CHAPTER 10

Automata, Grammars and Languages

10.1. Finite State Machines

10.1.1. Finite-State Machines. Combinatorial circuits have no memory or internal states, their output depends only on the current values of their inputs. Finite state machines on the other hand have internal states, so their output may depend not only on its current inputs but also on the past history of those inputs.

A *finite-state machine* consists of the following:

1. A finite set of *states* $\mathcal{S}$.
2. A finite set of *input symbols* $\mathcal{I}$.
3. A finite set of *output symbols* $\mathcal{O}$.
4. A *next-state* or *transition function* $f : \mathcal{S} \times \mathcal{I} \to \mathcal{S}$.
5. An *output function* $g : \mathcal{S} \times \mathcal{I} \to \mathcal{O}$.
6. An *initial state* $\sigma \in \mathcal{S}$.

We represent the machine $M = (\mathcal{S}, \mathcal{I}, \mathcal{O}, f, g, \sigma)$

Example: We describe a finite state machine with two input symbols $\mathcal{I} = \{a, b\}$ and two output symbols $\mathcal{O} = \{0, 1\}$ that accepts any string from $\mathcal{I}^*$ and outputs as many 1's as a's there are at the beginning of the string, then it outputs only 0's. The internal states are $\mathcal{S} = \{\sigma_0, \sigma_1\}$, where σ_0 is the initial state—we interpret it as not having seeing any "b" yet; then the machine will switch to σ_1 as soon as the first "b" arrives. The next-state and output functions are as follows:

	f		g	
$\mathcal{I}$ / $\mathcal{S}$	a	b	a	b
σ_0	σ_0	σ_1	1	0
σ_1	σ_1	σ_1	0	0

This finite-state machine also can be represented with the following *transition diagram*:

$$\begin{tikzcd}
\text{start} \arrow[r] & \sigma_0 \arrow[loop above, "a/1"] \arrow[r, "b/0"] & \sigma_1 \arrow[loop above, "a/0"] \arrow[loop right, "b/0"]
\end{tikzcd}$$

The vertices of the diagram are the states. If in state σ an input i causes the machine to output o and go to state σ' then we draw an arrow from σ to σ' labeled i/o or i, o.

Example: The following example is similar to the previous one but the machine outputs 1 only after a change of input symbol, otherwise it outputs 0:

$$\begin{tikzcd}
\text{start} \arrow[r] & \sigma_0 \arrow[dl, "a/0"'] \arrow[dr, "b/0"] & \\
\sigma_1 \arrow[loop left, "a/0"] \arrow[rr, bend left, "b/1"] & & \sigma_2 \arrow[ll, bend left, "a/1"] \arrow[loop right, "b/0"]
\end{tikzcd}$$

Example: A Serial-Adder. A serial adder accepts two bits and outputs its sum. So the input set is $\mathcal{I} = \{00, 01, 10, 11\}$. The output set is $\mathcal{O} = \{0, 1\}$. The set of states is $\mathcal{S} = \{NC, C\}$, which stands for "no carry" and "carry" respectively. The transition diagram is the following:

$$\begin{tikzcd}
\text{start} \arrow[r] & NC \arrow[loop above, "00/0"] \arrow[loop above, "01/1"] \arrow[loop below, "10/1"] \arrow[r, bend left, "11/0"] & C \arrow[l, bend left, "00/1"] \arrow[loop above, "01/0"] \arrow[loop right, "10/0"] \arrow[loop below, "11/1"]
\end{tikzcd}$$

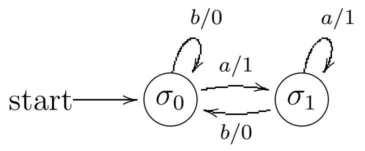
10.1.2. Finite-State Automata. A *finite-state automaton* is similar to a finite-state machine but with no output, and with a set of states called *accepting* or *final states*. More specifically, finite-state automaton consists of:

1. A finite set of *states* $\mathcal{S}$.
2. A finite set of *input symbols* $\mathcal{I}$.
3. A *next-state* or *transition function* $f : \mathcal{S} \times \mathcal{I} \to \mathcal{S}$.

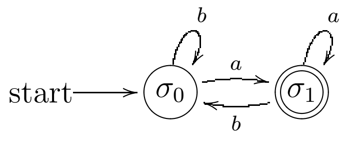
4. An *initial state* $\sigma \in \mathcal{S}$.
5. A subset $\mathcal{F} \subseteq \mathcal{S}$ of *accepting* or *final states*.

We represent the automaton $A = (\mathcal{S}, \mathcal{I}, f, \sigma, \mathcal{F})$. We say that an automaton *accepts* or *recognizes* a given string of input symbols if that strings takes the automaton from the start state to a final state.

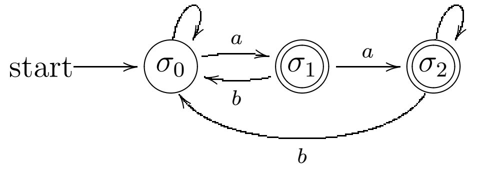
Example: The following transition diagrams represent an automaton accepting any string of a's and b's ending with an a. The first diagram uses the same scheme as with finite-state machines, with 1 representing "accept" or "recognize", and "0" representing "not accept":



The second kind of diagram omits the outputs and represents the accepting states with double circles:



Two finite-state automata that accept exactly the same set of strings are said to be *equivalent*. For instance the following automaton also accepts precisely strings of a's abd b's that end with an a, so it is equivalent to the automaton shown above:



Example: The following automaton accepts strings of a's and b's with exactly an even number of a's:

$$\begin{tikzcd}
\text{start} \arrow[r] & E \arrow[loop above, "b"] \arrow[r, bend left, "a"] & O \arrow[loop above, "b"] \arrow[l, bend left, "a"]
\end{tikzcd}$$

Example: The following automaton accepts strings starting with one a followed by any number of b's:

$$\begin{tikzcd}
\text{start} \arrow[r] & \sigma_0 \arrow[r, "a"] \arrow[rr, bend right, "b"'] & \sigma_1 \arrow[loop above, "b"] \arrow[r, "a"] & \sigma_2 \arrow[loop above, "a"] \arrow[loop right, "b"]
\end{tikzcd}$$

Example: The following automaton accepts strings ending with aba:

$$\begin{tikzcd}
\text{start} \arrow[r] & \sigma_0 \arrow[loop below, "b"] \arrow[r, "a"] & \sigma_1 \arrow[loop below, "a"] \arrow[r, "b"] & \sigma_2 \arrow[ll, bend right, "b"'] \arrow[r, bend left, "a"] & \sigma_3 \arrow[l, bend left, "b"] \arrow[ll, bend left, "a"]
\end{tikzcd}$$