



# Introduction to Embedded Systems

Sanjit A. Seshia

UC Berkeley  
EECS 149/249A  
Fall 2015

© 2008-2015: E. A. Lee, A. L. Sangiovanni-Vincentelli, S. A. Seshia. All rights reserved.

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

EECS 149/249A, UC Berkeley: 3

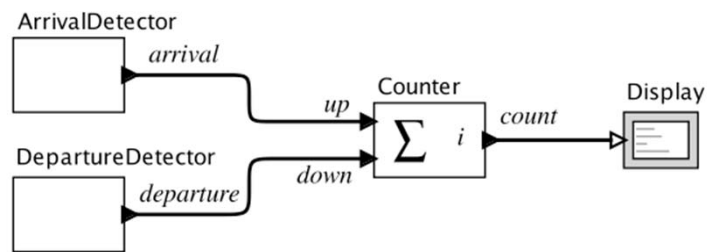
## Discrete Systems: Example Design Problem

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.

EECS 149/249A, UC Berkeley: 4

## Discrete Systems

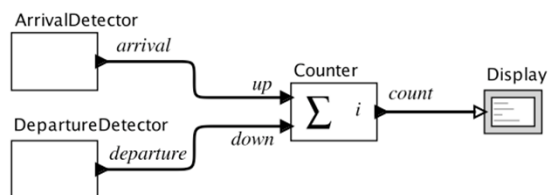
Example: count the number of cars in a parking garage by sensing those that enter and leave:



EECS 149/249A, UC Berkeley: 5

## Discrete Systems

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

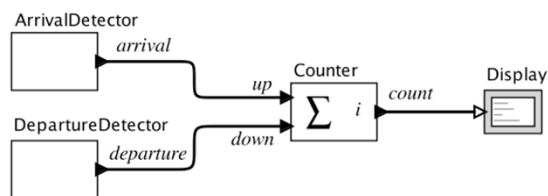


Pure signal:  $up: \mathbb{R} \rightarrow \{absent, present\}$

EECS 149/249A, UC Berkeley: 6

## Discrete Systems

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



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

Discrete actor:

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

$P = \{up, down\}$

EECS 149/249A, UC Berkeley: 7

## Demonstration of Ptolemy II Model (“Program”)

EECS 149/249A, UC Berkeley: 8

## Other Actor Models with different interface?

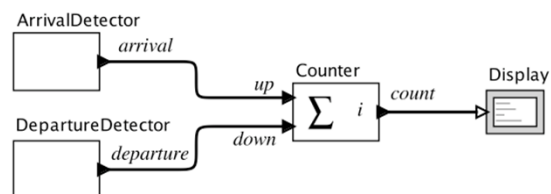
EECS 149/249A, UC Berkeley: 9

## 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



EECS 149/249A, UC Berkeley: 10

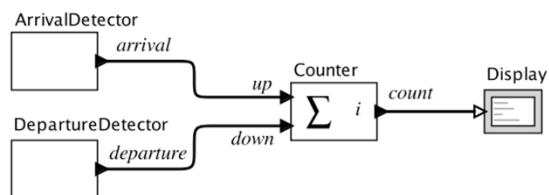
## Inputs and Outputs at a Reaction

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

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

and the outputs are in a set

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

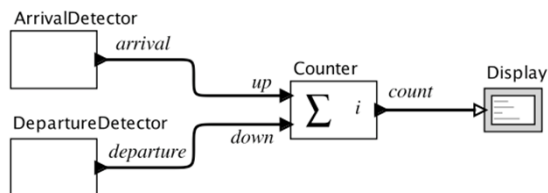


EECS 149/249A, UC Berkeley: 11

## State Space

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

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



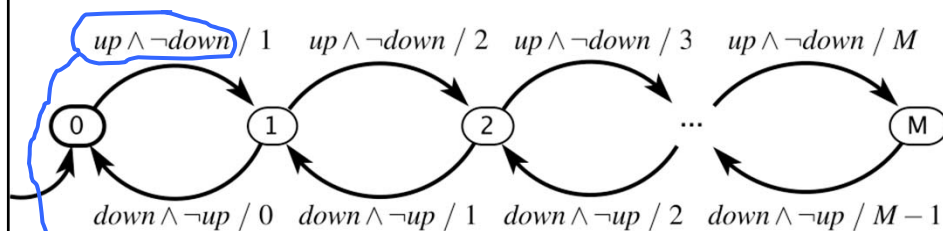
EECS 149/249A, UC Berkeley: 12

## Question

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

EECS 149/249A, UC Berkeley: 13

## Garage Counter Finite State Machine (FSM) in Pictures



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

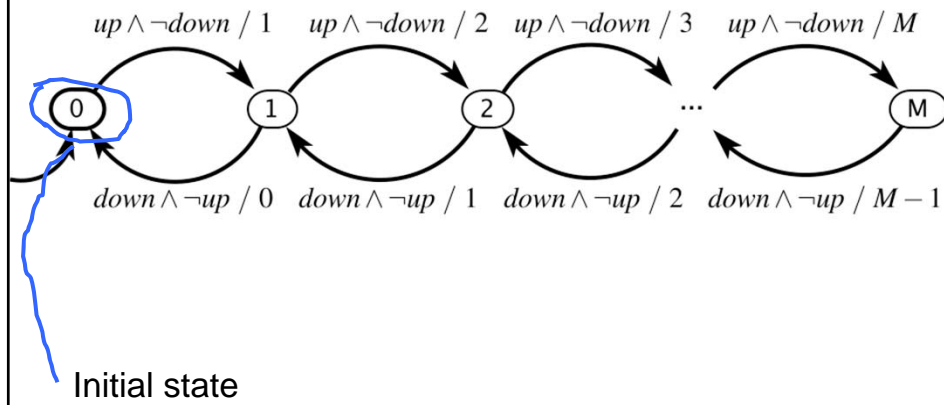
$$up \wedge \neg down$$

which means

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

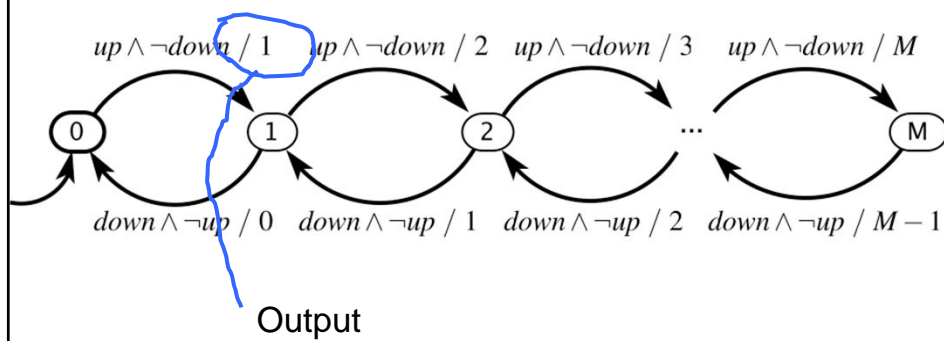
EECS 149/249A, UC Berkeley: 14

## Garage Counter Finite State Machine (FSM) in Pictures



EECS 149/249A, UC Berkeley: 15

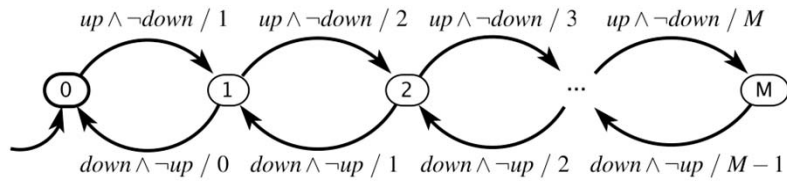
## Garage Counter Finite State Machine (FSM) in Pictures



EECS 149/249A, UC Berkeley: 16



## Garage Counter Mathematical Model



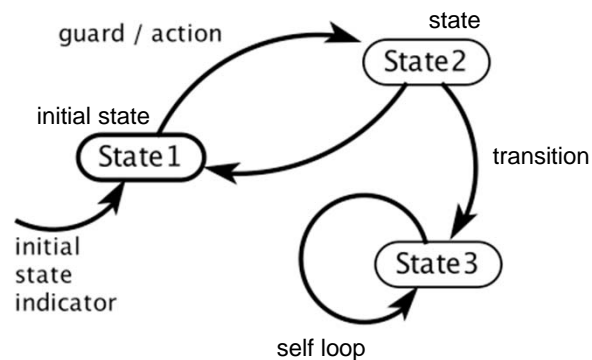
Formally:  $(States, Inputs, Outputs, update, initialState)$ , where

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

The picture above defines the update function.

EECS 149/249A, UC Berkeley: 17

## FSM Notation



EECS 149/249A, UC Berkeley: 19

## Examples of Guards for Pure Signals

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

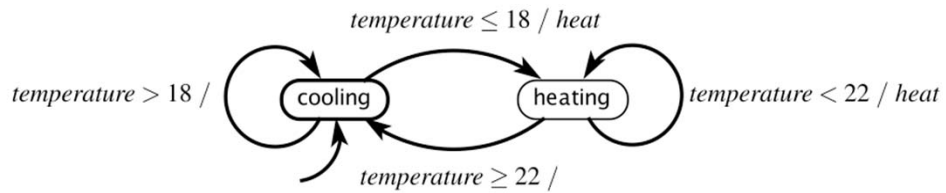
EECS 149/249A, UC Berkeley: 20

## Examples of Guards for Signals with Numerical Values

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

EECS 149/249A, UC Berkeley: 21

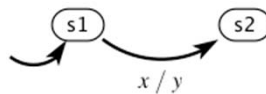
## Example of *Modal* Model: Thermostat



EECS 149/249A, UC Berkeley: 22

## When does a reaction occur?

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



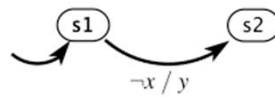
Suppose all inputs are discrete and a reaction occurs *when any input is present*. Then the above transition will be taken whenever the current state is s1 and x is present.

This is an *event-triggered model*.

EECS 149/249A, UC Berkeley: 23

## When does a reaction occur?

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



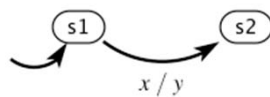
Suppose  $x$  and  $y$  are discrete and pure signals.  
When does the transition occur?

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!

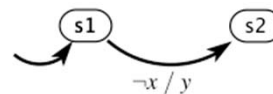
EECS 149/249A, UC Berkeley: 24

## When does a reaction occur?

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



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

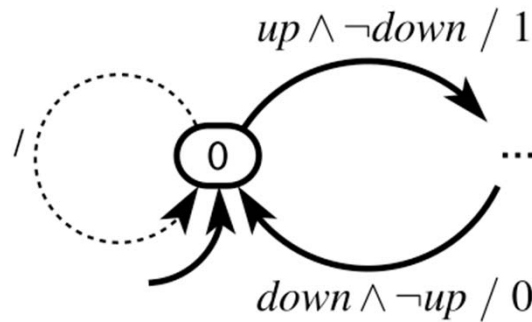


Suppose all inputs are discrete and a reaction occurs *on the tick of an external clock*.

This is a *time-triggered model*.

EECS 149/249A, UC Berkeley: 25

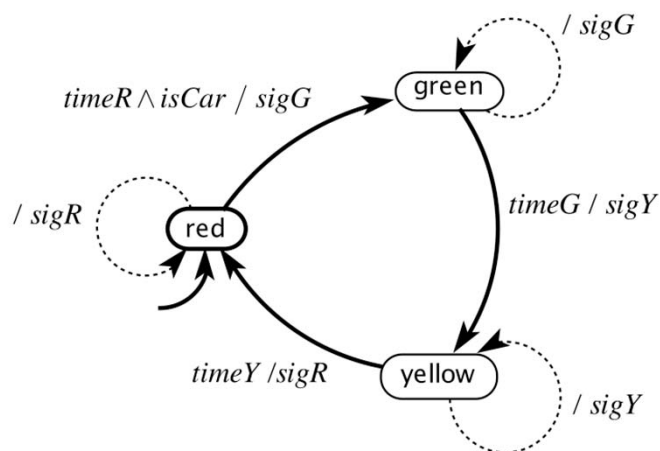
## More Notation: Default Transitions



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?

EECS 149/249A, UC Berkeley: 26

Only show default transitions if they are guarded or produce outputs (or go to other states)  
 Example: Traffic Light Controller



EECS 149/249A, UC Berkeley: 27

## 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.

EECS 149/249A, UC Berkeley: 29

## Test Your Understanding: Three Kinds of Transitions

Self-Loop

Default Transition

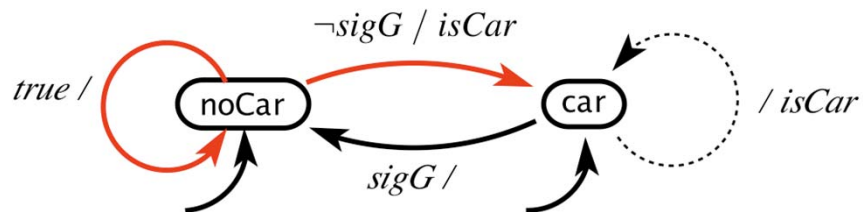
Stuttering Transition

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?

EECS 149/249A, UC Berkeley: 30

## Example: Nondeterministic FSM

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



Formally, the update function is replaced by a function  
 $possibleUpdates : States \times Inputs \rightarrow 2^{States \times Outputs}$

EECS 149/249A, UC Berkeley: 31

$$\begin{aligned}
 & u(\text{noCar}, \overset{\text{sigR}}{\wedge} \overset{\wedge}{\neg \text{sigG}} \overset{\wedge}{\neg \text{sigY}}) \\
 & = \left\{ (\text{car}, \text{isCar}), (\text{noCar}, \neg \text{isCar}) \right\}
 \end{aligned}$$

EECS 149/249A, UC Berkeley: 32

## 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?

EECS 149/249A, UC Berkeley: 33

## 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).

EECS 149/249A, UC Berkeley: 35



## Non-deterministic Behavior: Tree of Computations

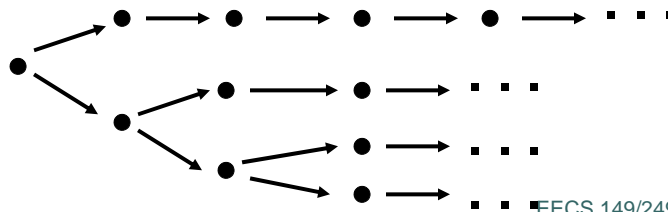
For a fixed input sequence:

- 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:



EECS 149/249A, UC Berkeley: 36

## 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.

EECS 149/249A, UC Berkeley: 37

## Review: Concepts covered

Models = Programs

Actor Models of Discrete Systems: Types and Interfaces

States, Transitions, Guards

Determinism, Receptiveness, etc.

EECS 149/249A, UC Berkeley: 38