Department of Computer Science
National Tsing Hua University

CS 5244: Introduction to Cyber Physical Systems

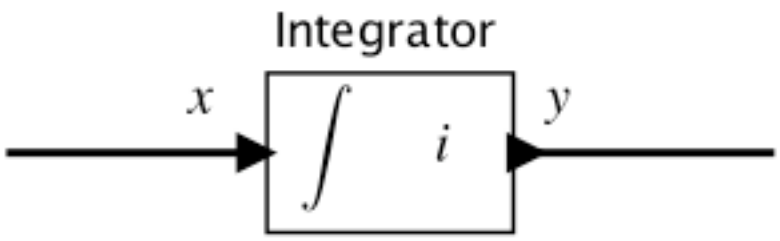
Unit 3: Modeling Modal Behavior (Ch. 3)

Instructor: Cheng-Hsin Hsu

Acknowledgement: The instructor thanks Profs. Edward A. Lee & Sanjit A. Seshia at UC Berkeley for sharing their course materials

Recall Actor Model of a Continuous-Time System

Example: integrator:

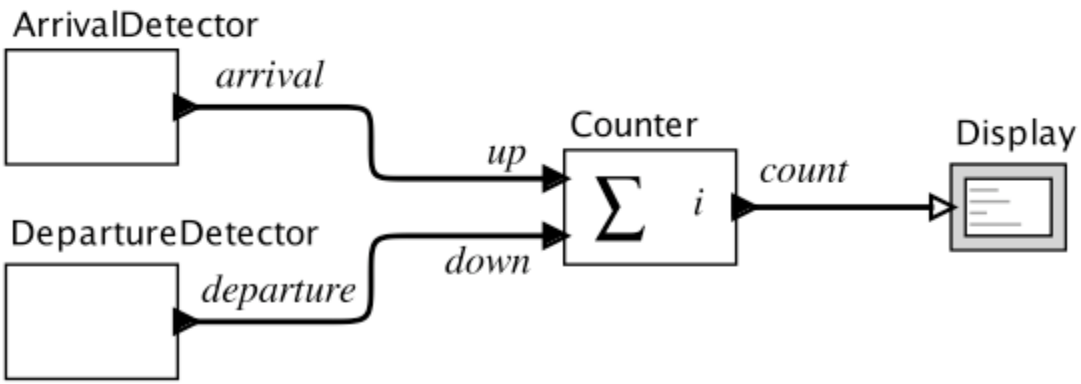


Continuous-time signal: $x\colon \mathbb{R} \to \mathbb{R}, \quad x \in (\mathbb{R} \to \mathbb{R}), \quad x \in \mathbb{R}^{\mathbb{R}}$

Continuous-time actor: $Integrator\colon \mathbb{R}^{\mathbb{R}} \to \mathbb{R}^{\mathbb{R}}$

Discrete Systems

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



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

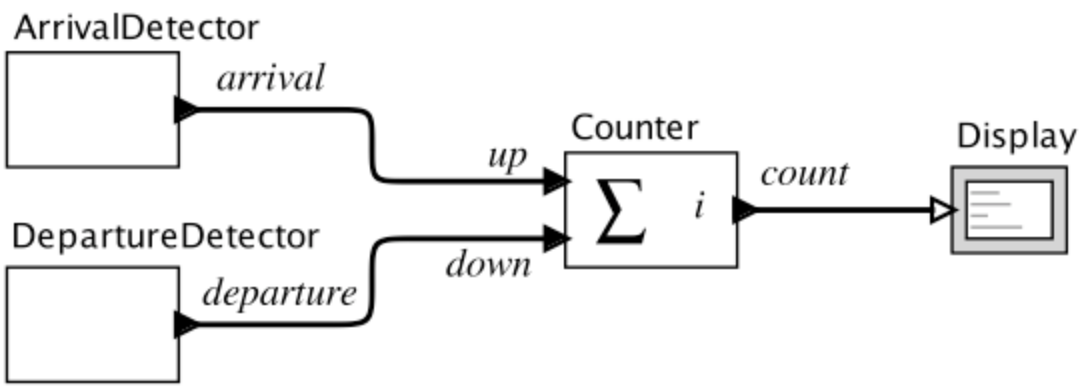
Discrete actor:

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

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

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**.

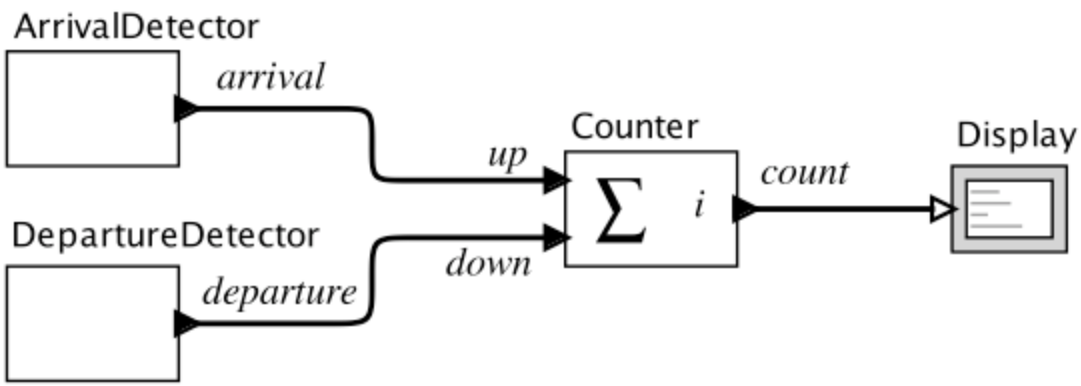


$$Counter\colon (\mathbb{R} \to \{absent, present\})^P \to (\mathbb{R} \to \{absent\} \cup \mathbb{N})$$
$$P = \{up, down\}$$

Input and Output Valuations at a Reaction

For $t \in \mathbb{R}$ a port p has a **valuation**, which is an assignment of a value in V_p (the **type** of port p). A valuation of the input ports $P = \{\textit{up}, \textit{down}\}$ assigns to each port a value in $\{absent, present\}$.

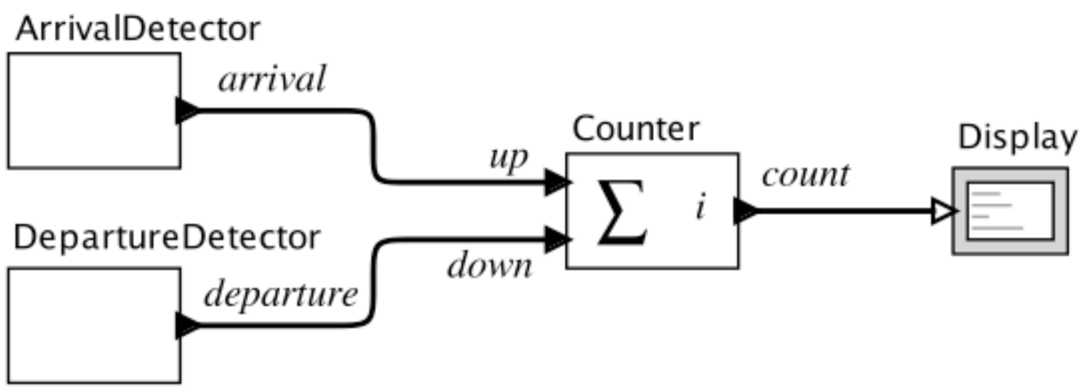
A **reaction** gives a valuation to the output port *count* in the set $\{absent\} \cup \mathbb{N}$.



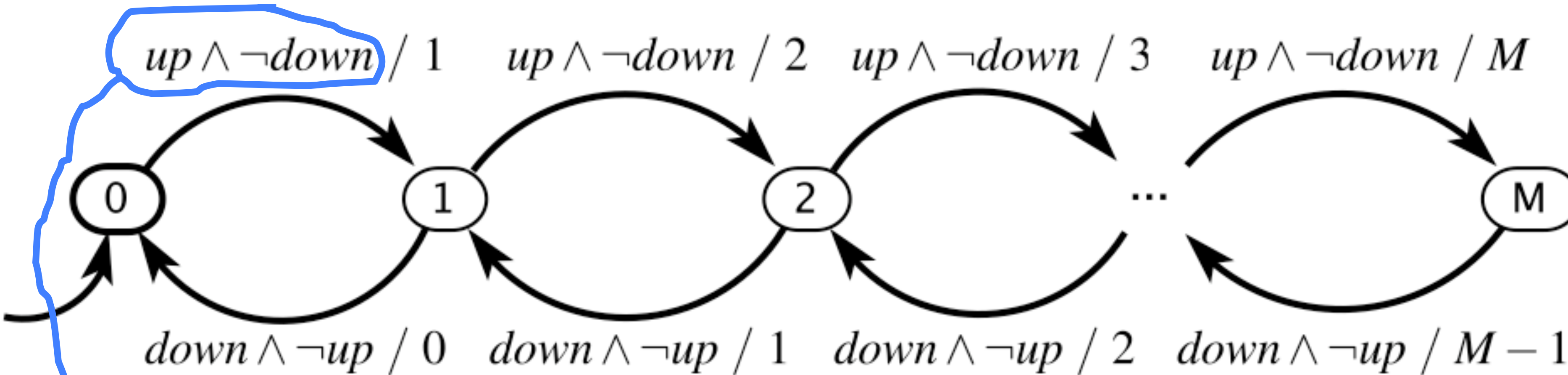
State Space

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

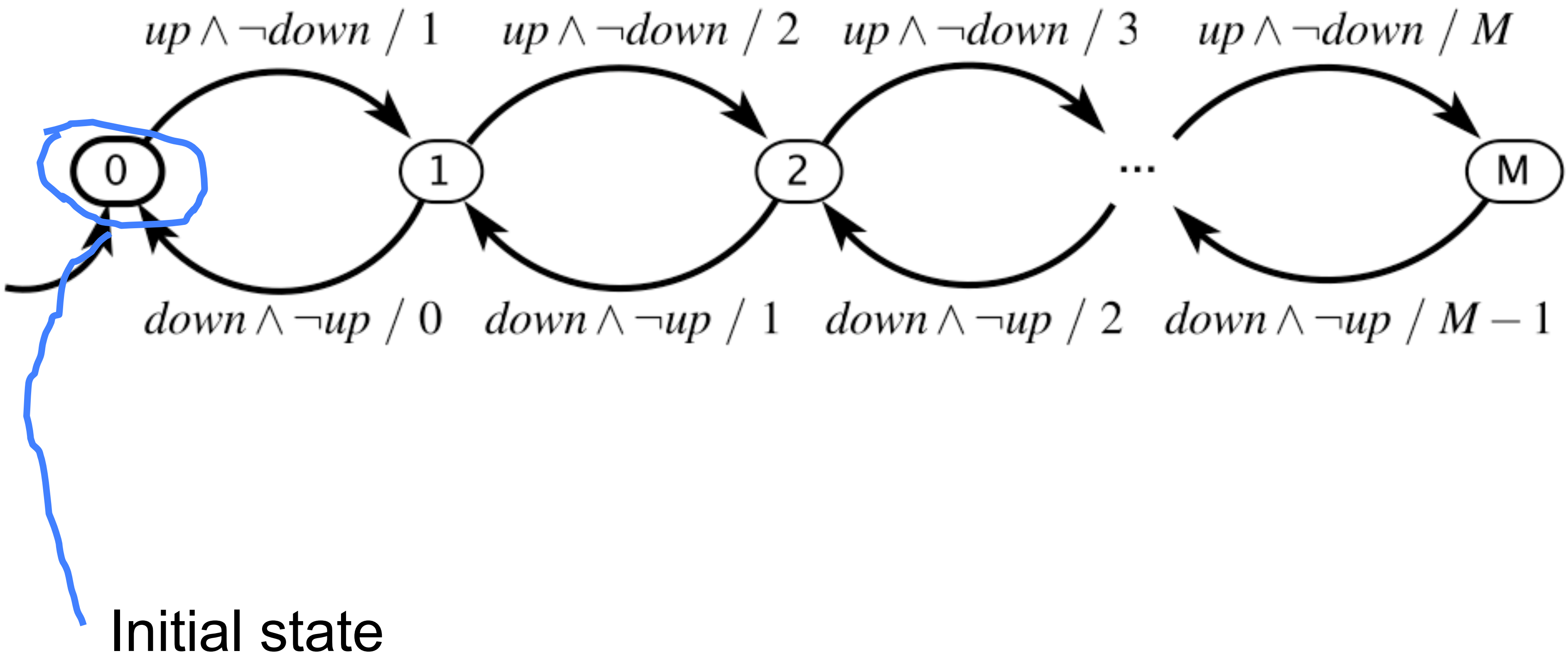


Guard g is specified using the predicate

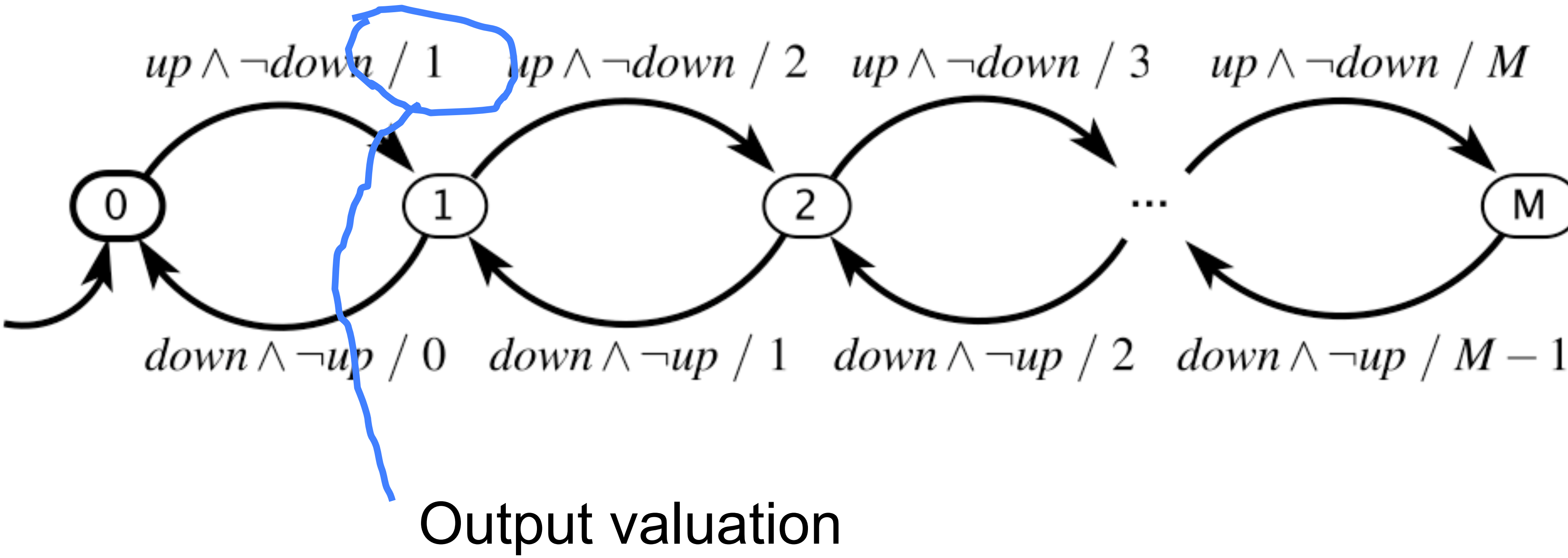
$$up \wedge \neg down$$

which means that *up* has value *present* and *down* has value *absent*.

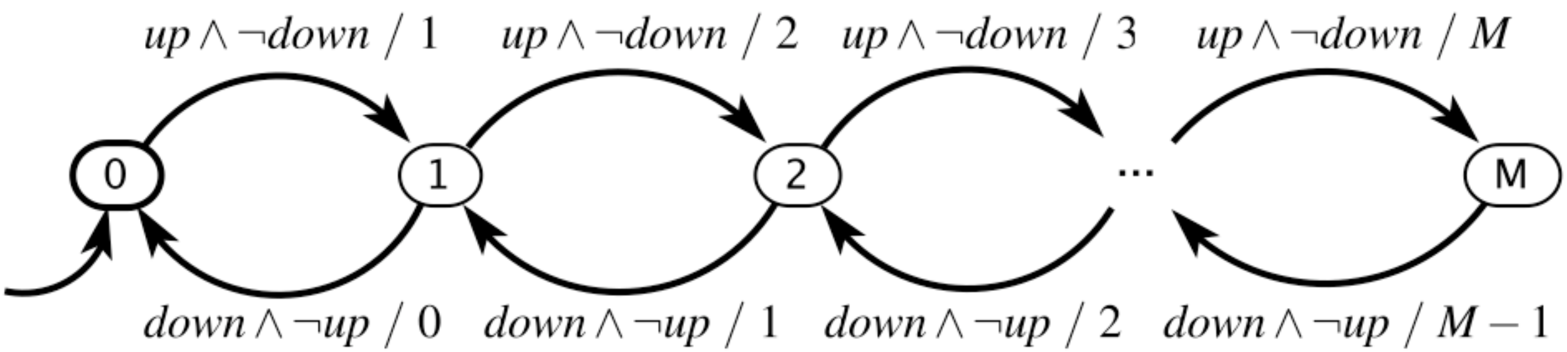
Garage Counter Finite State Machine (FSM) in Pictures



Garage Counter Finite State Machine (FSM) in Pictures



Garage Counter Mathematical Model

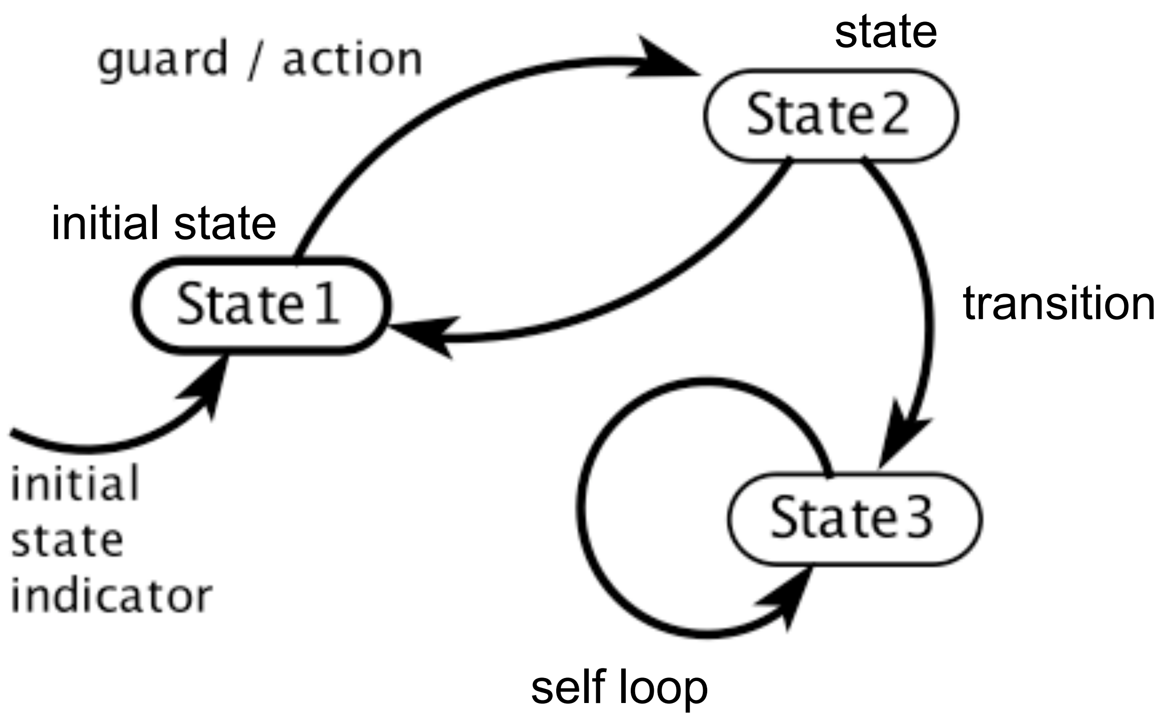


Formally: $(States, Inputs, Outputs, update, initialState)$, where

- $States = \{0, 1, \cdots, M\}$
- $Inputs$ is a set of input valuations
- $Outputs$ is a set of output valuations
- $update : States \times Inputs \to States \times Outputs$
- $initialState = 0$

The picture above defines the update function.

FSM Notation



Examples of Guards for Pure Signals

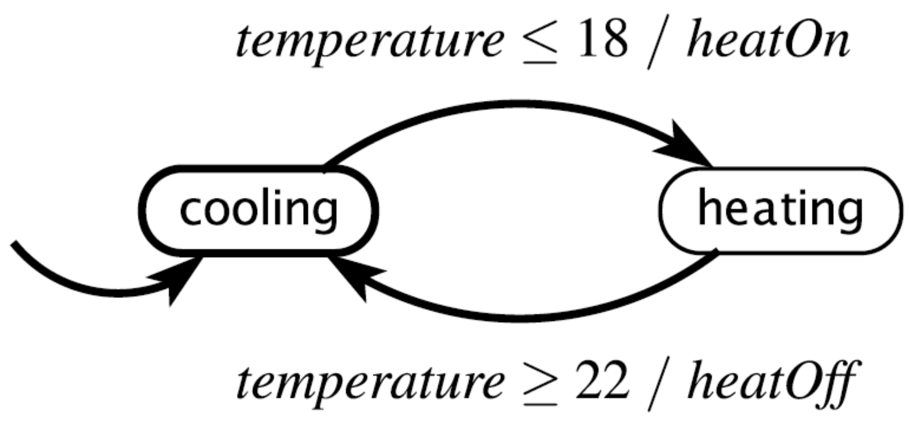
true	Transition is always enabled.
p_1	Transition is enabled if p_1 is $present$.
$\neg p_1$	Transition is enabled if p_1 is $absent$.
$p_1 \wedge p_2$	Transition is enabled if both p_1 and p_2 are $present$.
$p_1 \vee p_2$	Transition is enabled if either p_1 or p_2 is $present$.
$p_1 \wedge \neg p_2$	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.
$p_3 = 1 \wedge p_1$	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.

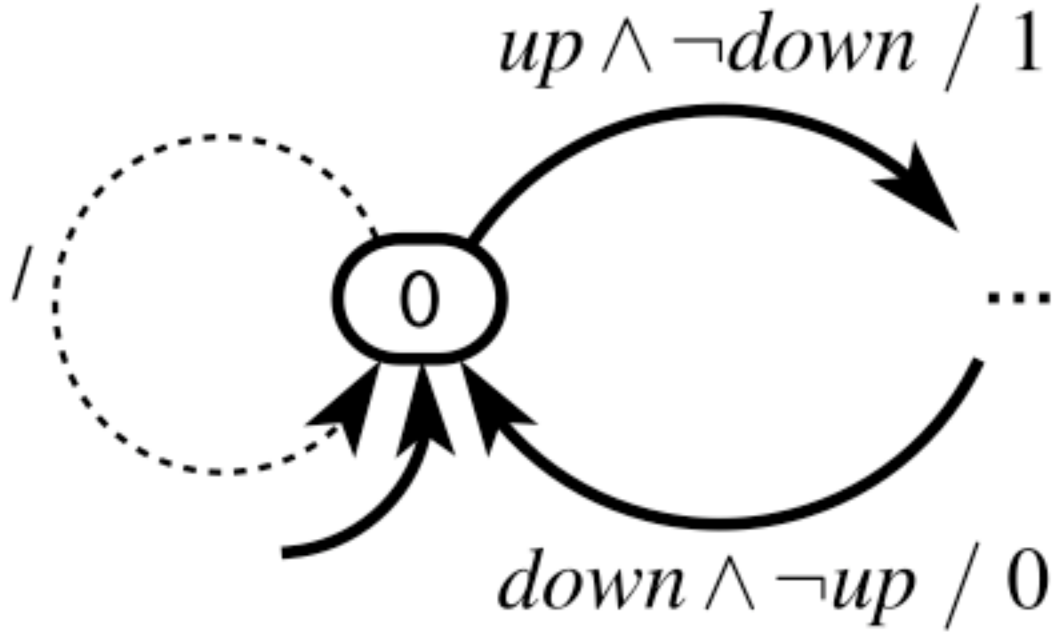
Example: Thermostat

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



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

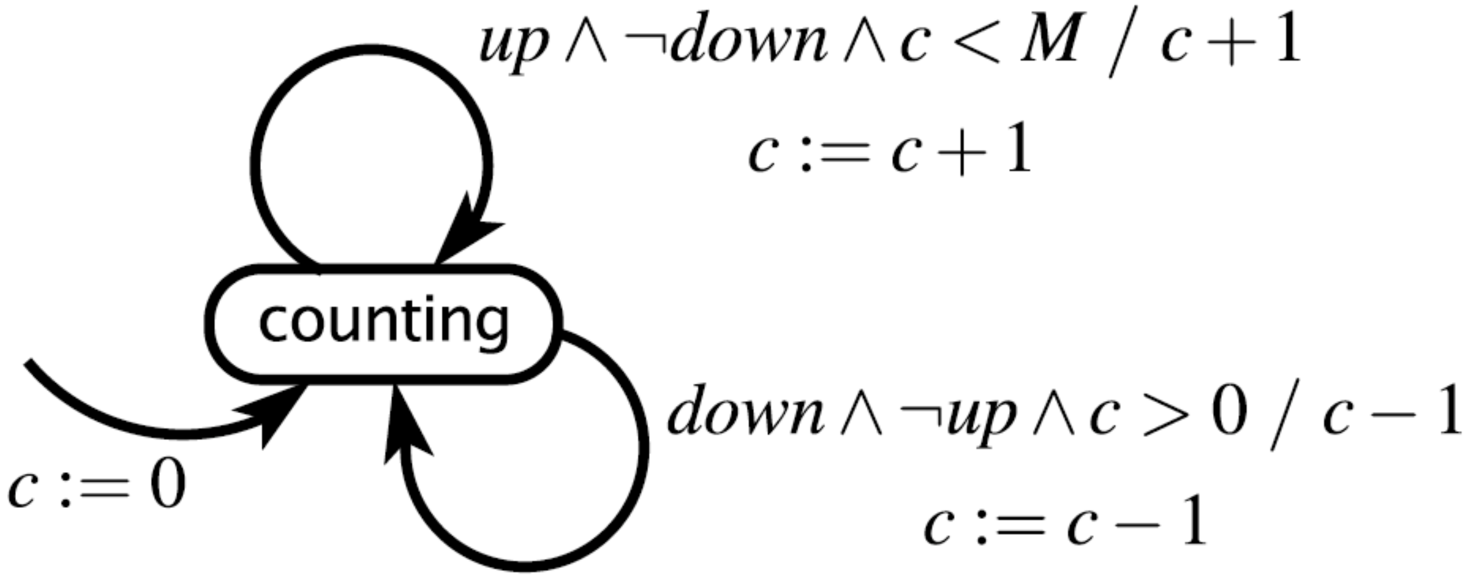
More Notation: Default Transitions



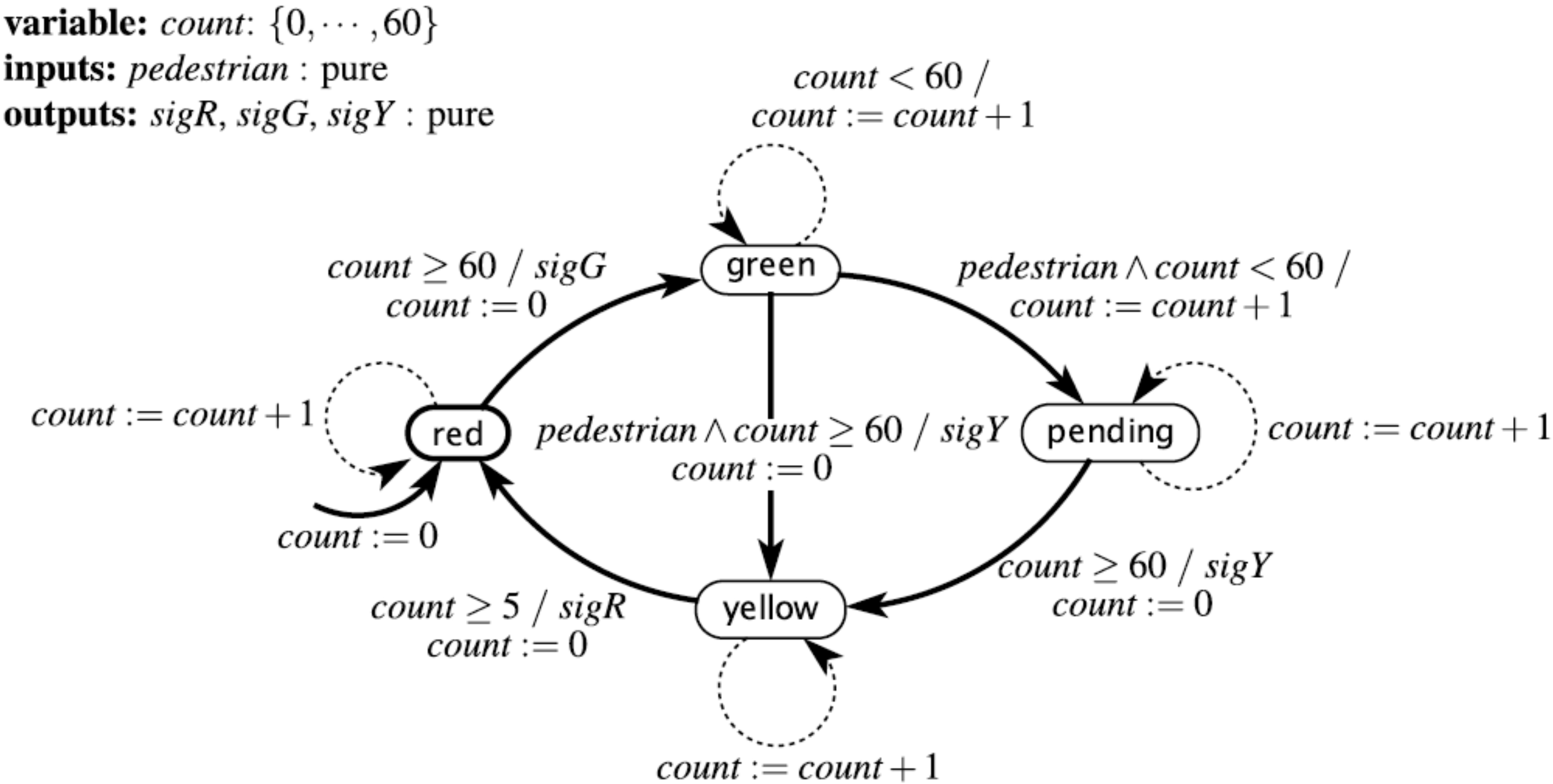
A default transition is enabled if no non-default transition is enabled and it either has no guard or the guard evaluates to true. When is the above default transition enabled?

Extended State Machines

variable: $c: \{0, \cdots, M\}$
inputs: *up*, *down*: pure
output: *count*: $\{0, \cdots, M\}$



Traffic Light Controller

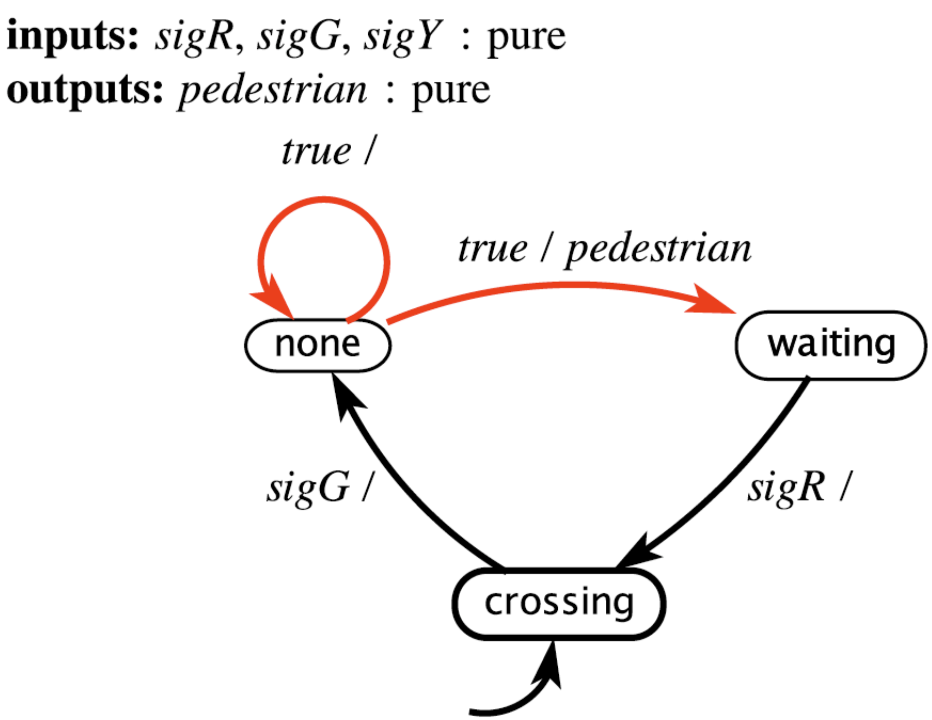


Definitions

- **Stuttering transition**: Implicit default transition that is enabled when inputs are absent 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: Nondeterminate FSM

Nondeterminate model of pedestrians arriving at a crosswalk:

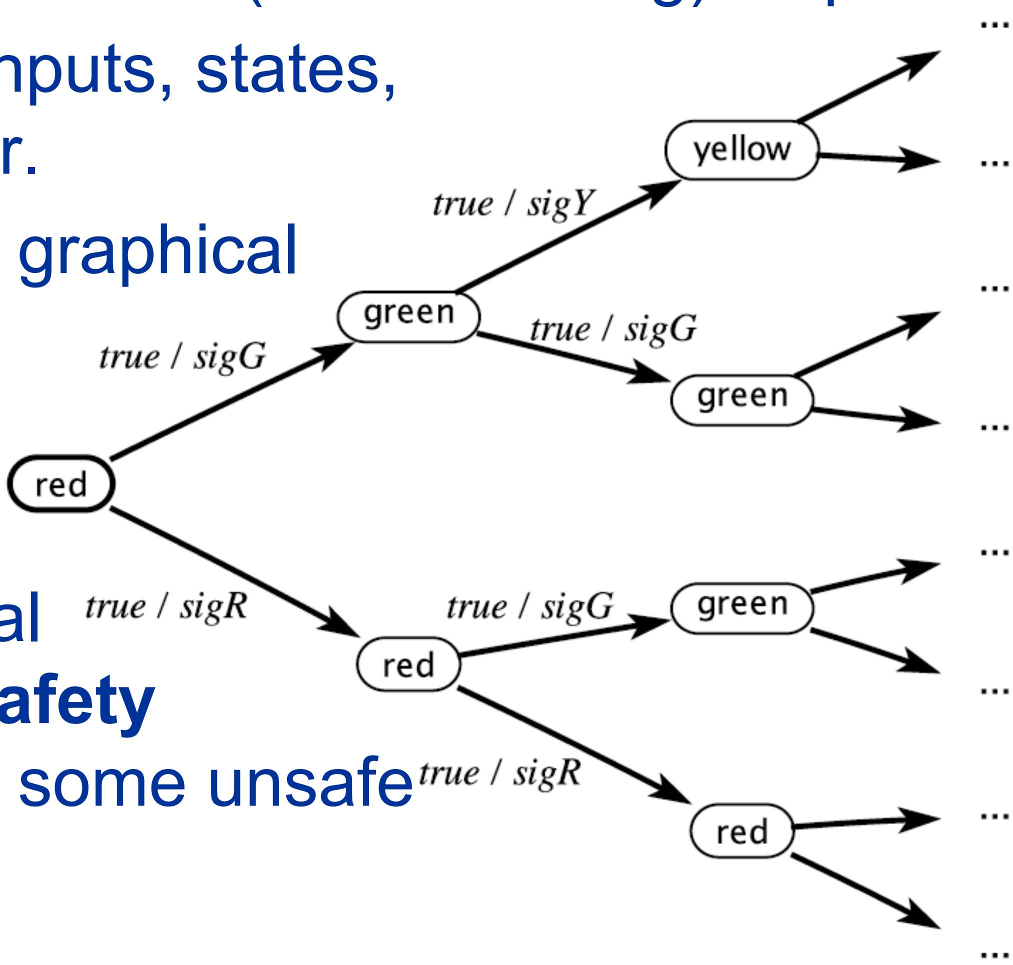


Formally, the update function is replaced by a function

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

Behaviors and Traces

- FSM **behavior** is a sequence of (non-stuttering) steps.
- A **trace** is the record of inputs, states, and outputs in a behavior.
- 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.

Uses of nondeterminism

1. Modeling unknown aspects of the environment or system
   - Such as: how the environment changes the iRobot's orientation
2. Hiding detail in a *specification* of the system
   - We will see an example of this later (see notes)

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

Size Matters

Non-deterministic FSMs are more compact than deterministic FSMs

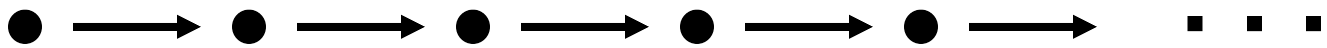
- ND FSM → D FSM: Exponential blow-up in #states in worst case

Non-deterministic Behavior: Tree of Computations

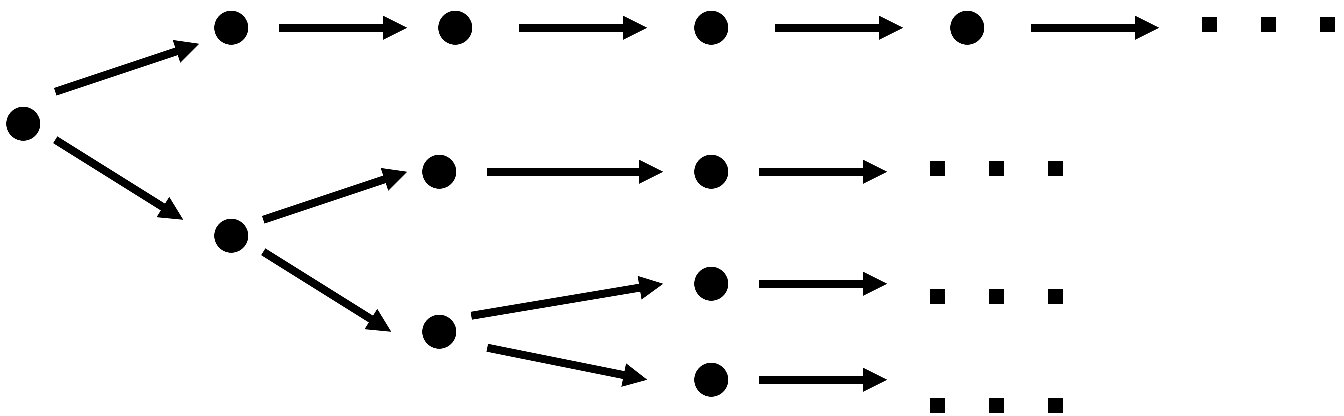
For a fixed input sequence:

- A deterministic system exhibits a single behavior
- A non-deterministic system exhibits a **set of behaviors**

Deterministic FSM behavior for a particular input sequence:



Non-deterministic FSM behavior for an input sequence:

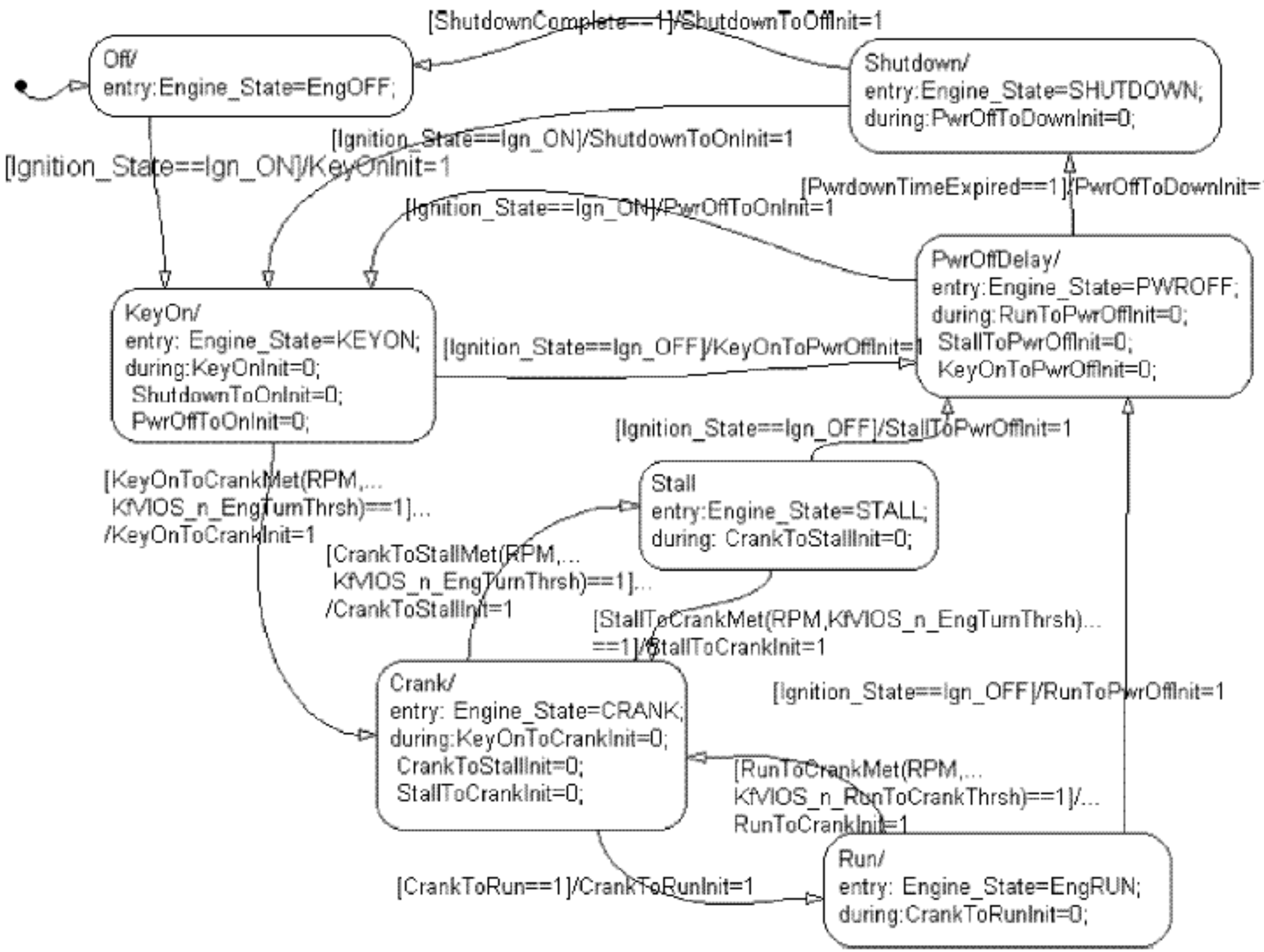


Related points

What does receptiveness mean for non-deterministic state machines?

Non-deterministic $\neq$ Probabilistic

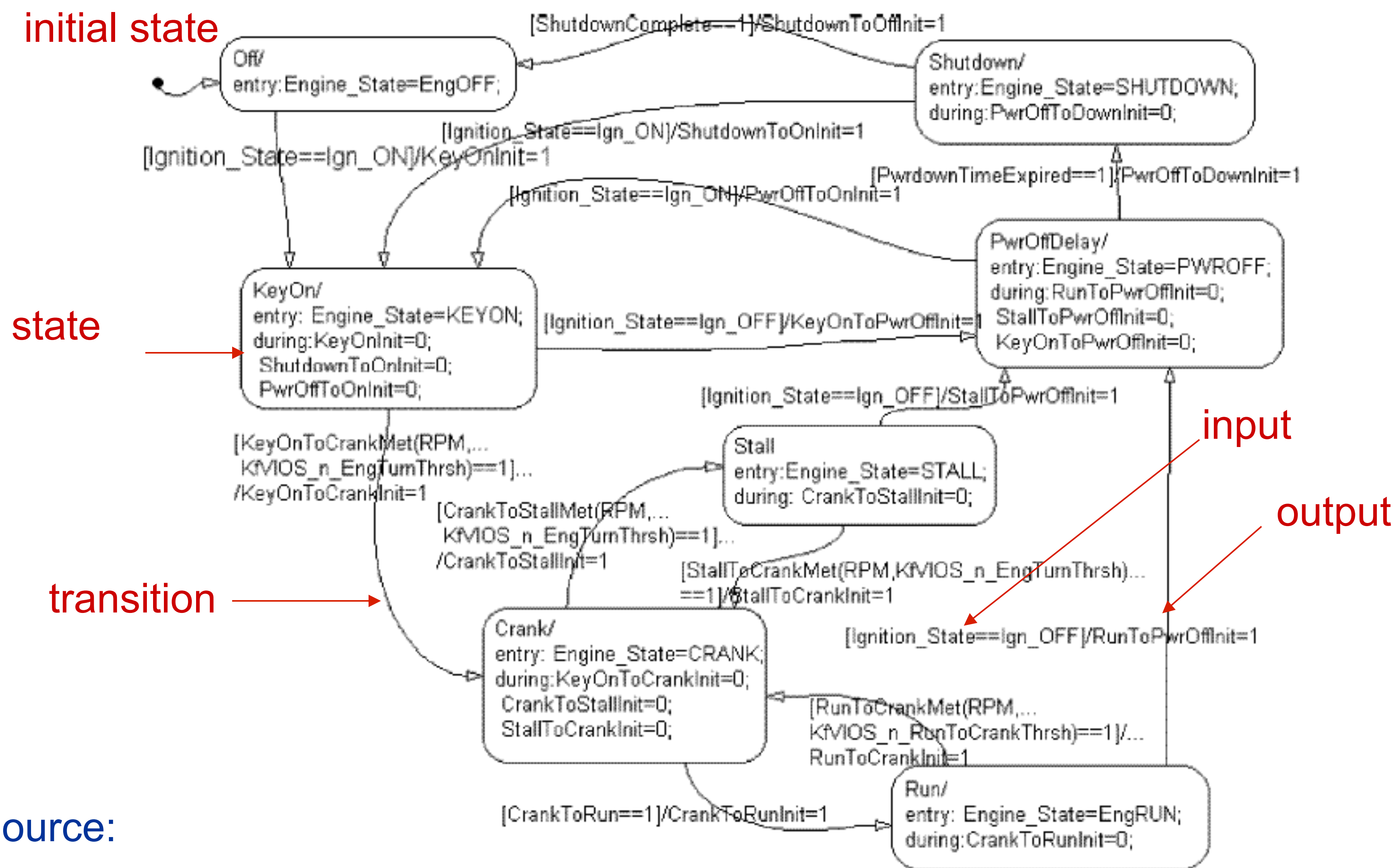
Example from Industry: Engine Control



Source:

Delphi Automotive Systems (2001)

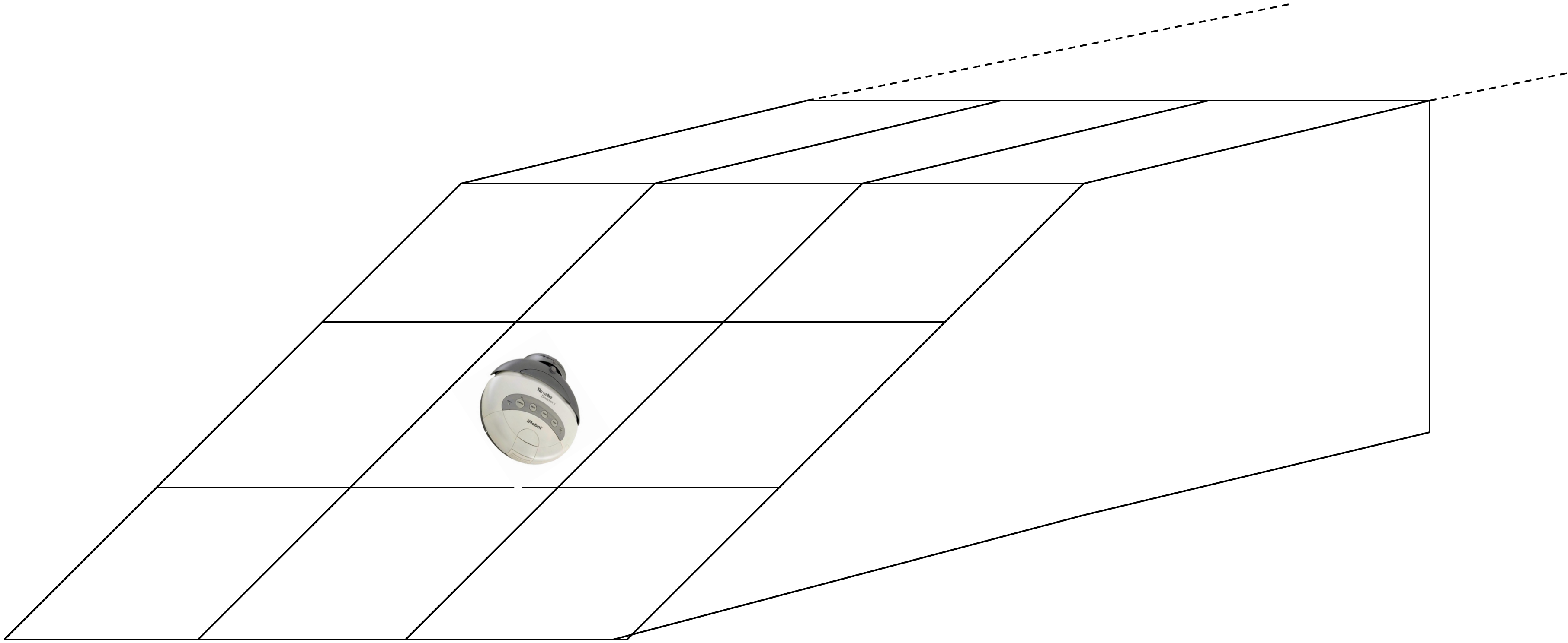
Elements of a Modal Model (FSM)



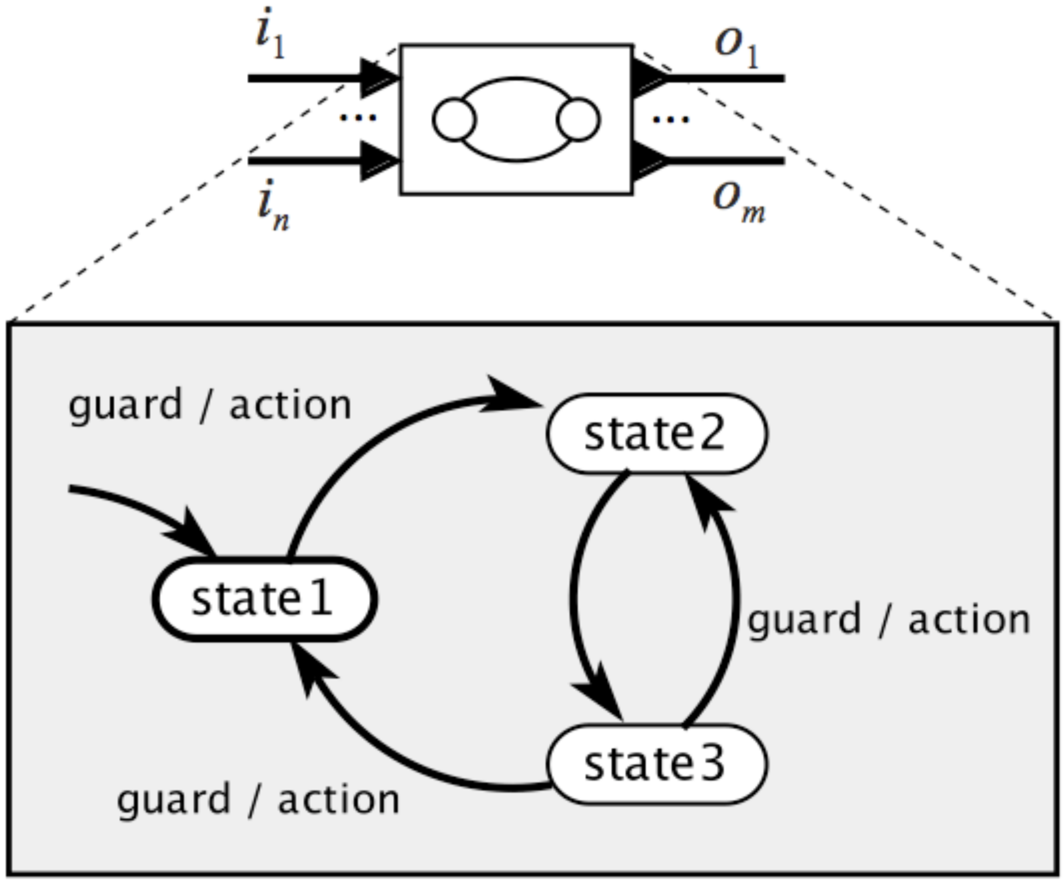
Source:

Delphi Automotive Systems (2001)

It is sometimes useful to even model continuous systems as FSMs by discretizing their state space. E.g.: Discretized iRobot Hill Climber



Actor Model of an FSM



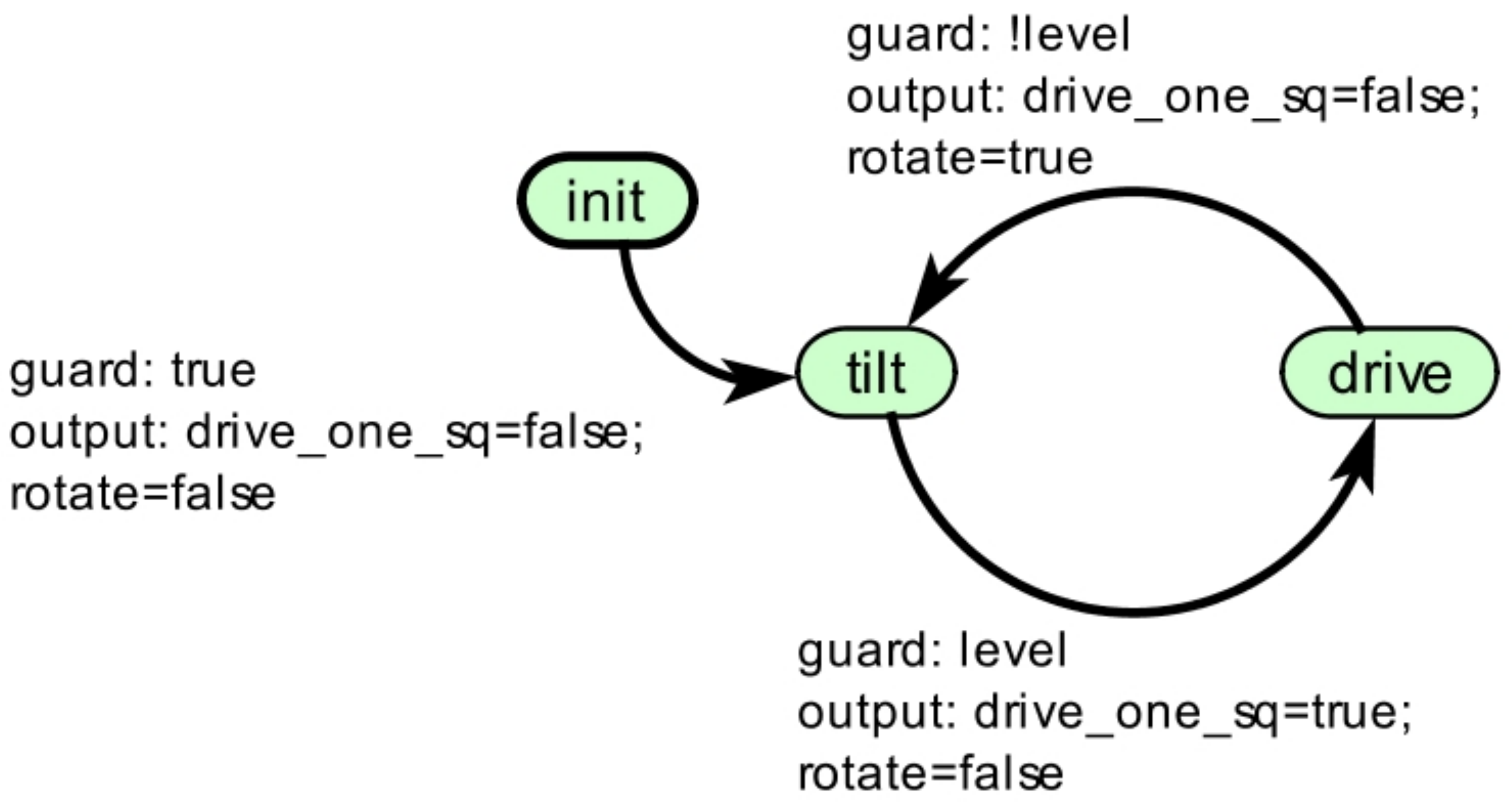
This model enables ***composition*** *of state machines.*

What we will be able to do with FSMs

FSMs provide:

1. A way to represent the system for:
   - Mathematical analysis
   - So that 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.

FSM Controller for iRobot

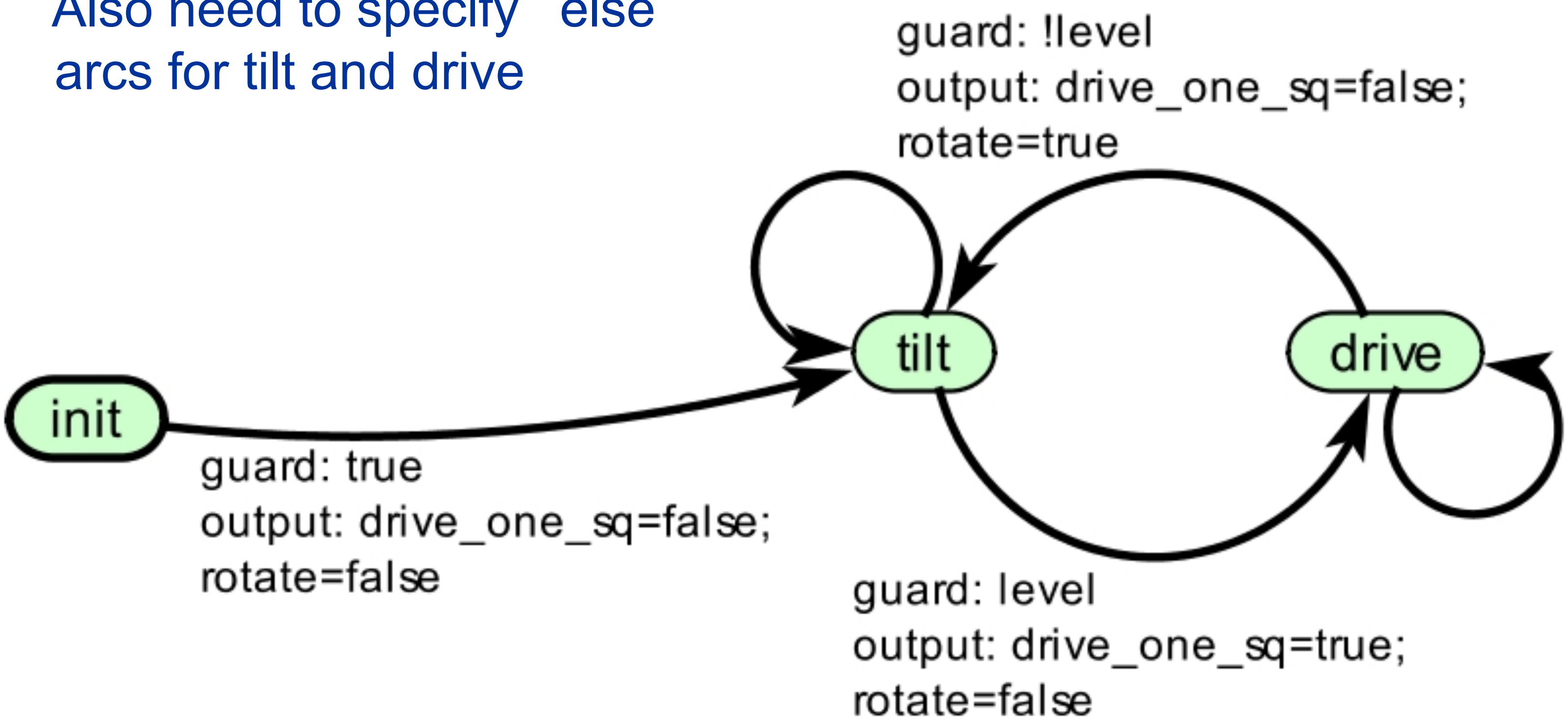


States = {init, tilt, drive} Inputs = ? Outputs = ?

update = ? Any transitions missing?

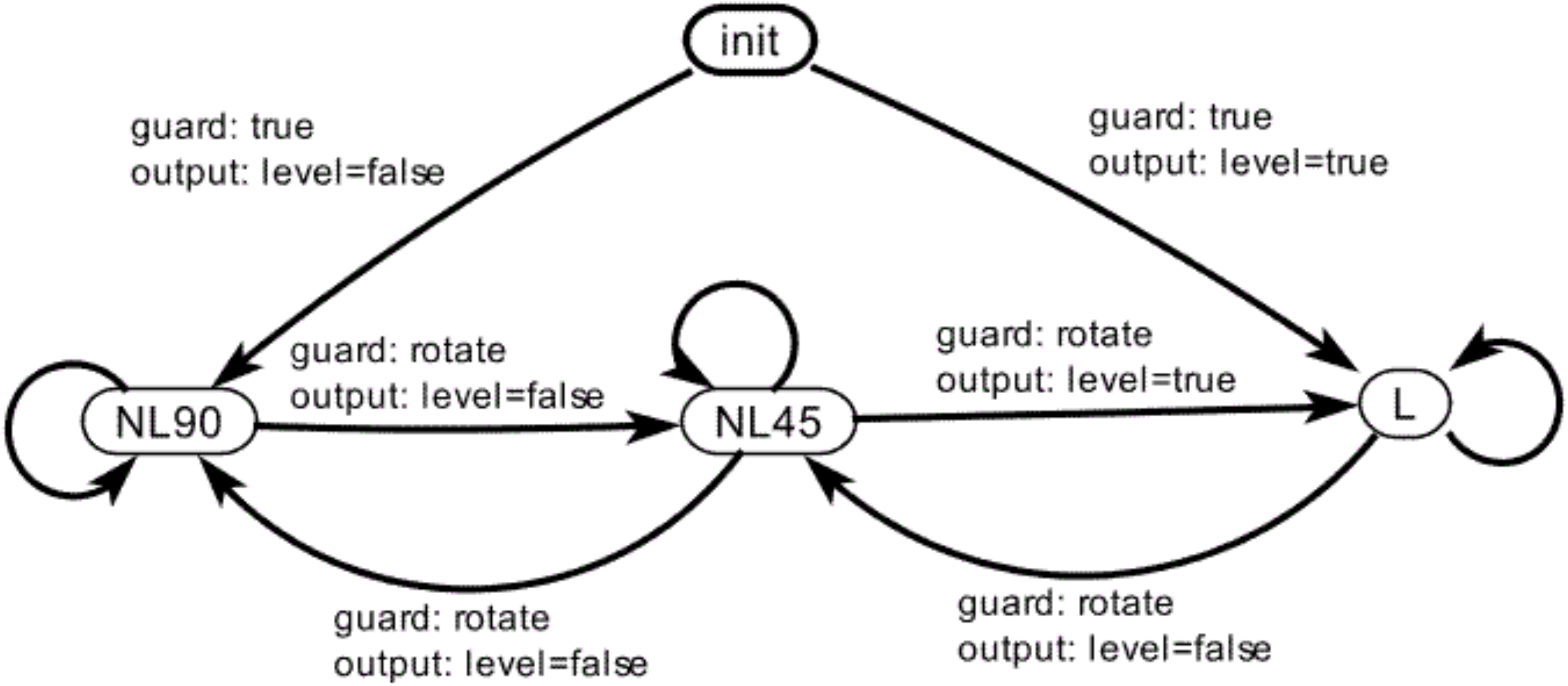
FSM Controller for iRobot (version 2)

Also need to specify "else" arcs for tilt and drive



Will this robot always drive uphill?
(assume that it starts facing uphill)

Modeling the iRobot's environment



Is this model **deterministic**?

L level=true

NL45 level=false, 45° offset

NL90 level=false, 90° offset

Self loops on: rotate=false

Representing a state machine

1. Pictorial notation

2. Table representing transition relation

3. Functional notation

When would you use each representation?