CIS 4930/6930: Principles of Cyber-Physical Systems

Chapter 4: Hybrid Systems

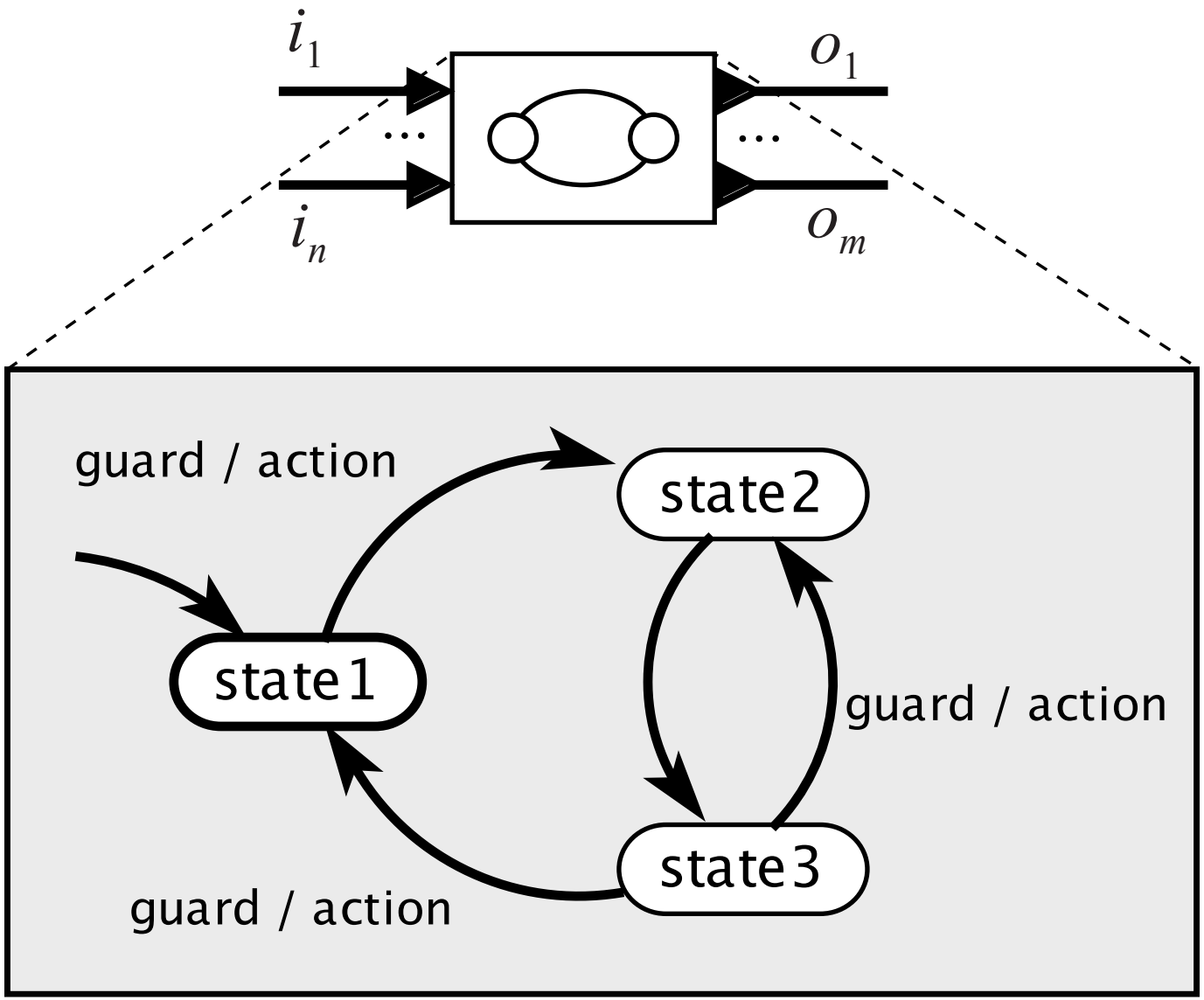
Hao Zheng

Department of Computer Science and Engineering
University of South Florida

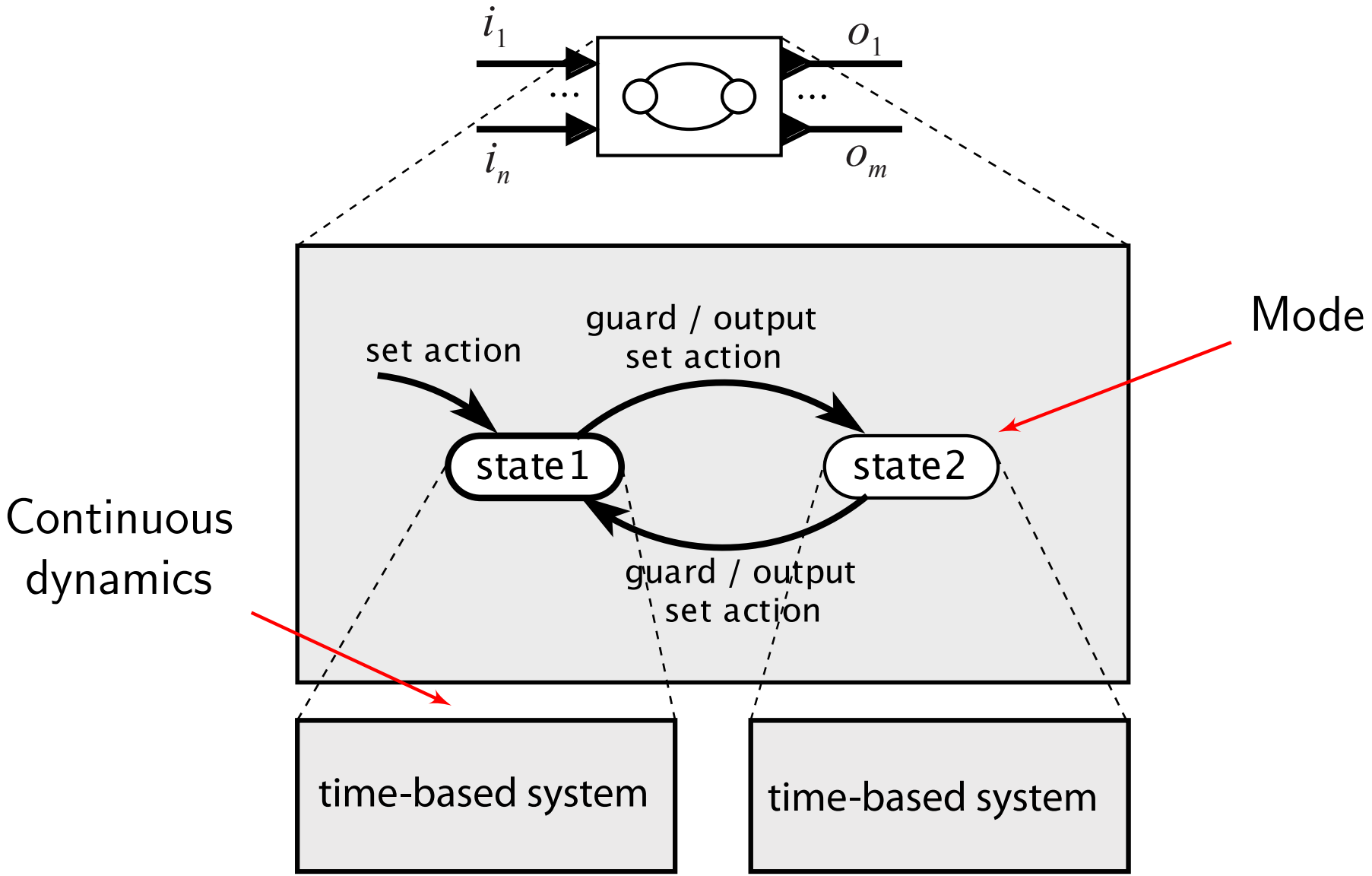
Hybrid Systems

- Differential equations are used to model continuous dynamics.
- State machines are used to model discrete dynamics.
- Cyber-physical systems are *hybrid systems* that include both continuous and discrete dynamics.
- *Hybrid system models* must represent continuous and discrete dynamics.

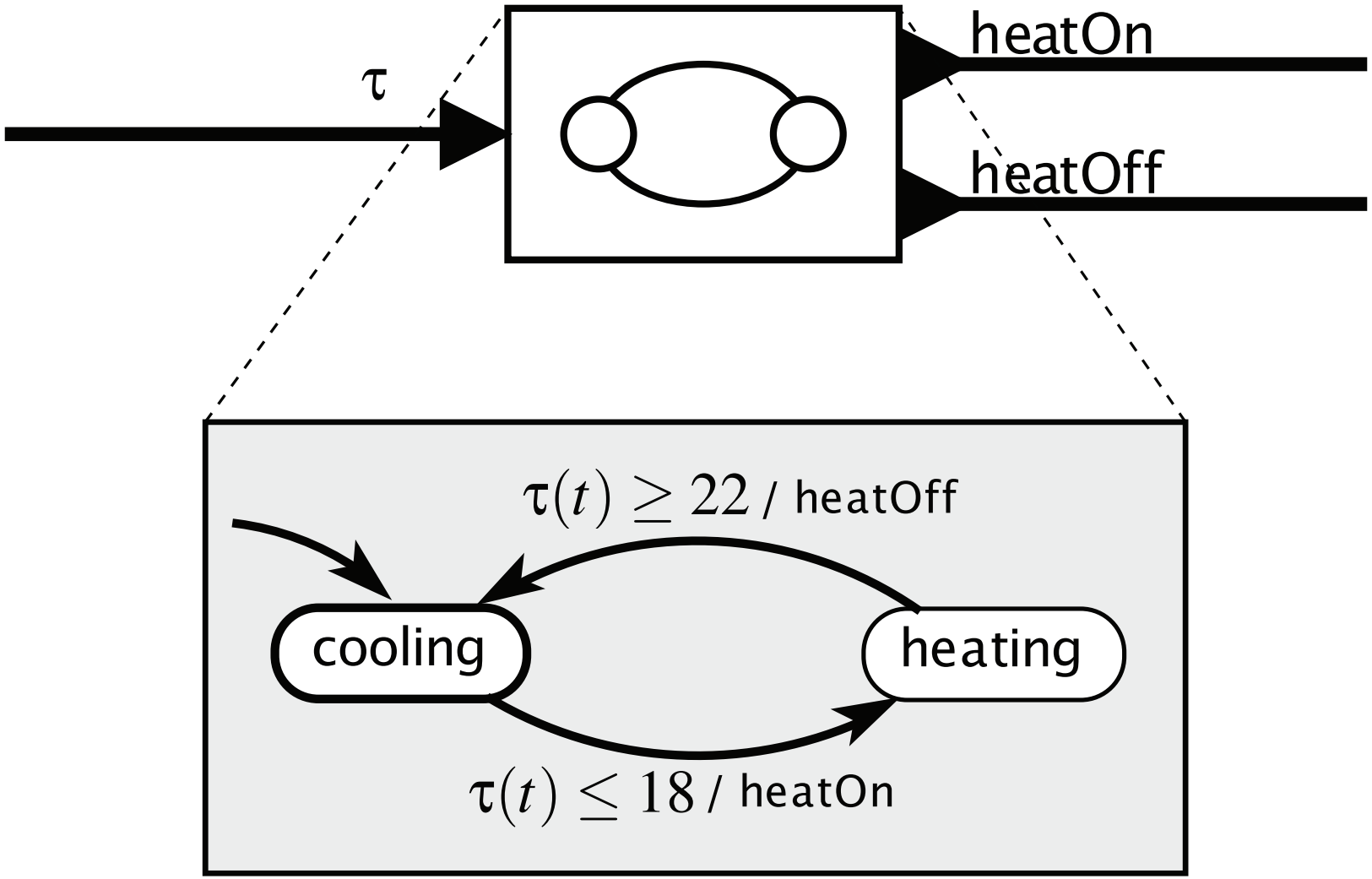
FSM Model


i_1
i_n
o_1
o_m
guard / action
state2
state1
guard / action
state3
guard / action

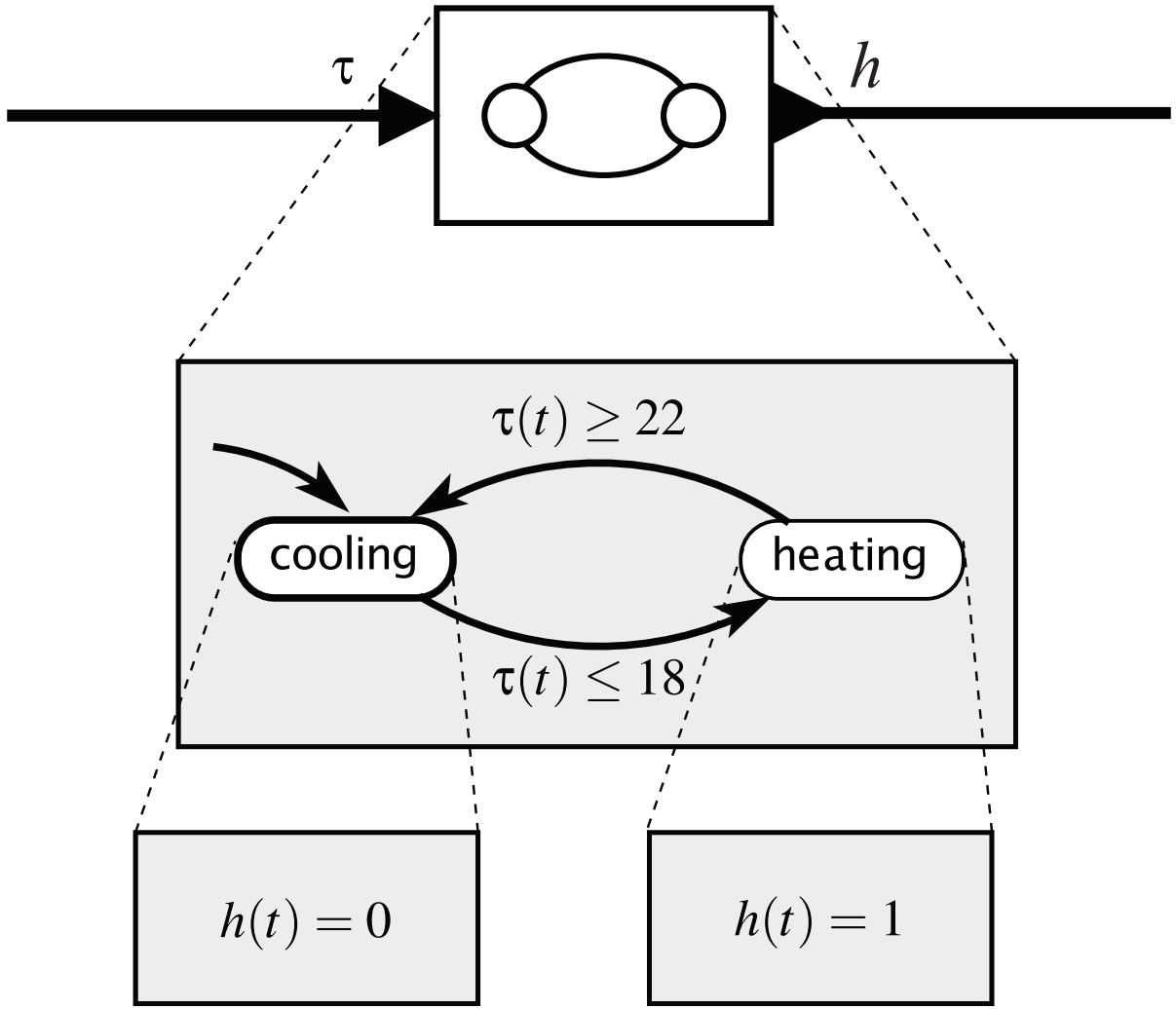
Hybrid System, or Modal Model



A Thermostat Model with a Continuous-Time Input Signal



A Thermostat Model with a Continuous-Time Output Signal



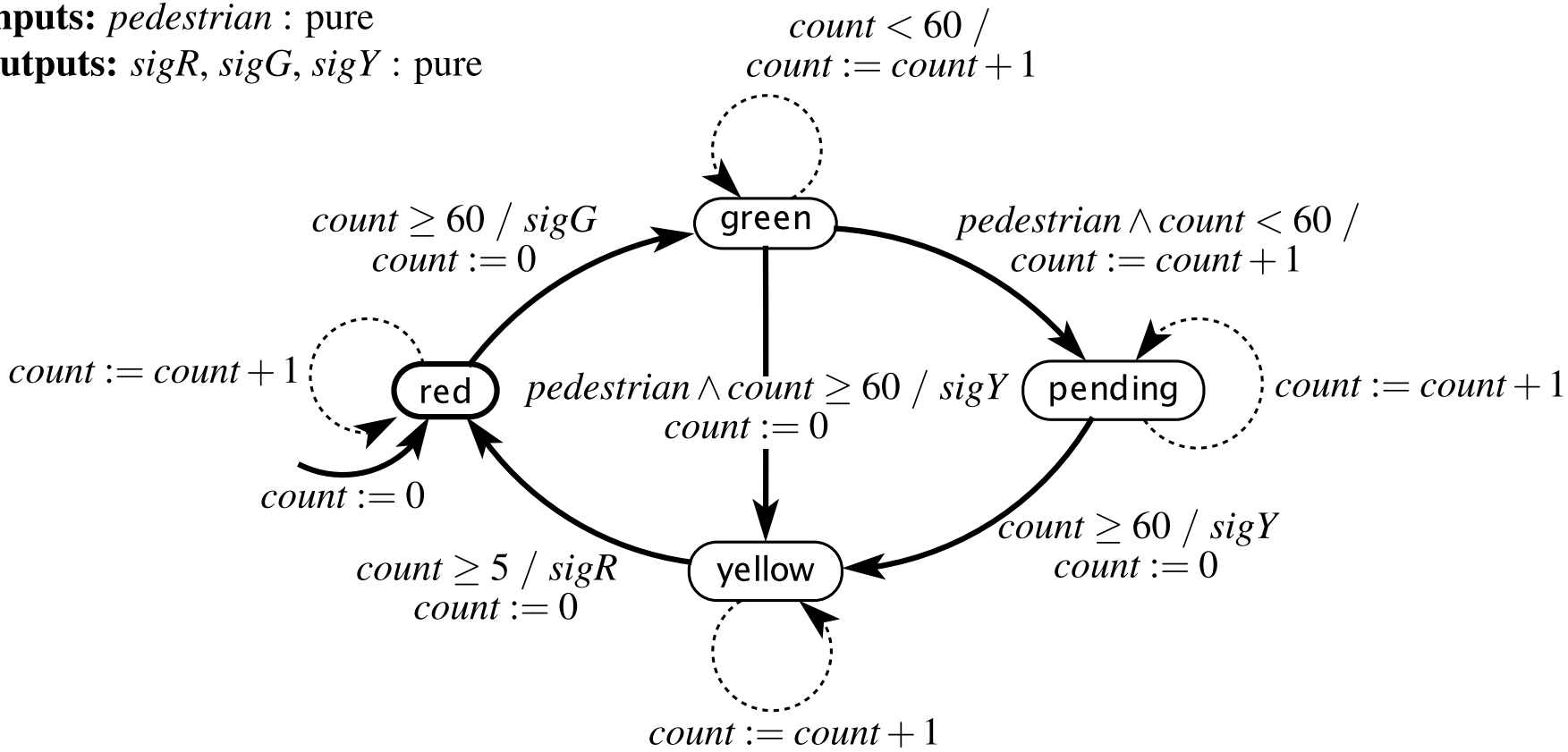
Timed Automata: Modeling and Analysis

Motivation

- In time-critical systems, reactions must happen not only correctly but also timely.
- Applications:
  - ABS in cars
  - Traffic control
  - Flight control
- How is time modeled? Discrete or continuous?

Discrete Modeling of Time

variable: $count$: $\{0, \cdots, 60\}$
inputs: $pedestrian$: pure
outputs: $sigR$, $sigG$, $sigY$: pure



- Time is represented as multiples of basic units.
- Leads to large state space.

Timed Automata: Overview

- Time automata = FSMs extended with clock variables.
- Clocks are dynamic variables that progress linearly in time.

$$\forall t \in T_m, \quad \dot{s}(t) = 1$$

- $s : \mathbb{R} \to \mathbb{R}$ is a continuous-time signal,
- $s(t)$ is the value of the clock at time t,
- All clocks progress synchronously.

Timed Automata: Syntax

A timed automata is defined with (ignoring discrete variables)

- L: a finite set of locations.
- $l_0 \in L$: the initial location.
- C: a finite set of clock variables.
- A: a finite set of actions.
- E: a finite set of edges connecting locations.
- I: location invariants.

For each $e \in E$, $e = (l_1, \alpha, cc, reset, l_2)$ where

- $\alpha \in A$ is an action,
- $cc \in B(C)$ is a clock constraint,
- $reset \subset C$ is a subset of clocks to reset to 0.

Timed Automata: Clock Constraints: Syntax

- In timed automata, only two operations can be applied to clocks
  - It is reset to 0, or
  - its value can read and tested for some condition.
- Atomic clock constraints on clock variables $x, y \in C$,

$$x \bowtie \mathbf{c} \quad \text{or} \quad x - y \bowtie \mathbf{c}$$

  where $\mathbf{c}$ is a rational constant, and $\bowtie \in \{<, \leq, >, \geq\}$.
- Clock constraints $B(C)$ is a set of conjunctions over the atomic clock constraints.
- Examples:

$$x = \mathbf{c} \quad \equiv \quad (x \leq \mathbf{c}) \wedge (x \geq \mathbf{c})$$

Timed Automata: Clock Constraints: Semantics

Given a clock $x \in C$, let $u : C \to \mathbb{R}_{\geq 0}$ be an assignment of non-negative real numbers to clocks in C, and $u(x)$ return the value of $x \in C$.

$$
\begin{array}{lll}
u \models x \bowtie \mathbf{c} & \textit{iff} & u(x) \bowtie \mathbf{c} \\
u \models x - y \bowtie \mathbf{c} & \textit{iff} & u(x) - u(y) \bowtie \mathbf{c} \\
u \models cc_1 \wedge cc_2 & \textit{iff} & u \models cc_1 \wedge u \models cc_2
\end{array}
$$

Example:

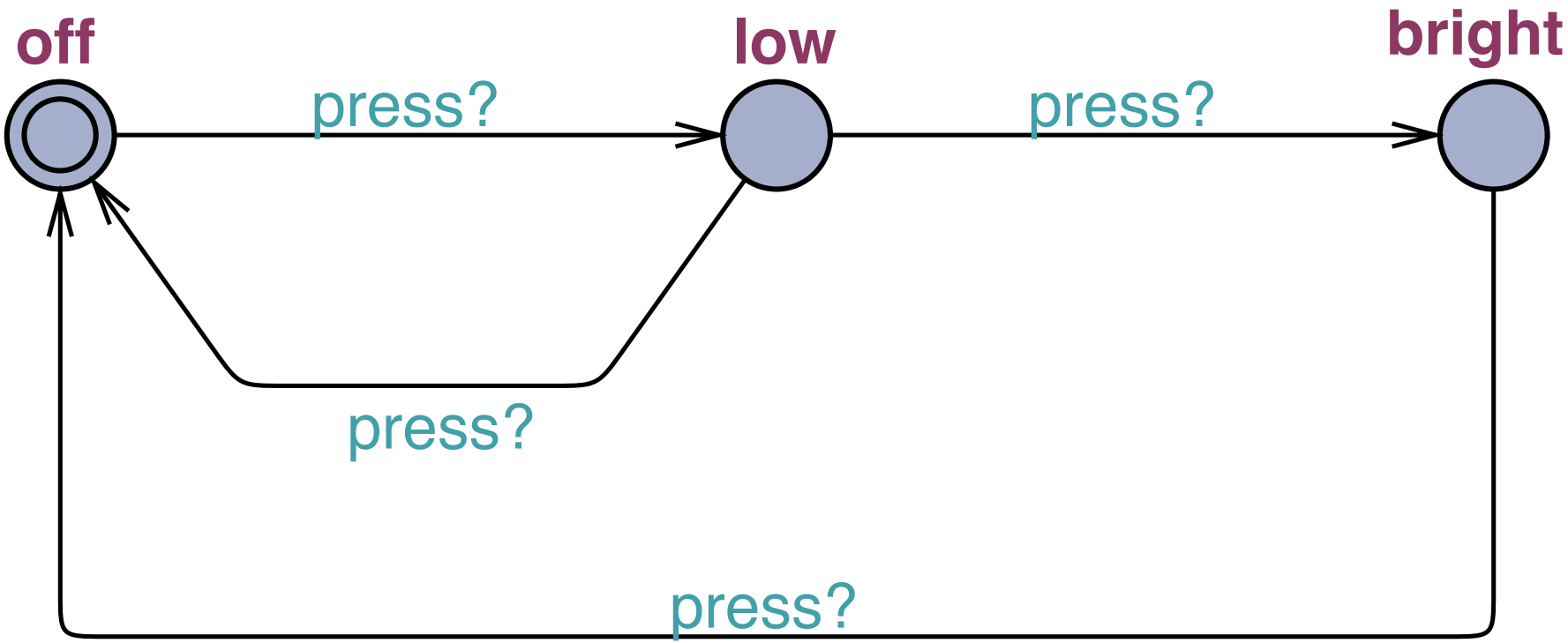
$$x = 0.5, y = 1.39 \models (x < 1) \wedge (y \leq 5)$$

while

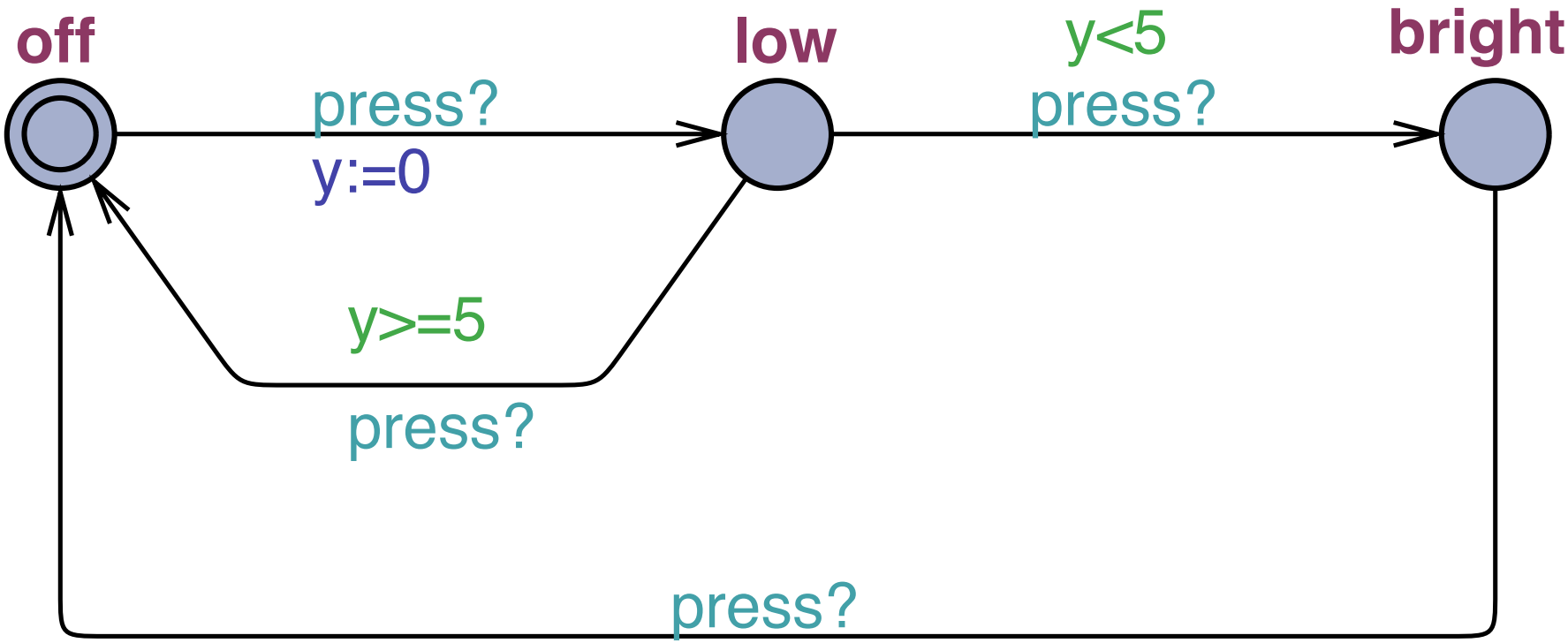
$$x = 1.01, y = 1.39 \not\models (x < 1) \wedge (y \leq 5)$$

Timed Automata: An Example

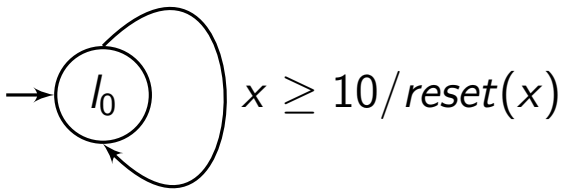
A lamp has a button. When the button is pushed once, the lamp lights on at the **low** level. When the button is pushed twice in a row, the lamp lights on at the **bright** level. In either level, the lamp lights off when the button is pushed again.

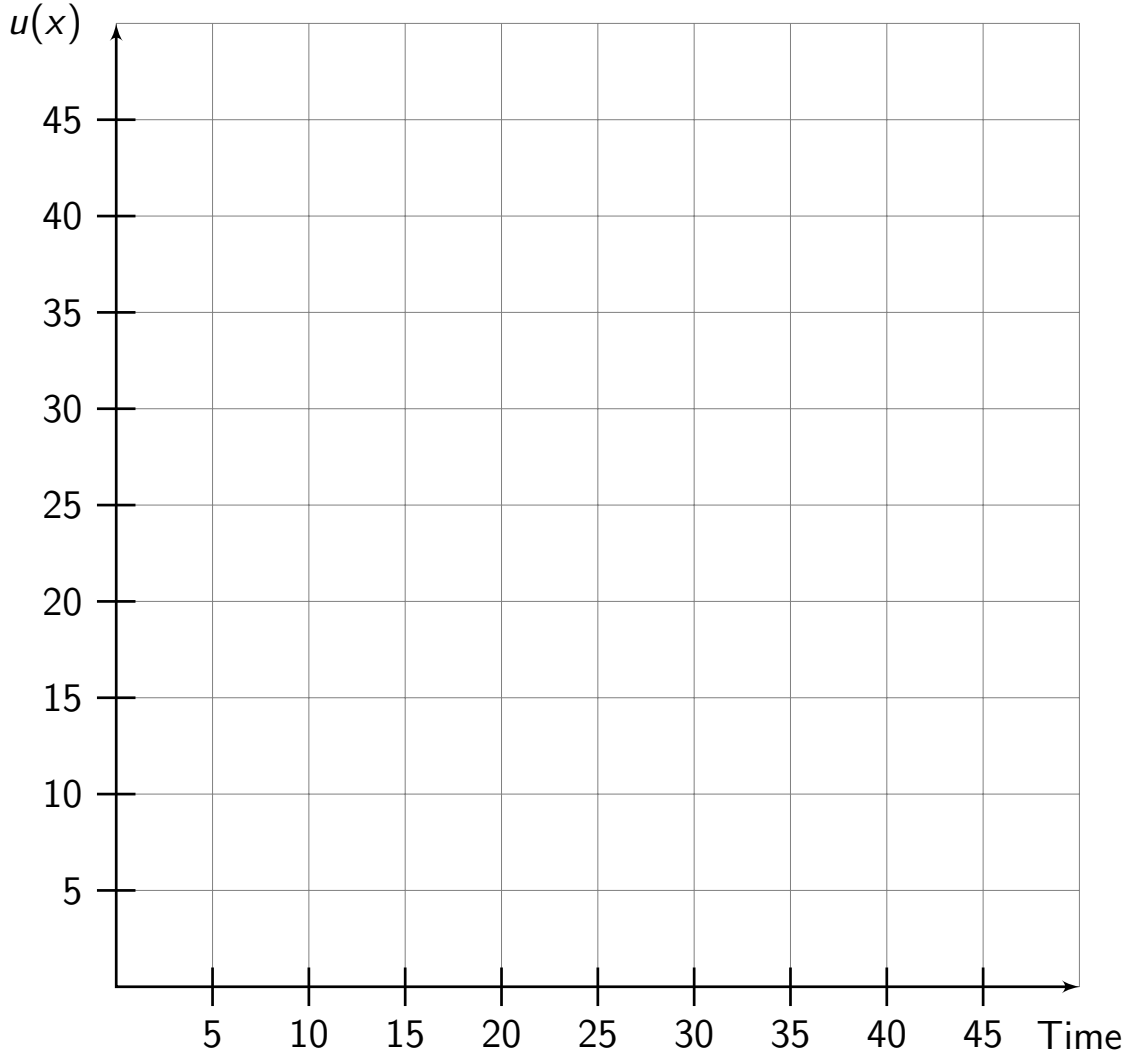


Timed Automata: An Example

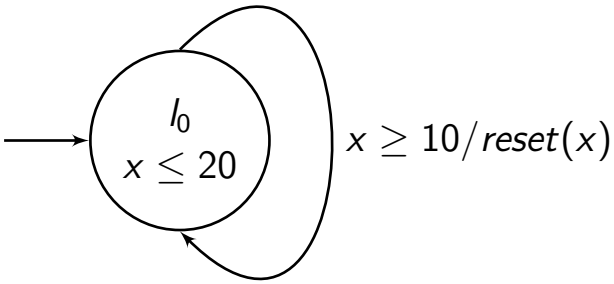


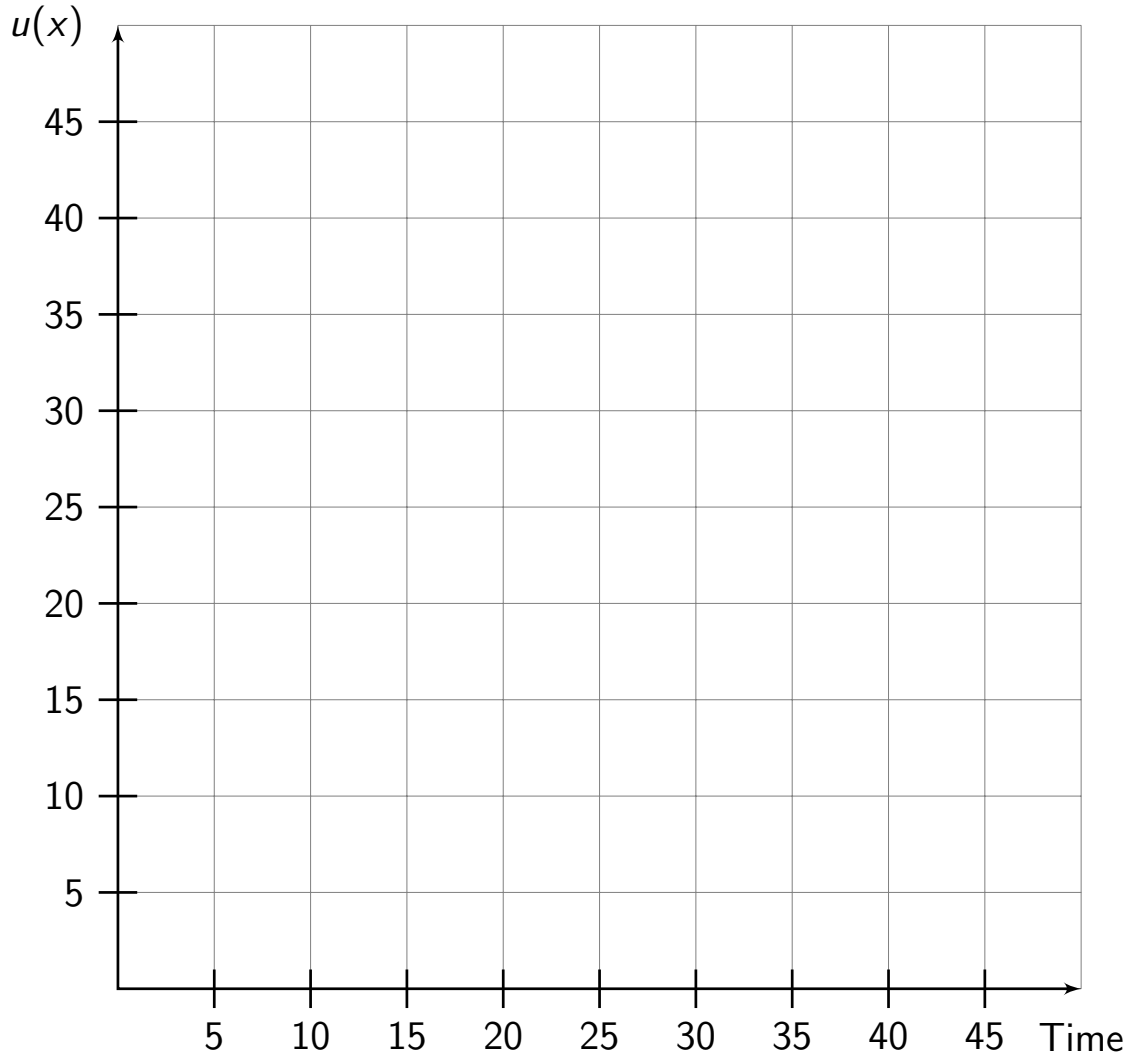
Timed Automata: Understand Time



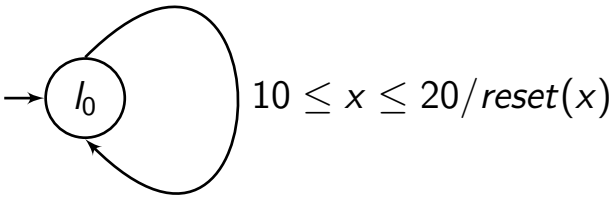


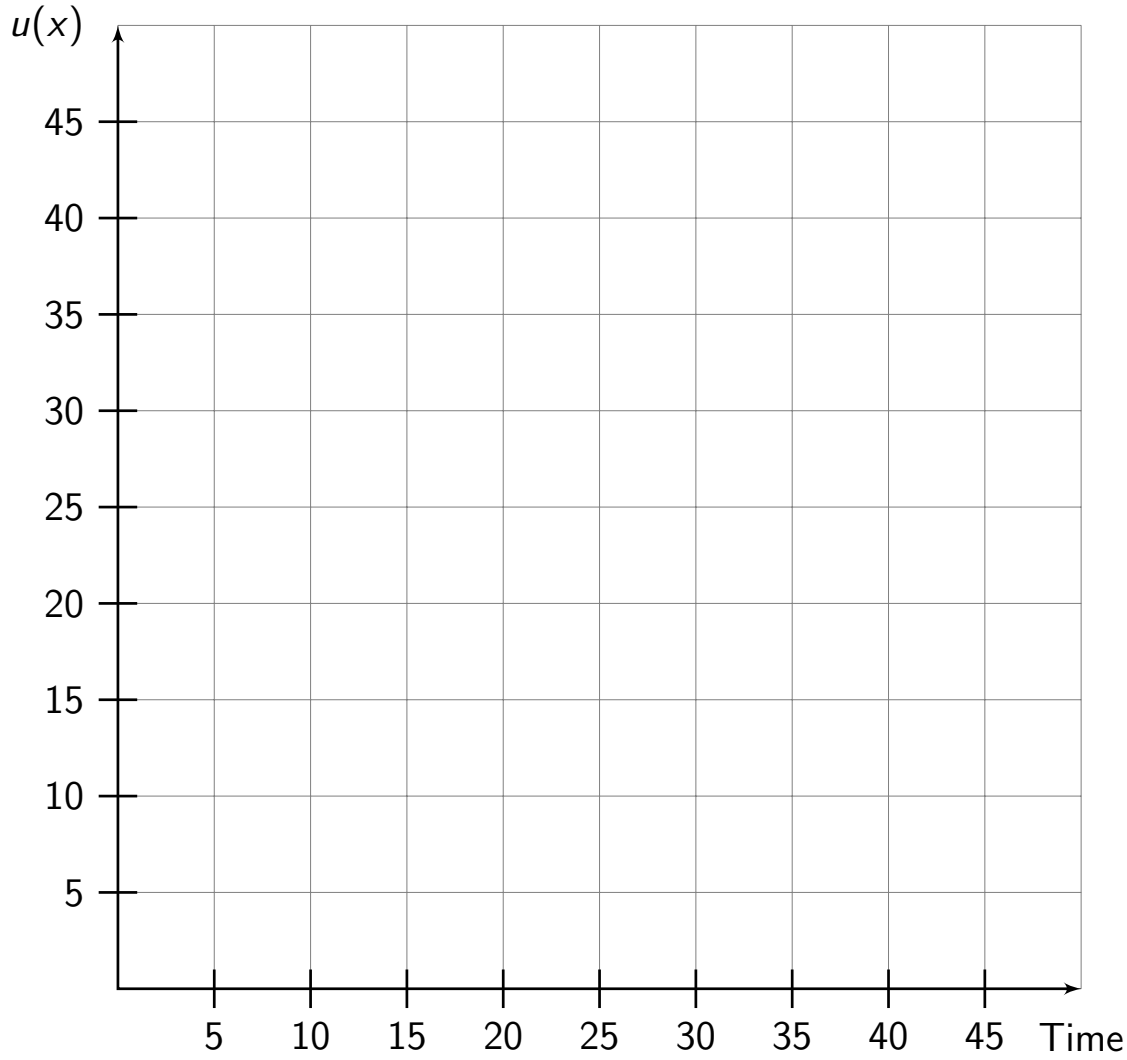
Timed Automata: Understand Time





Timed Automata: Understand Time





Timed Automata: Semantics

Transitions

- A state of a timed automata is (l, u).
- The initial state is (l_0, u_0).

Discrete transition: $(l_1, u_1) \xrightarrow{e} (l_2, u_2)$

- An edge $(l_1, \alpha, cc, reset, l_2) \in E$ is enabled/executable in a state (l, u) if
  - $l = l_1$, $u \models cc$, and
  - there is a matching synchronization action to a.
- A new state (l', u') after executing e such that

  $l' = l_2$, u' is the same as u except all clocks in *reset* reset to 0.

Delay transition: $(l, u_1) \xrightarrow{\delta} (l, u_2),\ \delta \in \mathbb{R}^+$

$u_2 = u_1 + \delta$ where $u_1 + \delta$ means $u(x) + \delta$ for every $x \in C$.

Execution Traces

- Execution step: $\rightarrow = \xrightarrow{e} \cup \xrightarrow{\delta}$
- Execution trace:

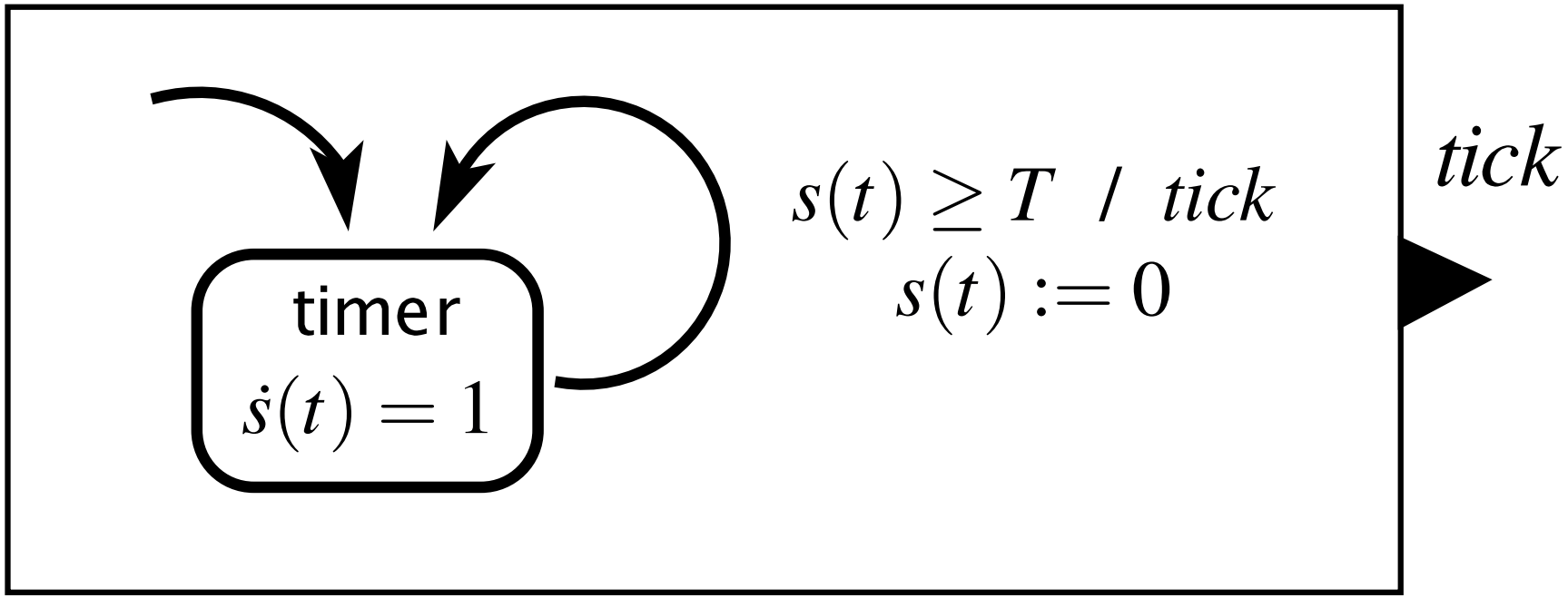
$$(l_0, u_0) \rightarrow (l_1, u_1) \rightarrow (l_2, u_2) \ldots$$

- Reachability: (i, u) is reachable if there exists a trace

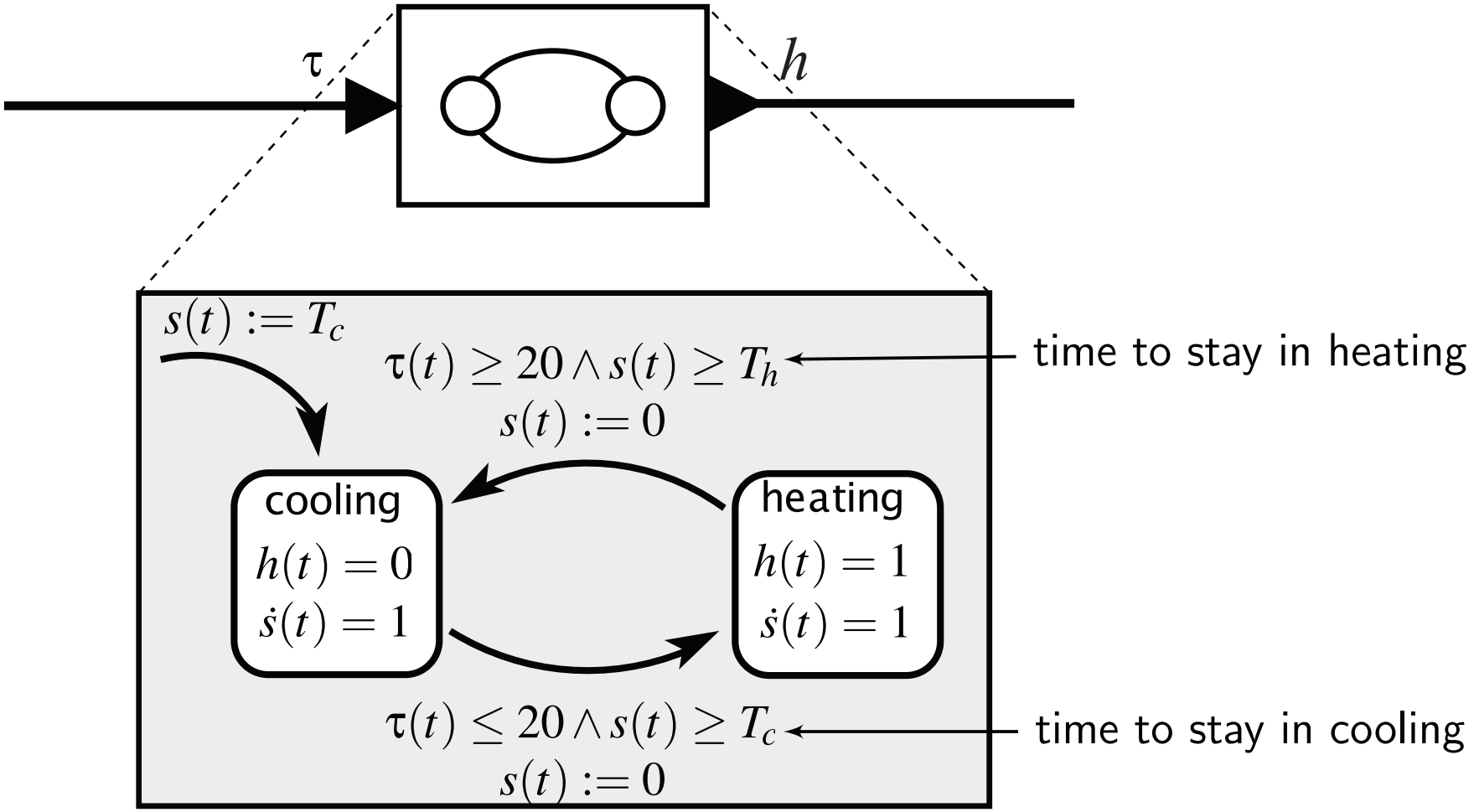
$$(l_0, u_0) \rightarrow (l_1, u_1) \ldots \rightarrow (l_n, u_n)$$

such that $l = l_n$ and $u = u_n$.

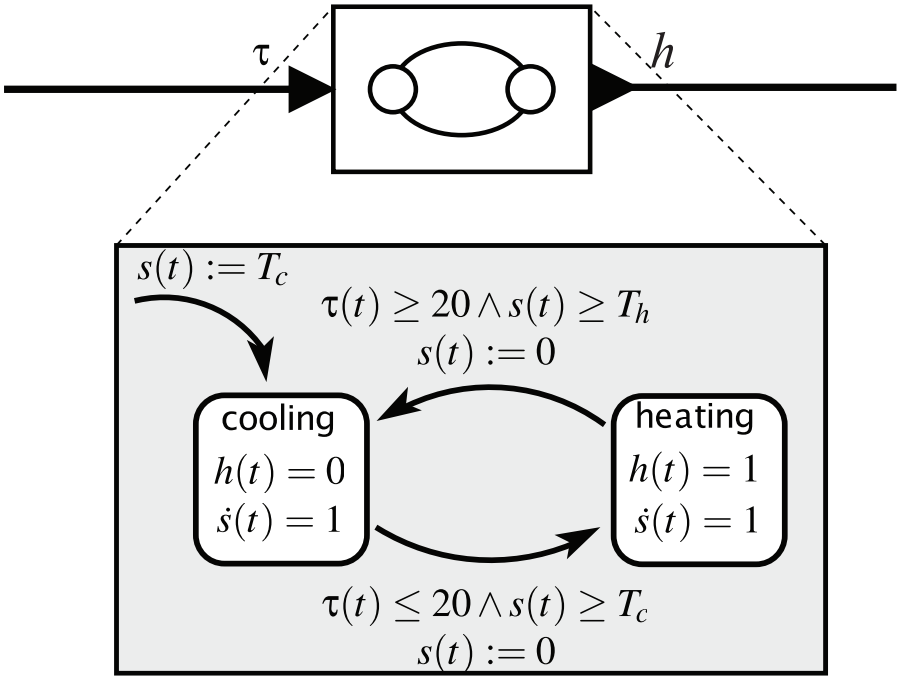
A Timed Automaton that Generates a Pure Output



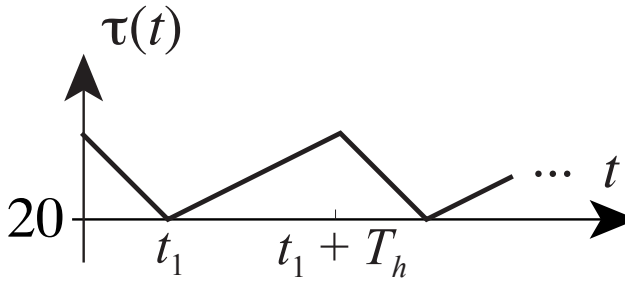
Timed Automaton Model of a Thermostat



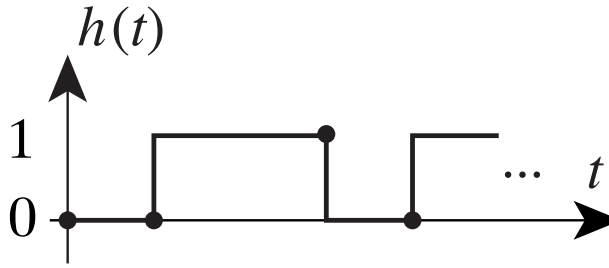
Possible Execution of the Timed Thermostat Model



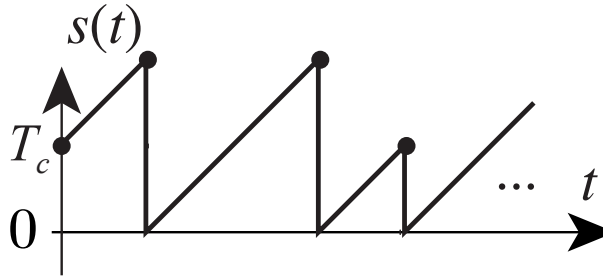
(a)



(b)

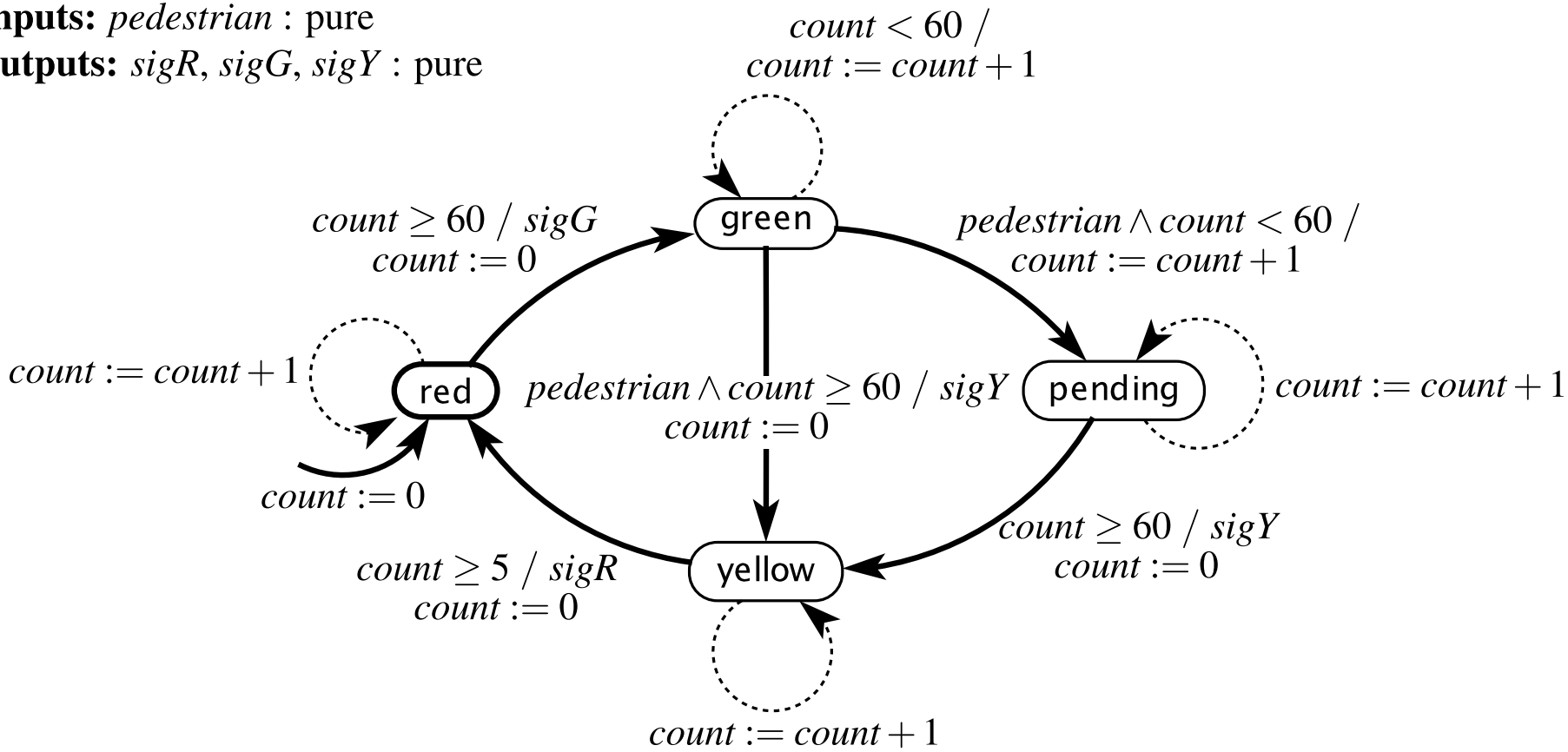


(c)



FSM for the Traffic Light Controller

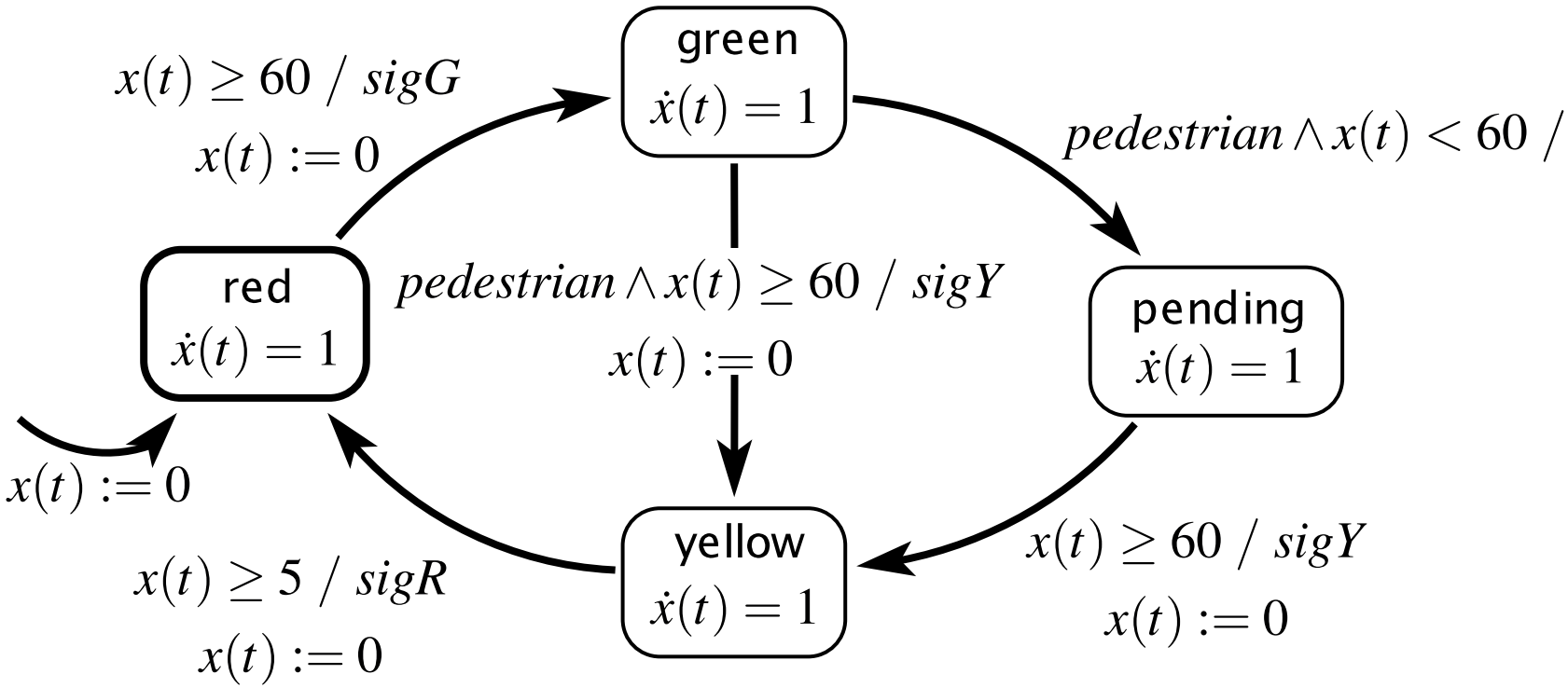
variable: *count*: $\{0, \cdots, 60\}$
inputs: *pedestrian* : pure
outputs: *sigR*, *sigG*, *sigY* : pure



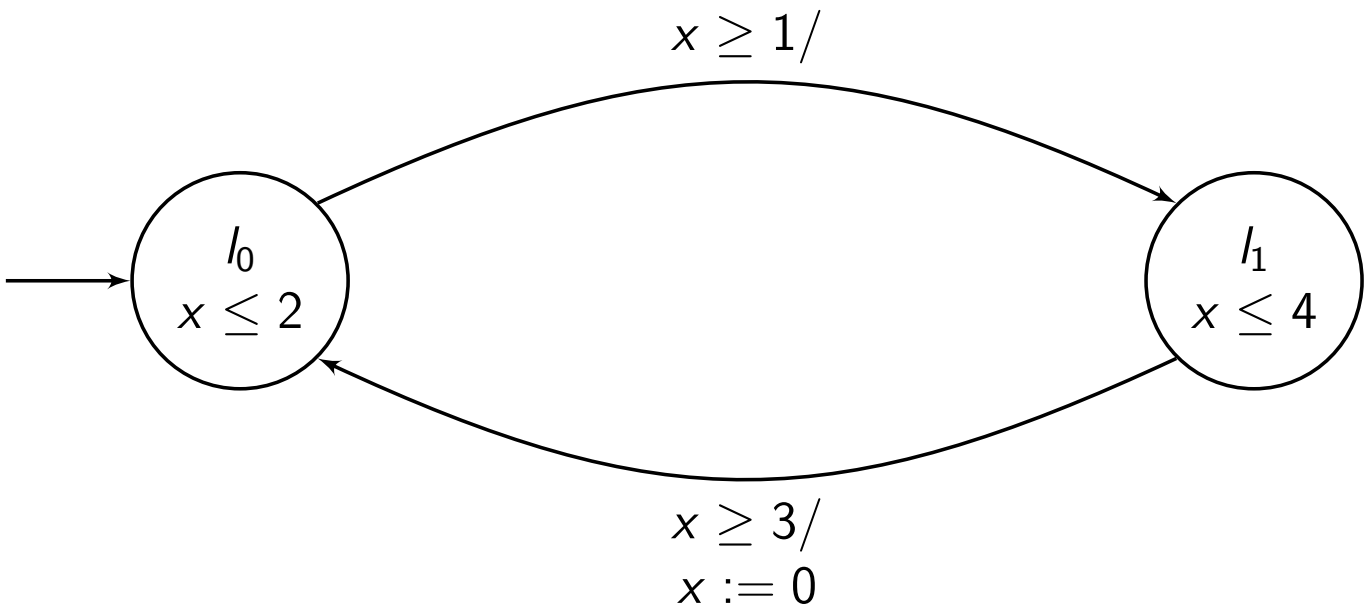
It reacts in every second.

Timed Automaton for the Traffic Light Controller

continuous variable: $x(t) : \mathbb{R}$
inputs: *pedestrian* : pure
outputs: *sigR*, *sigG*, *sigY* : pure

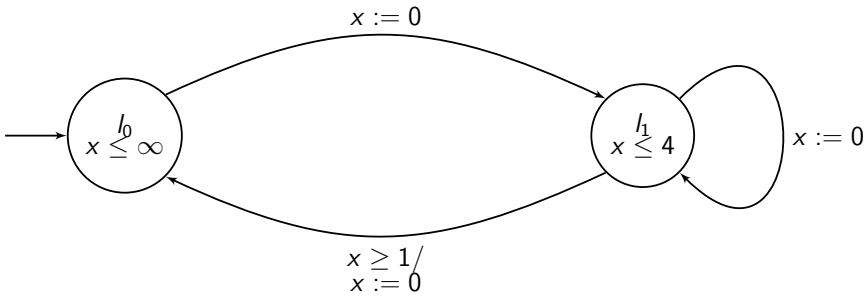


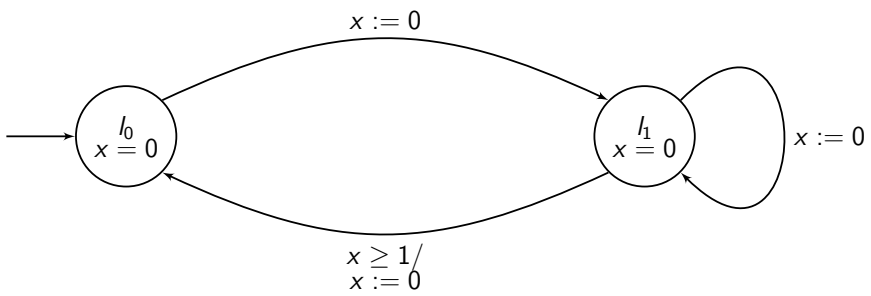
Timed Automaton: Exercise



Zenoness

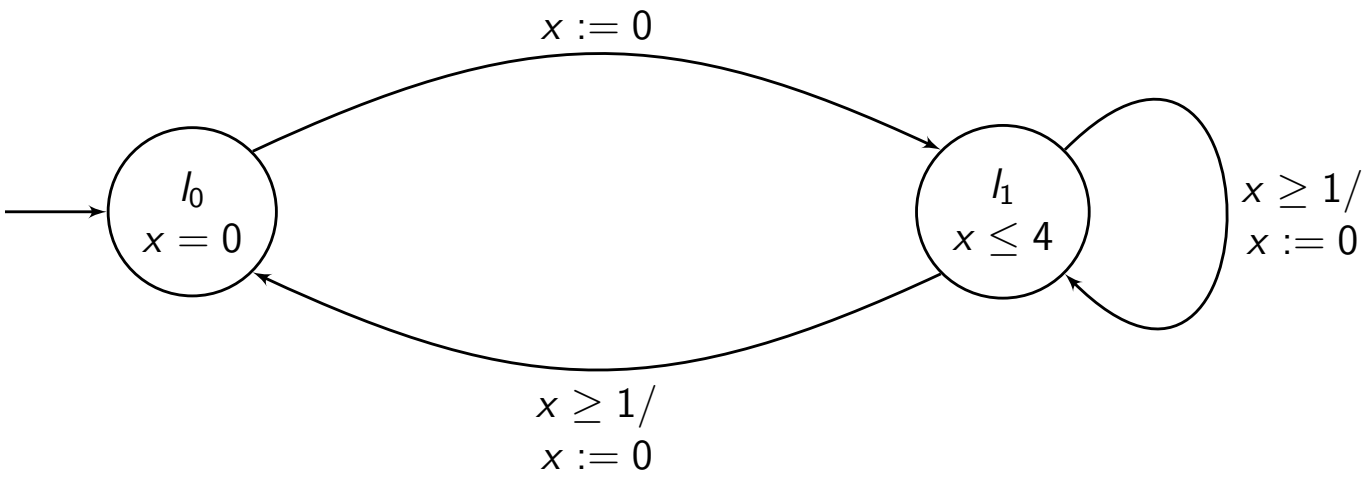
A **zeno** trace of a timed automata has infinite number of discrete transitions within finite amount of time.





Eliminate Zenoness

Make sure that time can progress on every cycle in timed automata.



Composing Timed Automata

Parallel Composition of Timed Automata

Two TAs $T_1 = (L_1, l_{10}, C_1, A_1, E_1, Inv_1)$ and $T_2 = (L_2, l_{20}, C_2, A_2, E_2, Inv_2)$ such that $C_1 \cap C_2 = \emptyset$, their parallel composition, $T_1 \| T_2$ is a TA (L, l_0, C, A, E, Inv) where

- $L = L_1 \times L_2$,
- $l_0 = (l_{10}, l_{20})$;
- $C = C_1 \cup C_2$,
- $A = A_1 \cup A_2$,
- $Inv = I_1(l_1) \wedge I_2(l_2)$ for all $(l_1, l_2) \in L$,
- $E = \{\ldots\}$,

Parallel Composition of Timed Automata

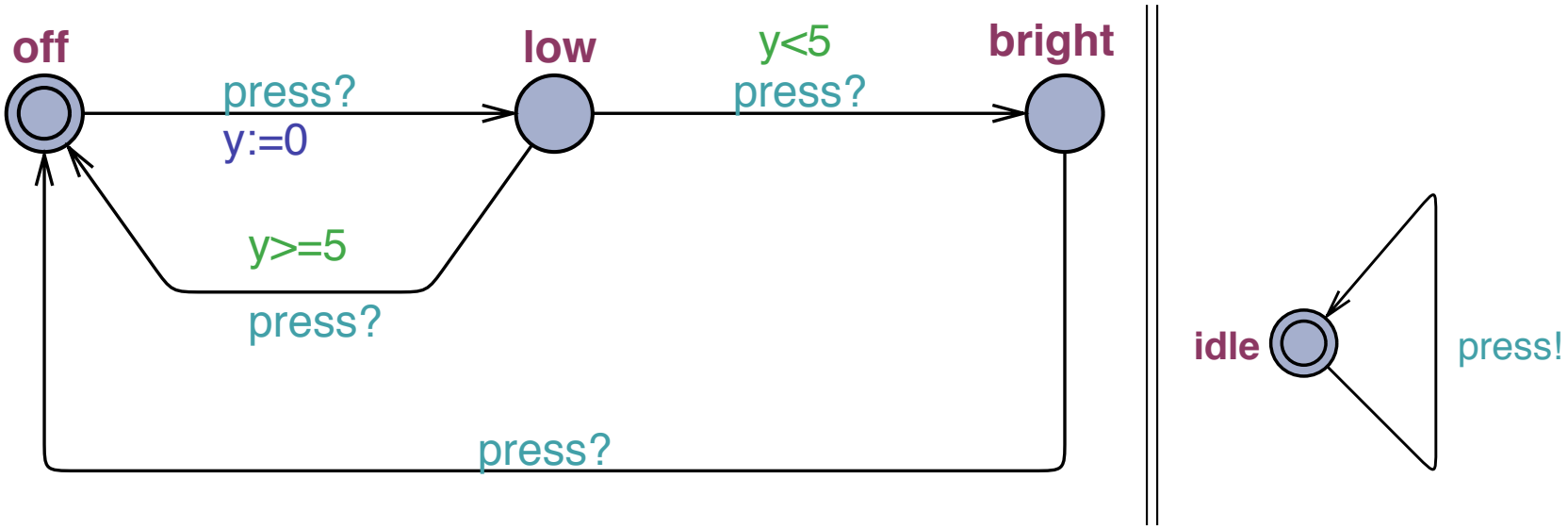
E includes edges defined as follows.

$$\frac{(l_1, \alpha, cc_1, reset_1, l_1') \in E_1 \quad (l_2, \alpha, cc_2, reset_2, l_2') \in E_2}{((l_1, l_2), \alpha, cc_1 \wedge cc_2, reset_1 \cup reset_2, (l_1', l_2')) \in E} \; Sync$$

$$\frac{(l_1, \alpha, cc_1, reset_1, l_1') \in E_1 \quad \alpha \notin A_2}{((l_1, l_2), \alpha, cc_1, reset_1, (l_1', l_2)) \in E} \; Async$$

$$\frac{(l_2, \alpha, cc_2, reset_2, l_2') \in E_2 \quad \alpha \notin A_1}{((l_1, l_2), \alpha, cc_2, reset_2, (l_1, l_2')) \in E} \; Async$$

A Lamp



A Classic Example

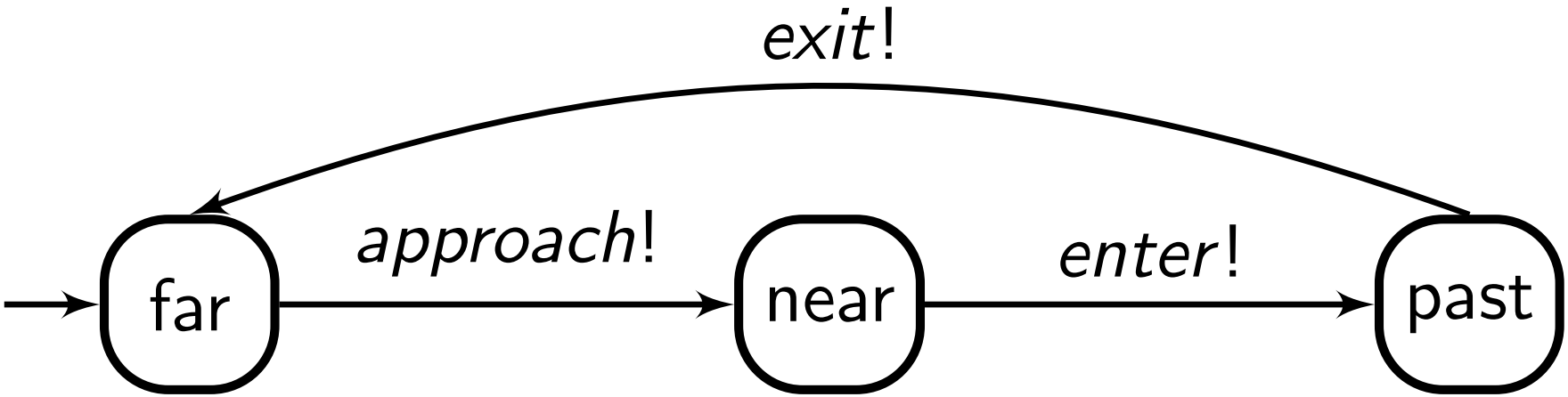
A Train-Gate-Control Example

A road crosses a railway. In the cross, gates are controlled to block traffic on the road for safety.

- Trains communicates with the controller about its position relative to the cross. Trains signal the controller with *approach* and *exit*.
- The controller reacts to *approach* by signaling the gate with *lower*, and reacts to *exit* by signaling the gate with *raise*.
- The gate reacts to *lower* by closing the gate, and reacts to *raise* by opening the gate.

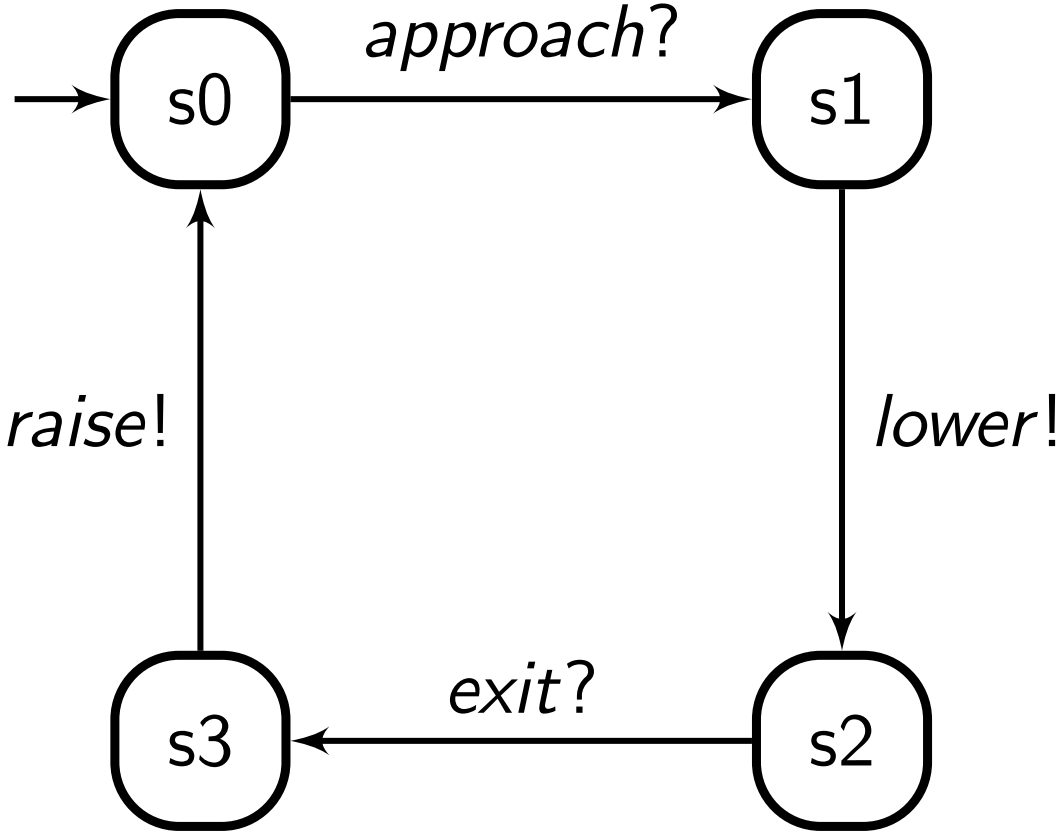
A Train-Gate-Control Example

Trains communicates with the controller about its position relative to the cross. Trains signal the controller with *approach* and *exit*.



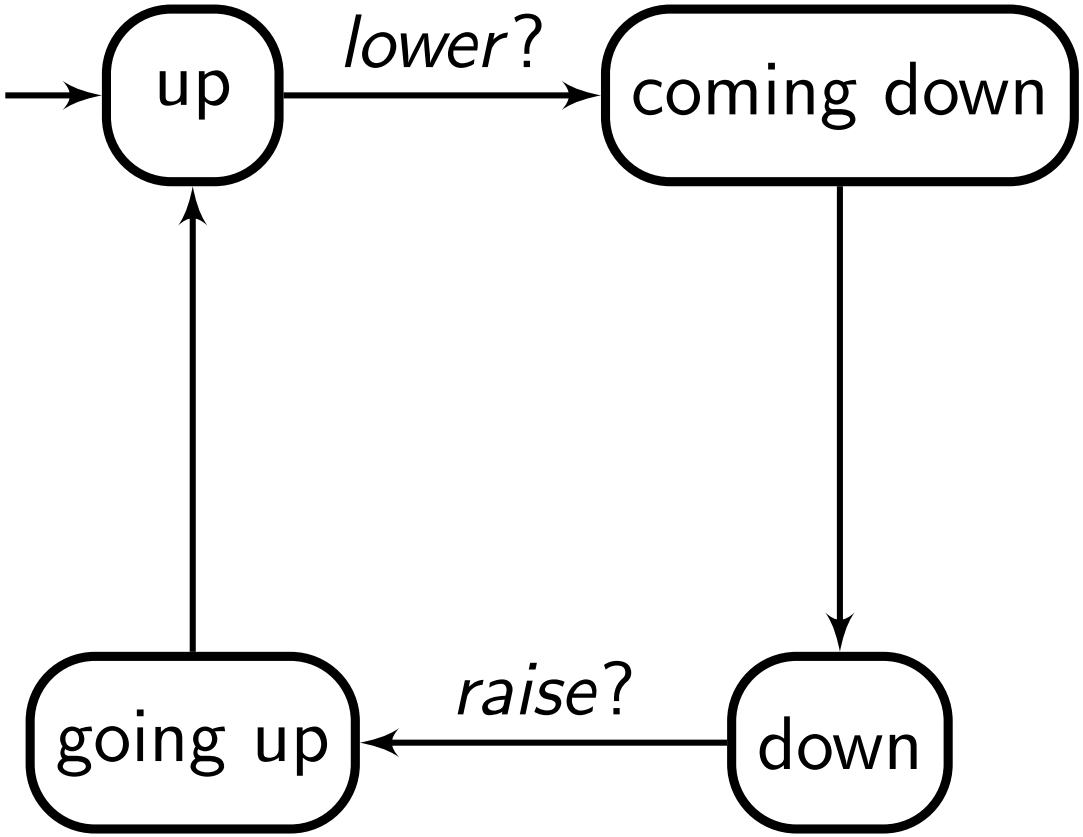
A Train-Gate-Control Example

The controller reacts to *approach* by signaling the gate with *lower*, and reacts to *exit* by signaling the gate with *raise*.

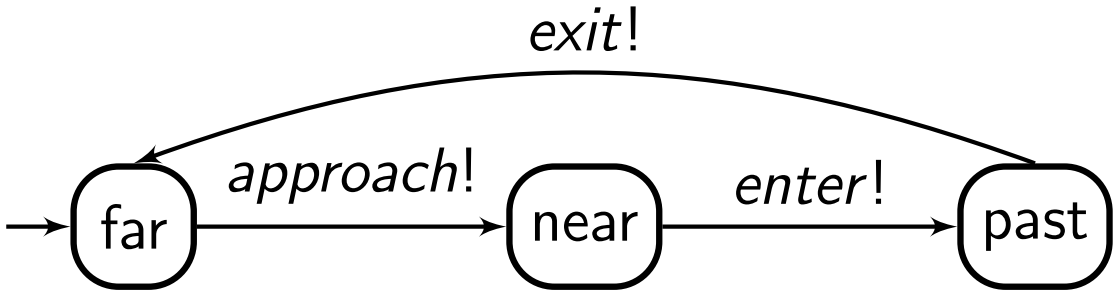


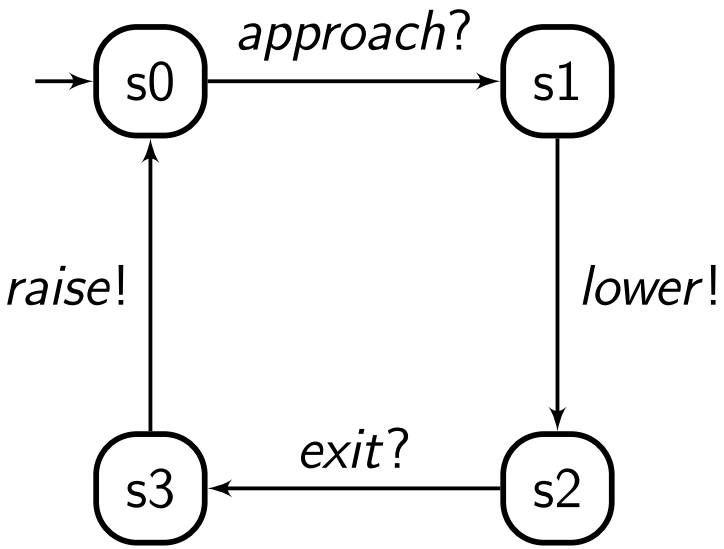
A Train-Gate-Control Example

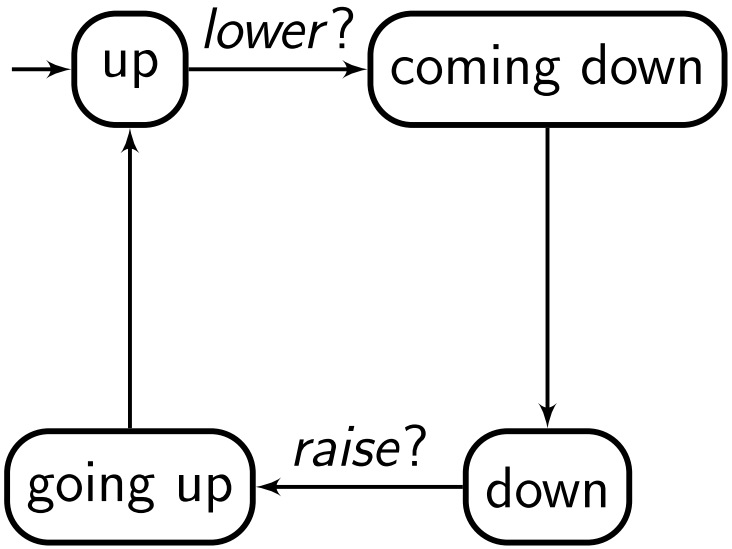
The gate reacts to *lower* by closing the gate, and reacts to *raise* by opening the gate.



Train-Gate-Control: the Whole Picture

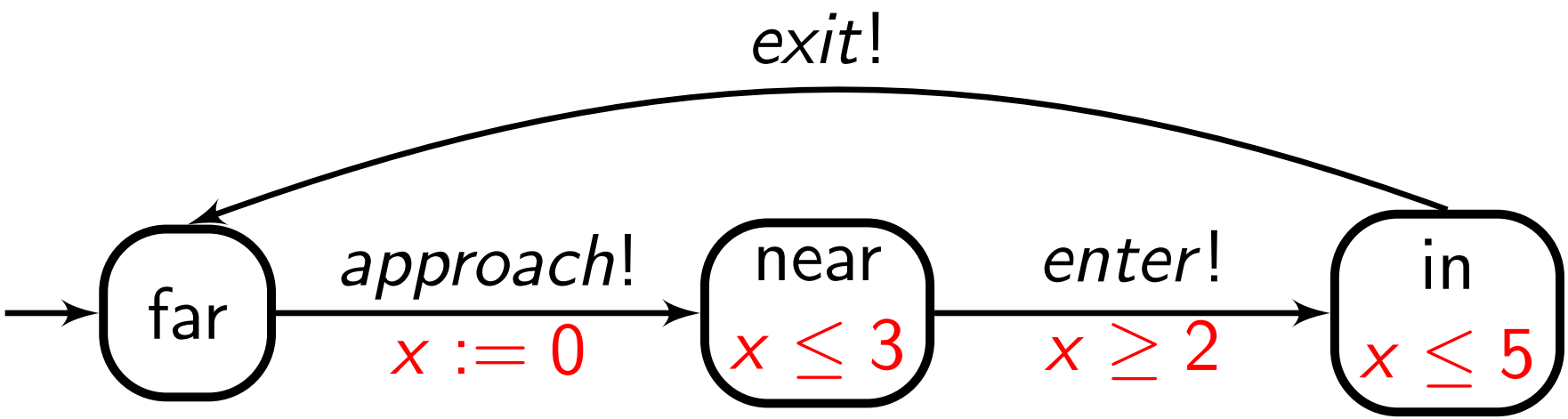






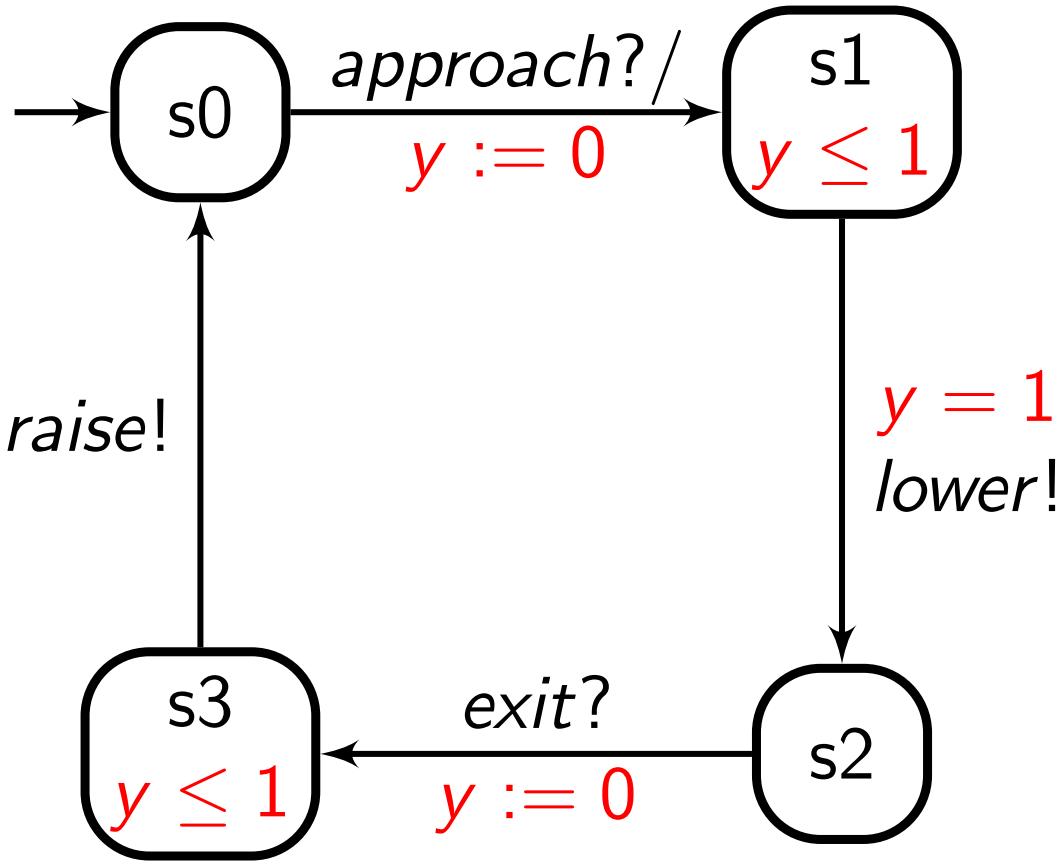
A Train-Gate-Control: Timing

2 or 3 minutes after the train signals the controller *approach*, it reaches the gate. At most 5 minutes after the train signals *approach*, it leaves the gate.



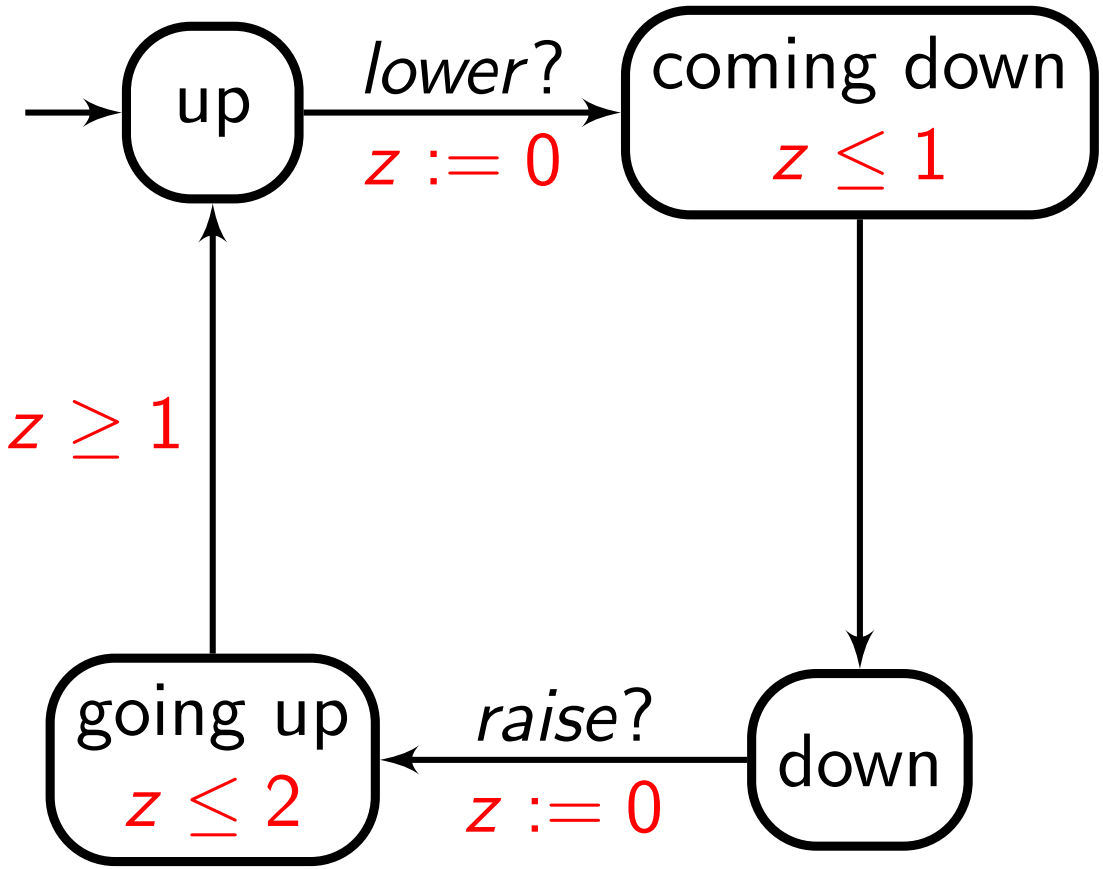
A Train-Gate-Control: Timing

After receiving *approach*, it takes the controller 1 minute to produce signal *lower* to the gate. After receiving *exit*, it takes no more than 1 minute for the controller to produce *raise*.

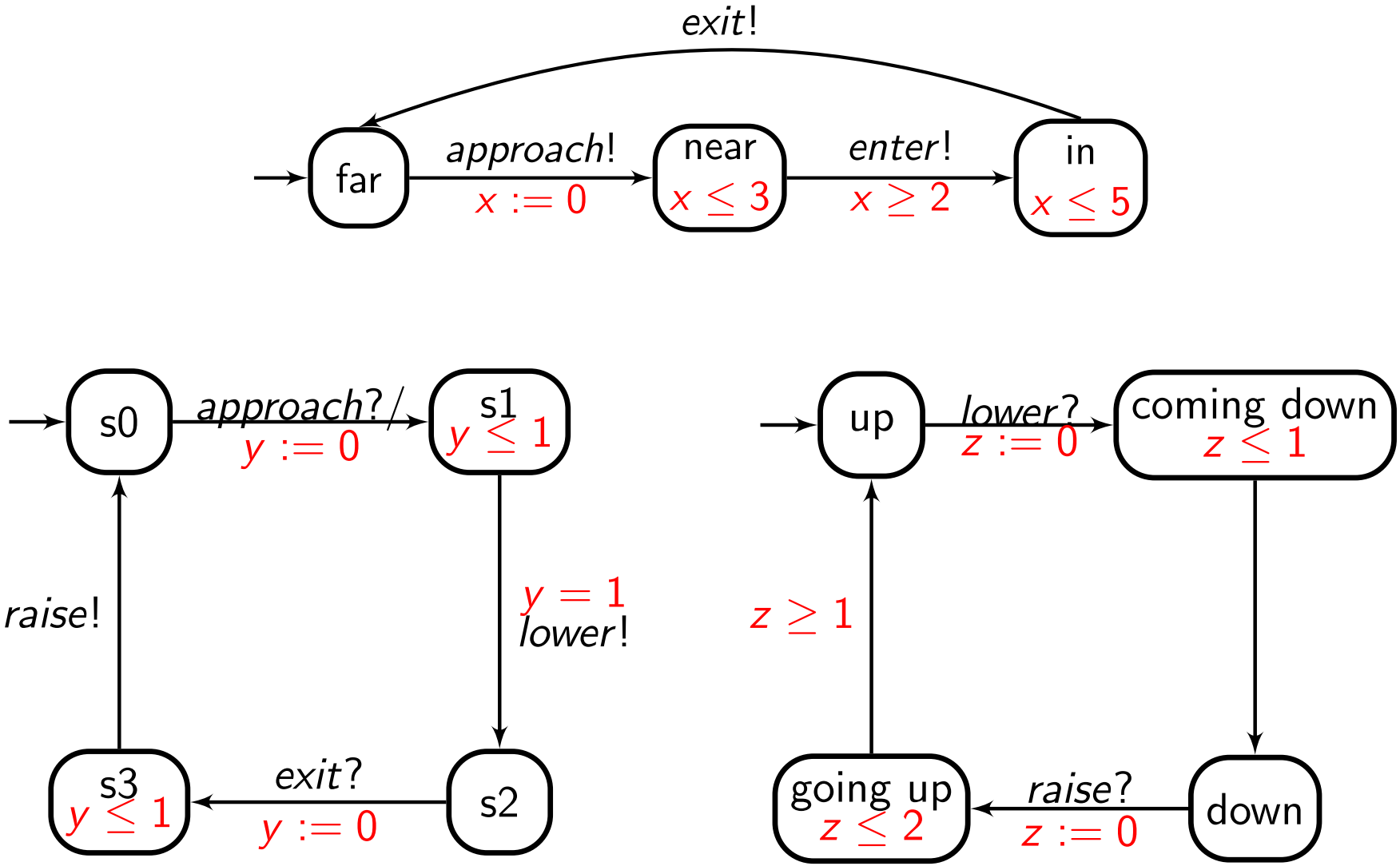


A Train-Gate-Control: Timing

The gate needs at most 1 minute to be closed, and between 1 and 2 minutes to be open.

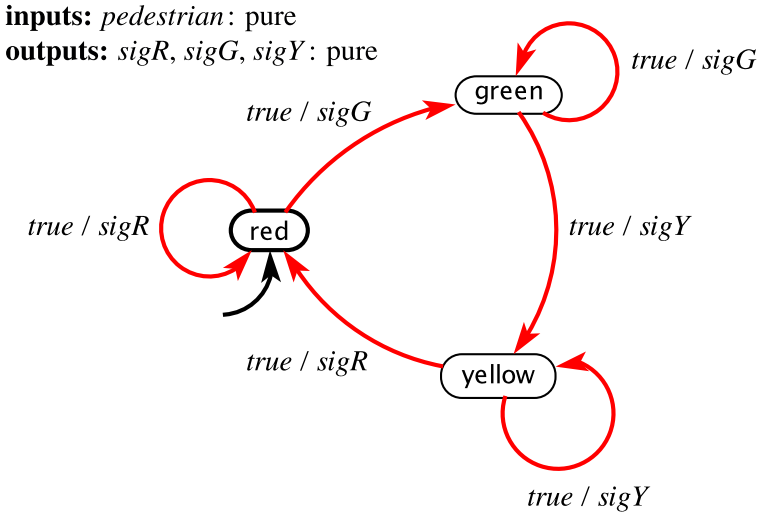


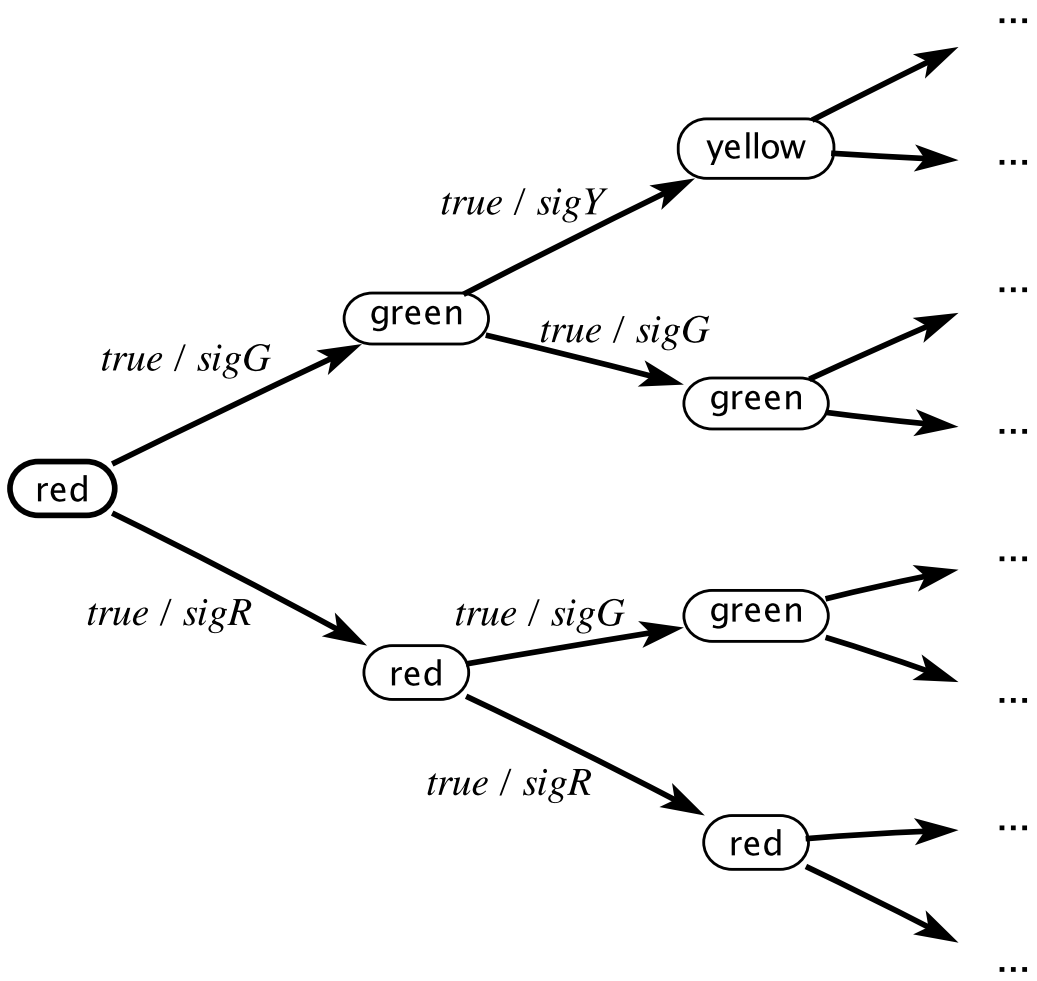
Timed Train-Gate-Control: the Whole Picture



Specification in UPPAAL

Computation Tree





A state of TA = current locations + values of discrete and clock variables.

Timed Computation Tree Logic

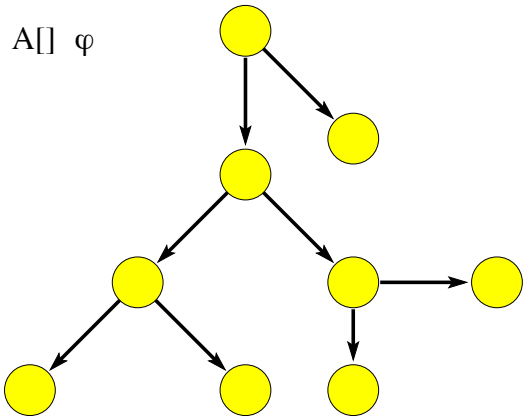
- State formulas φ: expressions whose truth can be decided on individual states.

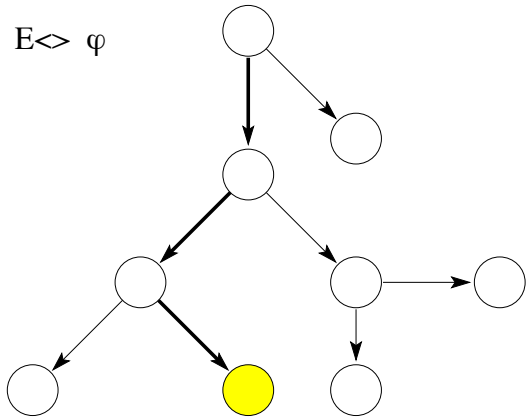
$$i = 7, \quad x \leq 7 \wedge y > 9$$

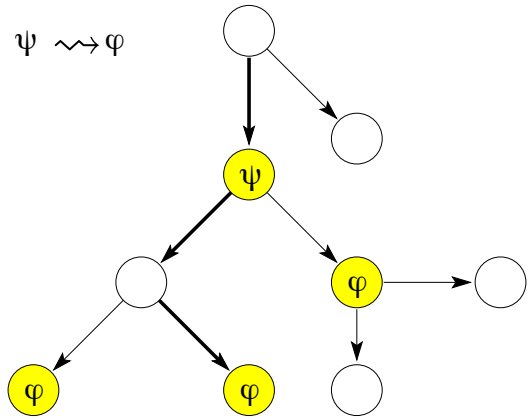
  UPPAAL has a keyword `deadlock`, which, when true, indicates a deadlock.
- Path formulas ϕ are
  - $\Box\varphi$: φ holds in every state of a trace,
  - $\Diamond\varphi$: φ holds in some state of a trace.
- Path quantifiers:
  - **A**ϕ: ϕ holds on *every* path from the initial state.
  - **E**ϕ: ϕ holds on *some* path from the initial state.

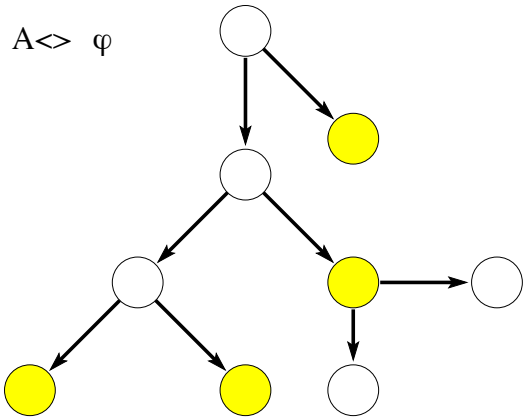
Timed Computation Tree Logic

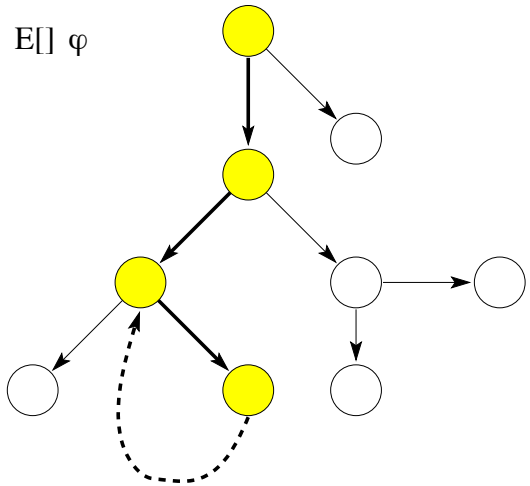
- Reachability Properties: decide if certain states are reachable.



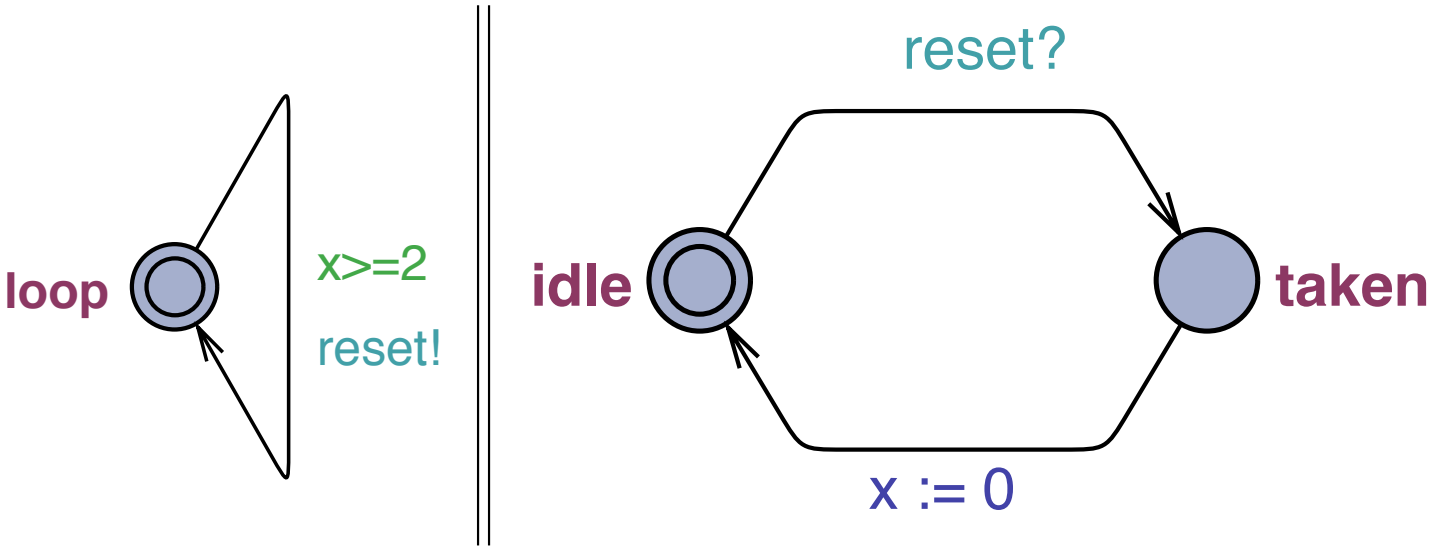








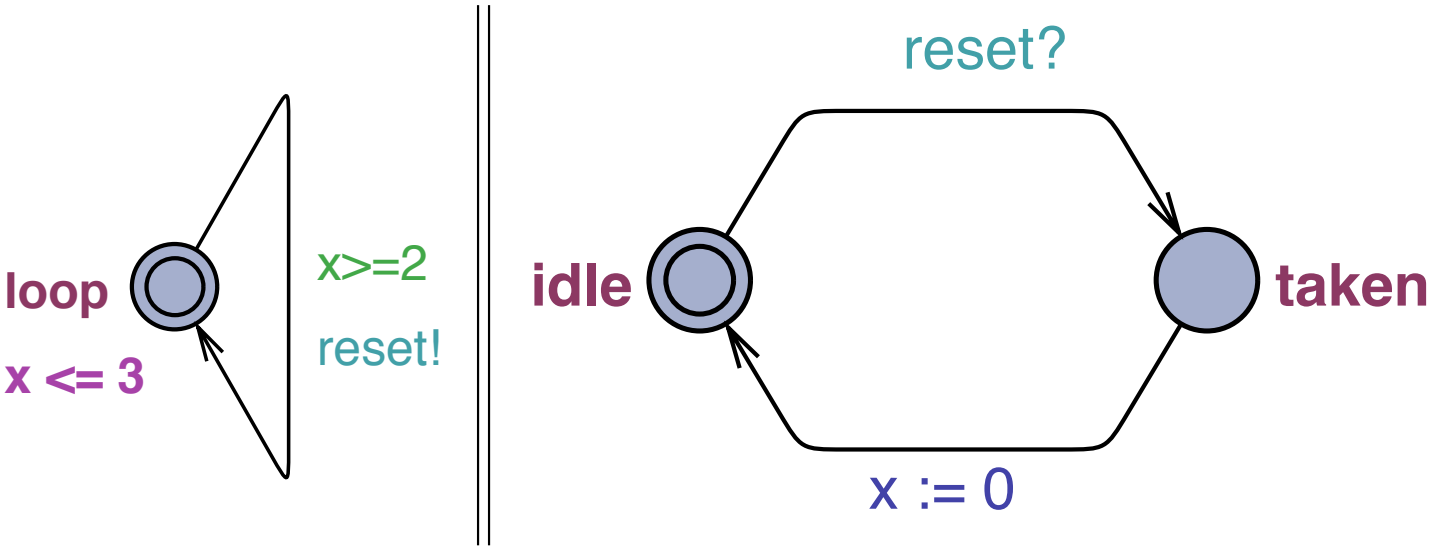
Computation Tree Logic: Example



TCTL Properties:

- `A[] Obs.taken imply x>=2`
- `E[] Obs.idle and x>3`

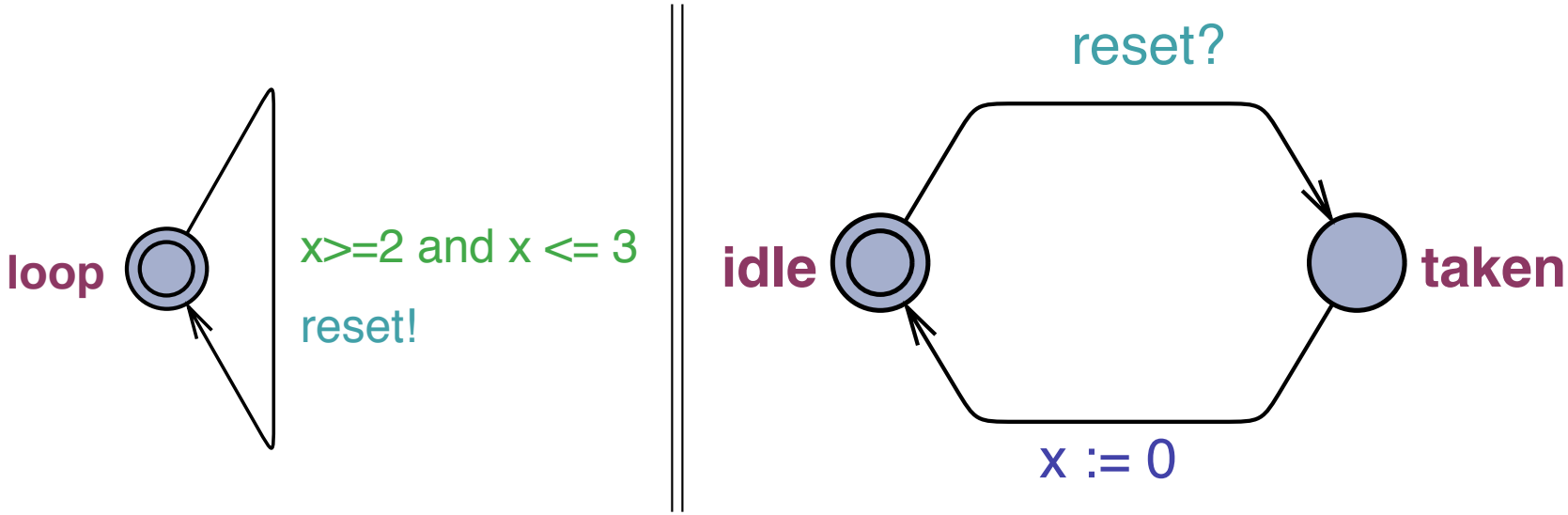
Computation Tree Logic: Example



TCTL Properties:

- `A[] Obs.taken imply x>=2`
- `E[] Obs.idle and x>3`
- `A[] Obs.taken imply (x >=2 and x <=3)`

Computation Tree Logic: Example



TCTL Properties:

- `A[] Obs.taken imply (x >=2 and x <=3)`
- `E<> deadlock`