2 Finite Automata

2.1 DETERMINISTIC FINITE AUTOMATA

This book is about mathematical models of computers and algorithms. In this and the next two chapters we shall define increasingly powerful models of computation, more and more sophisticated devices for accepting and generating languages. Seen in the light of the whole development of the book, this chapter will seem a rather humble beginning: Here we take up a severely restricted model of an actual computer called a **finite automaton,**[†] or **finite-state machine**. The finite automaton shares with a real computer the fact that it has a "central processing unit" of fixed, finite capacity. It receives its input as a string, delivered to it on an input tape. It delivers no output at all, except an indication of whether or not the input is considered acceptable. It is, in other words, a language recognition device, as described at the end of Chapter 1. What makes the finite automaton such a restricted model of real computers is the *complete absence of memory* outside its fixed central processor.

Such a simple computational model might at first be considered too trivial to merit serious study: of what use is a computer with no memory? But a finite automaton is not really without memory; it simply has a memory capacity that is fixed "at the factory" and cannot thereafter be expanded. It can be argued that the memory capacity of any computer is limited —by budget constraints, by physical limits, ultimately by the size of the universe. Whether the best mathematical model for a computer is one that has finite memory or one that has unbounded memory is an interesting philosophical question; we shall study both kinds of models, starting with the finite one and later dwelling much more

[†] An *automaton* (pronounced: o-to-ma-ton, plural: *automata*) is a machine designed to respond to encoded instructions; a robot.

on the unbounded one.

Even if one thinks, as we do, that the correct way to model computers and algorithms is in terms of an unbounded memory capacity, we should first be sure that the theory of finite automata is well understood. It turns out that their theory is rich and elegant, and when we understand it we shall be in a better position to appreciate exactly what the addition of auxiliary memory accomplishes in the way of added computational power.

A further reason for studying finite automata is their applicability to the design of several common types of computer algorithms and programs. For example, the *lexical analysis* phase of a compiler (in which program units such as 'begin' and '+' are identified) is often based on the simulation of a finite automaton. Also, the problem of finding an occurrence of a string within another —for example, whether any of the strings *air, water, earth,* and *fire* occur in the text of *Elements of the Theory of Computation*[†]— can also be solved efficiently by methods originating from the theory of finite automata.

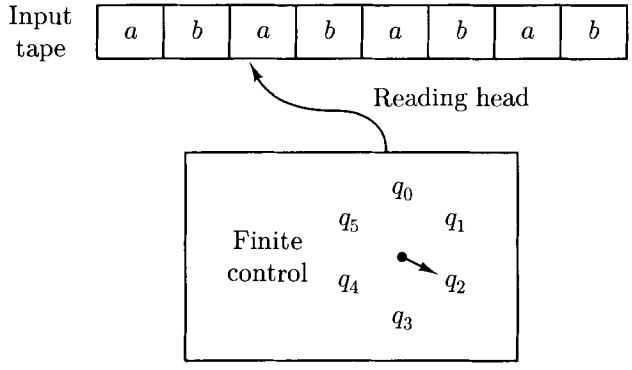


Figure 2-1

Let us now describe the operation of a finite automaton in more detail. Strings are fed into the device by means of an **input tape**, which is divided into squares, with one symbol inscribed in each tape square (see Figure 2-1). The main part of the machine itself is a "black box" with innards that can be, at any specified moment, in one of a finite number of distinct internal **states**. This black box —called the **finite control**— can sense what symbol is written at any position on the input tape by means of a movable **reading head**. Initially, the reading head is placed at the leftmost square of the tape and the finite control is set in a designated **initial state**. At regular intervals the automaton reads one symbol from the input tape and then enters a new state *that depends only on the*

[†] All four of them do, three of them outside this page.

current state and the symbol just read —this is why we shall call this device a *deterministic* finite automaton, to be contrasted to the *nondeterministic* version introduced in the next section. After reading an input symbol, the reading head moves one square to the right on the input tape so that on the next move it will read the symbol in the next tape square. This process is repeated again and again; a symbol is read, the reading head moves to the right, and the state of the finite control changes. Eventually the reading head reaches the end of the input string. The automaton then indicates its approval or disapproval of what it has read by the state it is in at the end: if it winds up in one of a set of **final states** the input string is considered to be **accepted**. The **language accepted** by the machine is the set of strings it accepts.

When this informal account is boiled down to its mathematical essentials, the following formal definition results.

Definition 2.1.1: A **deterministic finite automaton** is a quintuple $M = (K, \Sigma, \delta, s, F)$ where

K is a finite set of **states**,
Σ is an alphabet,
$s \in K$ is the **initial state**,
$F \subseteq K$ is the set of **final states**, and
δ, the **transition function**, is a function from $K \times \Sigma$ to K.

The rules according to which the automaton M picks its next state are encoded into the transition function. Thus if M is in state $q \in K$ and the symbol read from the input tape is $a \in \Sigma$, then $\delta(q, a) \in K$ is the uniquely determined state to which K passes.

Having formalized the basic structure of a deterministic finite automaton, we must next render into mathematical terms the notion of a *computation* by an automaton on an input string. This will be, roughly speaking, a sequence of **configurations** that represent the status of the machine (the finite control, reading head, and input tape) at successive moments. But since a deterministic finite automaton is not allowed to move its reading head back into the part of the input string that has already been read, the portion of the string to the left of the reading head cannot influence the future operation of the machine. Thus a configuration is determined by the current state and the unread part of the string being processed. In other words, a **configuration of a deterministic finite automaton** $(K, \Sigma, \delta, s, F)$ is any element of $K \times \Sigma^*$. For example, the configuration illustrated in Figure 2-1 is $(q_2, ababab)$.

The binary relation $\vdash_M$ holds between two configurations of M if and only if the machine can pass from one to the other as a result of a single move. Thus if (q, w) and (q', w') are two configurations of M, then $(q, w) \vdash_M (q', w')$ if and only if $w = aw'$ for some symbol $a \in \Sigma$, and $\delta(q, a) = q'$. In this case we say

that (q, w) **yields** (q', w') **in one step**. Note that in fact $\vdash_M$ is a function from $K \times \Sigma^+$ to $K \times \Sigma^*$, that is, for every configuration except those of the form (q, e) there is a uniquely determined next configuration. A configuration of the form (q, e) signifies that M has consumed all its input, and hence its operation ceases at this point.

We denote the reflexive, transitive closure of $\vdash_M$ by $\vdash_M^*$; $(q, w) \vdash_M^* (q', w')$ is read, (q, w) **yields** (q', w') (after some number, possibly zero, of steps). A string $w \in \Sigma^*$ is said to be **accepted** by M if and only if there is a state $q \in F$ such that $(s, w) \vdash_M^* (q, e)$. Finally, **the language accepted by** M, $L(M)$, is the set of all strings accepted by M.

Example 2.1.1: Let M be the deterministic finite automaton $(K, \Sigma, \delta, s, F)$, where

$$K = \{q_0, q_1\},$$
$$\Sigma = \{a, b\},$$
$$s = q_0,$$
$$F = \{q_0\},$$

and δ is the function tabulated below.

q	σ	$\delta(q, \sigma)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_1
q_1	b	q_0

It is then easy to see that $L(M)$ is the set of all strings in $\{a, b\}^*$ that have an even number of b's. For M passes from state q_0 to q_1 or from q_1 back to q_0 when a b is read, but M essentially ignores a's, always remaining in its current state when an a is read. Thus M counts b's modulo 2, and since q_0 (the initial state) is also the sole final state, M accepts a string if and only if the number of b's is even.

If M is given the input $aabba$, its initial configuration is $(q_0, aabba)$. Then

$$\begin{aligned}(q_0, aabba) &\vdash_M (q_0, abba)\\ &\vdash_M (q_0, bba)\\ &\vdash_M (q_1, ba)\\ &\vdash_M (q_0, a)\\ &\vdash_M (q_0, e)\end{aligned}$$

Therefore $(q_0, aabba) \vdash_M^* (q_0, e)$, and so $aabba$ is accepted by M.$\diamondsuit$

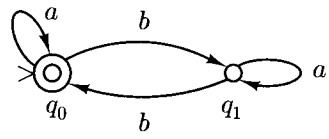


Figure 2-2

The tabular representation of the transition function used in this example is not the clearest description of a machine. We generally use a more convenient graphical representation called the **state diagram** (Figure 2-2). The state diagram is a directed graph, with certain additional information incorporated into the picture. States are represented by nodes, and there is an arrow labeled with a from node q to q' whenever $\delta(q,a) = q'$. Final states are indicated by double circles, and the initial state is shown by a $>$. Figure 2-2 shows the state diagram of the deterministic finite automaton of Example 2.1.1.

Example 2.1.2: Let us design a deterministic finite automaton M that accepts the language $L(M) = \{w \in \{a,b\}^* : w \text{ does not contain three consecutive } b\text{'s}\}$. We let $M = (K, \Sigma, \delta, s, F)$, where

$$\begin{aligned} K &= \{q_0, q_1, q_2, q_3\}, \\ \Sigma &= \{a, b\}, \\ s &= q_0, \\ F &= \{q_0, q_1, q_2\}, \end{aligned}$$

and δ is given by the following table.

q	σ	$\delta(q,\sigma)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_0
q_1	b	q_2
q_2	a	q_0
q_2	b	q_3
q_3	a	q_3
q_3	b	q_3

The state diagram is shown in Figure 2-3. To see that M does indeed accept the specified language, note that as long as three consecutive b's have not been read, M is in state q_i (where i is 0, 1, or 2) immediately after reading a run of i consecutive b's that either began the input string or was preceded by an a. In particular, whenever an a is read and M is in state q_0, q_1, or q_2, M returns to

its initial state q_0. States q_0, q_1, and q_2 are all final, so any input string not containing three consecutive b's will be accepted. However, a run of three b's will drive M to state q_3, which is not final, and M will then remain in this state regardless of the symbols in the rest of the input string. State q_3 is said to be a *dead state*, and if M reaches state q_3 it is said to be *trapped*, since no further input can enable it to escape from this state.◇

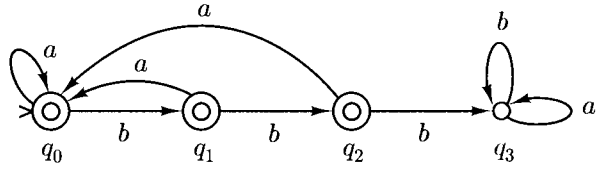


Figure 2-3

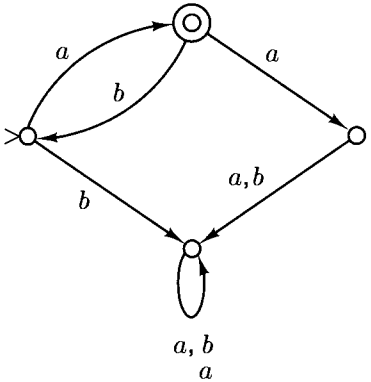
Problems for Section 2.1

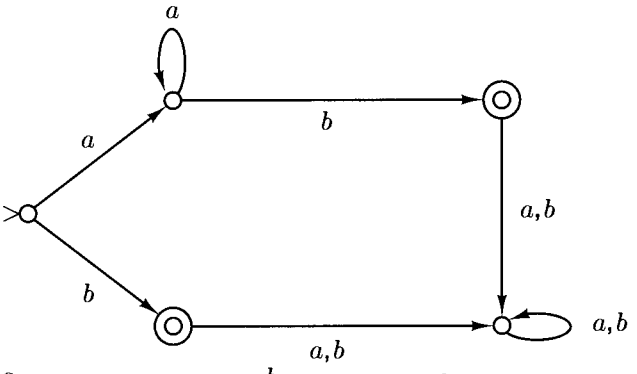
2.1.1. Let M be a deterministic finite automaton. Under exactly what circumstances is $e \in L(M)$? Prove your answer.

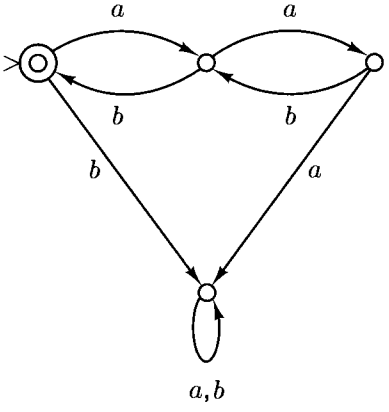
2.1.2. Describe informally the languages accepted by the deterministic finite automata shown in the next page.

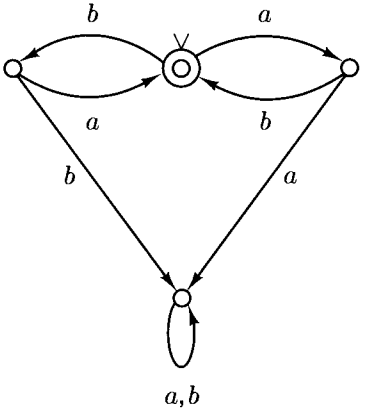
2.1.3. Construct deterministic finite automata accepting each of the following languages.
(a) $\{w \in \{a,b\}^* : \text{each } a \text{ in } w \text{ is immediately preceded by a } b\}$.
(b) $\{w \in \{a,b\}^* : w \text{ has } abab \text{ as a substring}\}$.
(c) $\{w \in \{a,b\}^* : w \text{ has neither } aa \text{ nor } bb \text{ as a substring}\}$.
(d) $\{w \in \{a,b\}^* : w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s}\}$.
(e) $\{w \in \{a,b\}^* : w \text{ has both } ab \text{ and } ba \text{ as substrings}\}$.

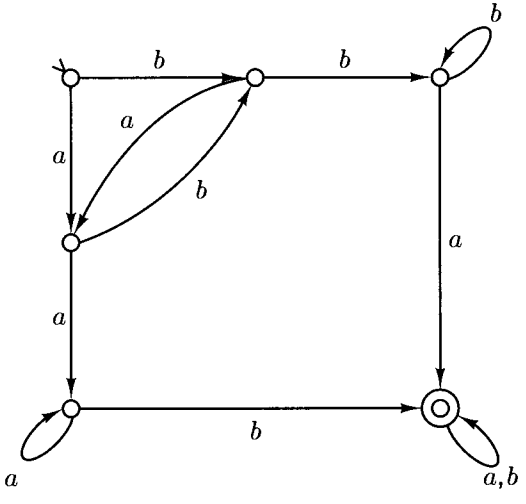
2.1.4. A **deterministic finite-state transducer** is a device much like a deterministic finite automaton, except that its purpose is not to accept strings or languages but to transform input strings into output strings. Informally, it starts in a designated initial state and moves from state to state, depending on the input, just as a deterministic finite automaton does. On each step, however, it emits (or writes onto an output tape) a string of zero or one or more symbols, depending on the current state and the input symbol. The state diagram for a deterministic finite-state transducer looks like that for a deterministic finite automaton, except that the label on an arrow looks like

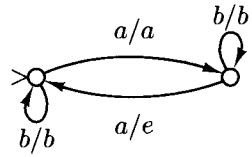

a
a
b
a,b
b
a, b


a
b
a
a,b
b
a,b
a,b


a
a
b
b
b
a
a,b


b
a
a
b
b
a
a,b


b
b
b
a
a
b
a
a
b
a
a,b



a/w, which means "if the input symbol is a, follow this arrow and emit output w". For example, the deterministic finite-state transducer over $\{a, b\}$ shown above transmits all b's in the input string but omits every other a.

(a) Draw state diagrams for deterministic finite-state transducers over $\{a, b\}$ that do the following.

(i) On input w, produce output a^n, where n is the number of occurrences of the substring ab in w.

(ii) On input w, produce output a^n, where n is the number of occurrences of the substring aba in w.

(iii) On input w, produce a string of length w whose ith symbol is an a if $i = 1$, or if $i > 1$ and the ith and $(i-1)$st symbols of w are different; otherwise, the ith symbol of the output is a b. For example, on input *aabba* the transducer should output *ababa*, and on input *aaaab* it should output *abbba*.

(b) Formally define

(i) a deterministic finite-state transducer;

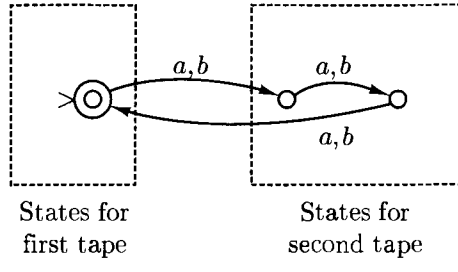
(ii) the notion of a configuration for such an automaton;

(iii) the yields in one step relation $\vdash$ between configurations;

(iv) the notion that such an automaton produces output u on input w;

(v) the notion that such an automaton computes a function.

2.1.5. A deterministic 2-tape finite automaton is a device like a deterministic finite automaton for accepting pairs of strings. Each state is in one of two sets; depending on which set the state is in, the transition function refers to the first or second tape. For example, the automaton shown below accepts all pairs of strings $(w_1, w_2) \in \{a, b\}^* \times \{a, b\}^*$ such that $|w_2| = 2|w_1|$.



(a) Draw state diagrams for deterministic 2-tape finite automata that accept each of the following.

(i) All pairs of strings (w_1, w_2) in $\{a,b\}^* \times \{a,b\}^*$ such that $|w_1| = |w_2|$, and $w_1(i) \neq w_2(i)$ for all i.

(ii) All pairs of strings (w_1, w_2) in $\{a,b\}^* \times \{a,b\}^*$ such that the length of w_2 is twice the number of a's in w_1 plus three times the number of b's in w_1.

(iii) $\{(a^n b, a^n b^m) : n, m \geq 0\}$.

(iv) $\{(a^n b, a^m b^n) : n, m \geq 0\}$.

(b) Formally define

(i) a deterministic 2-tape finite automaton;

(ii) the notion of a configuration for such an automaton;

(iii) the yields in one step relation $\vdash$ between configurations;

(iv) the notion that such an automaton accepts an ordered pair of strings;

(v) the notion that such an automaton accepts a set of ordered pairs of strings.

2.1.6. This problem refers to Problems 2.1.5 and 2.1.6. Show that if $f : \Sigma^* \mapsto \Sigma^*$ is a function that can be computed by a deterministic finite-state transducer, then $\{(w, f(w)) : w \in \Sigma^*\}$ is a set of pairs of strings accepted by some deterministic two-tape finite automaton.

2.1.7. We say that state q of a deterministic finite automaton $M = (K, \Sigma, \delta, q_0, F)$ is *reachable* if there exists $w \in \Sigma^*$ such that $(q_0, w) \vdash_M^* (q, e)$. Show that if we delete from M any nonreachable state, an automaton results that accepts the same language. Give an efficient algorithm for computing the set of all reachable states of a deterministic finite automaton.

2.2 NONDETERMINISTIC FINITE AUTOMATA

In this section we add a powerful and intriguing feature to finite automata. This feature is called **nondeterminism**, and is essentially the ability to change states in a way that is only partially determined by the current state and input symbol. That is, we shall now permit several possible "next states" for a given combination of current state and input symbol. The automaton, as it reads the input string, may choose at each step to go into any one of these legal next states; the choice is not determined by anything in our model, and is therefore said to be *nondeterministic*. On the other hand, the choice is not wholly unlimited either; only those next states that are legal from a given state with a given input symbol can be chosen.