3 The Theory of Automata

In this chapter we begin with the study of automaton. We deal with transition systems which are more general than finite automata. We define the acceptability of strings by finite automata and prove that nondeterministic finite automata have the same capability as the deterministic automata as far as acceptability is concerned. Besides, we discuss the equivalence of Mealy and Moore models. Finally, in the last section, we give an algorithm to construct a minimum state automaton equivalent to a given finite automaton.

3.1 DEFINITION OF AN AUTOMATON

We shall give the most general definition of an automaton and later modify it to computer applications. An automaton is defined as a system where energy, materials and information are transformed, transmitted and used for performing some functions without direct participation of man. Examples are automatic machine tools, automatic packing machines, and automatic photo printing machines.

In computer science the term 'automaton' means 'discrete automaton' and is defined in a more abstract way as shown in Fig. 3.1.

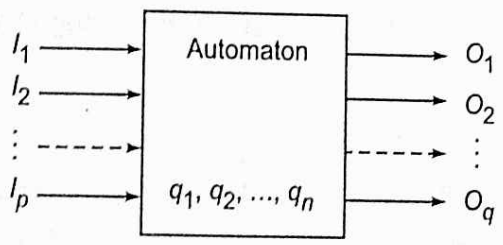

Fig. 3.1 Model of a discrete automaton.

The characteristics of automaton are now described.

(i) *Input.* At each of the discrete instants of time $t_1, t_2, \ldots, t_m$ the input values $I_1, I_2, \ldots, I_p$, each of which can take a finite number of fixed values from the input alphabet Σ, are applied to the input side of the model shown in Fig. 3.1.

(ii) *Output.* $O_1, O_2, \ldots, O_q$ are the outputs of the model, each of which can take a finite number of fixed values from an output O.

(iii) *States.* At any instant of time the automaton can be in one of the states $q_1, q_2, \ldots, q_n$.

(iv) *State relation.* The next state of an automaton at any instant of time is determined by the present state and the present input.

(v) *Output relation.* The output is related to either state only or to both the input and the state. It should be noted that at any instant of time the automaton is in some state. On 'reading' an input symbol, the automaton moves to a next state which is given by the state relation.

Note: An automaton in which the output depends only on the input is called an automaton without a memory. An automaton in which the output depends on the states as well, is called automaton with a finite memory. An automaton in which the output depends only on the states of the machine is called a *Moore machine.* An automaton in which the output depends on the state as well as on the input at any instant of time is called a *Mealy machine.*

EXAMPLE 3.1

Consider the simple shift register shown in Fig. 3.2 as a finite-state machine and study its operation.

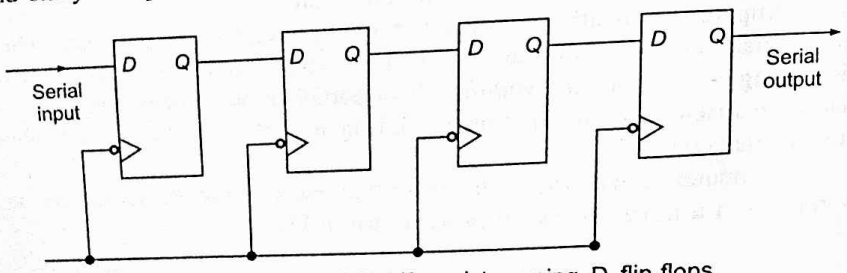

Fig. 3.2 A 4-bit serial shift register using D flip-flops.

Solution

The shift register (Fig. 3.2) can have $2^4 = 16$ states (0000, 0001, . . ., 1111), and one serial input and one serial output. The input alphabet is $\Sigma = \{0, 1\}$, and the output alphabet is $O = \{0, 1\}$. This 4-bit serial shift register can be further represented as in Fig. 3.3.

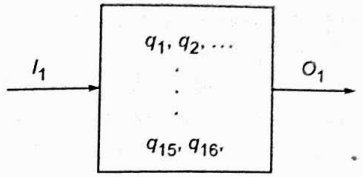

Fig. 3.3 A shift register as a finite-state machine.

From the operation, it is clear that the output will depend upon both the input and the state and so it is a Mealy machine.

In general, any sequential machine behaviour can be represented by an automaton.

3.2 DESCRIPTION OF A FINITE AUTOMATON

Definition 3.1 Analytically, a finite automaton can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

(i) Q is a finite nonempty set of states.
(ii) Σ is a finite nonempty set of inputs called the *input alphabet.*
(iii) δ is a function which maps $Q \times \Sigma$ into Q and is usually called the *direct transition function.* This is the function which describes the change of states during the transition. This mapping is usually represented by a transition table or a transition diagram.
(iv) $q_0 \in Q$ is the initial state.
(v) $F \subseteq Q$ is the set of final states. It is assumed here that there may be more than one final state.

Note: The transition function which maps $Q \times \Sigma^*$ into Q (i.e. maps a state and a string of input symbols including the empty string into a state) is called the *indirect transition function.* We shall use the same symbol δ to represent both types of transition functions and the difference can be easily identified by the nature of mapping (symbol or a string), i.e. by the argument. δ is also called the next state function. The above model can be represented graphically by Fig. 3.4.

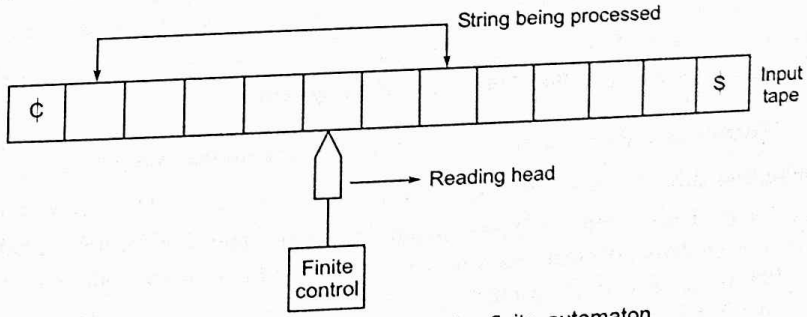

Fig. 3.4 Block diagram of a finite automaton.

Figure 3.4 is the block diagram for a finite automaton. The various components are explained as follows:

(i) *Input tape.* The input tape is divided into squares, each square containing a single symbol from the input alphabet Σ. The end squares of the tape contain the endmarker ¢ at the left end and the endmarker $ at the right end. The absence of endmarkers indicates that the tape is of infinite length. The left-to-right sequence of symbols between the two endmarkers is the input string to be processed.

(ii) *Reading head.* The head examines only one square at a time and can move one square either to the left or to the right. For further analysis, we restrict the movement of the R-head only to the right side.

(iii) *Finite control.* The input to the finite control will usually be the symbol under the R-head, say a, and the present state of the machine, say q, to give the following outputs: (a) A motion of R-head along the tape to the next square (in some a null move, i.e. the R-head remaining to the same square is permitted); (b) the next state of the finite state machine given by $\delta(q, a)$.

3.3 TRANSITION SYSTEMS

A transition graph or a transition system is a finite directed labelled graph in which each vertex (or node) represents a state and the directed edges indicate the transition of a state and the edges are labelled with input/output.

A typical transition system is shown in Fig. 3.5. In the figure, the initial state is represented by a circle with an arrow pointing towards it, the final state by two concentric circles, and the other states are represented by just a circle. The edges are labelled by input/output (e.g. by 1/0 or 1/1). For example, if the system is in the state q_0 and the input 1 is applied, the system moves to state q_1 as there is a directed edge from q_0 to q_1 with label 1/0. It outputs 0.

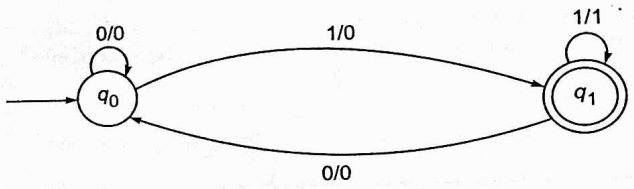

Fig. 3.5 A transition system.

We now give the (analytical) definition of a transition system.

Definition 3.2 A transition system is a 5-tuple $(Q, \Sigma, \delta, Q_0, F)$, where

(i) Q, Σ and F are the finite nonempty set of states, the input alphabet, and the set of final states, respectively, as in the case of finite automata;

(ii) $Q_0 \subseteq Q$, and Q_0 is nonempty; and

(iii) δ is a finite subset of $Q \times \Sigma^* \times Q$.

In other words, if (q_1, w, q_2) is in δ, it means that the graph starts at the vertex q_1, goes along a set of edges, and reaches the vertex q_2. The concatenation of the label of all the edges thus encountered is w.

Definition 3.3 A transition system accepts a string w in Σ^* if

(i) there exists a path which originates from some initial state, goes along the arrows, and terminates at some final state; and

(ii) the path value obtained by concatenation of all edge-labels of the path is equal to w.

Example 3.2

Consider the transition system given in Fig. 3.6.

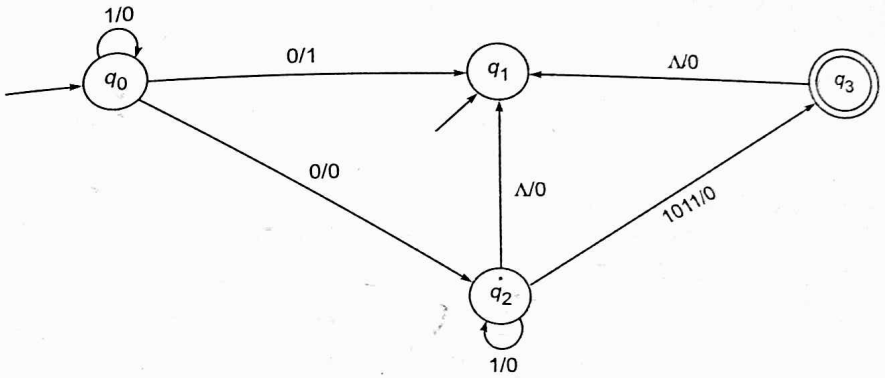

Fig. 3.6 Transition system for Example 3.2.

Determine the initial states, the final states, and the acceptability of 101011, 111010.

Solution

The initial states are q_0 and q_1. There is only one final state, namely q_3.

The path-value of $q_0q_0q_2q_3$ is 101011. As q_3 is the final state, 101011 is accepted by the transition system. But, 111010 is not accepted by the transition system as there is no path with path value 111010.

Note: Every finite automaton $(Q, \Sigma, \delta, q_0, F)$ can be viewed as a transition system $(Q, \Sigma, \delta' Q_0, F)$ if we take $Q_0 = \{q_0\}$ and $\delta' = \{(q, w, \delta(q, w))|q \in Q, w \in \Sigma^*\}$. But a transition system need not be a finite automaton. For example, a transition system may contain more than one initial state.

3.4 PROPERTIES OF TRANSITION FUNCTIONS

Property 1 $\delta(q, \Lambda) = q$ is a finite automaton. This means that the state of the system can be changed only by an input symbol.

Property 2 For all strings w and input symbols a,

$$\delta(q, aw) = \delta(\delta(q, a), w)$$

$$\delta(q, wa) = \delta(\delta(q, w), a)$$

This property gives the state after the automaton consumes or reads the first symbol of a string aw and the state after the automaton consumes a prefix of the string wa.

EXAMPLE 3.3

Prove that for any transition function δ and for any two input strings x and y,

$$\delta(q, xy) = \delta(\delta(q, x), y) \tag{3.1}$$

Proof By the method of induction on $|y|$, i.e. length of y.

Basis: When $|y| = 1$, $y = a \in \Sigma$

$$\begin{aligned} \text{L.H.S. of (3.1)} &= \delta(q, xa) \\ &= \delta(\delta(q, x), a) \quad \text{by Property 2} \\ &= \text{R.H.S. of (3.1)} \end{aligned}$$

Assume the result, i.e. (3.1) for all strings x and strings y with $|y| = n$. Let y be a string of length $n + 1$. Write $y = y_1a$ where $|y_1| = n$.

$$\begin{aligned} \text{L.H.S. of (3.1)} &= \delta(q, xy_1a) = \delta(q, x_1a), \quad x_1 = xy_1 \\ &= \delta(\delta(q, x_1), a) \quad \text{by Property 2} \\ &= \delta(\delta(q, xy_1), a) \\ &= \delta(\delta(\delta(q, x), y_1), a) \quad \text{by induction hypothesis} \\ \text{R.H.S. of (3.1)} &= \delta(\delta(q, x), y_1a) \\ &= \delta(\delta(\delta(q, x), y_1), a) \quad \text{by Property 2} \end{aligned}$$

Hence, L.H.S. = R.H.S. This proves (3.1) for any string y of length $n + 1$. By the principle of induction, (3.1) is true for all strings. ∎

EXAMPLE 3.4

Prove that if $\delta(q, x) = \delta(q, y)$, then $\delta(q, xz) = \delta(q, yz)$ for all strings z in Σ^+.

Solution

$$\begin{aligned} \delta(q, xz) &= \delta(\delta(q, x), z) \quad \text{by Example 3.3} \\ &= \delta(\delta(q, y), z) \end{aligned} \tag{3.2}$$

By Example 3.3,

$$\begin{aligned} \delta(q, yz) &= \delta(\delta(q, y), z) \\ &= \delta(q, xz) \end{aligned} \tag{3.3}$$

3.5 ACCEPTABILITY OF A STRING BY A FINITE AUTOMATON

Definition 3.4 A string x is accepted by a finite automaton

$$M = (Q, \Sigma, \delta, q_0, F)$$

if $\delta(q_0, x) = q$ for some $q \in F$.

This is basically the acceptability of a string by the final state.

Note: A final state is also called an accepting state.

EXAMPLE 3.5

Consider the finite state machine whose transition function δ is given by Table 3.1 in the form of a transition table. Here, $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, $F = \{q_0\}$. Give the entire sequence of states for the input string 110101.

TABLE 3.1 Transition Function Table for Example 3.5

State	Input	
	0	1
$\rightarrow (q_0)$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Solution

$$\begin{aligned}
\delta(q_0, 110101) &= \delta(q_1, 10101) \\
&= \delta(q_0, 0101) \\
&= \delta(q_2, 101) \\
&= \delta(q_3, 01) \\
&= \delta(q_1, 1) \\
&= \delta(q_0, \Lambda) \\
&= q_0
\end{aligned}$$

Hence,

$$q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_0 \xrightarrow{0} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_1 \xrightarrow{1} q_0$$

The symbol $\downarrow$ indicates that the current input symbol is being processed by the machine.

3.6 NONDETERMINISTIC FINITE STATE MACHINES

We explain the concept of nondeterministic finite automaton using a transiti diagram (Fig. 3.7).

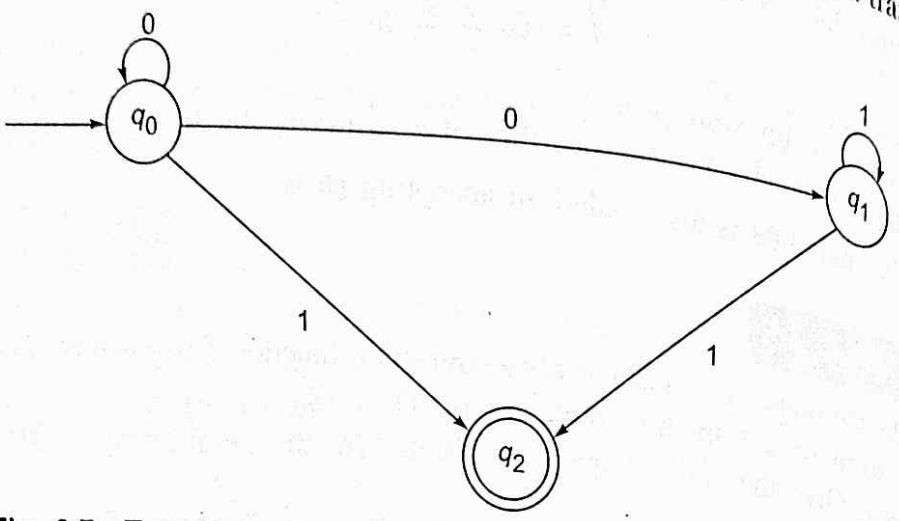

Fig. 3.7 Transition system representing nondeterministic automaton.

If the automaton is in a state $\{q_0\}$ and the input symbol is 0, what will be the next state? From the figure it is clear that the next state will be either $\{q_0\}$ or $\{q_1\}$. Thus some moves of the machine cannot be determined uniquely by the input symbol and the present state. Such machines are called nondeterministic automata, the formal definition of which is now given.

Definition 3.5 A nondeterministic finite automaton (NDFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

(i) Q is a finite nonempty set of states;
(ii) Σ is a finite nonempty set of inputs;
(iii) δ is the transition function mapping from $Q \times \Sigma$ into 2^Q which is the power set of Q, the set of all subsets of Q;
(iv) $q_0 \in Q$ is the initial state; and
(v) $F \subseteq Q$ is the set of final states.

We note that the difference between the deterministic and nondeterministic automata is only in δ. For deterministic automaton (DFA), the outcome is a state, i.e. an element of Q; for nondeterministic automaton the outcome is a subset of Q.

Consider, for example, the nondeterministic automaton whose transition diagram is described by Fig. 3.8.

The sequence of states for the input string 0100 is given in Fig. 3.9. Hence,

$$\delta(q_0, 0100) = \{q_0, q_3, q_4\}$$

Since q_4 is an accepting state, the input string 0100 will be accepted by the nondeterministic automaton.

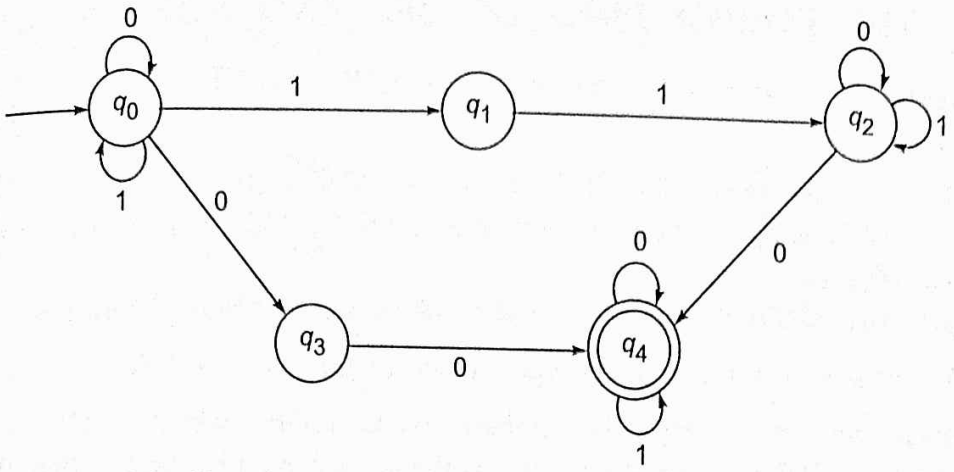

Fig. 3.8 Transition system for a nondeterministic automaton.

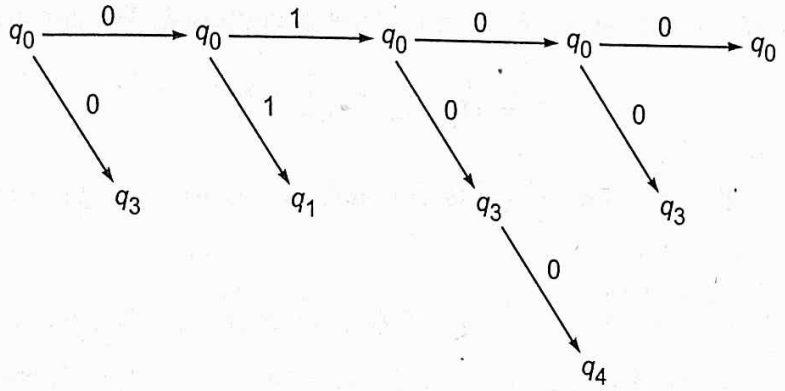

Fig. 3.9 States reached while processing 0100.

Definition 3.6 A string $w \in \Sigma^*$ is accepted by NDFA M if $\delta(q_0, w)$ contains some final state.

Note: As M is nondeterministic, $\delta(q_0, w)$ may have more than one state. So w is accepted by M if a final state is *one* among the possible states that M can reach on application of w.

We can visualize the working of an NDFA M as follows: Suppose M reaches a state q and reads an input symbol a. If $\delta(q, a)$ has n elements, the automaton splits into n identical copies of itself; each copy pursuing one choice determined by an element of $\delta(q, a)$. This type of parallel computation continues. When a copy encounters (q, a) for which $\delta(q, a) = \emptyset$, this copy of the machine 'dies'; however the computation is pursued by the other copies. If any one of the copies of M reaches a final state after processing the entire input string w, then we say that M accepts w. Another way of looking at the computation by an NDFA M is to assign a tree structure for computing $\delta(q, w)$. The root of the tree has the label q. For every input symbol in w, the tree branches itself. When a leaf of the tree has a final state as its label, then M accepts w.

Definition 3.7 The set accepted by an automaton M (deterministic or nondeterministic) is the set of all input strings accepted by M. It is denoted by $T(M)$.

3.7 THE EQUIVALENCE OF DFA AND NDFA

We naturally try to find the relation between DFA and NDFA. Intuitively we now feel that:

(i) A DFA can simulate the behaviour of NDFA. (In other words, a DFA $(Q, \Sigma, \delta, q_0, F)$ can be viewed as an NDFA $(Q, \Sigma, \delta', q_0, F)$ by defining $\delta'(q, a) = \{\delta(q, a)\}$.)

(ii) Any NDFA is a more general machine without being more powerful.

We now give a theorem on equivalence of DFA and NDFA.

Theorem 3.1 For every NDFA, there exists a DFA which simulates the behaviour of NDFA. Alternatively, if L is the set accepted by NDFA, then there exists a DFA which also accepts L.

Proof Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NDFA accepting L. We construct a DFA M' as:

$$M' = (Q', \Sigma, \delta, q_0', F')$$

where

(i) $Q' = 2^Q$ (any state in Q' is denoted by $[q_1, q_2, \ldots, q_j]$, where $q_1, q_2, \ldots, q_j \in Q$);

(ii) $q_0' = [q_0]$; and

(iii) F' is the set of all subsets of Q containing an element of F.

Before defining δ', let us look at the construction of Q', q_0' and F'. M is initially at q_0. But on application of an input symbol, say a, M can reach any of the states $\delta(q_0, a)$. To describe M, just after the application of the input symbol a, we require all the possible states that M can reach after the application of a. So, M' has to remember all these possible states at any instant of time. Hence the states of M' are defined as subsets of Q. As M starts with the initial state q_0, q_0' is defined as $[q_0]$. A string w belongs to $T(M)$ if a final state is one of the possible states that M reaches on processing w. So, a final state in M' (i.e. an element of F') is any subset of Q containing some final state of M.

Now we can define δ':

(iv) $\delta'([q_1, q_2, \ldots, q_i], a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \ldots \cup \delta(q_i, a)$.

Equivalently,

$$\delta'([q_1, q_2, \ldots, q_i], a) = [p_1, \ldots, p_j]$$

if and only if

$$\delta(\{q_1, \ldots, q_i\}, a) = \{p_1, p_2, \ldots, p_j\}.$$

Before proving $L = T(M')$, we prove an auxiliary result

$$\delta'(q_0', x) = [q_1, \ldots, q_i], \tag{3.4}$$

if and only if $\delta(q_0, x) = \{q_1, \ldots, q_i\}$ for all x in Σ^*.

We prove by induction on $|x|$, the 'if' part, i.e.

$$\delta'(q_0', x) = [q_1, q_2, \ldots, q_i] \tag{3.5}$$

if $\delta(q_0, x) = \{q_1, \ldots, q_i\}$.

When $|x| = 0$, $\delta(q_0, \Lambda) = \{q_0\}$, and by definition of δ', $\delta'(q_0', \Lambda) = q_0' = [q_0]$. So, (3.5) is true for x with $|x| = 0$. Thus there is basis for induction.

Assume that (3.5) is true for all strings y with $|y| \leq m$. Let x be a string of length $m + 1$. We can write x as ya, where $|y| = m$ and $a \in \Sigma$. Let $\delta(q_0, y) = \{p_1, \ldots, p_j\}$ and $\delta(q_0, ya) = \{r_1, r_2, \ldots, r_k\}$. As $|y| \leq m$, by induction hypothesis we have

$$\delta'(q_0', y) = [p_1, \ldots, p_j] \tag{3.6}$$

Also,

$$\{r_1, r_2, \ldots, r_k\} = \delta(q_0, ya) = \delta(\delta(q_0, y), a) = \delta(\{p_1, \ldots, p_j\}, a)$$

By definition of δ',

$$\delta'([p_1, \ldots, p_j], a) = [r_1, \ldots, r_k] \tag{3.7}$$

Hence,

$$\begin{aligned} \delta'(q_0', ya) &= \delta'(\delta'(q'_0, y), a) = \delta'([p_1, \ldots, p_j], a) && \text{by (3.6)} \\ &= [r_1, \ldots, r_k] && \text{by (3.7)} \end{aligned}$$

Thus we have proved (3.5) for $x = ya$.

By induction, (3.5) is true for all strings x. The other part (i.e. the 'only if' part), can be proved similarly, and so (3.4) is established.

Now, $x \in T(M)$ if and only if $\delta(q, x)$ contains a state of F. By (3.4), $\delta(q_0, x)$ contains a state of F if and only if $\delta'(q'_0, x)$ is in F'. Hence, $x \in T(M)$ if and only if $x \in T(M')$. This proves that DFA M' accepts L. ∎

Note: In the construction of a deterministic finite automaton M_1 equivalent to a given nondeterministic automaton M, the only difficult part is the construction of δ' for M_1. By definition,

$$\delta'([q_1 \ldots q_k], a) = \bigcup_{i=1}^{k} \delta(q_i, a)$$

So we have to apply δ to (q_i, a) for each $i = 1, 2, \ldots, k$ and take their union to get $\delta'([q_1 \ldots q_k], a)$.

When δ for M is given in terms of a state table, the construction is simpler. $\delta(q_i, a)$ is given by the row corresponding to q_i and the column corresponding to a. To construct $\delta'([q_1 \ldots q_k], a)$, consider the states appearing in the rows corresponding to $q_1, \ldots, q_k$, and the column corresponding to a. These states constitute $\delta'([q_1 \ldots q_k], a)$.

Note: We write δ' as δ itself when there is no ambiguity. We also mark the initial state with $\rightarrow$ and the final state with a circle in the state table.

EXAMPLE 3.6

Construct a deterministic automaton equivalent to

$$M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_0\})$$

where δ is defined by its state table (see Table 3.2).

TABLE 3.2 State Table for Example 3.6

State/Σ	0	1
$\rightarrow (q_0)$	q_0	q_1
q_1	q_1	q_0, q_1

Solution

For the deterministic automaton M_1,

(i) the states are subsets of $\{q_0, q_1\}$, i.e. $\emptyset$, $[q_0]$, $[q_0, q_1]$, $[q_1]$;
(ii) $[q_0]$ is the initial state;
(iii) $[q_0]$ and $[q_0, q_1]$ are the final states as these are the only states containing q_0; and
(iv) δ is defined by the state table given by Table 3.3.

TABLE 3.3 State Table of M_1 for Example 3.6

State/Σ	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$[q_0]$	$[q_0]$	$[q_1]$
$[q_1]$	$[q_1]$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$

The states q_0 and q_1 appear in the rows corresponding to q_0 and q_1 and the column corresponding to 0. So, $\delta([q_0, q_1], 0) = [q_0, q_1]$.

When M has n states, the corresponding finite automaton has 2^n states. However, we need not construct δ for all these 2^n states, but only for those states that are reachable from $[q_0]$. This is because our interest is only in constructing M_1 accepting $T(M)$. So, we start the construction of δ for $[q_0]$. We continue by considering only the states appearing earlier under the input columns and constructing δ for such states. We halt when no more new states appear under the input columns.

EXAMPLE 3.7

Find a deterministic acceptor equivalent to

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$$

where δ is as given by Table 3.4.

TABLE 3.4 State Table for Example 3.7

State/Σ	a	b
$\rightarrow q_0$	q_0, q_1	q_2
q_1	q_0	q_1
(q_2)		q_0, q_1

Solution

The deterministic automaton M_1 equivalent to M is defined as follows:

$$M_1 = (2^Q, \{a, b\}, \delta, [q_0], F')$$

where

$$F = \{[q_2], [q_0, q_2], [q_1, q_2], [q_0, q_1, q_2]\}$$

We start the construction by considering $[q_0]$ first. We get $[q_2]$ and $[q_0, q_1]$. Then we construct δ for $[q_2]$ and $[q_0, q_1]$. $[q_1, q_2]$ is a new state appearing under the input columns. After constructing δ for $[q_1, q_2]$, we do not get any new states and so we terminate the construction of δ. The state table is given by Table 3.5.

TABLE 3.5 State Table of M_1 for Example 3.7

State/Σ	a	b
$[q_0]$	$[q_0, q_1]$	$[q_2]$
$[q_2]$	$\emptyset$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_1, q_2]$
$[q_1, q_2]$	$[q_0]$	$[q_0, q_1]$

EXAMPLE 3.8

Construct a deterministic finite automaton equivalent to

$$M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}), \delta, q_0, \{q_3\})$$

where δ is given by Table 3.6.

TABLE 3.6 State Table for Example 3.8

State/Σ	a	b
$\rightarrow q_0$	q_0, q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_3
(q_3)		q_2

Solution

Let $Q = \{q_0, q_1, q_2, q_3\}$. Then the deterministic automaton M_1 equivalent to M is given by

$$M_1 = (2^Q, \{a, b\}, \delta, [q_0], F)$$

where F consists of:

$[q_3]$, $[q_0, q_3]$, $[q_1, q_3]$, $[q_2, q_3]$, $[q_0, q_1, q_3]$, $[q_0, q_2, q_3]$, $[q_1, q_2, q_3]$

and

$$[q_0, q_1, q_2, q_3]$$

and where δ is defined by the state table given by Table 3.7.

TABLE 3.7 State Table of M_1 for Example 3.8

State/Σ	a	b
$[q_0]$	$[q_0, q_1]$	$[q_0]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_3]$
$[q_0, q_1, q_3]$	$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$
$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$

3.8 MEALY AND MOORE MODELS

3.8.1 FINITE AUTOMATA WITH OUTPUTS

The finite automata which we considered in the earlier sections have binary output, i.e. either they accept the string or they do not accept the string. This acceptability was decided on the basis of reachability of the final state by the initial state. Now, we remove this restriction and consider the model where the outputs can be chosen from some other alphabet. The value of the output function $Z(t)$ in the most general case is a function of the present state $q(t)$ and the present input $x(t)$, i.e.

$$Z(t) = \lambda(q(t), x(t))$$

where λ is called the output function. This generalized model is usually called the *Mealy machine*. If the output function $Z(t)$ depends only on the present state and is independent of the current input, the output function may be written as

$$Z(t) = \lambda(q(t))$$

This restricted model is called the *Moore machine*. It is more convenient to use Moore machine in automata theory. We now give the most general definitions of these machines.

Definition 3.8 A Moore machine is a six-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where

(i) Q is a finite set of states;
(ii) Σ is the input alphabet;
(iii) Δ is the output alphabet;
(iv) δ is the transition function from $\Sigma \times Q$ into Q;
(v) λ is the output function mapping Q into Δ; and
(vi) q_0 is the initial state.

Definition 3.9 A Mealy machine is a six-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where all the symbols except λ have the same meaning as in the Moore machine. λ is the output function mapping $\Sigma \times Q$ into Δ.

For example, Table 3.8 describes a Moore machine. The initial state q_0 is marked with an arrow. The table defines δ and λ.

TABLE 3.8 A Moore Machine

Present state	Next state δ		Output
	$a = 0$	$a = 1$	λ
$\rightarrow q_0$	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

For the input string 0111, the transition of states is given by $q_0 \rightarrow q_3 \rightarrow q_0 \rightarrow q_1 \rightarrow q_2$. The output string is 00010. For the input string Λ, the output is $\lambda(q_0) = 0$.

Transition Table 3.9 describes a Mealy machine.

TABLE 3.9 A Mealy Machine

Present state	Next state			
	$a = 0$		$a = 1$	
	state	output	state	output
$\rightarrow q_1$	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

Note: For the input string 0011, the transition of states is given by $q_1 \rightarrow q_3 \rightarrow q_2 \rightarrow q_4 \rightarrow q_3$, and the output string is 0100. In the case of a Mealy machine, we get an output only on the application of an input symbol. So for the input string Λ, the output is only Λ. It may be observed that in the case of a Moore machine, we get $\lambda(q_0)$ for the input string Λ.

Remark A finite automaton can be converted into a Moore machine by introducing $\Delta = \{0, 1\}$ and defining $\lambda(q) = 1$ if $q \in F$ and $\lambda(q) = 0$ if $q \notin F$.

For a Moore machine if the input string is of length n, the output string is of length $n + 1$. The first output is $\lambda(q_0)$ for all output strings. In the case of a Mealy machine if the input string is of length n, the output string is also of the same length n.

3.8.2 Procedure for Transforming a Mealy Machine into a Moore Machine

We develop procedures for transforming a Mealy machine into a Moore machine and vice versa so that for a given input string the output strings are the same (except for the first symbol) in both the machines.

EXAMPLE 3.9

Consider the Mealy machine described by the transition table given by Table 3.10. Construct a Moore machine which is equivalent to the Mealy machine.

TABLE 3.10 Mealy Machine of Example 3.9

Present state	Next state			
	Input $a = 0$		Input $a = 1$	
	state	output	state	output
$\rightarrow q_1$	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

Solution

At the first stage we develop the procedure so that both machines accept exactly the same set of input sequences. We look into the next state column for any state, say q_i, and determine the number of different outputs associated with q_i in that column.

We split q_i into several different states, the number of such states being equal to the number of different outputs associated with q_i. For example, in this problem, q_1 is associated with one output 1 and q_2 is associated with two different outputs 0 and 1. Similarly, q_3 and q_4 are associated with the outputs 0 and 0, 1, respectively. So, we split q_2 into q_{20} and q_{21}. Similarly, q_4 is split into q_{40} and q_{41}. Now Table 3.10 can be reconstructed for the new states as given by Table 3.11.

TABLE 3.11 State Table for Example 3.9

Present state	Next state			
	Input $a = 0$		Input $a = 1$	
	state	output	state	output
$\rightarrow q_1$	q_3	0	q_{20}	0
q_{20}	q_1	1	q_{40}	0
q_{21}	q_1	1	q_{40}	0
q_3	q_{21}	1	q_1	1
q_{40}	q_{41}	1	q_3	0
q_{41}	q_{41}	1	q_3	0

The pair of states and outputs in the next state column can be rearranged as given by Table 3.12.

TABLE 3.12 Revised State Table for Example 3.9

Present state	Next state		Output
	$a = 0$	$a = 1$	
$\rightarrow q_1$	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

Table 3.12 gives the Moore machine. Here we observe that the initial state q_1 is associated with output 1. This means that with input Λ we get an output of 1, if the machine starts at state q_1. Thus this Moore machine accepts a zero-length sequence (null sequence) which is not accepted by the Mealy machine. To overcome this situation, either we must neglect the response of a Moore machine to input Λ, or we must add a new starting state q_0, whose state transitions are identical with those of q_1 but whose output is 0. So Table 3.12 is transformed to Table 3.13.

TABLE 3.13 Moore Machine of Example 3.9

Present state	Next state		Output
	$a = 0$	$a = 1$	
$\rightarrow q_0$	q_3	q_{20}	0
q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

From the foregoing procedure it is clear that if we have an m-output, n-state Mealy machine, the corresponding m-output Moore machine has no more than $mn + 1$ states.

3.8.3 Procedure for Transforming a Moore Machine into a Mealy Machine

We modify the acceptability of input string by a Moore machine by neglecting the response of the Moore machine to input Λ. We thus define that Mealy Machine M and Moore Machine M' are equivalent if for all input strings w, $bZ_M(w) = Z_{M'}(w)$, where b is the output of the Moore machine for its initial state. We give the following result: Let $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ be a Moore machine. Then the following procedure may be adopted to construct an equivalent Mealy machine M_2.

Construction

(i) We have to define the output function λ' for the Mealy machine as a function of the present state and the input symbol. We define λ' by

$$\lambda'(q, a) = \lambda(\delta(q, a)) \quad \text{for all states } q \text{ and input symbols } a.$$

(ii) The transition function is the same as that of the given Moore machine.

EXAMPLE 3.10

Construct a Mealy Machine which is equivalent to the Moore machine given by Table 3.14.

TABLE 3.14 Moore Machine of Example 3.10

Present state	Next state		Output
	$a = 0$	$a = 1$	
$\rightarrow q_0$	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

Solution

We must follow the reverse procedure of converting a Mealy machine into a Moore machine. In the case of the Moore machine, for every input symbol we form the pair consisting of the next state and the corresponding output and reconstruct the table for the Mealy Machine. For example, the states q_3 and q_1 in the next state column should be associated with outputs 0 and 1, respectively. The transition table for the Mealy machine is given by Table 3.15.

TABLE 3.15 Mealy Machine of Example 3.10

Present state	Next state			
	$a = 0$		$a = 1$	
	state	output	state	output
$\rightarrow q_0$	q_3	0	q_1	1
q_1	q_1	1	q_2	0
q_2	q_2	0	q_3	0
q_3	q_3	0	q_0	0

Note: We can reduce the number of states in any model by considering states with identical transitions. If two states have identical transitions (i.e. the rows corresponding to these two states are identical), then we can delete one of them.

EXAMPLE 3.11

Consider the Moore machine described by the transition table given by Table 3.16. Construct the corresponding Mealy machine.

TABLE 3.16 Moore Machine of Example 3.11

Present state	*Next state*		*Output*
	$a = 0$	$a = 1$	
$\rightarrow q_1$	q_1	q_2	0
q_2	q_1	q_3	0
q_3	q_1	q_3	1

Solution

We construct the transition table as in Table 3.17 by associating the output with the transitions.

In Table 3.17, the rows corresponding to q_2 and q_3 are identical. So, we can delete one of the two states, i.e. q_2 or q_3. We delete q_3. Table 3.18 gives the reconstructed table.

TABLE 3.17 Transition Table for Example 3.11

Present state	*Next state*			
	$a = 0$		$a = 1$	
	state	*output*	*state*	*output*
$\rightarrow q_1$	q_1	0	q_2	0
q_2	q_1	0	q_3	1
q_3	q_1	0	q_3	1

TABLE 3.18 Mealy Machine of Example 3.11

Present state	*Next state*			
	$a = 0$		$a = 1$	
	state	*output*	*state*	*output*
$\rightarrow q_1$	q_1	0	q_2	0
q_2	q_1	0	q_2	1

In Table 3.18, we have deleted the q_3-row and replaced q_3 by q_2 in the other rows.

EXAMPLE 3.12

Consider a Mealy machine represented by Fig. 3.10. Construct a Moore machine equivalent to this Mealy machine.

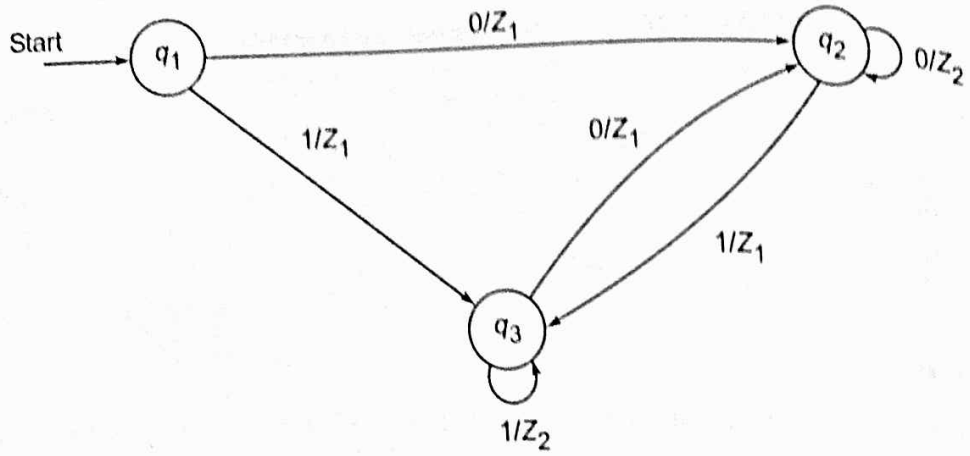

Fig. 3.10 Mealy machine of Example 3.12.

Solution

Let us convert the transition diagram into the transition Table 3.19. For the given problem: q_1 is not associated with any output; q_2 is associated with two different outputs Z_1 and Z_2; q_3 is associated with two different outputs Z_1 and Z_2. Thus we must split q_2 into q_{21} and q_{22} with outputs Z_1 and Z_2, respectively and q_3 into q_{31} and q_{32} with outputs Z_1 and Z_2, respectively. Table 3.19 may be reconstructed as Table 3.20.

TABLE 3.19 Transition Table for Example 3.12

Present state	*Next state*			
	a = 0		a = 1	
	state	*output*	*state*	*output*
$\rightarrow q_1$	q_2	Z_1	q_3	Z_1
q_2	q_2	Z_2	q_3	Z_1
q_3	q_2	Z_1	q_3	Z_2

TABLE 3.20 Transition Table of Moore Machine for Example 3.12

Present state	*Next state*		*Output*
	a = 0	a = 1	
$\rightarrow q_1$	q_{21}	q_{31}	
q_{21}	q_{22}	q_{31}	Z_1
q_{22}	q_{22}	q_{31}	Z_2
q_{31}	q_{21}	q_{32}	Z_1
q_{32}	q_{21}	q_{32}	Z_2

Figure 3.11 gives the transition diagram of the required Moore machine.

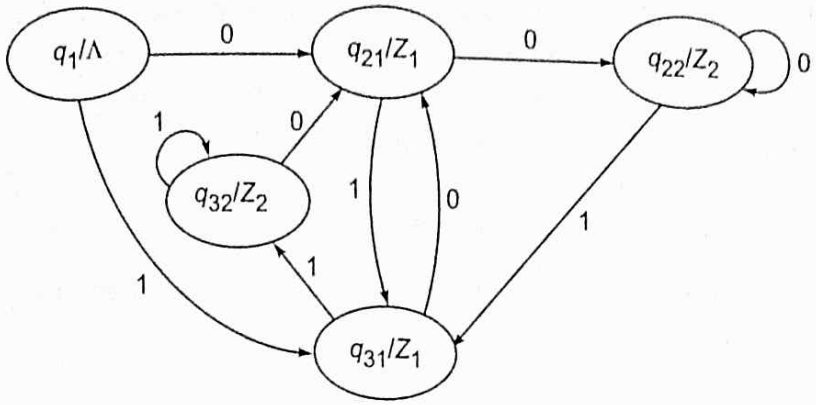

Fig. 3.11 Moore machine of Example 3.12.

3.9 MINIMIZATION OF FINITE AUTOMATA

In this section we construct an automaton with the minimum number of states equivalent to a given automaton M.

As our interest lies only in strings accepted by M, what really matters is whether a state is a final state or not. We define some relations in Q.

Definition 3.10 Two states q_1 and q_2 are equivalent (denoted by $q_1 \equiv q_2$) if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states, or both of them are nonfinal states for all $x \in \Sigma^*$.

As it is difficult to construct $\delta(q_1, x)$ and $\delta(q_2, x)$ for all $x \in \Sigma^*$ (there are an infinite number of strings in Σ^*), we give one more definition.

Definition 3.11 Two states q_1 and q_2 are k-equivalent ($k \geq 0$) if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states or both nonfinal states for all strings x of length k or less. In particular, any two final states are 0-equivalent and any two nonfinal states are also 0-equivalent.

We mention some of the properties of these relations.

Property 1 The relations we have defined, i.e. equivalence and k-equivalence, are equivalence relations, i.e. they are reflexive, symmetric and transitive.

Property 2 By Theorem 2.1, these induce partitions of Q. These partitions can be denoted by π and π_k, respectively. The elements of π_k are k-equivalence classes.

Property 3 If q_1 and q_2 are k-equivalent for all $k \geq 0$, then they are equivalent.

Property 4 If q_1 and q_2 are $(k + 1)$-equivalent, then they are k-equivalent.

Property 5 $\pi_n = \pi_{n+1}$ for some n. (π_n denotes the set of equivalence classes under n-equivalence.)

The following result is the key to the construction of minimum state automaton.

RESULT Two states q_1 and q_2 are $(k + 1)$-equivalent if (i) they are k-equivalent; (ii) $\delta(q_1, a)$ and $\delta(q_2, a)$ are also k-equivalent for every $a \in \Sigma$.

Proof We prove the result by contradiction. Suppose q_1 and q_2 are not $(k+1)$-equivalent. Then there exists a string $w = aw_1$ of length $k+1$ such that $\delta(q_1, aw_1)$ is a final state and $\delta(q_2, aw_1)$ is not a final state (or vice versa; the proof is similar). So $\delta(\delta(q_1, a), w_1)$ is a final state and $\delta(\delta(q_2, a), w_1)$ is not a final state. As w_1 is a string of length k, $\delta(q_1, a)$ and $\delta(q_2, a)$ are not k-equivalent. This is a contradiction, and hence the result is proved. ∎

Using the previous result we can construct the $(k+1)$-equivalence classes once the k-equivalence classes are known.

3.9.1 Construction of Minimum Automaton

Step 1 (Construction of π_0). By definition of 0-equivalence, $\pi_0 = \{Q_1^0, Q_2^0\}$ where Q_1^0 is the set of all final states and $Q_2^0 = Q - Q_1^0$.

Step 2 (Construction of π_{k+1} from π_k). Let Q_i^k be any subset in π_k. If q_1 and q_2 are in Q_i^k, they are $(k+1)$-equivalent provided $\delta(q_1, a)$ and $\delta(q_2, a)$ are k-equivalent. Find out whether $\delta(q_1, a)$ and $\delta(q_2, a)$ are in the same equivalence class in π_k for every $a \in \Sigma$. If so, q_1 and q_2 are $(k+1)$-equivalent. In this way, Q_i^k is further divided into $(k+1)$-equivalence classes. Repeat this for every Q_i^k in π_k to get all the elements of π_{k+1}.

Step 3 Construct π_n for $n = 1, 2, \ldots$ until $\pi_n = \pi_{n+1}$.

Step 4 (Construction of minimum automaton). For the required minimum state automaton, the states are the equivalence classes obtained in step 3, i.e. the elements of π_n. The state table is obtained by replacing a state q by the corresponding equivalence class $[q]$.

Remark In the above construction, the crucial part is the construction of equivalence classes; for, after getting the equivalence classes, the table for minimum automaton is obtained by replacing states by the corresponding equivalence classes. The number of equivalence classes is less than or equal to $|Q|$. Consider an equivalence class $[q_1] = \{q_1, q_2, \ldots, q_k\}$. If q_1 is reached while processing $w_1w_2 \in T(M)$ with $\delta(q_0, w_1) = q_1$, then $\delta(q_1, w_2) \in F$. So, $\delta(q_i, w_2) \in F$ for $i = 2, \ldots, k$. Thus we see that q_1, $i = 2, \ldots, k$ is reached on processing some $w \in T(M)$ iff q_1 is reached on processing w, i.e. q_1 of $[q_1]$ can play the role of $q_2, \ldots, q_k$. The above argument explains why we replace a state by the corresponding equivalence class.

Note: The construction of π_0, π_1, π_2, etc. is easy when the transition table is given. $\pi_0 = \{Q_1^0, Q_2^0\}$, where $Q_1^0 = F$ and $Q_2^0 = Q - F$. The subsets in π_1 are obtained by further partitioning the subsets of π_0. If $q_1, q_2 \in Q_1^0$, consider the states in each a-column, where $a \in \Sigma$ corresponding to q_1 and q_2. If they are in the same subset of π_0, q_1 and q_2 are 1-equivalent. If the states under some a-column are in different subsets of π_0, then q_1 and q_2 are not 1-equivalent. In general, $(k+1)$-equivalent states are obtained by applying the above method for q_1 and q_2 in Q_i^k.

EXAMPLE 3.13

Construct a minimum state automaton equivalent to the finite automaton described by Fig. 3.12.

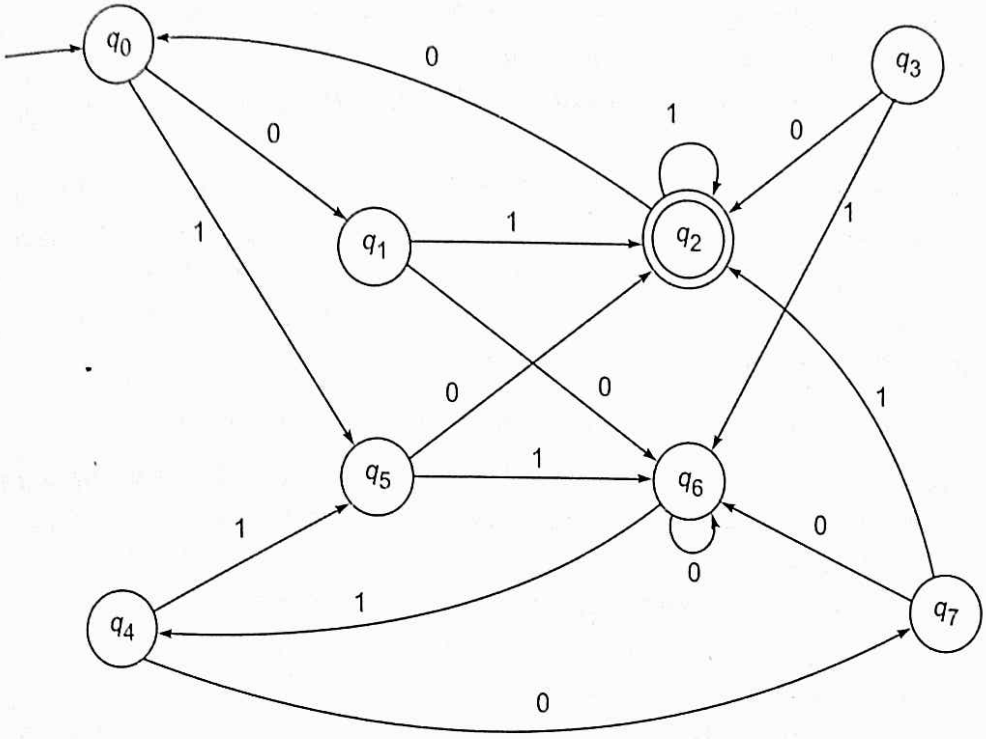

Fig. 3.12 Finite automaton of Example 3.13.

Solution

It will be easier if we construct the transition table as shown in Table 3.21.

TABLE 3.21 Transition Table for Example 3.13

State/Σ	0	1
$\rightarrow q_0$	q_1	q_5
q_1	q_6	q_2
(q_2)	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

By applying step 1, we get

$$Q_1^0 = F = \{q_2\}, \qquad Q_2^0 = Q - Q_1^0$$

So,

$$\pi_0 = \{\{q_2\}, \{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}\}$$

The $\{q_2\}$ in π_0 cannot be further partitioned. So, $Q'_1 = \{q_2\}$. Consider q_0 and $q_1 \in Q_2^0$. The entries under the 0-column corresponding to q_0 and q_1 are q_1 and q_6; they lie in Q_2^0. The entries under the 1-column are q_5 and q_2. $q_2 \in Q_1^0$ and $q_5 \in Q_2^0$. Therefore, q_0 and q_1 are not 1-equivalent. Similarly, q_0 is not 1-equivalent to q_3, q_5 and q_7.

Now, consider q_0 and q_4. The entries under the 0-column are q_1 and q_7. Both are in Q_2^0. The entries under the 1-column are q_5, q_5. So q_4 and q_0 are 1-equivalent. Similarly, q_0 is 1-equivalent to q_6. $\{q_0, q_4, q_6\}$ is a subset in π_1. So, $Q'_2 = \{q_0, q_4, q_6\}$.

Repeat the construction by considering q_1 and any one of the states q_3, q_5, q_7. Now, q_1 is not 1-equivalent to q_3 or q_5 but 1-equivalent to q_7. Hence, $Q'_3 = \{q_1, q_7\}$. The elements left over in Q_2^0 are q_3 and q_5. By considering the entries under the 0-column and the 1-column, we see that q_3 and q_5 are 1-equivalent. So $Q'_4 = \{q_3, q_5\}$. Therefore,

$$\pi_1 = \{\{q_2\}, \{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$$

The $\{q_2\}$ is also in π_2 as it cannot be further partitioned. Now, the entries under the 0-column corresponding to q_0 and q_4 are q_1 and q_7, and these lie in the same equivalence class in π_1. The entries under the 1-column are q_5, q_5. So q_0 and q_4 are 2-equivalent. But q_0 and q_6 are not 2-equivalent. Hence, $\{q_0, q_4, q_6\}$ is partitioned into $\{q_0, q_4\}$ and $\{q_6\}$. q_1 and q_7 are 2-equivalent. q_3 and q_5 are also 2-equivalent. Thus, $\pi_2 = \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$. q_0 and q_4 are 3-equivalent. The q_1 and q_7 are 3-equivalent. Also, q_3 and q_5 are 3-equivalent. Therefore,

$$\pi_3 = \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$$

As $\pi_2 = \pi_3$, π_2 gives us the equivalence classes, the minimum state automaton is

$$M' = (Q', \{0, 1\}, \delta', q'_0, F')$$

where

$$Q' = \{[q_2], [q_0, q_4], [q_6], [q_1, q_7], [q_3, q_5]\}$$

$$q'_0 = [q_0, q_4], \qquad F' = [q_2]$$

and δ' is defined by Table 3.22.

TABLE 3.22 Transition Table of Minimum State Automaton for Example 3.13

State/Σ	0	1
$[q_0, q_4]$	$[q_1, q_7]$	$[q_3, q_5]$
$[q_1, q_7]$	$[q_6]$	$[q_2]$
$[q_2]$	$[q_0, q_4]$	$[q_2]$
$[q_3, q_5]$	$[q_2]$	$[q_6]$
$[q_6]$	$[q_6]$	$[q_0, q_4]$

Note: The transition diagram for the minimum state automaton is described by Fig. 3.13. The states q_0 and q_4 are identified and treated as one state. (So also are q_1, q_7 and q_3, q_5.) But the transitions in both the diagrams (i.e. Figs. 3.12 and 3.13) are the same. If there is an arrow from q_i to q_j with label a, then there is an arrow from $[q_i]$ to $[q_j]$ with the same label in the diagram for minimum state automaton. Symbolically, if $\delta(q_i, a) = q_j$, then $\delta'([q_i], a) = [q_j]$.

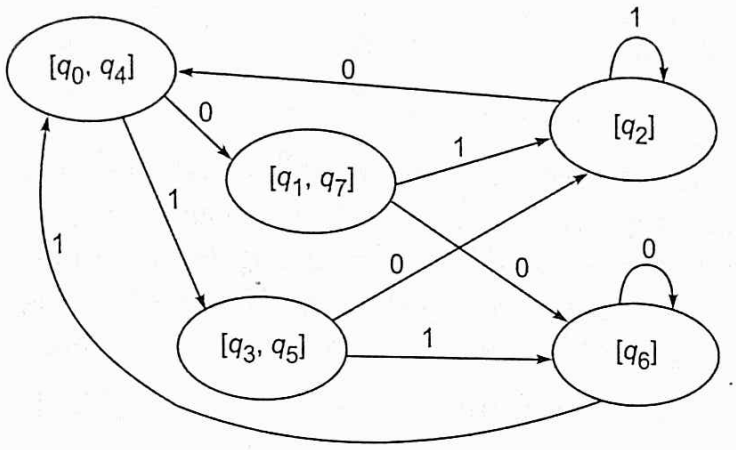

Fig. 3.13 Minimum state automaton of Example 3.13.

EXAMPLE 3.14

Construct the minimum state automaton equivalent to the transition diagram given by Fig. 3.14.

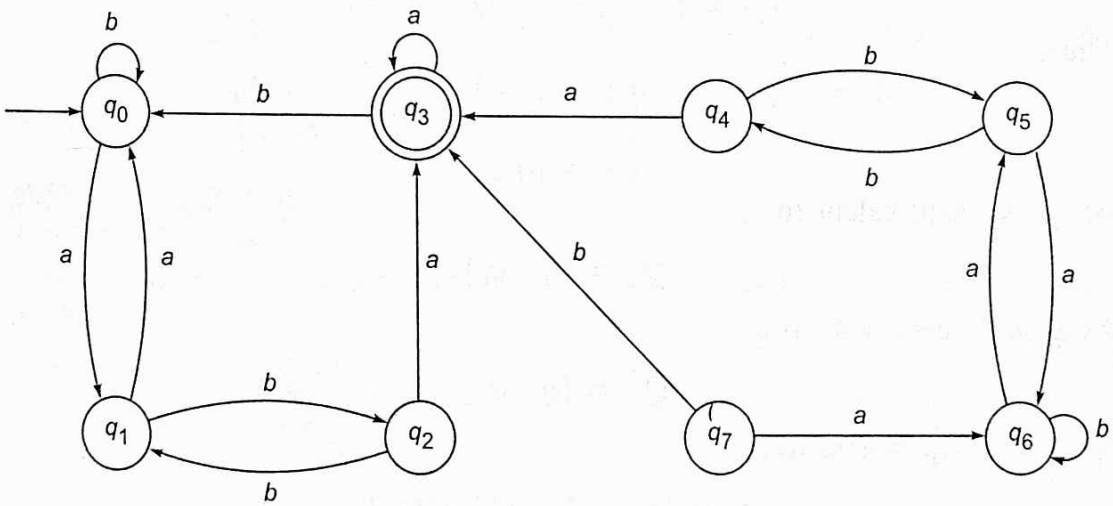

Fig. 3.14 Finite automaton of Example 3.14.

Solution

We construct the transition table as given by Table 3.23.

TABLE 3.23 Transition Table for Example 3.14

State/Σ	a	b
→ q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
(q_3)	q_3	q_0
q_4	q_3	q_5
q_5	q_6	q_4
q_6	q_5	q_6
q_7	q_6	q_3

Since there is only one final state q_3, $Q_1^0 = \{q_3\}$, $Q_2^0 = Q - Q_1^0$. Hence, $\pi_0 = \{\{q_3\}, \{q_0, q_1, q_2, q_4, q_5, q_6, q_7\}\}$. As $\{q_3\}$ cannot be partitioned further, $Q'_1 = \{q_3\}$. Now q_0 is 1-equivalent to q_1, q_5, q_6, but not to q_2, q_4, q_7, and so $Q'_2 = \{q_0, q_1, q_5, q_6\}$. q_2 is 1-equivalent to q_4. Hence, $Q'_3 = \{q_2, q_4\}$. The only element remaining in Q_2^0 is q_7. Therefore, $Q'_4 = \{q_7\}$. Thus,

$$\pi_1 = \{\{q_3\}, \{q_0, q_1, q_5, q_6\}, \{q_2, q_4\}, \{q_7\}\}$$

$$Q_1^2 = \{q_3\}$$

q_0 is 2-equivalent to q_6 but not to q_1 or q_5. So,

$$Q_2^2 = \{q_0, q_6\}$$

As q_1 is 2-equivalent to q_5,

$$Q_3^2 = \{q_1, q_5\}$$

As q_2 is 2-equivalent to q_4,

$$Q_4^2 = \{q_2, q_4\}, \qquad Q_5^2 = \{q_7\}$$

Thus,

$$\pi_2 = \{\{q_3\}, \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}, \{q_7\}\}$$

$$Q_1^3 = \{q_3\}$$

As q_0 is 3-equivalent to q_6,

$$Q_2^3 = \{q_0, q_6\}$$

As q_1 is 3-equivalent to q_5,

$$Q_3^3 = \{q_1, q_5\}$$

As q_2 is 3-equivalent to q_4,

$$Q_4^3 = \{q_2, q_4\}, \quad Q_5^3 = \{q_7\}$$

Therefore,

$$\pi_3 = \{\{q_3\}, \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}, \{q_7\}\}$$

As $\pi_3 = \pi_2$, π_2 gives us the equivalence classes, the minimum state automaton is

$$M' = (Q', \{a, b\}, \delta', q'_0, F')$$

where

$$Q' = \{[q_3],\ [q_0,\ q_6],\ [q_1,\ q_5],\ [q_2,\ q_4],\ [q_7]\}$$

$$q_0' = [q_0,\ q_6], \qquad F' = [q_3]$$

and δ' is defined by Table 3.24.

TABLE 3.24 Transition Table of Minimum State Automaton for Example 3.14

State/Σ	a	b
$[q_0, q_6]$	$[q_1, q_5]$	$[q_0, q_6]$
$[q_1, q_5]$	$[q_0, q_6]$	$[q_2, q_4]$
$[q_2, q_4]$	$[q_3]$	$[q_1, q_5]$
$[q_3]$	$[q_3]$	$[q_0, q_6]$
$[q_7]$	$[q_0, q_6]$	$[q_3]$

Note: The transition diagram for M' is given by Fig. 3.15.

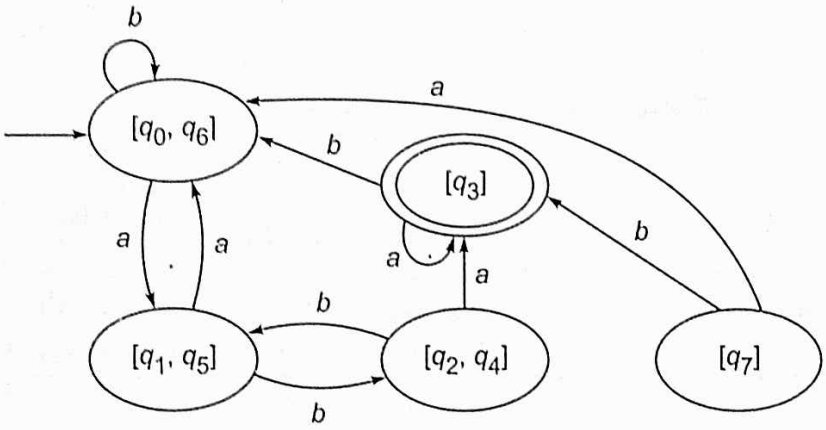

Fig. 3.15 Minimum state automaton of Example 3.14.

3.10 SUPPLEMENTARY EXAMPLES

EXAMPLE 3.15

Construct a DFA equivalent to the NDFA M whose transition diagram is given by Fig. 3.16.

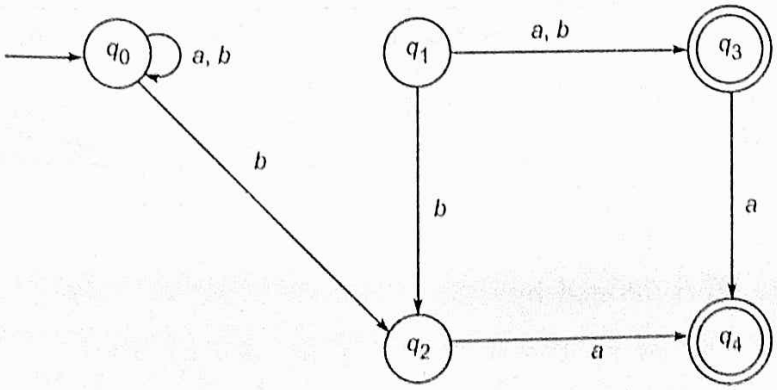

Fig. 3.16 NDFA of Example 3.15.

Solution

The transition table of M is given by Table 3.25.

TABLE 3.25 Transition Table for Example 3.15

State	a	b
$\rightarrow q_0$	q_0	q_0, q_2
q_1	q_3	q_2, q_3
q_2	q_4	—
(q_3)	q_4	—
(q_4)	—	—

For the equivalent DFA:

(i) The states are subsets of $Q = \{q_0, q_1, q_2, q_3, q_4\}$.
(ii) $[q_0]$ is the initial state.
(iii) The subsets of Q containing q_3 or q_4 are the final states.
(iv) δ is defined by Table 3.26. We start from $[q_0]$ and construct δ, only for those states reachable from $[q_0]$ (as in Example 3.8).

TABLE 3.26 Transition Table of DFA for Example 3.15

State	a	b
$[q_0]$	$[q_0]$	$[q_0, q_2]$
$[q_0, q_2]$	$[q_0, q_4]$	$[q_0, q_2]$
$[q_0, q_4]$	$[q_0]$	$[q_0, q_2]$

EXAMPLE 3.16

Construct a DFA equivalent to an NDFA whose transition table is defined by Table 3.27.

TABLE 3.27 Transition Table of NDFA for Example 3.16

State	a	b
q_0	q_1, q_3	q_2, q_3
q_1	q_1	q_3
q_2	q_3	q_2
(q_3)	—	—

Solution

Let M be the DFA defined by

$$M = (2^{\{q_0, q_1, q_2, q_3\}}, \{a, b\}, \delta, [q_0], F)$$

where F is the set of all subsets of $\{q_0, q_1, q_2, q_3\}$ containing q_3. δ is defined by Table 3.28.

TABLE 3.28 Transition Table of DFA for Example 3.16

State	a	b
$[q_0]$	$[q_1, q_3]$	$[q_2, q_3]$
$[q_1, q_3]$	$[q_1]$	$[q_3]$
$[q_1]$	$[q_1]$	$[q_3]$
$[q_3]$	$\emptyset$	$\emptyset$
$[q_2, q_3]$	$[q_3]$	$[q_2]$
$[q_2]$	$[q_3]$	$[q_2]$
$\emptyset$	$\emptyset$	$\emptyset$

EXAMPLE 3.17

Construct a DFA accepting all strings w over $\{0, 1\}$ such that the number of 1's in w is 3 mod 4.

Solution

Let M be the required NDFA. As the condition on strings of $T(M)$ does not at all involve 0, we can assume that M does not change state on input 0. If 1 appears in w $(4k + 3)$ times, M can come back to the initial state, after reading 4 1's and to a final state after reading 3 1's.

The required DFA is given by Fig. 3.17.

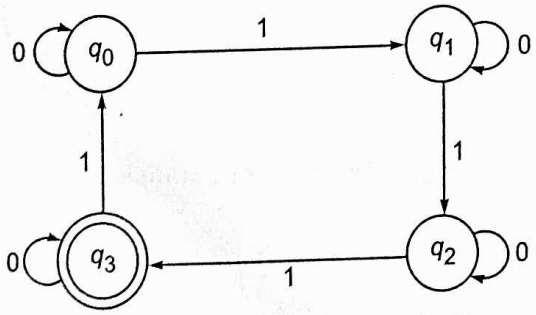

Fig. 3.17 DFA of Example 3.17.

EXAMPLE 3.18

Construct a DFA accepting all strings over $\{a, b\}$ ending in ab.

Solution

We require two transitions for accepting the string ab. If the symbol b is processed after aa or ba, then also we end in ab. So we can have states for

remembering *aa*, *ab*, *ba*, *bb*. The state corresponding to *ab* can be the final state in our DFA. Keeping these in mind we construct the required DFA. Its transition diagram is described by Fig. 3.18.

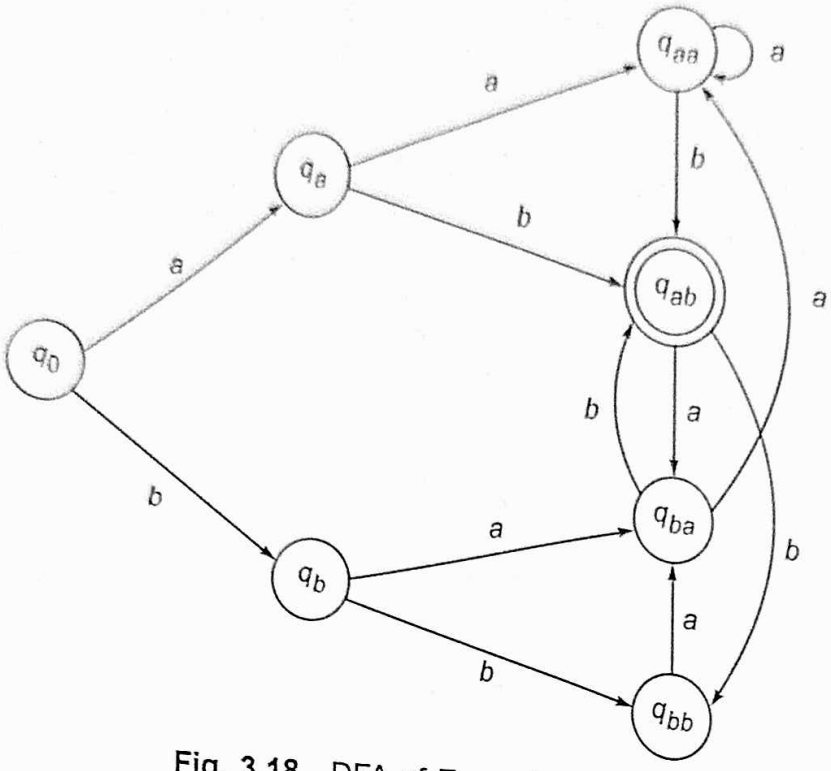

Fig. 3.18 DFA of Example 3.18.

EXAMPLE 3.19

Find $T(M)$ for the DFA M described by Fig. 3.19.

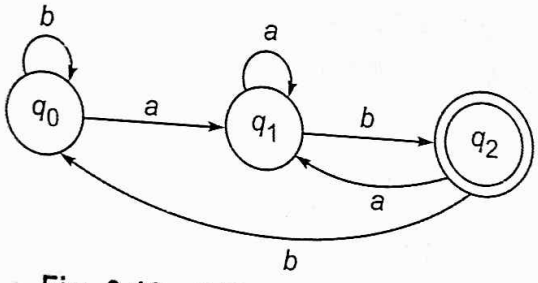

Fig. 3.19 DFA of Example 3.19.

Solution

$T(M) = \{w \in \{a, b\}^* \mid w$ ends in the substring $ab\}$

Note: If we apply the minimization algorithm to the DFA in Example 3.18, we get the DFA as in Example 3.19. (The student is advised to check.)

EXAMPLE 3.20

Construct a minimum state automaton equivalent to an automaton whose transition table is defined by Table 3.29.

TABLE 3.29 DFA of Example 3.20

State	a	b
$\rightarrow q_0$	q_1	q_2
q_1	q_1	q_3
q_2	q_3	q_4
q_3	q_1	q_5
q_4	q_4	q_2
(q_5)	q_5	q_5

Solution

$Q_1^0 = \{q_5\}$, $Q_2^0 = \{q_0, q_1, q_2, q_3, q_4\}$. So, $\pi_0 = \{\{q_5\}, \{q_0, q_1, q_2, q_3, q_4\}\}$
Q_1^0 cannot be partitioned further. So $\{q_5\} \in \pi_1$. Consider Q_2^0. q_0 is equivalent to q_1, q_2 and q_4. But q_0 is not equivalent to q_3 since $\delta(q_0, b) = q_2$ and $\delta(q_3, b) = \{q_5\}$.
Hence,

$$Q_1^1 = \{q_5\}, \quad Q_2^1 = \{q_0, q_1, q_2, q_4\}, \quad Q_3^1 = \{q_3\}$$

Therefore,

$$\pi_1 = \{\{q_5\}, \{q_0, q_1, q_2, q_4\}, \{q_3\}\}$$

q_0 is 2-equivalent to q_4 but not 2-equivalent to q_1 or q_2.
Hence,

$$\{q_0, q_4\} \in \pi_2$$

q_1 and q_2 are not 2-equivalent.
Therefore,

$$\pi_2 = \{\{q_5\}, \{q_3\}, \{q_0, q_4\}, \{q_1\}, \{q_2\}\}$$

As q_0 is not 3-equivalent to q_4, $\{q_0, q_4\}$ is further partitioned into $\{q_1\}$ and $\{q_4\}$.
So,

$$\pi_3 = \{\{q_0\}, \{q_1\}, \{q_2\}, \{q_3\}, \{q_4\}, \{q_5\}\}$$

Hence the minimum state automaton M' is the same as the given M.

EXAMPLE 3.21

Construct a minimum state automaton equivalent to a DFA whose transition table is defined by Table 3.30.

TABLE 3.30 DFA of Example 3.21

State	a	b
$\rightarrow q_0$	q_1	q_2
q_1	q_4	q_3
q_2	q_4	q_3
(q_3)	q_5	q_6
(q_4)	q_7	q_6
q_5	q_3	q_6
q_6	q_6	q_6
q_7	q_4	q_6

Solution

$$Q_1^0 = \{q_3, q_4\}, \quad Q_2^0 = \{q_0, q_1, q_2, q_5, q_6, q_7\}$$

$$\pi_0 = \{\{q_3, q_4\}, \{q_0, q_1, q_2, q_5, q_6, q_7\}\}$$

q_3 is 1-equivalent to q_4. So, $\{q_3, q_4\} \in \pi_1$.

q_0 is not 1-equivalent to q_1, q_2, q_5 but q_0 is 1-equivalent to q_6.

Hence $\{q_0, q_6\} \in \pi_1$. q_1 is 1-equivalent to q_2 but not 1-equivalent to q_5, q_6 or q_7. So, $\{q_1, q_2\} \in \pi_1$.

q_5 is not 1-equivalent to q_6 but to q_7. So, $\{q_5, q_7\} \in \pi_1$

Hence,

$$\pi_1 = \{\{q_3, q_4\}, \{q_0, q_6\}, \{q_1, q_2\}, \{q_5, q_7\}\}$$

q_3 is 2-equivalent to q_4. So, $\{q_3, q_4\} \in \pi_2$.

q_0 is not 2-equivalent to q_6. So, $\{q_0\}, \{q_6\} \in \pi_2$.

q_1 is 2-equivalent to q_2. So, $\{q_1, q_2\} \in \pi_2$.

q_5 is 2-equivalent to q_7. So, $\{q_5, q_7\} \in \pi_2$.

Hence,

$$\pi_2 = \{\{q_3, q_4\}, \{q_0\}, \{q_6\}, \{q_1, q_2\}, \{q_5, q_7\}\}$$

q_3 is 3-equivalent to q_4; q_1 is 3-equivalent to q_2 and q_5 is 3-equivalent to q_7. Hence,

$$\pi_3 = \{\{q_0\}, \{q_1, q_2\}, \{q_3, q_4\}, \{q_5, q_7\}, \{q_6\}\}$$

As $\pi_3 = \pi_2$, the minimum state automaton is

$$M' = (Q', \{a, b\}, \delta', [q_0], \{[q_3, q_4]\})$$

where δ' is defined by Table 3.31.

TABLE 3.31 Transition Table of DFA for Example 3.21

State	a	b
$[q_0]$	$[q_1, q_2]$	$[q_1, q_2]$
$[q_1, q_2]$	$[q_3, q_4]$	$[q_3, q_4]$
$[q_3, q_4]$	$[q_5, q_7]$	$[q_6]$
$[q_5, q_7]$	$[q_3, q_4]$	$[q_6]$
$[q_6]$	$[q_6]$	$[q_6]$

SELF-TEST

Study the automaton given in Fig. 3.20 and choose the correct answers to Questions 1–5:

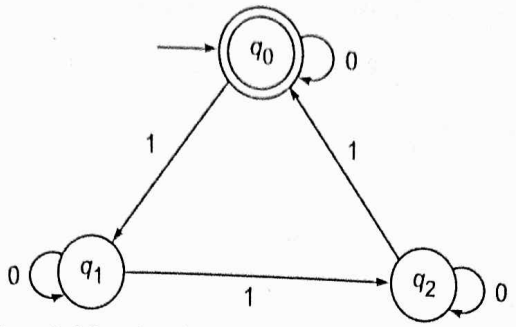

Fig. 3.20 Automaton for Questions 1–5.

1. M is a
   (a) nondeterministic automaton
   (b) deterministic automaton accepting $\{0, 1\}^*$
   (c) deterministic automaton accepting all strings over $\{0, 1\}$ having $3m$ 0's and $3n$ 1's, $m, n \geq 1$
   (d) deterministic automaton

2. M accepts
   (a) 01110 (b) 10001 (c) 01010 (d) 11111

3. $T(M)$ is equal to
   (a) $\{0^{3m}\ 1^{3n} \mid m, n \geq 0\}$
   (b) $\{0^{3m}\ 1^{3n} \mid m, n \geq 1\}$
   (c) $\{w \mid w$ has 111 as a substring$\}$
   (d) $\{w \mid w$ has $3n$ 1's, $n \geq 1\}$

4. If q_2 is also made a final state, then M accepts
   (a) 01110 and 01100
   (b) 10001 and 10000
   (c) 0110 but not 0111101
   (d) 0^{3n}, $n \geq 1$ but not 1^{3n}, $n \geq 1$

5. If q_2 is also made a final state, then $T(M)$ is equal to
   (a) $\{0^{3m}\ 1^{3n} \mid m, n \geq 0\} \cup \{0^{2m}\ 1^{n} \mid m, n \geq 0\}$
   (b) $\{0^{3m}\ 1^{3n} \mid m, n \geq 1\} \cup \{0^{2m}\ 1^{n} \mid m, n \geq 1\}$
   (c) $\{w \mid w$ has 111 as a substring or 11 as a substring$\}$
   (d) $\{w \mid$ the number of 1's in w is divisible by 2 or 3$\}$

Study the automaton given in Fig. 3.21 and state whether the Statements 6–15 are true or false:

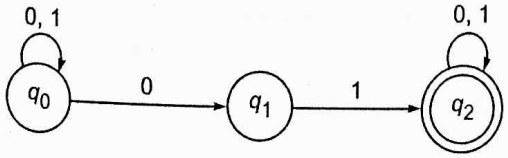

Fig. 3.21 Automaton for Statements 6–15.

6. M is a nondeterministic automaton.
7. $\delta(q_1, 1)$ is defined.
8. 0100111 is accepted by M.
9. 010101010 is not accepted by M.
10. $\delta(q_0, 01001) = \{q_1\}$.
11. $\delta(q_0, 011000) = \{q_0, q_1, q_2\}$.
12. $\delta(q_2, w) = q_2$ for any string $w \in \{0, 1\}^*$.
13. $\delta(q_1, 11001) \neq \emptyset$.
14. $T(M) = \{w \mid w = x00y$, where $x, y \in \{0, 1\}^*\}$.
15. A string having an even number of 0's is accepted by M.

EXERCISES

3.1 For the finite state machine M described by Table 3.1, find the strings among the following strings which are accepted by M: (a) 101101, (b) 11111, (c) 000000.

3.2 For the transition system M described by Fig. 3.8, obtain the sequence of states for the input sequence 000101. Also, find an input sequence not accepted by M.

3.3 Test whether 110011 and 110110 are accepted by the transition system described by Fig. 3.6.

3.4 Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton. Let R be a relation in Q defined by $q_1 R q_2$ if $\delta(q_1, a) = \delta(q_2, a)$ for all $a \in \Sigma$. Is R an equivalence relation?

3.5 Construct a nondeterministic finite automaton accepting $\{ab, ba\}$, and use it to find a deterministic automaton accepting the same set.

3.6 Construct a nondeterministic finite automaton accepting the set of all strings over $\{a, b\}$ ending in aba. Use it to construct a DFA accepting the same set of strings.

3.7 The transition table of a nondeterministic finite automaton M is defined by Table 3.32. Construct a deterministic finite automaton equivalent to M.

TABLE 3.32 Transition Table for Exercise 3.7

State	0	1	2
$\rightarrow q_0$	$q_1 q_4$	q_4	$q_2 q_3$
q_1		q_4	
q_2			$q_2 q_3$
(q_3)		q_4	
q_4			

3.8 Construct a DFA equivalent to the NDFA described by Fig. 3.8.

3.9 $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \delta, q_1, \{q_3\})$ is a nondeterministic finite automaton, where δ is given by

$$\delta(q_1, 0) = \{q_2, q_3\}, \qquad \delta(q_1, 1) = \{q_1\}$$
$$\delta(q_2, 0) = \{q_1, q_2\}, \qquad \delta(q_2, 1) = \emptyset$$
$$\delta(q_3, 0) = \{q_2\}, \qquad \delta(q_3, 1) = \{q_1, q_2\}$$

Construct an equivalent DFA.

3.10 Construct a transition system which can accept strings over the alphabet $a, b, \ldots$ containing either *cat* or *rat*.

3.11 Construct a Mealy machine which is equivalent to the Moore machine defined by Table 3.33.

TABLE 3.33 Moore Machine of Exercise 3.11

Present state	*Next state*		*Output*
	$a = 0$	$a = 1$	
$\rightarrow q_0$	q_1	q_2	1
q_1	q_3	q_2	0
q_2	q_2	q_1	1
q_3	q_0	q_3	1

3.12 Construct a Moore machine equivalent to the Mealy machine M defined by Table 3.34.

TABLE 3.34 Mealy Machine of Exercise 3.12

Present state	*Next state*			
	$a = 0$		$a = 1$	
	state	*output*	*state*	*output*
$\rightarrow q_1$	q_1	1	q_2	0
q_2	q_4	1	q_4	1
q_3	q_2	1	q_3	1
q_4	q_3	0	q_1	1

3.13 Construct a Mealy machine which can output EVEN, ODD according as the total number of 1's encountered is even or odd. The input symbols are 0 and 1.

3.14 Construct a minimum state automaton equivalent to a given automaton M whose transition table is defined by Table 3.35.

TABLE 3.35 Finite Automaton of Exercise 3.14

State	Input	
	a	b
$\rightarrow q_0$	q_0	q_3
q_1	q_2	q_5
q_2	q_3	q_4
q_3	q_0	q_5
q_4	q_0	q_6
q_5	q_1	q_4
(q_6)	q_1	q_3

3.15 Construct a minimum state automaton equivalent to the DFA described by Fig. 3.18. Compare it with the DFA described by Fig. 3.19.

3.16 Construct a minimum state automaton equivalent to the DFA described by Fig. 3.22.

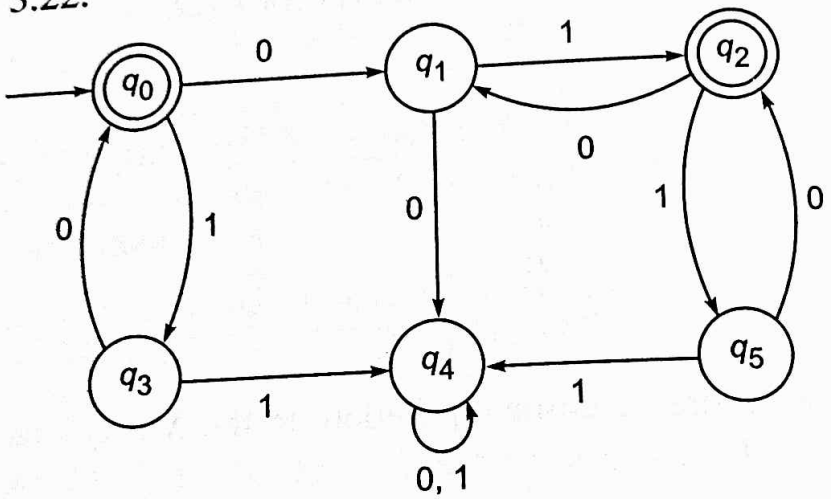

Fig. 3.22 DFA of Exercise 3.16.

4 Formal Languages

In this chapter we introduce the concepts of grammars and formal languages and discuss the Chomsky classification of languages. We also study the inclusion relation between the four classes of languages. Finally, we discuss the closure properties of these classes under the various operations.

4.1 BASIC DEFINITIONS AND EXAMPLES

The theory of formal languages is an area with a number of applications in computer science. Linguists were trying in the early 1950s to define precisely valid sentences and give structural descriptions of sentences. They wanted to define a formal grammar (i.e. to describe the rules of grammar in a rigorous mathematical way) to describe English. They thought that such a description of natural languages (the languages that we use in everyday life such as English, Hindi, French, etc.) would make language translation using computers easy. It was Noam Chomsky who gave a mathematical model of a grammar in 1956. Although it was not useful for describing natural languages such as English, it turned out to be useful for computer languages. In fact, the Backus-Naur form used to describe ALGOL followed the definition of grammar (a context-free grammar) given by Chomsky.

Before giving the definition of grammar, we shall study, for the sake of simplicity, two types of sentences in English with a view to formalising the construction of these sentences. The sentences we consider are those with a noun and a verb, or those with a noun–verb and adverb (such as 'Ram ate quickly' or 'Sam ran'). The sentence 'Ram ate quickly' has the words 'Ram', 'ate', 'quickly' written in that order. If we replace 'Ram' by 'Sam', 'Tom', 'Gita', etc. i.e. by any noun, 'ate' by 'ran', 'walked', etc. i.e. by any verb in the

past tense, and 'quickly' by 'slowly', i.e. by any adverb, we get other grammatically correct sentences. So the structure of 'Ram ate quickly' can be given as ⟨noun⟩ ⟨verb⟩ ⟨adverb⟩. For ⟨noun⟩ we can substitute 'Ram', 'Sam', 'Tom', 'Gita', etc. . Similarly, we can substitute 'ate', 'walked', 'ran', etc. for ⟨verb⟩, and 'quickly', 'slowly' for ⟨adverb⟩. Similarly, the structure of 'Sam ran' can be given in the form ⟨noun⟩ ⟨verb⟩.

We have to note that ⟨noun⟩ ⟨verb⟩ ⟨adverb⟩ is not a sentence but only the description of a particular type of sentence. If we replace ⟨noun⟩, ⟨verb⟩ and ⟨adverb⟩ by suitable words, we get actual grammatically correct sentences. Let us call ⟨noun⟩, ⟨verb⟩, ⟨adverb⟩ as variables. Words like 'Ram', 'Sam', 'ate', 'ran', 'quickly', 'slowly' which form sentences can be called terminals. So our sentences turn out to be strings of terminals. Let S be a variable denoting a sentence. Now, we can form the following rules to generate two types of sentences:

$S \rightarrow$ ⟨noun⟩ ⟨verb⟩ ⟨adverb⟩
$S \rightarrow$ ⟨noun⟩ ⟨verb⟩
⟨noun⟩ $\rightarrow$ Sam
⟨noun⟩ $\rightarrow$ Ram
⟨noun⟩ $\rightarrow$ Gita
⟨verb⟩ $\rightarrow$ ran
⟨verb⟩ $\rightarrow$ ate
⟨verb⟩ $\rightarrow$ walked
⟨adverb⟩ $\rightarrow$ slowly
⟨adverb⟩ $\rightarrow$ quickly

(Each arrow represents a rule meaning that the word on the right side of the arrow can replace the word on the left side of the arrow.) Let us denote the collection of the rules given above by P.

If our vocabulary is thus restricted to 'Ram', 'Sam', 'Gita', 'ate', 'ran', 'walked', 'quickly' and 'slowly', and our sentences are of the form ⟨noun⟩ ⟨verb⟩ ⟨adverb⟩ and ⟨noun⟩ ⟨verb⟩, we can describe the grammar by a 4-tuple (V_N, Σ, P, S), where

V_N = {⟨noun⟩, ⟨verb⟩, ⟨adverb⟩}

Σ = {Ram, Sam, Gita, ate, ran, walked, quickly, slowly}

P is the collection of rules described above (the rules may be called productions),

S is the special symbol denoting a sentence.

The sentences are obtained by (i) starting with S, (ii) replacing words using the productions, and (iii) terminating when a string of terminals is obtained.

With this background we can give the definition of a grammar. As mentioned earlier, this definition is due to Noam Chomsky.

4.1.1 DEFINITION OF A GRAMMAR

Definition 4.1 A phrase-structure grammar (or simply a grammar) is (V_N, Σ, P, S), where

(i) V_N is a finite nonempty set whose elements are called variables,
(ii) Σ is a finite nonempty set whose elements are called terminals,
(iii) $V_N \cap \Sigma = \emptyset$,
(iv) S is a special variable (i.e. an element of V_N) called the start symbol, and
(v) P is a finite set whose elements are $\alpha \to \beta$, where α and β are strings on $V_N \cup \Sigma$. α has at least one symbol from V_N. The elements of P are called productions or production rules or rewriting rules.

Note: The set of productions is the kernel of grammars and language specification. We observe the following regarding the production rules.

(i) Reverse substitution is not permitted. For example, if $S \to AB$ is a production, then we can replace S by AB, but we cannot replace AB by S.
(ii) No inversion operation is permitted. For example, if $S \to AB$ is a production, it is not necessary that $AB \to S$ is a production.

EXAMPLE 4.1

$$G = (V_N, \Sigma, P, S) \text{ is a grammar}$$

where

$V_N = \{\langle\text{sentence}\rangle, \langle\text{noun}\rangle, \langle\text{verb}\rangle, \langle\text{adverb}\rangle\}$
$\Sigma = \{\text{Ram, Sam, ate, sang, well}\}$
$S = \langle\text{sentence}\rangle$

P consists of the following productions:

$\langle\text{sentence}\rangle \to \langle\text{noun}\rangle\ \langle\text{verb}\rangle$
$\langle\text{sentence}\rangle \to \langle\text{noun}\rangle\ \langle\text{verb}\rangle\ \langle\text{adverb}\rangle$
$\langle\text{noun}\rangle \to \text{Ram}$
$\langle\text{noun}\rangle \to \text{Sam}$
$\langle\text{verb}\rangle \to \text{ate}$
$\langle\text{verb}\rangle \to \text{sang}$
$\langle\text{adverb}\rangle \to \text{well}$

NOTATION: (i) If A is any set, then A^* denotes the set of all strings over A. A^+ denotes $A^* - \{\Lambda\}$, where Λ is the empty string.
(ii) $A, B, C, A_1, A_2, \ldots$ denote the variables.
(iii) $a, b, c, \ldots$ denote the terminals.
(iv) $x, y, z, w, \ldots$ denote the strings of terminals.
(v) $\alpha, \beta, \gamma, \ldots$ denote the elements of $(V_N \cup \Sigma)^*$.
(vi) $X^0 = \Lambda$ for any symbol X in $V_N \cup \Sigma$.

4.1.2 Derivations and the Language Generated by a Grammar

Productions are used to derive one string over $V_N \cup \Sigma$ from another string. We give a formal definition of derivation as follows:

Definition 4.2 If $\alpha \to \beta$ is a production in a grammar G and γ, δ are any two strings on $V_N \cup \Sigma$, then we say that $\gamma\alpha\delta$ directly derives $\gamma\beta\delta$ in G (we write this as $\gamma\alpha\delta \underset{G}{\Rightarrow} \gamma\beta\delta$). This process is called one-step derivation. In particular, if $\alpha \to \beta$ is a production, then $\alpha \underset{G}{\Rightarrow} \beta$.

Note: If α is a part of a string and $\alpha \to \beta$ is a production, we can replace α by β in that string (without altering the remaining parts). In this case we say that the string we started with directly derives the new string.

For example,

$$G = (\{S\}, \{0, 1\}, \{S \to 0S1, S \to 01\}, S)$$

has the production $S \to 0S1$. So, S in 0^4S1^4 can be replaced by $0S1$. The resulting string is 0^40S11^4. Thus, we have $0^4S1^4 \underset{G}{\Rightarrow} 0^40S11^4$.

Note: $\underset{G}{\Rightarrow}$ induces a relation R on $(V_N \cup \Sigma)^*$, i.e. $\alpha R \beta$ if $\alpha \underset{G}{\Rightarrow} \beta$.

Definition 4.3 If α and β are strings on $V_N \cup \Sigma$, then we say that α derives β if $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$. Here $\overset{*}{\underset{G}{\Rightarrow}}$ denotes the reflexive-transitive closure of the relation $\underset{G}{\Rightarrow}$ in $(V_N \cup \Sigma)^*$ (refer to Section 2.1.5).

Note: We can note in particular that $\alpha \overset{*}{\underset{G}{\Rightarrow}} \alpha$. Also, if $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$, $\alpha \neq \beta$, then there exist strings $\alpha_1, \alpha_2, \ldots, \alpha_n$, where $n \geq 2$ such that

$$\alpha = \alpha_1 \underset{G}{\Rightarrow} \alpha_2 \underset{G}{\Rightarrow} \alpha_3 \cdots \underset{G}{\Rightarrow} \alpha_n = \beta$$

When $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$ is in n steps, we write $\alpha \overset{n}{\underset{G}{\Rightarrow}} \beta$.

Consider, for example, $G = (\{S\}, \{0, 1\}, \{S \to 0S1, S \to 01\}, S)$.

As $S \underset{G}{\Rightarrow} 0S1 \underset{G}{\Rightarrow} 0^2S1^2 \underset{G}{\Rightarrow} 0^3S1^3$, we have $S \overset{*}{\underset{G}{\Rightarrow}} 0^3S1^3$. We also have $0^3S1^3 \overset{*}{\underset{G}{\Rightarrow}} 0^3S1^3$ (as $\alpha \overset{*}{\underset{G}{\Rightarrow}} \alpha$).

Definition 4.4 The language generated by a grammar G (denoted by $L(G)$) is defined as $\{w \in \Sigma^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w\}$. The elements of $L(G)$ are called *sentences*.

Stated in another way, $L(G)$ is the set of all terminal strings derived from the start symbol S.

Definition 4.5 If $S \overset{*}{\underset{G}{\Rightarrow}} \alpha$, then α is called a *sentential form*. We can note that the elements of $L(G)$ are sentential forms but not vice versa.

Definition 4.6 G_1 and G_2 are equivalent if $L(G_1) = L(G_2)$.

Remarks on Derivation

1. Any derivation involves the application of productions. When the number of times we apply productions is one, we write $\alpha \underset{G}{\Rightarrow} \beta$; when it is more than one, we write $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$ (*Note:* $\alpha \overset{*}{\underset{G}{\Rightarrow}} \alpha$).
2. The string generated by the most recent application of production is called the working string.
3. The derivation of a string is complete when the working string cannot be modified. If the final string does not contain any variable, it is a sentence in the language. If the final string contains a variable, it is a sentential form and in this case the production generator gets 'stuck'.

NOTATION: (i) We write $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$ simply as $\alpha \overset{*}{\Rightarrow} \beta$ if G is clear from the context.

(ii) If $A \to \alpha$ is a production where $A \in V_N$, then it is called an A-production.

(iii) If $A \to \alpha_1, A \to \alpha_2, \ldots, A \to \alpha_m$ are A-productions, these productions are written as $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_m$.

We give several examples of grammars and languages generated by them.

EXAMPLE 4.2

If $G = (\{S\}, \{0, 1\}, \{S \to 0S1, S \to \Lambda\}, S)$, find $L(G)$.

Solution

As $S \to \Lambda$ is a production, $S \underset{G}{\Rightarrow} \Lambda$. So Λ is in $L(G)$. Also, for all $n \geq 1$,

$$S \underset{G}{\Rightarrow} 0S1 \underset{G}{\Rightarrow} 0^2S1^2 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} 0^nS1^n \underset{G}{\Rightarrow} 0^n1^n$$

Therefore,

$$0^n1^n \in L(G) \text{ for } n \geq 0$$

(Note that in the above derivation, $S \to 0S1$ is applied at every step except in the last step. In the last step, we apply $S \to \Lambda$). Hence, $\{0^n1^n \mid n \geq 0\} \subseteq L(G)$.

To show that $L(G) \subseteq \{0^n1^n \mid n \geq 0\}$, we start with w in $L(G)$. The derivation of w starts with S. If $S \to \Lambda$ is applied first, we get Λ. In this case $w = \Lambda$. Otherwise the first production to be applied is $S \to 0S1$. At any stage if we apply $S \to \Lambda$, we get a terminal string. Also, the terminal string is obtained only by applying $S \to \Lambda$. Thus the derivation of w is of the form

$$S \overset{*}{\underset{G}{\Rightarrow}} 0^nS1^n \underset{G}{\Rightarrow} 0^n1^n \qquad \text{for some } n \geq 1$$

i.e.

$$L(G) \subseteq \{0^n1^n \mid n \geq 0\}$$

Therefore,

$$L(G) = \{0^n 1^n \mid n \geq 0\}$$

EXAMPLE 4.3

If $G = (\{S\}, \{a\}, \{S \to SS\}, S)$, find the language generated by G.

Solution

$L(G) = \emptyset$, since the only production $S \to SS$ in G has no terminal on the right-hand side.

EXAMPLE 4.4

Let $G = (\{S, C\}, \{a, b\}, P, S)$, where P consists of $S \to aCa$, $C \to aCa \mid b$. Find $L(G)$.

Solution

$S \Rightarrow aCa \Rightarrow aba$. So $aba \in L(G)$

$S \Rightarrow aCa$ (by application of $S \to aCa$)

$\overset{*}{\Rightarrow} a^nCa^n$ (by application of $C \to aCa$ $(n-1)$ times)

$\Rightarrow a^nba^n$ (by application of $C \to b$)

Hence, $a^nba^n \in L(G)$, where $n \geq 1$. Therefore,

$$\{a^nba^n \mid n \geq 1\} \subseteq L(G)$$

As the only S-production is $S \to aCa$, this is the first production we have to apply in the derivation of any terminal string. If we apply $C \to b$, we get aba. Otherwise we have to apply only $C \to aCa$, either once or several times. So we get a^nCa^n with a single variable C. To get a terminal string we have to replace C by b, by applying $C \to b$. So any derivation is of the form

$$S \overset{*}{\Rightarrow} a^nba^n \text{ with } n \geq 1$$

Therefore,

$$L(G) \subseteq \{a^nba^n \mid n \geq 1\}$$

Thus,

$$L(G) = \{a^nba^n \mid n \geq 1\}$$

EXERCISE Construct a grammar G so that $L(G) = \{a^nba^m \mid n, m \geq 1\}$.

Remark By applying the convention regarding the notation of variables, terminals and the start symbol, it will be clear from the context whether a symbol denotes a variable or terminal. We can specify a grammar by its productions alone.

EXAMPLE 4.5

If G is $S \to aS \mid bS \mid a \mid b$, find $L(G)$.

Solution

We show that $L(G) = \{a, b\}^+$. As we have only two terminals a, b, $L(G) \subseteq \{a, b\}^*$. All productions are S-productions, and so Λ can be in $L(G)$ only when $S \to \Lambda$ is a production in the grammar G. Thus,

$$L(G) \subseteq \{a, b\}^* - \{\Lambda\} = \{a, b\}^+$$

To show $\{a, b\}^+ \subseteq L(G)$, consider any string $a_1a_2 \ldots a_n$, where each a_i is either a or b. The first production in the derivation of $a_1a_2 \ldots a_n$ is $S \to aS$ or $S \to bS$ according as $a_1 = a$ or $a_1 = b$. The subsequent productions are obtained in a similar way. The last production is $S \to a$ or $S \to b$ according as $a_n = a$ or $a_n = b$. So $a_1a_2 \ldots a_n \in L(G)$. Thus, we have $L(G) = \{a, b\}^+$.

EXERCISE If G is $S \to aS \mid a$, then show that $L(G) = \{a\}^+$.

Some of the following examples illustrate the method of constructing a grammar G generating a given subset of strings over Σ. The difficult part is the construction of productions. We try to define the given set by recursion and then develop productions generating the strings in the given subset of Σ^*.

EXAMPLE 4.6

Let L be the set of all palindromes over $\{a, b\}$. Construct a grammar G generating L.

Solution

For constructing a grammar G generating the set of all palindromes, we use the recursive definition (given in Section 2.4) to observe the following:

(i) Λ is a palindrome.
(ii) a, b are palindromes.
(iii) If x is a palindrome axa, then bxb are palindromes.

So we define P as the set consisting of:

(i) $S \to \Lambda$
(ii) $S \to a$ and $S \to b$
(iii) $S \to aSa$ and $S \to bSb$

Let $G = (\{S\}, \{a, b\}, P, S)$. Then

$$S \Rightarrow \Lambda, \quad S \Rightarrow a, \quad S \Rightarrow b$$

Therefore,

$$\Lambda, a, b \in L(G)$$

If x is a palindrome of even length, then $x = a_1a_2 \ldots a_m a_m \ldots a_1$, where each a_i is either a or b. Then $S \overset{*}{\Rightarrow} a_1a_2 \ldots a_m a_m a_{m-1} \ldots a_1$ by applying $S \to aSa$ or $S \to bSb$. Thus, $x \in L(G)$.

If x is a palindrome of odd length, then $x = a_1a_2 \ldots a_nca_n \ldots a_1$, where a_i's and c are either a or b. So $S \overset{*}{\Rightarrow} a_1 \ldots a_nSa_n \ldots a_1 \Rightarrow x$ by applying $S \to aSa$, $S \to bSb$ and finally, $S \to a$ or $S \to b$. Thus, $x \in L(G)$. This proves $L = L(G)$.

EXAMPLE 4.7

Construct a grammar generating $L = \{wcw^T \mid w \in \{a, b\}^*\}$.

Solution

Let $G = (\{S\}, \{a, b, c\}, P, S)$, where P is defined as $S \to aSa \mid bSb \mid c$. It is easy to see the idea behind the construction. Any string in L is generated by recursion as follows: (i) $c \in L$; (ii) if $x \in L$, then $wxw^T \in L$. So, as in the earlier example, we have the productions $S \to aSa \mid bSb \mid c$.

EXAMPLE 4.8

Find a grammar generating $L = \{a^nb^nc^i \mid n \geq 1, i \geq 0\}$.

Solution

$$L = L_1 \cup L_2$$
$$L_1 = \{a^nb^n \mid n \geq 1\}$$
$$L_2 = \{a^nb^nc^i \mid n \geq 1, i \geq 1\}$$

We construct L_1 by recursion and L_2 by concatenating the elements of L_1 and c^i, $i \geq 1$. We define P as the set of the following productions:

$$S \to A, \qquad A \to ab, \qquad A \to aAb, \qquad S \to Sc$$

Let $G = (\{S, A\}, \{a, b, c\}, P, S)$. For $n \geq 1$, $i \geq 0$, we have

$$S \overset{*}{\Rightarrow} Sc^i \Rightarrow Ac^i \overset{*}{\Rightarrow} a^{n-1}Ab^{n-1}c^i \Rightarrow a^{n-1}abb^{n-1}c^i = a^nb^nc^i$$

Thus,

$$\{a^nb^nc^i \mid n \geq 1, i \geq 0\} \subseteq L(G)$$

To prove the reverse inclusion, we note that the only S-productions are $S \to Sc$ and $S \to A$. If we start with $S \to A$, we have to apply

$$A \Rightarrow a^{n-1}Ab^{n-1} \overset{*}{\Rightarrow} a^nb^n, \text{ and so } a^nb^nc^0 \in L(G)$$

If we start with $S \to Sc$, we have to apply $S \to Sc$ repeatedly to get Sc^i. But to get a terminal string, we have to apply $S \to A$. As $A \overset{*}{\Rightarrow} a^nb^n$, the resulting terminal string is $a^nb^nc^i$. Thus, we have shown that

$$L(G) \subseteq \{a^nb^nc^i \mid n \geq 1, i \geq 0\}$$

Therefore,

$$L(G) = \{a^nb^nc^i \mid n \geq 1, i \geq 0\}$$

EXAMPLE 4.9

Find a grammar generating $\{a^ib^nc^n \mid n \geq 1, j \geq 0\}$.

Solution

Let $G = (\{S, A\}, \{a, b, c\}, P, S)$, where P consists of $S \to aS$, $S \to A$, $A \to bAc \mid bc$. As in the previous example, we can prove that G is the required grammar.

EXAMPLE 4.10

Let $G = (\{S, A_1\}, \{0, 1, 2\}, P, S)$, where P consists of $S \to 0SA_12$, $S \to 012$, $2A_1 \to A_12$, $1A_1 \to 11$. Show that

$$L(G) = \{0^n1^n2^n \mid n \geq 1\}$$

Solution

As $S \to 012$ is a production, we have $S \Rightarrow 012$, i.e. $012 \in L(G)$. Also,

$$\begin{aligned}
S &\overset{*}{\Rightarrow} 0^{n-1}S(A_12)^{n-1} && \text{by applying } S \to 0SA_12 \quad (n-1) \text{ times}\\
&\Rightarrow 0^n12(A_12)^{n-1} && \text{by applying } S \to 012\\
&\overset{*}{\Rightarrow} 0^n1A_1^{n-1}2^n && \text{by applying } 2A_1 \to A_12 \quad \text{several times}\\
&\overset{*}{\Rightarrow} 0^n1^n2^n && \text{by applying } 1A_1 \to 11 \quad (n-1) \text{ times}
\end{aligned}$$

Therefore,

$$0^n1^n2^n \in L(G) \qquad \text{for all } n \geq 1$$

To prove that $L(G) \subseteq \{0^n1^n2^n \mid n \geq 1\}$, we proceed as follows: If the first production that we apply is $S \to 012$, we get 012. Otherwise we have to apply $S \to 0SA_12$ once or several times to get $0^{n-1}S(A_12)^{n-1}$. To eliminate S, we have to apply $S \to 012$. Thus we arrive at a sentential form $0^n12(A_12)^{n-1}$. To eliminate the variable A_1, we have to apply $2A_1 \to A_12$ or $1A_1 \to 11$. Now, $2A_1 \to A_12$ interchanges 2 and A_1. Only $1A_1 \to 11$ eliminates A_1. The sentential form we have obtained is $0^n12A_12A_12 \ldots A_12$. If we use $1A_1 \to 11$ before taking all 2's to the right, we will get 12 in the middle of the string. The A_1's appearing subsequently cannot be eliminated. So we have to bring all 2's to the right by applying $2A_1 \to A_12$ several times. Then we can apply $1A_1 \to 11$ repeatedly and get $0^n1^n2^n$ (as derived in the first part of the proof). Thus,

$$L(G) \subseteq \{0^n1^n2^n \mid n \geq 1\}$$

This shows that

$$L(G) = \{0^n1^n2^n \mid n \geq 1\}$$

In the next example we construct a grammar generating

$$\{a^nb^nc^n \mid n \geq 1\}$$

EXAMPLE 4.11

Construct a grammar G generating $\{a^n b^n c^n \mid n \geq 1\}$.

Solution

Let $L = \{a^n b^n c^n \mid n \geq 1\}$. We try to construct L by recursion. We already know how to construct $a^n b^n$ recursively.

As it is difficult to construct $a^n b^n c^n$ recursively, we do it in two stages: (i) we construct $a^n \alpha^n$ and (ii) we convert α^n into $b^n c^n$. For stage (i), we can have the following productions $S \to aS\alpha \mid a\alpha$. A natural choice for α (to execute stage (ii)) is bc. But converting $(bc)^n$ into $b^n c^n$ is not possible as $(bc)^n$ has no variables. So we can take $\alpha = BC$, where B and C are variables. To bring B's together we introduce $CB \to BC$. We introduce some more productions to convert B's into b's and C's into c's. So we define G as

$$G = (\{S, B, C\}, \{a, b, c\}, P, S)$$

where P consists of

$$S \to aSBC \mid aBC,\ CB \to BC,\ aB \to ab,\ bB \to bb,\ bC \to bc,\ cC \to cc$$

$$S \Rightarrow aBC \Rightarrow abC \Rightarrow abc$$

Thus,

$$abc \in L(G)$$

Also,

$S \overset{*}{\Rightarrow} a^{n-1}S(BC)^{n-1}$	by applying $S \to aSBC$	$(n-1)$ times
$\Rightarrow a^{n-1}aBC(BC)^{n-1}$	by applying $S \to aBC$	
$\overset{*}{\Rightarrow} a^n B^n C^n$	by applying $CB \to BC$ (since $CB \to BC$ interchanges B and C)	several times
$\Rightarrow a^{n-1}abB^{n-1}C^n$	by applying $aB \to ab$	once
$\overset{*}{\Rightarrow} a^n b^n C^n$	by applying $bB \to bb$	several times
$\Rightarrow a^n b^{n-1} bcC^{n-1}$	by applying $bC \to bc$	once
$\Rightarrow a^n b^n c^n$	by applying $cC \to cc$	several times

Therefore,

$$L(G) \subseteq \{a^n b^n c^n \mid n \geq 1\}$$

To show that $\{a^n b^n c^n \mid n \geq 1\} \subseteq L(G)$, it is enough to prove that the only way to arrive at a terminal string is to proceed as above in deriving $a^n b^n c^n$ $(n \geq 1)$.

To start with, we have to apply only S-production. If we apply $S \to aBC$, first we get ab. Otherwise we have to apply $S \to aSBC$ once or several times and get the sentential form $a^{n-1}S(BC)^{n-1}$. At this stage the only production we can apply is $S \to aBC$, and the resulting string is $a^n(BC)^n$.

In the derivation of $a^n b^n c^n$, we converted all B's into b's and only then converted C's into c's. We show that this is the only way of arriving at a terminal string.

$a^n(BC)^n$ is a string of terminals followed by a string of variables. The productions we can apply to $a^n(BC)^n$ are either $CB \to BC$ or one of $aB \to ab$, $bB \to bb$, $bC \to bc$, $cC \to cc$. By the application of any one of these productions, we get a sentential form which is a string of terminals followed by a string of variables. Suppose a C is converted before converting all B's. Then we have $a^n(BC)^n \overset{*}{\Rightarrow} a^n b^i c\alpha$, where $i < n$ and α is a string of B's and C's containing at least one B. In $a^n b^i c\alpha$, the variables appear only in α. As c appears just before α, the only production we can apply is $cC \to cc$. If α starts with B, we cannot proceed. Otherwise we apply $cC \to cc$ repeatedly until we obtain the string of the form $a^n b^i c^j B\alpha'$. But the only productions involving B are $aB \to ab$ and $bB \to bb$. As B is preceded by c in $a^n b^i c^j B\alpha'$, we cannot convert B, and so we cannot get a terminal string. So $L(G) \subseteq \{a^n b^n c^n \mid n \geq 1\}$. Thus, we have proved that

$$L(G) = \{a^n b^n c^n \mid n \geq 1\}$$

EXAMPLE 4.12

Construct a grammar G generating $\{xx \mid x \in \{a, b\}^*\}$.

Solution

We construct G as follows:

$$G = (\{S, S_1, S_2, S_3, A, B\}, \{a, b\}, P, S)$$

where P consists of

$$P_1: \quad S \to S_1S_2S_3$$

$$P_2, P_3: \quad S_1S_2 \to aS_1A, \quad S_1S_2 \to bS_1B$$

$$P_4, P_5: \quad AS_3 \to S_2aS_3, \quad BS_3 \to S_2bS_3$$

$$P_6, P_7, P_8, P_9: \quad Aa \to aA, \quad Ab \to bA, \quad Ba \to aB, \quad Bb \to bB$$

$$P_{10}, P_{11}: \quad aS_2 \to S_2a, \quad bS_2 \to S_2b$$

$$P_{12}, P_{13}: \quad S_1S_2 \to \Lambda, \quad S_3 \to \Lambda$$

Remarks The following remarks give us an idea about the construction of productions P_1–P_{13}.

1. P_1 is the only S-production.
2. Using $S_1S_2 \to aS_1A$, we can add terminal a to the left of S_1 and variable A to the right. A is used to make us remember that we have added the terminal a to the left of S_1. Using $AS_3 \to S_2aS_3$, we add a to the right of S_2.

3. Using $S_1S_2 \to bS_1B$, we add b to the left of S_1 and variable B to the right. Using $BS_3 \to S_2bS_3$, we add b to the right of S_2.
4. S_2 acts as a centre-marker.
5. We can add terminals only by using P_2–P_5.
6. P_6–P_9 simply interchange symbols. They push A or B to the right. This enables us to place A or B to the left of S_3. (Only then we can apply P_4 or P_5.)
7. $S_1S_2 \to \Lambda$, $S_3 \to \Lambda$ are used to completely eliminate S_1, S_2, S_3.
8. P_{10}, P_{11} are used to push S_2 to the left. This enables us to get S_2 to the right of S_1 (so that we can apply P_{12}).

Let $L = \{xx \mid x \in \{a, b\}^*\}$. We first prove that $L \subseteq L(G)$. Now, we have

$$S \Rightarrow S_1S_2S_3 \Rightarrow aS_1AS_3 \Rightarrow aS_1S_2aS_3 \tag{4.1}$$

or

$$S \Rightarrow S_1S_2S_3 \Rightarrow bS_1BS_3 \Rightarrow bS_1S_2bS_3 \tag{4.2}$$

Let us start with xx with $x \in \{a, b\}^*$. We can apply (4.1) or (4.2), depending on the first symbol of x. If the first two symbols in x are ab (the other cases are similar), we have

$$S \overset{*}{\Rightarrow} a\underline{S_1S_2}aS_3 \Rightarrow abS_1\underline{Ba}S_3 \Rightarrow abS_1a\underline{BS_3} \Rightarrow abS_1\underline{aS_2}bS_3 \Rightarrow abS_1S_2abS_3$$

Repeating the construction for every symbol in x, we get $xS_1S_2xS_3$. On application of P_{12} and P_{13}, we get

$$S \overset{*}{\Rightarrow} xS_1S_2xS_3 \overset{*}{\Rightarrow} x\Lambda x\Lambda = xx$$

Thus, $L \subseteq L(G)$.

To prove that $L(G) \subseteq L$, we note that the first three steps in any derivation of $L(G)$ are given by (4.1) or (4.2). Thus in any derivation (except $S \to \Lambda$), we get $aS_1S_2aS_3$ or $bS_1S_2bS_3$ as a sentential form.

We can discuss the possible ways of reducing $aS_1S_2aS_3$ (the other case is similar) to a terminal string. The first production that we can apply to $aS_1S_2aS_3$ is one of $S_1S_2 \to \Lambda$, $S_3 \to \Lambda$, $S_1S_2 \to aS_1A$, $S_1S_2 \to bS_1B$.

Case 1 We apply $S_1S_2 \to \Lambda$ to $aS_1S_2aS_3$. In this case we get $a\Lambda aS_3$. As the productions involving S_3 on the left are P_4, P_5 or P_{13}, we have to apply only $S_3 \to \Lambda$ to aaS_3 and get $aa \in L$.

Case 2 We apply $S_3 \to \Lambda$ to $aS_1S_2aS_3$. In this case we get $aS_1S_2a\Lambda$. If we apply $S_1S_2 \to \Lambda$, we get $a\Lambda a\Lambda = aa \in L$; or we can apply $S_1S_2 \to aS_1A$ to aS_1S_2a to get aaS_1Aa. In the latter case, we can apply only $Aa \to aA$ to aaS_1Aa. The resulting string is aaS_1aA which cannot be reduced further.

From Cases 1 and 2 we see that either we have to apply both P_{12} and P_{13} or neither of them.

Case 3 In this case we apply $S_1S_2 \to aS_1A$ or $S_1S_2 \to bS_1B$. If we apply $S_1S_2 \to aS_1A$ to $aS_1S_2aS_3$, we get aaS_1AaS_3. By the nature of productions we have to follow only $aaS_1AaS_3 \Rightarrow aaS_1aAS_3 \Rightarrow a^2S_1aS_2aS_3 \Rightarrow a^2S_1S_2a^2S_3$. If we apply $S_1S_2 \to bS_1B$, we get $abS_1S_2abS_3$. Thus the effect of applying $S_1S_2 \to aS_1A$ is to add a to the left of S_1S_2 and S_3.

If we apply $S_1S_2 \to \Lambda$, $S_3 \to \Lambda$ (By Cases 1 and 2 we have to apply both) we get $abab \in L$. Otherwise, by application of P_2 or P_3, we add the same terminal symbol to the left of S_1S_2 and S_3. The resulting string is of the form $xS_1S_2xS_3$. Ultimately, we have to apply P_{12} and P_{13} and get $x\Lambda x\Lambda = xx \in L$. So $L(G) \subseteq L$. Hence, $L(G) = L$.

EXAMPLE 4.13

Let $G = (\{S, A_1, A_2\}, \{a, b\}, P, S)$, where P consists of

$S \to aA_1A_2a$, $A_1 \to baA_1A_2b$, $A_2 \to A_1ab$, $aA_1 \to baa$, $bA_2b \to abab$

Test whether $w = baabbabaaabbaba$

is in $L(G)$.

Solution

We have to start with an S-production. At every stage we apply a suitable production which is likely to derive w. In this example, we underline the substring to be replaced by the use of a production.

$$\begin{aligned} S &\Rightarrow \underline{aA_1}A_2a \\ &\Rightarrow baa\underline{A_2}a \\ &\Rightarrow baa\underline{A_1}aba \\ &\Rightarrow baab\underline{aA_1}A_2baba \\ &\Rightarrow baabbaa\underline{A_2}baba \\ &\Rightarrow baabba\underline{aA_1}abbaba \\ &\Rightarrow baabbabaaabbaba = w \end{aligned}$$

Therefore,

$$w \in L(G)$$

EXAMPLE 4.14

If the grammar G is given by the productions $S \to aSa \mid bSb \mid aa \mid bb \mid \Lambda$, show that (i) $L(G)$ has no strings of odd length, (ii) any string in $L(G)$ is of length $2n$, $n \geq 0$, and (iii) the number of strings of length $2n$ is 2^n.

Solution

On application of any production (except $S \to \Lambda$), a variable is replaced by two terminals and at the most one variable. So, every step in any derivation increases the number of terminals by 2 except that involving $S \to \Lambda$. Thus, we have proved (i) and (ii).

To prove (iii), consider any string w of length $2n$. Then it is of the form $a_1a_2 \ldots a_na_n \ldots a_1$ involving n 'parameters' $a_1, a_2, \ldots, a_n$. Each a_i can be either a or b. So the number of such strings is 2^n. This proves (iii).

4.2 CHOMSKY CLASSIFICATION OF LANGUAGES

In the definition of a grammar (V_N, Σ, P, S), V_N and Σ are the sets of symbols and $S \in V_N$. So if we want to classify grammars, we have to do it only by considering the form of productions. Chomsky classified the grammars into four types in terms of productions (types 0–3).

A type 0 grammar is any phrase structure grammar without any restrictions. (All the grammars we have considered are type 0 grammars.)

To define the other types of grammars, we need a definition.

In a production of the form $\phi A\psi \to \phi\alpha\psi$, where A is a variable, ϕ is called the left context, ψ the right context, and $\phi\alpha\psi$ the replacement string.

EXAMPLE 4.15

(a) In $ab\underline{A}bcd \to ab\underline{AB}bcd$, ab is the left context, bcd is the right context, $\alpha = AB$.

(b) In $A\underline{C} \to A$, A is the left context, Λ is the right context. $\alpha = \Lambda$. The production simply erases C when the left context is A and the right context is Λ.

(c) For $C \to \Lambda$, the left and right contexts are Λ. And $\alpha = \Lambda$. The production simply erases C in any context.

A production without any restrictions is called a type 0 production.

A production of the form $\phi A\psi \to \phi\alpha\psi$ is called a type 1 production if $\alpha \neq \Lambda$. In type 1 productions, erasing of A is not permitted.

EXAMPLE 4.16

(a) $a\underline{A}bcD \to a\underline{bcD}bcD$ is a type 1 production where a, bcD are the left context and right context, respectively. A is replaced by $bcD \neq \Lambda$.

(b) $A\underline{B} \to A\underline{bBc}$ is a type 1 production. The left context is A, the right context is Λ.

(c) $A \to abA$ is a type 1 production. Here both the left and right contexts are Λ.

Definition 4.7 A grammar is called type 1 or context-sensitive or context-dependent if all its productions are type 1 productions. The production $S \to \Lambda$ is also allowed in a type 1 grammar, but in this case S does not appear on the right-hand side of any production.

Definition 4.8 The language generated by a type 1 grammar is called a type 1 or context-sensitive language.

Note: In a context-sensitive grammar G, we allow $S \to \Lambda$ for including Λ in $L(G)$. Apart from $S \to \Lambda$, all the other productions do not decrease the length of the working string.

A type 1 production $\phi A\psi \to \phi\alpha\psi$ does not decrease the length of the working string. In other words, $|\phi A\psi| \leq |\phi\alpha\psi|$ as $\alpha \neq \Lambda$. But if $\alpha \to \beta$ is a production such that $|\alpha| \leq |\beta|$, then it need not be a type 1 production. For example, $BC \to CB$ is not of type 1. We prove that such productions can be replaced by a set of type 1 productions (Theorem 4.2).

Theorem 4.1 Let G be a type 0 grammar. Then we can find an equivalent grammar G_1 in which each production is either of the form $\alpha \to \beta$, where α and β are strings of variables only, or of the form $A \to a$, where A is a variable and a is a terminal. G_1 is of type 1, 2 or 3 according as G is of type 1, 2 or 3.

Proof We construct G_1 as follows: For constructing productions of G_1, consider a production $\alpha \to \beta$ in G, where α or β has some terminals. In both α and β we replace every terminal by a new variable C_a and get α' and β'. Thus, corresponding to every $\alpha \to \beta$, where α or β contains some terminal, we construct $\alpha' \to \beta'$ and productions of the form $C_a \to a$ for every terminal a appearing in α or β. The construction is performed for every such $\alpha \to \beta$. The productions for G_1 are the new productions we have obtained through the above construction. For G_1 the variables are the variables of G together with the new variables (of the form C_a). The terminals and the start symbol of G_1 are those of G. G_1 satisfies the required conditions and is equivalent to G. So $L(G) = L(G_1)$. ∎

Definition 4.9 A grammar $G = (V_N, \Sigma, P, S)$ is monotonic (or length-increasing) if every production in P is of the form $\alpha \to \beta$ with $|\alpha| \leq |\beta|$ or $S \to \Lambda$. In the second case, S does not appear on the right-hand side of any production in P.

Theorem 4.2 Every monotonic grammar G is equivalent to a type 1 grammar.

Proof We apply Theorem 4.1 to get an equivalent grammar G_1. We construct G' equivalent to grammar G_1 as follows: Consider a production $A_1A_2 \ldots A_m \to B_1B_2 \ldots B_n$ with $n \geq m$ in G_1. If $m = 1$, then the above production is of type 1 (with left and right contexts being Λ). Suppose $m \geq 2$. Corresponding to $A_1A_2 \ldots A_m \to B_1B_2 \ldots B_n$, we construct the following type 1 productions introducing the new variables $C_1, C_2, \ldots, C_m$.

$$\underline{A_1}\,A_2 \ldots A_m \to \underline{C_1}\,A_2 \ldots A_m$$

$$C_1\,\underline{A_2} \ldots A_m \to C_1\,\underline{C_2}\,A_3 \ldots A_m$$

$$C_1C_2\,\underline{A_3} \ldots A_m \to C_1C_2\,\underline{C_3}\,A_4 \ldots A_m \ldots$$

$$C_1C_2 \ldots C_{m-1}\,\underline{A_m} \to C_1C_2 \ldots C_mB_{m+1}B_{m+2} \ldots B_n$$

$$\underline{C_1}C_2 \ldots C_mB_{m+1} \ldots B_n \rightarrow \underline{B_1}C_2 \ldots C_mB_{m+1} \ldots B_n$$

$$B_1\underline{C_2}C_3 \ldots B_n \rightarrow B_1\underline{B_2}C_3 \ldots B_n \ldots$$

$$B_1B_2 \ldots \underline{C_m}B_{m+1} \ldots B_n \rightarrow B_1B_2 \ldots \underline{B_m} \ldots B_n$$

The above construction can be explained as follows. The production

$$A_1A_2 \ldots A_m \rightarrow B_1B_2 \ldots B_n$$

is not of type 1 as we replace more than one symbol on L.H.S. In the chain of productions we have constructed, we replace A_1 by C_1, A_2 by $C_2 \ldots$, A_m by $C_mB_{m+1} \ldots B_n$. Afterwards, we start replacing C_1 by B_1, C_2 by B_2, etc. As we replace only one variable at a time, these productions are of type 1.

We repeat the construction for every production in G_1 which is not of type 1. For the new grammar G', the variables are the variables of G_1 together with the new variables. The productions of G' are the new type 1 productions obtained through the above construction. The terminals and the start symbol of G' are those of G_1.

G' is context-sensitive and from the construction it is easy to see that $L(G') = L(G_1) = L(G)$. ∎

Definition 4.10 A type 2 production is a production of the form $A \rightarrow \alpha$, where $A \in V_N$ and $\alpha \in (V_N \cup \Sigma)^*$. In other words, the L.H.S. has no left context or right context. For example, $S \rightarrow Aa$, $A \rightarrow a$, $B \rightarrow abc$, $A \rightarrow \Lambda$ are type 2 productions.

Definition 4.11 A grammar is called a type 2 grammar if it contains only type 2 productions. It is also called a context-free grammar (as A can be replaced by α in any context). A language generated by a context-free grammar is called a type 2 language or a context-free language.

Definition 4.12 A production of the form $A \rightarrow a$ or $A \rightarrow aB$, where $A, B \in V_N$ and $a \in \Sigma$, is called a type 3 production.

Definition 4.13 A grammar is called a type 3 or regular grammar if all its productions are type 3 productions. A production $S \rightarrow \Lambda$ is allowed in type 3 grammar, but in this case S does not appear on the right-hand side of any production.

EXAMPLE 4.17

Find the highest type number which can be applied to the following productions:

(a) $S \rightarrow Aa, \quad A \rightarrow c \mid Ba, \quad B \rightarrow abc$
(b) $S \rightarrow ASB \mid d, \quad A \rightarrow aA$
(c) $S \rightarrow aS \mid ab$

Solution

(a) $S \rightarrow Aa$, $A \rightarrow Ba$, $B \rightarrow abc$ are type 2 and $A \rightarrow c$ is type 3. So the highest type number is 2.
(b) $S \rightarrow ASB$ is type 2, $S \rightarrow d$, $A \rightarrow aA$ are type 3. Therefore, the highest type number is 2.
(c) $S \rightarrow aS$ is type 3 and $S \rightarrow ab$ is type 2. Hence the highest type number is 2.

4.3 LANGUAGES AND THEIR RELATION

In this section we discuss the relation between the classes of languages that we have defined under the Chomsky classification.

Let $\mathcal{L}_0$, $\mathcal{L}_{csl}$, $\mathcal{L}_{cfl}$ and $\mathcal{L}_{rl}$ denote the family of type 0 languages, context-sensitive languages, context-free languages and regular languages, respectively.

Property 1 From the definition, it follows that $\mathcal{L}_{rl} \subseteq \mathcal{L}_{cfl}$, $\mathcal{L}_{csl} \subseteq \mathcal{L}_0$, $\mathcal{L}_{cfl} \subseteq \mathcal{L}_0$.

Property 2 $\mathcal{L}_{cfl} \subseteq \mathcal{L}_{csl}$. The inclusion relation is not immediate as we allow $A \rightarrow \Lambda$ in context-free grammars even when $A \neq S$, but not in context-sensitive grammars (we allow only $S \rightarrow \Lambda$ in context-sensitive grammars). In Chapter 6 we prove that a context-free grammar G with productions of the form $A \rightarrow \Lambda$ is equivalent to a context-free grammar G_1 which has no productions of the form $A \rightarrow \Lambda$ (except $S \rightarrow \Lambda$). Also, when G_1 has $S \rightarrow \Lambda$, S does not appear on the right-hand side of any production. So G_1 is context-sensitive. This proves $\mathcal{L}_{cfl} \subseteq \mathcal{L}_{csl}$.

Property 3 $\mathcal{L}_{rl} \subseteq \mathcal{L}_{cfl} \subseteq \mathcal{L}_{csl} \subseteq \mathcal{L}_0$. This follows from properties 1 and 2.

Property 4 $\mathcal{L}_{rl} \subsetneq \mathcal{L}_{cfl} \subsetneq \mathcal{L}_{csl} \subsetneq \mathcal{L}_0$.

In Chapter 5, we shall prove that $\mathcal{L}_{rl} \subsetneq \mathcal{L}_{cfl}$. In Chapter 6, we shall prove that $\mathcal{L}_{cfl} \subsetneq \mathcal{L}_{csl}$. In Section 9.7, we shall establish that $\mathcal{L}_{csl} \subsetneq \mathcal{L}_0$.

Remarks 1. The grammars given in Examples 4.1–4.4 and 4.6–4.9 are context- free but not regular. The grammar given in Example 4.5 is regular. The grammars given in Examples 4.10 and 4.11 are not context-sensitive as we have productions of the form $2A_1 \rightarrow A_12$, $CB \rightarrow BC$ which are not type 1 rules. But they are equivalent to a context-sensitive grammar by Theorem 4.2.

2. Two grammars of different types may generate the same language. For example, consider the regular grammar G given in Example 4.5. It generates $\{a, b\}^+$. Let G' be given by $S \rightarrow SS \mid aS \mid bS \mid a \mid b$. Then $L(G') = L(G)$ as the productions $S \rightarrow aS \mid bS \mid a \mid b$ are in G as well, and $S \rightarrow SS$ does not generate any more string.

3. The type of a given grammar is easily decided by the nature of productions. But to decide on the type of a given subset of Σ^*, it is more difficult. By Remark 2, the same set of strings may be generated by a grammar of higher type. To prove that a given language is not regular or context-free, we need powerful theorems like Pumping Lemma.

4.4 RECURSIVE AND RECURSIVELY ENUMERABLE SETS

The results given in this section will be used to prove $\mathscr{L}_{csl} \subset_{\neq} \mathscr{L}_0$ in Section 9.7. For defining recursive sets, we need the definition of a procedure and an algorithm.

A *procedure* for solving a problem is a finite sequence of instructions which can be mechanically carried out given any input.

An *algorithm* is a procedure that terminates after a finite number of steps for any input.

Definition 4.14 A set X is recursive if we have an algorithm to determine whether a given element belongs to X or not.

Definition 4.15 A recursively enumerable set is a set X for which we have a procedure to determine whether a given element belongs to X or not.

It is clear that a recursive set is recursively enumerable.

Theorem 4.3 A context-sensitive language is recursive.

Proof Let $G = (V_N, \Sigma, P, S)$ and $w \in \Sigma^*$. We have to construct an algorithm to test whether $w \in L(G)$ or not. If $w = \Lambda$, then $w \in L(G)$ iff $S \to \Lambda$ is in P. As there are only a finite number of productions in P, we have to test whether $S \to \Lambda$ is in P or not.

Let $|w| = n \geq 1$. The algorithm is based on the construction of a sequence $\{W_i\}$ of subsets of $(V_N \cup \Sigma)^*$. W_i is simply the set of all sentential forms of length less than or equal to n, derivable in at most i steps. The construction is done recursively as follows:

(i) $W_0 = \{S\}$.

(ii) $W_{i+1} = W_i \cup \{\beta \in (V_N \cup \Sigma)^* |$ there exists α in W_i such that $\alpha \Rightarrow \beta$ and $|\beta| \leq n\}$.

W_i's satisfy the following:

(iii) $W_i \subseteq W_{i+1}$ for all $i \geq 0$.

(iv) There exists k such that $W_k = W_{k+1}$.

(v) If k is the smallest integer such that $W_k = W_{k+1}$, then $W_k = \{\alpha \in (V_N \cup \Sigma)^* | S \overset{*}{\Rightarrow} \alpha$ and $|\alpha| \leq n\}$.

The point (iii) follows from the point (ii). To prove the point (iv), we consider the number N of strings over $V_N \cup \Sigma$ of length less than or equal to n. If $|V_N \cup \Sigma| = m$, then $N = 1 + m + m^2 + \ldots + m^n$ since m^i is the number of strings of length i over $V_N \cup \Sigma$, i.e. $N = (m^{n+1} - 1)/(m - 1)$, and N is fixed as it depends only on n and m. As any string in W_i is of length at most n, $|W_i| \leq N$. Therefore, $W_k = W_{k+1}$ for some $k \leq N$. This proves the point (iv).

From point (ii) it follows that $W_k = W_{k+1}$ implies $W_{k+1} = W_{k+2}$.

$$\{\alpha \in (V_N \cup \Sigma)^* | S \overset{*}{\Rightarrow} \alpha, |\alpha| \leq n\} = W_1 \cup W_2 \cup \ldots \cup W_k \cup W_{k+1} \cdots$$
$$= W_1 \cup W_2 \cup \ldots \cup W_k$$
$$= W_k \text{ from point (iii)}$$

This proves the point (v).

From the point (v) it follows that $w \in L(G)$ (i.e. $S \overset{*}{\Rightarrow} w$) if and only if $w \in W_k$. Also, $W_1, W_2, \ldots, W_k$ can be constructed in a finite number of steps. We give the required algorithm as follows:

Algorithm to test whether $w \in L(G)$. 1. Construct $W_1, W_2, \ldots$ using the points (i) and (ii). We terminate the construction when $W_{k+1} = W_k$ for the first time.

2. If $w \in W_k$, then $w \in L(G)$. Otherwise, $w \notin L(G)$. (As $|W_k| \leq N$, testing whether w is in W_k requires at most N steps.) ∎

EXAMPLE 4.18

Consider the grammar G given by $S \to 0SA_12$, $S \to 012$, $2A_1 \to A_12$, $1A_1 \to 11$. Test whether (a) $00112 \in L(G)$ and (b) $001122 \in L(G)$.

Solution

(a) To test whether $w = 00112 \in L(G)$, we construct the sets W_0, W_1, W_2 etc. $|w| = 5$.

$$W_0 = \{S\}$$
$$W_1 = \{012, S, 0SA_12\}$$
$$W_2 = \{012, S, 0SA_12\}$$

As $W_2 = W_1$, we terminate. (Although $0SA_12 \Rightarrow 0012A_12$, we cannot include $0012A_12$ in W_1 as its length is > 5.) Then $00112 \notin W_1$. Hence, $00112 \notin L(G)$.

(b) To test whether $w = 001122 \in L(G)$. Here, $|w| = 6$. We construct W_0, W_1, W_2, etc.

$$W_0 = \{S\}$$
$$W_1 = \{012, S, 0SA_12\}$$
$$W_2 = \{012, S, 0SA_12, 0012A_12\}$$
$$W_3 = \{012, S, 0SA_12, 0012A_12, 001A_122\}$$
$$W_4 = \{012, S, 0SA_12, 0012A_12, 001A_122, 001122\}$$
$$W_5 = \{012, S, 0SA_12, 0012A_12, 001A_122, 001122\}$$

As $W_5 = W_4$, we terminate. Then $001122 \in W_4$. Thus, $001122 \in L(G)$.

The following theorem is of theoretical interest, and shows that there exists a recursive set over $\{0, 1\}$ which is not a context-sensitive language. The proof is by the diagonalization method which is used quite often in set theory.

Theorem 4.4 There exists a recursive set which is not a context-sensitive language over $\{0, 1\}$.

Proof Let $\Sigma = \{0, 1\}$. We write the elements of Σ^* as a sequence (i.e. the elements of Σ^* are enumerated as the first element, second element, etc.) For

example, one such way of writing is Λ, 0, 1, 00, 01, 10, 11, 000, ... In this case, 010 will be the 10th element.

As every grammar is defined in terms of a finite alphabet set and a finite set of productions, we can also write all context-sensitive grammars over Σ as a sequence, say $G_1, G_2, \ldots$.

We define $X = \{w_i \in \Sigma^* \mid w_i \notin L(G_i)\}$. We can show that X is recursive. If $w \in \Sigma^*$, then we can find i such that $w = w_i$. This can be done in a finite number of steps (depending on $|w|$). For example, if $w = 0100$, then $w = w_{20}$. As G_{20} is context-sensitive, we have an algorithm to test whether $w = w_{20} \in L(G_{20})$ by Theorem 4.3. So X is recursive.

We prove by contradiction that X is not a context-sensitive language. If it is so, then $X = L(G_n)$ for some n. Consider w_n (the nth element in Σ^*). By definition of X, $w_n \in X$ implies $w_n \notin L(G_n)$. This contradicts $X = L(G_n)$. $w_n \notin X$ implies $w_n \in L(G_n)$ and once again, this contradicts $X = L(G_n)$. Thus, $X \neq L(G_n)$ for any n, i.e. X is not a context-sensitive language. ∎

4.5 OPERATIONS ON LANGUAGES

We consider the effect of applying set operations on $\mathscr{L}_0$, $\mathscr{L}_{cs1}$, $\mathscr{L}_{cf1}$, $\mathscr{L}_{r1}$. Let A and B be any two sets of strings. The concatenation AB of A and B is defined by $AB = \{uv \mid u \in A, v \in B\}$. (Here, uv is the concatenation of the strings u and v.)

We define A^1 as A and A^{n+1} as A^nA for all $n \geq 1$.

The transpose set A^T of A is defined by

$$A^T = \{u^T \mid u \in A\}$$

Theorem 4.5 Each of the classes $\mathscr{L}_0$, $\mathscr{L}_{cs1}$, $\mathscr{L}_{cf1}$, $\mathscr{L}_{r1}$ is closed under union.

Proof Let L_1 and L_2 be two languages of the same type i. We can apply Theorem 4.1 to get grammars

$$G_1 = (V'_N, \Sigma_1, P_1, S_1) \quad \text{and} \quad G_2 = (V''_N, \Sigma_2, P_2, S_2)$$

of type i generating L_1 and L_2, respectively. So any production in G_1 or G_2 is either $\alpha \to \beta$, where α, β contain only variables or $A \to a$, where $A \in V_N$, $a \in \Sigma$.

We can further assume that $V'_N \cap V''_N = \emptyset$. (This is achieved by renaming the variables of V''_N if they occur in V'_N.)

Define a new grammar G_u as follows:

$$G_u = (V'_N \cup V''_N \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_u, S)$$

where S is a new symbol, i.e. $S \notin V'_N \cup V''_N$

$$P_u = P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}$$

We prove $L(G_u) = L_1 \cup L_2$ as follows: If $w \in L_1 \cup L_2$, then $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} w$ or $S_2 \overset{*}{\underset{G_2}{\Rightarrow}} w$. Therefore,

$$S \underset{G_u}{\Rightarrow} S_1 \overset{*}{\underset{G_u}{\Rightarrow}} w \quad \text{or} \quad S \underset{G_u}{\Rightarrow} S_2 \overset{*}{\underset{G_u}{\Rightarrow}} w, \text{ i.e. } w \in L(G_u)$$

Thus, $L_1 \cup L_2 \subseteq L(G_u)$.

To prove that $L(G_u) \subseteq L_1 \cup L_2$, consider a derivation of w. The first step should be $S \Rightarrow S_1$ or $S \Rightarrow S_2$. If $S \Rightarrow S_1$ is the first step, in the subsequent steps S_1 is changed. As $V'_N \cap V''_N \neq \emptyset$, these steps should involve only the variables of V'_N and the productions we apply are in P_1. So $S \overset{*}{\underset{G_1}{\Rightarrow}} w$. Similarly, if the first step is $S \Rightarrow S_2$, then $S \underset{G_2}{\Rightarrow} S_2 \overset{*}{\underset{G_2}{\Rightarrow}} w$. Thus, $L(G_u) = L_1 \cup L_2$. Also, $L(G_u)$ is of type 0 or type 2 according as L_1 and L_2 are of type 0 or type 2. If Λ is not in $L_1 \cup L_2$, then $L(G_u)$ is of type 3 or type 1 according as L_1 and L_2 are of type 3 or type 1.

Suppose $\Lambda \in L_1$. In this case, define

$$G_u = (V'_N \cup V''_N \cup \{S, S'\}, \Sigma_1 \cup \Sigma_2, P_u, S')$$

where (i) S' is a new symbol, i.e. $S' \notin V'_N \cup V''_N \cup \{S\}$, and (ii) $P_u = P_1 \cup P_2 \cup \{S' \to S, S \to S_1, S \to S_2\}$. So, $L(G_u)$ is of type 1 or type 3 according as L_1 and L_2 are of type 1 or type 3. When $\Lambda \in L_2$, the proof is similar. ∎

Theorem 4.6 Each of the classes $\mathscr{L}_0$, $\mathscr{L}_{cs1}$, $\mathscr{L}_{cf1}$, $\mathscr{L}_{r1}$ is closed under concatenation.

Proof Let L_1 and L_2 be two languages of type i. Then, as in Theorem 4.5, we get $G_1 = (V'_N, \Sigma_1, P_1, S_1)$ and $G_2 = (V''_N, \Sigma_2, P_2, S_2)$ of the same type i. We have to prove that L_1L_2 is of type i.

Construct a new grammar G_{con} as follows:

$$G_{con} = (V'_N \cup V''_N \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_{con}, S)$$

where $S \notin V'_N \cup V''_N$.

$$P_{con} = P_1 \cup P_2 \cup \{S \to S_1S_2\}$$

We prove $L_1L_2 = L(G_{con})$. If $w = w_1w_2 \in L_1L_2$, then

$$S_1 \overset{*}{\underset{G_1}{\Rightarrow}} w_1, \qquad S_2 \overset{*}{\underset{G_2}{\Rightarrow}} w_2$$

So,

$$S \underset{G_{con}}{\Rightarrow} S_1S_2 \overset{*}{\underset{G_{con}}{\Rightarrow}} w_1w_2$$

Therefore,

$$L_1L_2 \subseteq L(G_{con})$$

If $w \in L(G_{con})$, then the first step in the derivation of w is $S \Rightarrow S_1S_2$. As $V'_N \cap V''_N = \emptyset$ and the productions in G_1 or G_2 involve only the variables (except those of the form $A \to a$), we have $w = w_1w_2$, where $S \overset{*}{\underset{G_1}{\Rightarrow}} w_1$ and $S \overset{*}{\underset{G_2}{\Rightarrow}} w_2$. Thus $L_1L_2 = L(G_{con})$. Also, G_{con} is of type 0 or type 2 according as G_1 and G_2 are of type 0 or type 2. The above construction is sufficient when G_1 and G_2 are also of type 3 or type 1 provided $\Lambda \notin L_1 \cup L_2$.

Suppose G_1 and G_2 are of type 1 or type 3 and $\Lambda \in L_1$ or $\Lambda \in L_2$. Let $L'_1 = L_1 - \{\Lambda\}$, $L'_2 = L_2 - \{\Lambda\}$. Then

$$L_1L_2 = \begin{cases} L'_1L'_2 \cup L'_2 & \text{if } \Lambda \text{ is in } L_1 \text{ but not in } L_2 \\ L'_1L'_2 \cup L'_1 & \text{if } \Lambda \text{ is in } L_2 \text{ but not in } L_1 \\ L'_1L'_2 \cup L'_1 \cup L'_2 \cup \{\Lambda\} & \text{if } \Lambda \text{ is in } L_1 \text{ and also in } L_2 \end{cases}$$

As we have already shown that $\mathcal{L}_{cs1}$ and $\mathcal{L}_{rl}$ are closed under union, L_1L_2 is of type 1 or type 3 according as L_1 and L_2 are of type 1 or type 3. ∎

Theorem 4.7 Each of the classes $\mathcal{L}_0$, $\mathcal{L}_{cs1}$, $\mathcal{L}_{cfl}$, $\mathcal{L}_{rl}$ is closed under the transpose operation.

Proof Let L be a language of type i. Then $L = L(G)$, where G is of type i.

We construct a new grammar G^T as follows: $G^T = (V_N, \Sigma, P^T, S)$, where the productions of P^T are constructed by reversing the symbols on L.H.S. and R.H.S. of every production in P. Symbolically, $\alpha^T \to \beta^T$ is in P^T if $\alpha \to \beta$ is in P.

From the construction it is obvious that G^T is of type 0, 1 or 2 according as G is of type 0, 1 or 2 and $L(G^T) = L^T$. For regular grammar, the proof is given in Chapter 5.

It is more difficult to establish the closure property under intersection at present as we need the properties of families of languages under consideration. We state the results without proof. We prove some of them in Chapter 8.

Theorem 4.8 (i) Each of the families $\mathcal{L}_{01}$, $\mathcal{L}_{cs1}$, $\mathcal{L}_{rl}$ is closed under intersection.

(ii) $\mathcal{L}_{cfl}$ is not closed under intersection. But the intersection of a context-free language and a regular language is context-free.

4.6 LANGUAGES AND AUTOMATA

In Chapters 7 and 9, we shall construct accepting devices for the four types of languages. Figure 4.1 describes the relation between the four types of languages and automata: TM, LBA, pda, and FA stand for Turing machine, linear bounded automaton, pushdown automaton and finite automaton, respectively.

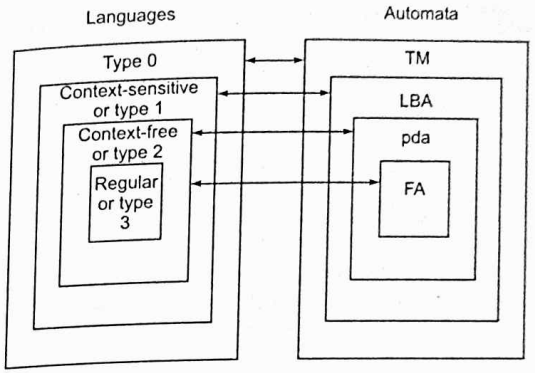

Fig. 4.1 Languages and the corresponding automata.

4.7 SUPPLEMENTARY EXAMPLES

EXAMPLE 4.19

Construct a context-free grammar generating

(a) $L_1 = \{a^nb^{2n} \mid n \geq 1\}$
(b) $L_2 = \{a^mb^n \mid m > n,\ m,\ n \geq 1\}$
(c) $L_3 = \{a^mb^n \mid m < n,\ m,\ n,\ \geq 1\}$
(d) $L_4 = \{a^mb^n \mid m,\ n \geq 0,\ m \neq n\}$

Solution

(a) Let $G_1 = (\{S\}, \{a, b\}, P, S)$ where P consists of $S \to aSbb$, $S \to abb$.

(b) Let $G_2 = (\{S, A\}, \{a, b\}, P, S)$ where P consists of $S \to aS \mid aA$, $A \to aAb$, $A \to ab$. It is easy to see that $L(G) \subseteq L_2$. We prove the difficult part. Let $a^mb^n \in L_2$. Then, $m > n \geq 1$. As $m > n$, we have $m - n \geq 1$. So the derivation of $a^mb^n = a^{m-n}a^nb^n$ is

$$S \overset{*}{\Rightarrow} a^{m-n-1}S \Rightarrow a^{m-n}aA \overset{*}{\Rightarrow} a^{m-n}a^{n-1}Ab^{n-1} \Rightarrow a^mb^n$$

(c) Let $G_3 = (\{S, B\}, \{a, b\}, P, S)$ where P consists of $S \to Sb \mid Bb$, $B \to aBb$, $B \to ab$. This construction is similar to construction in (b). $S \to Sb \mid Bb$ are used to generate b^{n-m}. The remaining productions will generate a^mb^m. Hence $L(G_3) = L_3$.

(d) Note that $L_4 = L_2 \cup L_3 \cup L' \cup L''$ where $L' = \{b^n \mid n \geq 1\}$ and $L'' = \{a^n \mid n \geq 1\}$. It is easy to see how to construct grammars generating L_2, L_3 and L' and L''. Define G_4 by combining these constructions. Let

$G_4 = (\{S, S_1, S_2, S_3, S_4, A\}, \{a, b\}, P_4, S)$ where P_4 consists of

$$S \to S_1 \mid S_2 \mid S_3 \mid S_4, \quad S_1 \to aS_1 \mid Aa \mid aAb \mid ab,$$

$$S_2 \to S_2b \mid Ab, \quad S_3 \to bS_3 \mid b, \quad \text{and} \quad S_4 \to aS_4 \mid a.$$

It is easy to see that $L(G_4) = L_4$.

$G = (\{S, S_1, S_2\}, \{a, b\}, P, S)$ where P consists of

$S \to S_1 \mid S_2$

$S_1 \to a \mid aS_1 \mid S_1a \mid bS_1S_1 \mid S_1bS_1 \mid S_1S_1b$

$S_2 \to b \mid bS_2 \mid S_2b \mid aS_2S_2 \mid S_2aS_2 \mid SS_2a.$

G generates all strings over $\{a, b\}$ having an unequal number of a's and b's.

EXAMPLE 4.24

If L_1 and L_2 are the subsets of $\{a, b\}^*$, prove or disprove:

(a) If $L_1 \subseteq L_2$ and L_1 is not regular, then L_2 is not regular.

(b) If $L_1 \subseteq L_2$ and L_2 is not regular, then L_1 is not regular.

Solution

(a) Let $L_1 = \{a^n b^n \mid n \geq 1\}$. L_1 is not regular. Let $L_2 = \{a, b\}^*$. By Example 4.5, L_2 is regular. Hence (a) is not true.

(b) Let $L_2 = \{a^n b^n \mid n \geq 1\}$. It is not regular. But any finite subset is regular. Taking L_1 to be a finite subset of L_2, we disprove (b).

EXAMPLE 4.25

Show that the set of all non-palindromes over $\{a, b\}$ is a context-free language.

Solution

Let $w \in \{a, b\}^*$ be a non-palindrome. Then w may have the same symbol in the first and last places, same in the second place from the left and from the right, etc.; this pattern will not be there after a particular stage. The productions $S \to aSa \mid bSb \mid \Lambda$ may be used for fulfilling the palindrome-condition for the first and last few places. For violating the palindrome condition, the productions of the form $A \to aBb \mid bBa$ and $B \to aB \mid bB \mid \Lambda$ will be useful. So the required grammar is $G = (\{S, A, B\}, \{a, b\}, P, S)$ where P consists of

$$S \to aSa \mid bSb \mid A$$

$$A \to aBb \mid bBa$$

$$B \to aB \mid bB \mid \Lambda$$

SELF-TEST

Choose the correct answers to Questions 1–12:

1. For a grammar G with productions $S \to SS$, $S \to aSb$, $S \to bSa$, $S \to \Lambda$,

(a) $S \Rightarrow abba$ (b) $S \overset{*}{\Rightarrow} abba$

(c) $abba \notin L(G)$ (d) $S \overset{*}{\Rightarrow} aaa$

2. If $\alpha \overset{*}{\Rightarrow} \beta$ in a grammar G, then

(a) $\alpha \Rightarrow \beta$ (b) $\beta \Rightarrow \alpha$

(c) $\beta \overset{*}{\Rightarrow} \alpha$ (d) none of these

3. If $\alpha \to \beta$ is a production in a grammar G, then

(a) $\alpha\alpha \overset{*}{\Rightarrow} \beta\beta$ (b) $\alpha\alpha\beta \Rightarrow \beta\beta\alpha$

(c) $\alpha\alpha \Rightarrow \beta\alpha$ (d) $\alpha\alpha\alpha \overset{*}{\Rightarrow} \beta\beta\beta$

4. If a grammar G has three productions $S \to aSa \mid bsb \mid c$, then

(a) $abcba$ and $bacab \in L(G)$ (b) $abcba$ and $abcab \in L(G)$

(c) $accca$ and $bcccb \in L(G)$ (d) $acccb$ and $bccca \in L(G)$

5. The minimum number of productions for a grammar $G = (\{S\}, \{0, 1, 2, \ldots, 9\}, P, S)$ for generating $\{0, 1, 2, \ldots, 9\}$ is

(a) 9 (b) 10

(c) 1 (d) 2

6. If $G_1 = (N, T, P_1, S)$ and $G_2 = (N, T, P_2, S)$ and $P_1 \subseteq P_2$, then

(a) $L(G_1) \subseteq L(G_2)$ (b) $L(G_2) \subseteq L(G_1)$

(c) $L(G_1) = L(G_2)$ (d) none of these.

7. The regular grammar generating $\{a^n : n \geq 1\}$ is

(a) $(\{S\}, \{a\}, \{S \to aS\}, S)$

(b) $(\{S\}, \{a\}, \{S \to SS, S \to a\})$

(c) $(\{S\}, \{a\}, \{S \to aS\}, S)$

(d) $(\{S\}, \{a\}, \{S \to aS, S \to a\}\ S)$

8. $L = \{\text{theory, of, computer, science}\}$ can be generated by

(a) a regular grammar

(b) a context-free grammar but not a regular grammar

(c) a context-sensitive grammar but not a context-free grammar

(d) only by a type 0 grammar.

9. $\{a^n \mid n \geq 1\}$ is

(a) regular

(b) context-free but not regular

(c) context-sensitive but not context-free

(d) none of these.

10. $\{a^n b^n \mid n \geq 1\}$ is

(a) regular

(b) context-free but not regular

(c) context-sensitive but not context-free

(d) none of these.

11. $\{a^n b^n c^n \mid n \geq 1\}$ is

(a) regular

(b) context-free but not regular

(c) context-sensitive but not context-free

(d) none of these.

12. $\{a^n b^n c^m \mid n, m \geq 1\}$ is
    (a) regular
    (b) context-free but not regular
    (c) context-sensitive but not context-free
    (d) none of these.

State whether the following Statements 13–20 are true or false:

13. In a grammar $G = (V_N, \Sigma, P, S)$, V_N and Σ are finite but P can be infinite.
14. Two grammars of different types can generate the same language.
15. If $G = (V_N, \Sigma, P, S)$ and $P \neq \emptyset$, then $L(G) \neq \emptyset$.
16. If a grammar G has three productions, i.e. $S \to AA$, $A \to aa$, $A \to bb$, then $L(G)$ is finite.
17. If $L_1 = \{a^n b^m \mid m, n \geq 1\}$ and $L_2 = \{b^m c^p \mid m, p \geq 1\}$, then $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 1\}$.
18. If a grammar G has productions $S \to aS \mid bS \mid a$, then $L(G)$ = the set of all strings over $\{a, b\}$ ending in a.
19. The language $\{a^n bc^n \mid n \geq 1\}$ is regular.
20. If the productions of G are $S \to aS \mid Sb \mid a \mid b$, then $abab \in L(G)$.

EXERCISES

4.1 Find the language generated by the following grammars:
(a) $S \to 0S1 \mid 0A1$, $A \to 1A \mid 1$
(b) $S \to 0S1 \mid 0A \mid 0 \mid 1B \mid 1$, $A \to 0A \mid 0$, $B \to 1B \mid 1$
(c) $S \to 0SBA \mid 01A$, $AB \to BA$, $1B \to 11$, $1A \to 10$, $0A \to 00$
(d) $S \to 0S1 \mid 0A1$, $A \to 1A0 \mid 10$
(e) $S \to 0A \mid 1S \mid 0 \mid 1$, $A \to 1A \mid 1S \mid 1$

4.2 Construct the grammar, accepting each of the following sets:
(a) The set of all strings over $\{0, 1\}$ consisting of an equal number of 0's and 1's.
(b) $\{0^n 1^m 0^m 1^n \mid m, n \geq 1\}$
(c) $\{0^n 1^{2n} \mid n \geq 1\}$
(d) $\{0^n 1^n \mid n \geq 1\} \cup \{1^m 0^m \mid m \geq 1\}$
(e) $\{0^n 1^m 0^n \mid m, n \geq 1\} \cup \{0^n 1^m 2^m \mid m, n \geq 1\}$.

4.3 Test whether 001100, 001010, 01010 are in the language generated by the grammar given in Exercise 4.1(b).

4.4 Let $G = (\{A, B, S\}, \{0, 1\}, P, S)$, where P consists of $S \to 0AB$, $A_0 \to S0B$, $A_1 \to SB1$, $B \to SA$, $B \to 01$. Show that $L(G) = \emptyset$.

4.5 Find the language generated by the grammar $S \to AB$, $A \to A1 \mid 0$, $B \to 2B \mid 3$. Can the above language be generated by a grammar of higher type?

4.6 State whether the following statements are true or false. Justify your answer with a proof or a counter-example.
(a) If G_1 and G_2 are equivalent, then they are of the same type.
(b) If L is a finite subset of Σ^*, then L is a context-free language.
(c) If L is a finite subset of Σ^*, then L is a regular language.

4.7 Show that $\{a^{n^2} \mid n \geq 1\}$ is generated by the grammar $S \to a$, $S \to A_3 A_4$, $A_3 \to A_1 A_3 A_2$, $A_3 \to A_1 A_2$, $A_1 A_2 \to a A_2 A_1$, $A_1 a \to a A_1$, $A_2 a \to a A_2$, $A_1 A_4 \to A_4 a$, $A_2 A_4 \to A_5 a$, $A_2 A_5 \to A_5 a$, $A_5 \to a$.

4.8 Construct (i) a context-sensitive but not context-free grammar, (ii) a context-free but not regular grammar, and (iii) a regular grammar to generate $\{a^n \mid n \geq 1\}$.

4.9 Construct a grammar which generates all even integers up to 998.

4.10 Construct context-free grammars to generate the following:
(a) $\{0^m 1^n \mid m \neq n, m, n \geq 1\}$.
(b) $\{a^l b^m c^n \mid$ one of l, m, n equals 1 and the remaining two are equal$\}$.
(c) $\{0^m 1^n \mid 1 \leq m \leq n\}$.
(d) $\{a^l b^m c^n \mid l + m = n\}$.
(e) The set of all strings over $\{0, 1\}$ containing twice as many 0's as 1's.

4.11. Construct regular grammars to generate the following:
(a) $\{a^{2n} \mid n \geq 1\}$.
(b) The set of all strings over $\{a, b\}$ ending in a.
(c) The set of all strings over $\{a, b\}$ beginning with a.
(d) $\{a^l b^m c^n \mid l, m, n \geq 1\}$.
(e) $\{(ab)^n \mid n \geq 1\}$.

4.12. Is $\underset{G}{\Rightarrow}$ an equivalence relation on $(V_N \cup \Sigma)^*$?

4.13. Show that $G_1 = (\{S\}, \{a, b\}, P_1, S)$, where $P_1 = \{S \to aSb \mid ab\}$ is equivalent to $G_2 = (\{S, A, B, C\}, \{a, b\}, P_2, S)$. Here P_2 consists of $S \to AC$, $C \to SB$, $S \to AB$, $A \to a$, $B \to b$.

4.14. If each production in a grammar G has some variable on its right-hand side, what can you say about $L(G)$?

4.15. Show that $\{abc, bca, cab\}$ can be generated by a regular grammar whose terminal set is $\{a, b, c\}$.

4.16. Construct a grammar to generate $\{(ab)^n \mid n \geq 1\} \cup \{(ba)^n \mid n \geq 1\}$.

4.17. Show that a grammar consisting of productions of the form $A \to xB \mid y$, where x, y are in Σ^* and $A, B \in V_N$, is equivalent to a regular grammar.