Announcements

- Quiz 9 is today
- Homework 10 is due on Thursday
- Read 13.4 and 13.5

13.2 – FSMs with Output

- Finite-State Machines
- Language Recognizers
Finite-State Machines

- A Finite-State Machine (FSM) is a computer having only a constant, finite number of distinct logical states.
  - A constant independent of the input size.

- All real computers are FSMs, in a certain sense.
  - But only if you deny yourself any possibility of adding more memory, disk, or other external storage.
  - If so, then there will certain problems that your computer can never solve simple because it will run out of memory.
    - But it could have solved the problem if it had more memory!

Finite-State Machines

- We use FSMs to characterize what classes of problems a computer can solve even if it only has a very small memory.

- At any point in time, an FSM has a current state, and (optionally) an input, and an output.
  - We typically express time for an FSM in discrete steps.

- We can represent an FSM with a directed graph showing the movement from one state to the next.
Describing FSMs with State Tables

• An FSM is **deterministic** if the state is completely determined by the previous state and the FSM’s input.

• The most general way of describing a deterministic FSM is with a **transition function** that maps (old state, input) to (new state, output).

• Any transition function can be expressed using a **state table**. (Lists all function values).

Motivation to use Finite State Machines

• Many applications of FSMs

• Control systems of all sorts
  – GUI interaction model
  – Elevator, vending machine, car-building robot

• Compilers, interpreters

• Simple parsers for languages
Elements of a FSM

• 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.

Example - Serial Adder

• Gets inputs as bits
• The state here is:
  – Having a carry bit
• If there is a carry, add it to the sum
• Otherwise do nothing
• So INPUT & STATE determine together the OUTPUT

\[ \begin{align*}
\text{start} & \overset{00/0}{\rightarrow} \overset{01/1}{\rightarrow} \\overset{11/0}{\rightarrow} \\overset{01/0}{\rightarrow} \\
\text{NC} & \overset{10/1}{\rightarrow} \\overset{00/1}{\rightarrow} \\overset{11/1}{\rightarrow} \\
\text{C} & \overset{10/0}{\rightarrow} \\
\end{align*} \]
Serial Adder FSM

If I am in state C and I receive a 0 and a 1, I am adding 0+1+1 (from the carry state). Therefore I stay in the carry state and output a 0.

Size of FSMs

- **The information capacity of an FSM is** $C = \log |S|$.
  - Thus, if we represent a machine having an information capacity of $C$ bits as an FSM, then its state transition graph will have $|S| = 2^C$ nodes.
- **Computers can be viewed as huge FSMs**
  - *E.g.* suppose your desktop computer has a 512MB memory, and 60GB hard drive.
    - Its information capacity, including the hard drive and memory (and ignoring the CPU’s internal state), is then roughly $512 \times 2^{23} + 60 \times 2^{33} = 519,691,042,816$ b.
    - How many states would be needed to write out the machine’s entire transition function graph?

2519,691,042,816

How many states do I need to represent 3 bits?

8
Vending Machine Example

A vending machine can be modeled as an FSM.

The *state* of the machine records how much money the customer has inserted.

The *input* of the machine is which coin was inserted, or button pressed.

The *output* of the machine is the release of products and/or change.

Suppose a certain vending machine accepts nickels, dimes, and quarters.

– If >30¢ is deposited, change is immediately returned.

If the “coke” button is pressed, the machine drops a coke.

– Can then accept a new payment.
Modeling the Machine

- **Input symbol set:**
  \[ I = \{\text{nickel, dime, quarter, button}\} \]
  - We could add “nothing” or \( \emptyset \) as an additional input symbol if we want.
  - Representing “no input at a given time.”

- **Output symbol set:**
  \[ O = \{\emptyset, 5\, \text{¢}, 10\, \text{¢}, 15\, \text{¢}, 20\, \text{¢}, 25\, \text{¢}, \text{coke}\} \]

- **State set:**
  \[ S = \{0, 5, 10, 15, 20, 25, 30\} \]
  - Representing how much money has been taken.

---

**Transition Function Table**

<table>
<thead>
<tr>
<th>Old state</th>
<th>Input</th>
<th>New state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>n</td>
<td>5</td>
<td>\emptyset</td>
</tr>
<tr>
<td>0</td>
<td>d</td>
<td>10</td>
<td>\emptyset</td>
</tr>
<tr>
<td>0</td>
<td>q</td>
<td>25</td>
<td>\emptyset</td>
</tr>
<tr>
<td>0</td>
<td>b</td>
<td>0</td>
<td>\emptyset</td>
</tr>
<tr>
<td>5</td>
<td>n</td>
<td>10</td>
<td>\emptyset</td>
</tr>
<tr>
<td>5</td>
<td>d</td>
<td>15</td>
<td>\emptyset</td>
</tr>
<tr>
<td>5</td>
<td>q</td>
<td>30</td>
<td>\emptyset</td>
</tr>
<tr>
<td>5</td>
<td>b</td>
<td>5</td>
<td>\emptyset</td>
</tr>
<tr>
<td>10</td>
<td>n</td>
<td>15</td>
<td>\emptyset</td>
</tr>
<tr>
<td>10</td>
<td>d</td>
<td>20</td>
<td>\emptyset</td>
</tr>
<tr>
<td>10</td>
<td>q</td>
<td>30</td>
<td>5¢</td>
</tr>
<tr>
<td>10</td>
<td>b</td>
<td>10</td>
<td>\emptyset</td>
</tr>
<tr>
<td>15</td>
<td>n</td>
<td>20</td>
<td>\emptyset</td>
</tr>
<tr>
<td>15</td>
<td>d</td>
<td>25</td>
<td>\emptyset</td>
</tr>
<tr>
<td>15</td>
<td>q</td>
<td>30</td>
<td>10¢</td>
</tr>
<tr>
<td>15</td>
<td>b</td>
<td>15</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>
Transition Function Table cont.

<table>
<thead>
<tr>
<th>Old state</th>
<th>Input</th>
<th>New state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>n</td>
<td>25</td>
<td>∅</td>
</tr>
<tr>
<td>20</td>
<td>d</td>
<td>30</td>
<td>∅</td>
</tr>
<tr>
<td>20</td>
<td>q</td>
<td>30</td>
<td>15¢</td>
</tr>
<tr>
<td>20</td>
<td>b</td>
<td>20</td>
<td>∅</td>
</tr>
<tr>
<td>25</td>
<td>n</td>
<td>30</td>
<td>∅</td>
</tr>
<tr>
<td>25</td>
<td>d</td>
<td>30</td>
<td>5¢</td>
</tr>
<tr>
<td>25</td>
<td>q</td>
<td>30</td>
<td>20¢</td>
</tr>
<tr>
<td>25</td>
<td>b</td>
<td>25</td>
<td>∅</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Old state</th>
<th>Input</th>
<th>New state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>n</td>
<td>30</td>
<td>5¢</td>
</tr>
<tr>
<td>30</td>
<td>d</td>
<td>30</td>
<td>10¢</td>
</tr>
<tr>
<td>30</td>
<td>q</td>
<td>30</td>
<td>25¢</td>
</tr>
<tr>
<td>30</td>
<td>b</td>
<td>0</td>
<td>coke</td>
</tr>
</tbody>
</table>

Another Format: State Table

<table>
<thead>
<tr>
<th>Old state</th>
<th>n</th>
<th>d</th>
<th>q</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5,∅</td>
<td>10,∅</td>
<td>25,∅</td>
<td>0,∅</td>
</tr>
<tr>
<td>5</td>
<td>10,∅</td>
<td>15,∅</td>
<td>30,∅</td>
<td>5,∅</td>
</tr>
<tr>
<td>10</td>
<td>15,∅</td>
<td>20,∅</td>
<td>30,5¢</td>
<td>10,∅</td>
</tr>
<tr>
<td>15</td>
<td>20,∅</td>
<td>25,∅</td>
<td>30,10¢</td>
<td>15,∅</td>
</tr>
<tr>
<td>20</td>
<td>25,∅</td>
<td>30,∅</td>
<td>30,15¢</td>
<td>20,∅</td>
</tr>
<tr>
<td>25</td>
<td>30,∅</td>
<td>30,5¢</td>
<td>30,20¢</td>
<td>25,∅</td>
</tr>
<tr>
<td>30</td>
<td>30,5¢</td>
<td>30,10¢</td>
<td>30,25¢</td>
<td>0,coke</td>
</tr>
</tbody>
</table>

Each entry shows new state, output symbol
Directed-Graph State Diagram

As you can see, these can get kind of busy.

Formalizing FSMs

A finite-state machine \( M=(S, I, O, f, g, s_0) \)

- \( S \) is the state set.
- \( I \) is the alphabet (vocabulary) of input symbols
- \( O \) is the alphabet (vocab.) of output symbols
- \( f \) is the state transition function
- \( g \) is the output function
- \( s_0 \) is the initial state.

- Our transition function from before is \( T=(f,g) \).
13.3– FSMs w/o Output

- Kleene closures
- Deterministic Finite-state Automata (DFA)
- Nondeterministic Finite-state Automata (NFA)
- Language recognition

Background

- Suppose A and B are sets of containing strings
  - We define the concatenation of A and B, denoted by AB, as the set of all strings of the form xy, where x is a string in A and y is a string in B
- Example:
  - A = \{0,11\} B = \{1,10,110\}

- Find AB and BA
  - AB = ?
    - \{01, 010, 0110, 11, 1110, 11110\}
  - BA = ?
    - \{10, 111, 100, 1011, 1100, 11011\}

- Definition:
  - An empty string or null string is defined as \(\lambda\) (lambda)
Background

- We also define $A^n$ for $n = 0, 1, 2$ as a recursive definition of concatenation
  - $A^0 = \{\lambda\}$
  - $A^{n+1} = A^n A$ for $n = 0, 1, 2, ...$

- Example:
  Let $A = \{1, 00\}$ Find $A^n$ for $n=0,1,2$
  
  $A^0 = \ ?$
  - $\{\lambda\}$
  $A^1 = \ ?$
  - $A^0 A = \{\lambda\} A = \{1, 00\}$
  $A^2 = \ ?$
  - $\{11, 100, 001, 0000\}$

Kleene Closure

- $A^*$, The Kleene closure of $A$, is $\bigcup_{k \geq 0} A^k$.
  - The set of all strings of symbols in $A$.

- What is the Kleene closure of the set $A = \{1\}$?
  - $A^* = \{\lambda, 1, 1111, ...\}$

- How about the set $B = \{0, 1\}$?
  - $B^* = \{\lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, ...\}$
Finite-State Automata

- A.k.a., “Deterministic Finite Automata” (DFAs)
- Like FSMs, but without producing an output at every step.
- Instead, there is a set of final states.
- The “output” of the machine can be considered to be the choice of whether it is in a final state (and if so, which one) at a given point in time.

DFA’s, formally

A deterministic finite automaton is a quintuple
\[ M = (S, I, f, s_0, F) \]
- \( S \) = set of states
- \( I \) = finite input alphabet
- \( f \) = transition function that assigns a next state to every pair of input and state
- \( s_0 \) = start state
- \( F \) = Set of final states.
Examples

What language does this recognize?

Set of bit strings that end with two 0s

Examples

What language does this recognize?

Set of bit strings that begin with two 0s
Examples

What language does this recognize?

Set of bit strings that contain two consecutive 0s

Examples

What language does this recognize?

Set of bit strings that contain at least two 0s
Examples

What language does this recognize?

Set of bit strings that do not contain two consecutive 0s

Non-Deterministic Finite-State Automata (NFA)

- Similar to DFA but many contain multiple state transitions for a particular input symbol

- A language is accepted if one of the possible states the NFA could be in accepts the language

- Any NFA can be converted into a DFA
Nondeterministic Finite Automaton (NFA)

Alphabet = \{a\}

\[ \begin{align*}
    q_0 &\xrightarrow{a} q_1 \\
    q_1 &\xrightarrow{a} q_2 \\
    q_0 &\xrightarrow{a} q_3 \\
\end{align*} \]

Two choices
Alphabet = \{a\}

Two choices

First Choice
First Choice

All input is consumed

Second Choice
Second Choice

No transition: the automaton hangs
An NFA accepts a string:
if there is a computation of the NFA
that accepts the string

i.e., all the input is consumed and the automaton
is in an accepting state
Example

$aa$ is accepted by the NFA:

```
0
q0  a
   q1  a
      q2
q0  a
   q3
```

“accept”

because this computation accepts $aa$

```
0
q0  a
   q1  a
      q2
```

“reject”

Rejection example

```
q0
  a
  q1  a
  q2
```

```
q0
  a
  q3
```

```
a
```
First Choice

\[ a \]

```
q0 \rightarrow a \quad q1 \quad a \quad q2
```

“reject”

```
q0 \rightarrow a \quad q1 \quad a \quad q2
```

```
q0 \rightarrow a \quad q1 \quad a \quad q2
```

```
q0 \rightarrow a \quad q1 \quad a \quad q2
```

```
q0 \rightarrow a \quad q1 \quad a \quad q2
```

```
q0 \rightarrow a \quad q1 \quad a \quad q2
```
An NFA rejects a string: if there is no computation of the NFA that accepts the string.

For each computation:

All the input is consumed and the automaton is in a non final state

OR

The input cannot be consumed
Example

\(a\) is rejected by the NFA:

```
q0 \rightarrow a \rightarrow q1 \rightarrow a \rightarrow q2
```

“reject”

```
q0 \rightarrow a \rightarrow q1 \rightarrow a \rightarrow q2
```

Rejection example

```
a  a  a
```

```
q0 \rightarrow a \rightarrow q1 \rightarrow a \rightarrow q2
```

```
q0 \rightarrow a \rightarrow q1 \rightarrow a \rightarrow q2
```

```
q0 \rightarrow a
```

```
q0 \rightarrow a
```

```
q3
```

```
q3
```
No transition: the automaton hangs
First Choice

```
Input cannot be consumed
```

Second Choice

```
```
Second Choice

\[
\begin{array}{c}
q_0 \\
\uparrow \quad \quad \quad \quad \quad a \\
q_1 \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad a \\
q_2 \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad a \\
q_3
\end{array}
\]

No transition: the automaton hangs
Second Choice

Input cannot be consumed

aaa is rejected by the NFA:

All possible computations lead to rejection
Language accepted: \[ L = \{ aa \} \]

Quiz