Finite State Machines

... The Simplest Model of Computation
What is a Finite State Machine?

- Mathematical model of Computation
- Abstract Machine
- Is in exactly one state at any given time
- Changes state based on input
- Surprisingly flexible
- Recognizes a Language

Practical examples:

- Vending machines
- Elevators
- Traffic signals
- Combination locks
- Antikythera mechanism
- Automatons

Are Robots FSMs? Why or why not?
FSM Characteristics

Limited Memory
- Small Computer
- Microcontroller

Finite (*It’s in the name!*)

Family of:
- Regular Languages
- Regular Expressions

Nodes = States
Edges = Transitions

A picture is worth a thousand words...
Formal Definition of a Finite State Machine

\[ M = (Q, \Sigma, \delta, q_0, F) \]

- \( Q \): Set of states (finite)
- \( \Sigma \): Alphabet of symbols (finite)
- \( \delta \): The transition function \( \delta : Q \times \Sigma \rightarrow Q \)
- \( q_0 \): The starting (initial) state \( q_0 \in Q \)
- \( F \): The set of “Accept” states \( F \subseteq Q \)
Formal Definition of a Finite State Machine

\[ M = (Q, \Sigma, \delta, q_0, F) \]

\[ Q = \{q_1, q_2, q_3\} \]

\[ \Sigma = \{0, 1\} \]

\[ \delta \]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_1</td>
<td>q_1</td>
<td>q_2</td>
</tr>
<tr>
<td>q_2</td>
<td>q_3</td>
<td>q_2</td>
</tr>
<tr>
<td>q_3</td>
<td>q_2</td>
<td>q_2</td>
</tr>
</tbody>
</table>

\[ q_0 = q_1 \]

\[ F = \{q_2\} \]

\[ M = (\{q_1, q_2, q_3\}, \{0, 1\}, \delta, q_1, \{q_2\}) \]
FSM Use #1: Generating Strings

1. Begin at starting state
2. Take transitions at random
   *Transitions are recorded, which is the string being generated*
3. End only on valid states

What is the set of strings that can be generated?

What Language will this FSM generate?
FSM Use #2: Accepting Strings

1. Begin at starting state
2. Start at the 1st symbol of the string
3. Follow transitions as determined by the symbol, 1 symbol per transition
4. Process ALL symbols in the string
5. Is the machine in a final state?

A string is either “accepted” or “rejected”
Other FSM Considerations

Empty strings
- \( \varepsilon \)
- Starting state is also an accept state

Empty Language
- \( \emptyset = {} \)
- There is no path from the starting state to any accept state

Important:
- \( \varepsilon \neq \emptyset \)
- \( [\varepsilon] \neq \emptyset \)

Dead states
- A state that exists as a “reject” state
- Often omitted from diagrams
- If an edge is omitted it is assumed to be a transition to the dead state
- Understood as being a sink node (no escape once reached)
Example

Construct a FSM that will not accept any string unless it has an even number of 0s and 1s, where \( \Sigma = \{0, 1\} \).

What is its complement?
Formal Definition of Computation

Let $M = (Q, \Sigma, \delta, q_0, F)$

Let $w_1w_2...w_n$ be a string $w$ where $w_i \in \Sigma$

$M$ accepts $w$ if there is a sequence of states $r_0, r_1, r_2...r_n$ in $Q$ such that:

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $0 \leq i < n$, and
3. $r_n \in F$

$M$ “recognizes” language $A$ if

$A = \{w \mid M \text{ accepts } w\}$

We now have a tool that we can use to understand Regular Languages!