Lecture 3

Finite Automata and Regular Sets

States and Transitions

Intuitively, a *state* of a system is an instantaneous description of that system, a snapshot of reality frozen in time. A state gives all relevant information necessary to determine how the system can evolve from that point on. *Transitions* are changes of state; they can happen spontaneously or in response to external inputs.

We assume that state transitions are instantaneous. This is a mathematical abstraction. In reality, transitions usually take time. Clock cycles in digital computers enforce this abstraction and allow us to treat computers as digital instead of analog devices.

There are innumerable examples of state transition systems in the real world: electronic circuits, digital watches, elevators, Rubik's cube ($54!/9!^6$ states and twelve transitions, not counting peeling the little sticky squares off), the game of Life (2^k states on a screen with k cells, one transition).

A system that consists of only finitely many states and transitions among them is called a *finite-state transition system*. We model these abstractly by a mathematical model called a *finite automaton*.

Finite Automata

Formally, a *deterministic finite automaton* (DFA) is a structure

$$M = (Q, \Sigma, \delta, s, F)$$

where:

- Q is a finite set; elements of Q are called *states*
- Σ is a finite set, the *input alphabet*
- $\delta : Q \times \Sigma \to Q$ is the *transition function* (recall $Q \times \Sigma$ is the set of ordered pairs $\{(q, a) \mid q \in Q \text{ and } a \in \Sigma\}$). Intuitively, δ is a function that tells which state to move to in response to an input: if M is in state q and sees input a, it moves to state $\delta(q, a)$.
- $s \in Q$ is the *start state*
- F is a subset of Q; elements of F are called *accept* or *final states*.

When you specify a finite automaton, you must give all five parts. Automata may be specified in this set theoretic form, or as a transition diagram or table as in the example below.

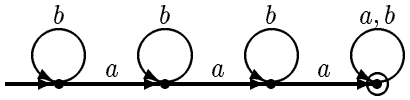
Example 3.1 Here is an example of a simple four-state finite automaton. We'll take the set of states to be $\{0, 1, 2, 3\}$, the input alphabet to be $\{a, b\}$, the start state to be 0, the set of accept states to be $\{3\}$, and the transition function to be

$$\begin{aligned}
&\delta(0, a) = 1 \\
&\delta(1, a) = 2 \\
&\delta(2, a) = \delta(3, a) = 3 \\
&\delta(q, b) = q\ , \quad q \in \{0, 1, 2, 3\}\ .
\end{aligned}$$

All parts of the automaton are completely specified. We can also specify the automaton by means of a table

		a	b
$\to$	0	1	0
	1	2	1
	2	3	2
	$3F$	3	3

or transition diagram



The final states are indicated by an F in the table and by a circle in the transition diagram. In both, the start state is indicated by $\rightarrow$. The states in the transition diagram from left to right correspond to the states $0, 1, 2, 3$ in the table. One advantage of transition diagrams is that you don't have to name the states. □

Another convenient representation of finite automata is transition matrices; see Miscellaneous Exercise 7.

Informally, here is how a finite automaton operates. An input can be any string $x \in \Sigma^*$. Put a pebble down on the start state s. Scan the input string x from left to right, one symbol at a time, moving the pebble according to δ: if the next symbol of x is b and the pebble is on state q, move the pebble to $\delta(q, b)$. When we come to the end of the input string, the pebble is on some state p. The string x is said to be *accepted* by the machine M if $p \in F$, *rejected* if $p \notin F$. There is no formal mechanism for scanning or moving the pebble; these are just intuitive devices.

For example, the automaton of Example 3.1, starting in its start state 0, will be in state 3 after scanning the input string *baabbaab*, so that string is accepted; whereas it will be in state 2 after scanning the string *babbbab*, so that string is rejected. For this automaton, a moment's thought reveals that when scanning any input string, the automaton will be in state 0 if it has seen no a's, 1 if it has seen one a, 2 if it has seen two a's, and 3 if it has seen three or more a's.

This is how we do formally what we just described informally above. We first define a function

$$\widehat{\delta} : Q \times \Sigma^* \rightarrow Q$$

from δ by induction on the length of x:

$$\widehat{\delta}(q, \epsilon) \stackrel{\text{def}}{=} q \tag{3.1}$$

$$\widehat{\delta}(q, xa) \stackrel{\text{def}}{=} \delta(\widehat{\delta}(q, x), a) \tag{3.2}$$

The function $\widehat{\delta}$ maps a state q and a string x to a new state $\widehat{\delta}(q, x)$. Intuitively, $\widehat{\delta}$ is the multi-step version of δ. The state $\widehat{\delta}(q, x)$ is the state M ends up in when started in state q and fed the input x, moving in response to each symbol of x according to δ. Equation (3.1) is the basis of the inductive definition; it says that the machine doesn't move anywhere under the null input. Equation (3.2) is the induction step; it says that the state reachable

from q under input string xa is the state reachable from p under input symbol a, where p is the state reachable from q under input string x.

Note that the second argument to $\widehat{\delta}$ can be any string in Σ^*, not just a string of length one as with δ; but $\widehat{\delta}$ and δ agree on strings of length one:

$$\begin{aligned}\widehat{\delta}(q,a) &= \widehat{\delta}(q,\epsilon a) && \text{since } a = \epsilon a\\ &= \delta(\widehat{\delta}(q,\epsilon),a) && \text{by (3.2), taking } x = \epsilon\\ &= \delta(q,a) && \text{by (3.1).}\end{aligned}$$

Formally, a string x is said to be *accepted* by the automaton M if

$$\widehat{\delta}(s,x) \in F$$

and *rejected* by the automaton M if

$$\widehat{\delta}(s,x) \notin F$$

where s is the start state and F is the set of accept states. This captures formally the intuitive notion of acceptance and rejection described above.

The *set* or *language accepted by* M is the set of all strings accepted by M, and is denoted $L(M)$:

$$L(M) \stackrel{\text{def}}{=} \{x \in \Sigma^* \mid \widehat{\delta}(s,x) \in F\} .$$

A subset $A \subseteq \Sigma^*$ is said to be *regular* if $A = L(M)$ for some finite automaton M. The set of strings accepted by the automaton of Example 3.1 is the set

$$\{x \in \{a,b\}^* \mid x \text{ contains at least three } a\text{'s}\} ,$$

so this is a regular set.

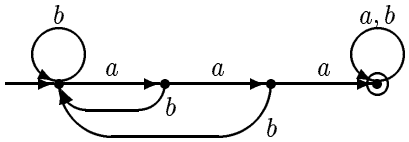
Here is another example of a regular set and a finite automaton accepting it.

Example 3.2 Consider the set

$$\begin{aligned}&\{xaaay \mid x,y \in \{a,b\}^*\}\\ &= \{x \in \{a,b\}^* \mid x \text{ contains a substring of three consecutive } a\text{'s}\} .\end{aligned}$$

For example, *baabaaaab* is in the set and should be accepted, whereas *babbabab* is not in the set and should be rejected (because the three a's are not consecutive). Here is an automaton for this set, specified in both table and transition diagram form:

		a	b
$\rightarrow$	0	1	0
	1	2	0
	2	3	0
	$3F$	3	3



□

The idea here is that you use the states to count the number of consecutive a's you have seen. If you haven't seen three a's in a row and you see a b, you must go back to the start. Once you have seen three a's in a row, though, you stay in the accept state.