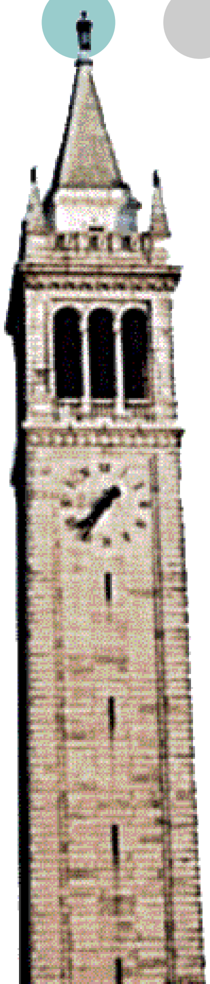
Introduction to Embedded Systems



Sanjit A. Seshia
UC Berkeley

EECS 149/249A
Fall 2015

Chapter 3: Discrete Dynamics, State Machines

Discrete Systems

Discrete = "individually separate / distinct"

A **discrete system** is one that operates in a sequence of discrete *steps* or has signals taking discrete *values*.

It is said to have **discrete dynamics**.

Concepts covered in Today's Lecture

Models = Programs

Actor Models of Discrete Systems: Types and Interfaces

States, Transitions, Guards

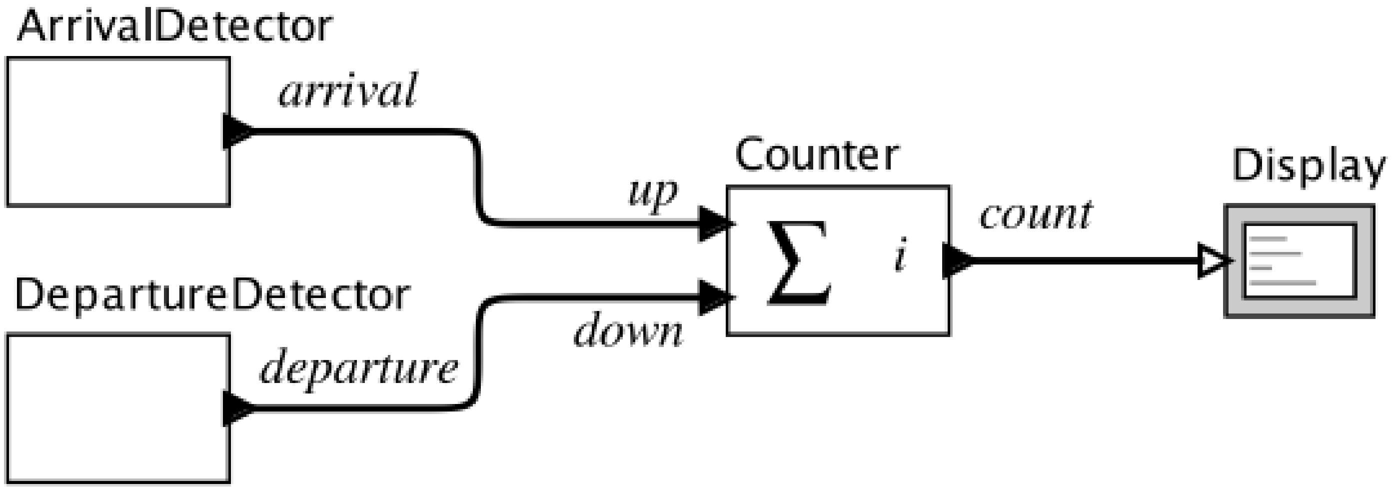
Determinism and Receptiveness

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.

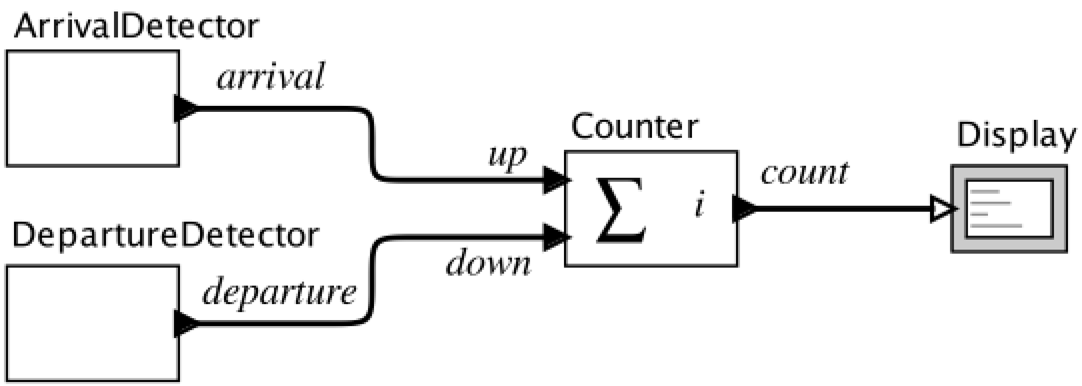
Discrete Systems

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



Discrete Systems

Example: count the number of cars that enter and leave a parking garage:

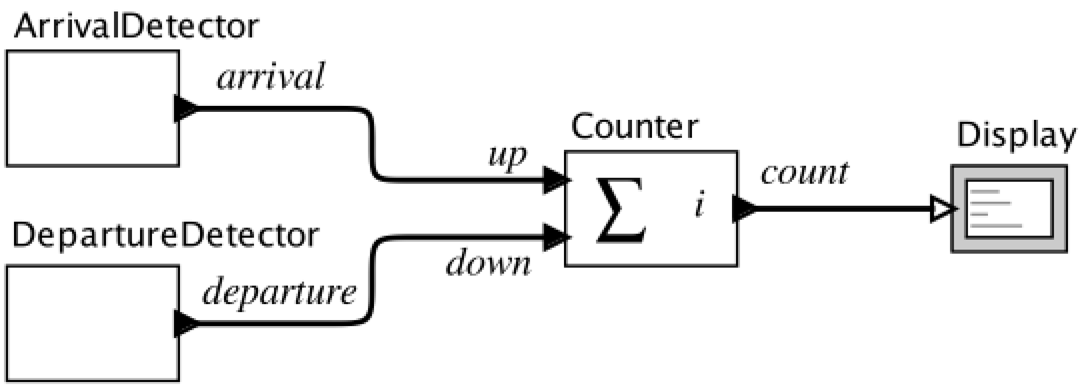


Pure signal: $up : \mathbb{R} \to \{absent, present\}$

(can be modeled as Boolean data)(N. Cleju)

Discrete Systems

Example: count the number of cars that enter and leave a parking garage:



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

Discrete actor:

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

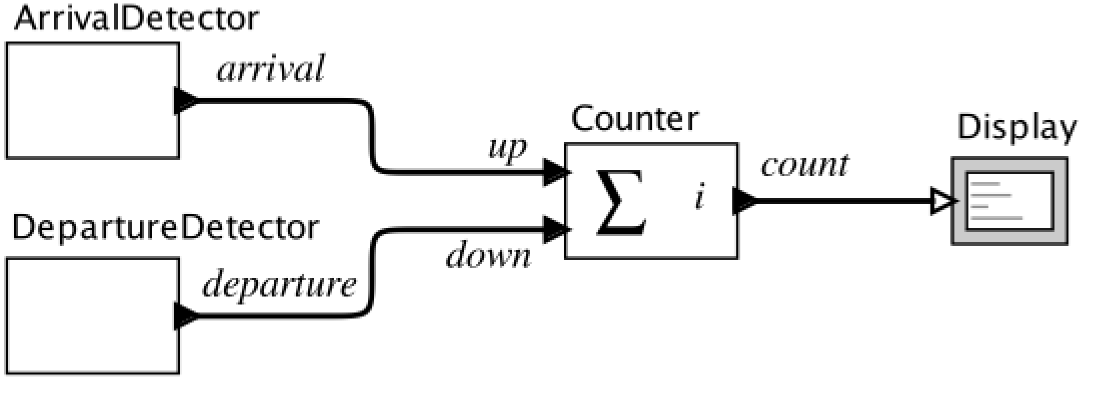
$$P = \{up, down\}$$

Reaction / Transition

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



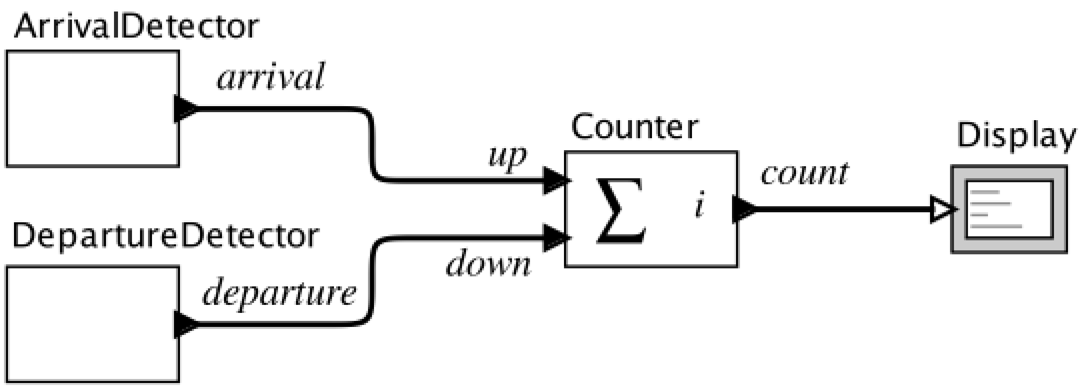
Inputs and Outputs at a Reaction

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

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

and the outputs are in a set

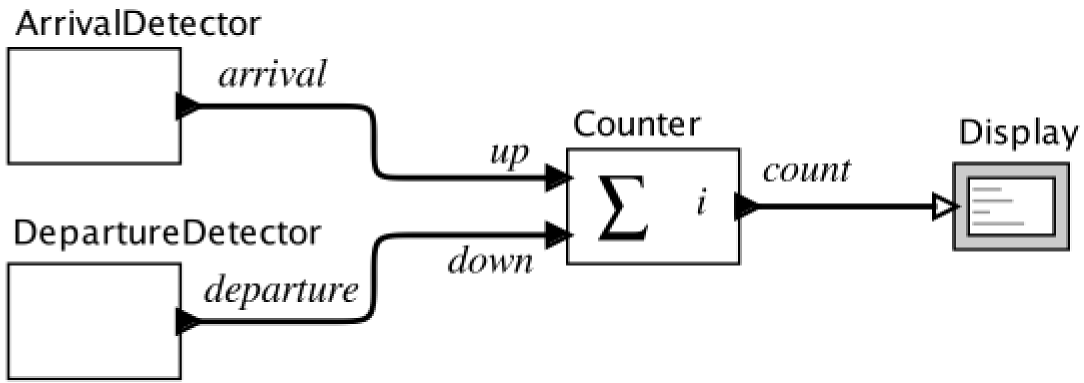
$$Outputs = (\{count\} \to \{\, absent\} \cup \mathbb{N}) \;,$$



State Space

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

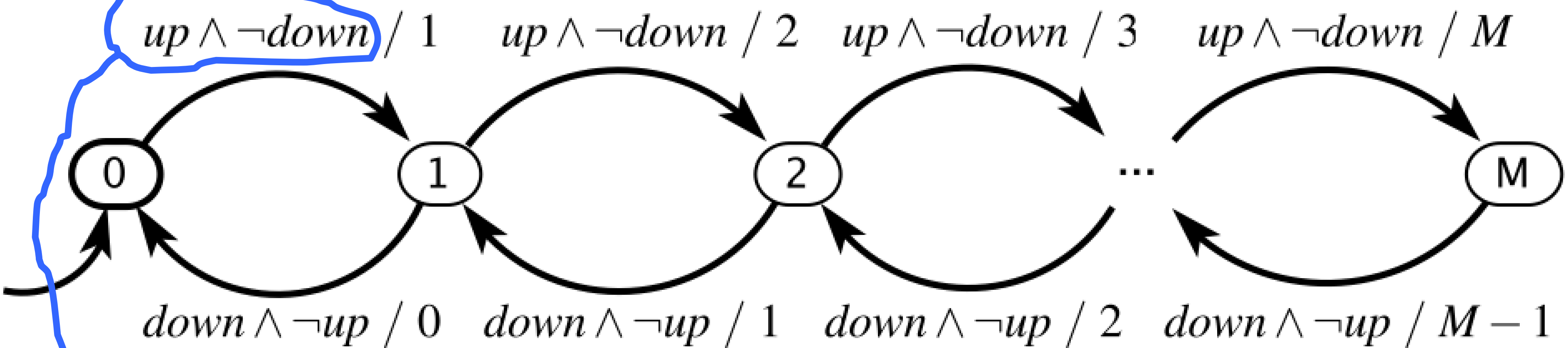
$$States = \{0, 1, 2, \cdots, M\}\ .$$



Question

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

Garage Counter Finite State Machine (FSM) in Pictures



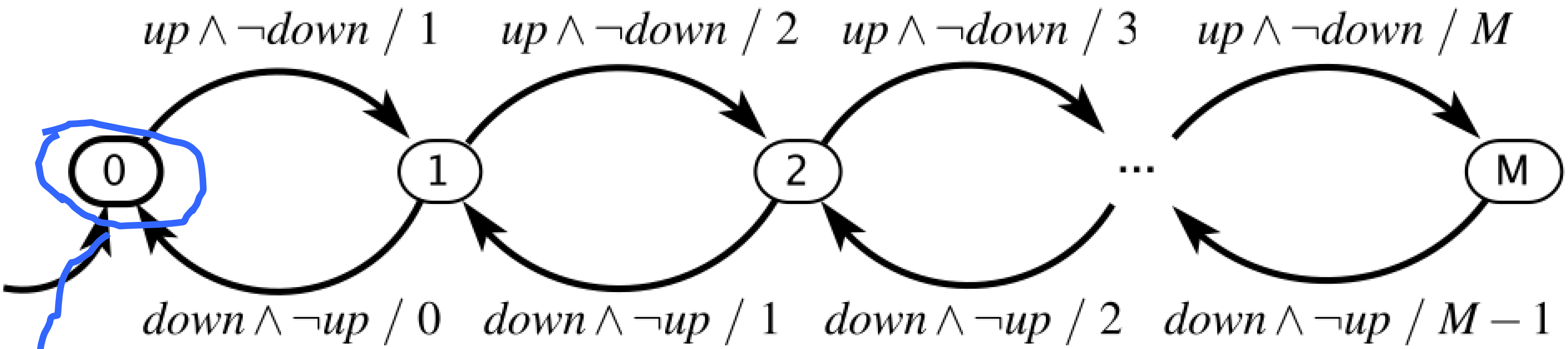
Guard $g \subseteq Inputs$ is specified using the shorthand

$$up \wedge \neg down$$

which means

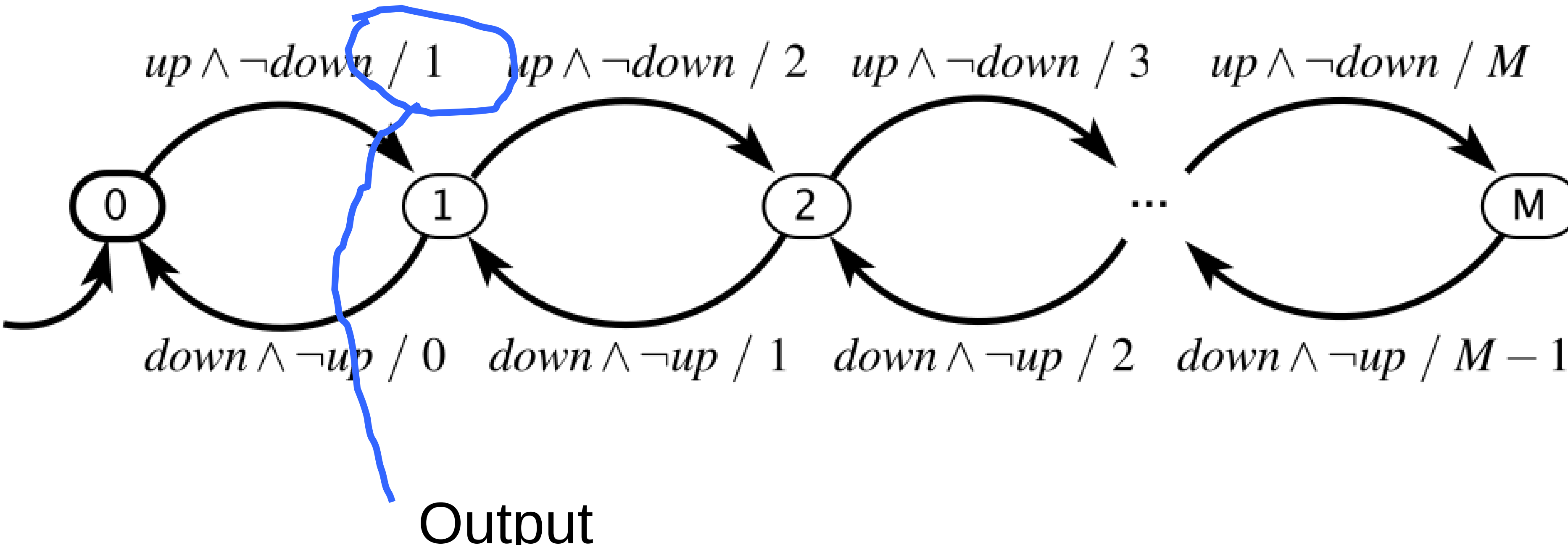
$$g = \{\{up\}\} \ .$$

Garage Counter Finite State Machine (FSM) in Pictures

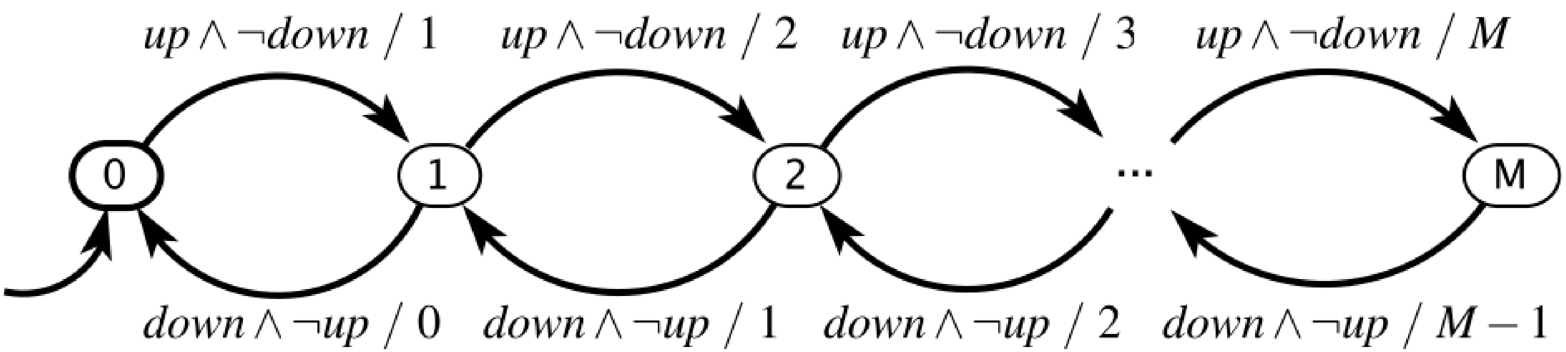


Initial state

Garage Counter Finite State Machine (FSM) in Pictures



Garage Counter Mathematical Model

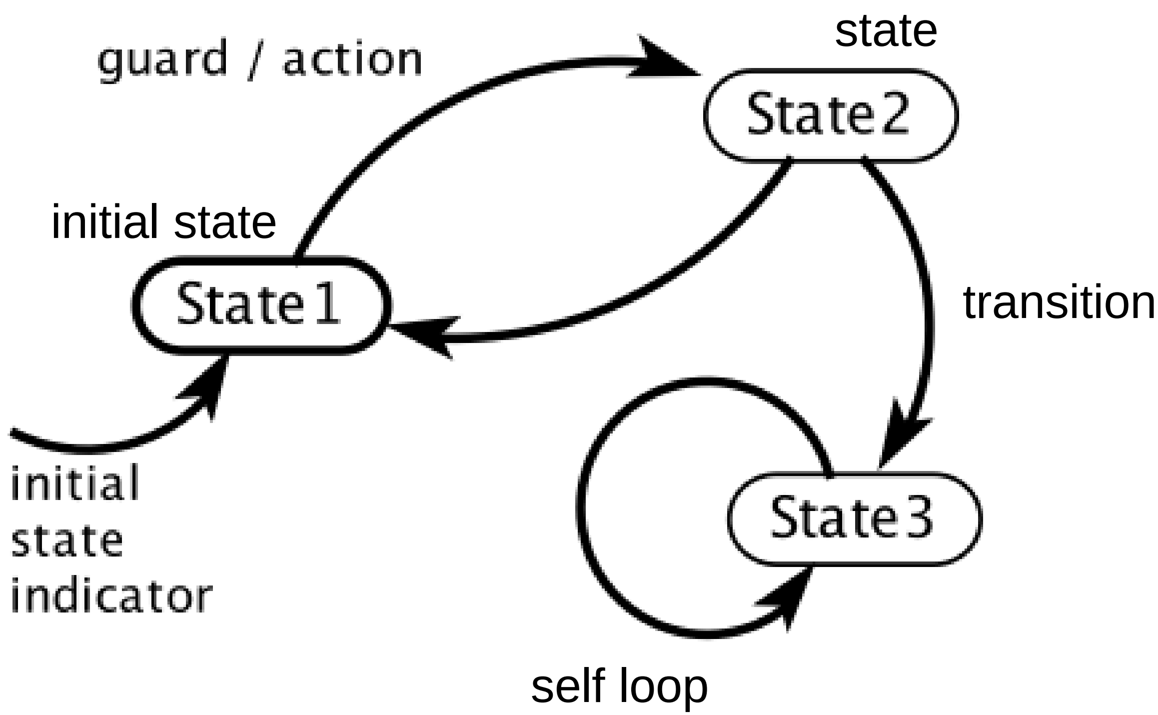


Formally: $(States, Inputs, Outputs, update, initialState)$, where

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

The picture above defines the update function.

FSM Notation



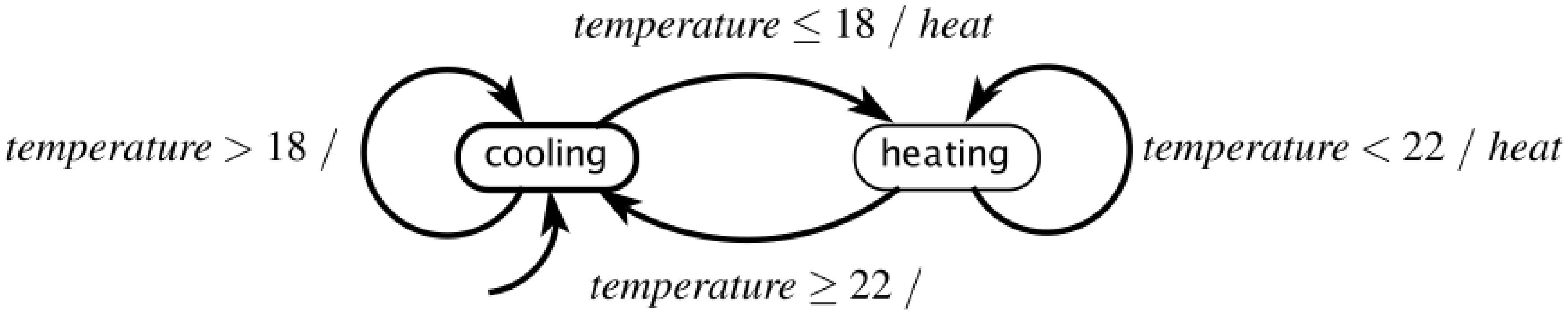
Examples of Guards for Pure Signals

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 $absent$.

Examples of Guards for Signals with Numerical Values

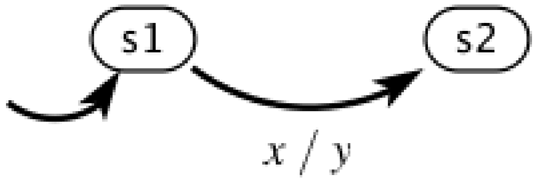
p_3	Transition is enabled if p_3 is *present* (not *absent*).
$p_3 = 1$	Transition is enabled if p_3 is *present* and has value 1.
$p_3 = 1 \land p_1$	Transition is enabled if p_3 has value 1 and p_1 is *present*.
$p_3 > 5$	Transition is enabled if p_3 is *present* with value greater than 5.

Example of *Modal* Model: Thermostat



When does a reaction occur?

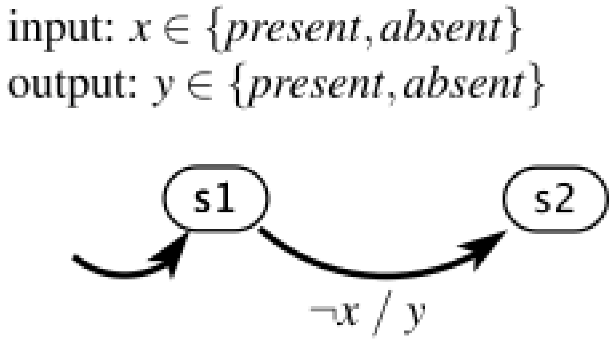
input: $x \in \{present, absent\}$
output: $y \in \{present, absent\}$



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.

This is an *event-triggered model*.

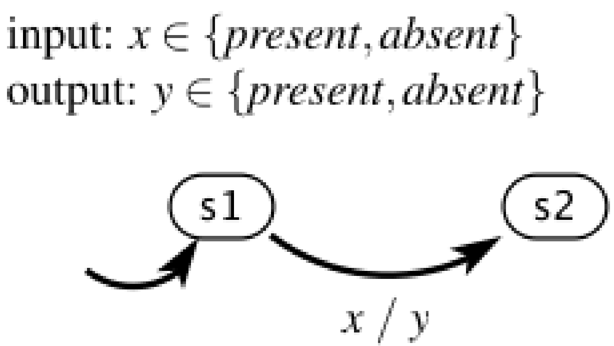
When does a reaction occur?

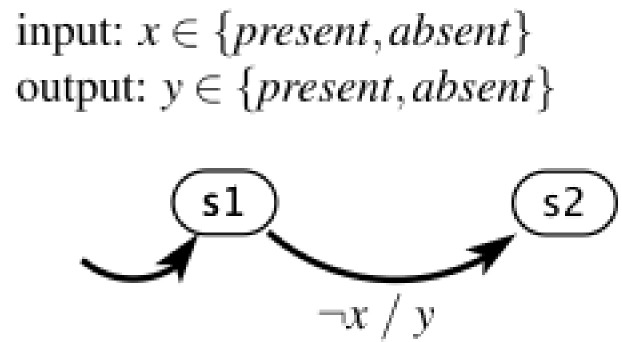


Suppose *x* and *y* are discrete and pure signals. When does the transition occur?

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?

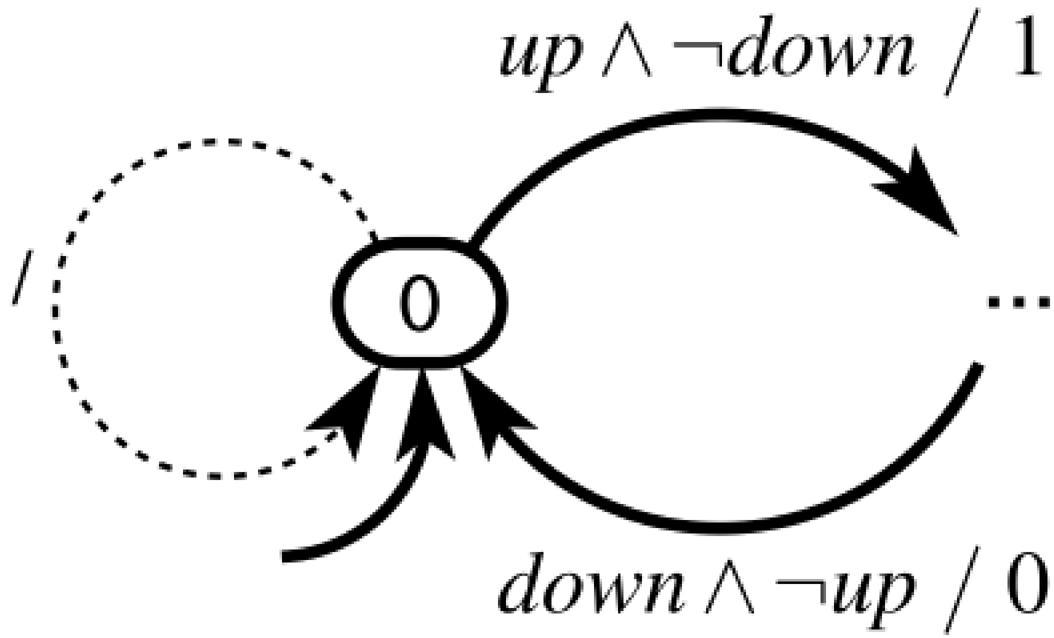




Suppose all inputs are discrete and a reaction occurs *on the tick of an external clock*.

This is a *time-triggered model*.

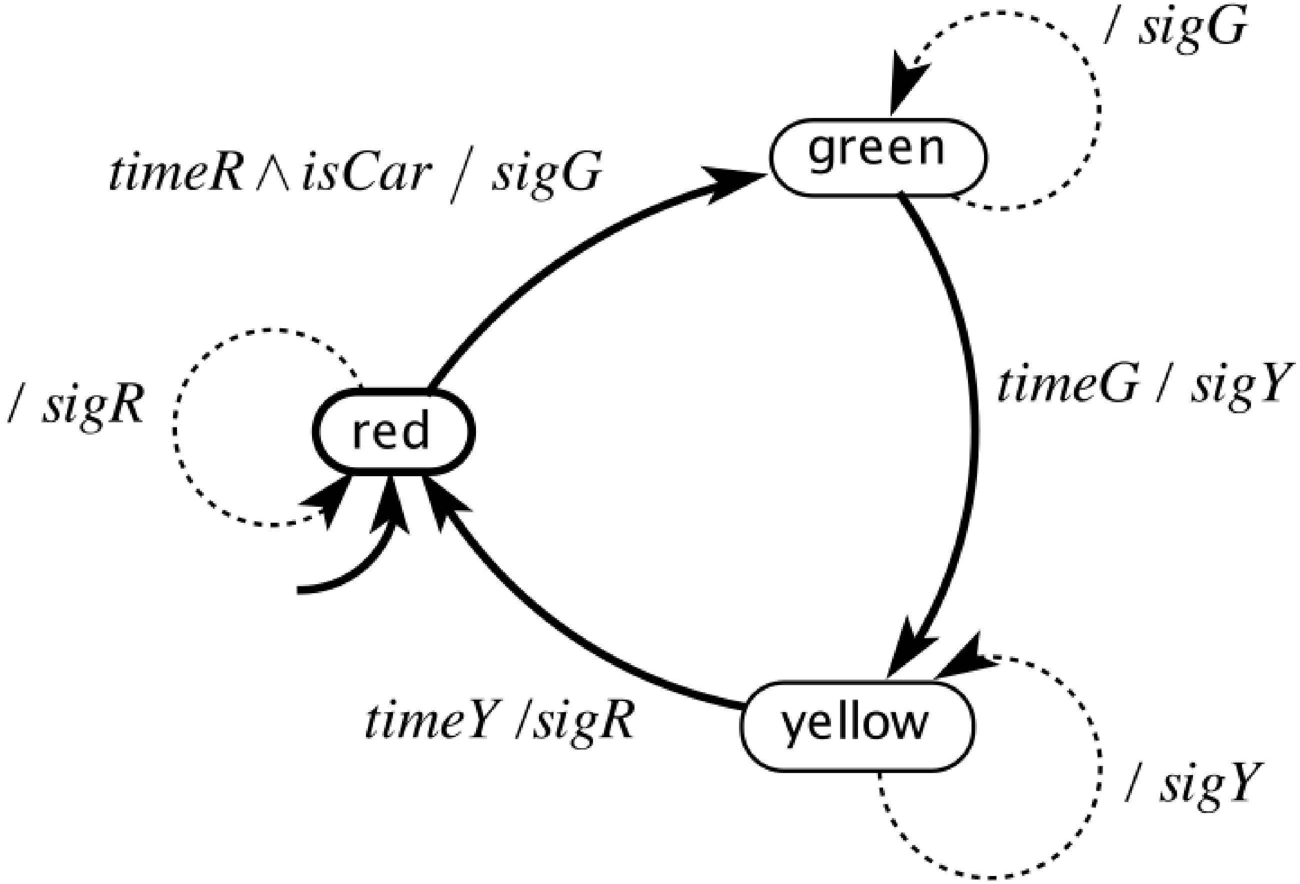
More Notation: Default Transitions



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?

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

Example: Traffic Light Controller



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.

Test Your Understanding: Three Kinds of Transitions

Self-Loop

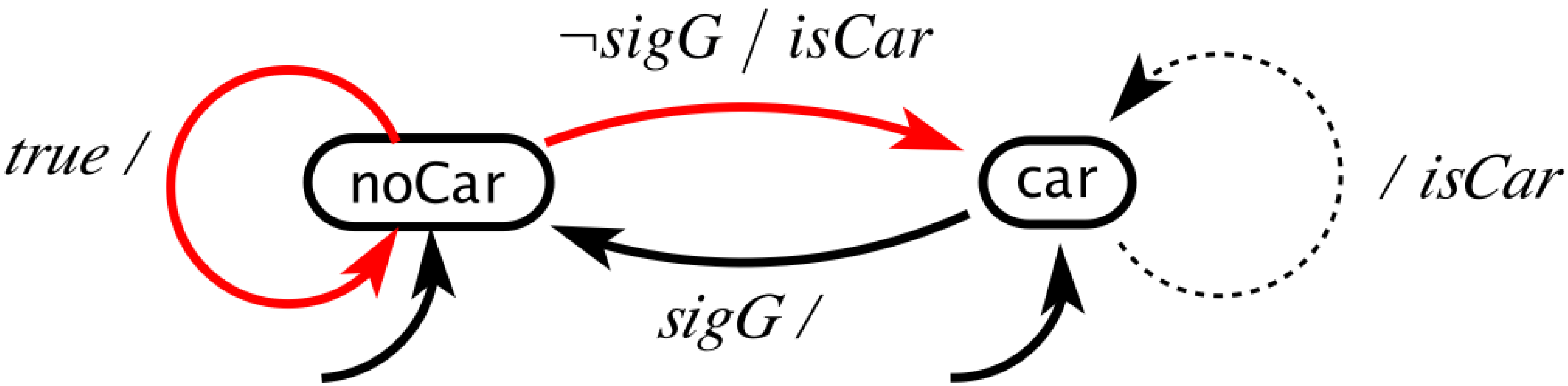
Default Transition

Stuttering Transition

1. Is a default transition always a self-loop?
2. Is a stuttering transition always a self-loop?
3. Is a self-loop always stuttering?

Example: Nondeterministic FSM

Model of the environment for a traffic light, abstracted using nondeterminism:



Formally, the update function is replaced by a function

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

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
   - We will see an example of this later (see the text)

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

Size Matters

Non-deterministic FSMs are more compact than deterministic FSMs

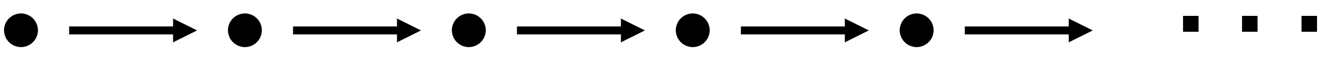
- A classic result in automata theory shows that a nondeterministic FSM has a related deterministic FSM that is equivalent in a technical sense (language equivalence, covered in Chapter 13, for FSMs with finite-length executions).
- But the deterministic machine has, in the worst case, many more states (exponential in the number of states of the nondeterministic machine, see Appendix B).

Non-deterministic Behavior: Tree of Computations

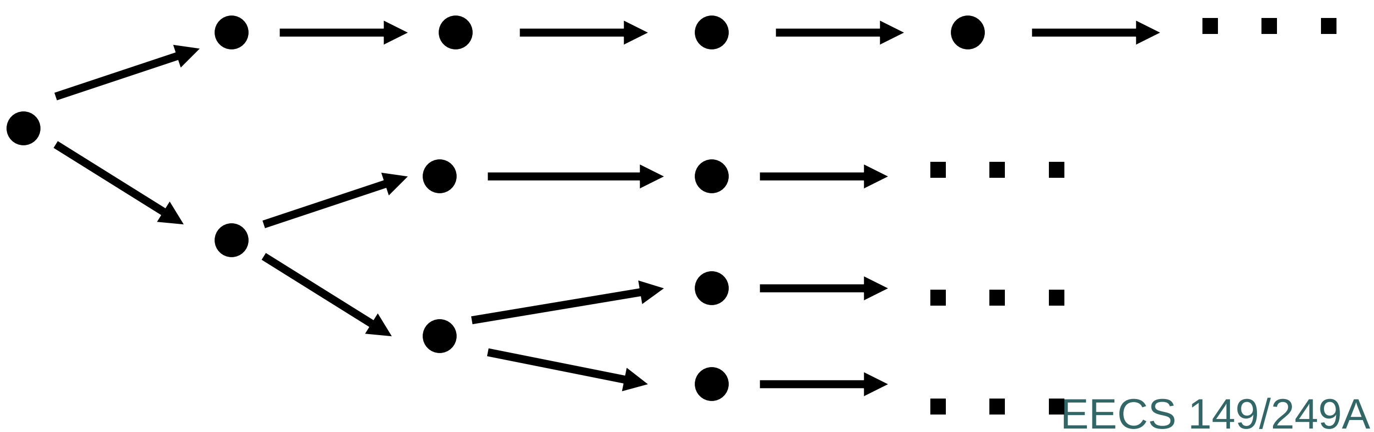
For a fixed input sequence:

- 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 $\neq$ 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.

Review: Concepts covered

Models = Programs

Actor Models of Discrete Systems: Types and Interfaces

States, Transitions, Guards

Determinism, Receptiveness, etc.