Lecture Outline

Finite Automata
  Formal Definition
  DFA Computation
  Regular Operations
Finite Automata

The *finite state machine* or *finite automaton* is the simplest computational model of limited memory computers.

Example: an automatic door opener at a supermarket decides when to open or close the door, depending on the input provided by its sensors.
Finite Automata

The *finite state machine* or *finite automaton* is the simplest computational model of limited memory computers.

Example: an automatic door opener at a supermarket decides when to open or close the door, depending on the input provided by its sensors.

Finite automata are designed to solve *decision problems*, i.e., to decide whether a given input satisfies certain conditions.

Examples of decision problems:

▶ does a given string have an even number of 1’s;
▶ is the number of 0’s (in a given string) multiple of 4;
▶ does a given string end in 00.
Example: Door Opener

states: closed, open

input conditions: front, rear, both, neither

nonloop transitions:
  closed $\rightarrow$ open on front
  open $\rightarrow$ closed on neither
Example: Door Opener

states: closed, open
input conditions: front, rear, both, neither
nonloop transitions:
  closed → open on front
  open → closed on neither
Formal Definition

Definition
A deterministic finite automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- $Q$ is a finite set whose members are called states,
- $\Sigma$ is a finite alphabet whose members are called symbols,
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is the set of accept states (or final states).

DFA computation can be described informally using a tape, cells with symbols, and a finite-state control with a read head advancing over the input. Given an input string over $\Sigma$ (written on the input tape), an automaton reads its symbols one-by-one and changes its state (starting from $q_0$) according to $\delta$. The automaton “accepts” the input if its resulting state (after reading of the input string is complete) belongs to $F$; otherwise “rejects”.
What is what in Door Opener

Q: Is it a DFA?
What is what in Door Opener

This automaton is not a DFA since it supposedly operates infinitely long and thus there are no starting state $q_0$ and final states $F$ defined. However, it does correspond to some $Q$, $\Sigma$, $\delta$.

Recall that $Q$ is a finite set whose members are called states. What are the states of the Door Opener?
This automaton is not a DFA since it supposedly operates infinitely long and thus there are no starting state $q_0$ and final states $F$ defined. However, it does correspond to some $Q, \Sigma, \delta$.

$$Q = \{\text{CLOSED, OPEN}\}$$
What is what in Door Opener

This automaton is not a DFA since it supposedly operates infinitely long and thus there are no starting state $q_0$ and final states $F$ defined. However, it does correspond to some $Q$, $\Sigma$, $\delta$.

Recall that $\Sigma$ is a finite alphabet containing possible input symbols. What are they in the Door Opener?
What is what in Door Opener

This automaton is not a DFA since it supposedly operates infinitely long and thus there are no starting state $q_0$ and final states $F$ defined. However, it does correspond to some $Q$, $\Sigma$, $\delta$.

$$\Sigma = \{\text{FRONT, REAR, BOTH, NEITHER}\}$$
What is what in Door Opener

This automaton is not a DFA since it supposedly operates infinitely long and thus there are no starting state $q_0$ and final states $F$ defined. However, it does correspond to some $Q$, $\Sigma$, $\delta$.

Recall that transition function is a function $\delta : Q \times \Sigma \rightarrow Q$ defining how the automaton behaves.
What is what in Door Opener

This automaton is not a DFA since it supposedly operates infinitely long and thus there are no starting state $q_0$ and final states $F$ defined. However, it does correspond to some $Q$, $\Sigma$, $\delta$.  

<table>
<thead>
<tr>
<th></th>
<th>FRONT</th>
<th>REAR</th>
<th>BOTH</th>
<th>NEITHER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOSED</td>
<td>OPEN</td>
<td>CLOSED</td>
<td>CLOSED</td>
<td>CLOSED</td>
</tr>
<tr>
<td>OPEN</td>
<td>OPEN</td>
<td>OPEN</td>
<td>OPEN</td>
<td>CLOSED</td>
</tr>
</tbody>
</table>
State Diagram of a DFA

- Nodes encode the states (elements of $Q$).
- Directed edges, labelled with symbols (elements of $\Sigma$), encode $\delta$. Thus, each node has $|\Sigma|$ outgoing edges. Parallel edges can be combined into a single edge with multiple labels.
- An incoming edge from nowhere encodes the starting state.
- Nodes with double border encode accept states (elements of $F$).
Q: Draw a state diagram of a DFA $M_1$ with state set $Q = \{q_1, q_2, q_3\}$, alphabet $\Sigma = \{0, 1\}$, start state $q_1$, final state set $F = \{q_2\}$, and transition function $\delta$ given by the following table:

<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>$q_1$</td>
</tr>
</tbody>
</table>

We start with drawing nodes (labeled with elements of $Q$), marking the starting and accepting states:
Drawing a State Diagram

Q: Draw a state diagram of a DFA $M_1$ with state set $Q = \{q_1, q_2, q_3\}$, alphabet $\Sigma = \{0, 1\}$, start state $q_1$, final state set $F = \{q_2\}$, and transition function $\delta$ given by the following table:

<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>$q_1$</td>
</tr>
</tbody>
</table>

Then we draw outgoing edges (defined by $\delta$) for the first node ($q_1$ is this example):

```
\begin{center}
\begin{tikzpicture}
    \node[state] (q1) at (0,0) {$q_1$};
    \node[state, fill=white, fill opacity=0.5] (q2) at (1,0) {$q_2$};
    \node[state, fill=white, fill opacity=0.5] (q3) at (2,0) {$q_3$};
    \path[->] (q1) edge [loop below] node {0} (q1);
    \path[->] (q1) edge node {1} (q2);
\end{tikzpicture}
\end{center}
```
Q: Draw a state diagram of a DFA $M_1$ with state set $Q = \{q_1, q_2, q_3\}$, alphabet $\Sigma = \{0, 1\}$, start state $q_1$, final state set $F = \{q_2\}$, and transition function $\delta$ given by the following table:

<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>$q_1$</td>
</tr>
</tbody>
</table>

Similarly we draw outgoing edges for the other nodes and we are done:
Languages

Definition
Let $\Sigma$ be an alphabet. We let $\Sigma^*$ denote the set of all (finite) strings over $\Sigma$. A language over $\Sigma$ is any subset of $\Sigma^*$, i.e., any set of strings over $\Sigma$.

We use languages to encode decision problems. Given an input (string), is the answer “yes” or “no”? Strings for which the answer is “yes” are called yes-instances and the others are no-instances. The language corresponding to a decision problem is the set of strings encoding yes-instances.
Recognizing DFA

**Definition**
Let $M$ be a DFA with alphabet $\Sigma$ and let $A \subseteq \Sigma^*$ be a language over $\Sigma$. We say that $M$ recognizes $A$ iff $M$ accepts every string in $A$ and rejects every string in $\overline{A} = \Sigma^* \setminus A$. We let $L(M)$ denote the language recognized by $M$.

Thus recognizing a language means being able to distinguish membership from nonmembership in the language, thus solving the corresponding decision problem. Every DFA recognizes a unique language (consisting of the strings it accept).
Q: Design a DFA recognizing the language \( B = \{ x \in \{0, 1\}^* | |x| \geq 2 \text{ and the 1st symbol of } x \text{ equals the last symbol of } x \} \).

In order to design such a DFA we need:

- to “record” the first symbol of the input (using states as the only available sort of memory);
- to distinguish whether the current symbol (that will be the last one at some point) matches the recorded first symbol;
- (at the end) accept if it does, reject otherwise.
DFA Computation

We now define a DFA computation formally.

Definition
Let \( M = (Q, \Sigma, \delta, q_0, F) \) be a DFA and let \( w \in \Sigma^* \) be a string over \( \Sigma \). Suppose \( w = w_1w_2 \cdots w_n \) where \( w_i \in \Sigma \) for all \( 1 \leq i \leq n \). The computation of \( M \) on input \( w \) is the unique sequence of states \((s_0, s_1, \ldots, s_n)\) where

- each \( s_i \in Q \),
- \( s_0 = q_0 \), the start state, and
- \( s_i = \delta(s_{i-1}, w_i) \) for all \( 1 \leq i \leq n \).

The computation \((s_0, \ldots, s_n)\) is accepting if \( s_n \in F \), and is rejecting otherwise. If the former holds, we say that \( M \) accepts \( w \), and if the latter holds, we say that \( M \) rejects \( w \).

Thus \( s_0, s_1, \ldots \) is the sequence of states that \( M \) goes through while reading \( w \) from left to right, starting with the start state.
Examples

Example: DFA that recognizes multiples of 3 in unary, binary.
Example: DFA that accepts strings over \{a, b, c\} containing \textit{abacab} as a substring. (Idea is useful for text search.)
Example: Strings over \{0, 1\} with an even number of 1s. Strings over \{0, 1\} with an even number of 1s or an odd number of 0s.

Definition

A language $A \subseteq \Sigma^*$ is \textit{regular} iff some DFA recognizes it, i.e., $A = L(M)$ for some DFA $M$. 
Regular Operations

Let $A$ and $B$ be two languages. We define the following regular operations:

Union: $A \cup B = \{x \mid x \in A \lor x \in B\}$

Concatenation: $A \circ B = \{xy \mid x \in A \land y \in B\}$

Star: $A^* = \{x_1x_2 \ldots x_k \mid k \in \mathbb{Z} \land k \geq 0 \land \forall i = 1, 2, \ldots, k, \; x_i \in A\}$

Theorem

*If $A$ and $B$ are regular languages then so are $A \cup B$ and $A \circ B$. In other words, the class of regular languages is closed under the union and concatenation operations.*

Read proofs in Sipser pp.44-47.