Discrete Dynamics and State Machine

01266212
CYBER PHYSICAL SYSTEM DESIGN

SEMESTER 1-2021

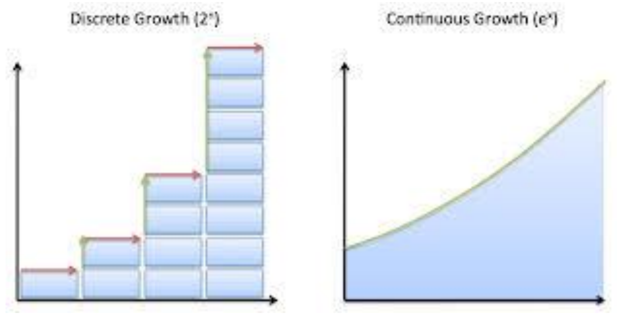
Original contents from
Edward A. Lee and Prabal Dutta, UC Berkeley, EECS 149/249A

Topics

- Discrete Dynamics
- Actor Models of Discrete Systems: Types and Interfaces
- States, Transitions, Guards
- Determinism and Receptiveness

Continuous vs Discrete

continuous dynamic behavior – a system that can both *flow* (described by a differential equation) and discrete dynamic behavior – a system that can *jump* (described by a state machine or automaton).



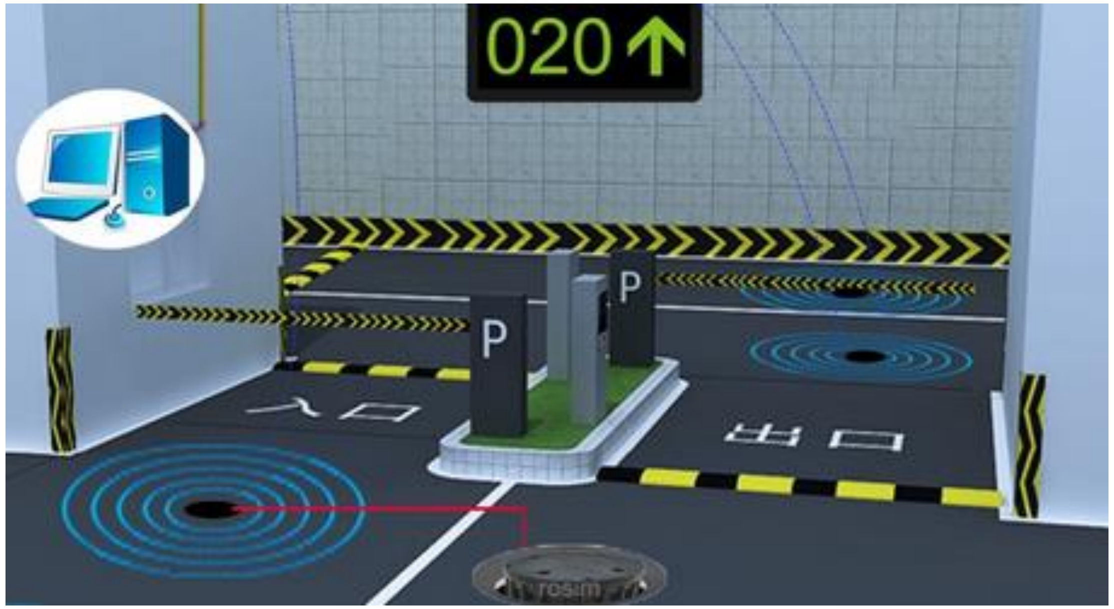
Loosely, continuous components evolve smoothly, while discrete components evolve abruptly.

Discrete Systems

- **Discrete** = “individually separate / distinct”
- A **discrete system that** have discrete dynamics is one that operates in a sequence of discrete *steps* or has signals taking discrete *values*.
- A discrete event occurs at an instant of time rather than over time.
- Example: Arrival of email, Arrival of customer at Shop.

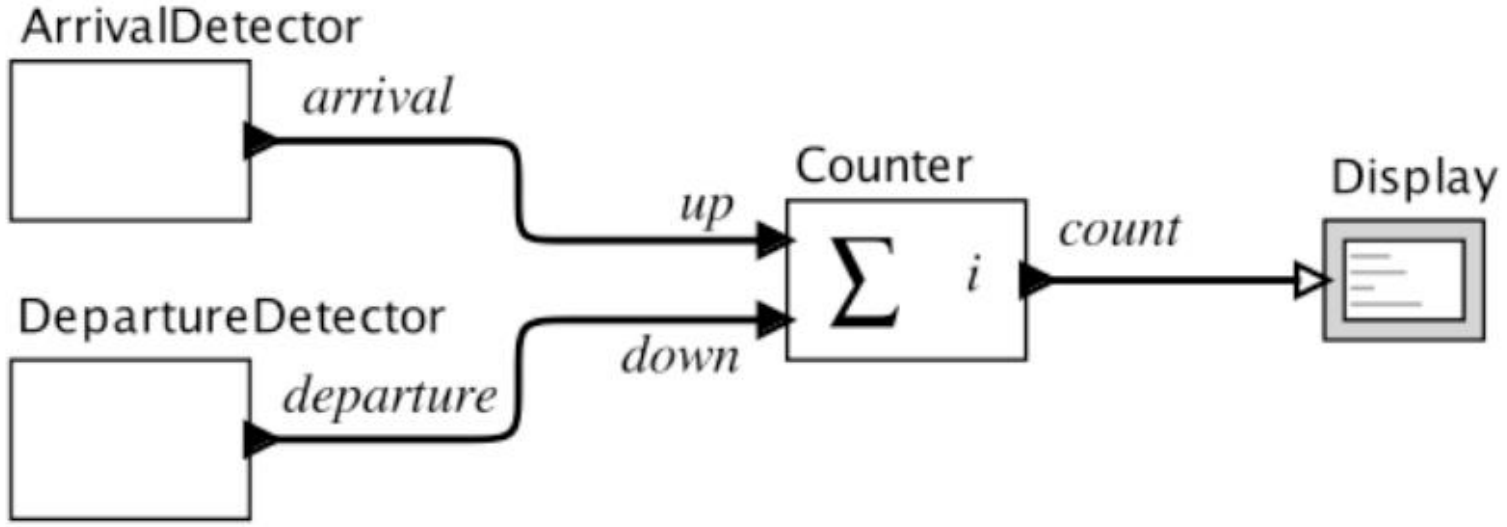
Discrete Systems: Example Design Problem

Count the number of cars that are present in a parking garage by sensing cars enter and leave the garage. Show this count on a display.



Discrete Systems: Example Design Problem

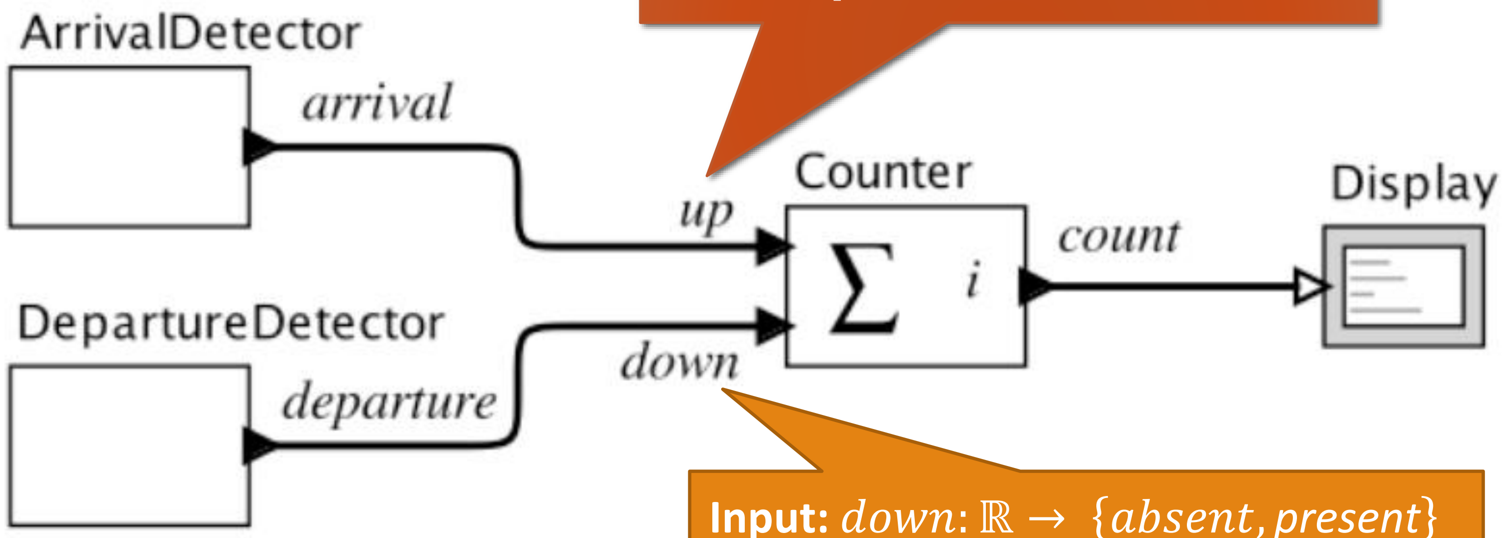
Example: Count the number of cars in a parking garage by sensing those that enter and leave:



Discrete Systems: Example Design Problem

Example: Count the number of cars in a parking garage by sensing those that enter and leave:

Input: $up\colon \mathbb{R} \to \{absent, present\}$



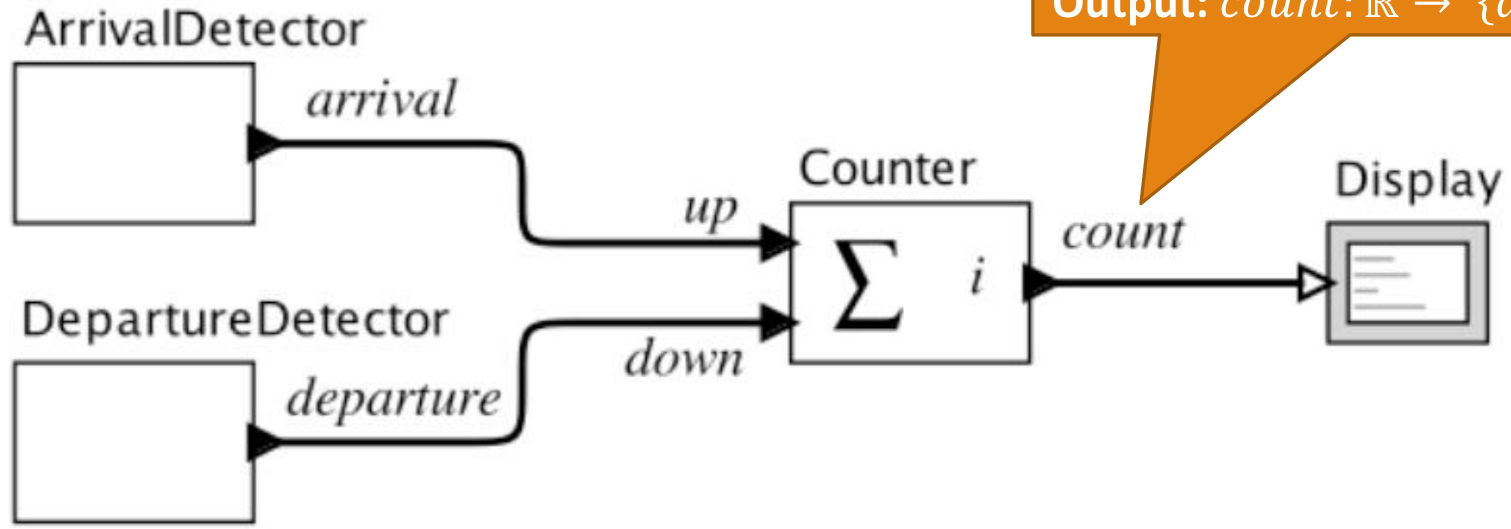
Input: $down\colon \mathbb{R} \to \{absent, present\}$

Pure signal: that at any time $t \in \mathbb{R}$, the input $u(t)$ is either *absent*, meaning that there is no event at that time, or *present*, meaning that there is.

Discrete Systems: Example

Example: count the number of cars in a parking garage by sensing those that enter and leave:

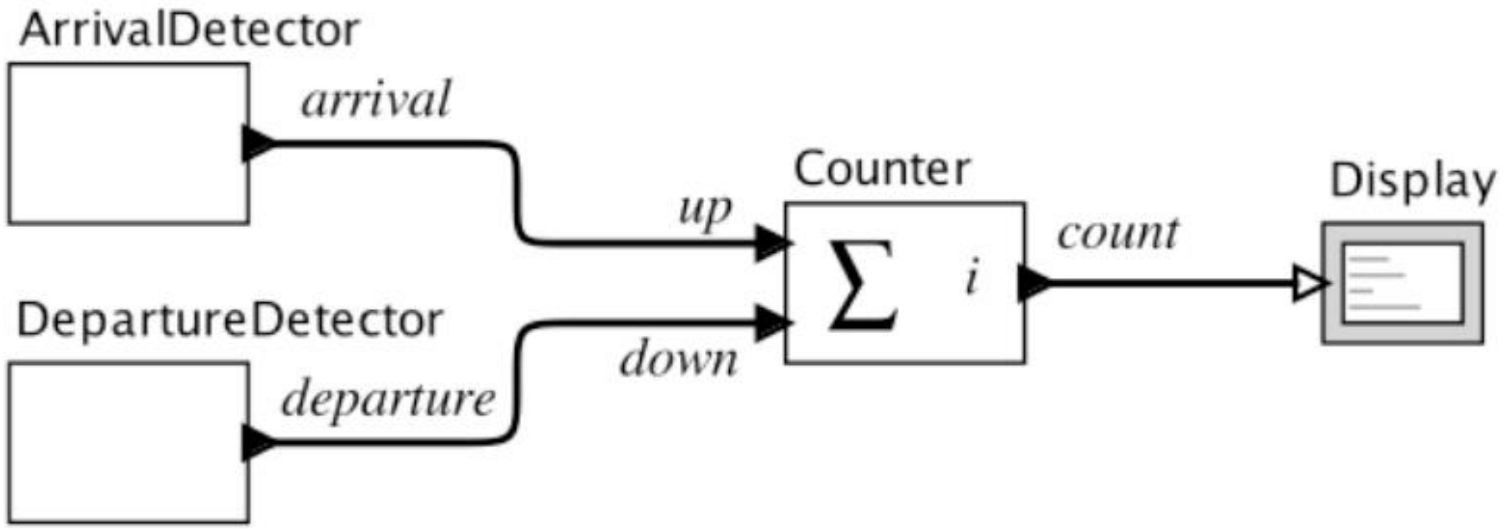
Output: $count: \mathbb{R} \rightarrow \{absent\} \cup \mathbb{N}$



This signal output *count* is not pure, but like *up* and *down*, it is either absent or present. However, it has a value (an integer) when it is present.

Discrete Systems: Example

Example: count the number of cars in a parking garage by sensing those that enter and leave:



Discrete actor: $Counter\colon (\mathbb{R} \to \{absent, present\})^P \to (\mathbb{R} \to \{absent\} \cup \mathbb{N})$

$P = \{up, down\}$ is the number of input ports.

Discrete Signals

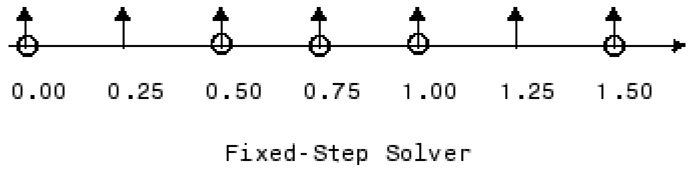
- Discrete signals consist of a sequence of instantaneous events in time.

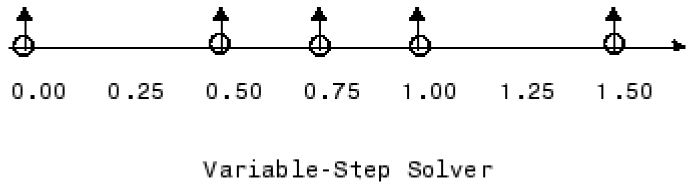
$$e(t)\colon \mathbb{R} \to \{absent\} \cup X \text{ where } X \text{ is any set of values}$$

- It is absent most of time, but we can count the times at which it is present (a discrete event)
- Let $T \subseteq \mathbb{R}$ be the set of times where e is present.

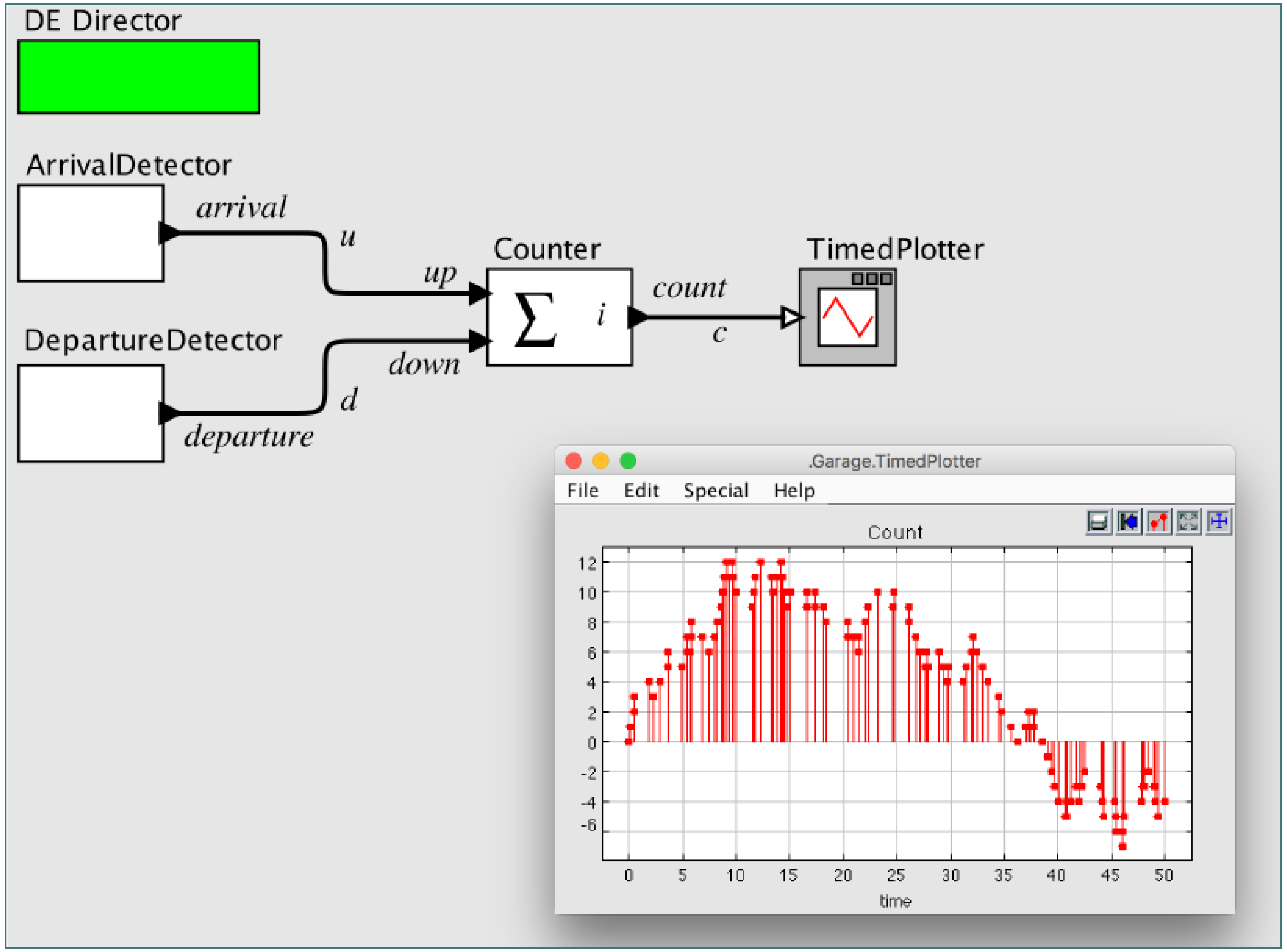
$$T = \{t \in \mathbb{R}: e(t) \neq absent\}$$

- a one-to-one function $f: T \to \mathbb{N}$ that is **order preserving**.



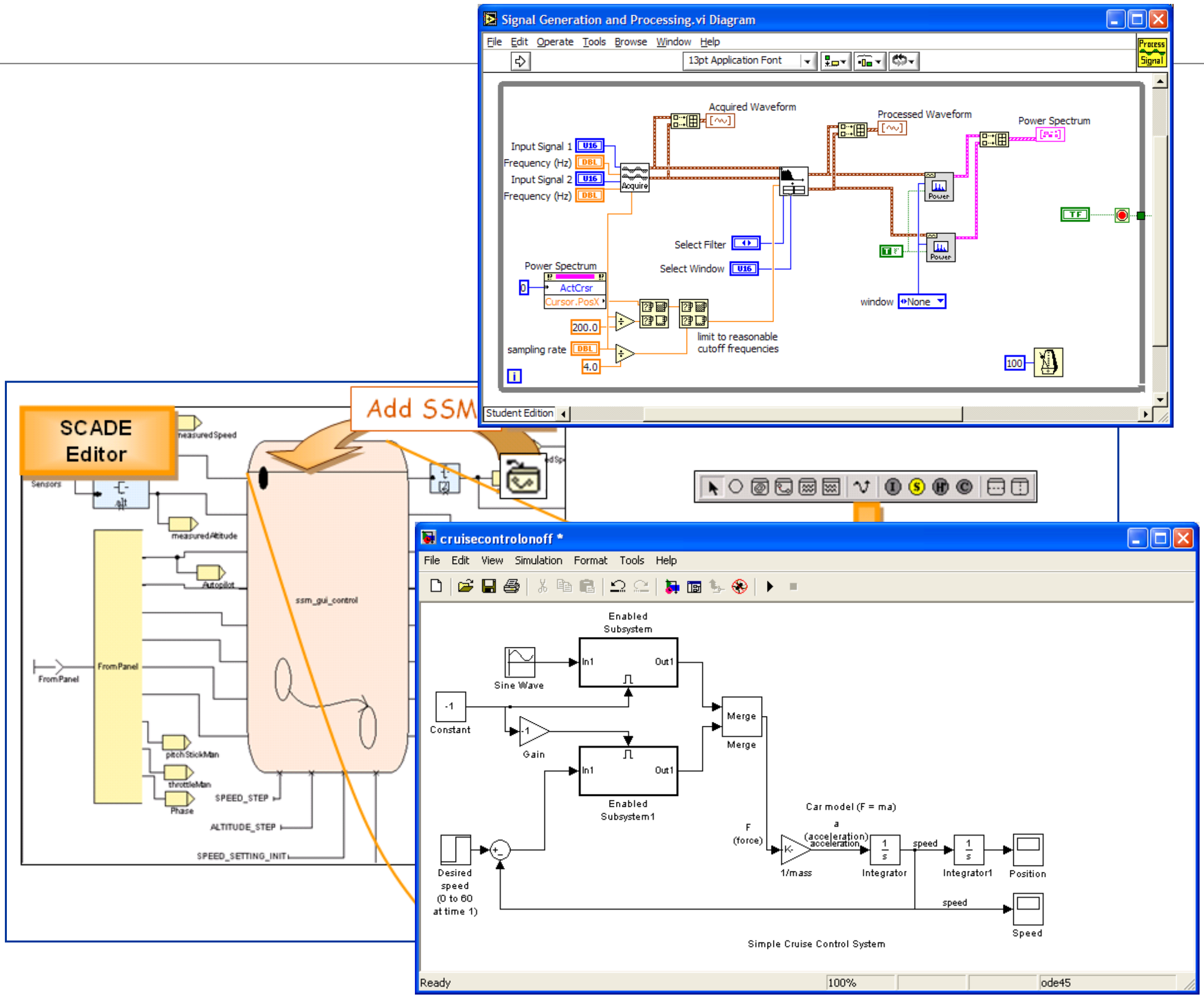


Demonstration of Ptolemy II Model ("Program")

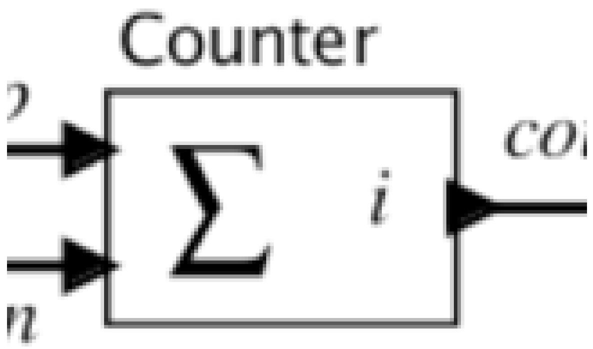


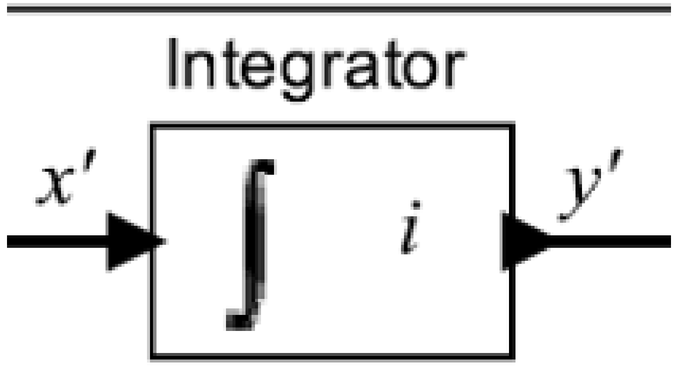
Actor Modeling Languages / Frameworks

- LabVIEW
- Simulink
- Scade
- …
- Reactors
- StreamIT
- …



Counter in Discrete ⇔ Integrator in Continuous





Actor Model for Integrator/ Counter

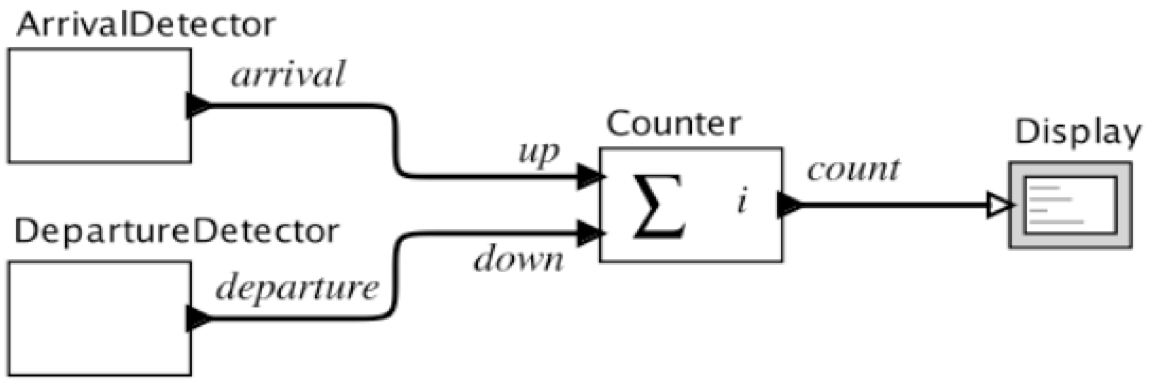
- Both Integrator actor and Counter actor have state.
- The state of Integrator actor model at a time t is the value of the integral of the input signal up to time *t*.
- The state space of the Integrator is $States \in \mathbb{R}$.
- The state of Counter actor is an accumulation of past input values, but it operates discretely.
- The state at time *t* is an integer. $States \subset \mathbb{Z}$

Reaction

For any $t \in \mathbb{R}$ where $up(t) \neq absent$ or $down(t) \neq absent$ the Counter **reacts**. It produces an output value in $\mathbb{N}$ and changes its internal **state**.

State: condition of the system at a particular point in time

- Encodes everything about the past that influences the system's reaction to current input



Event triggered: Reactions of the Counter actor are triggered when one or more input events are present

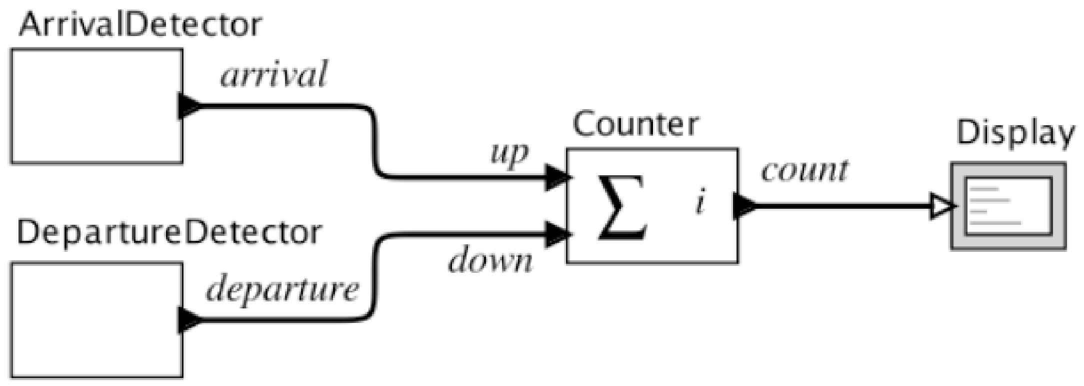
Inputs and Outputs at a Reaction

For $t \in \mathbb{R}$ the inputs are in a set

$$Inputs = (\{\mathsf{up}, \mathsf{down}\} \to \{\, absent, present\})$$

and the outputs are in a set

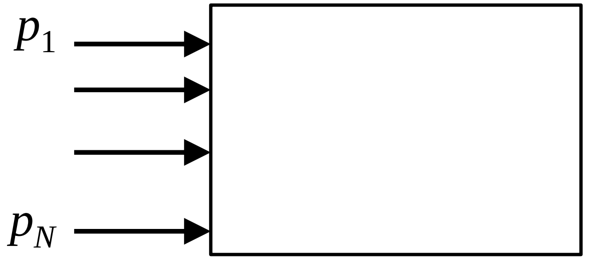
$$Outputs = (\{count\} \to \{\, absent\} \cup \mathbb{N})\ ,$$



Suppose an actor has input ports $P = \{p_1, \ldots, p_\mathrm{N}\}$, where p_i is the name of the i-th input port.

For each input port $p \in P$, *a set* V_p denotes the values that may be received on port p when the input is present. Vp is called the **type** of port p.

At a reaction we treat each $p \in P$ as a variable that takes on a value $p \in V_p \cup \{absent\}$. A **valuation** of the inputs P is an assignment of a value in V_p to each $p \in P$ or an assertion that p is absent.



Question

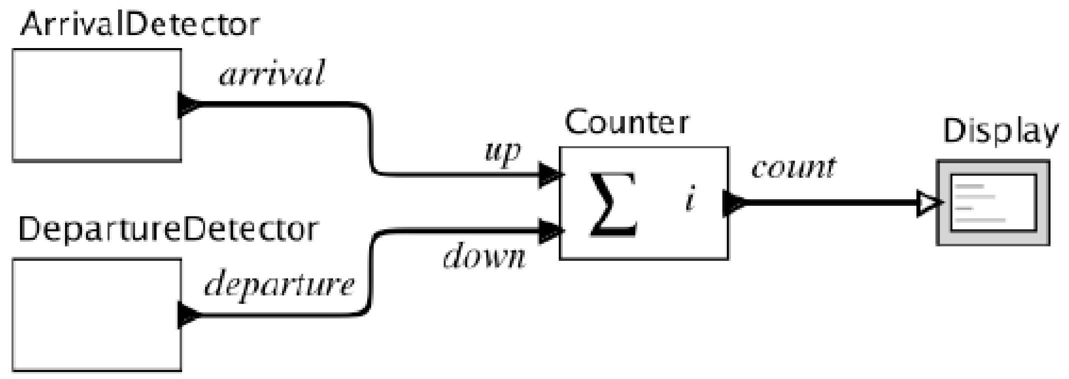
What are some scenarios that the given parking garage (interface) design does not handle well?

For $t \in \mathbb{R}$ the inputs are in a set

$$Inputs = (\{\mathsf{up}, \mathsf{down}\} \to \{absent, present\})$$

and the outputs are in a set

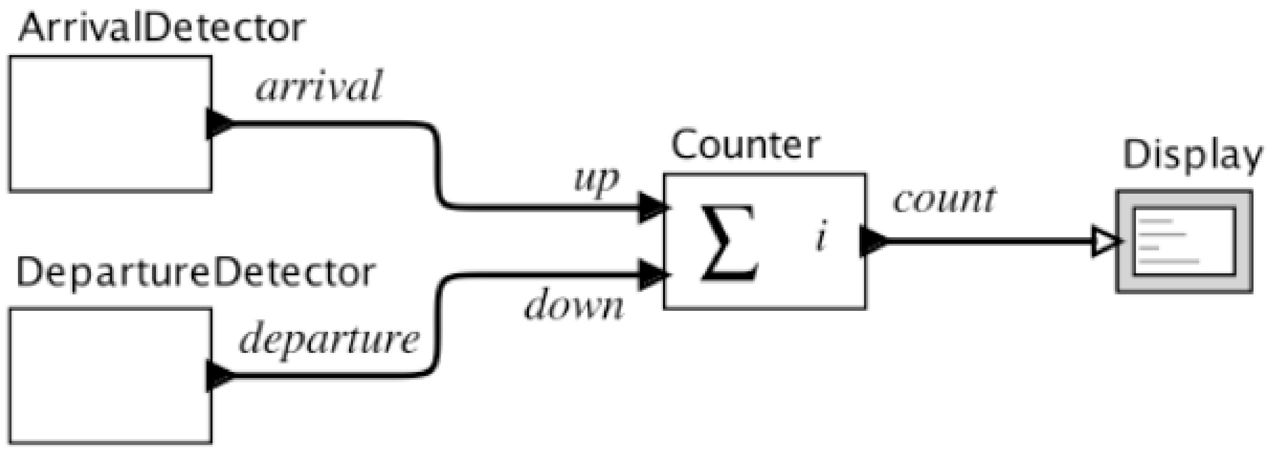
$$Outputs = (\{count\} \to \{absent\} \cup \mathbb{N}) ,$$



State Space

A practical parking garage has a finite number *M* of spaces, so the state space for the counter is

$$States = \{0, 1, 2, \cdots, M\} .$$



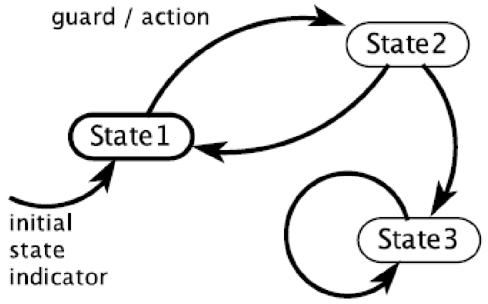
Finite-State Machines

- **Discrete models** with finite state spaces are called **finite-state machines (FSMs).**
- A state machine is a model of a system with *discrete dynamics* that at each reaction maps *valuations* of the inputs to *valuations* of the outputs, where the map may depend on its current state.

$$s(Current\ State, Input) = Output$$

- For a FSM system with a small number of states , each state is represented by a bubble, so for this diagram, the set of states is given by

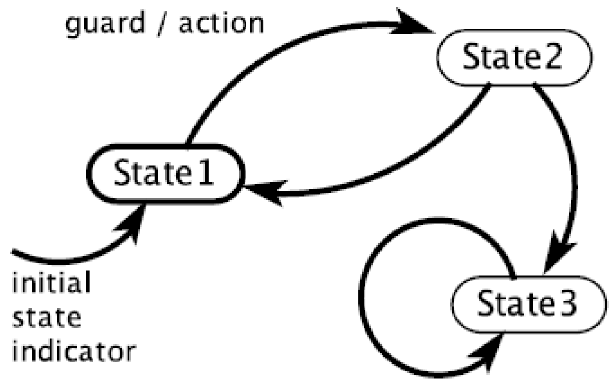
$$States = \{\text{State1}, \text{State2}, \text{State3}\}$$



Transitions

- **Transitions** between states govern the discrete dynamics of the state machine and the mapping of input valuations to output valuations.
- A transition is represented as a curved arrow.
- A transition may also start and end at the same state, is called a **self transition**.
- The transition from State1 to State2 is labeled with "**guard / action**."

The **guard** determines whether the transition may be taken on a reaction.



The **action** specifies what outputs are produced on each reaction.

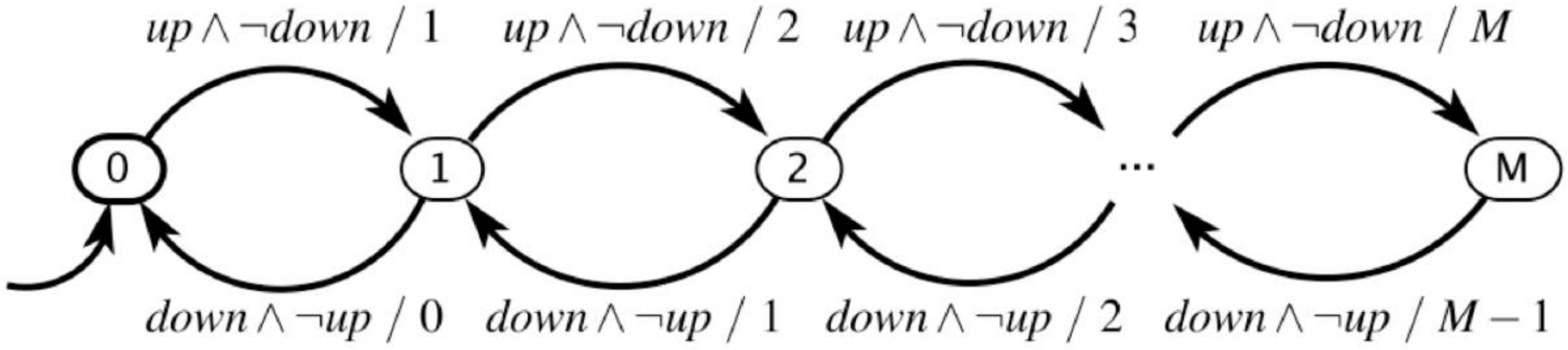
Guard(Input) and Action(Output)

- A guard is a **predicate** (a boolean-valued expression) that evaluates to true when the transition should be taken, changing the state from that at the beginning of the transition to that at the end.
- A predicate is **a function**. It takes some variable(s) as arguments; it returns either *true* or *false* (but not both) for each combination of the argument values.

Example:: $P(x, y) : x + y > 7$

- A When a guard evaluates to *true* we say that the transition is enabled.
- An action is an assignment of values (or absent) to the output ports.
- Any output port not mentioned in a transition that is taken is implicitly *absent*.

Garage Counter Finite State Machine (FSM) in Pictures


$up \wedge \neg down$ / 1
$up \wedge \neg down$ / 2
$up \wedge \neg down$ / 3
$up \wedge \neg down$ / M
0
1
2
...
M
$down \wedge \neg up$ / 0
$down \wedge \neg up$ / 1
$down \wedge \neg up$ / 2
$down \wedge \neg up$ / $M-1$

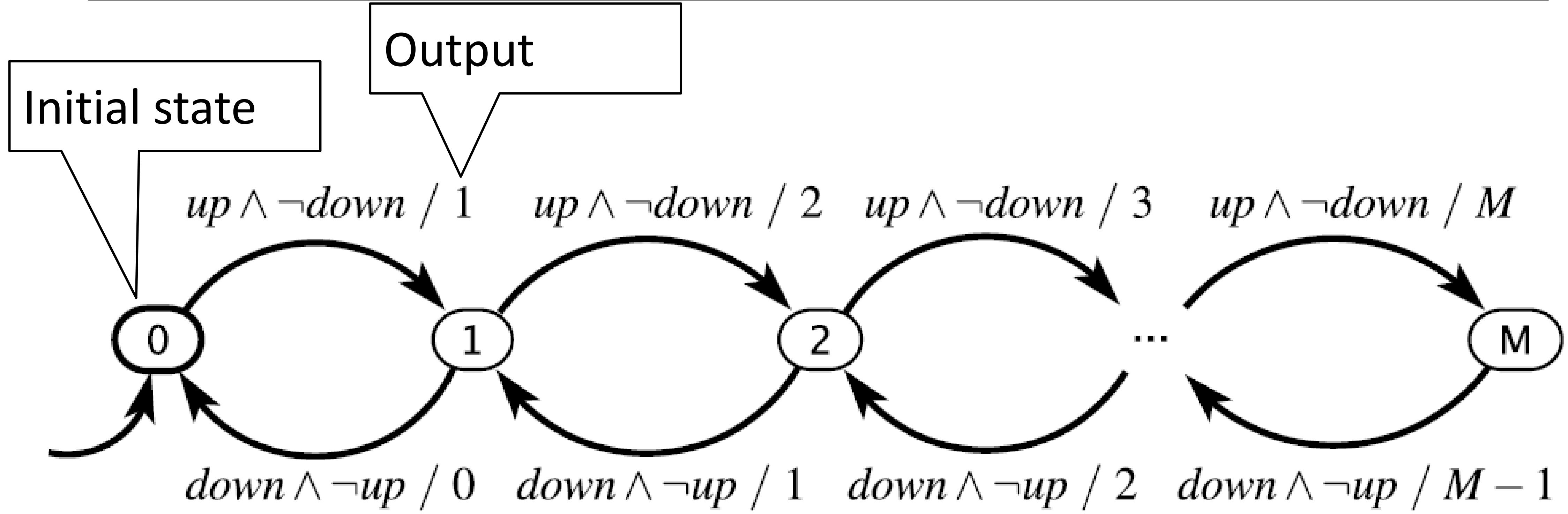
Guard $g \subseteq Inputs$ is specified using the shorthand

$$up \wedge \neg down$$

which means

$$Inputs(up) = present \text{ and } Inputs(down) = absent$$

Garage Counter Finite State Machine (FSM) in Pictures



Examples of Guards for Pure Signals

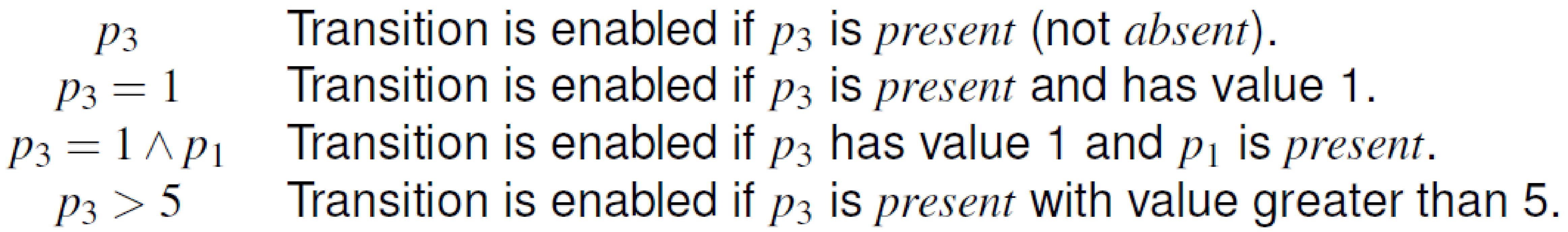
If p_1 and p_2 are pure inputs to a discrete system,

true	Transition is always enabled.
p_1	Transition is enabled if p_1 is $present$.
$\neg p_1$	Transition is enabled if p_1 is $absent$.
$p_1 \wedge p_2$	Transition is enabled if both p_1 and p_2 are $present$.
$p_1 \vee p_2$	Transition is enabled if either p_1 or p_2 is $present$.
$p_1 \wedge \neg p_2$	Transition is enabled if p_1 is $present$ and p_2 is $absent$.

The symbol $\neg$ represents logical negation. The operator $\wedge$ is logical conjunction (logical AND), and $\vee$ is logical disjunction (logical OR).

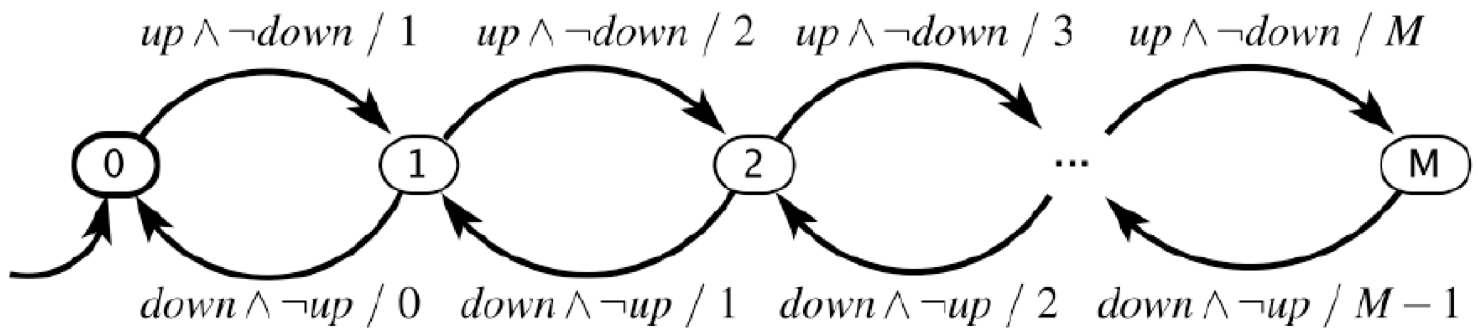
Examples of Guards for Signals with Numerical Values (Not pure signal)

In addition, the discrete system has a third input port p_3 with type $V_{p_3} = \mathbb{N}$



p_3	Transition is enabled if p_3 is *present* (not *absent*).
$p_3 = 1$	Transition is enabled if p_3 is *present* and has value 1.
$p_3 = 1 \wedge p_1$	Transition is enabled if p_3 has value 1 and p_1 is *present*.
$p_3 > 5$	Transition is enabled if p_3 is *present* with value greater than 5.

Garage Counter Mathematical Model


$up \wedge \neg down \,/\, 1$
$up \wedge \neg down \,/\, 2$
$up \wedge \neg down \,/\, 3$
$up \wedge \neg down \,/\, M$
0
1
2
...
M
$down \wedge \neg up \,/\, 0$
$down \wedge \neg up \,/\, 1$
$down \wedge \neg up \,/\, 2$
$down \wedge \neg up \,/\, M-1$

Formally: $(States, Inputs, Outputs, update, initialState)$, where

- $States = \{0, 1, \cdots, M\}$
- $Inputs = (\{\mathrm{up}, \mathrm{down}\} \to \{absent, present\}$
- $Outputs = (\{count\} \to \{absent\} \cup \mathbb{N})$
- $update : States \times Inputs \to States \times Outputs$
- $initialState = 0$

The picture above defines the update function.

$$\begin{tikzcd}
{} \arrow[r] & 0 \arrow[r, bend left, "up \wedge \neg down \,/\, 1"] & 1 \arrow[l, bend left, "down \wedge \neg up \,/\, 0"] \arrow[r, bend left, "up \wedge \neg down \,/\, 2"] & 2 \arrow[l, bend left, "down \wedge \neg up \,/\, 1"] \arrow[r, bend left, "up \wedge \neg down \,/\, 3"] & \cdots \arrow[l, bend left, "down \wedge \neg up \,/\, 2"] \arrow[r, bend left, "up \wedge \neg down \,/\, M"] & M \arrow[l, bend left, "down \wedge \neg up \,/\, M-1"]
\end{tikzcd}$$

$$\begin{aligned}
\mathit{States} &= \{0, 1, \cdots, M\} \\
\mathit{Inputs} &= (\{\mathit{up}, \mathit{down}\} \to \{\mathit{present}, \mathit{absent}\}) \\
\mathit{Outputs} &= (\{\mathit{count}\} \to \{0, 1, \cdots, M, \mathit{absent}\}) \\
\mathit{initialState} &= 0
\end{aligned}$$

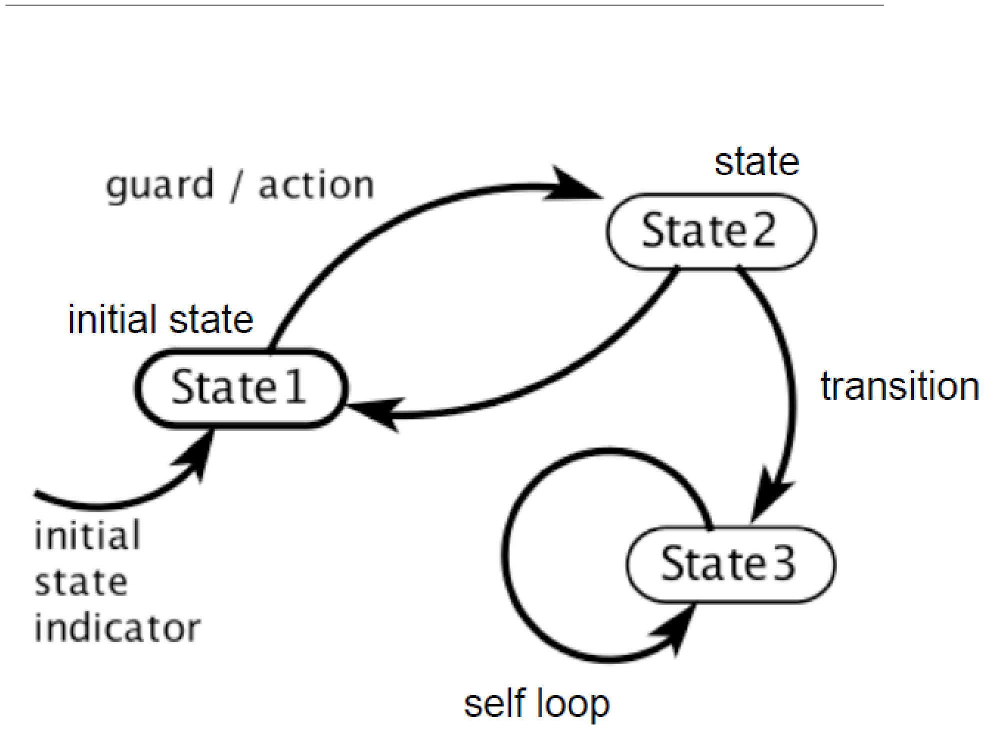
The update function is given by

$$\mathit{update}(s, i) = \begin{cases} (s+1, s+1) & \text{if } s < M \\ & \wedge\, i(\mathit{up}) = \mathit{present} \\ & \wedge\, i(\mathit{down}) = \mathit{absent} \\ (s-1, s-1) & \text{if } s > 0 \\ & \wedge\, i(\mathit{up}) = \mathit{absent} \\ & \wedge\, i(\mathit{down}) = \mathit{present} \\ (s, \mathit{absent}) & \text{otherwise} \end{cases}$$

for all $s \in \mathit{States}$ and $i \in \mathit{Inputs}$. Note that an output valuation $o \in$ Outputs is a function of the form $o : \{\mathit{count}\} \to \{0, 1, \ldots, M, \mathit{absent}\}$

FSM Summary

- ❑ **Transitions** between states govern the <u>discrete dynamics</u> of the state machine and the mapping of input valuations to output valuations.
- ❑ A transition may also start and end at the same state is called **a self transition.**
- ❑ The **guard** determines whether the transition may be taken on a reaction.
- ❑ The **action** specifies what <u>outputs</u> are produced on each reaction.



Mathematical Notation FSM

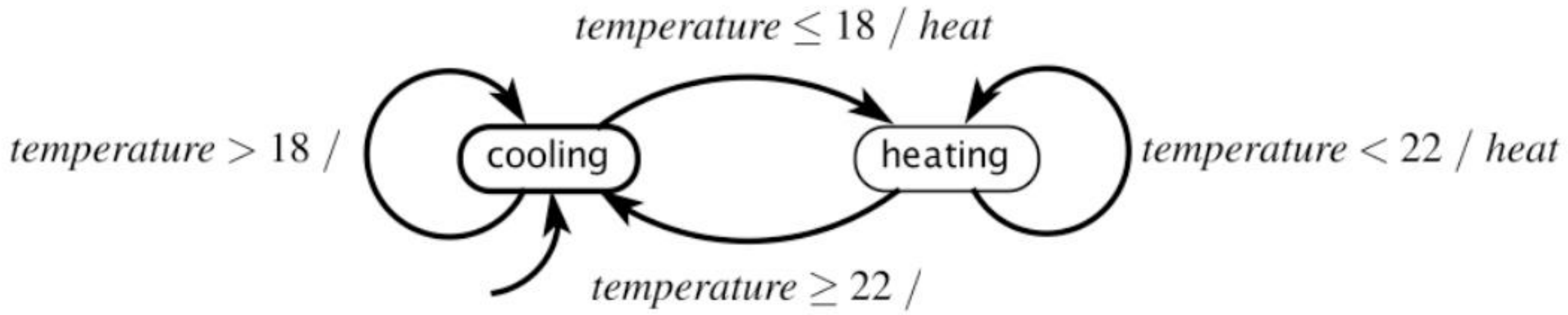
A finite-state machine is a five-tuple (States; Inputs; Outputs; update; initialState)

- *States* is a finite set of states;
- *Inputs* is a set of input valuations;
- *Outputs* is a set of output valuations;
- $update : States \times Inputs \to States \times Outputs$ is an **update function**, mapping a state and an input valuation to a *next* state and an output valuation;
- *initialState* is the initial state.

The FSM reacts in a sequence of reactions. At each reaction, the FSM has a current state, and the reaction may transition to a next state, which will be the current state of the next reaction.

Example of Modal Model: Thermostat

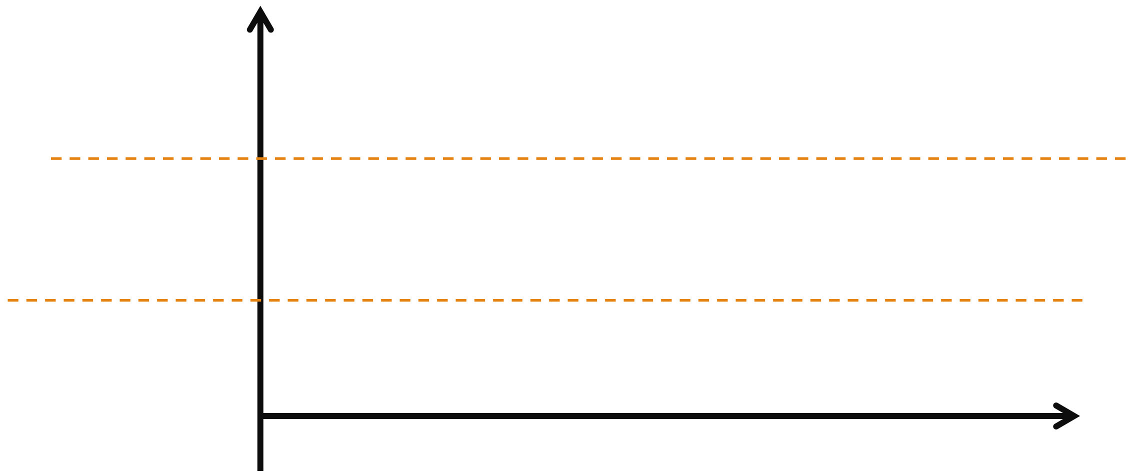
A major use of energy worldwide is in heating, ventilation, and air conditioning (HVAC) systems. Accurate models of temperature dynamics and temperature control systems can significantly improve energy conservation.
Such modeling begins with a modest thermostat, which regulates temperature to maintain a setpoint, or target temperature.

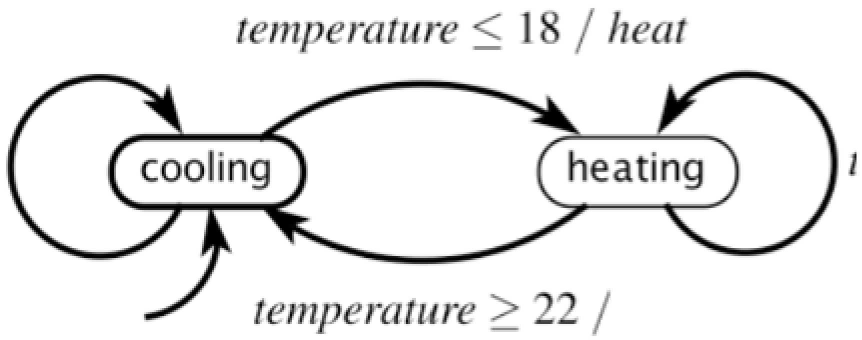


Hysteresis

The phenomenon in which the value of a physical property lags behind changes in the effect causing it.

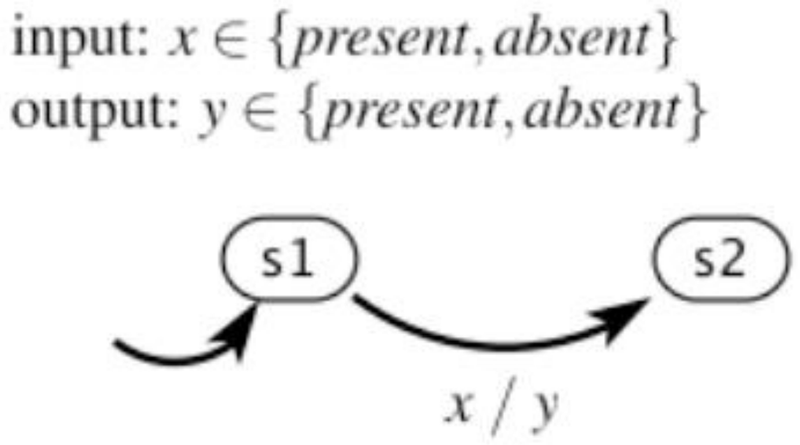
In control systems, hysteresis can be used to prevent that the output/action changes rapidly depending on the threshold level.





When does a reaction occur?

Event-triggered model: Suppose all inputs are discrete and a reaction occurs when any input is present. Then the above transition will be taken whenever the current state is s1 and x is present.

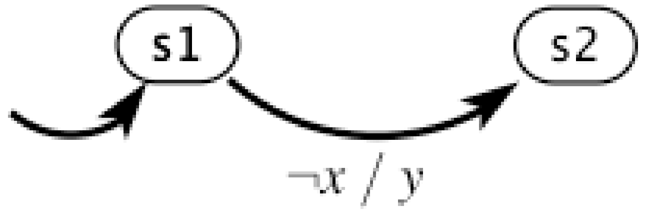

input: $x \in \{present, absent\}$
output: $y \in \{present, absent\}$
s1
s2
x / y

When does a reaction occur?

Suppose x and y are discrete and pure signals.

When does the transition occur?

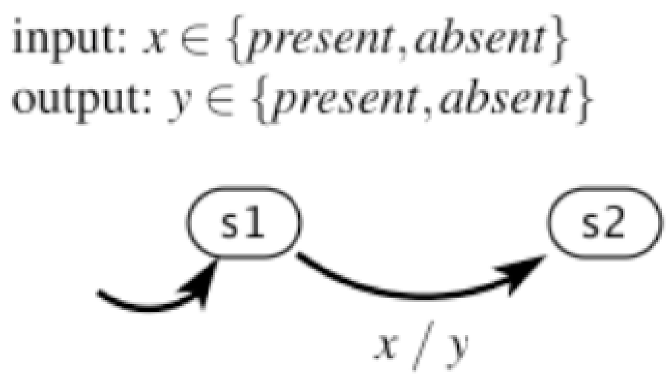
input: $x \in \{present, absent\}$
output: $y \in \{present, absent\}$

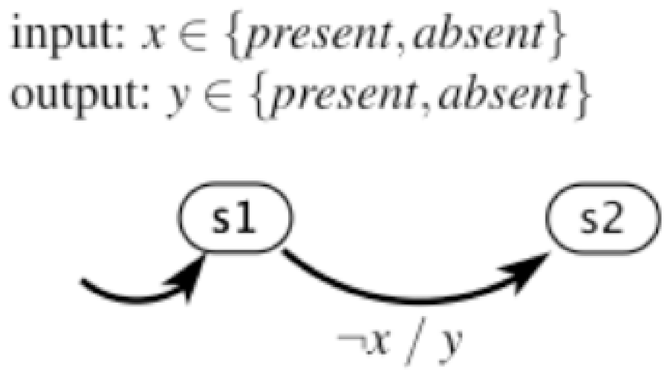


Answer: when the *environment* triggers a reaction and x is absent. If this is a (complete) event-triggered model, then the transition will never be taken because the reaction will only occur when x is present!

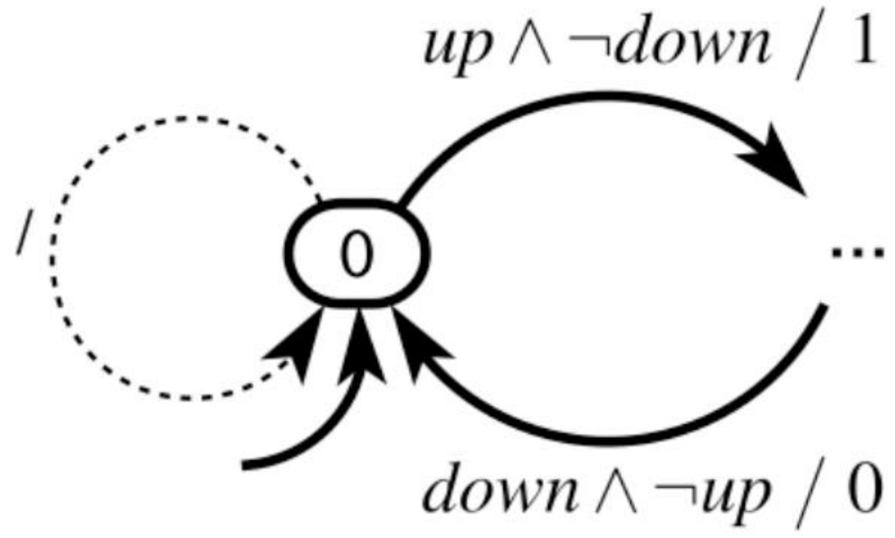
When does a reaction occur?

Time-triggered model: Suppose all inputs are discrete and a reaction occurs *on the tick of an* *external clock*.


input: $x \in \{present, absent\}$
output: $y \in \{present, absent\}$
s1
s2
$x\ /\ y$


input: $x \in \{present, absent\}$
output: $y \in \{present, absent\}$
s1
s2
$\neg x\ /\ y$

More Notation: Default Transitions

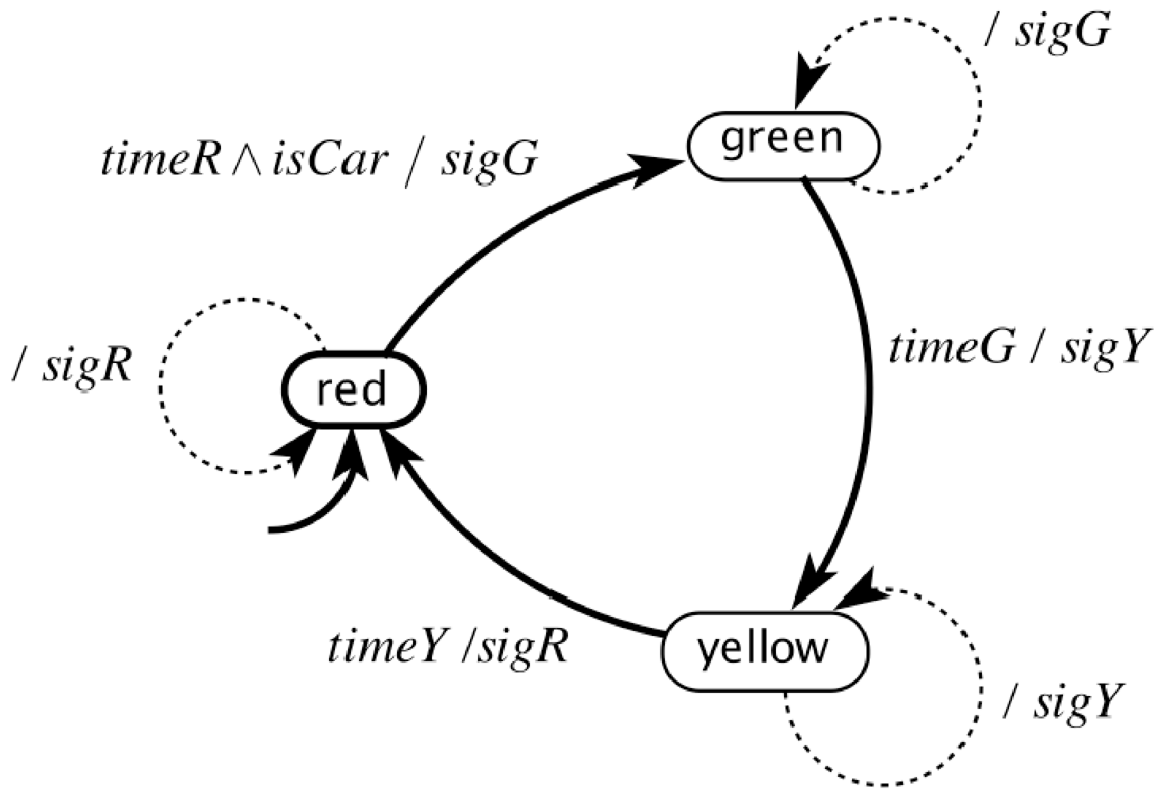


A default transition is enabled if no non-default transition is enabled and it either has no guard or the guard evaluates to true.

When is the above default transition enabled?

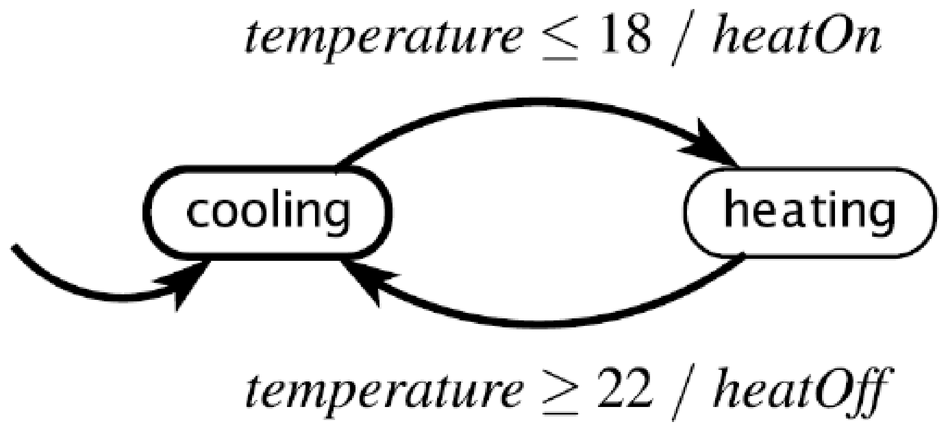
Example: Traffic Light Controller

Only show default transitions if they are guarded or produce outputs (or go to other states)



Example where default transitions need not be shown

input: *temperature* : $\mathbb{R}$
outputs: *heatOn*, *heatOff* : pure

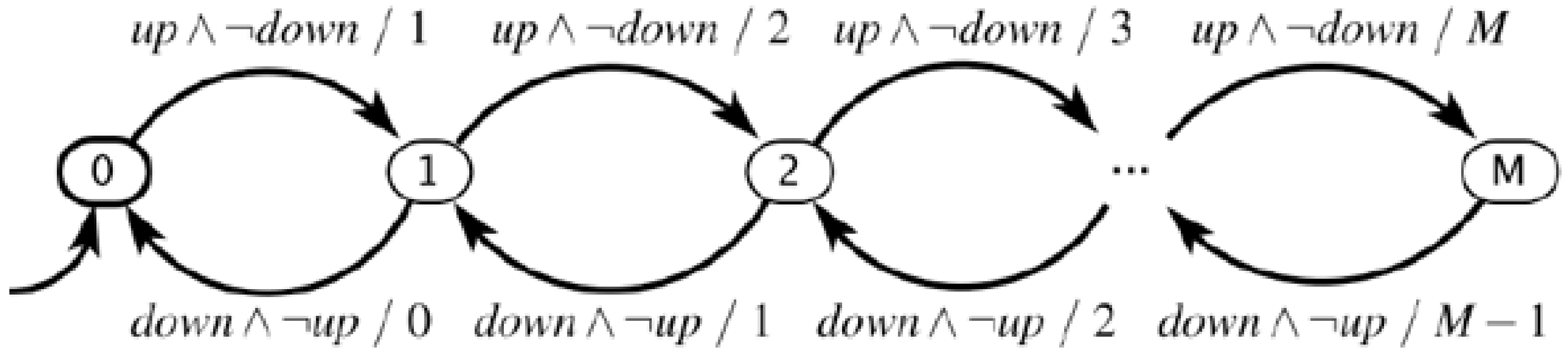

$temperature \leq 18$ / $heatOn$
cooling
heating
$temperature \geq 22$ / $heatOff$

Exercise: From this picture, construct the formal mathematical model.

FSM with large number of states

The notation for FSMs becomes awkward when the number of states gets large.

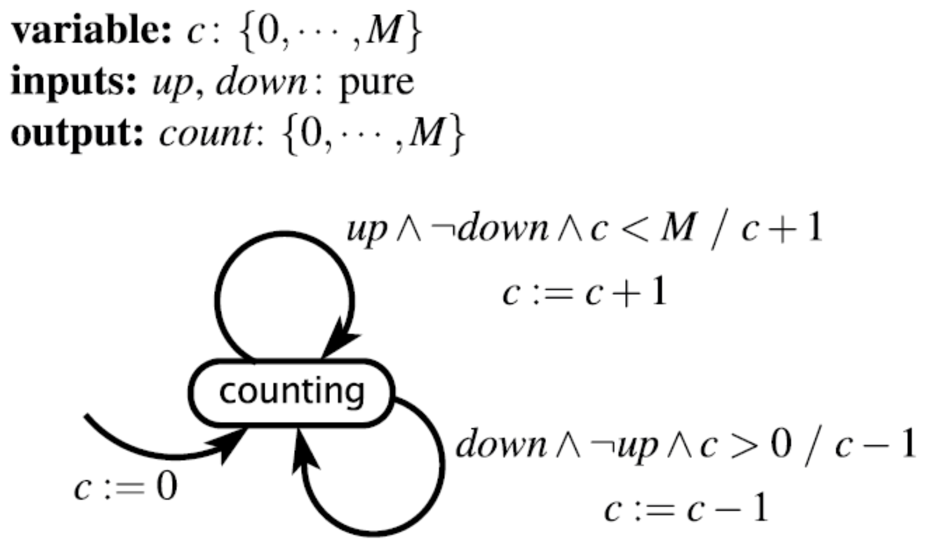
If M is large, the bubble-and-arc notation becomes unimageable, which is why we resort to a less formal use of "…" in the figure.



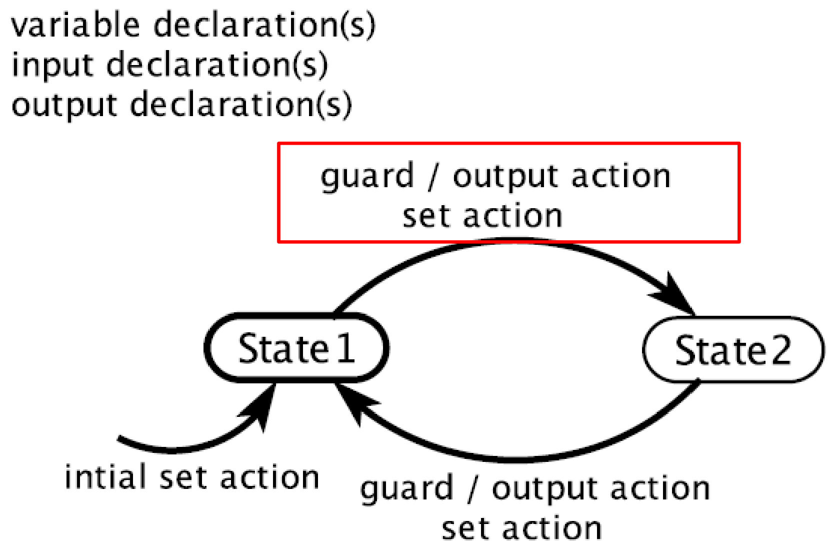
Extended State Machines

An extended state machine solves this problem by augmenting the FSM model with variables that may be read and written as part of taking a transition between states.

In example, a variable *c*, declared explicitly at the upper left to make it clear that *c* is a variable and not an input or an output.



variable: c: $\{0, \cdots, M\}$
inputs: up, $down$: pure
output: $count$: $\{0, \cdots, M\}$
$up \wedge \neg down \wedge c < M \;/\; c+1$
$c := c+1$
counting
$down \wedge \neg up \wedge c > 0 \;/\; c-1$
$c := c-1$
$c := 0$



Nondeterministic FSM

Most interesting state machines react to inputs and produce outputs.

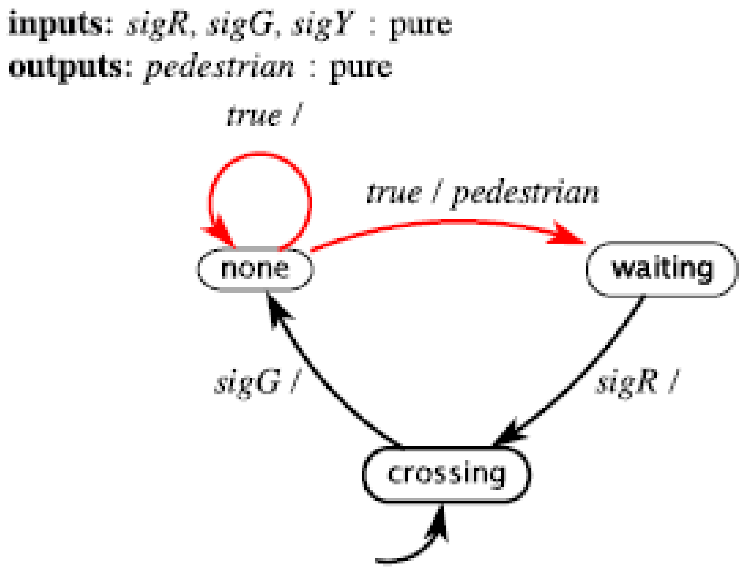
These inputs must come from somewhere, and the outputs must go somewhere. We refer to this “somewhere” as the environment of the state machine.

A question becomes how to model the environment.

If for any state of a state machine, there are two distinct transitions with guards that can evaluate to true in the same reaction, then the state machine is nondeterministic.

Example: Nondeterministic FSM

Nondeterministic model of pedestrians that arrive at a crosswalk.

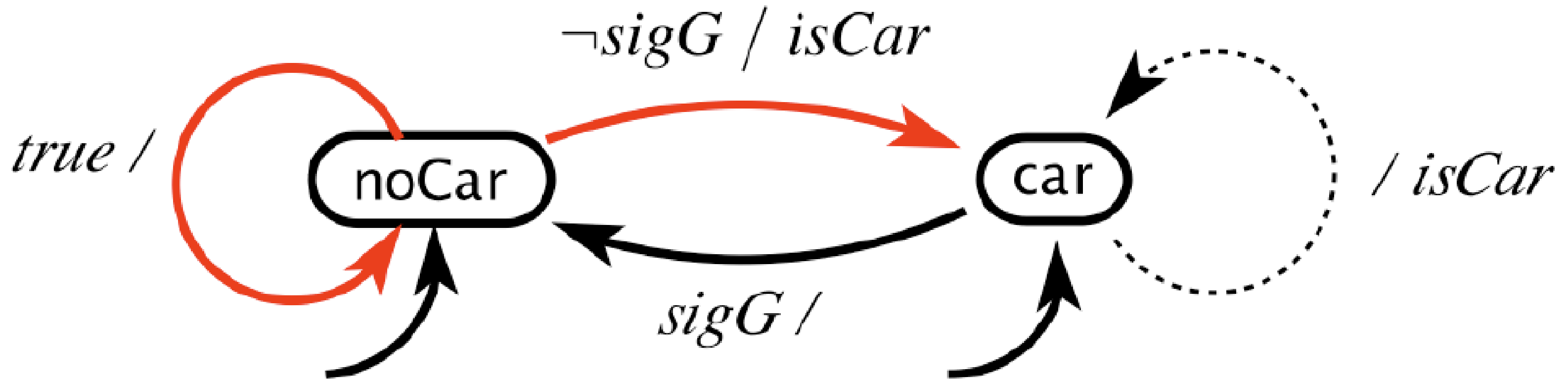


When *sigG* is received, the FSM transitions to *none*.
Both transitions from this state have guard true, indicating that they are always enabled.
Since both are enabled, this machine is nondeterministic.

Example: Nondeterministic FSM

Environment for a traffic light:

inputs: *sigG*, *sigR*, *sigY*: pure
output: *isCar*: pure



Formally, the update function is replaced by a function

$$possibleUpdates : States \times Inputs \to 2^{States \times Outputs}$$

Uses of Nondeterminism

1. Modeling unknown aspects of the environment or system
   - Such as: how the environment changes a robot’s orientation
2. Hiding detail in a specification of the system

Any other reasons why nondeterministic FSMs might be preferred over deterministic FSMs?

Behaviors

An FSM has discrete dynamics.

We can abstract away the passage of time and consider only the sequence of reactions (the behavior of an FSM, without concern for when in time each reaction occurs.

Consider a port p of a state machine with type Vp, so this port will have a sequence of values from the set $Vp \cup \{absent\}$

$$s_p : \mathbb{N} \to V_p \cup \{absent\} .$$

A **behavior of a state machine** is an assignment of such a signal to each port such that the signal on any output port is the output sequence produced for the given input signals.

Behaviors

A behavior may be more conveniently represented as a sequence of valuations called an **observable trace**.

Let x_i represent the valuation of the input ports and y_i the valuation of the output ports at reaction i.

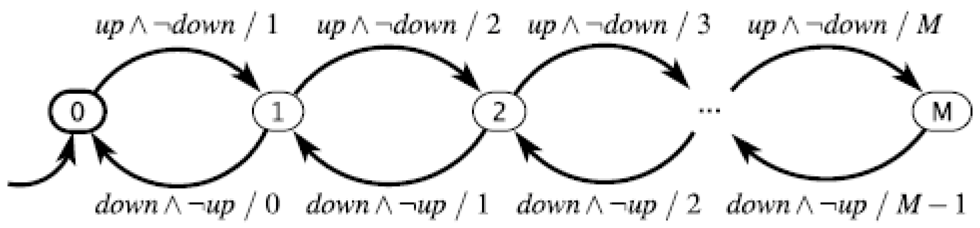
$$((x_0, y_0), (x_1, y_1), (x_2, y_2), \cdots) \; .$$

An **execution trace** includes the state trajectory

$$((x_0, s_0, y_0), (x_1, s_1, y_1), (x_2, s_2, y_2), \cdots) \; ,$$

$$s_0 \xrightarrow{x_0/y_0} s_1 \xrightarrow{x_1/y_1} s_2 \xrightarrow{x_2/y_2} \cdots$$

Example



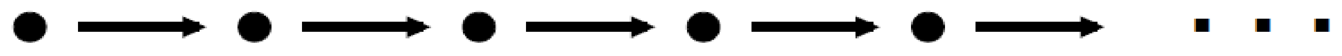
$$0 \xrightarrow{up \wedge down\ /} 0 \xrightarrow{\ \ /\ \ } 0 \xrightarrow{up\ /\ 1} 1 \xrightarrow{down\ /\ 0} 0 \xrightarrow{up\ /\ 1} \cdots$$

Non-deterministic Behavior: Tree of Computations

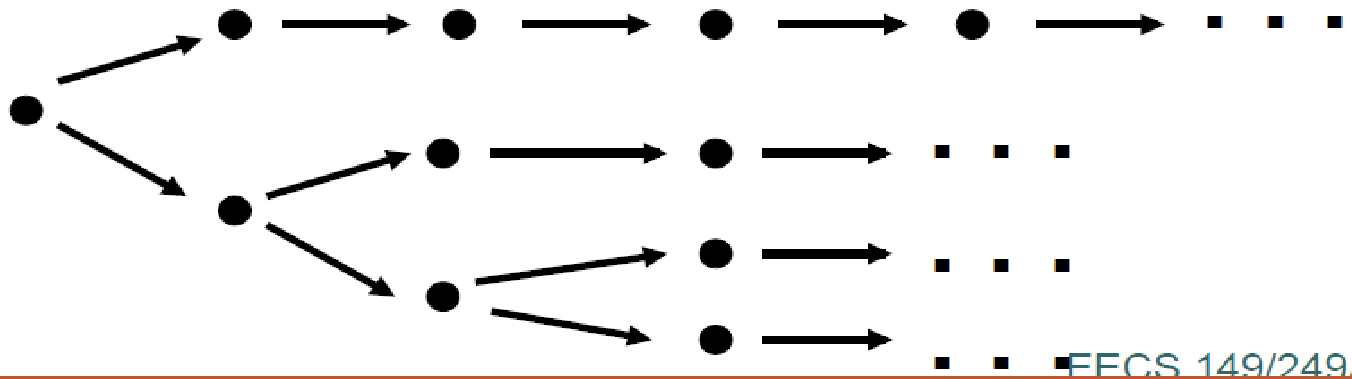
For a fixed input sequence:

- A deterministic system exhibits a single behavior
- A non-deterministic system exhibits a **set of behaviors**
  - visualized as a *computation tree*

Deterministic FSM behavior:



Non-deterministic FSM behavior:



Non-deterministic ≠ Probabilistic (Stochastic)

In a probabilistic FSM, each transition has an associated probability with which it is taken.

In a non-deterministic FSM, no such probability is known.

We just know that any of the enabled transitions from a state can be taken.

Some Definitions

Stuttering transition: (possibly implicit) default transition that is enabled when inputs are absent, that does not change state, and that produces absent outputs.

Receptiveness: For any input values, some transition is enabled. Our structure together with the implicit default transition ensures that our FSMs are receptive.

Determinism: In every state, for all input values, exactly one (possibly implicit) transition is enabled.

Reference

- Lee, Edward & Seshia, Sanjit. (2011). Introduction to Embedded Systems - A Cyber-Physical Systems Approach.
- Lecture Note Slides from EECS 149/249A: Introduction to Embedded Systems (UC Berkeley) by Prof. Prabal Dutta and Sanjit A. Seshia