Finite State Machines

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

As a Theoretical Model:
- Mathematical model of Computation
- No “memory”
- Is in exactly one state at any given time
- Changes state based on input
- Surprisingly flexible
- Goal: Recognize a Language

Practical examples:
- Vending machines
- Elevators
- Traffic signals
- Combination locks
- Antikythera mechanism
- Automatons

Why would you ever want to use a FSM?
**FSM Characteristics**

Often used in **limited memory** environments
- Small Computer
- Microcontroller

**Finite** *(It's in the name!)*

Family of:
- Regular Languages
- Regular Expressions

---

A picture is worth a thousand words...

**Nodes = States**
**Edges = Transitions**
Formal Definition of a Finite State Machine

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

- **Q**: Set of states (finite)
- **Σ**: Alphabet of symbols (finite)
- **δ**: The transition function \( \delta : Q \times \Sigma \rightarrow Q \)
- **q₀**: The starting (initial) state \( q₀ \in Q \)
- **F**: The set of “accept” states \( F \subseteq Q \)
Formal Definition of a Finite State Machine

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

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

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

\[ \delta = \begin{array}{c|cc}
q_1 & 0 & 1 \\
q_2 & q_3 & q_2 \\
q_3 & q_2 & q_2 \\
\end{array} \]

\[ q_0 = q_1 \]

\[ F = \{q_2\} \]
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

Give examples of strings that can be generated.

What **Language** will this FSM generate?
1. Begin at starting state
2. Start at the 1st symbol of the string
3. Follow transitions as determined by the symbol, exactly 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"

The machine "decides" if the input is valid or not.
Other FSM Considerations

Empty strings
- $\varepsilon$ is the empty string
- 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 an even number of 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 $A = \{w \mid M$ accepts $w\}$

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

What is this machine missing?

Memory.

We have defined a **Machine** (FSM) and a mode of operation.

- States
- Input
- Transitions
- Start & End

We have defined what constitutes a valid **Computation**.
Example FSM: Even \# of 0’s and even \# of 1’s.

How would you program this normally?

How was this technique represented in the FSM?

Is this possible for every variable? Why or why not?