

**CSCE 310J**  
**Data Structures & Algorithms**

**Introduction to  
Finite State Automata  
and  
Finite State Machines**

**Dr. Steve Goddard**  
***goddard@cse.unl.edu***

<http://www.cse.unl.edu/~goddard/Courses/CSCE310J>

1

**CSCE 310J**  
**Data Structures & Algorithms**

- ◆ Giving credit where credit is due:
  - » Most of the lecture notes are based on slides created by Drs. Doug Hogan (Penn State), A.E. Eiben, Stephen Cheney, Niraj Shah, Joseph Conron, John Laird and Mike van Lent.
  - » Epp, Susanna S. *Discrete Mathematics with Applications*. 2nd Ed. Belmont, CA: Brooks, 1995.
  - » I have modified them and added new slides

2

**What is a finite state automaton?**

- ◆ A model of computation
- ◆ A set of *states* and how to get from some state to other states

3

**Why do we care?**

- ◆ An ideal representation of a computer
  - » Why?
    - ❖ A state describes the computer at any given point
      - ◆ Programs, memory, etc.
    - ❖ Many states, but still a finite number
- ◆ Represents legal steps of a process
  - » Some computation
  - » Valid inputs
  - » Valid words in a language
    - ❖ Compilers

4

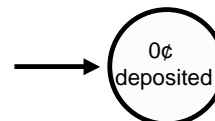
**Vending Machine Example**

- ◆ Accepts only nickels and dimes
- ◆ Everything costs 20 cents
  
- ◆ Consider the *state* of the vending machine
  - » We'll model the state by how much money has been deposited.
  - » What is the initial state of the vending machine?

5

**Vending Machine Initial State**

- ◆ Start with 0¢ deposited.
  - » Called the **initial state** of the machine.
  - » A circle is drawn for the state



- » An arrow pointing to this state marks it as the initial state.

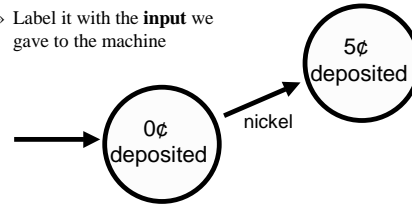
6

### Where to from here?

- ◆ Make transitions to different states...
- ◆ What can we do?
  - » Deposit a nickel
  - » Deposit a dime

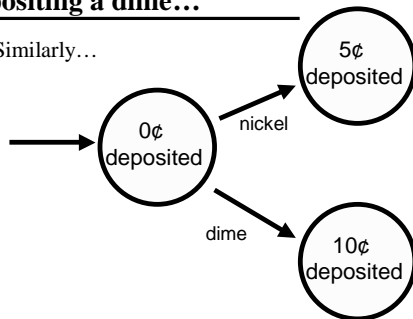
### Depositing a nickel...

- ◆ Takes us to a new state: 5¢ deposited
- ◆ Add an arrow for the transition
  - » Label it with the **input** we gave to the machine



### Depositing a dime...

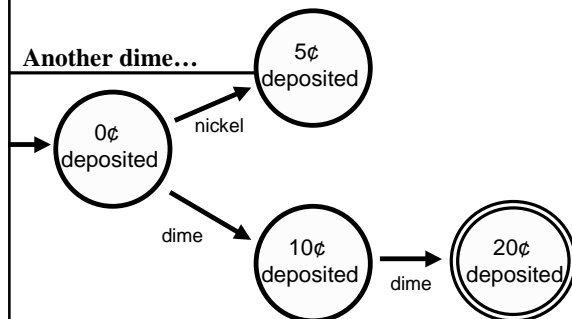
- ◆ Similarly...



### From here...

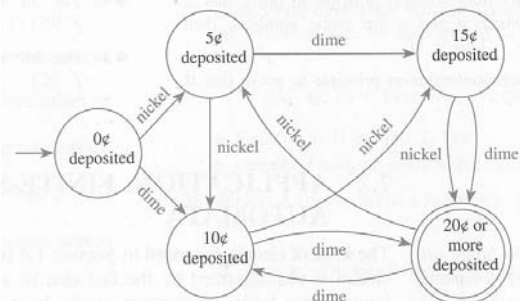
- ◆ There are many possibilities as to what could happen
- ◆ We're going to consider what happens when we deposit another dime.
  - » Goes to a new state: 20¢ deposited.
    - ❖ That means we've put in enough money to buy something.
    - ❖ The vending machine "accepts" 20¢ as a place to stop.
      - ◆ called an **accepting state**

### Another dime...



- ◆ Add new state with **double circle** to denote that it's an **accepting state**.

### Complete Vending Machine

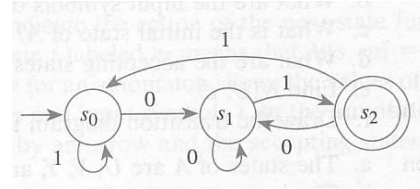


## Another Representation: Next-State Table

TABLE 7.2.1 Next-state Table

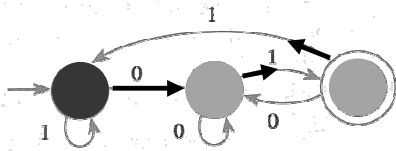
	Input		
	Nickel	Dime	
→	0¢ deposited	5¢ deposited	10¢ deposited
	5¢ deposited	10¢ deposited	15¢ deposited
	10¢ deposited	15¢ deposited	20¢ or more deposited
State	15¢ deposited	20¢ or more deposited	20¢ or more deposited
⊙	20¢ or more deposited	5¢ deposited	10¢ deposited

## Another example



- ◆ Initial state?
- ◆ Accepting state?

## Another example



- ◆ What happens if we input 01?
  - » String is "accepted" by this automaton
- ◆ What about 011?
  - » Go back to non-accepting state. String is rejected.

## Finite State Machine (FSM)

- ◆ Finite automata are similar to FSM's, but
  - » they do not produce any outputs,
  - » they just accept input sequences (an *accepting set of states* is given).
- ◆ Let's define the idea of a "machine"
  - » organism (real or synthetic) that responds to a countable (**finite**) set of stimuli (**events**) by generating predictable responses (**outputs**) based on a history of prior events (current **state**).
- ◆ A finite state machine (FSM) is a computational model of a machine.

## FSM Elements

- ◆ **States** represent the particular configurations that our machine can assume.
- ◆ **Events** define the various inputs that a machine will recognize
- ◆ **Transitions** represent a change of state from a current state to another (possibly the same) state that is dependent upon a specific event.
- ◆ The **Start State** is the state of the machine before it has received any events

## Finite State Machine (FSM)

Moore Machine: is a quintuple:  $M(S, I, O, \delta, \lambda)$

- »  $S$ : finite non-empty set of states
- »  $I$ : finite non-empty set of inputs
- »  $O$ : finite non-empty set of outputs
- »  $\delta$ :  $S \times I \rightarrow S$  transition (or next state) function
- »  $\lambda$ :  $S \rightarrow O$  output function (note: output only a function of present state)

Mealy Machine:  $M(S, I, O, \delta, \lambda)$  but

- »  $\lambda$ :  $S \times I \rightarrow O$  (i.e. output depends on both present state and present input)

- » for digital circuits, typically  $I = \{0,1\}^m$  and  $O = \{0,1\}^n$

In addition, (for both Moore and Mealy machines) certain states are classified as *reset or initial states*

## Machine Types

- ◆ **Mealy machine**
  - » one that generates an output for each transition,
- ◆ **Moore machine**
  - » one that generates an output for each state.
- ◆ Moore machines can do anything a Mealy machine can do (and vice versa).
- ◆ The following example FSM is a Mealy machine.

19

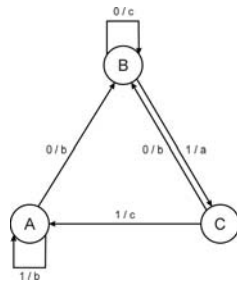
## Prediction by finite state machines

- ◆ Finite state machine (FSM):
  - » States  $S$
  - » Inputs  $I$
  - » Outputs  $O$
  - » Transition function  $\delta : S \times I \rightarrow S \times O$
  - » Transforms input stream into output stream
- ◆ Can be used for predictions, e.g. to predict next input symbol in a sequence

20

## FSM example

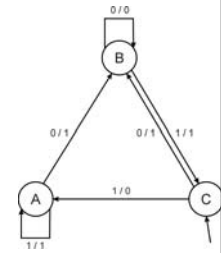
- ◆ Consider the FSM with:
  - »  $S = \{A, B, C\}$
  - »  $I = \{0, 1\}$
  - »  $O = \{a, b, c\}$
  - »  $\delta$  given by a diagram



21

## FSM as predictor

- ◆ Consider the following FSM
- ◆ Task: predict next input
- ◆ Quality: % of  $in_{(i+1)} = out_i$
- ◆ Given initial state C
- ◆ Input sequence 011101
- ◆ Leads to output 110111
- ◆ Quality: 3 out of 5



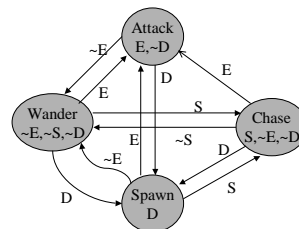
22

## Quake Bot Example

- ◆ Types of behavior to capture:
  - » Wander randomly if don't see or hear an enemy
  - » When see enemy, attack
  - » When hear an enemy, chase enemy
  - » When die, respawn
  - » When health is low and see an enemy, retreat
- ◆ Extensions:
  - » When see power-ups during wandering, collect them
- ◆ Borrowed from John Laird and Mike van Lent's GDC tutorial

23

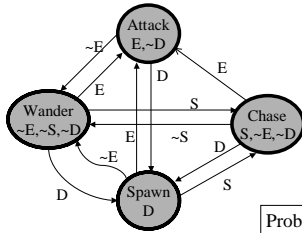
## Example FSM



- ◆ States:
  - » E: enemy in sight
  - » S: sound audible
  - » D: dead
- ◆ Events:
  - » E: see an enemy
  - » S: hear a sound
  - » D: die
- ◆ Action performed:
  - » On each transition
  - » On each update in some states (e.g. attack)

24

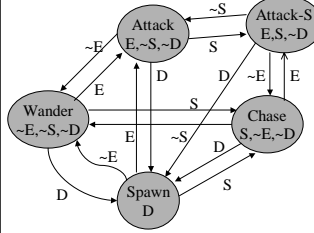
### Example FSM Problem



- ◆ States:
  - » E: enemy in sight
  - » S: sound audible
  - » D: dead
- ◆ Events:
  - » E: see an enemy
  - » S: hear a sound
  - » D: die

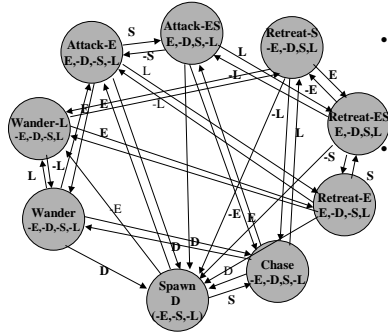
Problem: Can't go directly from attack to chase. Why not?

### Better Example FSM



- ◆ States:
  - » E: enemy in sight
  - » S: sound audible
  - » D: dead
- ◆ Events:
  - » E: see an enemy
  - » S: hear a sound
  - » D: die
- ◆ Extra state to recall whether or not heard a sound while attacking

### Example FSM with Retreat

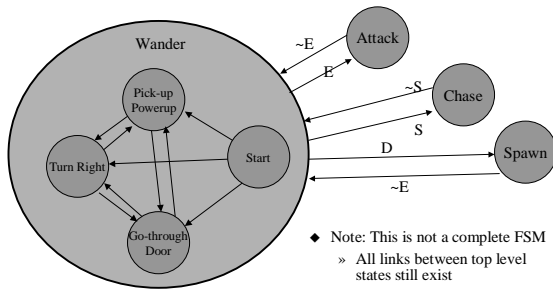


- States:
  - E: enemy in sight
  - S: sound audible
  - D: dead
  - L: Low health
- Worst case: Each extra state variable can add  $2n$  extra states
- $n$  = number of existing states

### Hierarchical FSMs

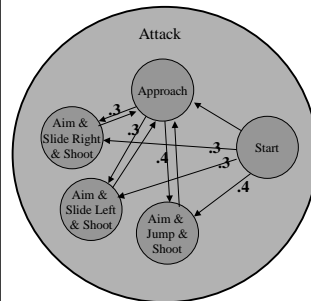
- ◆ What if there is no simple action for a state?
- ◆ Expand a state into its own FSM, which explains what to do if in that state
- ◆ Some events move you around the same level in the hierarchy, some move you up a level
- ◆ When entering a state, have to choose a state for it's child in the hierarchy
  - » Set a default, and always go to that
  - » Or, random choice
  - » Depends on the nature of the behavior

### Hierarchical FSM Example



- ◆ Note: This is not a complete FSM
  - » All links between top level states still exist
  - » Need more states for wander

### Non-Deterministic Hierarchical FSM (Markov Model)



- ◆ Adds variety to actions
- ◆ Have multiple transitions for the same event
- ◆ Label each with a probability that it will be taken
- ◆ Randomly choose a transition at run-time
- ◆ Markov Model: New state only depends on the previous state