Finite Automata & Regular Languages

Class of languages recognized by finite automata.

Abstractly, for a language recognized by a F.A.

YES/Accept if \( w \in L \)

NO/Reject if \( w \notin L \)

Input \( w \in \Sigma^*, |w| = n \)

Input is read only once.

Motivation - Controllers for sliding door, elevator, washing m/c etc.

- Constantly many "states";
- Change state according to input via simple rules.

Examples of F.A.

1. \( \Sigma = \{a, b\} \)

\[ q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_1 \]

\( q_1 \): start state, indicated by squiggly in-arrow.

\( q_2 \): accept state, " double circle.

There could be many (or no) accept states.
Given input say `baabba`, the FA starts in state `q_1`, scans input left to right, one symbol at a time and makes transitions accordingly. Generically

```
Current state → input symbol → Next state
```

E.g. on input `baabba`

```
q_1 → q_1 → q_2 → q_1 → q_1 → q_1 → q_2.
```

Since the final state `q_2` is an accept state, the F.A. is said to accept input `baabba`.

E.g. Inputs accepted: `a`, `abbaa`, `babbaba`,

Inputs rejected: `ε`, `abba`, `aaaa`, ...

If `M` denotes this F.A., the language recognized by it is denoted/defined as

```
L(M) = \{ w \in \Sigma^* | M \text{ accepts } w \}
```

I.e. running `M` on input `w` starting in state `q_1` makes `M` end up in an accept state.
Evidently, in this example,

\[ L(M) = \{w \in \{a, b\}^* \mid \text{w has an odd number of } a\text{'s.} \} \]

2. \( \Sigma = \{0, 1\} \).

\[ \begin{array}{c}
\overset{1}{q_1} \rightarrow \overset{0}{q_2} \rightarrow \overset{1}{q_3} \\
0 \rightarrow \rightarrow 1 \rightarrow 0 \rightarrow 1
\end{array} \]

\( M \).

\[ L(M) = \{w \in \{0, 1\}^* \mid \text{w has at least one } 1 \text{ and after the first occurrence of } 1 \text{ has an even number of } 0\text{'s.} \} \]

3. \( \Sigma = \{0, 1\} \).

\[ \begin{array}{c}
\overset{0}{q_1} \rightarrow \overset{1}{q_2} \rightarrow \overset{0}{q_3} \rightarrow \overset{0}{q_4} \rightarrow \overset{0}{q_5} \\
0 \rightarrow 1 \rightarrow 1 \rightarrow 1 \rightarrow 1
\end{array} \]

\( M \).

\[ L(M) = \{w \in \{0, 1\}^* \mid \text{w has } 0000 \text{ as a consecutive substring.} \} \]
Formal Definition of F.A.

Def A finite automaton \( M \) is a 5-tuple \( M = (Q, \Sigma, \delta, q_0, F) \) where:

- \( Q \) is a finite set of states.
- \( \Sigma \) is a finite alphabet.
- \( q_0 \in Q \) is the start state.
- \( F \subseteq Q \) is the subset of accept states.
- \( \delta : Q \times \Sigma \rightarrow Q \) is the transition function.

Note The transition function is interpreted as \( \delta(\text{current state}, \text{input symbol}) = \text{next state} \)

i.e.

The pictorial graphical representation before is referred to as the state diagram.
Example 2, in this formal notation, is:

- \( Q = \{ q_1, q_2, q_3 \} \)
- \( \Sigma = \{ 0, 1 \} \)
- \( F = \{ q_2 \} \)

<table>
<thead>
<tr>
<th>( \delta )</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_3 )</td>
</tr>
</tbody>
</table>

Alternately as:

\[ \delta(q_1, 0) = q_1, \quad \delta(q_2, 0) = q_3, \quad \delta(q_3, 0) = q_2 \]
\[ \delta(q_1, 1) = q_2, \quad \delta(q_2, 1) = q_2, \quad \delta(q_3, 1) = q_3. \]

Operation of F.A. (Informal) \((Q, \Sigma, \delta, q_1, F)\):

- Start in \( q_1 \).
- Scan input left to right, one symbol at a time, change state according to \( \delta \).
- When input is exhausted, accept iff the F.A. is in an accept state.

Note F.A. accepts the empty string \( \varepsilon \) iff \( q_1 \in F \).
Operation of F.A. (Formal). \((Q, \Sigma, \delta, q_0, F) = M\)

On input \(w = w_1w_2 \ldots w_n \in \Sigma^*\),
let \(r_1 = q_1\),
let \(r_{i+1} = \delta(q_i, w_i)\) for \(i = 1, 2, \ldots, n\),
Accept iff \(r_{n+1} \in F\).

\[
L(M) = \{ w \in \Sigma^* | M \text{ accepts } w \}\.
\]

Def: The class of languages accepted by finite automata is called the class of regular languages.

Alternately, a language \(L \subseteq \Sigma^*\) is called regular iff \(L = L(M)\) for some finite automaton \(M\) with alphabet \(\Sigma\).

Intuition: F.A. are a computational model with
- constant amount of memory and
- read once input.
Consider the FA

\[ \Sigma = \{a, b\} \]

It is not difficult to see that it recognizes

\[ L = \{ w \in \{a, b\}^* \mid \text{the first and the last symbol of } w \text{ is the same.} \} \]

(with understanding that \( \varepsilon, a, b \in L \)).

Intuitively the F.A. "remembers" the first symbol and depending on it, "branches out", and the two branches separately "check" that the last symbol matches the "remembered" first symbol.

Similarly, \( L = \{ w \mid \text{the first 5 symbols of } w \text{ match the last 5 symbols in reverse} \} \) is regular. One "remembers" 5 symbols now.
However, as we'll show later, the following languages are not regular:

- \( L = \{ w \in \{a,b\}^* \mid w \text{ is a palindrome (reads same forward and backward)} \} \)
- \( L = \{ a^n b^n \mid n \geq 0 \} \)

In both cases, the m/c would (intuitively) need super-constant amount of memory (\( \frac{|w|}{2} \) symbols to "remember" the first half of \( w \) and in the second example, \( \log_2 n \) bits to "count" to \( n \)).

**Designing F.A. More Illustrative Examples**

**Question** Design FA that recognizes

\[ L = \{ w \in \{0,1\}^* \mid w \text{ has } 001 \text{ as a consecutive substring} \} \]
- 4 states corresponding to prefixes of 001, ε, 0, 00, 001.
- Transitions capture "progress".

**Exercise** Construct FA that detects a consecutive substring 00101.

**Question** Design FA that recognizes

\[ L = \{ w \in \{a\}^* | \text{lw| is divisible by 5} \} \]

Note After reading prefix \( a^i \), the FA is in state \( q_{(i \mod 5)} \).

How would one recognize

\[ L = \{ w \in \{a\}^* | \text{lw| is either 2 or 4 mod 5} \} \]
Design FA that recognizes
\[ L = \{ w \in \{0,1\}^* \mid \text{w in binary represents an integer divisible by 5} \} \].

E.g., 1, 101, 1010, 00101, ... \in L.

1, 001, 1001, 1110, ... \notin L.

Observation: If the string \( w_1w_2...w_k \) represents integer \( N \) in binary, then
the string \( w_1w_2...w_k0 \) represents \( 2N \) in binary,
\( w_1w_2...w_k1 \) \( \Rightarrow \) \( 2N+1 \).

As before, after reading input \( w_1w_2...w_k \),
the FA, by design, will be in state \( q_{(N \mod 5)} \).

By above observation, a typical transition looks like
\[ q_{c_1} \xrightarrow{0} q_{(2i \mod 5)} \]
\[ q_{c_1} \xrightarrow{1} q_{(2i+1 \mod 5)} \]

Diagram:
- \( q_0 \) \( \xrightarrow{0} q_1 \) \( \xrightarrow{1} q_2 \)
- \( q_1 \) \( \xrightarrow{0} q_0 \) \( \xrightarrow{1} q_3 \)
- \( q_2 \) \( \xrightarrow{0} q_1 \) \( \xrightarrow{1} q_4 \)
- \( q_3 \) \( \xrightarrow{0} q_2 \) \( \xrightarrow{1} q_1 \)
- \( q_4 \) \( \xrightarrow{0} q_3 \) \( \xrightarrow{1} q_4 \)
Question: Design FA that recognizes $L = \{ w \in \{0,1\}^* \mid w$ has 1 in every odd position $\}.$

It is understood that $\varepsilon \in L.$
Regular Operators on Languages: $U, \cdot, \ast$

**Definition:** Let $A, B$ be languages (over some alphabet). The regular operators $U, \cdot, \ast$ are:

- $A \cup B = \{ x \mid x \in A \text{ or } x \in B \}$
- $A \cdot B = \{ xy \mid x \in A \text{ and } y \in B \}$
- $A^\ast = \{ x_1 x_2 \cdots x_k \mid x_i \in A \text{ for } 1 \leq i \leq k, k \geq 0 \}$

By definition $\varepsilon \in A^\ast$ (corresponds to $k=0$).

**Theorem:** The class of regular languages is closed under the regular operators. I.e. if $A, B$ are regular, so are $A \cup B, A \cdot B, A^\ast$.

**Proof (for $A \cup B$):** We'll show that $A \cup B$ is regular and postpone proofs for $A \cdot B, A^\ast$. 

[Diagrams of $M_A$, $M_B$]
let $M_A, M_B$ be F.A. that recognize $A, B$ respectively. We'll build a F.A. $M$ that recognizes $A \cup B$. For every input $x$ (e.g. $x = 001001110$), $M$ accepts $x$ $\iff$ Either $M_A$ accepts $x$ or $M_B$ accepts $x$.

**First attempt** Try "running" first $M_A$ on $x$ and then $M_B$ on $x$ and accept if either accepts.

However after running $M_A$ on $x$, one runs out of the input.

**Second attempt** So we try to "run" both $M_A$ and $M_B$ on $x$ simultaneously and accept if either accepts.

To run/simulate $M_A$, "remember" its state $q_i$.

To run $M_B$, \[ q_j \]
Hence, to run/simulate $M_A, M_B$ simultaneously, 
"remember" $(q_i, r_j)$.

Thus our F.A. $M$ will have states as pairs $(q_i, r_j)$ and it will simulate

$M_A$ on the first coordinate

and $M_B$ on second of the pair, and accept if either accepts.

---

Formal construction of $M$

Let

\[ M_A = (Q_1, \Sigma, \delta_1, q_1, F_1) \]

\[ M_B = (Q_2, \Sigma, \delta_2, r_1, F_2). \]

Then define

\[ M = (Q_1 \times Q_2, \Sigma, \delta, (q_1, r_1), F) \]

where

1. $\delta: (Q_1 \times Q_2) \times \Sigma \to Q_1 \times Q_2$ is defined as

\[ \delta((q, r), a) = (\delta_1(q, a), \delta_2(r, a)) \]

\[ \forall q \in Q_1, r \in Q_2, a \in \Sigma. \]

Let

2. $F = F_1 \times Q_2 \cup Q_1 \times F_2$. 
Exercise  Show that $M$ recognizes $A \cup B$.

Example  $\Sigma = \{0,1\}$. We know that

$A = \{ w \in \Sigma^* \mid |w| \text{ is even}\}$,

$B = \{ w \in \Sigma^* \mid w \text{ has 001 as consecutive substring}\}$

are both regular, with the corresponding FA. $M_A$, $M_B$ with 2 and 4 states resp.

Hence $A \cup B = \{ w \mid |w| \text{ is even or } w \text{ has 001 as cons. substr.}\}$

is recognized by a FA with $2 \times 4 = 8$ states.

It turns out that showing that $A \cdot B$, $A^*$ are regular (provided $A$, $B$ are) is more difficult. We first define a new variant of FA, called **Non-deterministic Finite Automata** (NFA), show that NFA are equivalent to FA (henceforth referred to as Deterministic FA (DFA)). It is then easy to construct NFAs for $A \cdot B$ and $A^*$!