CSE 311: Foundations of Computing

Lecture 23: Finite State Machines

[Diagram of a finite state machine with states q0, q1, q2, q3 and transitions labeled with 0 and 1]
Last time: Relations & Composition

Let A and B be sets,
A **binary relation from** A **to** B is a subset of A × B

Let A be a set,
A **binary relation on** A is a subset of A × A
Last time: Directed Graphs

\[ G = (V, E) \]

\( V \) – vertices

\( E \) – edges, ordered pairs of vertices
The composition of relation $R$ and $S$, $R \circ S$ is the relation defined by:

$$R \circ S = \{ (a, c) \mid \exists b \text{ such that } (a,b) \in R \text{ and } (b,c) \in S \}$$
If $S = \{(2, 2), (2, 3), (3, 1)\}$ and $R = \{(1, 2), (2, 1), (1, 3)\}$
Compute $R \circ S$
Relational Composition using Digraphs

If $R = \{(1, 2), (2, 1), (1, 3)\}$ and $R = \{(1, 2), (2, 1), (1, 3)\}$

Compute $R \circ R$

$$(a, c) \in R \circ R = R^2 \text{ iff } \exists b \ ((a, b) \in R \land (b, c) \in R)$$

iff $\exists b$ such that $a$, $b$, $c$ is a path
Def: The **length** of a path in a graph is the number of edges in it (counting repetitions if edge used > once).

Elements of $R^0$ correspond to paths of length 0.
Elements of $R^1 = R$ are paths of length 1.
Elements of $R^2$ are paths of length 2.
...

Last time: Paths in Relations and Graphs
Def: The **length** of a path in a graph is the number of edges in it (counting repetitions if edge used > once).

Let $R$ be a relation on a set $A$.

There is a path of length $n$ from $a$ to $b$ in the digraph for $R$ if and only if $(a, b) \in R^n$.
Let $R$ be a relation on a set $A$. The **connectivity** relation $R^*$ consists of the pairs $(a,b)$ such that there is a path from $a$ to $b$ in $R$.

$$R^* = \bigcup_{k=0}^{\infty} R^k$$

Note: Rosen text uses the wrong definition of this quantity. What the text defines (ignoring $k=0$) is usually called $R^+$.
Let $R$ be a relation on $A$.

$R$ is **reflexive** iff $(a,a) \in R$ for every $a \in A$

- [Diagram showing reflexive property](image)

$R$ is **symmetric** iff $(a,b) \in R$ implies $(b,a) \in R$

- [Diagram showing symmetric property](image)

$R$ is **antisymmetric** iff $(a,b) \in R$ and $a \neq b$ implies $(b,a) \notin R$

- [Diagram showing antisymmetric property](image)

$R$ is **transitive** iff $(a,b) \in R$ and $(b,c) \in R$ implies $(a,c) \in R$

- [Diagram showing transitive property](image)
Add the **minimum possible** number of edges to make the relation transitive and reflexive.
Transitive-Reflexive Closure

Relation with the **minimum possible** number of **extra edges** to make the relation both transitive and reflexive.

The **transitive-reflexive closure** of a relation $R$ is the connectivity relation $R^*$
Let $A_1, A_2, \ldots, A_n$ be sets. An $n$-ary relation on these sets is a subset of $A_1 \times A_2 \times \cdots \times A_n$. 

**$n$-ary Relations**
<table>
<thead>
<tr>
<th>Student_Name</th>
<th>ID_Number</th>
<th>Office</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knuth</td>
<td>328012098</td>
<td>022</td>
<td>4.00</td>
</tr>
<tr>
<td>Von Neuman</td>
<td>481080220</td>
<td>555</td>
<td>3.78</td>
</tr>
<tr>
<td>Russell</td>
<td>238082388</td>
<td>022</td>
<td>3.85</td>
</tr>
<tr>
<td>Einstein</td>
<td>238001920</td>
<td>022</td>
<td>2.11</td>
</tr>
<tr>
<td>Newton</td>
<td>1727017</td>
<td>333</td>
<td>3.61</td>
</tr>
<tr>
<td>Karp</td>
<td>348882811</td>
<td>022</td>
<td>3.98</td>
</tr>
<tr>
<td>Bernoulli</td>
<td>2921938</td>
<td>022</td>
<td>3.21</td>
</tr>
</tbody>
</table>
AND NOW BACK TO OUR REGULARLY SCHEDULED PROGRAMMING
Selecting strings using labeled graphs as “machines”
Finite State Machines

The circles are called “states”
We’re only in a single state at any point in time...

The “double circle” means “the input is good if it ends here”

“Start here”

“If I get this symbol, follow the arrow...”
Which strings does this machine say are OK?
Which strings does this machine say are OK?

The set of all binary strings that end in 0
Finite State Machines

- **States**
- **Transitions on input symbols**
- **Start state and final states**
- The “language recognized” by the machine is the set of strings that reach a final state from the start

<table>
<thead>
<tr>
<th>Old State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$s_0$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_0$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_0$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
</tbody>
</table>
Finite State Machines

- Each machine designed for strings over some fixed alphabet $\Sigma$.

- Must have a transition defined from each state for every symbol in $\Sigma$.

<table>
<thead>
<tr>
<th>Old State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$s_0$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_0$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_0$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
</tbody>
</table>
What language does this machine recognize?

<table>
<thead>
<tr>
<th>Old State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₀</td>
<td>s₀</td>
<td>s₁</td>
</tr>
<tr>
<td>s₁</td>
<td>s₀</td>
<td>s₂</td>
</tr>
<tr>
<td>s₂</td>
<td>s₀</td>
<td>s₃</td>
</tr>
<tr>
<td>s₃</td>
<td>s₃</td>
<td>s₃</td>
</tr>
</tbody>
</table>
What language does this machine recognize?

The set of all binary strings that contain 111 or don’t end in 1

<table>
<thead>
<tr>
<th>Old State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$s_0$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_0$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_0$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
</tbody>
</table>
Applications of FSMs (a.k.a. Finite Automata)

• Implementation of regular expression matching in programs like grep
• Control structures for sequential logic in digital circuits
• Algorithms for communication and cache-coherence protocols
  – Each agent runs its own FSM
• Design specifications for reactive systems
  – Components are communicating FSMs
Applications of FSMs (a.k.a. Finite Automata)

• Formal verification of systems
  – Is an unsafe state reachable?

• Computer games
  – FSMs implement non-player characters

• Minimization algorithms for FSMs can be extended to more general models used in
  – Text prediction
  – Speech recognition
Strings over \( \{0, 1, 2\} \)

\( M_1 \): Strings with an even number of 2’s
Strings over \{0, 1, 2\}

$M_1$: Strings with an even number of 2’s
Strings over \( \{0, 1, 2\} \)

\( M_2: \) Strings where the sum of digits mod 3 is 0
boolean sumCongruentToZero(String str) {
    int sum = 0;
    for (int i = 0; i < str.length(); i++) {
        if (str.charAt(i) == '2')
            sum = (sum + 2) % 3;
        if (str.charAt(i) == '1')
            sum = (sum + 1) % 3;
    }
    return sum == 0;
}
Strings over \{0, 1, 2\}

\(M_2\): Strings where the sum of digits mod 3 is 0
Strings over \(\{0, 1, 2\}\)

\(M_2\): Strings where the sum of digits mod 3 is 0
FSM as abstraction of Java code

```java
boolean sumCongruentToZero(String str) {
    int sum = 0;  // state
    for (int i = 0; i < str.length(); i++) {
        if (str.charAt(i) == '2')
            sum = (sum + 2) % 3;
        if (str.charAt(i) == '1')
            sum = (sum + 1) % 3;
    }
    return sum == 0;
}
```

FSMs can model Java code with a finite number of fixed-size variables that makes one pass through input.
FSM to Java code

```java
int[][] TRANSITION = {...};

boolean sumCongruentToZero(String str) {
    int state = 0;
    for (int i = 0; i < str.length(); i++) {
        int d = str.charAt(i) - '0';
        state = TRANSITION[state][d];
    }
    return state == 0;
}
```
Strings over \{0, 1, 2\}

**M₁**: Strings with an even number of 2’s

**M₂**: Strings where the sum of digits mod 3 is 0
Strings over \{0,1,2\} w/ even number of 2’s AND mod 3 sum 0
Strings over \{0,1,2\} w/ even number of 2’s AND mod 3 sum 0
Strings over \{0,1,2\} w/ even number of 2’s OR mod 3 sum 0
Strings over \{0,1,2\} w/ even number of 2’s XOR mod 3 sum 0
The set of binary strings with a 1 in the 3\textsuperscript{rd} position from the start
The set of binary strings with a 1 in the 3\textsuperscript{rd} position from the start
The set of binary strings with a 1 in the 3rd position from the end
3 bit shift register  "Remember the last three bits"
The set of binary strings with a 1 in the 3\textsuperscript{rd} position from the end
The set of binary strings with a 1 in the 3\textsuperscript{rd} position from the end
The beginning versus the end
Adding Output to Finite State Machines

• So far we have considered finite state machines that just accept/reject strings
  – called “Deterministic Finite Automata” or DFAs

• Now we consider finite state machines with output
  – These are the kinds used as controllers
Vending Machine

Enter 15 cents in dimes or nickels
Press S or B for a candy bar
Basic transitions on N (nickel), D (dime), B (butterfinger), S (snickers)
Adding output to states: N – Nickel, S – Snickers, B – Butterfinger
Adding additional “unexpected” transitions to cover all symbols for each state