Finite State Automata (Finite State Machines)

Deterministic Finite State Automata (FSA)
(Chapter 2, Section 2.1)

1. Introduction

FSA - the simplest model of a computing device - *a language recognition device*
Input: strings of symbols, delivered on a tape.

The operation of an FSA proceeds at discrete steps.
At each step of its operation FSA can read one symbol from the tape and can move ahead to read the next symbol. The purpose of FSA is to determine whether a given string written on the tape belongs to a particular language or not. Thus the set of inputs recognized at each particular step is fixed in the description of the particular FSA.

**Informally, FSA is described by**

a. a set of internal states. At each step of its operation FSA is at some internal state. Whether the current symbol on the tape is recognized or not depends on its current internal state. Thus we have the next part of the FSA description:

b. a set of symbols that can be recognized at each internal state of the FSA. This description is given by a *transition function* - telling the FSA what would be its next state when a given symbol is recognized.

c. The set of all symbols that can be recognized constitute the *alphabet*

d. There should be one initial state to start from

e. There should be also one or more final states to end the recognition process.

Here is an example of the laughing FSA - it recognizes the strings

ha!
ha ha! ha ha ha...ha!
q0, q1, q2, q3 - states of the FSA

FSA can be represented by graphs (drawing the picture of the graph), or by tables:

<table>
<thead>
<tr>
<th></th>
<th>h</th>
<th>a</th>
<th>!</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>q1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>q1</td>
<td>-</td>
<td>q2</td>
<td>-</td>
</tr>
<tr>
<td>q2</td>
<td>q1</td>
<td>-</td>
<td>q3</td>
</tr>
<tr>
<td>q3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Initial state: q0,
Final state q3.

At state q0 FSA accepts 'h' and enters q1
At state q1 FSA accepts 'a' and enters q2
At state q2 FSA can accept 'h' and in that case it enters q1,
or it can accept '!' and in that case it enters q3, which is the final state.

FSA recognizes a string (accepts a string) if after the string has been read the FSA is at a final state. Otherwise the string is not recognized by the FSA.

A language is recognized (accepted) by an FSA if all its strings are recognized by the FSA.

Two FSAs are equivalent if they recognize one and the same language.

2. Formal Definitions

A deterministic finite automaton is a quintuple $M = (K, \Sigma, \delta, s, F)$, where

- $K$ is a finite set of states
- $\Sigma$ is an alphabet
- $s \in K$ is the initial state
- $F \subseteq K$ is the set of final states
- $\delta$ is the transition function from $K \times \Sigma$ to $K$

If $M$ is in state $q \in K$ and the symbol read is $a \in \Sigma$, then $\delta(q, a) \in K$ is the uniquely determined next state of $M$ on reading $a$. 
Important:

1. Why deterministic automaton?
   Because for a given state and a given input symbol there is not more than one state to enter next. If there were two or more possible next states, the automaton is called **nondeterministic**.

2. What if $a \in \sum$, however for some $q$ the function $\delta(q, a)$ is not given?
   In that case the FSA is called **incompletely defined**, i.e. the domain of the transition function is a subset of $K \times \sum: D(\delta) \subseteq K \times \sum$

   An incompletely defined FSA can be extended to a completely defined FSA by introducing a new state $q'$, which is not final, and adding $\delta(q', a) = q'$ for each pair $(q, a)$ for which $\delta(q, a)$ has not been initially defined.

   $\delta(q', a) = q'$ for each $a \in \sum$.

   The new FSA will recognize all strings recognized by the incompletely defined FSA. If a symbol is read for which the transition function of the initial FSA was not specified, the new FSA will enter the state $q'$ and will stay there till the end of the string. Since the new state $q'$ is not a final state, at the end the FSA will not be at a final state, which means that the string is not recognized.

**Example:**
Making the laughing machine a complete FSA:

<table>
<thead>
<tr>
<th></th>
<th>h</th>
<th>a</th>
<th>!</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>q1</td>
<td>q4</td>
<td>q4</td>
</tr>
<tr>
<td>q1</td>
<td>q4</td>
<td>q2</td>
<td>q4</td>
</tr>
<tr>
<td>q2</td>
<td>q1</td>
<td>q4</td>
<td>q3</td>
</tr>
<tr>
<td>q3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>q4</td>
<td>q4</td>
<td>q4</td>
<td>q4</td>
</tr>
</tbody>
</table>

Initial state: $q0$.
Final state $q3$.

**q4:** Additional state to make the FSA completely defined:
3. Computation and Configurations

**Definition:** *computation* by an FSA on an input string is a *sequence of configurations* that represent the status of the machine at successive moments.

A configuration is determined by the current state and the unread part of the string. Hence a configuration is any element \((q, w)\) in \(K \times \sum^*\).

**Definition:** If \((q, w)\) and \((q', w')\) are two configurations, then \((q, w)\) *yields* \((q', w')\) in one *step* if and only if \(w = aw'\) for some \(a \in \sum\), and \(\delta(q, a) = q'\).

This is written as:

\[(q, w) \vdash_M (q', w')\]

The transitive closure of \(\vdash_M\) is denoted by \(\vdash_M^*\).

\[(q, w) \vdash_M^* (q', w')\] means that \((q, w)\) *yields* \((q', w')\) after some number of steps.

**Definition:** A string \(w \in \sum^*\) is said to be accepted by \(M\) iff there is a state \(q \in F\) such that \((s, w) \vdash_M^* (q, \epsilon)\).

**Definition:** The language accepted by \(M\), \(L(M)\), is the set of all strings accepted by \(M\).
Examples

1. An FSA that accepts even number of b's

Initial state: q0
Final state: q0

Input aaabba

\[
(q0, aaabba) \xrightarrow{M} (q0, aabba) \\
(q0, aabba) \xrightarrow{M} (q0, abba) \\
(q0, abba) \xrightarrow{M} (q0, bba) \\
(q0, bba) \xrightarrow{M} (q1, ba) \\
(q1, ba) \xrightarrow{M} (q0, a) \\
(q0, a) \xrightarrow{M} (q0, e) \]

Since q0 is a final state, the string was accepted

Input aaabbba

\[
(q0, aaabbba) \xrightarrow{M} (q0, aabbbba) \\
(q0, aabbbba) \xrightarrow{M} (q0, abbbba) \\
(q0, abbbba) \xrightarrow{M} (q0, bbba) \\
(q0, bbba) \xrightarrow{M} (q1, bba) \\
(q1, bba) \xrightarrow{M} (q0, ba) \\
(q0, ba) \xrightarrow{M} (q1, a) \\
(q1, a) \xrightarrow{M} (q1, e) \]

q1 is not a final state, so the string is not accepted
2. An FSA that accepts strings not containing three consecutive b's

Initial state: q0
Final states: q0, q1, q2

Input: baabb

(q0, baabb) ⊢_M (q1, aabb)
⊢_M (q0, abb)
⊢_M (q1, b)
⊢_M (q2, e) Since q0 is a final state, the string is accepted

Input: abbaabbba

(q0, abbaabbba) ⊢_M (q0, bbaabbba)
⊢_M (q1, baabbba)
⊢_M (q2, aabbbba)
⊢_M (q0, abbba)
⊢_M (q0, bbba)
⊢_M (q1, bba)
⊢_M (q2, ba)
⊢_M (q3, ba)
⊢_M (q3, a)
⊢_M (q3, e) Since q3 is not a final state, the string is not accepted