LECTURE 3: STATES AND EVENTS
Example: States and Events
Example: States and Events
Event Driven Transitions

Finite State Automata
States and Events

- A **state** is a condition of a system at some point in time.
- An **event** is something that cause a system to shift from one state to another.
Finite State Automata (FSA)

- a set of states
  - one designated start state
- input alphabet
  - these are events
- a transition function that maps input symbols and current states to a next state
FSA Example: A Light Switch
FSA Example: A Light Switch

- states = \{ ON, OFF \}
  - start state = OFF
- alphabet = \{ toggle \}
- transition function:

<table>
<thead>
<tr>
<th>current state</th>
<th>ON</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>toggle</td>
<td>OFF</td>
<td>ON</td>
</tr>
</tbody>
</table>
FSA: Ghost

- ROAM
  - see=false
  - blue=true

- CHASE
  - see=true

- EVADE
  - see=true
  - blue=true
FSA Example: Ghost

- states = { ROAM, CHASE, EVADE }
  - start state = ROAM
- alphabet = { see=true, see=false, blue=true }
- transition function:

<table>
<thead>
<tr>
<th>current state</th>
<th>ROAM</th>
<th>CHASE</th>
<th>EVADE</th>
</tr>
</thead>
<tbody>
<tr>
<td>see=true</td>
<td>CHASE</td>
<td>CHASE</td>
<td>CHASE</td>
</tr>
<tr>
<td>see=false</td>
<td>ROAM</td>
<td>ROAM</td>
<td>ROAM</td>
</tr>
<tr>
<td>blue=true</td>
<td>EVADE</td>
<td>EVADE</td>
<td>EVADE</td>
</tr>
</tbody>
</table>
Traffic Lights: System

inputs:
4 sensors
clock (every 5 seconds)

sensors

lights

output:
4 lights: red, yellow, green
Traffic Lights: Analysis

- East-West lights are the same
- North-South lights are the same
- Clock is used to introduce delays
  - lights do not immediately switch when a vehicle is sensed
  - caution (yellow) for 10 seconds before stop (red)
Traffic Lights: States

- State is made up of three values:
  - NS: color of north and south lights
  - EW: color of east and west lights
  - D: integer used to distinguish between delay states

NS: G
EW: R
D: 0
Traffic Lights: Inputs

- Input Alphabet:
  - $S_{NS}$: north-bound or south-bound sensor triggered
  - $S_{EW}$: east-bound or west-bound sensor triggered
  - $C$: clock elapsed
Traffic Lights: FSA

NS: G  EW: R  D: 0

NS: G  EW: R  D: 1

NS: Y  EW: R  D: 2

NS: Y  EW: R  D: 1

NS: R  EW: G  D: 0

NS: R  EW: G  D: 1

NS: R  EW: Y  D: 1

NS: R  EW: Y  D: 2

C

S_EW

C

S_NS
Once the FSA is in tabular form, implementation is straightforward:

- hardcode: nested switch statements
- flexible: read FSA from a table data structure
## Traffic Lights: Tabular FSA

<table>
<thead>
<tr>
<th></th>
<th>GR0</th>
<th>GR1</th>
<th>YR2</th>
<th>YR1</th>
<th>RG0</th>
<th>RG1</th>
<th>RY2</th>
<th>RY1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 = C</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>2 = S_{NS}</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>3 = S_{EW}</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

States and inputs mapped to integer codes, to enable table lookup.
Implementation: Python

class TrafficLights:

    def __init__(self):
        self.state = 0
        self.transitions = ((0,2,3,4,4,6,7,0), (0,1,2,3,5,5,6,7), (1,1,2,3,4,5,6,7))
        self.NS_output = ('green', 'green', 'yellow', 'yellow', 'red', 'red', 'red', 'red')
        self.EW_output = ('red', 'red', 'red', 'red', 'green', 'green', 'yellow', 'yellow')

    def update(self, input):
        if (input < 1) or (input > 3): return
        self.state = self.transitions[input-1][self.state]

    def display(self):
        print
        print 'NS lights: ' + self.NS_output[self.state]
        print 'EW lights: ' + self.EW_output[self.state]
        print
Implementation: Python

```python
lights = TrafficLights()

while True:
    lights.display()
    input = int(raw_input("next input: "))
    if input == 0: break
    lights.update(input)
```
Exercise

- Design a FSA for a soda vending machine
Probabilistic Transitions

Markov Models
Nondeterministic Automata

- A nondeterministic automata has probabilistic transitions
  - called Markov models
- Implemented using time-stepping
  - at each clock, probabilities are used to determine next state
Markov Example: Diabetes Patients

- **Three states:**
  - CD: controlled diabetes
  - ESRD: end-stage renal disease
  - D: death

- **Transitions:**
  - Each year, patients with controlled diabetes have an 85% chance of staying in that state, a 10% chance of moving to the ESRD state, and a 5% chance of dying.
  - Each year, patients with ESRD have a 70% chance of continuing in the ESRD state and a 30% chance of dying.
  - Patients who have died have a 100% chance of staying dead.
Markov Example: Diabetes Patients

- Time step: 1 year

- Transition probabilities:
  - From CD to CD: 0.85
  - From CD to ESRD: 0.1
  - From ESRD to CD: 0.05
  - From ESRD to D: 0.7
  - From D to ESRD: 0.3
  - From D to D: 1.0

- Probabilities out of any state must sum to 1.0
Markov Model Implementation

- Generate a random number (0.0, 1.0)
- Use random number and current state to generate next state

  if state is CD:
    if \( \text{rand} < 0.1 \): state \( \leftarrow \) ESRD
    else if \( \text{rand} < 0.1+0.05 \): state \( \leftarrow \) D
    else state \( \leftarrow \) CD
  else if state is ESRD:
    if \( \text{rand} < 0.3 \): state \( \leftarrow \) D
    else state \( \leftarrow \) ESRD
  else:
    state \( \leftarrow \) D

note: need to add up previous probabilities
Implementation: C++

class DiabeticPatient
{
  public:

    enum State { CD, ESRD, D };

    // constructor: set initial state and transition probabilities
    DiabeticPatient();

    // Transition function: changes state
    void update();

    // output functions: simply return current state
    State output() { return state; }

  private:

    State state; // current state
    double transition_probability[3][3]; // transition table
};
Implementation: C++

// constructor: set initial state and transition probabilities

DiabeticPatient::DiabeticPatient() : state(CD)
{
    transition_probability[CD][CD]   = 0.85;
    transition_probability[CD][ESRD] = 0.10;
    transition_probability[CD][DEAD] = 0.05;
    transition_probability[ESRD][CD] = 0.00;
    transition_probability[ESRD][ESRD] = 0.70;
    transition_probability[ESRD][DEAD] = 0.30;
    transition_probability[DEAD][CD] = 0.00;
    transition_probability[DEAD][ESRD] = 0.00;
    transition_probability[DEAD][DEAD] = 1.00;
}
Implementation: C++

// Transition function: change state based on current state and probability parameters.
// Each update represents one year.

void DiabeticPatient::update() {
    float r = double(rand())/RAND_MAX;
    if (r < transition_probability[state][CD]) state = CD;
    else if (r < transition_probability[state][CD] + transition_probability[state][ESRD]) state = ESRD;
    else state = D;
}
Utilizing the Simulation

- How would you use this simulation to answer the following questions?
  - Suppose we begin with 10,000 patients with controlled diabetes. In 10 years, how many patients will still have controlled diabetes? How many will have ESRD? How many will have died?
  - In 20 years, how many patients will be in each state?
  - How many years will it take before all of the patients are dead?
Implementation: C++

```cpp
void runSimulation(int years, int num_patients)
{
    int t, p;
    int counts[3] = {0,0,0};

    for (p=0; p<num_patients; p++)
    {
        DiabeticPatient dp;
        for (t=0; t<years; t++) dp.update();
        counts[dp.output()]++;
    }

    cout << endl;
    cout << "Simulation of " << num_patients
         << " for " << years << " years." << endl;
    cout << "Controlled diabetes: " << counts[0] << endl;
    cout << "End-stage renal disease: " << counts[1] << endl;
    cout << "Dead: " << counts[2] << endl;
}
```
Implementation: C++

```c++
void main()
{
    // initialize random number generator.
    srand(time(0));

    runSimulation(10, 10000);
    runSimulation(20, 10000);
}
```

Sample run:

Simulation of 10000 for 10 years.
Controlled diabetes: 1971
End-stage renal disease: 1130
Dead: 6899

Simulation of 10000 for 20 years.
Controlled diabetes: 388
End-stage renal disease: 243
Dead: 9369
Exercise

The ACME Chocolate Factory makes four types of chocolates: plain, almond, coconut and almond with coconut.

• The factory produces one piece of candy at time - the type depends on the state of two switches (an almond switch and a coconut switch).
• The switches cannot be moved while a piece of chocolate is being made, but either or both can be moved between pieces.
• When the factory is turned on at the beginning of the day, each switch has a 50% chance of being on and a 50% chance of being off.

Unfortunately, someone has let a monkey loose in the factory. It has locked itself in the control room where the switches are located.

• In between pieces of candy, there is a 30% chance that the monkey will change the state of each control level.
• The probabilities for each switch are independent of each other.

Build a model that would allow you to simulate the sequence of chocolates produced under the monkey's control.
Transition Rules

Production Models
Production Models

- Production models allow us to define rules for transitions between *groups* of states.
- This gives us a more expressive notation than the basic FSA.
- Production models are based on AI techniques.
Since search forms the core of many intelligent processes, it is useful to structure AI programs in a way that facilitates describing and performing the search process. Production systems provide such structures. A definition of a production system is given below. Do not be confused by other uses of the word production, such as to describe what is done in factories. A production system consists of:

- A set of rules, each consisting of a left side (a pattern) that determines the applicability of the rule and a right side that describes the operation to be performed if the rule is applied.\(^3\)

- One or more knowledge/databases that contain whatever information is appropriate for the particular task. Some parts of the database may be permanent, while other parts of it may pertain only to the solution of the current problem. The information in these databases may be structured in any appropriate way.

- A control strategy that specifies the order in which the rules will be compared to the database and a way of resolving the conflicts that arise when several rules match at once.

- A rule applier.
Example: Water Jug Puzzle

- jug A hold 3 gallons, jug B holds 4 gallons
- Operations:
  - Fill either jug
  - Empty either jug
  - Transfer all water from one jug to the other
  - Fill one jug from the other jug
- Goal: Leave 2 gallons in jug B
Jug Problem: FSA Solution

- There are 14 states and 8 input symbols.
- It is tedious to individually define all possible transitions between these states.
- We can instead parameterize the operations to develop transition rules (productions).
1. OPERATOR 1: empty(J).

(a) Empty jug J ∈ \{A, B\}.
(b) For J = A, (X, Y|X > 0) → (0, Y) and ΔT = X/10.
(c) For J = B, (X, Y|Y > 0) → (X, 0) and ΔT = Y/10.
2. OPERATOR 2: \textit{fill}(J).

(a) Fill jug $J \in \{A, B\}$.

(b) For $J = A$, $(X, Y|X < 3) \rightarrow (3, Y)$ and $\Delta T = (3 - X)/2$.

(c) For $J = B$, $(X, Y|Y < 4) \rightarrow (X, 4)$ and $\Delta T = (4 - Y)/2$. 
3. OPERATOR 3: \textit{transfer\_all}(J1, J2).

(a) Transfer all water from jug $J1$ to jug $J2$.

(b) For $J1 = A$, $(X, Y | X + Y \leq 4 \land X > 0 \land Y < 4) \rightarrow (0, X + Y)$, and $\Delta T = X/10$.

(c) For $J1 = B$, $(X, Y | X + Y \leq 3 \land Y > 0 \land X < 3) \rightarrow (X + Y, 0)$, and $\Delta T = Y/10$. 
Production Rule 4

4. OPERATOR 4: $\text{transfer\_full}(J_1, J_2)$.

(a) Transfer enough water from jug $J_1$ to fill $J_2$.
(b) For $J_1 = A$, $(X, Y | X + Y \geq 4 \land X > 0 \land Y < 4) \rightarrow (X - (4 - Y), 4)$, and $\Delta T = (4 - Y) / 10$.
(c) For $J_1 = B$, $(X, Y | X + Y \geq 3 \land Y > 0 \land X < 3) \rightarrow (3, Y - (3 - X))$, and $\Delta T = (3 - X) / 10$. 
Water Jug System: FSA

Start state

Goal state
Water Jug System: Solution
SOURCES

- Traffic Light System, Water Jug Problem:
  - *Simulation Model Design and Execution*
    by Paul A. Fishwick, Prentice Hall, 1995

- Diabetes Model:
  - http://araw.medec.uic.edu/~alansz/courses/mhpe494/markov.html

- Ghost and Ant FSA:
  - *AI for Game Developers*
    by David M. Bourg & Glenn Seeman, O’Reilly, 2004