Finite State Machines

In this lecture we consider certain very simple machines which are capable of recognizing the strings of a regular language. That is, for each regular language we want to construct a machine that can somehow read a string (from left to right) and then tell us whether or not that string is in the language.

These machines have a finite number of states and are called finite automata or finite state machines.

Deterministic finite automata

We begin with a particularly simple type of machine called a deterministic finite automaton or DFA for short. We can draw such a machine using circles to indicate the states and arrows labelled with letters from some alphabet to indicate transitions from state to state.
Some states, called **accepting states**, are represented by double circles.

There is also an unlabelled arrow indicating the state at which the machine begins, called the **initial state** or **start state**.

The idea is that the machine begins in the initial state and reads a string of symbols (from its left hand end) and changes state according to its current state and the symbol just read. If the machine ends up in an accepting state, it **accepts** the string, otherwise it **rejects** it.
The precise definition of a deterministic finite automaton or DFA is that it consists of:

- An alphabet $\Sigma$.

- A finite set $S$ of states.

- An initial state.

- A set $A \subseteq S$ of accepting states.

- A transition function $f : S \times \Sigma \to S$.

We draw the diagram of a DFA by drawing a circle for each state, and we use double circles to mark the accepting states. Then for each state $A \in S$ and each symbol $x \in \Sigma$ we draw an arrow labelled $x$ from $A$ to $f(A, x)$. The initial state is marked by an unlabelled arrow leading to it.

The machine is called deterministic because given any state $A$ and any symbol $x$, the next state is completely determined by $A$ and $x$.

For our convenience, in drawing diagrams we sometimes label an arrow with more than one symbol. This is an abbreviation for several arrows, each with a single symbol.
Dead-end states

A **dead-end** state is one from which it is impossible to get to an accepting state, no matter what the input. Sometimes we omit dead-end states from our diagrams. After we remove dead-end states there is at most one arrow with a given label leaving each state.

Inaccessible states

An **inaccessible** state is one which cannot be reached by following arrows from the initial state. As with dead-end states, it is often convenient to omit them from our diagrams.
Paths

Let $M$ be a DFA. Then a path in the diagram of $M$ is a sequence of arrows $a_0 a_1 \ldots a_k$ such that the state that $a_{i-1}$ leads to is the same state that $a_i$ leaves from, for $i = 1, 2, \ldots, k$. If arrow $a_i$ carries the label $x_i$, then the string $x_0 x_1 \ldots x_k$ is the label of the path.

Accepted strings

A string is accepted by $M$ if there is a path beginning at the initial state and ending at an accepting state whose label is the given string.

The language of a machine

The strings accepted by a DFA $M$ form a language denoted by $L(M)$. We call this the language accepted by $M$. 
Example Consider the DFA $M$ with

- The alphabet $\Sigma = \{a, b\}$.

- The finite set $S = \{s_1, s_2, s_3, s_4, s_5\}$ of states.

- The initial state $s_1$.

- The set $A = \{s_4\} \subseteq S$ of accepting states.

- The transition function given by the following table in which the labels on the rows are the states, the labels on the columns are the input symbols and the entries indicate the next state.

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$s_2$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_2$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_4$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$s_4$</td>
<td>$s_5$</td>
<td>$s_5$</td>
</tr>
<tr>
<td>$s_5$</td>
<td>$s_5$</td>
<td>$s_5$</td>
</tr>
</tbody>
</table>
The example on the previous slide has the following diagram

Here is the description you would use for the `fsm` program, omitting the dead-end state.

The first line is the number of states, the second line is the list of accepting states and the remaining lines give the transitions. States are labelled 1, 2, … and the initial state is always labelled 1.

```
4
4
1 a 2
1 b 3
2 a 2
2 b 3
3 a 4
3 b 2
```