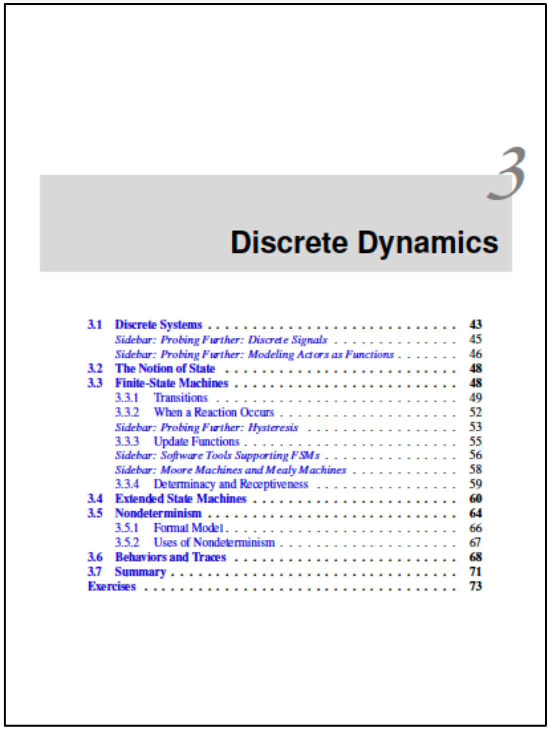
Lectures 9/10 : Discrete Dynamics

Slides were originally developed by Profs. Edward Lee and Sanjit Seshia, and subsequently updated by Profs. Gavin Buskes and Iman Shames.

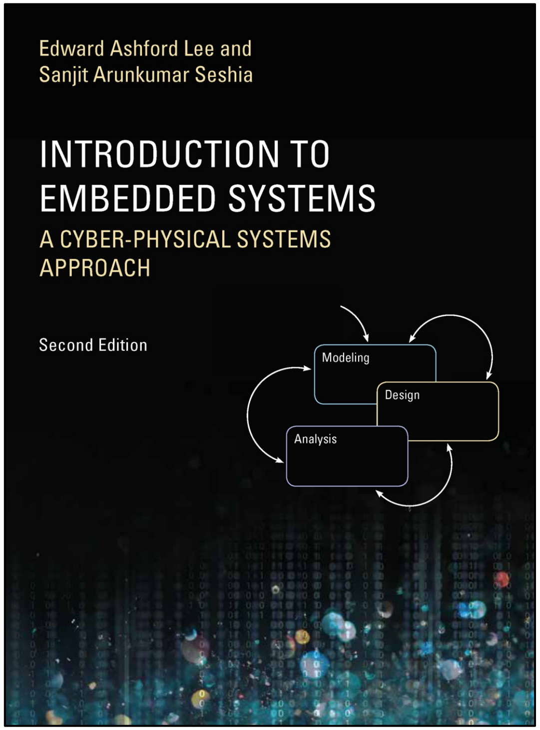
Outline

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



3

Discrete Dynamics



Discrete Systems

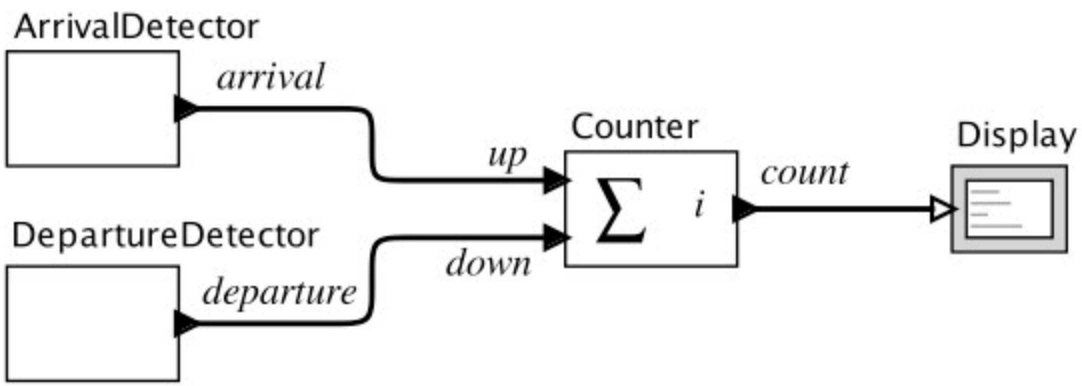
- Discrete = “individually separate / distinct”

- 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**.

Discrete Systems : Example Design Problem

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



- Pure signal: $up : \mathbb{R} \rightarrow \{absent, present\}$
- Discrete actor:

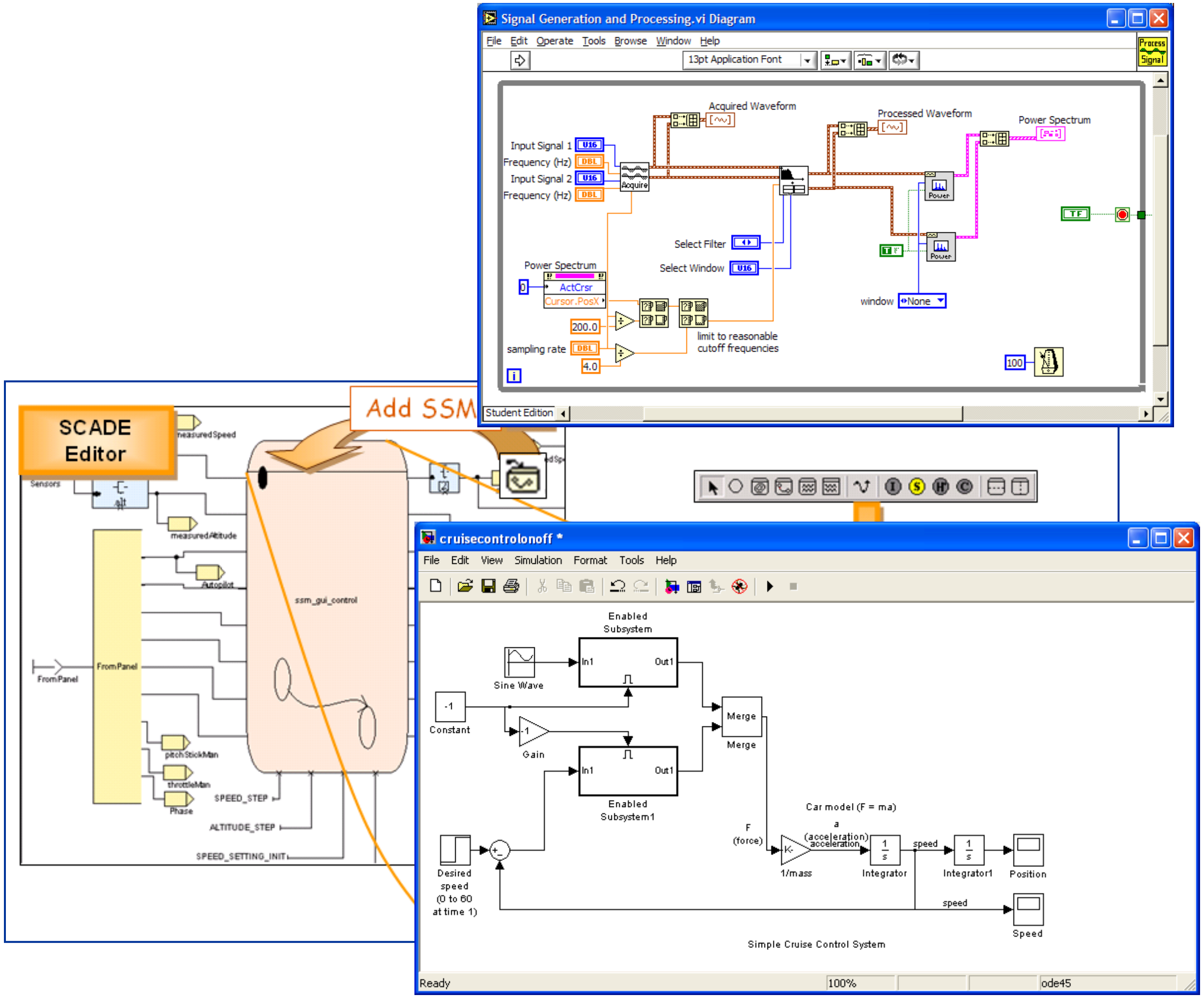
$$Counter : (\mathbb{R} \rightarrow \{absent, present\})^P \rightarrow (\mathbb{R} \rightarrow \{absent\} \cup \mathbb{N})$$

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

Actor Modeling Languages / Frameworks

- LabVIEW
- Simulink
- Scade
- …

- Reactors
- StreamIT
- …

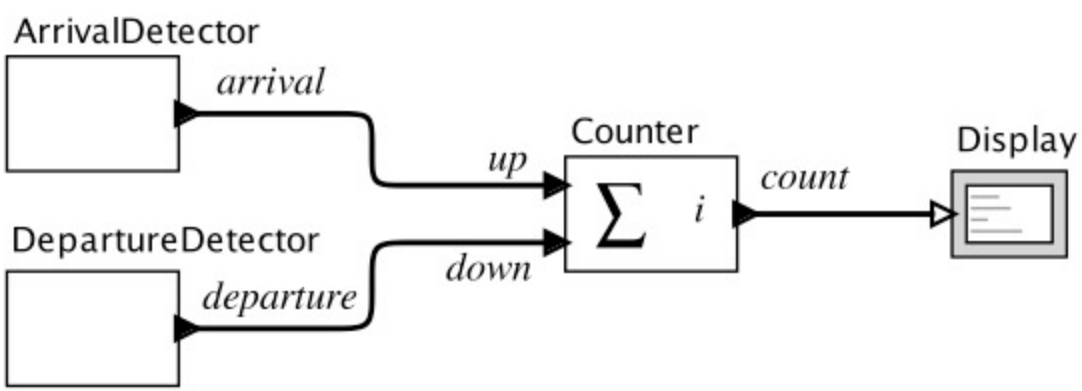


Reaction / Transition

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



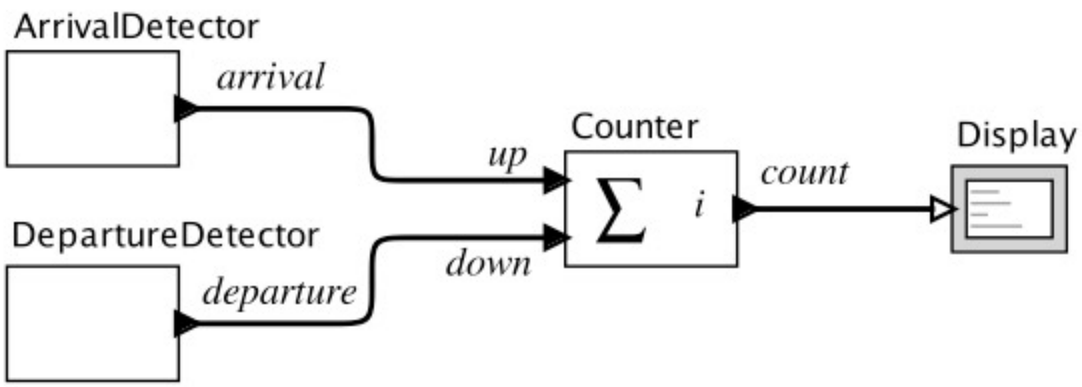
Inputs and Outputs at a Reaction

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

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

and the outputs are in a set

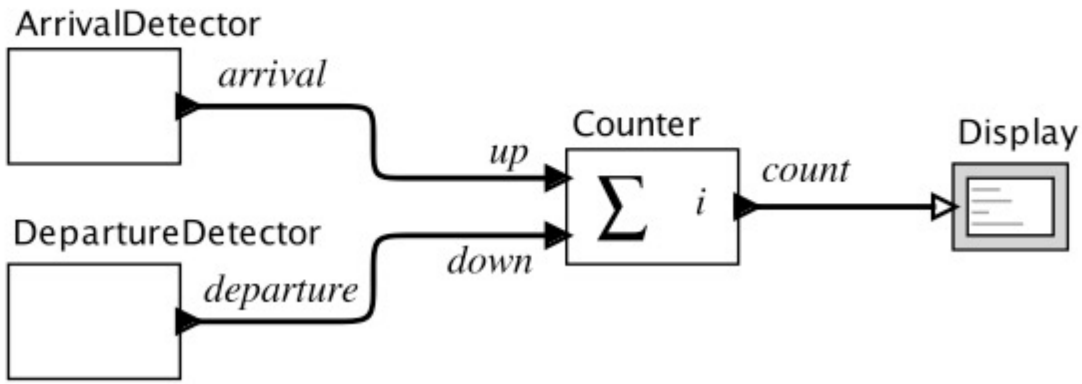
$$Outputs = (\{count\} \to \{\, absent\} \cup \mathbb{N}) \ ,$$



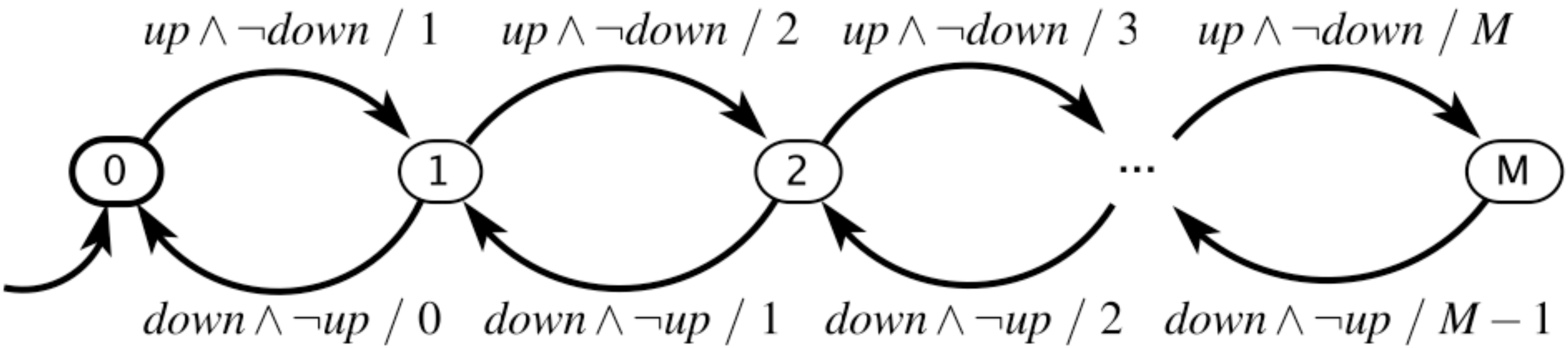
State Space

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

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



Garage Counter Finite State Machine (FSM) in Pictures



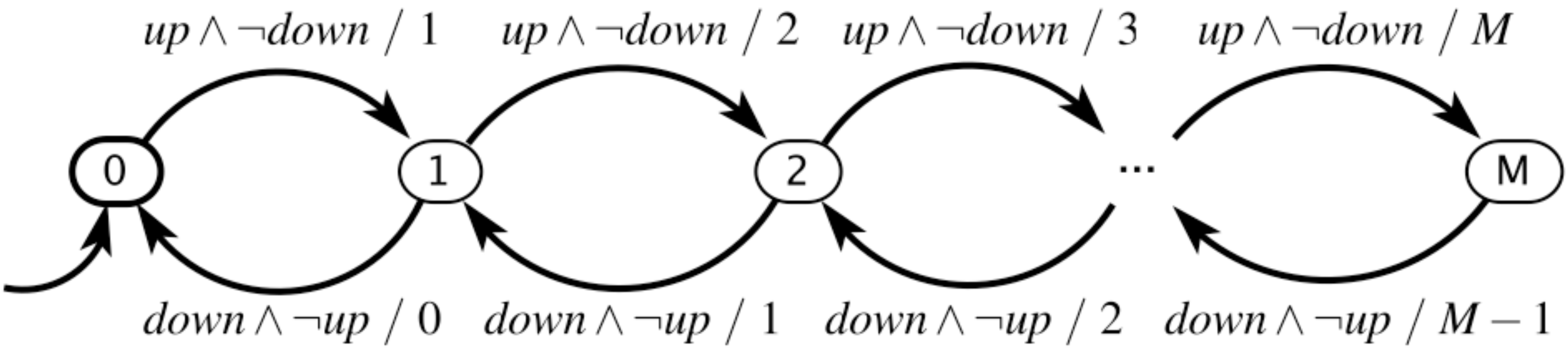
Guard $g \subseteq Inputs$ is specified using the shorthand

$$up \wedge \neg down$$

which means

Inputs(*up*) = *present*, *Inputs*(*down*) = *absent*

Garage Counter Finite State Machine (FSM) in Pictures

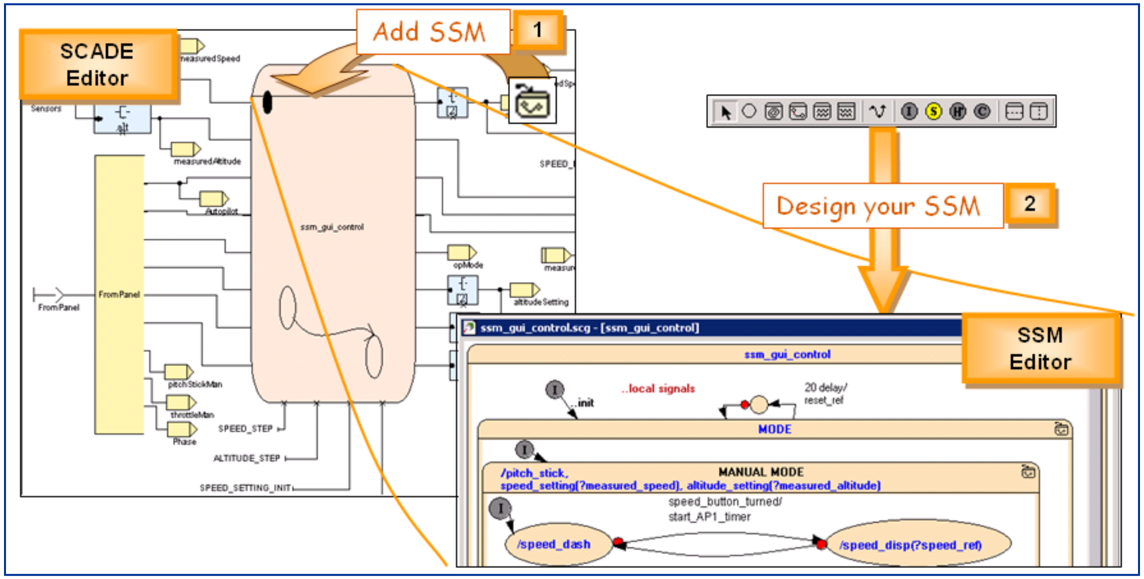


Output

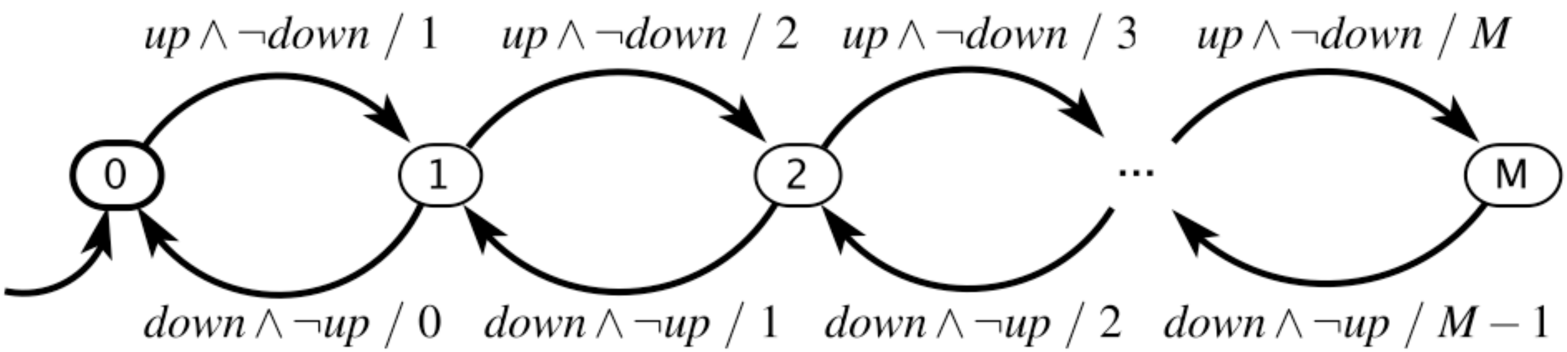
Initial state

FSM Modeling Languages / Frameworks

- LabVIEW Statecharts
- Simulink Stateflow
- Scade
- …



Garage Counter Mathematical Model



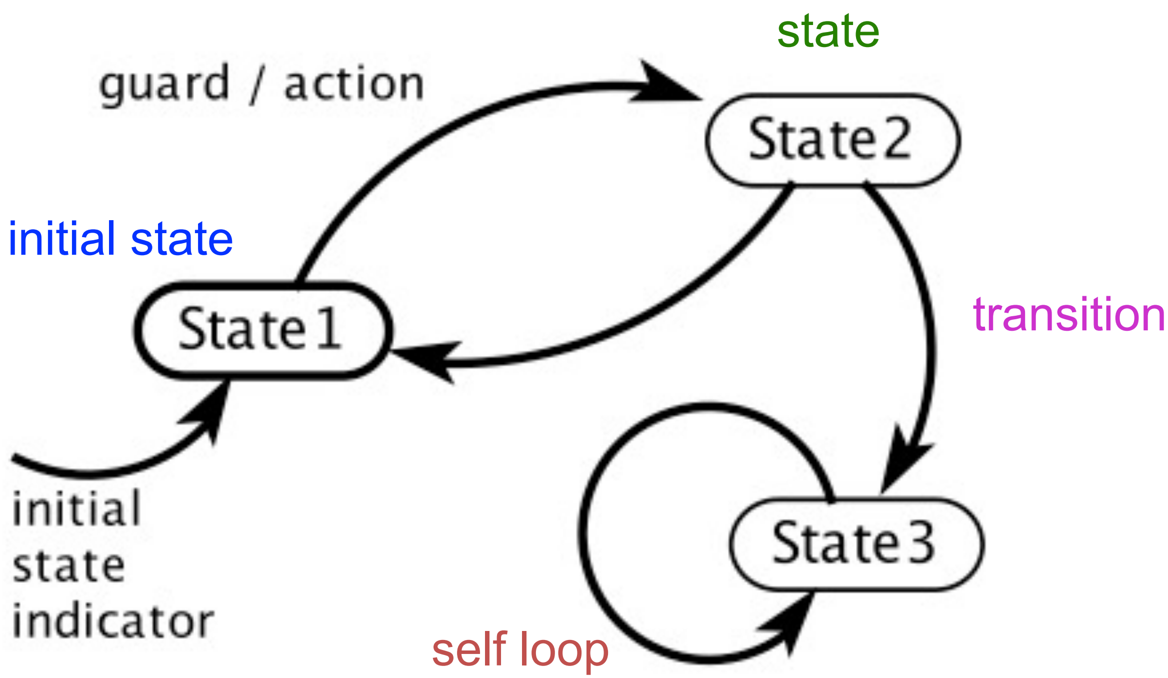
Formally: $(States, Inputs, Outputs, update, initialState)$, where

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

The picture above defines the update function.

FSM Notation in Lee & Seshia

Input declarations
Output declarations
Extended state declarations



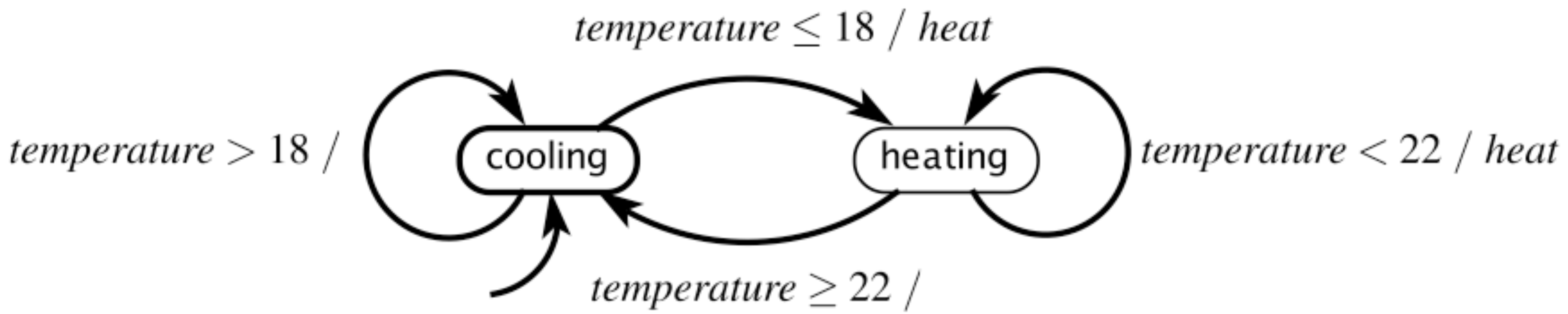
Examples of Guards for Pure Signals

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 $absent$.

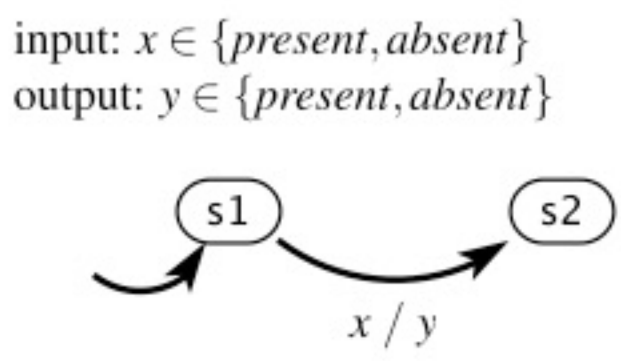
Examples of Guards for Signals with Numerical Values

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.
$p_3 = 1 \wedge p_1$	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.

Example of *Modal* Model: Thermostat



When does a reaction occur?

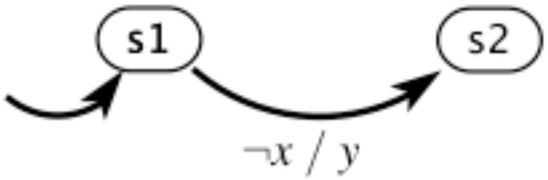


- Suppose all inputs are discrete and a reaction occurs *when any input is present*. Then the above transition will be taken whenever the current state is s1 and *x* is present.

- This is an *event-triggered model*.

When does a reaction occur?

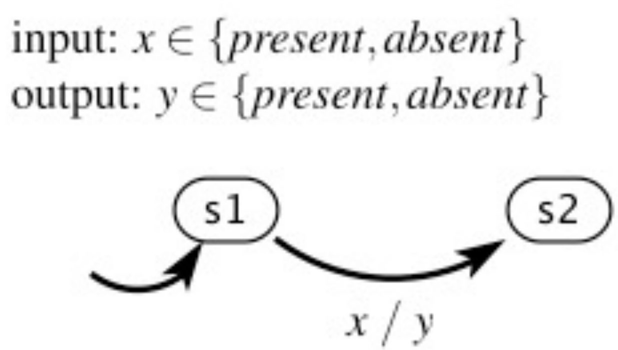
input: $x \in \{present, absent\}$
output: $y \in \{present, absent\}$

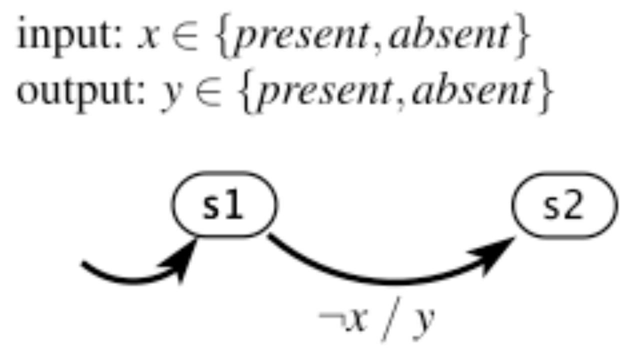


- Suppose *x* and *y* are discrete and pure signals. When does the transition occur?

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?

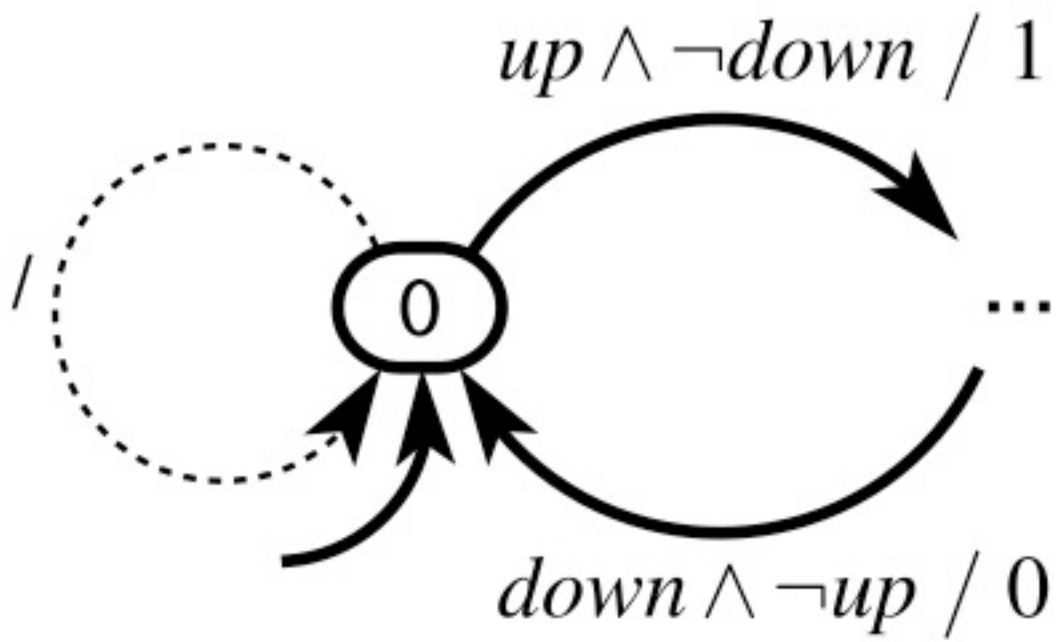




- Suppose all inputs are discrete and a reaction occurs *on the tick of an external clock*.

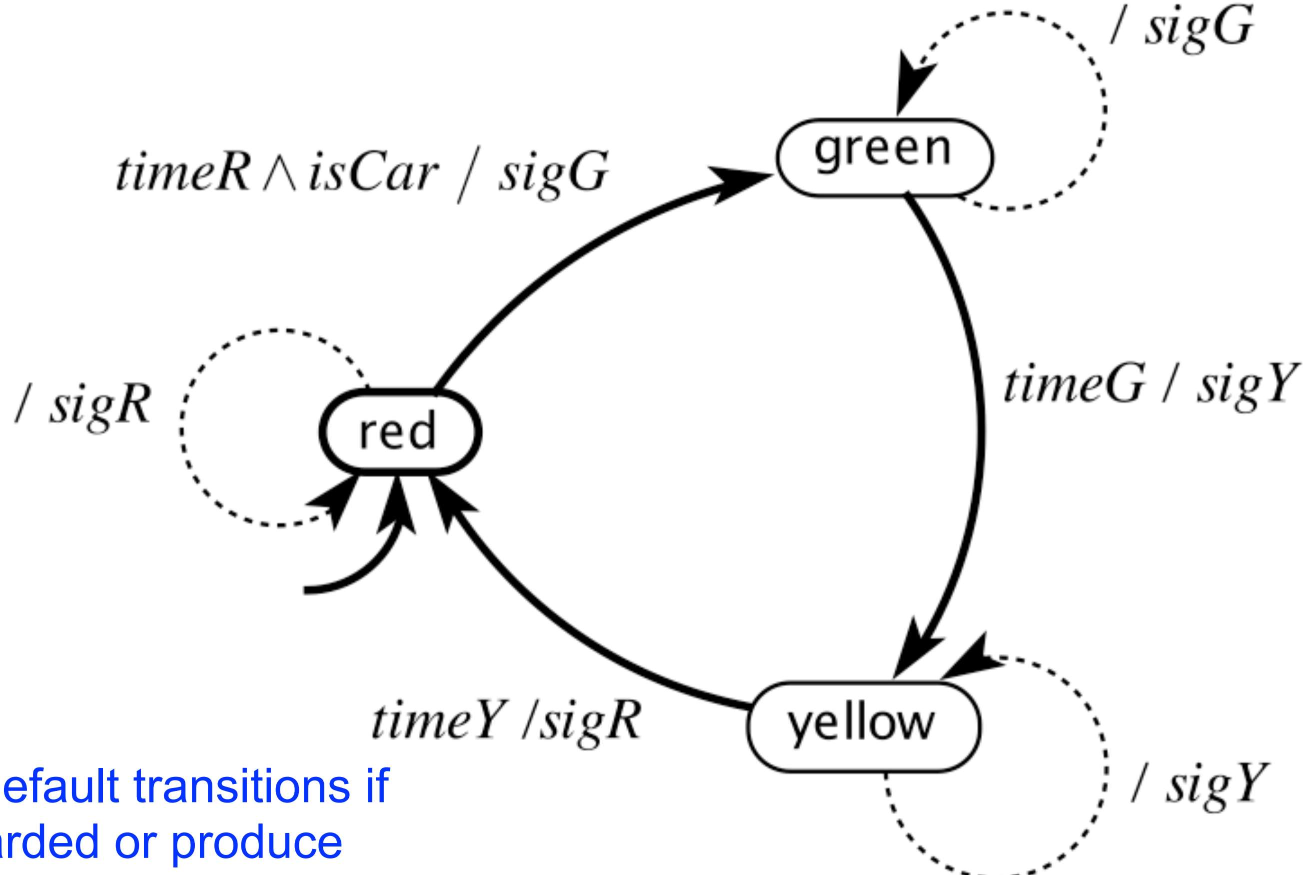
- This is a *time-triggered model*.

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?

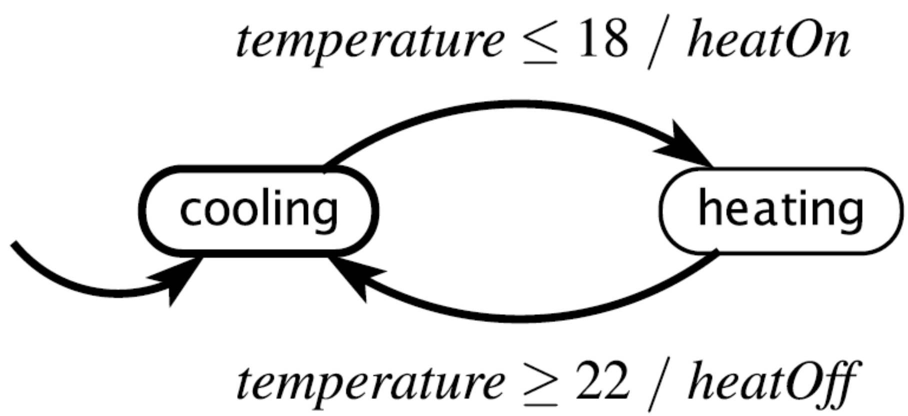
Example: Traffic Light Controller



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



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

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.

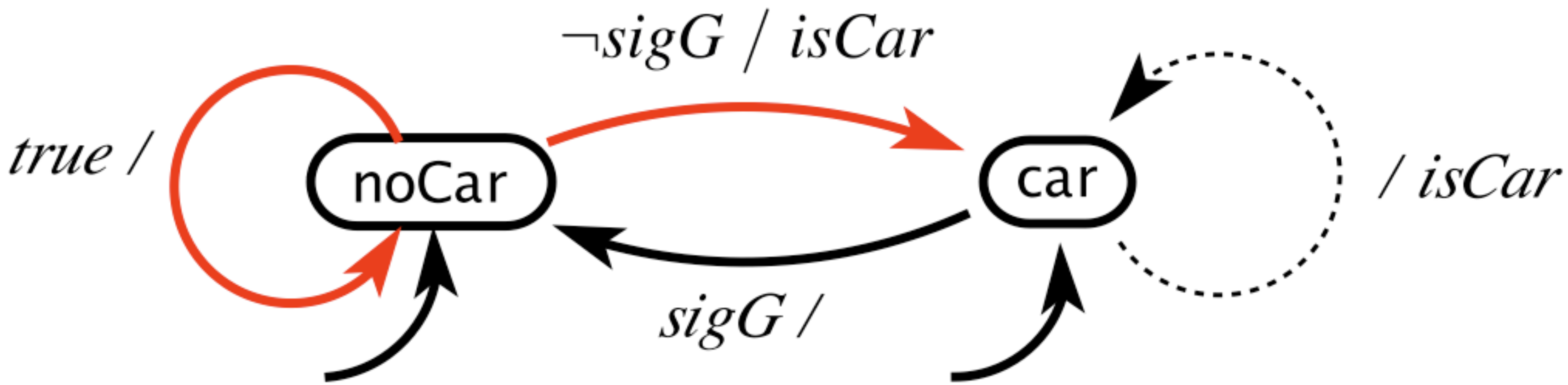
- **Receptiveness**: For any input values, some transition is enabled. Our structure together with the implicit default transition ensures that our FSMs are receptive.

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

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

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

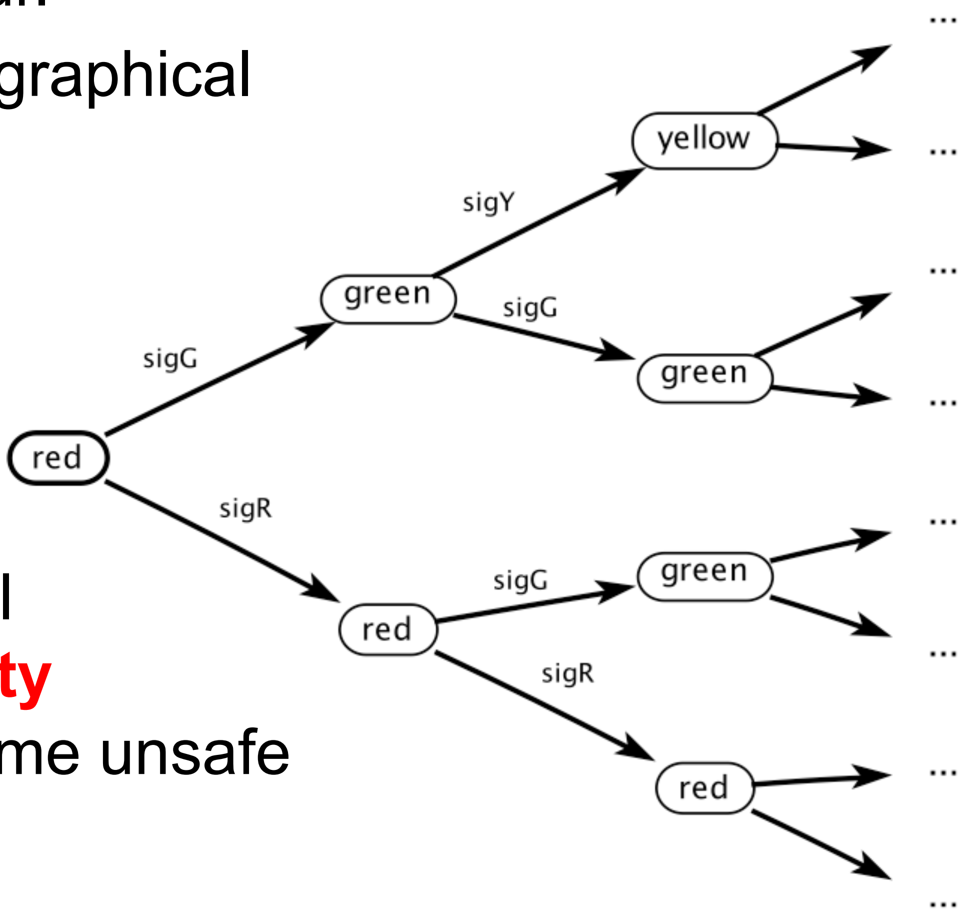
Uses of Nondeterminism

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

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

Behaviours and Traces

- FSM **behaviour** is a sequence of (non-stuttering) steps.
- A **trace** is the record of inputs, states, and outputs in a behaviour.
- A **computation tree** is a graphical representation of all possible traces.



FSMs are suitable for formal analysis. For example, **safety** analysis might show that some unsafe state is not reachable.

Size Matters

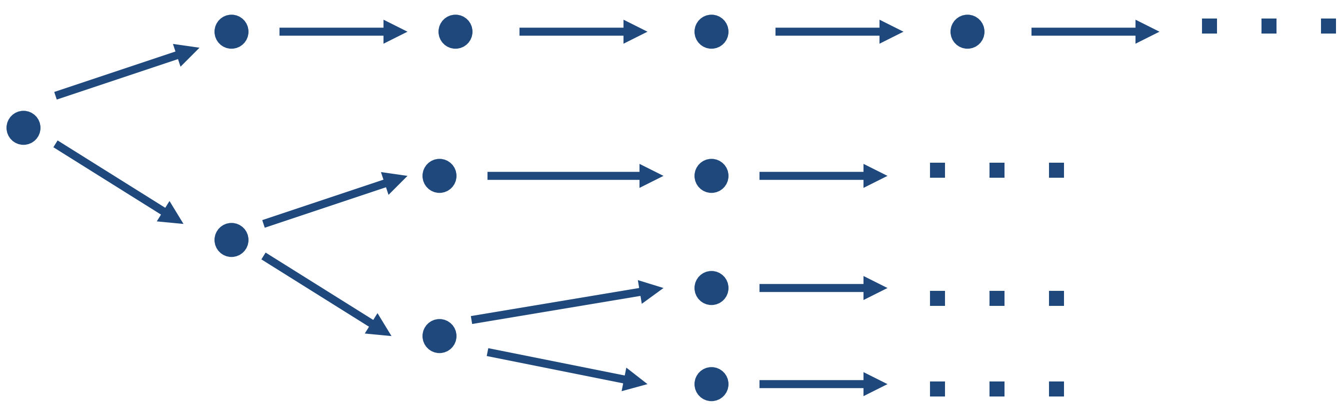
- Non-deterministic FSMs are more compact than deterministic FSMs
  - A classic result in automata theory shows that a nondeterministic FSM has a related deterministic FSM that is equivalent in a technical sense (language equivalence, covered in Chapter 13, for FSMs with finite-length executions).
  - 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 Behaviour: Tree of Computations

- For a fixed input sequence:
  - A deterministic system exhibits a single behaviour
  - A non-deterministic system exhibits a **set of behaviours**
    - visualised as a *computation tree*

Deterministic FSM behaviour:

Non-deterministic FSM behaviour:

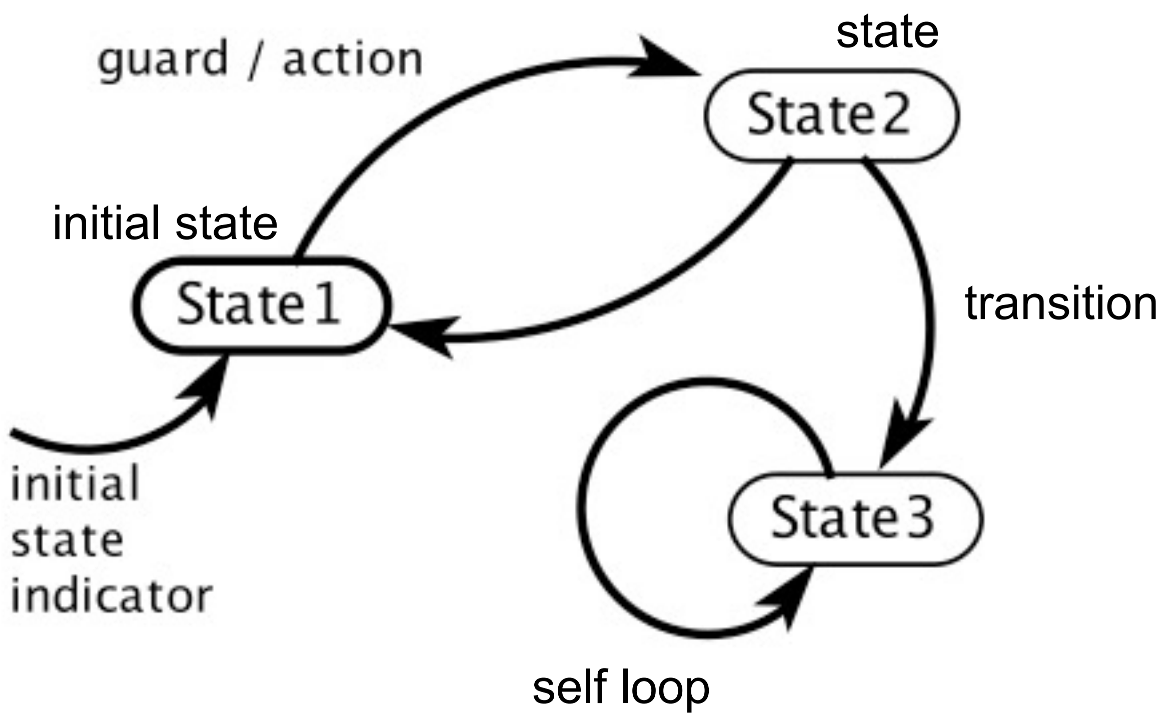


Non-deterministic $\neq$ Probabilistic (Stochastic)

- 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.

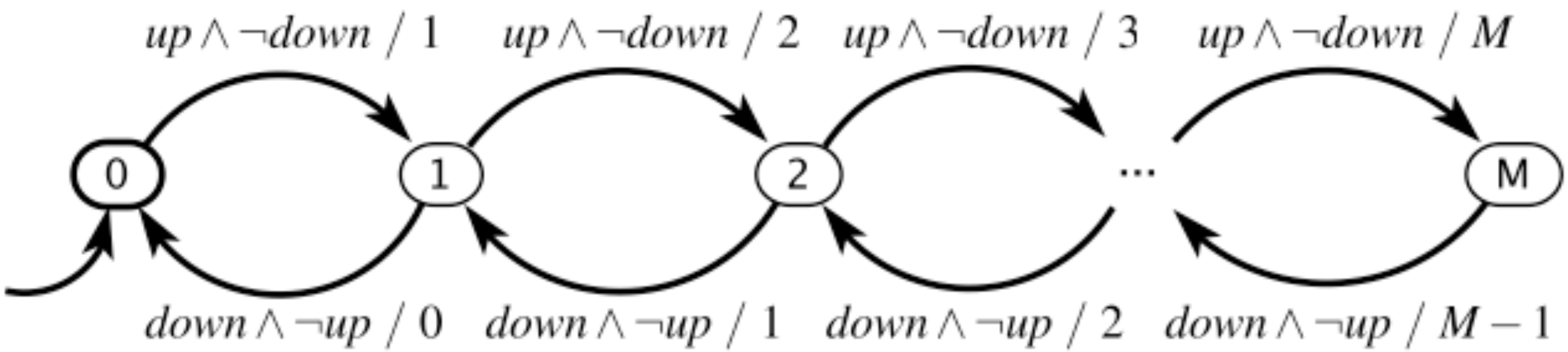
Recall FSM Notation



Garage Counter Example

- Recall this example, which counts cars in a parking garage:

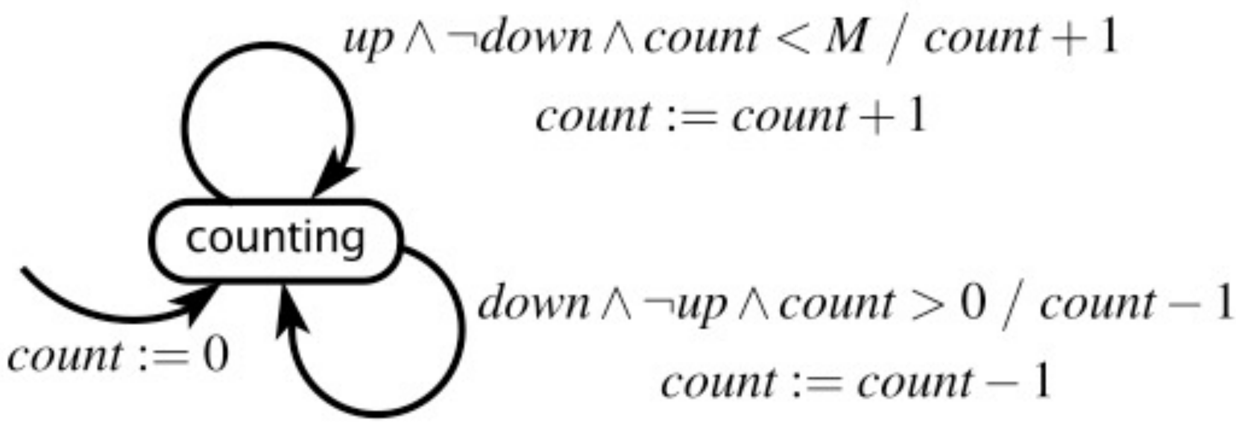
inputs: $up, down \in \{present, absent\}$
output $\in \{0, \cdots, M\}$



Extended State Machines

- Extended state machines augment the FSM model with *variables* that may be read or written. e.g.:

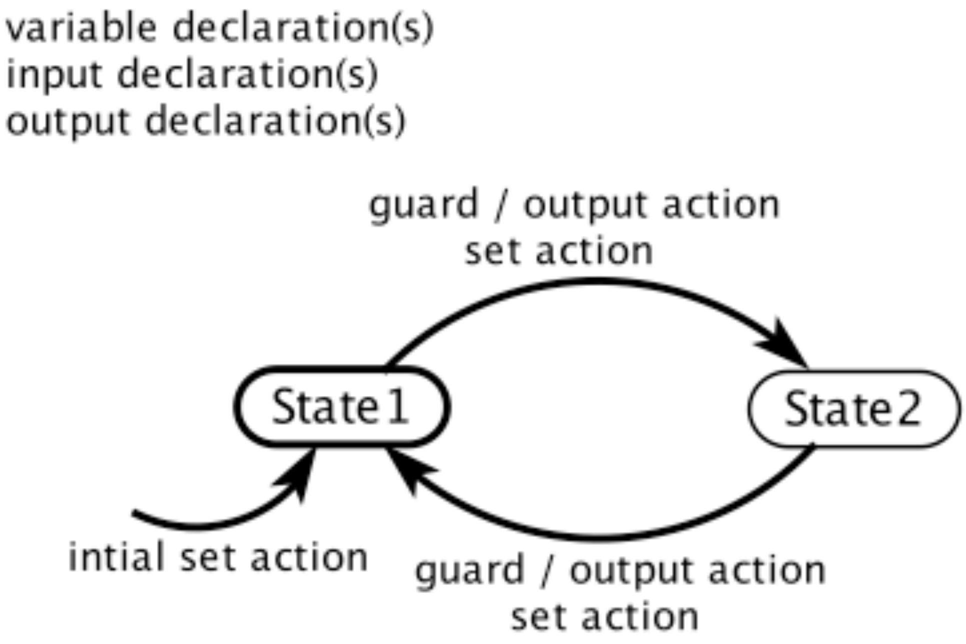
variable: $count \in \{0, \cdots, M\}$
inputs: $up, down \in \{present, absent\}$
output $\in \{0, \cdots, M\}$



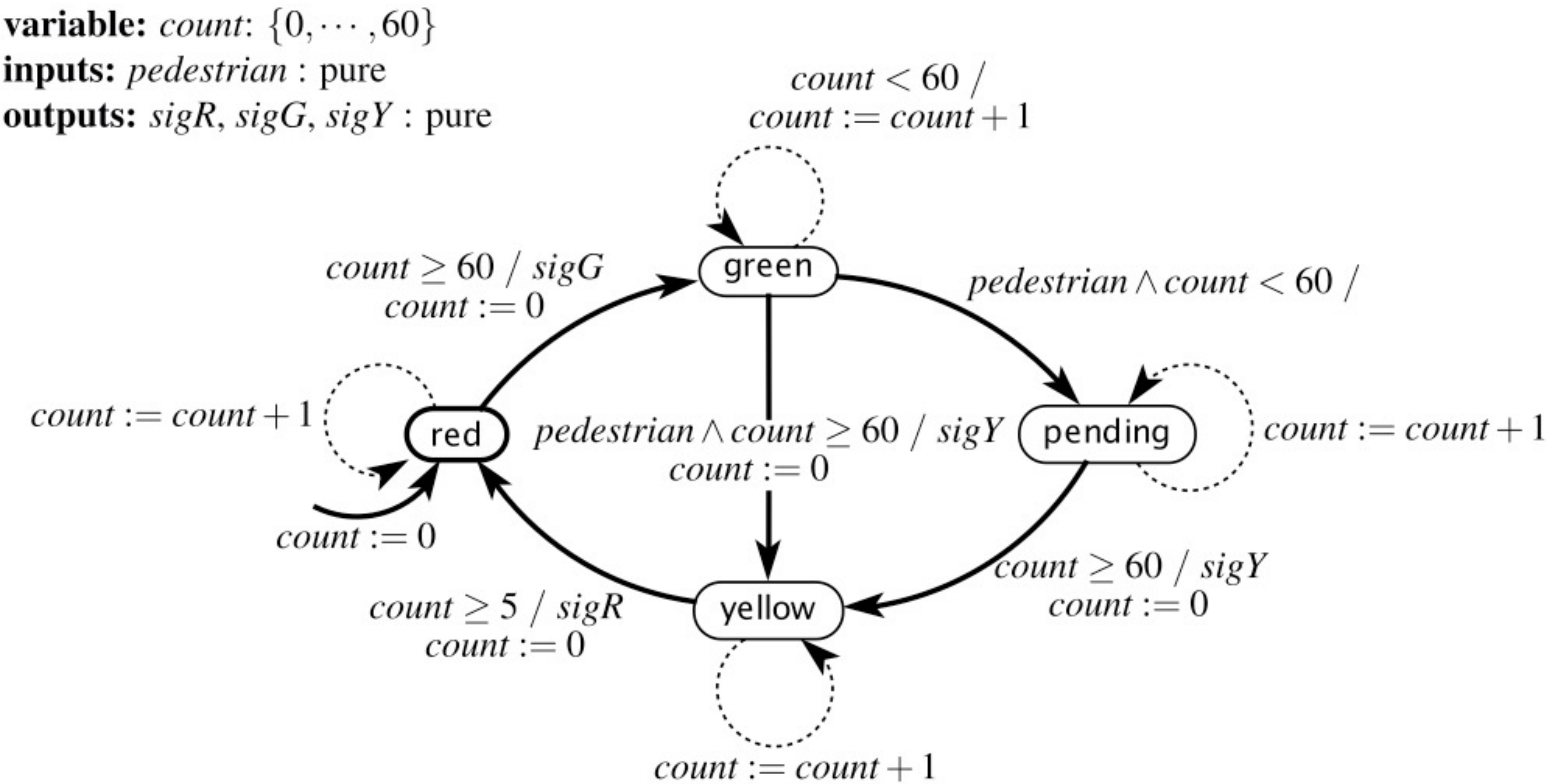
Question: What is the size of the state space?

General Notation for Extended State Machines

- We make explicit declarations of variables, inputs, and outputs to help distinguish the three.



Extended state machine model of a traffic light controller at a pedestrian crossing



- This model assumes one reaction per second (a *time-triggered* model)

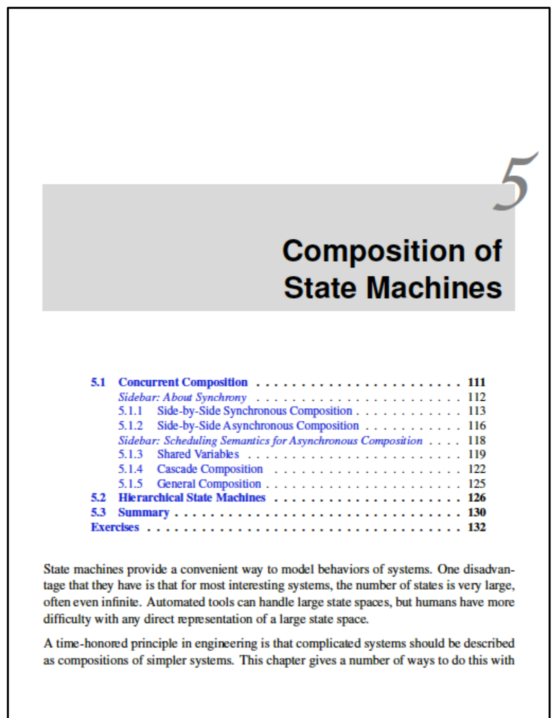
What we will be able to do with FSMs

FSMs provide:

1. A way to represent the system :
   - For mathematical analysis
   - So a computer program can manipulate it
2. A way to model the environment of a system
3. A way to represent what the system must do and must not do – its specification.
4. A way to check whether the system satisfies its specification in its operating environment.

Things to do …

- Download the textbook and read Chapter 5
- Read over Workshop 4. There is a per-workshop to do!



5

Composition of State Machines

State machines provide a convenient way to model behaviors of systems. One disadvantage that they have is that for most interesting systems, the number of states is very large, often even infinite. Automated tools can handle large state spaces, but humans have more difficulty with any direct representation of a large state space.

A time-honored principle in engineering is that complicated systems should be described as compositions of simpler systems. This chapter gives a number of ways to do this with