Embedded Systems Design and Modeling

Chapter 3
Modeling Discrete Dynamics

Outline

- Introduction to discrete systems
- Discrete systems modeling by examples:
  - Parking lot counter
  - Temperature controller
  - Traffic light controller
- Finite state machines:
  - Different representations
  - Formal definition
- System properties

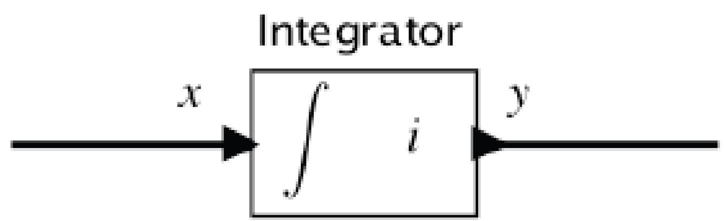
What Is A Discrete System?

- Discrete system: operates in a sequence of discrete steps rather than continuous time
- Steps might be based on external events or passage of time
- The system may truly operate discretely (inherently discrete)
- Or it may be inherently continuous but modeled in a discrete way

Actor Model Review

Recall Actor Model of a Continuous-Time System

Example: integrator:



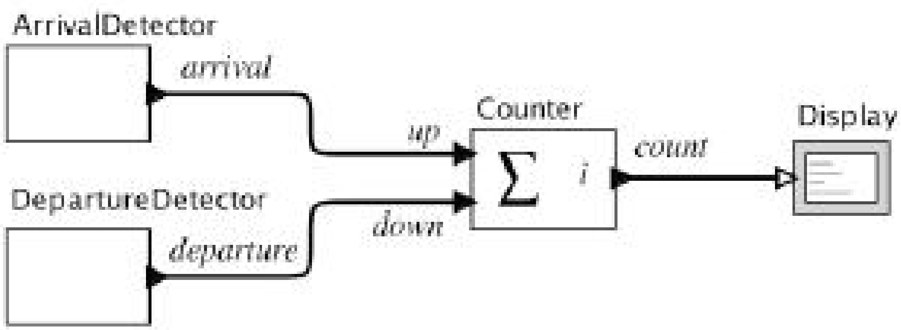
Continuous-time signal: $x : \mathbb{R} \to \mathbb{R}, \quad x \in (\mathbb{R} \to \mathbb{R}), \quad x \in \mathbb{R}^{\mathbb{R}}$

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

Example: Parking Counter

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:

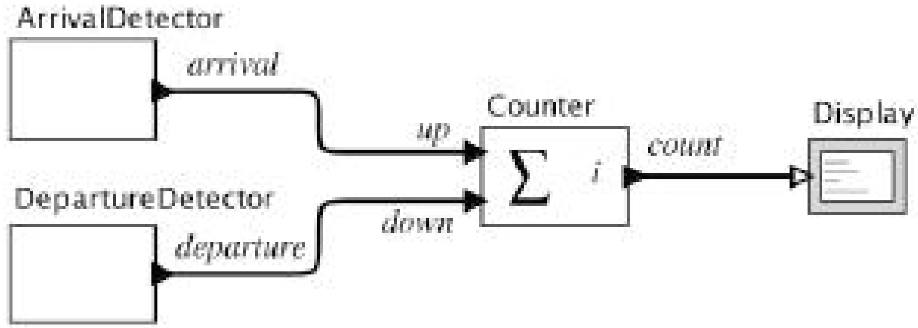
$$Counter: (\mathbb{R} \to \{absent, present\})^P \to (\mathbb{R} \to \{absent\} \cup \mathbb{N})$$

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

Basic Definitions

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.**



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

Basic Definitions …

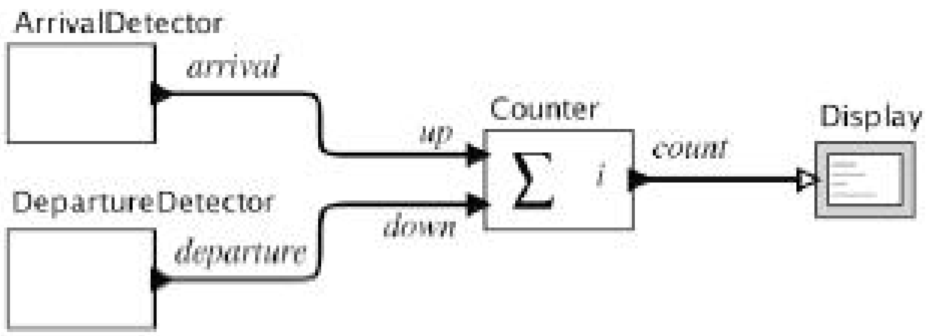
Inputs and Outputs at a Reaction

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

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

and the outputs are in a set

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

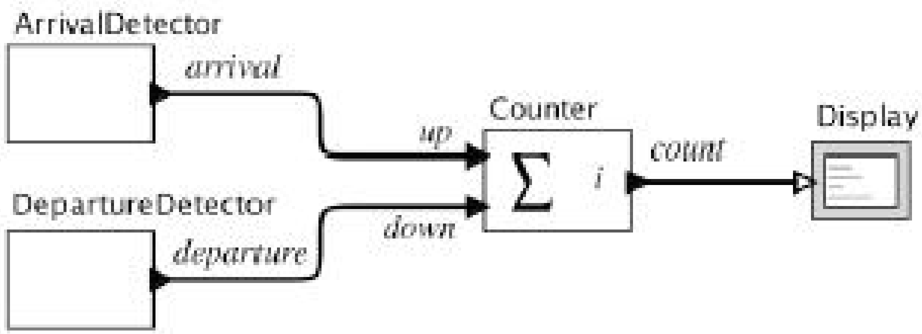


Basic Definitions …

State Space

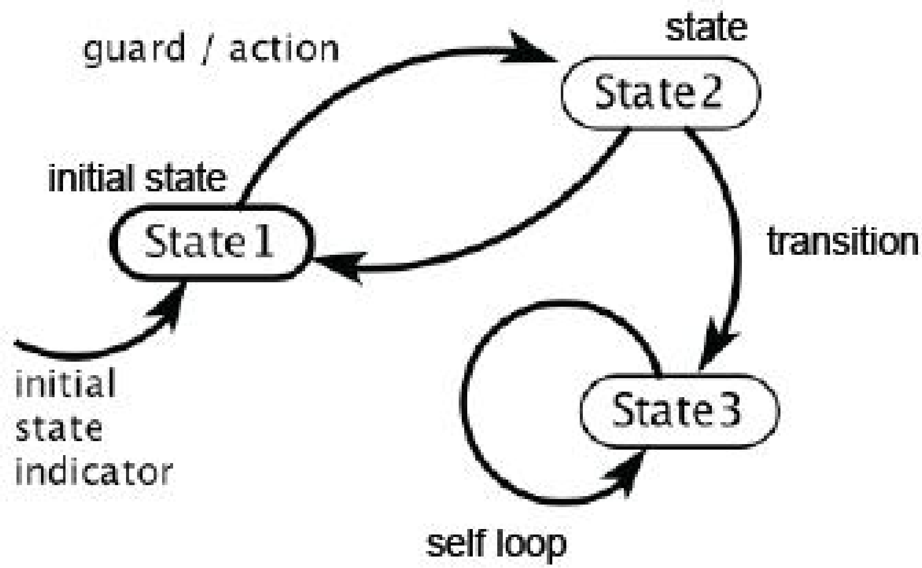
A practical parking garage has a finite number M of spaces, so the state space for the counter is

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



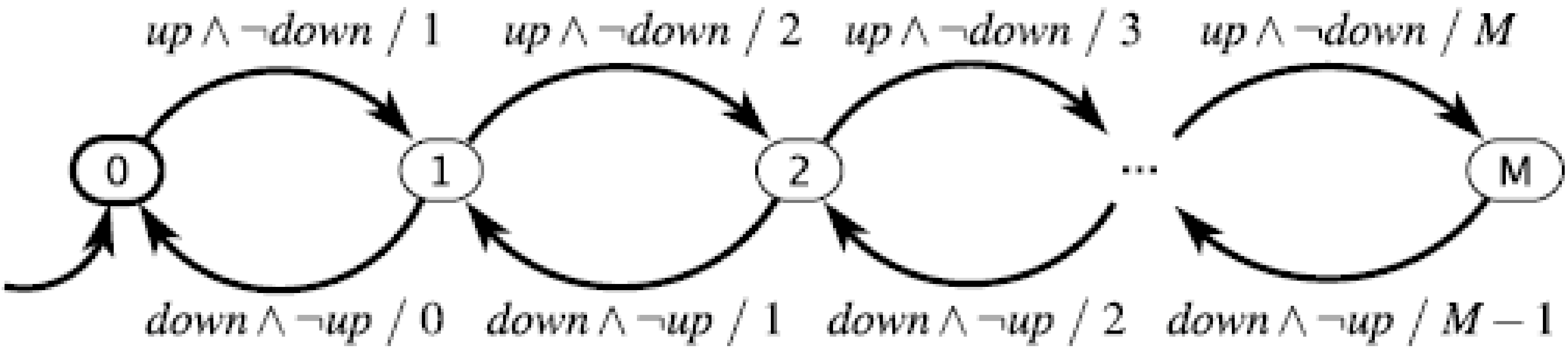
Finite State Machine Recall

FSM Notation



Example Modeled Using FSM

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



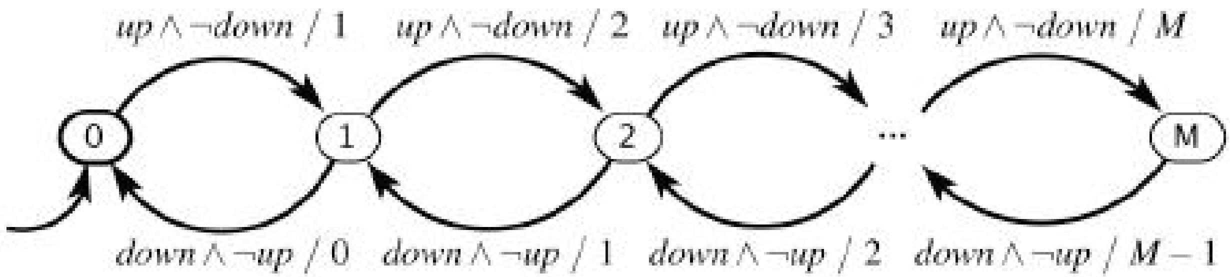
Notations Used

- Initial state: where the FSM starts at
- Actions: specifies what outputs are produced
- Guards: transition conditions expressed as Boolean expressions like these …

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*.

FSM Formal Representation

Garage Counter Mathematical Model



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

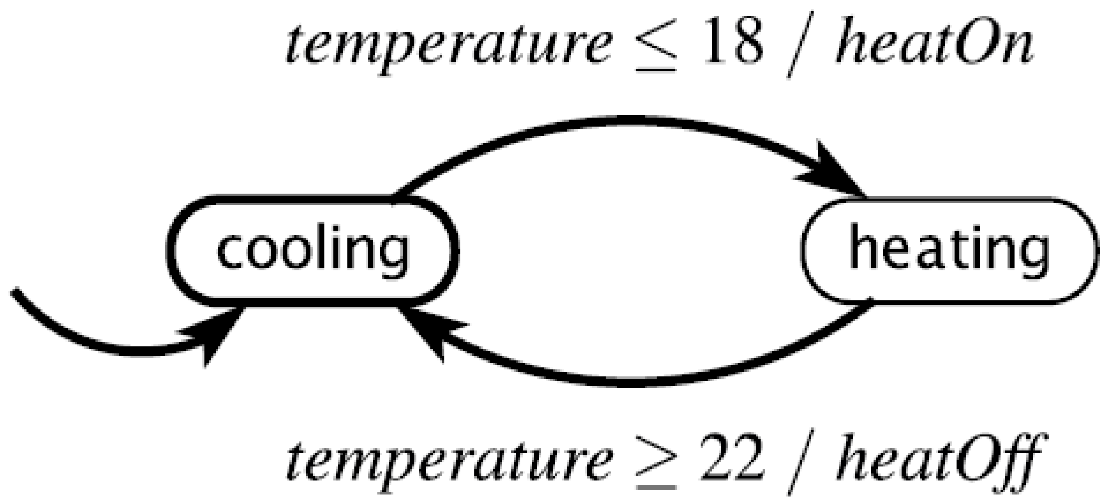
- $States = \{0, 1, \cdots, M\}$
- $Inputs = (\{\text{up}, \text{down}\} \to \{absent, present\}$
- $Outputs = (\{count\} \to \{absent\} \cup \mathbb{N})$
- $update : States \times Inputs \to States \times Outputs$
- $initialState = 0$

The picture above defines the update function.

Another Example Using FSM

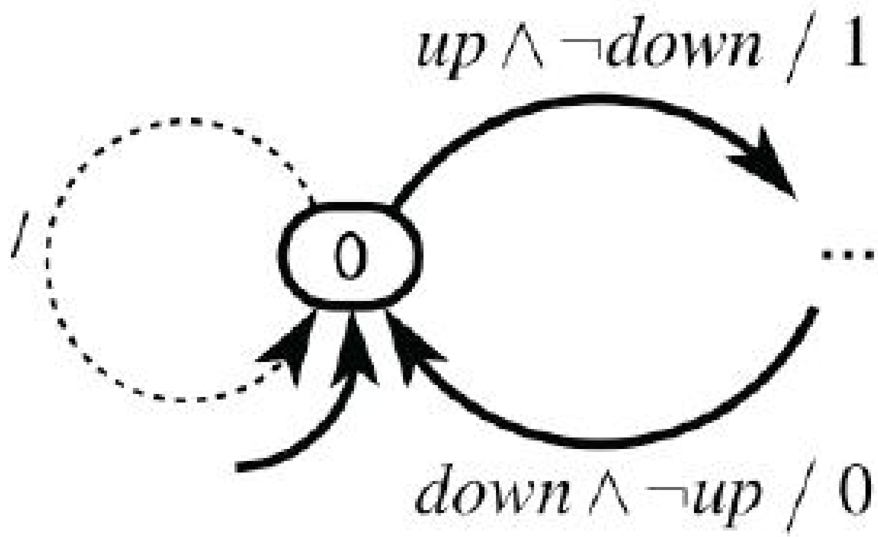
- A heating/cooling system with two states
- Two target temperatures: 18 and 22 degrees
- Thermostat has hysteresis (avoids chattering)

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



Default Transition

More Notation: Default Transitions

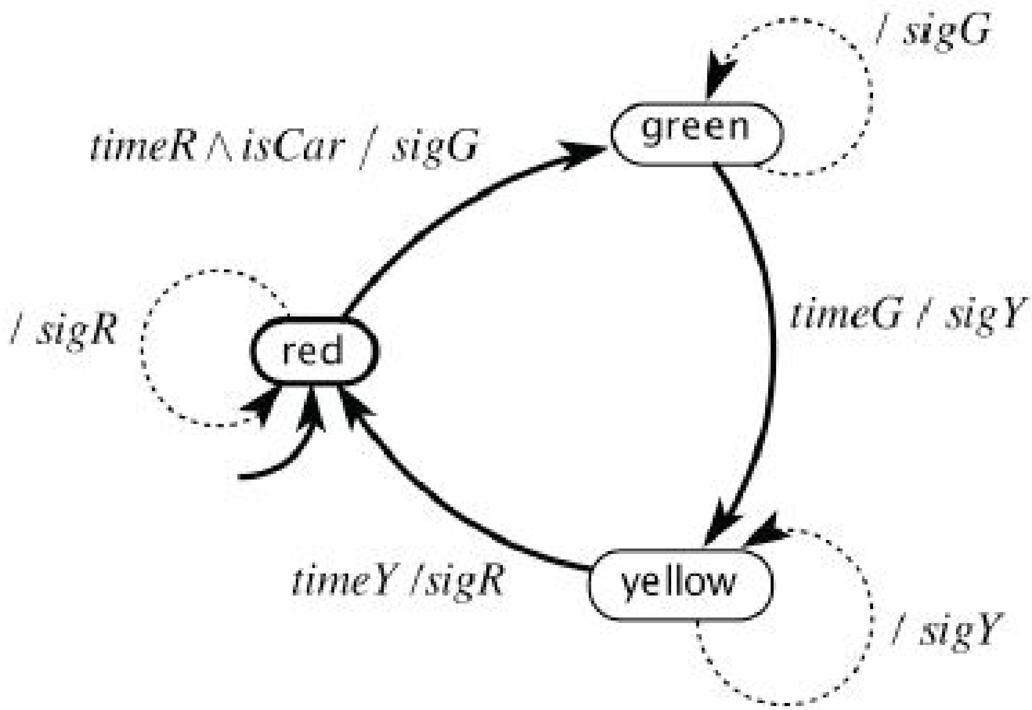


A default transition is enabled if no non-default transition is enabled and it either has no guard or the guard evaluates to true. When is the above default transition enabled?

Time-Triggered Discrete System

- Previous examples: event-based transitions
- This example: time-based transitions

Example: Traffic Light Controller



Some System Properties

- **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.

Nondeterminism

Uses of nondeterminism

1. Modeling unknown aspects of the environment or system

2. Hiding detail in a *specification* of the system

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

3rd Usage of Nondeterminism

Size Matters

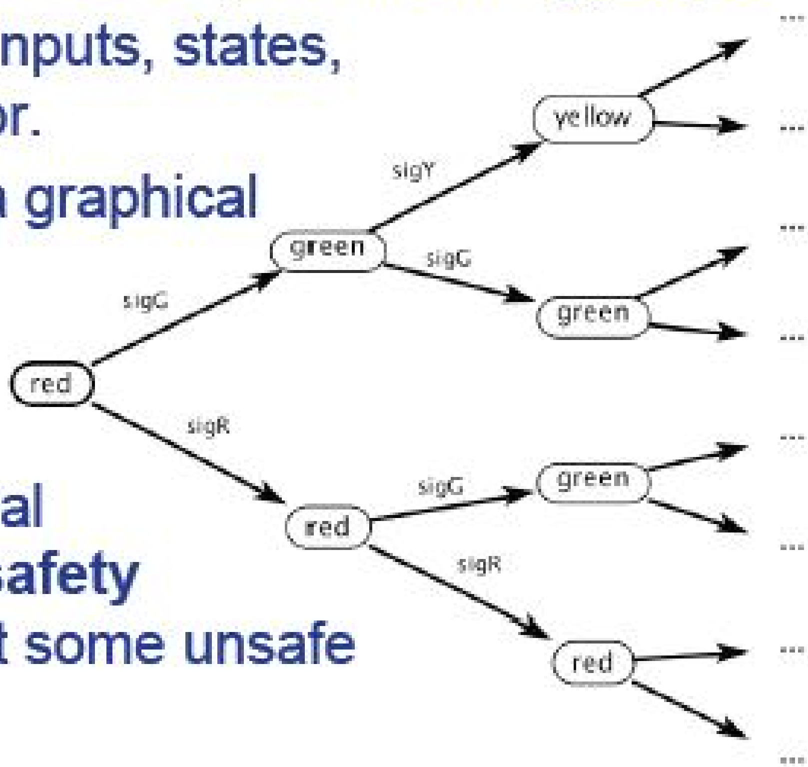
Non-deterministic FSMs are more compact than deterministic FSMs

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

Analysis

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.

Nondeterministic Computation Tree

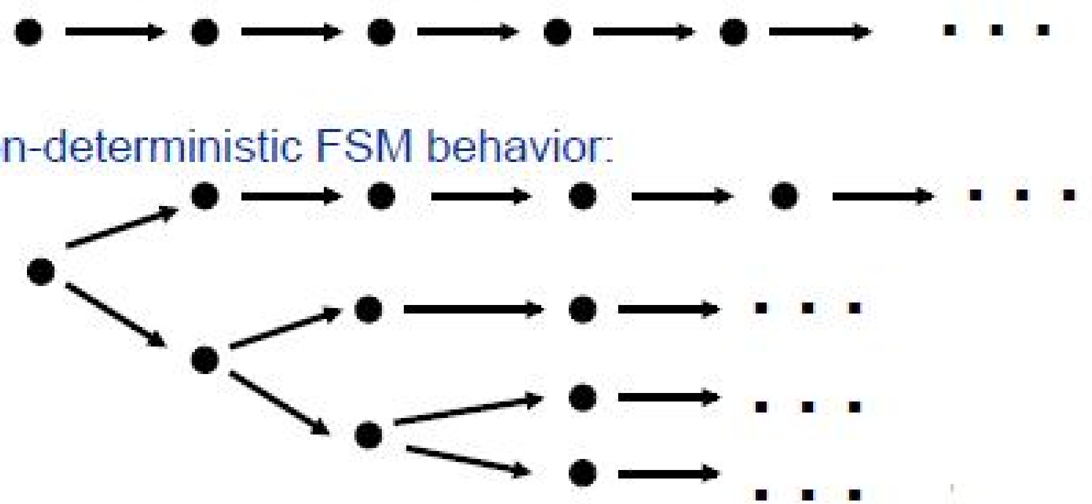
Non-deterministic Behavior: Tree of Computations

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:



Further Thoughts

Related points

What does receptiveness mean for non-deterministic state machines?

Non-deterministic $\neq$ Probabilistic

FSM Representations

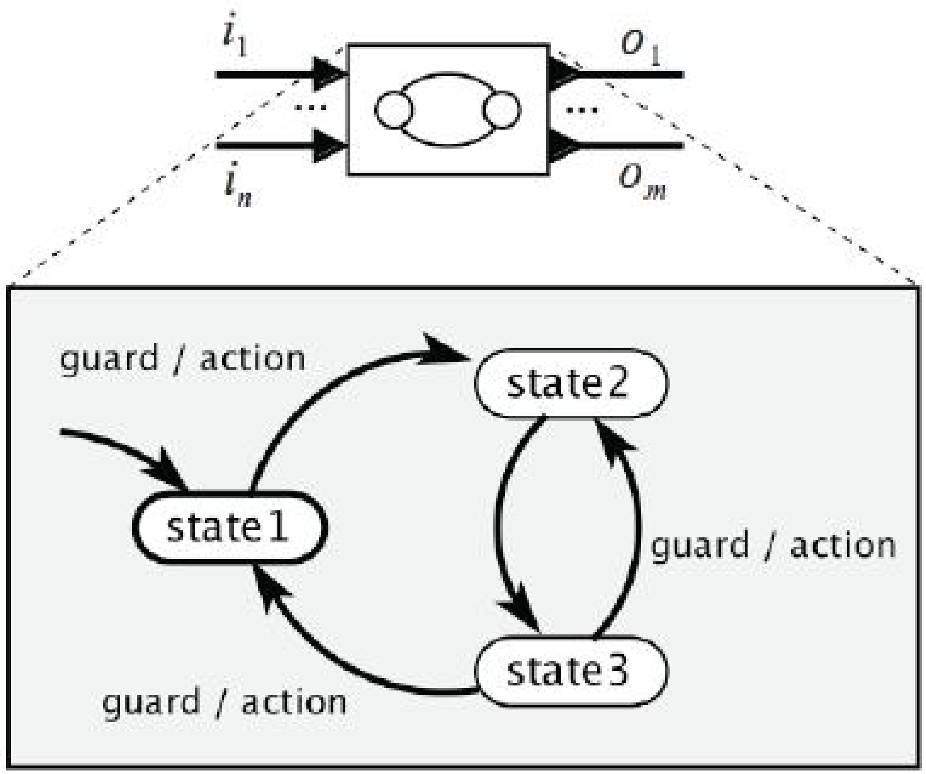
Representing a state machine

1. Pictorial notation
2. Table representing transition relation
3. Functional notation

When would you use each representation?

Introducing FSM Composition

Actor Model of an FSM



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

Summary

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.

Homework Assignments

- Chapter 3:
  - 2, 3, 4, 5, 6
  - 7 and 8: optional
  - Due date: Tuesday 1402/12/22