motion;
- new motion restarts the 30-second countdown.

Questions

In pairs, consider the following questions:

- What information must the controller remember?
- How can this information change while no event occurs?

Could we simply use:

$$elapsed := elapsed + 1 \ ?$$

2 minutes

Timed automata (1/3)
Why do we need clocks?

Automatic light

An automatic light controller must satisfy the following requirements:

- motion turns the light on;
- the light turns off after 30 seconds without motion;
- new motion restarts the 30-second countdown.

Limitation of ordinary variables

An ordinary variable changes when a transition is executed.
To measure elapsed time even when no event occurs, the controller needs a *clock*.

Questions

In pairs, consider the following questions:

- What information must the controller remember?
- How can this information change while no event occurs?

Could we simply use:

$$elapsed := elapsed + 1 \ ?$$

2 minutes

Timed automata (2/3)
Clocks, guards and resets

Clock

A *clock* is a non-negative, real-valued variable:

$$x \in \mathbb{R}_{\geq 0}.$$

- It increases at the same rate as time:

$$x(t+\delta) = x(t) + \delta.$$

- A transition may reset it:

$$x := 0.$$

Time can pass

The clock value may change while the control state remains unchanged.

Time constraints

A clock can be used in:

- a transition guard: $[x \geq 30]$;
- a state invariant: $x \leq 30$.

A guard specifies when a transition may occur.
An invariant limits how long the system may remain in a state.

Examples

Reset the clock after detecting motion:

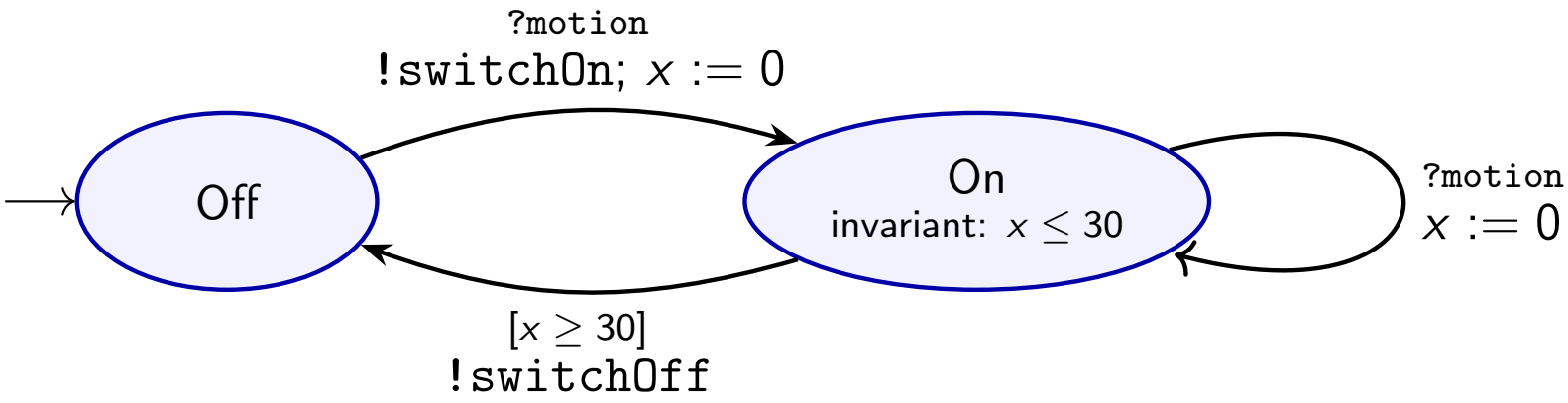
$$?motion \ / \ x := 0$$

Switch the light off after 30 seconds:

$$[x \geq 30] \ / \ !switchOff$$

Timed automata (3/3)

Timed automaton for an automatic light



Exactly 30 seconds

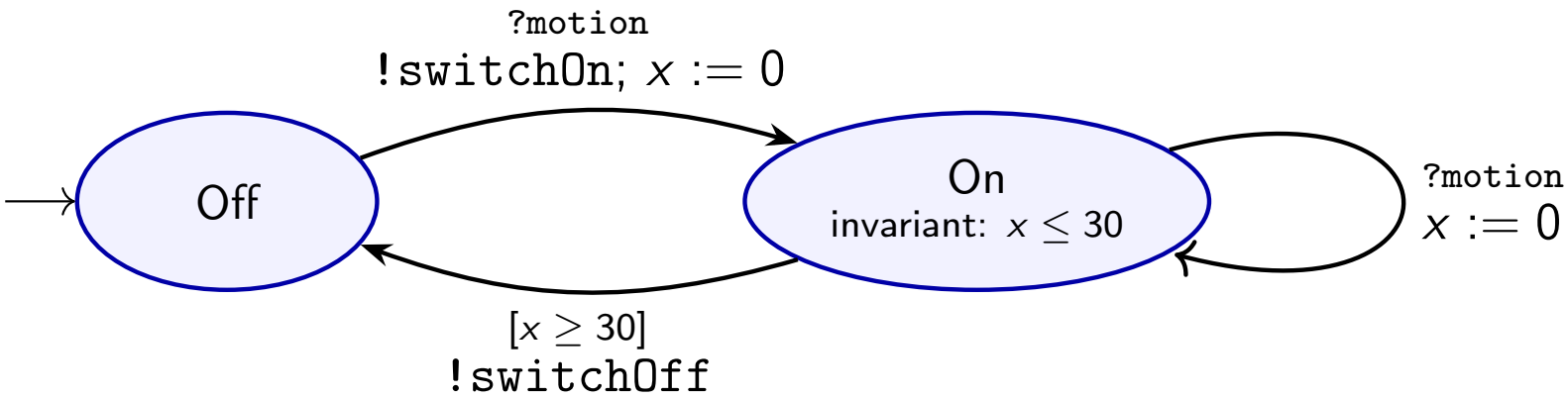
The guard $x \geq 30$ enables the transition after 30 seconds.
The invariant $x \leq 30$ prevents time from progressing beyond 30 seconds in the state *On*.

Quick check

Motion is detected at $t = 0$, then again at $t = 18$.
When does the light turn off?

Timed automata (3/3)

Timed automaton for an automatic light



Exactly 30 seconds

The guard $x \geq 30$ enables the transition after 30 seconds.
The invariant $x \leq 30$ prevents time from progressing beyond 30 seconds in the state *On*.

Quick check

Motion is detected at $t = 0$, then again at $t = 18$.
When does the light turn off?

Answer: $t = 48$ **seconds.**

Conclusion (1/3)
Limitations of finite-states automata

Limitations and solutions

1 automaton = 1 entity of the system

- How to coordinate 2 entities?
  ⇒ Communication between automata (*synchronous*/*asynchronous*)
- Is it possible to introduce some kind of memory?
  ⇒ Hybrid automata
- How to represent time-related transitions?
  ⇒ Timed automata
- How to represent a complex system?
  ⇒ Automata network, *Petri* network
- Automatised construction of automata?
  ⇒ Supervised/unsupervised learning

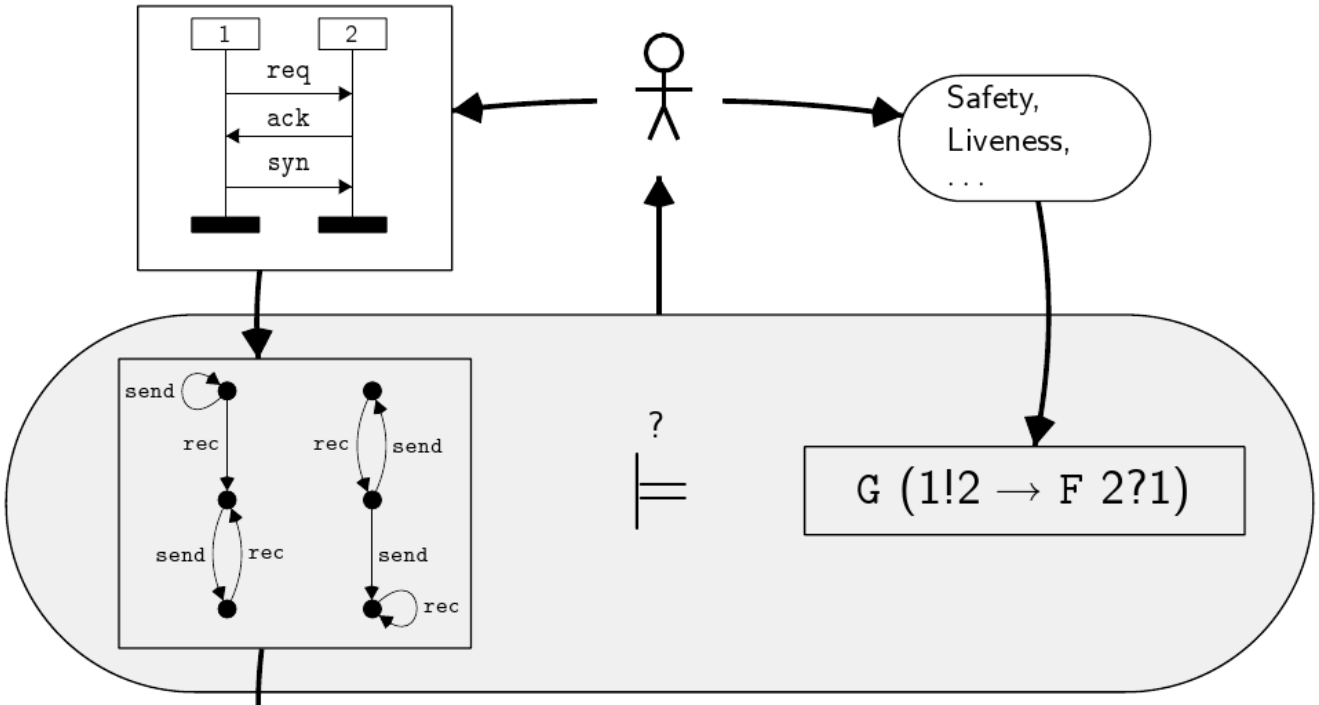
Conclusion (2/3)
Modelling and verification

specification

synthesis

modeling & verification

code generation




```
/* Sending Process */
initsend();
putint(2);
send("receiver",4,99);

/* Receiving Process */
int num;
recv(99);
getint(&num);
```

From Benedikt Bollig's lesson http://www.lsv.ens-cachan.fr/~bollig/mpri/2007/decembre10.pdf.

Conclusion (3/3)
References

Links

- http://www.pps.jussieu.fr/~rifflet/enseignements/AF3/
- http://www-igm.univ-mlv.fr/~desar/Cours/automates/ch1.pdf
- http://mpri-wiki.inria.fr/bin/view/WebSite/2007-2008-C-2-8
- http://lifc.univ-fcomte.fr/~julliand/CoursAutomates.pdf
- http://www-master.ufr-info-p6.jussieu.fr/2005/IMG/pdf/Automates_tempo.pdf
- http://www.lamsade.dauphine.fr/~berard/PDF/MCT.pdf
- http://www.loria.fr/~bournez/cours/CoursDEAModelChecking/10-N-AutomatesTemporises.pdf

Article

- "*A theory of timed automata*", R. Alur and D. L. Dill, in Theoretical computer science, volume 126, pp. 183-235, 1994.