5 Regular Sets and Regular Grammars

In this chapter, we first define regular expressions as a means of representing certain subsets of strings over Σ and prove that regular sets are precisely those accepted by finite automata or transition systems. We use pumping lemma for regular sets to prove that certain sets are not regular. We then discuss closure properties of regular sets. Finally, we give the relation between regular sets and regular grammars.

5.1 REGULAR EXPRESSIONS

The regular expressions are useful for representing certain sets of strings in an algebraic fashion. Actually these describe the languages accepted by finite state automata.

We give a formal recursive definition of regular expressions over Σ as follows:

1. Any terminal symbol (i.e. an element of Σ), Λ and $\emptyset$ are regular expressions. When we view a in Σ as a regular expression, we denote it by **a**.
2. The union of two regular expressions $\mathbf{R}_1$ and $\mathbf{R}_2$, written as $\mathbf{R}_1 + \mathbf{R}_2$, is also a regular expression.
3. The concatenation of two regular expressions $\mathbf{R}_1$ and $\mathbf{R}_2$, written as $\mathbf{R}_1\mathbf{R}_2$, is also a regular expression.
4. The iteration (or closure) of a regular expression $\mathbf{R}$, written as $\mathbf{R}^*$, is also a regular expression.
5. If $\mathbf{R}$ is a regular expression, then $(\mathbf{R})$ is also a regular expression.
6. The regular expressions over Σ are precisely those obtained recursively by the application of the rules 1–5 once or several times.

Notes: (1) We use **x** for a regular expression just to distinguish it from the symbol (or string) x.

(2) The parentheses used in Rule 5 influence the order of evaluation of a regular expression.

(3) In the absence of parentheses, we have the hierarchy of operations as follows: iteration (closure), concatenation, and union. That is, in evaluating a regular expression involving various operations, we perform iteration first, then concatenation, and finally union. This hierarchy is similar to that followed for arithmetic expressions (exponentiation, multiplication and addition).

Definition 5.1 Any set represented by a regular expression is called a *regular set.*

If, for example, $a, b \in \Sigma$, then (i) **a** denotes the set $\{a\}$, (ii) **a** + **b** denotes $\{a, b\}$, (iii) **ab** denotes $\{ab\}$, (iv) **a*** denotes the set $\{\Lambda, a, aa, aaa, \ldots\}$ and (v) **(a + b)*** denotes $\{a, b\}^*$.

The set represented by R is denoted by $L(\mathbf{R})$.

Now we shall explain the evaluation procedure for the three basic operations. Let $\mathbf{R}_1$ and $\mathbf{R}_2$ denote any two regular expressions. Then (i) a string in $L(\mathbf{R}_1 + \mathbf{R}_2)$ is a string from $\mathbf{R}_1$ or a string from $\mathbf{R}_2$; (ii) a string in $L(\mathbf{R}_1\mathbf{R}_2)$ is a string from $\mathbf{R}_1$ followed by a string from $\mathbf{R}_2$, and (iii) a string in $L(\mathbf{R}^*)$ is a string obtained by concatenating n elements for some $n \geq 0$. Consequently, (i) the set represented by $\mathbf{R}_1 + \mathbf{R}_2$ is the union of the sets represented by $\mathbf{R}_1$ and $\mathbf{R}_2$, (ii) the set represented by $\mathbf{R}_1\mathbf{R}_2$ is the concatenation of the sets represented by $\mathbf{R}_1$ and $\mathbf{R}_2$. (Recall that the concatenation AB of sets A and B of strings over Σ is given by $AB = \{w_1w_2 | w_1 \in A, w_2 \in B\}$, and (iii) the set represented by $\mathbf{R}^*$ is $\{w_1w_2 \ldots w_n | w_i$ is in the set represented by $\mathbf{R}$ and $n \geq 0.\}$ Hence,

$$L(\mathbf{R}_1 + \mathbf{R}_2) = L(\mathbf{R}_1) \cup L(\mathbf{R}_2), \qquad L(\mathbf{R}_1\mathbf{R}_2) = L(\mathbf{R}_1)L(\mathbf{R}_2)$$
$$L(\mathbf{R}^*) = (L(\mathbf{R}))^*$$

Also,

$$L(\mathbf{R}^*) = (L(\mathbf{R}))^* = \bigcup_{n=0}^{\infty} L(\mathbf{R})^n$$
$$L(\emptyset) = \emptyset, \qquad L(\mathbf{a}) = \{a\}.$$

Note: By the definition of regular expressions, the class of regular sets over Σ is closed under union, concatenation and closure (iteration) by the conditions 2, 3, 4 of the definition.

EXAMPLE 5.1

Describe the following sets by regular expressions: (a) $\{101\}$, (b) $\{abba\}$, (c) $\{01, 10\}$, (d) $\{\Lambda, ab\}$, (e) $\{abb, a, b, bba\}$, (f) $\{\Lambda, 0, 00, 000, \ldots\}$, and (g) $\{1, 11, 111, \ldots\}$.

Solution

(a) Now, $\{1\}$, $\{0\}$ are represented by **1** and **0**, respectively. 101 is obtained by concatenating 1, 0 and 1. So, $\{101\}$ is represented by **101**.

(b) **abba** represents $\{abba\}$.
(c) As $\{01, 10\}$ is the union of $\{01\}$ and $\{10\}$, we have $\{01, 10\}$ represented by $\mathbf{01 + 10}$.
(d) The set $\{\Lambda, abb\}$ is represented by $\mathbf{abb + a + b + bba}$.
(e) The set $\{abb, a, b, bba\}$ is represented by $\mathbf{abb + a + b + bba}$.
(f) As $\{\Lambda, 0, 00, 000, \ldots\}$ is simply $\{0\}^*$, it is represented by $\mathbf{0^*}$.
(g) Any element in $\{1, 11, 111, \ldots\}$ can be obtained by concatenating 1 and any element of $\{1\}^*$. Hence $\mathbf{1(1)^*}$ represents $\{1, 11, 111, \ldots\}$.

EXAMPLE 5.2

Describe the following sets by regular expressions:
(a) L_1 = the set of all strings of 0's and 1's ending in 00.
(b) L_2 = the set of all strings of 0's and 1's beginning with 0 and ending with 1.
(c) $L_3 = \{\Lambda, 11, 1111, 111111, \ldots\}$.

Solution

(a) Any string in L_1 is obtained by concatenating any string over $\{0, 1\}$ and the string 00. $\{0, 1\}$ is represented by $\mathbf{0 + 1}$. Hence L_1 is represented by $\mathbf{(0 + 1)^* 00}$.
(b) As any element of L_2 is obtained by concatenating 0, any string over $\{0, 1\}$ and 1, L_2 can be represented by $\mathbf{0(0 + 1)^* 1}$.
(c) Any element of L_3 is either Λ or a string of even number of 1's, i.e. a string of the form $(11)^n$, $n \geq 0$. So L_3 can be represented by $\mathbf{(11)^*}$.

5.1.1 IDENTITIES FOR REGULAR EXPRESSIONS

Two regular expressions **P** and **Q** are equivalent (we write **P = Q**) if **P** and **Q** represent the same set of strings.

We now give the identities for regular expressions; these are useful for simplifying regular expressions.

I_1 $\quad \emptyset + \mathbf{R} = \mathbf{R}$
I_2 $\quad \emptyset\mathbf{R} = \mathbf{R}\emptyset = \emptyset$
I_3 $\quad \Lambda\mathbf{R} = \mathbf{R}\Lambda = \mathbf{R}$
I_4 $\quad \Lambda^* = \Lambda$ and $\emptyset^* = \Lambda$
I_5 $\quad \mathbf{R + R = R}$
I_6 $\quad \mathbf{R^*R^* = R^*}$
I_7 $\quad \mathbf{RR^* = R^*R}$
I_8 $\quad \mathbf{(R^*)^* = R^*}$
I_9 $\quad \Lambda + \mathbf{RR^* = R^* = \Lambda + R^*R}$
I_{10} $\quad \mathbf{(PQ)^*P = P(QP)^*}$

I_{11} $\quad \mathbf{(P + Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*}$
I_{12} $\quad \mathbf{(P + Q)R = PR + QR}$ and $\mathbf{R(P + Q) = RP + RQ}$

Note: By the 'set **P**' we mean the set represented by the regular expression **P**.

The following theorem is very much useful in simplifying regular expressions (i.e. replacing a given regular expression **P** by a simpler regular expression equivalent to **P**).

Theorem 5.1 (Arden's theorem) Let **P** and **Q** be two regular expressions over Σ. If **P** does not contain Λ, then the following equation in **R**, namely

$$\mathbf{R = Q + RP} \tag{5.1}$$

has a unique solution (i.e. one and only one solution) given by $\mathbf{R = QP^*}$.

Proof $\mathbf{Q + (QP^*)P = Q(\Lambda + P^*P) = QP^*}$ by I_9

Hence (5.1) is satisfied when $\mathbf{R = QP^*}$. This means $\mathbf{R = QP^*}$ is a solution of (5.1).

To prove uniqueness, consider (5.1). Here, replacing **R** by **Q + RP** on the R.H.S., we get the equation

$$\begin{aligned}\mathbf{Q + RP} &= \mathbf{Q + (Q + RP)P}\\ &= \mathbf{Q + QP + RPP}\\ &= \mathbf{Q + QP + RP^2}\\ &= \mathbf{Q + QP + QP^2 + \cdots + QP^i + RP^{i+1}}\\ &= \mathbf{Q(\Lambda + P + P^2 + \cdots + P^i) + RP^{i+1}}\end{aligned}$$

From (5.1),

$$\mathbf{R = Q(\Lambda + P + P^2 + \cdots + P^i) + RP^{i+1}} \quad \text{for } i \geq 0 \tag{5.2}$$

We now show that any solution of (5.1) is equivalent to $\mathbf{QP^*}$. Suppose **R** satisfies (5.1), then it satisfies (5.2). Let w be a string of length i in the set **R**. Then w belongs to the set $Q(\Lambda + P + P^2 + \ldots + P^i) + RP^{i+1}$. As **P** does not contain Λ, $\mathbf{RP^{i+1}}$ has no string of length less than $i + 1$ and so w is not in the set $\mathbf{RP^{i+1}}$. This means that w belongs to the set $Q(\Lambda + P + P^2 + \ldots + P^i)$, and hence to $\mathbf{QP^*}$.

Consider a string w in the set $\mathbf{QP^*}$. Then w is in the set $\mathbf{QP^k}$ for some $k \geq 0$, and hence in $Q(\Lambda + P + P^2 + \cdots + P^k)$. So w is on the R.H.S. of (5.2). Therefore, w is in **R** (L.H.S. of (5.2)). Thus **R** and $\mathbf{QP^*}$ represent the same set. This proves the uniqueness of the solution of (5.1). ∎

Note: Henceforth in this text, the regular expressions will be abbreviated r.e.

Example 5.3

(a) Give an r.e. for representing the set L of strings in which every 0 is immediately followed by at least two 1's.
(b) Prove that the regular expression $\mathbf{R = \Lambda + 1^*(011)^*(1^*(011)^*)^*}$ also describes the same set of strings.

Solution

(a) If w is in L, then either (a) w does not contain any 0, or (b) it contains a 0 preceded by 1 and followed by 11. So w can be written as $w_1w_2 \ldots w_n$, where each w_i is either 1 or 011. So L is represented by the r.e. $(1 + 011)^*$.

(b) $R = \Lambda + P_1P_1^*$, where $P_1 = 1^*(011)^*$

$= P_1^*$ using I_9

$= (1^*(011)^*)^*$

$= (P_2^*P_3^*)^*$ letting $P_2 = 1$, $P_3 = 011$

$= (P_2 + P_3)^*$ using I_{11}

$= (1 + 011)^*$

EXAMPLE 5.4

Prove $(1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^* (0 + 10^*1) = 0^*1(0 + 10^*1)^*$.

Solution

L.H.S. $= (1 + 00^*1)(\Lambda + (0 + 10^*1)^* (0 + 10^*1)$ using I_{12}

$= (1 + 00^*1)(0 + 10^*1)^*$ using I_9

$= (\Lambda + 00^*)1 (0 + 10^*1)^*$ using I_{12} for $1 + 00^*1$

$= 0^*1(0 + 10^*1)^*$ using I_9

= R.H.S.

5.2 FINITE AUTOMATA AND REGULAR EXPRESSIONS

In this section we study regular expressions and their representation.

5.2.1 TRANSITION SYSTEM CONTAINING Λ-MOVES

The transition systems can be generalized by permitting Λ-transitions or Λ-moves which are associated with a null symbol Λ. These transitions can occur when no input is applied. But it is possible to convert a transition system with Λ-moves into an equivalent transition system without Λ-moves. We shall give a simple method of doing it with the help of an example.

Suppose we want to replace a Λ-move from vertex v_1 to vertex v_2. Then we proceed as follows:

Step 1 Find all the edges starting from v_2.

Step 2 Duplicate all these edges starting from v_1, without changing the edge labels.

Step 3 If v_1 is an initial state, make v_2 also as initial state.

Step 4 If v_2 is a final state, make v_1 also as the final state.

EXAMPLE 5.5

Consider a finite automaton, with Λ-moves, given in Fig. 5.1. Obtain an equivalent automaton without Λ-moves.

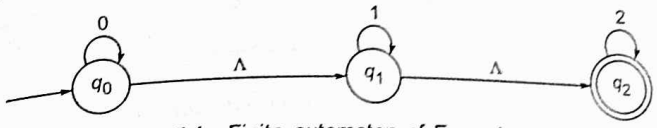

Fig. 5.1 Finite automaton of Example 5.5.

Solution

We first eliminate the Λ-move from q_0 to q_1 to get Fig. 5.2(a). q_1 is made an initial state. Then we eliminate the Λ-move from q_0 to q_2 in Fig. 5.2(a) to get Fig. 5.2(b). As q_2 is a final state, q_0 is also made a final state. Finally, the Λ-move from q_1 to q_2 is eliminated in Fig. 5.2(c).

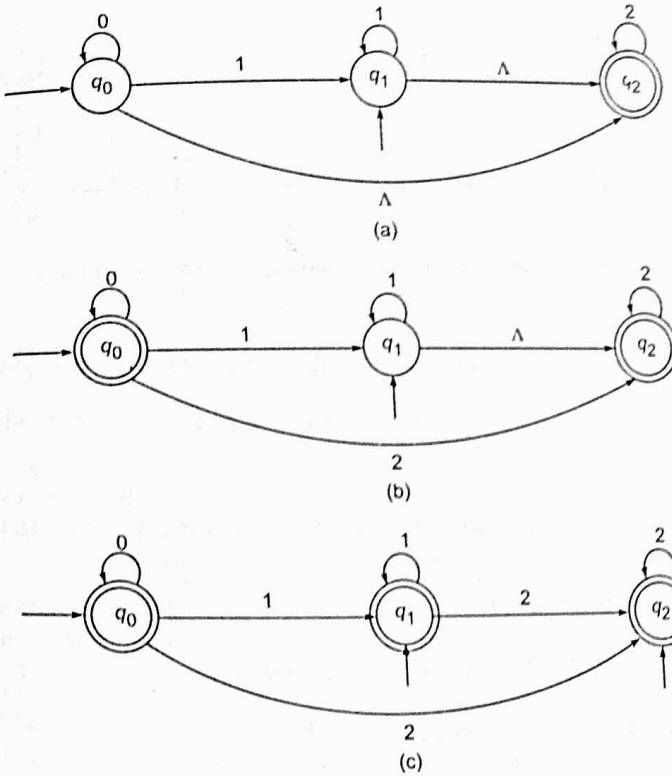

Fig. 5.2 Transition system for Example 5.5, without Λ-moves.

EXAMPLE 5.6

Consider a graph (i.e. transition system), containing a Λ-move, given in Fig. 5.3. Obtain an equivalent graph (i.e. transition system) without Λ-moves.

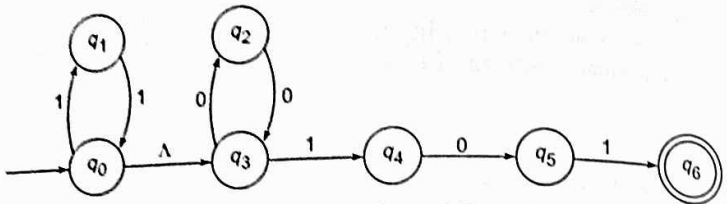

Fig. 5.3 Finite automaton of Example 5.6.

Solution

There is a Λ-move from q_0 to q_3. There are two edges, one from q_3 to q_2 with label 0 and another from q_3 to q_4 with label 1. We duplicate these edges from q_0. As q_0 is an initial state, q_3 is made an initial state. The resulting transition graph is given in Fig. 5.4.

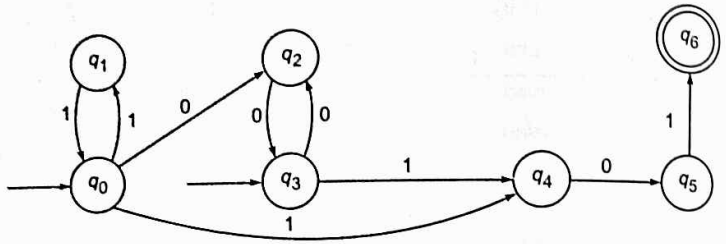

Fig. 5.4 Transition system for Example 5.6, without Λ-moves.

5.2.2 NDFAs WITH Λ-MOVES AND REGULAR EXPRESSIONS

In this section, we prove that every regular expression is recognized by a nondeterministic finite automaton (NDFA) with Λ-moves.

Theorem 5.2 (Kleene's theorem) If **R** is a regular expression over Σ representing $L \subseteq \Sigma^*$, then there exists an NDFA M with Λ-moves such that $L = T(M)$.

Proof The proof is by the principle of induction on the total number of characters in **R**. By 'character' we mean the elements of Σ, Λ, ∅, * and +. For example, if **R** = Λ + **10*11*0**, the characters are Λ, +, 1, 0, *, 1, 1, *, 0, and the number of characters is 9.

Let $L(\mathbf{R})$ denote the set represented by **R**.

Basis. Let the number of characters in **R** be 1. Then **R** = Λ, or **R** = ∅, or **R** = a_i, $a_i \in \Sigma$. The transition systems given in Fig. 5.5 will recognize these regular expressions.

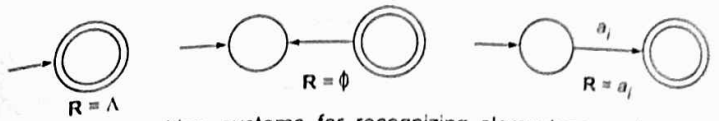

Fig. 5.5 Transition systems for recognizing elementary regular sets.

Induction step. Assume that the theorem is true for regular expressions having n characters. Let **R** be a regular expression having $n + 1$ characters. Then,

$$\mathbf{R} = \mathbf{P} + \mathbf{Q} \quad \text{or} \quad \mathbf{R} = \mathbf{PQ} \quad \text{or} \quad \mathbf{R} = \mathbf{P}^*$$

according as the last operator in **R** is +, product or closure. Also **P** and **Q** are regular expressions having n characters or less. By induction hypothesis, $L(\mathbf{P})$ and $L(\mathbf{Q})$ are recognized by M_1 and M_2 where M_1 and M_2 are NDFAs with Λ-moves, such that $L(\mathbf{P}) = T(M_1)$ and $L(\mathbf{Q}) = T(M_2)$. M_1 and M_2 are represented in Fig. 5.6.

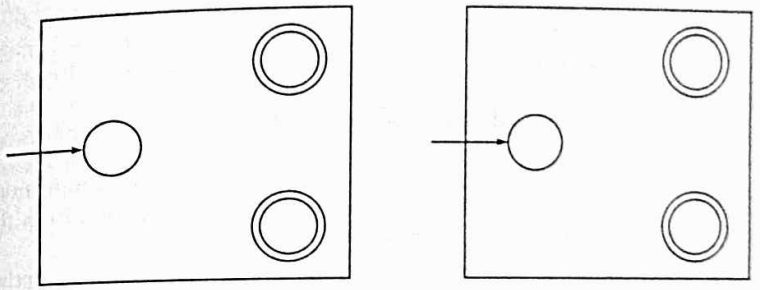

Fig. 5.6 Nondeterministic finite automata M_1 and M_2.

The initial state and the final states of M_1 and M_2 are represented in the usual way.

Case 1 **R = P + Q.** In this case we construct an NDFA M with Λ-moves that accepts $L(\mathbf{P} + \mathbf{Q})$ as follows: q_0 is the initial state of M, q_0 not in M_1 or M_2. q_f is the final state of M; once again q_f not in M_1 or M_2. M contains all the states of M_1 and M_2 and also their transitions. We add additional Λ-transitions from q_0 to the initial states of M_1 and M_2 and from the final states of M_1 and M_2 to q_f. The NDFA M is as in Fig. 5.7. It is easy to see that $T(M) = T(M_1) \cup T(M_2) = L(\mathbf{P} + \mathbf{Q})$.

Case 2 **R = PQ.** In this case we introduce q_0 as the initial state of M and q_f as the final state of M, both q_0, q_f not in M_1 or M_2. New Λ-transitions are added between q_0 and the initial state of M_1, between final states of M_1 and the initial state of M_2, and between final states of M_2 and the final state q_f of M. See Fig. 5.8.

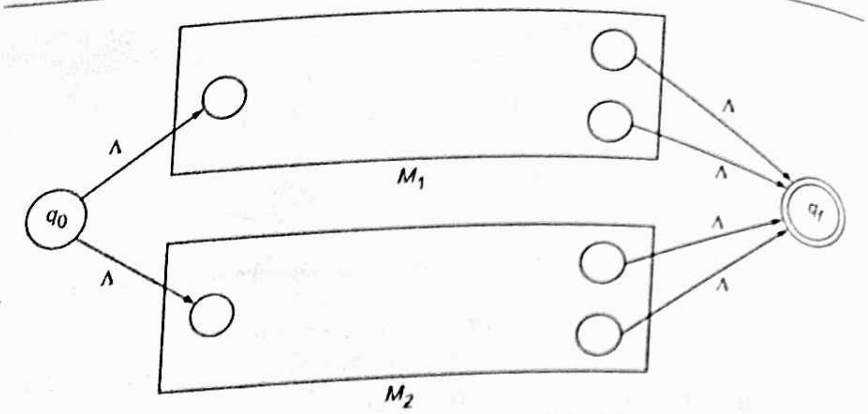

Fig. 5.7 NDFA accepting $L(P + Q)$.

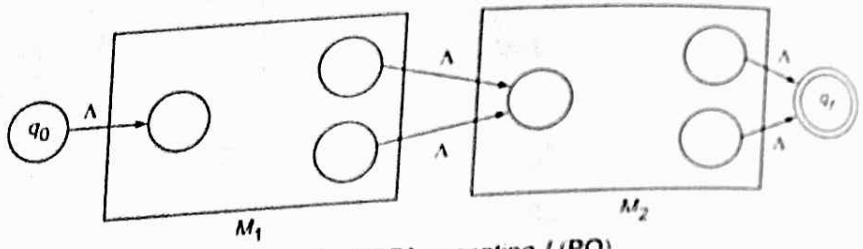

Fig. 5.8 NDFA accepting $L(PQ)$.

Case 3 $R = (P)^*$. In this case, q_0, q and q_f are introduced. New Λ-transitions are introduced from q_0 to q, q to q_f, q to the initial state of M_1 and from the final states of M_1 to q. See Fig. 5.9.

Thus in all the cases, there exists an NDFA M with Λ-moves, accepting the regular expression **R** with $n + 1$ characters. By the principle of induction, this theorem is true for all regular expressions. ∎

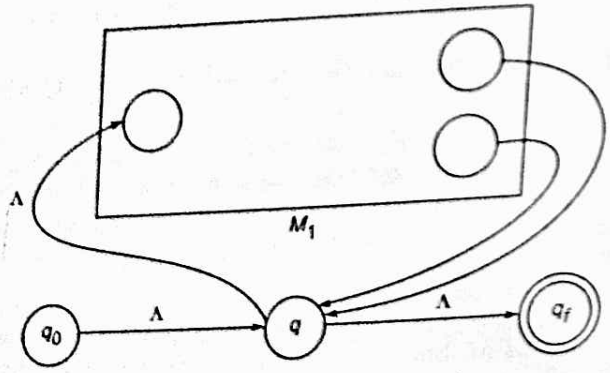

Fig. 5.9 NDFA accepting $L(P^*)$.

Theorem 5.1 gives a method of constructing NDFAs accepting **P** + **Q**, **PQ** and **P*** using the NDFAs corresponding to **P** and **Q**. In the later sections we give a method of converting NDFA M with Λ-moves into an NDFA M_1 without Λ-moves and then into a DFA M_2 such that $T(M) = T(M_1) = T(M_2)$. Thus, if a regular expression **P** is given, we can construct a DFA accepting $L(P)$.

The following theorem is regarding the converse. Both the Theorems 5.2 and 5.3 prove the equivalence of regular expressions or regular sets and the sets accepted by deterministic finite automata.

Theorem 5.3 Any set L accepted by a finite automaton M is represented by a regular expression.

Proof Let

$$M = (\{q_1 \ldots q_m\}, \Sigma, \delta, q_1, F)$$

The construction that we give can be better understood in terms of the state diagram of M. If a string $w \in \Sigma^*$ is accepted by M, then there is a path from q_1 to some final state with path value w. So to each final state, say q_j, there corresponds a subset of Σ^* consisting of path values of paths from q_0 to q_j. As $T(M)$ is the union of such subsets of Σ^*, it is enough to represent them by regular expressions. So the main part of the proof lies in the construction of subsets of path values of paths from the state q_i to the state q_j.

Let P_{ij}^k denote the set of path values of paths from q_i to q_j whose intermediate vertices lie in $\{q_1, \ldots, q_k\}$. We construct P_{ij}^k for $k = 0, 1, \ldots, n$ recursively as follows:

$$P_{ij}^0 = \{a \in \Sigma \mid \delta(q_i, a) = q_j\} \tag{5.3}$$

$$P_{ii}^0 = \{a \in \Sigma \mid \delta(q_i, a) = q_i\} \cup \{\Lambda\} \tag{5.4}$$

$$P_{ij}^k = P_{ik}^{k-1}(P_{kk}^{k-1})^* P_{kj}^{k-1} \cup P_{ij}^{k-1} \tag{5.5}$$

In terms of the state diagram, the construction can be understood better. P_{ij}^0 simply denotes the set of path values (i.e. labels) of edges from q_i to q_j. In P_{ii}^0 we include Λ in addition to labels of self-loops from q_i. This explains (5.3) and (5.4).

Consider a path from q_i to q_j whose intermediate vertices lie in $\{q_1, \ldots, q_k\}$. If the path does not pass through q_k, then its path value lies in P_{ij}^{k-1}. Otherwise, the path passes through q_k possibly more than once. The path can be split into several paths with path values $w_1, w_2 \ldots w_l$ as in Fig. 5.10. $w = w_1 w_2 \ldots w_l$. w_1 is the path value of the path from q_i to q_k (without passing through q_k, i.e. q_k is not an intermediate vertex). $w_2, \ldots, w_{l-1}$ are the path values of paths from q_k to itself without passing through q_k. w_l is the path value of the path from q_k to q_j without passing through q_k. So w_1 is in P_{ik}^{k-1}, $w_2, \ldots, w_{l-1}$ are in (P_{kk}^{k-1}), and w_l is in P_{kj}^{k-1}. This explains (5.5).

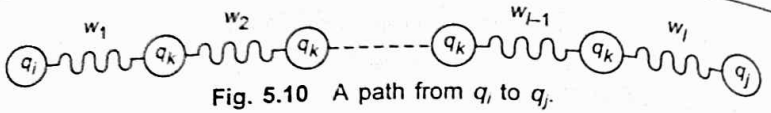

Fig. 5.10 A path from q_i to q_j.

We prove that the sets introduced by (5.3)–(5.5) are represented by regular expressions by induction on k (for all i and j). P_{ij}^0 is a finite subset of Σ, say $\{a_1, \ldots, a_r\}$. Then, P_{ij}^0 is represented by $\mathbf{P}_{ij}^0 = a_1 + a_2 + \cdots + a_r$. Similarly, we can construct $\mathbf{P}_{ii}^0$ representing P_{ii}^0. Thus, there is basis for induction.

Let us assume the result for $k-1$, i.e. P_{ij}^{k-1} is represented by a regular expression $\mathbf{P}_{ij}^{k-1}$ for all i and j. From (5.5), we have $P_{ij}^k = P_{ik}^{k-1}(P_{kk}^{k-1})^* P_{kj}^{k-1} \cup P_{ij}^{k-1}$. So it is obvious that P_{ij}^k is represented by $\mathbf{P}_{ij}^k = \mathbf{P}_{ik}^{k-1}(\mathbf{P}_{kk}^{k-1})^* \mathbf{P}_{kj}^{k-1} \cup \mathbf{P}_{ij}^{k-1}$. Therefore, the result is true for all k. By the principle of induction, the sets constructed by (5.3)–(5.5) are represented by regular expressions.

As $Q = \{q_1, \ldots, q_m\}$, $\mathbf{P}_{1j}^m$ denotes the set of path values of all paths from q_1 to q_j. If $F = \{q_{f_1}, \ldots, q_{f_n}\}$, then $T(M) = \bigcup_{j=1}^{n} P_{1f_j}^m$. So $T(M)$ is represented by the regular expression $\mathbf{P}_{1f_1}^m + \cdots + \mathbf{P}_{1f_n}^m$. Thus, $L = T(M)$ is represented by a regular expression.

Note: P_{ij}^0 and P_{ii}^0 are the subsets of $\Sigma \cup \{\Lambda\}$, and so they are finite sets. So every P_{ij}^k is obtained by applying union, concatenation and closure to the set of all singletons in $\Sigma \cup \{\Lambda\}$. Using this we prove Kleene's theorem (Theorem 5.4) at the end of this section. Kleene's theorem characterizes the regular sets in terms of subsets of Σ and operations (union, concatenation, closure) on singletons in $\Sigma \cup \{\Lambda\}$.

5.2.3 Conversion of Nondeterministic Systems to Deterministic Systems

The construction we are going to give is similar to the construction of a DFA equivalent to an NDFA and involves three steps.

Step 1 Convert the given transition system into state transition table where each state corresponds to a row and each input symbol corresponds to a column.

Step 2 Construct the successor table which lists the subsets of states reachable from the set of initial states. Denote this collection of subsets by Q'.

Step 3 The transition graph given by the successor table is the required deterministic system. The final states contain some final state of NDFA. If possible, reduce the number of states.

Note: The construction is similar to that given in Section 3.7 for automata except for the initial step. In the earlier method for automata, we started with $[q_0]$. Here we start with the set of all initial states. The other steps are similar.

EXAMPLE 5.7

Obtain the deterministic graph (system) equivalent to the transition system given in Fig. 5.11.

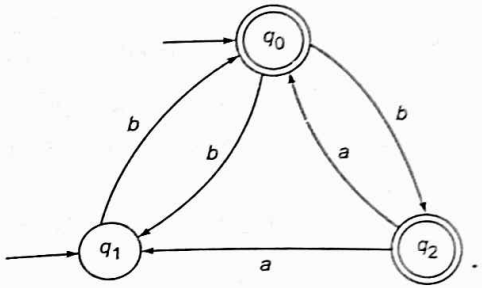

Fig. 5.11 Nondeterministic transition system of Example 5.7.

Solution

We construct the transition table corresponding to the given nondeterministic system. It is given in Table 5.1.

TABLE 5.1 Transition Table for Example 5.7

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

We construct the successor table by starting with $[q_0, q_1]$. From Table 5.1 we see that $[q_0, q_1, q_2]$ is reachable from $[q_0, q_1]$ by a b-path. There are no a-paths from $[q_0, q_1]$. Similarly, $[q_0, q_1]$ is reachable from $[q_0, q_1, q_2]$ by an a-path and $[q_0, q_1, q_2]$ is reachable from itself. We proceed with the construction for all the elements in Q'.

We terminate the construction when all the elements of Q' appear in the successor table. Table 5.2 gives the successor table. From the successor table it is easy to construct the deterministic transition system described by Fig. 5.12

TABLE 5.2 Deterministic Transition Table for Example 5.7

Q	a	b
$[q_0, q_1]$	$\emptyset$	$[q_0, q_1, q_2]$
$[q_0, q_1, q_2]$	$[q_0, q_1]$	$[q_0, q_1, q_2]$
$\emptyset$	$\emptyset$	$\emptyset$

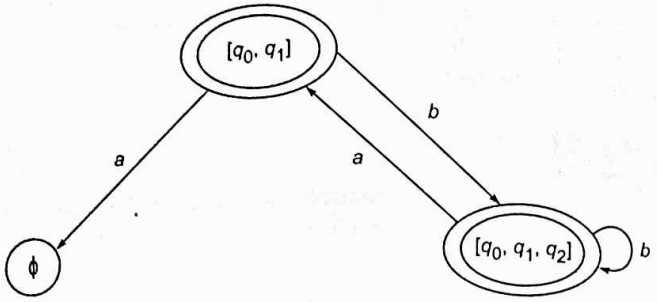

Fig. 5.12 Deterministic transition system for Example 5.7.

as q_0 and q_2 are the final states of the nondeterministic system $[q_0, q_1]$ and $[q_0, q_1, q_2]$ are the final states of the deterministic system.

5.2.4 Algebraic Method Using Arden's Theorem

The following method is an extension of the Arden's theorem (Theorem 5.1). This is used to find the r.e. recognized by a transition system.

The following assumptions are made regarding the transition system:

(i) The transition graph does not have Λ-moves.
(ii) It has only one initial state, say v_1.
(iii) Its vertices are $v_1 \ldots v_n$.
(iv) $\mathbf{V}_i$ the r.e. represents the set of strings accepted by the system even though v_i is a final state.
(v) α_{ij} denotes the r.e. representing the set of labels of edges from v_i to v_j. When there is no such edge, $\alpha_{ij} = \emptyset$. Consequently, we can get the following set of equations in $\mathbf{V}_1 \ldots \mathbf{V}_n$:

$$\mathbf{V}_1 = \mathbf{V}_1\alpha_{11} + \mathbf{V}_2\alpha_{21} + \cdots + \mathbf{V}_n\alpha_{n1} + \Lambda$$
$$\mathbf{V}_2 = \mathbf{V}_1\alpha_{12} + \mathbf{V}_2\alpha_{22} + \cdots + \mathbf{V}_n\alpha_{n2}$$
$$\vdots$$
$$\mathbf{V}_n = \mathbf{V}_1\alpha_{1n} + \mathbf{V}_2\alpha_{2n} + \cdots + \mathbf{V}_n\alpha_{nn}$$

By repeatedly applying substitutions and Theorem 5.1 (Arden's theorem), we can express $\mathbf{V}_i$ in terms of α_{ij}'s.

For getting the set of strings recognized by the transition system, we have to take the 'union' of all $\mathbf{V}_i$'s corresponding to final states.

EXAMPLE 5.8

Consider the transition system given in Fig. 5.13. Prove that the strings recognized are **(a + a(b + aa)*b)* a(b + aa)* a**.

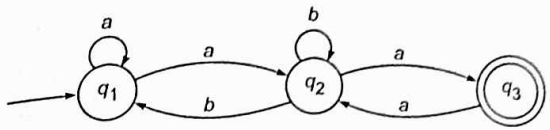

Fig. 5.13 Transition system of Example 5.8.

Solution

We can directly apply the above method since the graph does not contain any Λ-move and there is only one initial state.

The three equations for q_1, q_2 and q_3 can be written as

$$\mathbf{q}_1 = \mathbf{q}_1\mathbf{a} + \mathbf{q}_2\mathbf{b} + \Lambda, \qquad \mathbf{q}_2 = \mathbf{q}_1\mathbf{a} + \mathbf{q}_2\mathbf{b} + \mathbf{q}_3\mathbf{a}, \qquad \mathbf{q}_3 = \mathbf{q}_2\mathbf{a}$$

It is necessary to reduce the number of unknowns by repeated substitution. By substituting q_3 in the q_2-equation, we get by applying Theorem 5.1

$$\begin{aligned}\mathbf{q}_2 &= \mathbf{q}_1\mathbf{a} + \mathbf{q}_2\mathbf{b} + \mathbf{q}_2\mathbf{aa}\\ &= \mathbf{q}_1\mathbf{a} + \mathbf{q}_2(\mathbf{b} + \mathbf{aa})\\ &= \mathbf{q}_1\mathbf{a}(\mathbf{b} + \mathbf{aa})^*\end{aligned}$$

Substituting $\mathbf{q}_2$ in $\mathbf{q}_1$, we get

$$\begin{aligned}\mathbf{q}_1 &= \mathbf{q}_1\mathbf{a} + \mathbf{q}_1\mathbf{a}(\mathbf{b} + \mathbf{aa})^*\mathbf{b} + \Lambda\\ &= \mathbf{q}_1(\mathbf{a} + \mathbf{a}(\mathbf{b} + \mathbf{aa})^*\mathbf{b}) + \Lambda\end{aligned}$$

Hence,

$$\begin{aligned}\mathbf{q}_1 &= \Lambda(\mathbf{a} + \mathbf{a}(\mathbf{b} + \mathbf{aa})^*\mathbf{b})^*\\ \mathbf{q}_2 &= (\mathbf{a} + \mathbf{a}(\mathbf{b} + \mathbf{aa})^*\mathbf{b})^*\ \mathbf{a}(\mathbf{b} + \mathbf{aa})^*\\ \mathbf{q}_3 &= (\mathbf{a} + \mathbf{a}(\mathbf{b} + \mathbf{aa})^*\mathbf{b})^*\ \mathbf{a}(\mathbf{b} + \mathbf{aa})^*\mathbf{a}\end{aligned}$$

Since q_3 is a final state, the set of strings recognized by the graph is given by

$$(\mathbf{a} + \mathbf{a}(\mathbf{b} + \mathbf{aa})^*\mathbf{b})^*\mathbf{a}(\mathbf{b} + \mathbf{aa})^*\mathbf{a}$$

EXAMPLE 5.9

Prove that the finite automaton whose transition diagram is as shown in Fig. 5.14 accepts the set of all strings over the alphabet $\{a, b\}$ with an equal number of a's and b's, such that each prefix has at most one more a than the b's and at most one more b than the a's.

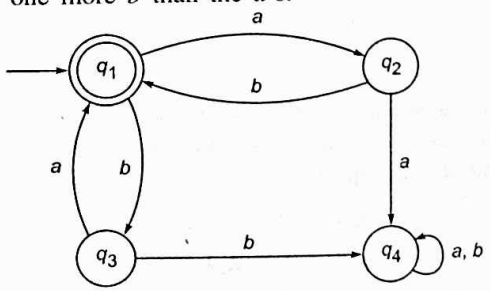

Fig. 5.14 Finite automaton of Example 5.9.

Solution

We can apply the above method directly since the graph does not contain the Λ-move and there is only one initial state. We get the following equations for q_1, q_2, q_3, q_4:

$$q_1 = q_2b + q_3a + \Lambda$$
$$q_2 = q_1a$$
$$q_3 = q_1b$$
$$q_4 = q_2a + q_3b + q_4a + q_4b$$

As q_1 is the only final state and the q_1-equation involves only q_2 and q_3, we use only q_2- and q_3-equations (the q_4-equation is redundant for our purposes). Substituting for q_2 and q_3, we get

$$q_1 = q_1ab + q_1ba + \Lambda = q_1(ab + ba) + \Lambda$$

By applying Theorem 5.1, we get

$$q_1 = \Lambda(ab + ba)^* = (ab + ba)^*$$

As q_1 is the only final state, the strings accepted by the given finite automaton are the strings given by $(ab + ba)^*$. As any such string is a string of *ab*'s, and *ba*'s, we get an equal number of *a*'s and *b*'s. If a prefix *x* of a sentence accepted by the finite automaton has an even number of symbols, then it should have an equal number of *a*'s and *b*'s since *x* is a substring formed by *ab*'s and *ba*'s. If the prefix *x* has an odd number of symbols, then we can write *x* as *ya* or *yb*. As *y* has an even number of symbols, *y* has an equal number of *a*'s and *b*'s. Thus, *x* has one more *a* than *b* or vice versa.

EXAMPLE 5.10

Describe in English the set accepted by the finite automaton whose transition diagram is as shown in Fig. 5.15.

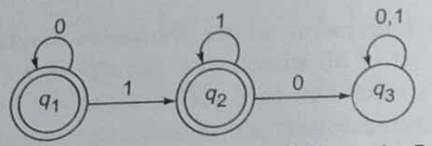

Fig. 5.15 Finite automaton of Example 5.10.

Solution

We can apply the above method directly as the transition diagram does not contain more than one initial state and there are no Λ-moves. We get the following equations for q_1, q_2, q_3.

$$q_1 = q_10 + \Lambda$$
$$q_2 = q_11 + q_21$$
$$q_3 = q_20 + q_3(0 + 1)$$

By applying Theorem 5.1 to the q_1-equation, we get

$$q_1 = \Lambda0^* = 0^*$$

So,
$$q_2 = q_11 + q_21 = 0^*1 + q_21$$

Therefore,
$$q_2 = (0^*1)1^*$$

As the final states are q_1 and q_2, we need not solve for q_3:

$$q_1 + q_2 = 0^* + 0^*(11^*) = 0^*(\Lambda + 11^*) = 0^*(1^*) \quad \text{by } I_9$$

The strings represented by the transition graph are 0^*1^*. We can interpret the strings in the English language in the following way: The strings accepted by the finite automaton are precisely the strings of any number of 0's (possibly Λ) followed by a string of any number of 1's (possibly Λ).

EXAMPLE 5.11

Construct a regular expression corresponding to the state diagram described by Fig. 5.16.

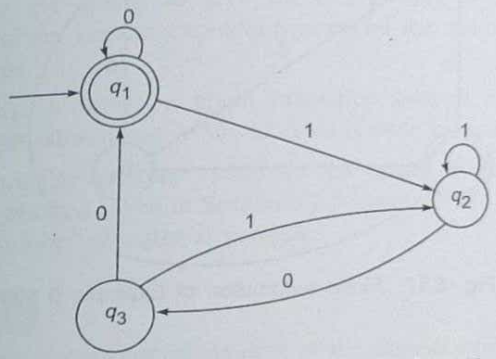

Fig. 5.16 Finite automaton of Example 5.11.

Solution

There is only one initial state. Also, there are no Λ-moves. The equations are

$$q_1 = q_10 + q_30 + \Lambda$$
$$q_2 = q_11 + q_21 + q_31$$
$$q_3 = q_20$$

So,
$$q_2 = q_11 + q_21 + (q_20)1 = q_11 + q_2(1 + 01)$$

By applying Theorem 5.1, we get

$$q_2 = q_11(1 + 01)^*$$

Also,

$$q_1 = q_1 0 + q_3 0 + \Lambda = q_1 0 + q_2 00 + \Lambda$$
$$= q_1 0 + (q_1 1(1 + 01)^*)00 + \Lambda$$
$$= q_1(0 + 1(1 + 01)^* 00) + \Lambda$$

Once again applying Theorem 5.1, we get

$$q_1 = \Lambda(0 + 1(1 + 01)^* 00)^* = (0 + 1(1 + 01)^* 00)^*$$

As q_1 is the only final state, the regular expression corresponding to the given diagram is $(0 + 1(1 + 01)^* 00)^*$.

EXAMPLE 5.12

Find the regular expression corresponding to Fig. 5.17.

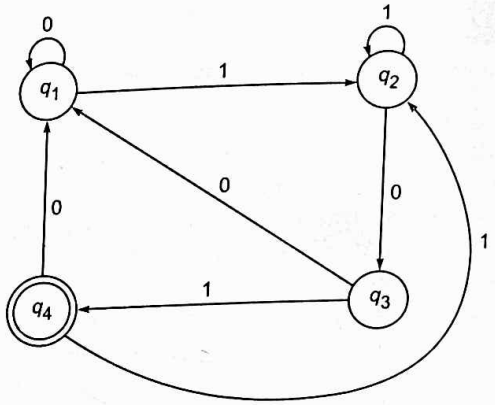

Fig. 5.17 Finite automaton of Example 5.12.

Solution

There is only one initial state, and there are no Λ-moves. So, we form the equations corresponding to q_1, q_2, q_3, q_4:

$$q_1 = q_1 0 + q_3 0 + q_4 0 + \Lambda$$
$$q_2 = q_1 1 + q_2 1 + q_4 1$$
$$q_3 = q_2 0$$
$$q_4 = q_3 1$$

Now,

$$q_4 = q_3 1 = (q_2 0)1 = q_2 01$$

Thus, we are able to write q_3, q_4 in terms of q_2. Using the q_2-equation, we get

$$q_2 = q_1 1 + q_2 1 + q_2 011 = q_1 1 + q_2(1 + 011)$$

By applying Theorem 5.1, we obtain

$$q_2 = (q_1 1)(1 + 011)^* = q_1(1(1 + 011)^*)$$

From the q_1-equation, we have

$$q_1 = q_1 0 + q_2 00 + q_2 010 + \Lambda$$
$$= q_1 0 + q_2(00 + 010) + \Lambda$$
$$= q_1 0 + q_1 1(1 + 011)^* (00 + 010) + \Lambda$$

Again, by applying Theorem 5.1, we obtain

$$q_1 = \Lambda(0 + 1(1 + 011)^* (00 + 010))^*$$
$$q_4 = q_2 01 = q_1 1(1 + 011)^* 01$$
$$= (0 + 1(1 + 011)^*(00 + 010))^*(1(1 + 011)^* 01)$$

5.2.5 Construction of Finite Automata Equivalent to a Regular Expression

The method we are going to give for constructing a finite automaton equivalent to a given regular expression is called the *subset method* which involves two steps.

Step 1 Construct a transition graph (transition system) equivalent to the given regular expression using Λ-moves. This is done by using Theorem 5.2.

Step 2 Construct the transition table for the transition graph obtained in step 1. Using the method given in Section 5.2.3, construct the equivalent DFA. We reduce the number of states if possible.

EXAMPLE 5.13

Construct the finite automaton equivalent to the regular expression

$$(0 + 1)^*(00 + 11)(0 + 1)^*$$

Solution

Step 1 (Construction of transition graph) First of all we construct the transition graph with Λ-moves using the constructions of Theorem 5.2. Then we eliminate Λ-moves as discussed in Section 5.2.1.

We start with Fig. 5.18(a).

We eliminate the concatenations in the given r.e. by introducing new vertices q_1 and q_2 and get Fig. 5.18(b).

We eliminate the $*$ operations in Fig. 5.18(b) by introducing two new vertices q_5 and q_6 and the Λ-moves as shown in Fig. 5.18(c).

We eliminate concatenations and + in Fig. 5.18(c) and get Fig. 5.18(d).

We eliminate the Λ-moves in Fig. 5.18(d) and get Fig. 5.18(e) which gives the NDFA equivalent to the given i.e.

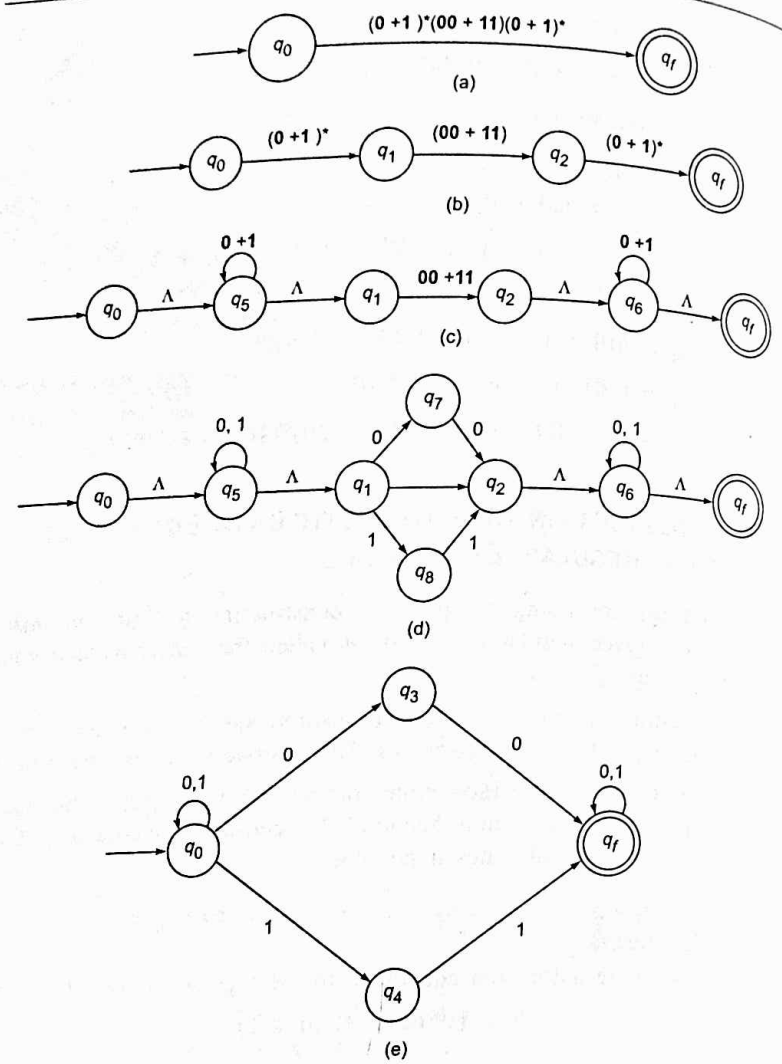

Fig. 5.18 Construction of finite automaton equivalent to **(0 + 1)*(00 + 11)(0 + 1)***.

Step 2 (Construction of DFA) We construct the transition table for the NDFA defined by Table 5.3.

TABLE 5.3 Transition Table for Example 5.13

State/Σ	0	1
→ q_0	q_0, q_3	q_0, q_4
q_3	q_f	
q_4		q_f
((q_f))	q_f	q_f

The successor table is constructed as given in Table 5.4.

TABLE 5.4 Transition Table for the DFA of Example 5.13.

Q	Q_0	Q_1
→ $[q_0]$	$[q_0, q_3]$	$[q_0, q_4]$
$[q_0, q_3]$	$[q_0, q_3, q_f]$	$[q_0, q_4]$
$[q_0, q_4]$	$[q_0, q_3]$	$[q_0, q_4, q_f]$
$[q_0, q_3, q_f]$	$[q_0, q_3, q_f]$	$[q_0, q_4, q_f]$
$[q_0, q_4, q_f]$	$[q_0, q_3, q_f]$	$[q_0, q_4, q_f]$

The state diagram for the successor table is the required DFA as described by Fig. 5.19. As q_f is the only final state of NDFA, $[q_0, q_3, q_f]$ and $[q_0, q_4, q_f]$ are the final states of DFA.

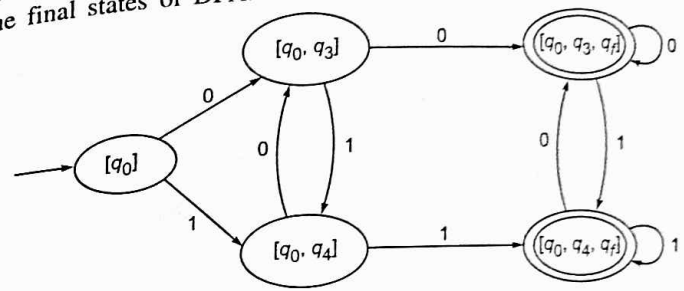

Fig. 5.19 Finite automaton of Example 5.13.

Finally, we try to reduce the number of states. (This is possible when two rows are identical in the successor table.) As the rows corresponding to $[q_0, q_3, q_f]$ and $[q_0, q_4, q_f]$ are identical, we identify them. The state diagram for the equivalent automaton, where the number of states is reduced, is described by Fig. 5.20.

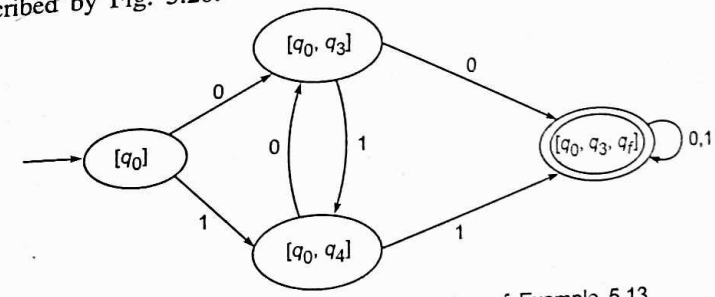

Fig. 5.20 Reduced finite automaton of Example 5.13.

Note: While constructing the transition graph equivalent to a given r.e., the operation (concatenation, *, +) that is eliminated first, depends on the regular expression.

EXAMPLE 5.14

Construct a DFA with reduced states equivalent to the r.e. $\mathbf{10 + (0 + 11))0^*1}$.

Solution

Step 1 (Construction of NDFA) The NDFA is constructed by eliminating the operation +, concatenation and *, and the Λ-moves in successive steps. The step-by-step construction is given in Figs. 5.21(a)–5.21(e).

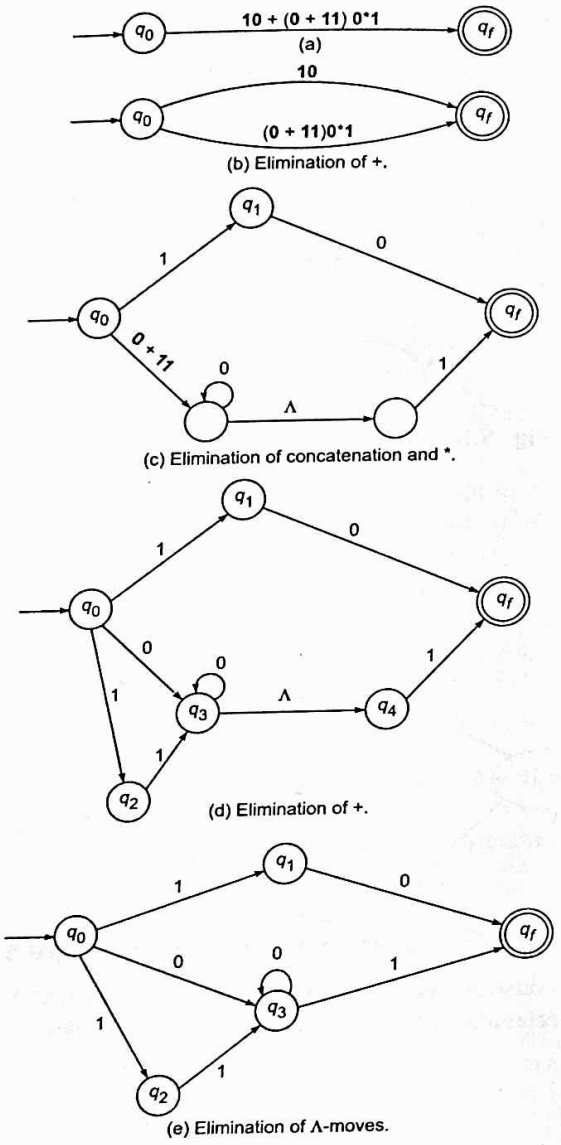

Fig. 5.21 Construction of finite automaton for Example 5.14.

Step 2 (Construction of DFA) For the NDFA given in Fig. 5.18(e), the corresponding transition table is defined by Table 5.5.

TABLE 5.5 Transition Table for Example 5.14

State/Σ	0	1
→ q_0	q_3	q_1, q_2
q_1	q_f	
q_2	—	q_3
q_3	q_3	q_f
(q_f)		

The successor table is constructed and given in Table 5.6.

In Table 5.6 the columns corresponding to $[q_f]$ and $\emptyset$ are identical. So we can identify $[q_f]$ and $\emptyset$.

TABLE 5.6 Transition Table of DFA for Example 5.14

Q	Q_0	Q_1
→ $[q_0]$	$[q_3]$	$[q_1, q_2]$
$[q_3]$	$[q_3]$	$[q_f]$
$[q_1, q_2]$	$[q_f]$	$[q_3]$
($[q_f]$)	$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$

The DFA with the reduced number of states corresponding to Table 5.6 is defined by Fig. 5.22.

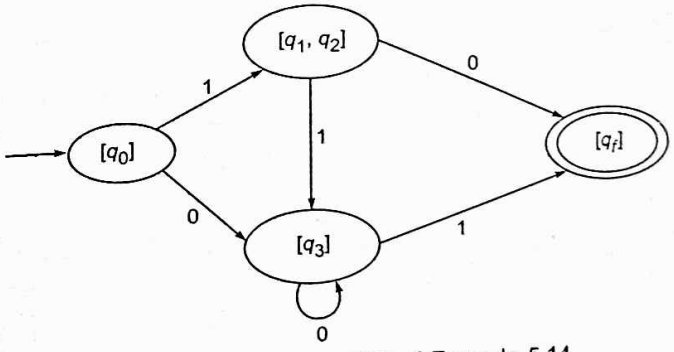

Fig. 5.22 Reduced DFA of Example 5.14.

5.2.6 EQUIVALENCE OF TWO FINITE AUTOMATA

Two finite automata over Σ are equivalent if they accept the same set of strings over Σ. When the two finite automata are not equivalent, there is some string

w over Σ satisfying the following: One automaton reaches a final state on application of w, whereas the other automaton reaches a nonfinal state.

We give below a method, called the *comparison method*, to test the equivalence of two finite automata over Σ.

Comparison Method

Let M and M' be two finite automata over Σ. We construct a comparison table consisting of $n + 1$ columns, where n is the number of input symbols. The first column consists of pairs of vertices of the form (q, q'), where $q \in M$ and $q' \in M'$. If (q, q') appears in some row of the first column, then the corresponding entry in the a-column ($a \in \Sigma$) is (q_a, q'_a), where q_a and q'_a are reachable from q and q', respectively on application of a (i.e. by a-paths).

The comparison table is constructed by starting with the pair of initial vertices q_{in}, q'_{in} of M and M' in the first column. The first elements in the subsequent columns are (q_a, q'_a), where q_a and q'_a are reachable by a-paths from q_{in} and q'_{in}. We repeat the construction by considering the pairs in the second and subsequent columns which are not in the first column.

The row-wise construction is repeated. There are two cases:

Case 1 If we reach a pair (q, q') such that q is a final state of M, and q' is a nonfinal state of M' or vice versa, we terminate the construction and conclude that M and M' are not equivalent.

Case 2 Here the construction is terminated when no new element appears in the second and subsequent columns which are not in the first column (i.e. when all the elements in the second and subsequent columns appear in the first column). In this case we conclude that M and M' are equivalent.

EXAMPLE 5.15

Consider the following two DFAs M and M' over $\{0, 1\}$ given in Fig. 5.23. Determine whether M and M' are equivalent.

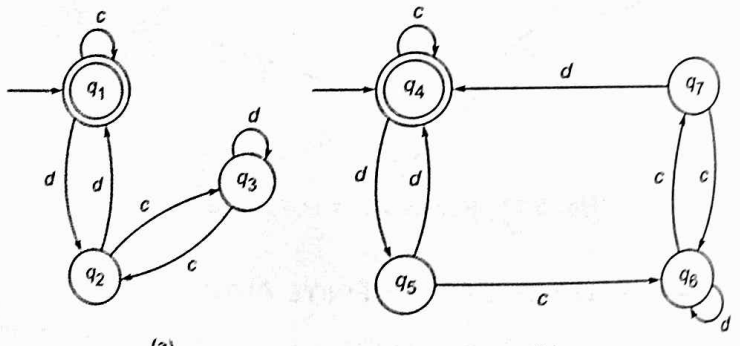

Fig. 5.23 (a) Automaton M and (b) automaton M'.

Solution

The initial states in M and M' are q_1 and q_4, respectively. Hence the first element of the first column in the comparison table must be (q_1, q_4). The first element in the second column is (q_1, q_4) since both q_1 and q_4 are c-reachable from the respective initial states. The complete table is given in Table 5.7.

TABLE 5.7 Comparison Table for Example 5.15

(q, q')	(q_c, q'_c)	(q_d, q'_d)
(q_1, q_4)	(q_1, q_4)	(q_2, q_5)
(q_2, q_5)	(q_3, q_6)	(q_1, q_4)
(q_3, q_6)	(q_2, q_7)	(q_3, q_6)
(q_2, q_7)	(q_3, q_6)	(q_1, q_4)

As we do not get a pair (q, q'), where q is a final state and q' is a nonfinal state (or vice versa) at every row, we proceed until all the elements in the second and third columns are also in the first column. Therefore, M and M' are equivalent.

EXAMPLE 5.16

Show that the automata M_1 and M_2 defined by Fig. 5.24 are not equivalent.

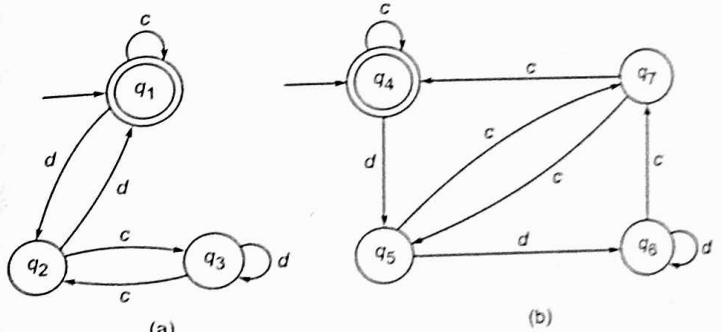

Fig. 5.24 (a) Automaton M_1 and (b) automaton M_2.

Solution

The initial states in M_1 and M_2 are q_1 and q_4, respectively. Hence the first column in the comparison table is (q_1, q_4). q_2 and q_5 are d-reachable from q_1 and q_4. We see from the comparison table given in Table 5.8 that q_1 and q_6 are d-reachable from q_2 and q_5, respectively. As q_1 is a final state in M_1, and q_6 is a nonfinal state in M_2, we see that M_1 and M_2 are not equivalent: we can also note that q_1 is dd-reachable from q_1, and hence dd is accepted by M_1. dd is not accepted by M_2 as only q_6 is dd-reachable from q_4, but q_6 is nonfinal.

TABLE 5.8 Comparison Table for Example 5.16

(q, q')	(q_c, q'_c)	(q_d, q'_d)
(q_1, q_4)	(q_1, q_4)	(q_2, q_5)
(q_2, q_5)	(q_3, q_7)	(q_1, q_6)

5.2.7 Equivalence of Two Regular Expressions

Suppose we are interested in testing the equivalence of two regular expressions, say **P** and **Q**. The regular expressions **P** and **Q** are equivalent iff they represent the same set. Also, **P** and **Q** are equivalent iff the corresponding finite automata are equivalent.

To prove the equivalence of **P** and **Q**, (i) we prove that the sets **P** and **Q** are the same. (For nonequivalence we find a string in one set but not in the other.) Or (ii) we use the identities to prove the equivalence of **P** and **Q**. Or (iii) we construct the corresponding FA M and M' and prove that M and M' are equivalent. (For nonequivalence we prove that M and M' are not equivalent.)

The method to be chosen depends on the problem.

EXAMPLE 5.17

Prove **(a + b)* = a*(ba*)***.

Solution

Let **P** and **Q** denote **(a + b)*** and **a*(ba*)***, respectively. Using the construction in Section 5.2.5, **P** is given by the transition system depicted in Fig. 5.25.

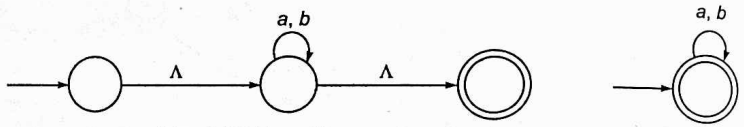

Fig. 5.25 Transition system for **(a + b)***.

The transition system for **Q** is depicted in Fig. 5.26.

It should be noted that Figs. 5.25 and 5.26 are obtained after eliminating Λ-moves. As these two transition diagrams are the same, we conclude that **P = Q**.

We now summarize all the results and constructions given in this section.

(i) Every r.e. is recognized by a transition system (Theorem 5.2).
(ii) A transition system M can be converted into a finite automaton accepting the same set as M (Section 5.2.3).
(iii) Any set accepted by finite automaton is represented by an r.e. (Theorem 5.3).
(iv) A set accepted by a transition system is represented by an r.e. (from (ii) and (iii)).

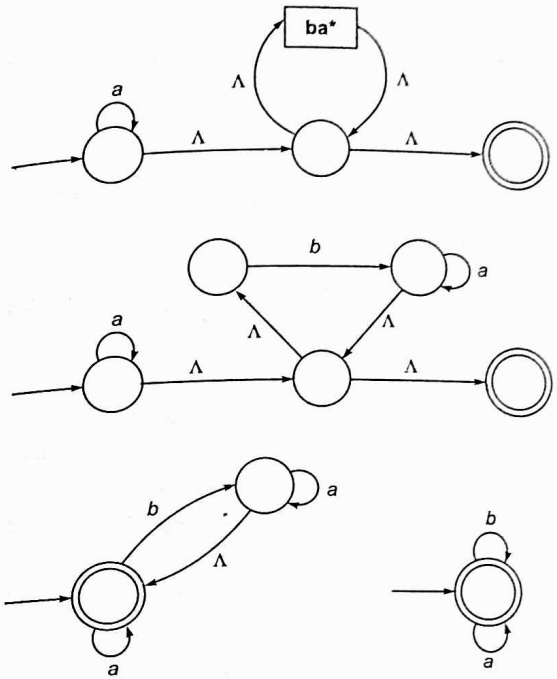

Fig. 5.26 Transition system for **a*(ba*)***.

(v) To get the r.e. representing a set accepted by a transition system, we can apply the algebraic method using the Arden's theorem (see Section 5.2.4).
(vi) If **P** is an r.e., then to construct a finite automaton accepting the set **P**, we can apply the construction given in Section 5.2.5.
(vii) A subset L of Σ^* is a regular set (or represented by an r.e.) iff it is accepted by an FA (from (i), (ii) and (iii)).
(viii) A subset L of Σ^* is a regular set iff it is recognized by a transition system (from (i) and (iv)).
(ix) The capabilities of finite automaton and transition systems are the same as far as acceptability of subsets of strings is concerned.
(x) To test the equivalence of two DFAs, we can apply the comparison method given in Section 5.2.6.

We conclude this section with the Kleene's theorem.

Theorem 5.4 (Kleene's theorem) The class of regular sets over Σ is the smallest class $\mathcal{R}$ containing $\{a\}$ for every $a \in \Sigma$ and closed under union, concatenation and closure.

Proof The set $\{a\}$ is represented by the regular expression **a**. So $\{a\}$ is regular for every $a \in \Sigma$. As the class of regular sets is closed under union, concatenation, and closure, $\mathcal{R}$ is contained in the class of regular sets.

Let L be a regular set. Then $L = T(M)$ for some DFA, $M = (\{q_1, \ldots, q_m\}, \Sigma, \delta, q_0, F)$. By Theorem 5.3,

$$L = \bigcup_{j=1}^{n} P_{1f_j}^m$$

where $F = \{q_{f_1} \ldots q_{f_n}\}$ and $P_{1f_j}^m$ is obtained by applying union, concatenation and closure to singletons in Σ. Thus, L is in $\mathcal{R}$. ∎

5.3 PUMPING LEMMA FOR REGULAR SETS

In this section we give a necessary condition for an input string to belong to a regular set. The result is called *pumping lemma* as it gives a method of pumping (generating) many input strings from a given string. As pumping lemma gives a necessary condition, it can be used to show that certain sets are not regular.

Theorem 5.5 (Pumping Lemma) Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton with n states. Let L be the regular set accepted by M. Let $w \in L$ and $|w| \geq m$. If $m \geq n$, then there exists x, y, z such that $w = xyz$, $y \neq \Lambda$ and $xy^iz \in L$ for each $i \geq 0$.

Proof Let

$$w = a_1a_2 \ldots a_m, \qquad m \geq n$$

$$\delta(q_0, a_1a_2 \ldots a_i) = q_i \quad \text{for } i = 1, 2, \ldots, m; \qquad Q_1 = \{q_0, q_1, \ldots, q_m\}$$

That is, Q_1 is the sequence of states in the path with path value $w = a_1a_2 \ldots a_m$. As there are only n distinct states, at least two states in Q_1 must coincide. Among the various pairs of repeated states, we take the first pair. Let us take them as q_j and $q_k (q_j = q_k)$. Then j and k satisfy the condition $0 \leq j < k \leq n$.

The string w can be decomposed into three substrings $a_1a_2 \ldots a_j$, $a_{j+1} \ldots a_k$ and $a_{k+1} \ldots a_m$. Let x, y, z denote these strings $a_1a_2 \ldots a_j$, $a_{j+1} \ldots a_k$, $a_{k+1} \ldots a_m$, respectively. As $k \leq n$, $|xy| \leq n$ and $w = xyz$. The path with the path value w in the transition diagram of M is shown in Fig. 5.27.

The automaton M starts from the initial state q_0. On applying the string x, it reaches $q_j (= q_k)$. On applying the string y, it comes back to $q_j (= q_k)$. So after application of y^i for each $i \geq 0$, the automaton is in the same state q_j. On applying z, it reaches q_m, a final state. Hence, $xy^iz \in L$. As every state in Q_1 is obtained by applying an input symbol, $y \neq \Lambda$. ∎

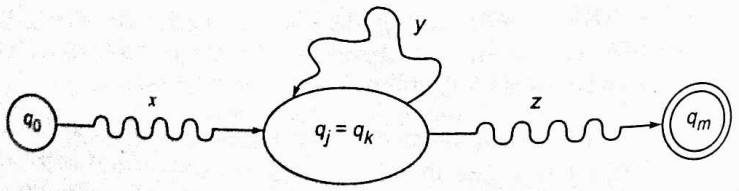

Fig. 5.27 String accepted by M.

Note: The decomposition is valid only for strings of length greater than or equal to the number of states. For such a string $w = xyz$, we can 'iterate' the substring y in xyz as many times as we like and get strings of the form xy^iz which are longer than xyz and are in L. By considering the path from q_0 to q_k and then the path from q_k to q_m (without going through the loop), we get a path ending in a final state with path value xz. (This corresponds to the case when $i = 0$.)

5.4 APPLICATION OF PUMPING LEMMA

This theorem can be used to prove that certain sets are not regular. We now give the steps needed for proving that a given set is not regular.

Step 1 Assume that L is regular. Let n be the number of states in the corresponding finite automaton.

Step 2 Choose a string w such that $|w| \geq n$. Use pumping lemma to write $w = xyz$, with $|xy| \leq n$ and $|y| > 0$.

Step 3 Find a suitable integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.

Note: The crucial part of the procedure is to find i such that $xy^iz \notin L$. In some cases we prove $xy^iz \notin L$ by considering $|xy^iz|$. In some cases we may have to use the 'structure' of strings in L.

EXAMPLE 5.18

Show that the set $L = \{a^{i^2} \mid i \geq 1\}$ is not regular.

Solution

Step 1 Suppose L is regular. Let n be the number of states in the finite automaton accepting L.

Step 2 Let $w = a^{n^2}$. Then $|w| = n^2 > n$. By pumping lemma, we can write $w = xyz$ with $|xy| \leq n$ and $|y| > 0$.

Step 3 Consider xy^2z. $|xy^2z| = |x| + 2|y| + |z| > |x| + |y| + |z|$ as $|y| > 0$. This means $n^2 = |xyz| = |x| + |y| + |z| < |xy^2z|$. As $|xy| \leq n$, we have $|y| \leq n$. Therefore,

$$|xy^2z| = |x| + 2|y| + |z| \leq n^2 + n$$

i.e.

$$n^2 < |xy^2z| \leq n^2 + n < n^2 + n + n + 1$$

Hence, $|xy^2z|$ strictly lies between n^2 and $(n + 1)^2$, but is not equal to any one of them. Thus $|xy^2z|$ is not a perfect square and so $xy^2z \notin L$. But by pumping lemma, $xy^2z \in L$. This is a contradiction.

EXAMPLE 5.19

Show that $L = \{a^p \mid p \text{ is a prime}\}$ is not regular.

Solution

Step 1 We suppose L is regular. Let n be the number of states in the finite automaton accepting L.

Step 2 Let p be a prime number greater than n. Let $w = a^p$. By pumping lemma, w can be written as $w = xyz$, with $|xy| \leq n$ and $|y| > 0$. x, y, z are simply strings of a's. So, $y = a^m$ for some $m \geq 1$ (and $\leq n$).

Step 3 Let $i = p + 1$. Then $|xy^iz| = |xyz| + |y^{i-1}| = p + (i-1)m = p + pm$. By pumping lemma, $xy^iz \in L$. But $|xy^iz| = p + pm = p(1 + m)$, and $p(1 + m)$ is not a prime. So $xy^iz \notin L$. This is a contradiction. Thus L is not regular.

EXAMPLE 5.20

Show that $L = \{0^i1^i \mid i \geq 1\}$ is not regular.

Solution

Step 1 Suppose L is regular. Let n be the number of states in the finite automaton accepting L.

Step 2 Let $w = 0^n1^n$. Then $|w| = 2n > n$. By pumping lemma, we write $w = xyz$ with $|xy| \leq n$ and $|y| \neq 0$.

Step 3 We want to find i so that $xy^iz \notin L$ for getting a contradiction. The string y can be in any of the following forms:

Case 1 y has 0's, i.e. $y = 0^k$ for some $k \geq 1$.

Case 2 y has only 1's, i.e. $y = 1^l$ for some $l \geq 1$.

Case 3 y has both 0's and 1's, i.e. $y = 0^k1^j$ for some $k, j \geq 1$.

In Case 1, we can take $i = 0$. As $xyz = 0^n1^n$, $xz = 0^{n-k}1^n$. As $k \geq 1$, $n - k \neq n$. So, $xz \notin L$.

In Case 2, take $i = 0$. As before, xz is 0^n1^{n-l} and $n \neq n - l$. So, $xz \notin L$.

In Case 3, take $i = 2$. As $xyz = 0^{n-k}0^k1^j1^{n-j}$, $xy^2z = 0^{n-k}\,0^k1^j0^k1^j1^{n-j}$. As xy^2z is not of the form 0^i1^i, $xy^2z \notin L$.

Thus in all the cases we get a contradiction. Therefore, L is not regular.

EXAMPLE 5.21

Show that $L = \{ww \mid w \in \{a, b\}^*\}$ is not regular.

Solution

Step 1 Suppose L is regular. Let n be the number of states in the automaton M accepting L.

Step 2 Let us consider $ww = a^nba^nb$ in L. $|ww| = 2(n + 1) > n$. We can apply pumping lemma to write $ww = xyz$ with $|y| \neq 0$, $|xy| \leq n$.

Step 3 We want to find i so that $xy^iz \notin L$ for getting a contradiction. The string y can be in only one of the following forms:

Case 1 y has no b's, i.e. $y = a^k$ for some $k \geq 1$.

Case 2 y has only one b.

We may note that y cannot have two b's. If so, $|y| \geq n + 2$. But $|y| \leq |xy| \leq n$. In Case 1, we can take $i = 0$. Then $xy^0z = xz$ is of the form a^mba^nb, where $m = n - k < n$ (or a^nba^mb). We cannot write xz in the form uu with $u \in \{a, b\}^*$, and so $xz \notin L$. In Case 2 too, we can take $i = 0$. Then $xy^0z = xz$ has only one b (as one b is removed from xyz, b being in y). So $xz \notin L$ as any element in L should have an even number of a's and an even number of b's.

Thus in both the cases we get a contradiction. Therefore, L is not regular.

Note: If a set L of strings over Σ is given and if we have to test whether L is regular or not, we try to write a regular expression representing L using the definition of L. If this is not possible, we use pumping lemma to prove that L is not regular.

EXAMPLE 5.22

Is $L = \{a^{2n} \mid n \geq 1\}$ regular?

Solution

We can write a^{2n} as $a(a^2)^ia$, where $i \geq 0$. Now $\{(a^2)^i \mid i \geq 0\}$ is simply $\{a^2\}^*$. So L is represented by the regular expression $\mathbf{a(P)^*a}$, where $\mathbf{P}$ represents $\{a^2\}$. The corresponding finite automaton (using the construction given in Section 5.2.5) is shown in Fig. 5.28.

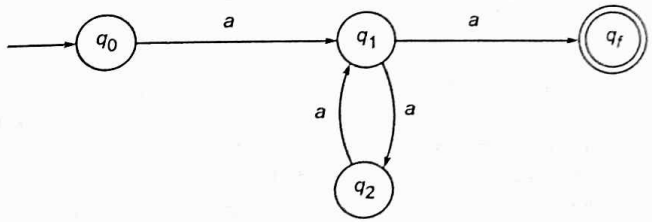

Fig. 5.28 Finite automaton of Example 5.22.

5.5 CLOSURE PROPERTIES OF REGULAR SETS

In this section we discuss the closure properties of regular sets under (i) set union, (ii) concatenation, (iii) closure (iteration), (iv) transpose, (v) set intersection, and (vi) complementation.

In Section 5.1, we have seen that the class of regular sets is closed under union, concatenation and closure.

Theorem 5.6 If L is regular then L^T is also regular.

Proof As L is regular by (vii), given at the end of Section 5.2.7, we can construct a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ such that $T(M) = L$.

We construct a transition system M' by starting with the state diagram of M, and reversing the direction of the directed edges. The set of initial states of M' is defined as the set F, and q_0 is defined as the (only) final state of M', i.e. $M' = (Q, \Sigma, \delta', F, \{q_0\})$.

If $w \in T(M)$, we have a path from q_0 to some final state in F with path value w. By 'reversing the edges', we get a path in M' from some final state in F to q_0. Its path value is w^T. So $w^T \in T(M')$. In a similar way, we can see that if $w_1 \in T(M')$, then $w_1^T \in T(M)$. Thus from the state diagram it is easy to see that $T(M') = T(M)^T$. We can prove rigorously that $w \in T(M)$ iff $w^T \in T(M')$ by induction on $|w|$. So $T(M)^T = T(M')$. By (viii) of Section 5.2.7, $T(M')$ is regular, i.e. $T(M)^T$ is regular. ∎

EXAMPLE 5.23

Consider the FA M given by Fig. 5.29. What is $T(M)$? Show that $T(M)^T$ is regular.

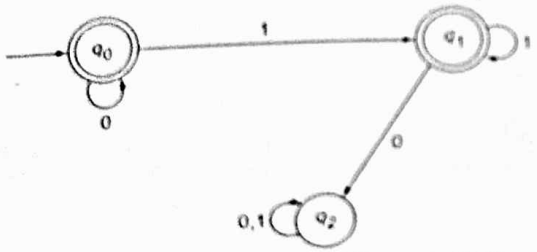

Fig. 5.29 Finite automaton of Example 5.23.

Solution

As the elements of $T(M)$ are given by path values of paths from q_0 to itself or from q_0 to q_1 (note that we have two final states q_0 and q_1), we can construct $T(M)$ by inspection.

As arrows do not come into q_0, the paths from q_0 to itself are self-loops repeated any number of times. The corresponding path values are 0^i, $i \geq 1$. As no arrow comes from q_2 to q_0 or q_1, the paths from q_0 to q_1 are of the form $q_0 \ldots \rightarrow q_0 \ldots q_1 \ldots \rightarrow q_1$. The corresponding path values are 0^i1^j, where $i \geq 0$ and $j \geq 1$. As the initial state q_0 is also a final state, $\Lambda \in T(M)$. Thus,

$$T(M) = \{0^i1^j \mid i, j \geq 0\}$$

Hence,

$$T(M)^T = \{1^j0^i \mid i, j \geq 0\}$$

The transition system M' is constructed as follows:

(i) The initial states of M' are q_0 and q_1.
(ii) The (only) final state of M' is q_0.
(iii) The direction of the directed edges is reversed. M' is given in Fig. 5.30.

From (i)–(iii) it follows that

$$T(M') = T(M)^T$$

Hence, $T(M)^T$ is regular.

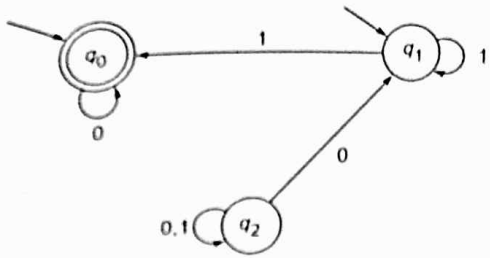

Fig. 5.30 Finite automaton of $T(M)^T$.

Note: In Example 5.23, we can see by inspection that $T(M') = \{1^j\, 0^i \mid i, j \geq 0\}$. The strings of $T(M')$ are obtained as path values of paths from q_0 to itself or from q_1 to q_0.

Theorem 5.7 If L is a regular set over Σ, then $\Sigma^* - L$ is also regular over Σ.

Proof As L is regular by (vii), given at the end of Section 5.2.7, we can construct a DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepting L, i.e. $L = T(M)$.

We construct another DFA $M' = (Q, \Sigma, \delta, q_0, F')$ by defining $F' = Q - F$, i.e. M and M' differ only in their final states. A final state of M' is a nonfinal state of M and vice versa. The state diagrams of M and M' are the same except for the final states.

$w \in T(M')$ if and only if $\delta(q_0, w) \in F' = Q - F$, i.e. iff $w \notin L$. This proves $T(M') = \Sigma^* - X$. ∎

Theorem 5.8 If X and Y are regular sets over Σ, then $X \cap Y$ is also regular over Σ.

Proof By DeMorgan's law for sets, $X \cap Y = \Sigma^* - ((\Sigma^* - X) \cup (\Sigma^* - Y))$. By Theorem 5.7, $\Sigma^* - L$ and $\Sigma^* - Y$ are regular. So, $(\Sigma^* - X) \cup (\Sigma^* - Y)$ is also regular. By applying Theorem 5.7, once again $\Sigma^* - ((\Sigma^* - X) \cup (\Sigma^* - Y))$ is regular, i.e. $X \cap Y$ is regular. ∎

5.6 REGULAR SETS AND REGULAR GRAMMARS

We have seen that regular sets are precisely those accepted by DFA. In this section we show that the class of regular sets over Σ is precisely the regular languages over the terminal set Σ.

5.6.1 Construction of a Regular Grammar Generating $T(M)$ for a Given DFA M

Let $M = (\{q_0, \ldots, q_n\}, \Sigma, \delta, q_0, F)$. If w is in $T(M)$, then it is obtained by concatenating the labels corresponding to several transitions, the first from q_0 and the last terminating at some final state. So for the grammar G to be constructed, productions should correspond to transitions. Also, there should be provision for terminating the derivation once a transition terminating at some final state is encountered. With these ideas in mind, we construct G as

$$G = (\{A_0, A_1, \ldots, A_n\}, \Sigma, P, A_0)$$

where P is defined by the following rules:

(i) $A_i \to aA_j$ is included in P if $\delta(q_i, a) = q_j \notin F$.

(ii) $A_i \to aA_j$ and $A_i \to a$ are included in P if $\delta(q_i, a) = q_j \in F$.

We can show that $L(G) = T(M)$ by using the construction of P. Such a construction gives

$$A_i \Rightarrow aA_j \quad \text{iff } \delta(q_i, a) = q_j$$
$$A_i \Rightarrow a \quad \text{iff } \delta(q_i, a) \in F$$

So,

$$A_0 \Rightarrow a_1A_1 \Rightarrow a_1a_2A_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_{k-1}A_k \Rightarrow a_1a_2 \ldots a_k$$

iff $\delta(q_0, a_1) = q_1, \quad \delta(q_1, a_2) = q_2, \ldots, \quad \delta(q_k, a_k) \in F$

This proves that $w = a_1 \ldots a_k \in L(G)$ iff $\delta(q_0, a_1 \ldots a_k) \in F$, i.e. iff $w \in T(M)$.

EXAMPLE 5.24

Construct a regular grammar G generating the regular set represented by $\mathbf{P = a^*b(a + b)^*}$.

Solution

We construct the DFA corresponding to **P** using the construction given in Section 5.2.5. The construction is shown in Fig. 5.31.

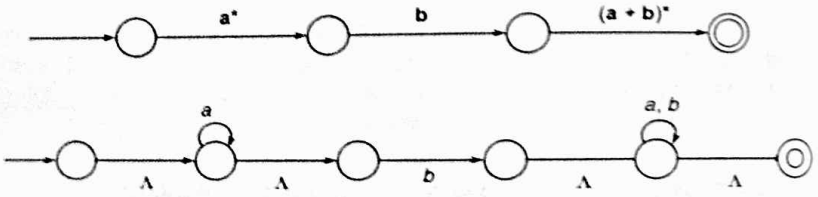

Fig. 5.31 DFA of Example 5.24, with Λ-moves.

After eliminating the Λ-moves, we get the DFA straightaway, as shown in Fig. 5.32.

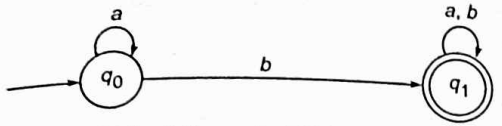

Fig. 5.32 DFA of Example 5.24, without Λ-moves.

Let $G = (\{A_0, A_1\}, \{a, b\}, P, A_0)$, where P is given by

$$A_0 \to aA_0, \quad A_0 \to bA_1, \quad A_0 \to b$$
$$A_1 \to aA_1, \quad A_1 \to bA_1, \quad A_1 \to a, \quad A_1 \to b$$

G is the required regular grammar.

5.6.2 Construction of a Transition System M Accepting $L(G)$ for a Given Regular Grammar G

Let $G = (\{A_0, A_1, \ldots, A_n\}, \Sigma, P, A_0)$. We construct a transition system M whose (i) states correspond to variables, (ii) initial state corresponds to A_0, and (iii) transitions in M correspond to productions in P. As the last production applied in any derivation is of the form $A_i \to a$, the corresponding transition terminates at a new state, and this is the unique final state.

We define M as $(\{q_0, \ldots, q_n, q_f\}, \Sigma, \delta, q_0, \{q_f\})$ where δ is defined as follows:

(i) Each production $A_i \to aA_j$ induces a transition from q_i to q_j with label a.

(ii) Each production $A_k \to a$ induces a transition from q_k to q_f with label a.

From the construction it is easy to see that $A_0 \Rightarrow a_1A_1 \Rightarrow a_1a_2A_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_{n-1}A_{n-1} \Rightarrow a_1 \ldots a_n$ is a derivation of $a_1a_2 \ldots a_n$ iff there is a path in M starting from q_0 and terminating in q_f with path value $a_1a_2 \ldots a_n$. Therefore, $L(G) = T(M)$.

EXAMPLE 5.25

Let $G = (\{A_0, A_1\}, \{a, b\}, P, A_0)$, where P consists of $A_0 \to aA_1$, $A_1 \to bA_1$, $A_1 \to a$, $A_1 \to bA_0$. Construct a transition system M accepting $L(G)$.

Solution

Let $M = (\{q_0, q_1, q_f\}, \{a, b\}, \delta, q_0, \{q_f\})$, where q_0 and q_1 correspond to A_0 and A_1, respectively and q_f is the new (final) state introduced. $A_0 \to aA_1$ induces a transition from q_0 to q_1 with label a. Similarly, $A_1 \to bA_1$ and $A_1 \to bA_0$ induce transitions from q_1 to q_1 with label b and from q_1 to q_0 with

label b, respectively. $A_1 \to a$ induces a transition from q_1 to q_f with label a. M is given in Fig. 5.33.

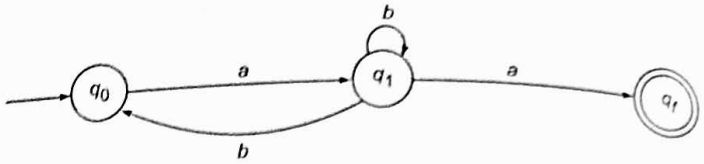

Fig. 5.33 Transition system for Example 5.25.

EXAMPLE 5.26

If a regular grammar G is given by $S \to aS \mid a$, find M accepting $L(G)$.

Solution

Let q_0 correspond to S and q_f be the new (final) state. M is given in Fig. 5.34. Symbolically,

$$M = (\{q_0, q_f\}, \{a\}, \delta, q_0, \{q_f\})$$

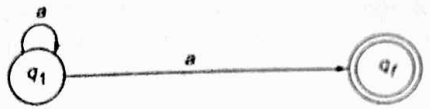

Fig. 5.34 Transition system for Example 5.26.

Note: If $S \to \Lambda$ is in P, the corresponding transition is from q_0 to q_f with label Λ.

By using the construction given in Section 5.2.3, we can construct a DFA M accepting $L(G)$ for a given regular grammar G.

5.7 SUPPLEMENTARY EXAMPLES

EXAMPLE 5.27

Find a regular expression corresponding to each of the following subsets of $\{a, b\}$.

(a) The set of all strings containing exactly $2a$'s.
(b) The set of all strings containing at least $2a$'s.
(c) The set of all strings containing at most $2a$'s.
(d) The set of all strings containing the substring aa.

Solution

(a) b*ab*ab*
(b) (a + b)*a(a + b)*a(a + b)*
(c) b*ab*ab* + b*ab*
(d) (a + b)*aa(a + b)*

EXAMPLE 5.28

Find a regular expression consisting of all strings over $\{a, b\}$ starting with any number of a's, followed by one or more b's, followed by one or more a's, followed by a single b, followed by any number of a's, followed by b and ending in any string of a's and b's.

Solution

The r.e. is a*b b*a a*b(a + b)*.

EXAMPLE 5.29

Find the regular expression representing the set of all strings of the form

(a) $a^m b^n c^p$ where $m, n, p \geq 1$
(b) $a^m b^{2n} c^{3p}$ where $m, n, p \geq 1$
(c) $a^n b a^{2m} b^2$ where $m \geq 0, n \geq 1$

Solution

(a) aa*bb*cc*
(b) aa*(bb)(bb)*ccc(ccc)*
(c) aa*b(aa)*bb

EXAMPLE 5.30

Find the sets represented by the following regular expressions.

(a) (a + b)*(aa + bb + ab + ba)*
(b) (aa)* + (aaa)*
(c) (1 + 01 + 001)*(Λ + 0 + 00)
(d) a + b(a + b)*

Solution

(a) The set of all strings having an odd number of symbols from $\{a, b\}^*$
(b) $\{x \in \{a\}^* \mid |x| \text{ is divisible by 2 or 3}\}$
(c) The set of all strings over $\{0, 1\}$ having no substring of more than two adjacent 0's.
(d) $\{a, b, ba, bb, baa, bab, bba, bbb, \ldots\}$

EXAMPLE 5.31

Show that $\{w \in \{a, b\}^* \mid w$ contains an equal number of a's and b's$\}$ is not regular.

Solution

We prove this by contradiction. Assume that $L = T(M)$ for some DFA M with n states. Let $w = a^n b^n \in L$ and $|w| = 2^n$. Using the pumping lemma, we write $w = xyz$ with $|xy| \leq n$ and $|y| > 0$. As $xyz = a^n b^n$, $xy = a^i$ where $i \leq n$ and hence $y = a^j$ for some j, $1 \leq j \leq n$. Consider xy^2z. Now xyz has an equal number of a's and b's. But xy^2z has $(n + j)$ a's and n b's. As $n + j \neq n$, $xy^2z \notin L$. This contradiction proves that L is not regular.

EXAMPLE 5.32

Show that $L = \{a^i b^j c^k \mid k > i + j\}$ is not regular.

Solution

We prove this by contradiction. Assume $L = T(M)$ for some DFA with n states. Choose $w = a^n b^n c^{3n}$ in L. Using the pumping lemma, we write $w = xyz$ with $|xy| \leq n$ and $|y| > 0$. As $w = a^n b^n c^{3n}$, $xy = a^i$ for some $i \leq n$. This means that $y = a^j$ for some j, $1 \leq j \leq n$. Then $xy^{k+1}z = a^{n+jk} b^n c^{3n}$. Choosing k large enough so that $n + jk > 2n$, we can make $n + jk + n > 3n$. So, $xy^{k+1}z \notin L$. Hence L is not regular.

EXAMPLE 5.33

Prove the identities I_5, I_6, I_7, I_8, I_{11}, I_{12} given in Section 5.1.1.

Proof $L(\mathbf{R} + \mathbf{R}) = L(\mathbf{R}) \cup L(\mathbf{R}) = L(\mathbf{R})$. Hence I_5.

$L(\mathbf{R^*R^*}) = L(\mathbf{R^*})L(\mathbf{R^*}) = \{w_1 w_2 \mid w_1, w_2 \in L(\mathbf{R^*})\}$. But $w_1 = x_1 x_2 \ldots x_n \in L(\mathbf{R})^*$ for $x_i \in L(\mathbf{R})$, $i = 1, 2, \ldots, n$. Similarly, $w_2 = y_1 y_2 \ldots y_m \in L(\mathbf{R^*})$ for $y_j \in L(\mathbf{R})$, $j = 1, 2, \ldots, m$. So $w_1 w_2 \in L(\mathbf{R})^*$, proving I_6.

An element of $L(\mathbf{RR^*})$ is of the form $xy_1 \ldots y_n$ for some $x, y_1, \ldots, y_n \in L(\mathbf{R})$.

As $xy_1 \ldots y_n = (xy_1 \ldots y_{n-1})y_n \in L(\mathbf{R^*R})$, I_7 follows.

It is easy to see that $L(\mathbf{R^*}) \subseteq L((\mathbf{R^*})^*)$. Take $w \in L((\mathbf{R^*})^*)$. Then $w = x_1 \ldots x_m$ where $x_i \in \mathbf{R^*}$, $i = 1, 2, \ldots, m$. Each x_i in turn can be written in the form $y_1 y_2 \ldots y_n$ for some $y_i \in L(\mathbf{R})$, $i = 1, 2, \ldots, n$. So, $w = x_1 \ldots x_m = z_1 \ldots z_k \in L(\mathbf{R^*})$.

(***Note:*** $x_1 = z_1 \ldots z_t$, $x_2 = z_{t+1} \ldots z_{t+l}$ etc.)

Hence I_8.

To prove I_{11}, take $w \in L(\mathbf{P} + \mathbf{Q})^*$. Then $w = w_1 \ldots w_n$ where $w_i \in L(\mathbf{P}) \cup L(\mathbf{Q})$. By writing $w_i = \Lambda w_i = w_i \Lambda$, we note that $w_i \in \mathbf{P^*Q^*}$. Hence $w \in L(\mathbf{P^*Q^*})^*$. To prove $L(\mathbf{P^*Q^*})^* \subseteq L(\mathbf{P} + \mathbf{Q})^*$, take $w \in L(\mathbf{P^*Q^*})^*$.

Then, $w = w_1 \ldots w_n$ where $w_i \in \mathbf{P^*Q^*}$. For simplicity take $n = 2$. (The result can be extended by induction for any n). Then $w_1 = x_1 x_2 \ldots x_k$, where $x_i \in L(\mathbf{P})$ and $w_2 = y_1 y_2 \ldots y_l$ where $y_j \in L(\mathbf{Q})$. So $w = x_1 x_2 \ldots x_k y_1 y_2 \ldots y_l$. Each x_i or y_j is in $L(\mathbf{P} + \mathbf{Q})$. Hence $w \in L(\mathbf{P} + \mathbf{Q})^*$, proving the first identity in I_{11}. The second identity can be proved in a similar way.

Finally, we prove I_{12}.

$$\begin{aligned} L((\mathbf{P} + \mathbf{Q})\mathbf{R}) &= L(\mathbf{P} + \mathbf{Q})L(\mathbf{R}) \\ &= (L(\mathbf{P}) \cup L(\mathbf{Q}))L(\mathbf{R}) \\ L(\mathbf{PR} + \mathbf{QR}) &= L(\mathbf{PR}) \cup L(\mathbf{QR}) \\ &= (L(\mathbf{P})L(\mathbf{R})) \cup (L(\mathbf{Q})L(\mathbf{R})) \end{aligned}$$

But $(A \cup B)C = AC \cup BC$ for $A, B, C \subseteq \Sigma^*$. For, a string w in $(A \cup B)C$ is the concatenation of a string w_1 in A or B and a string w_2 in C. If $w_1 \in A$, then $w_1 w_2 \in AC$; if $w_1 \in B$, then $w_1 w_2 \in BC$. Hence $w \in AC \cup BC$. The other inclusion can be proved similarly. I_{12} follows from $(A \cup B)C = AC \cup BC$.

EXAMPLE 5.34

Prove that $\mathbf{P + PQ^*Q = a^*bQ^*}$ where $\mathbf{P = b + aa^*b}$ and $\mathbf{Q}$ is any regular expression.

Proof

$$\begin{aligned} \text{L.H.S.} &= \mathbf{P\Lambda + PQ^*Q} && \text{by } I_3 \\ &= \mathbf{P(\Lambda + Q^*Q)} && \text{by } I_{12} \\ &= \mathbf{PQ^*} && \text{by } I_9 \\ &= \mathbf{(b + aa^*b)Q^*} && \text{by definition of P} \\ &= \mathbf{(\Lambda b + aa^*b)Q^*} && \text{by } I_3 \\ &= \mathbf{(\Lambda + aa^*)bQ^*} && \text{by } I_{12} \\ &= \mathbf{a^*bQ^*} && \text{by } I_9 \\ &= \text{R.H.S.} \end{aligned}$$

EXAMPLE 5.35

Construct a regular grammar accepting $L = \{w \in \{a, b\}^* \mid w$ is a string over $\{a, b\}$ such that the number of b's is 3 mod 4$\}$.

Solution

We construct a DFA M accepting L directly. The symbol a can occur in any place in w and b has to occur in $4k + 3$ places, where $k \geq 0$. So we can have states q_i, $i = 0, 1, 2, 3$, for remembering that the string processed so far has $4k$, $4k + 1$, $4k + 2$ and $4k + 3$ b's ($k \geq 0$). q_3 is the only final state. Also M does not change state on reading a's. The state diagram representing M is given in Fig. 5.35.

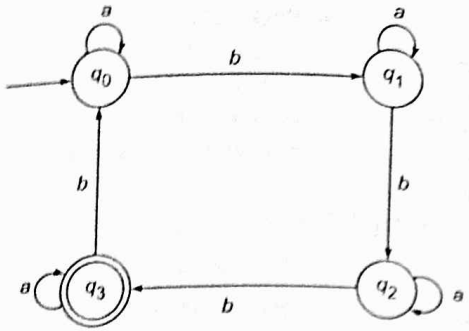

Fig. 5.35 DFA for Example 5.35.

By applying the construction given in Section 5.6.1, we can construct a regular grammar G accepting $L = T(M)$.

$G = (\{A_0, A_1, A_2, A_3\}, \{a, b\}, P, A_0)$ where P consists of $A_0 \to aA_0$, $A_0 \to bA_1$, $A_1 \to bA_1$, $A_1 \to bA_2$, $A_2 \to aA_2$, $A_2 \to bA_3$, $A_2 \to b$, $A_3 \to aA_2$, $A_3 \to aA_0$.

EXAMPLE 5.36

Let $G = (\{A_0, A_1, A_2, A_3\}, \{a, b\}, P, A_0)$, where P consists of $A_0 \to aA_0 \mid bA_1$, $A_1 \to aA_2 \mid aA_3$, $A_3 \to a \mid bA_1 \mid bA_3$, $A_3 \to b \mid bA_0$. Construct an NDFA accepting $L(G)$.

Solution

The NDFA accepting $L = L(G)$ is M where $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \delta, q_0, \{q_4\})$. δ is described by the state diagram shown in Fig. 5.36.

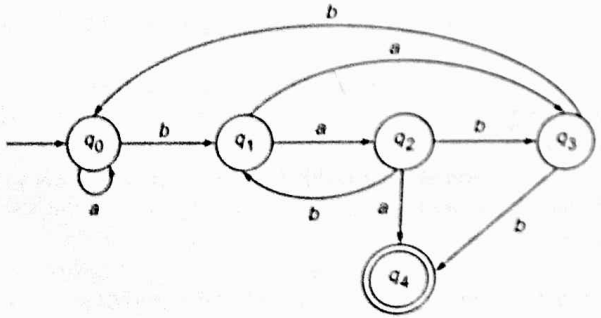

Fig. 5.36 NDFA for Example 5.36.

SELF-TEST

Choose the correct answer to Questions 1–10.

1. The set of all strings over $\{a, b\}$ of even length is represented by the regular expression
   (a) (ab + aa + bb + ba)* (b) (a + b)*(a* + b)*
   (c) (aa + bb)* (d) (ab + ba)*
2. The set of all strings over $\{a, b\}$ of length 4, starting with an a is represented by the regular expression
   (a) a(a + b)* (b) a(ab)*
   (c) (ab + ba)(aa + bb) (d) a(a + b)(a + b)(a + b)
3. (0*1*)* is the same as
   (a) (0 + 1)* (b) (01)*
   (c) (10)* (d) none of these.
4. If L is the set of all strings over $\{a, b\}$ containing at least one a, then it is not represented by the regular expression
   (a) b*a(a + b)* (b) (a + b)*a(b + a)*
   (c) (a + b)*ab* (d) (a + b)*a
5. $\{a^{2n} \mid n \geq 1\}$ is represented by the regular expression
   (a) (aa)* (b) a*
   (c) aa*a (d) a*a*
6. The set of strings over $\{a, b\}$ having exactly $3b$'s is represented by the regular expression
   (a) a*bbb (b) a*ba*ba*b
   (c) ba*ba*b (d) a*ba*ba*ba*
7. The set of all strings over $\{a, b\}$ having $abab$ as a substring is represented by
   (a) a*ababb* (b) (a + b)*abab(a + b)*
   (c) a*b*ababa*b* (d) (a + b)*abab
8. (a + a*)* is equivalent to
   (a) a(a*)* (b) a*
   (c) aa* (d) none of these.
9. a*(a + b)* is equivalent to
   (a) a* + b* (b) (ab)*
   (c) a*b* (d) none of these.
10. ab* + b* represents all strings w over $\{a, b\}$
    (a) starting with an a and having no other a's or having no a's but only b's
    (b) starting with an a followed by b's
    (c) having no a's but only b's
    (d) none of these.

State whether the following Statements 11–17 are true or false.

11. If Σ is finite then Σ^* is finite.

12. Every finite subset of Σ^* is a regular language.

13. Every regular language over Σ is finite.

14. a^4b^3 is in the regular set given by $\mathbf{a^*(a + b)b^*}$.

15. $\mathbf{aa^* + bb^*}$ is the same as $\mathbf{(a + b)^*}$.

16. The set of all strings starting with an a and ending in ab is defined by the regular expression $\mathbf{a(a + b)^*b}$.

17. The regular expression $\mathbf{(a + b)^*c^*}$ is the same as $\mathbf{a^*(b + c)^*}$.

EXERCISES

5.1 Represent the following sets by regular expressions:

(a) $\{0, 1, 2\}$.

(b) $\{1^{2n+1} \mid n > 0\}$.

(c) $\{w \in \{a, b\}^* \mid w$ has only one $a\}$.

(d) The set of all strings over $\{0, 1\}$ which has at most two zeros.

(e) $\{a^2, a^5, a^8, \ldots\}$.

(f) $\{a^n \mid n$ is divisible by 2 or 3 or $n = 5\}$.

(g) The set of all strings over $\{a, b\}$ beginning and ending with a.

5.2 Find all strings of length 5 or less in the regular set represented by the following regular expressions:

(a) $\mathbf{(ab + a)^*(aa + b)}$

(b) $\mathbf{(a^*b + b^*a)^*a}$

(c) $\mathbf{a^* + (ab + a)^*}$

5.3 Describe, in the English language, the sets represented by the following regular expressions:

(a) $\mathbf{a(a + b)^*ab}$

(b) $\mathbf{a^*b + b^*a}$

(c) $\mathbf{(aa + b)^*(bb + a)^*}$

5.4 Prove the following identity:

$$\mathbf{(a^*ab + ba)^*a^* = (a + ab + ba)^*}$$

5.5 Construct the transition systems equivalent to the regular expressions given in Exercise 5.2.

5.6 Construct the transition systems equivalent to the regular expressions given in Exercise 5.3.

5.7 Find the set of strings over $\Sigma = \{a, b\}$ recognized by the transition systems shown in Fig. 5.37(a–d).

5.8 Find the regular expression corresponding to the automaton given in Fig. 5.38.

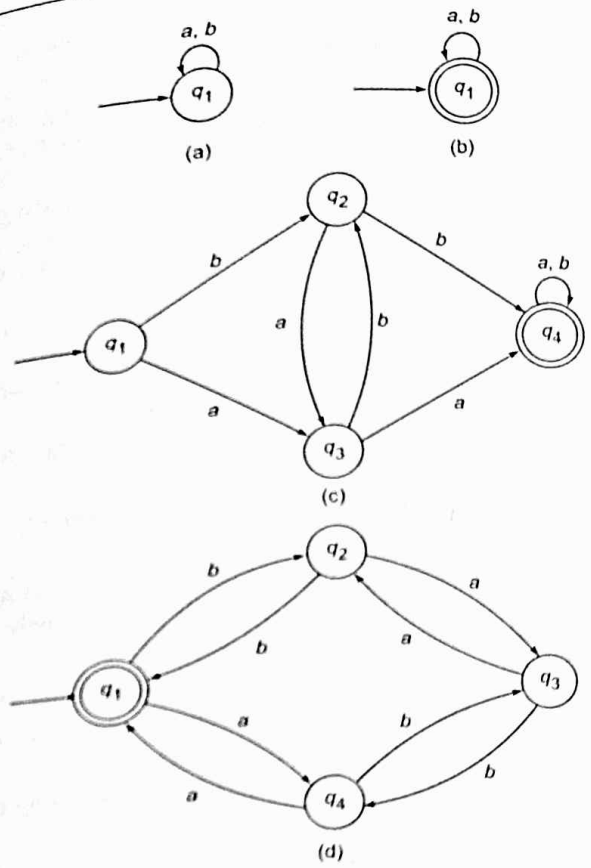

Fig. 5.37 Transition systems of Exercise 5.7.

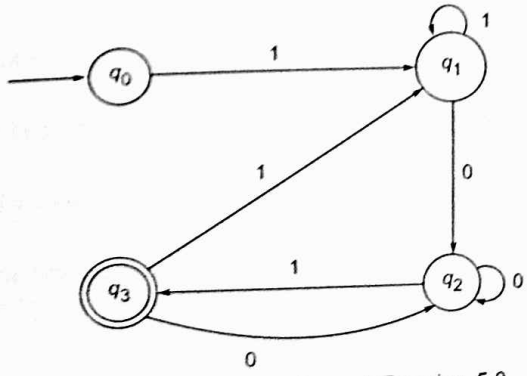

Fig. 5.38 Transition system of Exercise 5.8.

5.9 Construct a transition system corresponding to the regular expressions (i) $(\mathbf{ab} + \mathbf{c}^*)^*\mathbf{b}$ and (ii) $\mathbf{a} + \mathbf{bb} + \mathbf{bab}^*\mathbf{a}$.

5.10 Find the regular expressions representing the following sets:
(a) The set of all strings over $\{0, 1\}$ having at most one pair of 0's or at most one pair of 1's.
(b) The set of all strings over $\{a, b\}$ in which the number of occurrences of a is divisible by 3.
(c) The set of all strings over $\{a, b\}$ in which there are at least two occurrences of b between any two occurrences of a.
(d) The set of all strings over $\{a, b\}$ with three consecutive b's.
(e) The set of all strings over $\{0, 1\}$ beginning with 00.
(f) The set of all strings over $\{0, 1\}$ ending with 00 and beginning with 1.

5.11 Construct a deterministic finite automaton corresponding to the regular expression given in Exercise 5.2.

5.12 Construct a finite automaton accepting all strings over $\{0, 1\}$ ending in 010 or 0010.

5.13 Construct a finite automaton M which can recognize DFA in a given string over the alphabet $\{A, B, \ldots, Z\}$. For example, M has to recognize DFA in the string ATXDFAMN.

5.14 Construct a finite automaton for the regular expression $(\mathbf{a} + \mathbf{b})^*\mathbf{abb}$.

5.15 Show that there exists no finite automaton accepting all palindromes over $\{a, b\}$.

5.16 Show that $\{a^n b^n \mid n > 0\}$ is not a regular set without using the pumping lemma.

5.17 Using the pumping lemma, show that the following sets are not regular:
(a) $\{a^n b^{2n} \mid n > 0\}$;
(b) $\{a^n b^m \mid 0 < n < m\}$.

5.18 Show that $\{0^n 1^n \mid \text{g.c.d. } (m, n) = 1\}$ is not regular.

5.19 Show that a deterministic finite automaton with n states accepting a nonempty set accepts a string of length m, $m < n$.

5.20 Construct a finite automaton recognizing $L(G)$, where G is the grammar $S \to aS \mid bA \mid b$ and $A \to aA \mid bS \mid a$.

5.21 Find a regular grammar accepting the set recognized by the finite automaton given in Fig. 5.37(c).

5.22 Construct a regular grammar which can generate the set of all strings starting with a letter (A to Z) followed by a string of letters or digits (0 to 9).

5.23 Are the following true or false? Support your answer by giving proofs or counter-examples.
(a) If $L_1 \cup L_2$ is regular and L_1 is regular, then L_2 is regular.
(b) If $L_1 L_2$ is regular and L_1 is regular, then L_2 is regular.
(c) If L^* is regular, then L is regular.

5.24 Construct a deterministic finite automaton equivalent to the grammar $S \to aS \mid bS \mid aA$, $A \to bB$, $B \to aC$, $C \to \Lambda$.

6 Context-Free Languages

In this chapter we study context-free grammars and languages. We define derivation trees and give methods of simplifying context-free grammars. The two normal forms—Chomsky normal form and Greibach normal form—are dealt with. We conclude this chapter after proving pumping lemma and giving some decision algorithms.

6.1 CONTEXT-FREE LANGUAGES AND DERIVATION TREES

Context-free languages are applied in parser design. They are also useful for describing block structures in programming languages. It is easy to visualize derivations in context-free languages as we can represent derivations using tree structures.

Let us recall the definition of a context-free grammar (CFG). G is context-free if every production is of the form $A \to \alpha$, where $A \in V_N$ and $\alpha \in (V_N \cup \Sigma)^*$.

EXAMPLE 6.1

Construct a context-free grammar G generating all integers (with sign).

Solution

Let

$$G = (V_N, \Sigma, P, S)$$

where

$$V_N = \{S, \langle\text{sign}\rangle, \langle\text{digit}\rangle, \langle\text{Integer}\rangle\}$$
$$\Sigma = \{0, 1, 2, 3, \ldots, 9, +, -\}$$

P consists of $S \to \langle\text{sign}\rangle\ \langle\text{integer}\rangle$, $\langle\text{sign}\rangle \to + \mid -$,

$$\langle\text{integer}\rangle \to \langle\text{digit}\rangle\ \langle\text{integer}\rangle \mid \langle\text{digit}\rangle$$
$$\langle\text{digit}\rangle \to 0 \mid 1 \mid 2 \mid \ldots \mid 9$$

$L(G)$ = the set of all integers. For example, the derivation of -17 can be obtained as follows:

$$S \Rightarrow \langle\text{sign}\rangle\ \langle\text{integer}\rangle \Rightarrow -\ \langle\text{integer}\rangle$$
$$\Rightarrow -\ \langle\text{digit}\rangle\ \langle\text{integer}\rangle \Rightarrow -\ 1\ \langle\text{integer}\rangle \Rightarrow -1\ \langle\text{digit}\rangle$$
$$\Rightarrow -\ 17$$

6.1.1 DERIVATION TREES

The derivations in a CFG can be represented using trees. Such trees representing derivations are called derivation trees. We give below a rigorous definition of a derivation tree.

Definition 6.1 A derivation tree (also called a parse tree) for a CFG $G = (V_N, \Sigma, P, S)$ is a tree satisfying the following conditions:

(i) Every vertex has a label which is a variable or terminal or Λ.
(ii) The root has label S.
(iii) The label of an internal vertex is a variable.
(iv) If the vertices $n_1, n_2, \ldots, n_k$ written with labels $X_1, X_2, \ldots, X_k$ are the sons of vertex n with label A, then $A \to X_1X_2 \ldots X_k$ is a production in P.
(v) A vertex n is a leaf if its label is $a \in \Sigma$ or Λ; n is the only son of its father if its label is Λ.

For example, let $G = (\{S, A\}, \{a, b\}, P, S)$, where P consists of $S \to aAS \mid a \mid SS$, $A \to SbA \mid ba$. Figure 6.1 is an example of a derivation tree.

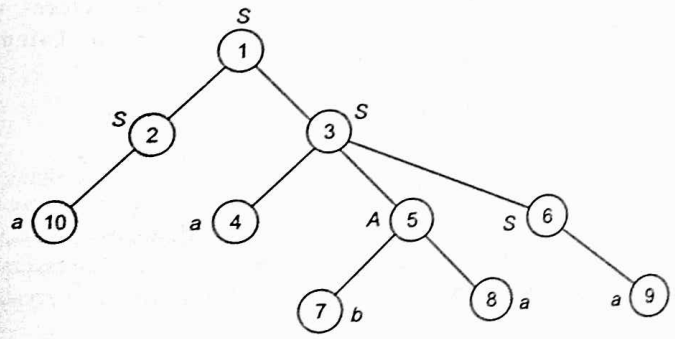

Fig. 6.1 An example of a derivation tree.

Note: Vertices 4-6 are the sons of 3 written from the left, and $S \to aAS$ is in P. Vertices 7 and 8 are the sons of 5 written from the left, and $A \to ba$ is a production in P. The vertex 5 is an internal vertex and its label is A, which is a variable.

Ordering of Leaves from the Left

We can order all the vertices of a tree in the following way: The successors of the root (i.e. sons of the root) are ordered from the left by the definition (refer to Section 1.2). So the vertices at level 1 are ordered from the left. If v_1 and v_2 are any two vertices at level 1 and v_1 is to the left of v_2, then we say that v_1 is to the left of any son of v_2. Also, any son of v_1 is to the left of v_2 and to the left of any son of v_2. Thus we get a left-to-right ordering of vertices at level 2. Repeating the process up to level k, where k is the height of the tree, we have an ordering of all vertices from the left.

Our main interest is in the ordering of leaves.

In Fig. 6.1, for example, the sons of the root are 2 and 3 ordered from the left. So, the son of 2, namely 10, is to the left of any son of 3. The sons of 3 ordered from the left are 4-5-6. The vertices at level 2 in the left-to-right ordering are 10-4-5-6. The vertex 4 is to the left of 6. The sons of 5 ordered from the left are 7-8. So 4 is to the left of 7. Similarly, 8 is to the left of 9. Thus the order of the leaves from the left is 10-4-7-8-9.

Note: If we draw the sons of any vertex keeping in mind the left-to-right ordering, we get the left-to-right ordering of leaves by 'reading' the leaves in the anticlockwise direction.

Definition 6.2 The yield of a derivation tree is the concatenation of the labels of the leaves without repetition in the left-to-right ordering.

The yield of the derivation tree of Fig. 6.1, for example, is *aabaa*.

Note: Consider the derivation tree in Fig. 6.1. As the sons of 1 are 2-3 in the left-to-right ordering, by condition (iv) of Definition 6.1, we have the production $S \to SS$. By applying the condition (iv) to other vertices, we get the productions $S \to a$, $S \to aAS$, $A \to ba$ and $S \to a$. Using these productions, we get the following derivation:

$$S \Rightarrow SS \Rightarrow as \Rightarrow aaAS \Rightarrow aabaS \Rightarrow aabaa$$

Thus the yield of the derivation tree is a sentential form in G.

Definition 6.3 A subtree of a derivation tree T is a tree (i) whose root is some vertex v of T, (ii) whose vertices are the descendants of v together with their labels, and (iii) whose edges are those connecting the descendants of v.

Figures 6.2 and 6.3, for example, give two subtrees of the derivation tree shown in Fig. 6.1.

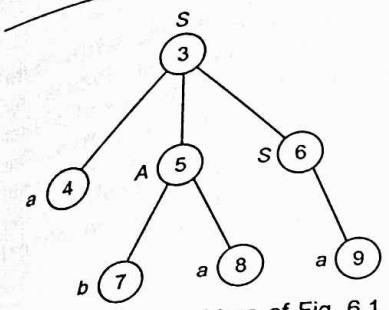

Fig. 6.2 A subtree of Fig. 6.1.

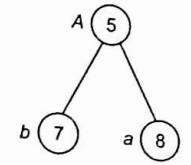

Fig. 6.3 Another subtree of Fig. 6.1.

Note: A subtree looks like a derivation tree except that the label of the root may not be S. It is called an A-tree if the label of its root is A.

Remark When there is no need for numbering the vertices, we represent the vertices by points. The following theorem asserts that sentential forms in CFG G are precisely the yields of derivation trees for G.

Theorem 6.1 Let $G = (V_N, \Sigma, P, S)$ be a CFG. Then $S \overset{*}{\Rightarrow} \alpha$ if and only if there is a derivation tree for G with yield α.

Proof We prove that $A \overset{*}{\Rightarrow} \alpha$ if and only if there is an A-tree with yield α. Once this is proved, the theorem follows by assuming that $A = S$.

Let α be the yield of an A-tree T. We prove that $A \overset{*}{\Rightarrow} \alpha$ by induction on the number of internal vertices in T.

When the tree has only one internal vertex, the remaining vertices are leaves and are the sons of the root. This is illustrated in Fig. 6.4.

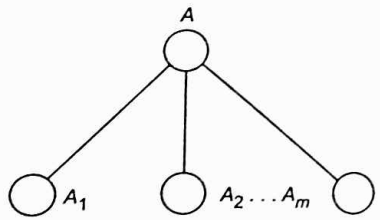

Fig. 6.4 A tree with only one internal vertex.

By condition (iv) of Definition 6.1, $A \to A_1A_2 \ldots A_m = \alpha$ is a production in G, i.e. $A \Rightarrow \alpha$. Thus there is basis for induction. Now assume the result for all trees with at most $k - 1$ internal vertices ($k > 1$).

Let T be an A-tree with k internal vertices ($k \geq 2$). Let $v_1, v_2, \ldots, v_m$ be the sons of the root in the left-to-right ordering. Let their labels be $X_1, X_2, \ldots, X_m$. By condition (iv) of Definition 6.1, $A \to X_1X_2 \ldots X_m$ is in P, and so

$$A \Rightarrow X_1X_2 \ldots X_m \tag{6.1}$$

As $k \geq 2$, at least one of the sons is an internal vertex. By the left-to-right ordering of leaves, α can be written as $\alpha_1\alpha_2 \ldots \alpha_m$, where α_i is obtained by the concatenation of the labels of the leaves which are descendants of vertex v_i. If v_i is an internal vertex, consider the subtree of T with v_i as its root. The number of internal vertices of the subtree is less than k (as there are k internal vertices in T and at least one of them, viz. its root, is not in the subtree). So by induction hypothesis applied to the subtree, $X_i \overset{*}{\Rightarrow} \alpha_i$. If v_i is not an internal vertex, i.e. a leaf, then $X_i = \alpha_i$.

Using (6.1), we get

$$A \Rightarrow X_1X_2 \ldots X_m \overset{*}{\Rightarrow} \alpha_1X_2X_3 \ldots X_m \ldots \overset{*}{\Rightarrow} \alpha_1\alpha_2 \ldots \alpha_m = \alpha,$$

i.e. $A \overset{*}{\Rightarrow} \alpha$. By the principle of induction, $A \overset{*}{\Rightarrow} \alpha$ whenever α is the yield of an A-tree.

To prove the 'only if' part, let us assume that $A \overset{*}{\Rightarrow} \alpha$. We have to construct an A-tree whose yield is α. We do this by induction on the number of steps in $A \overset{*}{\Rightarrow} \alpha$.

When $A \Rightarrow \alpha$, $A \to \alpha$ is a production in P. If $\alpha = X_1X_2 \ldots X_m$, the A-tree with yield α is constructed and given as in Fig. 6.5. So there is basis for induction. Assume the result for derivations in at most k steps. Let $A \overset{k}{\Rightarrow} \alpha$; we can split this as $A \Rightarrow X_1 \ldots X_m \overset{k-1}{\Rightarrow} \alpha$. Now, $A \Rightarrow X_1 \ldots X_m$ implies $A \to X_1X_2 \ldots X_m$ is a production in P. In the derivation $X_1X_2 \ldots X_m \overset{k-1}{\Rightarrow} \alpha$, either (i) X_i is not changed throughout the derivation, or (ii) X_i is changed in some subsequent step. Let α_i be the substring of α derived from X_i. Then $X_i \overset{*}{\Rightarrow} \alpha_i$ in (ii) and $X_i = \alpha_i$ in (i). As G is context-free, in every step of the derivation $X_1X_2 \ldots X_m \overset{*}{\Rightarrow} \alpha$, we replace a single variable by a string. As $\alpha_1, \alpha_2, \ldots, \alpha_m$ account for all the symbols in α, we have $\alpha = \alpha_1\alpha_2 \ldots \alpha_m$.

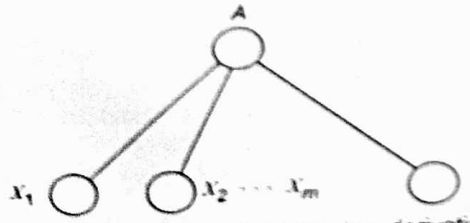

Fig. 6.5 Derivation tree for one-step derivation.

We construct the derivation tree with yield α as follows: As $A \to X_1 \ldots X_m$ is in P, we construct a tree with m leaves whose labels are $X_1, \ldots, X_m$ in the left-to-right ordering. This tree is given in Fig. 6.6. In (i) above, we leave the vertex v_i as it is. In (ii), $X_i \overset{*}{\Rightarrow} \alpha_i$ is less than k steps (as $X_1 \ldots X_m \overset{n-1}{\Rightarrow} a$). By induction hypothesis there exists an X_i-tree T_i with yield α_i. We attach the tree T_i at the vertex v_i (i.e. v_i is the root of T_i). The resulting tree is given in Fig. 6.7. In this figure, let i and j be the first and the last indexes such that X_i and X_j satisfy (ii). So, $\alpha_1 \ldots \alpha_{i-1}$ are the labels of leaves at level 1 in T. α_i is the yield of the X_i-tree T_i, etc.

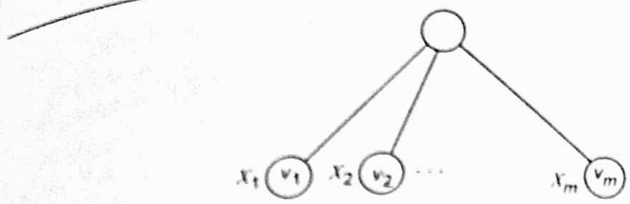

Fig. 6.6 Derivation tree with yield $X_1X_2 \ldots X_m$.

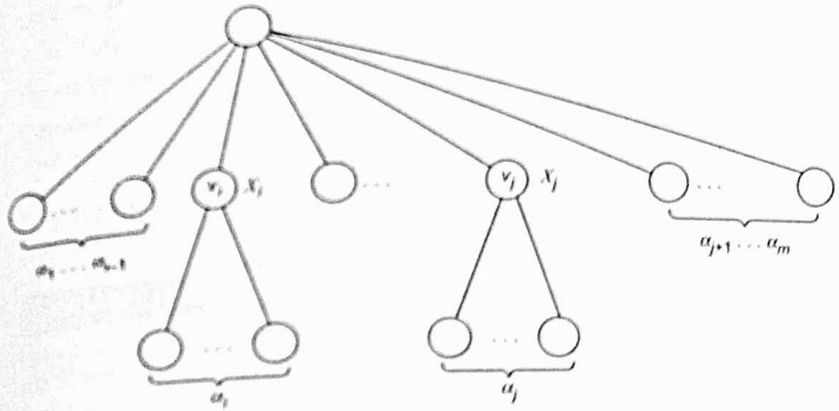

Fig. 6.7 Derivation tree with yield $\alpha_1\alpha_2 \ldots \alpha_n$.

Thus we get a derivation tree with yield α. By the principle of induction we can get the result for any derivation. This completes the proof of 'only if' part. ∎

Note: The derivation tree does not specify the order in which we apply the productions for getting α. So, the same derivation tree can induce several derivations.

The following remark is used quite often in proofs and constructions involving CFGs.

Remark If A derives a terminal string w and if the first step in the derivation is $A \Rightarrow A_1A_2 \ldots A_n$, then we can write w as $w_1w_2 \ldots w_n$ so that $A_i \overset{*}{\Rightarrow} w_i$. (Actually, in the derivation tree for w, the ith son of the root has the label A_i, and w_i is the yield of the subtree whose root is the ith son.)

EXAMPLE 6.2

Consider G whose productions are $S \to aAS \mid a$, $A \to SbA \mid SS \mid ba$. Show that $S \overset{*}{\Rightarrow} aabbaa$ and construct a derivation tree whose yield is $aabbaa$.

Solution

$$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow a^2bbaS \Rightarrow a^2b^2a^2$$

Hence, $S \overset{*}{\Rightarrow} a^2b^2a^2$. The derivation tree is given in Fig. 6.8. (6.2)

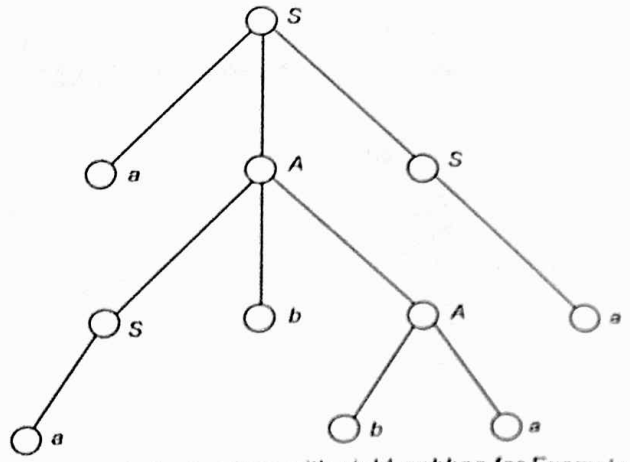

Fig. 6.8 The derivation tree with yield *aabbaa* for Example 6.2

Note: Consider G as given in Example 6.2. We have seen that $S \overset{*}{\Rightarrow} a^2b^2a^2$, and (6.2) gives a derivation of $a^2b^2a^2$.

Another derivation of $a^2b^2a^2$ is

$$S \Rightarrow aAS \Rightarrow aAa \Rightarrow aSbAa \Rightarrow aSbbaa \Rightarrow aabbaa \quad (6.3)$$

Yet another derivation of $a^2b^2a^2$ is

$$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aSbAa \Rightarrow aabAa \Rightarrow aabbaa \quad (6.4)$$

In derivation (6.2), whenever we replace a variable X using a production, there are no variables to the left of X. In derivation (6.3), there are no variables to the right of X. But in (6.4), no such conditions are satisfied. These lead to the following definitions.

Definition 6.4 A derivation $A \overset{*}{\Rightarrow} w$ is called a *leftmost* derivation if we apply a production only to the leftmost variable at every step.

Definition 6.5 A derivation $A \overset{*}{\Rightarrow} w$ is a *rightmost* derivation if we apply production to the rightmost variable at every step.

Relation (6.2), for example, is a leftmost derivation. Relation (6.3) is a rightmost derivation. But (6.4) is neither leftmost nor rightmost. In the second step of (6.4), the rightmost variable S is not replaced. So (6.4) is not a rightmost derivation. In the fourth step, the leftmost variable S is not replaced. So (6.4) is not a leftmost derivation.

Theorem 6.2 If $A \overset{*}{\Rightarrow} w$ in G, then there is a leftmost derivation of w.

Proof We prove the result for every A in V_N by induction on the number of steps in $A \overset{*}{\Rightarrow} w$. $A \Rightarrow w$ is a leftmost derivation as the L.H.S. has only one variable. So there is basis for induction. Let us assume the result for derivations in atmost k steps. Let $A \overset{n+1}{\Rightarrow} w$. The derivation can be split as $A \Rightarrow X_1X_2 \ldots X_m \overset{k}{\Rightarrow} w$.

The string w can be split as $w_1w_2 \ldots w_m$ such that $X_i \Rightarrow w_i$ (see the Remark appended before Example 6.2). As $X_i \overset{*}{\Rightarrow} w_i$ involves atmost k steps by induction hypothesis, we can find a leftmost derivation of w_i. Using these leftmost derivations, we get a leftmost derivation of w given by

$$A \Rightarrow X_1X_2 \ldots X_m \overset{*}{\Rightarrow} w_1X_2 \ldots X_m \overset{*}{\Rightarrow} w_1w_2X_3 \ldots X_m \ldots \overset{*}{\Rightarrow} w_1w_2 \ldots w_m$$

Hence by induction the result is true for all derivations $A \overset{*}{\Rightarrow} w$. ∎

Corollary Every derivation tree of w induces a leftmost derivation of w.

Once we get some derivation of w, it is easy to get a leftmost derivation of w in the following way: From the derivation tree for w, at every level consider the productions for the variables at that level, taken in the left-to-right ordering. The leftmost derivation is obtained by applying the productions in this order.

EXAMPLE 6.3

Let G be the grammar $S \to 0B \mid 1A$, $A \to 0 \mid 0S \mid 1AA$, $B \to 1 \mid 1S \mid 0BB$. For the string 00110101, find (a) the leftmost derivation, (b) the rightmost derivation, and (c) the derivation tree.

Solution

(a) $S \Rightarrow 0B \Rightarrow 00BB \Rightarrow 001B \Rightarrow 0011S$
$\Rightarrow 0^21^20B \Rightarrow 0^21^201S \Rightarrow 0^21^2010B \Rightarrow 0^21^20101$

(b) $S \Rightarrow 0B \Rightarrow 00BB \Rightarrow 00B1S \Rightarrow 00B10B$
$\Rightarrow 0^2B101S \Rightarrow 0^2B1010B \Rightarrow 0^2B10101 \Rightarrow 0^2110101.$

(c) The derivation tree is given in Fig. 6.9.

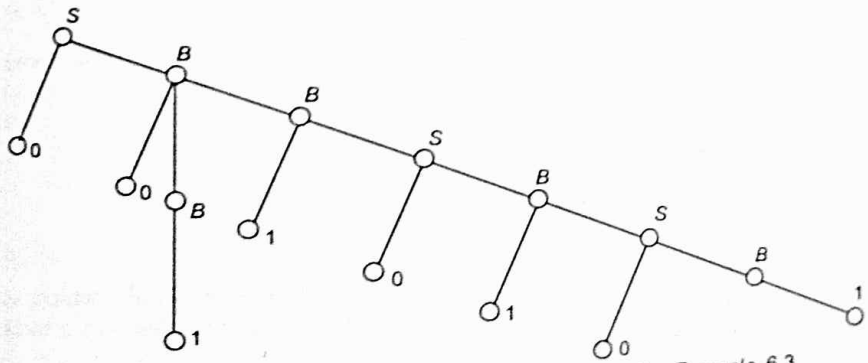

Fig. 6.9 The derivation tree with yield 00110101 for Example 6.3

6.2 AMBIGUITY IN CONTEXT-FREE GRAMMARS

Sometimes we come across ambiguous sentences in the language we are using. Consider the following sentence in English: "In books selected information is given." The word 'selected' may refer to books or information. So the sentence may be parsed in two different ways. The same situation may arise in context-free languages. The same terminal string may be the yield of two derivation trees. So there may be two different leftmost derivations of w by Theorem 6.2. This leads to the definition of ambiguous sentences in a context-free language.

Definition 6.6 A terminal string $w \in L(G)$ is ambiguous if there exist two or more derivation trees for w (or there exist two or more leftmost derivations of w).

Consider, for example, $G = (\{S\}, \{a, b, +, *\}, P, S)$, where P consists of $S \to S + S | S * S | a | b$. We have two derivation trees for $a + a * b$ given in Fig. 6.10.

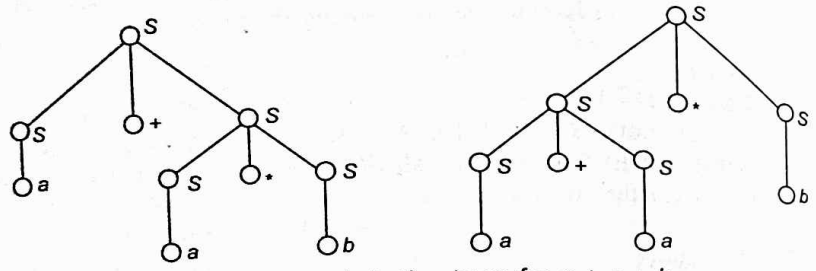

Fig. 6.10 Two derivation trees for $a + a * b$.

The leftmost derivations of $a + a * b$ induced by the two derivation trees are

$$S \Rightarrow S + S \Rightarrow a + S \Rightarrow a + S * S \Rightarrow a + a * S \Rightarrow a + a * b$$

$$S \Rightarrow S * S \Rightarrow S + S * S \Rightarrow a + S * S \Rightarrow a + a * S \Rightarrow a + a * b$$

Therefore, $a + a * b$ is ambiguous.

Definition 6.7 A context-free grammar G is ambiguous if there exists some $w \in L(G)$, which is ambiguous.

EXAMPLE 6.4

If G is the grammar $S \to SbS | a$, show that G is ambiguous.

Solution

To prove that G is ambiguous, we have to find a $w \in L(G)$, which is ambiguous. Consider $w = abababa \in L(G)$. Then we get two derivation trees for w (see Fig. 6.11). Thus, G is ambiguous.

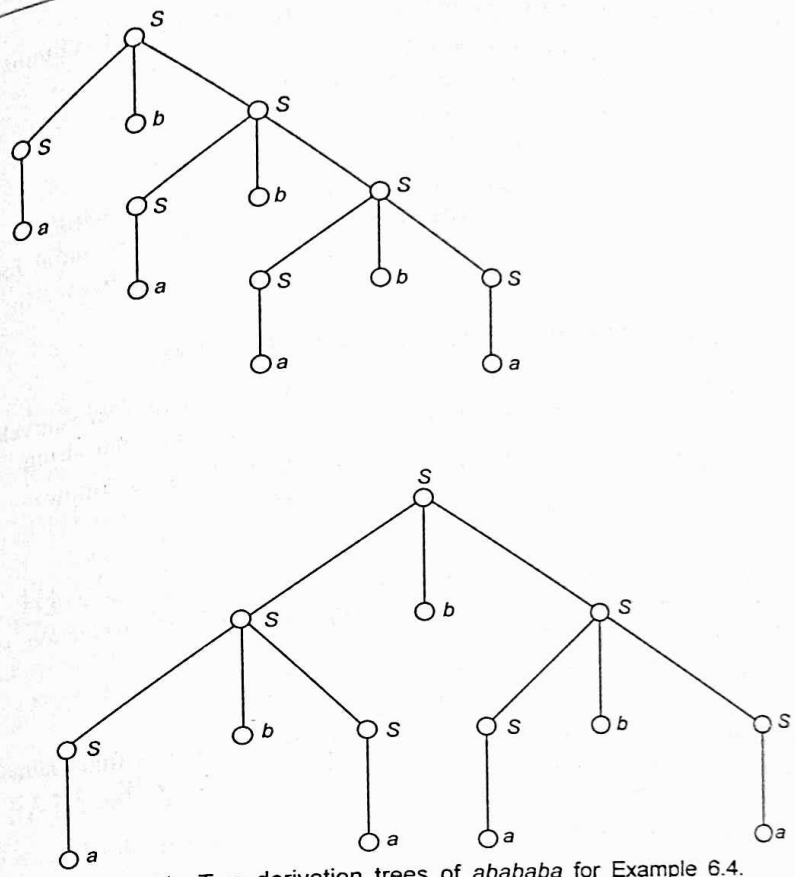

Fig. 6.11 Two derivation trees of *abababa* for Example 6.4.

6.3 SIMPLIFICATION OF CONTEXT-FREE GRAMMARS

In a CFG G, it may not be necessary to use all the symbols in $V_N \cup \Sigma$, or all the productions in P for deriving sentences. So when we study a context-free language $L(G)$, we try to eliminate those symbols and productions in G which are not useful for the derivation of sentences.

Consider, for example,

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

where

$$P = \{S \to AB, A \to a, B \to b, B \to C, E \to c | \Lambda\}$$

It is easy to see that $L(G) = \{ab\}$. Let $G' = (\{S, A, B\}, \{a, b\}, P', S)$, where P' consists of $S \to AB$, $A \to a$, $B \to b$. $L(G) = L(G')$. We have eliminated the symbols C, E and c and the productions $B \to C$, $E \to c | \Lambda$. We note the

following points regarding the symbols and productions which are eliminated:

(i) C does not derive any terminal string.
(ii) E and c do not appear in any sentential form.
(iii) $E \to \Lambda$ is a null production.
(iv) $B \to C$ simply replaces B by C.

In this section, we give the construction to eliminate (i) variables not deriving terminal strings, (ii) symbols not appearing in any sentential form, (iii) null productions, and (iv) productions of the form $A \to B$.

6.3.1 Construction of Reduced Grammars

Theorem 6.3 If G is a CFG such that $L(G) \neq \emptyset$, we can find an equivalent grammar G' such that each variable in G' derives some terminal string.

Proof *Let* $G = (V_N, \Sigma, P, S)$. We define $G' = (V'_N, \Sigma, P', S)$ as follows:

(a) *Construction of* V'_N:

We define $W_i \subseteq V_N$ by recursion:

$W_1 = \{A \in V_N | \text{there exists a production } A \to w \text{ where } w \in \Sigma^*\}$. (If $W_1 = \emptyset$, some variable will remain after the application of any production, and so $L(G) = \emptyset$.)

$$W_{i+1} = W_i \cup \{A \in V_N | \text{there exists some production } A \to \alpha \text{ with } \alpha \in (\Sigma \cup W_i)^*\}$$

By the definition of W_i, $W_i \subseteq W_{i+1}$ for all i. As V_N has only a finite number of variables, $W_k = W_{k+1}$ for some $k \leq |V_N|$. Therefore, $W_k = W_{k+j}$ for $j \geq 1$. We define $V'_N = W_k$.

(b) *Construction of* P':

$$P' = \{A \to \alpha | A, \alpha \in (V'_N \cup \Sigma)^*\}$$

We can define $G' = (V'_N, \Sigma, P', S)$. S is in V_N. (We are going to prove that every variable in V_N derives some terminal string. So if $S \notin V_N$, $L(G) = \emptyset$. But $L(G) \neq \emptyset$.)

Before proving that G' is the required grammar, we apply the construction to an example.

EXAMPLE 6.5

Let $G = (V_N, \Sigma, P, S)$ be given by the productions $S \to AB$, $A \to a$, $B \to b$, $B \to C$, $E \to c$. Find G' such that every variable in G' derives some terminal string.

Solution

(a) *Construction of* V'_N:

$W_1 = \{A, B, E\}$ since $A \to a$, $B \to b$, $E \to c$ are productions with a terminal string on the R.H.S.

$$W_2 = W_1 \cup \{A_1 \in V_N | A_1 \to \alpha \text{ for some } \alpha \in (\Sigma \cup \{A, B, E\})^*\}$$
$$= W_1 \cup \{S\} = \{A, B, E, S\}$$
$$W_3 = W_2 \cup \{A_1 \in V_N | A_1 \to \alpha \text{ for some } \alpha \in (\Sigma \cup \{S, A, B, E\})^*\}$$
$$= W_2 \cup \emptyset = W_2$$

Therefore, $V'_N = \{S, A, B, E\}$

(b) *Construction of* P':

$$P' = \{A_1 \to \alpha | A_1, \alpha \in (V'_N \cup \Sigma)^*\}$$
$$= \{S \to AB, A \to a, B \to b, E \to c\}$$

Therefore, $G' = (\{S, A, B, E\}, \{a, b, c\}, P', S)$

Now we prove:

(i) If each $A \in V'_N$, then $A \overset{*}{\underset{G'}{\Rightarrow}} w$ for some $w \in \Sigma^*$; conversely, if $A \overset{*}{\underset{G}{\Rightarrow}} w$, then $A \in V'_N$.
(ii) $L(G') = L(G)$.

To prove (i) we note that $W_k = W_1 \cup W_2 \ldots \cup W_k$. We prove by induction on i that for $i = 1, 2, \ldots, k$, $A \in W_i$ implies $A \overset{*}{\underset{G'}{\Rightarrow}} w$ for some $w \in \Sigma^*$. If $A \in W_1$, then $A \underset{G}{\Rightarrow} w$. So the production $A \to w$ is in P'. Therefore, $A \overset{*}{\underset{G'}{\Rightarrow}} w$. Thus there is basis for induction. Let us assume the result for i. Let $A \in W_{i+1}$. Then either $A \in W_i$, in which case, $A \overset{*}{\underset{G'}{\Rightarrow}} w$ for some $w \in \Sigma^*$ by induction hypothesis. Or, there exists a production $A \to \alpha$ with $\alpha \in (\Sigma \cup W_i)^*$. By definition of P', $A \to \alpha$ is in P'. We can write $\alpha = X_1 X_2 \ldots X_m$, where $X_j \in \Sigma \cup W_i$. If $X_j \in W_i$ by induction hypothesis, $X_j \overset{*}{\underset{G'}{\Rightarrow}} w_j$ for some $w_j \in \Sigma^*$. So, $A \overset{*}{\underset{G'}{\Rightarrow}} w_1 w_2 \ldots w_m \in \Sigma^*$ (when X_j is a terminal, $w_j = X_j$). By induction the result is true for $i = 1, 2, \ldots, k$.

The converse part can be proved in a similar way by induction on the number of steps in the derivation $A \overset{*}{\underset{G}{\Rightarrow}} w$. We see immediately that $L(G') \subseteq L(G)$ as $V'_N \subseteq V_N$ and $P' \subseteq P$. To prove $L(G) \subseteq L(G')$, we need an auxiliary result

$$A \overset{*}{\underset{G'}{\Rightarrow}} w \quad \text{if } A \overset{*}{\underset{G}{\Rightarrow}} w \text{ for some } w \in \Sigma^* \tag{6.5}$$

We prove (6.5) by induction on the number of steps in the derivation $A \overset{*}{\underset{G}{\Rightarrow}} w$. If $A \underset{G}{\Rightarrow} w$, then $A \to w$ is in P and $A \in W_1 \subseteq V'_N$. As $A \in V'_N$ and $w \in \Sigma^*$, $A \to w$ is in P'. So $A \underset{G'}{\Rightarrow} w$, and there is basis for induction. Assume (6.5) for derivations in at most k steps. Let $A \overset{k+1}{\underset{G}{\Rightarrow}} w$. By Remark appearing after

Theorem 6.1, we can split this as $A \underset{G}{\Rightarrow} X_1X_2 \ldots X_m \overset{*}{\underset{G}{\Rightarrow}} w_1w_2 \ldots w_m$ such that $X_j \overset{*}{\underset{G}{\Rightarrow}} w_j$. If $X_j \in \Sigma$, then $w_j = X_j$.

If $X_j \in V_N$ then by (i), $X_j \in V'_N$. As $X_j \overset{*}{\underset{G}{\Rightarrow}} w_j$ in at most k steps, $X_j \overset{*}{\underset{G'}{\Rightarrow}} w_j$. Also, $X_1, X_2, X_m \in (\Sigma \cup V'_N)^*$ implies that $A \to X_1X_2 \ldots X_m$ is in P'. Thus, $A \underset{G'}{\Rightarrow} X_1X_2 \ldots X_m \overset{*}{\underset{G'}{\Rightarrow}} w_1w_2 \ldots w_m$. Hence by induction, (6.5) is true for all derivations. In particular, $S \overset{*}{\underset{G}{\Rightarrow}} w$ implies $S \overset{*}{\underset{G'}{\Rightarrow}} w$. This proves that $L(G) \subseteq L(G')$, and (ii) is completely proved. ∎

Theorem 6.4 For every CFG $G = (V_N, \Sigma, P, S)$, we can construct an equivalent grammar $G' = (V_N, \Sigma', P', S)$ such that every symbol in $V'_N \cup \Sigma$ appears in some sentential form (i.e. for every X in $V'_N \cup \Sigma'$ there exists α such that $S \overset{*}{\underset{G'}{\Rightarrow}} \alpha$ and X is a symbol in the string α).

Proof We construct $G' = (V'_N, \Sigma', P', S)$ as follows:

(a) *Construction of W_i for $i \geq 1$:*

(i) $W_1 = \{S\}$.

(ii) $W_{i+1} = W_i \cup \{X \in V_N \cup \Sigma \mid$ there exists a production $A \to \alpha$ with $A \in W_i$ and α containing the symbol $X\}$.

We may note that $W_i \subseteq V_N \cup \Sigma$ and $W_i \subseteq W_{i+1}$. As we have only a finite number of elements in $V_N \cup \Sigma$, $W_k = W_{k+1}$ for some k. This means that $W_k = W_{k+j}$ for all $j \geq 0$.

(b) *Construction of V'_N, Σ' and P':*

We define

$$V'_N = V_N \cap W_k, \qquad \Sigma' = \Sigma \cap W_k$$
$$P' = \{A \to \alpha \mid A \in W_k\}.$$

Before proving that G' is the required grammar, we apply the construction to an example.

EXAMPLE 6.6

Consider $G = (\{S, A, B, E\}, \{a, b, c\}, P, S)$, where P consists of $S \to AB$, $A \to a$, $B \to b$, $E \to c$.

Solution

$$W_1 = \{S\}$$
$$W_2 = \{S\} \cup \{X \in V_N \cup \Sigma \mid \text{there exists a production } A \to \alpha \text{ with } A \in W_1 \text{ and } \alpha \text{ containing } X\}$$
$$= \{S\} \cup \{A, B\}$$

$$W_3 = \{S, A, B\} \cup \{a, b\}$$
$$W_4 = W_3$$
$$V'_N = \{S, A, B\} \qquad \Sigma' = \{a, b\}$$
$$P' = \{S \to AB, A \to a, B \to b\}$$

Thus the required grammar is $G' = (V'_N, \Sigma', P', S)$.

To complete the proof, we have to show that (i) every symbol in $V'_N \cup \Sigma'$ appears in some sentential form of G', and (ii) conversely, $L(G') = L(G)$.

To prove (i), consider $X \in V'_N \cup \Sigma' = W_k$. By construction $W_k = W_1 \cup W_2 \ldots \cup W_k$. We prove that $X \in W_i$, $i \leq k$, appears in some sentential form by induction on i. When $i = 1$, $X = S$ and $S \overset{*}{\underset{G'}{\Rightarrow}} S$. Thus, there is basis for induction. Assume the result for all variables in W_i. Let $X \in W_{i+1}$. Then either $X \in W_i$, in which case, X appears in some sentential form by induction hypothesis. Otherwise, there exists a production $A \to \alpha$, where $A \in W_i$ and α contains the symbol X_i. The A appears in some sentential form, say $\beta A\gamma$. Therefore,

$$S \overset{*}{\underset{G'}{\Rightarrow}} \beta A\gamma \underset{G'}{\Rightarrow} \beta\alpha\gamma$$

This means that $\beta\alpha\gamma$ is some sentential form and X is a symbol in $\beta\alpha\gamma$. Thus by induction the result is true for $X \in W_i$, $i \leq k$.

Conversely, if X appears in some sentential form, say $\beta X\gamma$, then $\Sigma \overset{l}{\underset{G}{\Rightarrow}} \beta X\gamma$. This implies $X \in W_l$. If $l \leq k$, then $W_l \subseteq W_k$. If $l > k$, then $W_l = W_k$. Hence X appears in $V'_N \cup \Sigma'$. This proves (i).

To prove (ii), we note $L(G') \subseteq L(G)$ as $V'_N \subseteq V_N$, $\Sigma' \subseteq \Sigma$ and $P' \subseteq P$. Let w be in $L(G)$ and $S = \alpha_1 \underset{G}{\Rightarrow} \alpha_2 \underset{G}{\Rightarrow} \alpha_3 = \ldots \Rightarrow \alpha_{n-1} \underset{G}{\Rightarrow} w$. We prove that every symbol in α_{i+1} is in W_{i+1} and $\alpha_i \underset{G'}{\Rightarrow} \alpha_{i+1}$ by induction on i. $\alpha_1 = S \underset{G}{\Rightarrow} \alpha_2$ implies $S \to \alpha_2$ is a production in P. By construction, every symbol in α_2 is in W_2 and $S \to \alpha_2$ is in P', i.e. $S \underset{G'}{\Rightarrow} \alpha_2$. Thus, there is basis for induction. Let us assume the result for i. Consider $\alpha_{i+1} \underset{G}{\Rightarrow} \alpha_{i+2}$. This one-step derivation can be written in the form

$$\beta_{i+1}A\gamma_{i+1} \Rightarrow \beta_{i+1}\alpha\gamma_{i+1}$$

where $A \to \alpha$ is the production we are applying. By induction hypothesis, $A \in W_{i+1}$. By construction of W_{i+2}, every symbol in α is in W_{i+2}. As all the symbols in β_{i+1} and γ_{i+1} are also in W_{i+1} by induction hypothesis, every symbol in $\beta_{i+1}\alpha\gamma_{i+1} = \alpha_{i+2}$ is in W_{i+2}. By the construction of P', $A \to \alpha$ is in P'. This means that $\alpha_{i+1} \underset{G'}{\Rightarrow} \alpha_{i+2}$. Thus the induction procedure is complete. So $S = \alpha_1 \underset{G'}{\Rightarrow} \alpha_2 \underset{G'}{\Rightarrow} \alpha_3 \overset{*}{\underset{G'}{\Rightarrow}} \ldots \alpha_{n-1} \underset{G'}{\Rightarrow} w$. Therefore, $w \in L(G')$. This proves (ii). ∎

Definition 6.8 Let $G = (V_N, \Sigma, P, S)$ be a CFG. G is said to be reduced or non-redundant if every symbol in $V_N \cup \Sigma$ appears in the course of the derivation of some terminal string, i.e. for every X in $V_N \cup \Sigma$, there exists a derivation $S \stackrel{*}{\Rightarrow} \alpha X \beta \stackrel{*}{\Rightarrow} w \in L(G)$. (We can say X is useful in the derivation of terminal strings.)

Theorem 6.5 For every CFG G there exists a reduced grammar G' which is equivalent to G.

Proof We construct the reduced grammar in two steps.

Step 1 We construct a grammar G_1 equivalent to the given grammar G so that every variable in G_1 derives some terminal string (Theorem 6.3).

Step 2 We construct a grammar $G' = (V'_N, \Sigma', P', S)$ equivalent to G_1 so that every symbol in G' appears in some sentential form of G' which is equivalent to G_1 and hence to G. G' is the required reduced grammar.

By step 2 every symbol X in G' appears in some sentential form, say $\alpha X \beta$. By step 1 every symbol in $\alpha X \beta$ derives some terminal string. Therefore, $S \stackrel{*}{\Rightarrow} \alpha X \beta \stackrel{*}{\Rightarrow} w$ for some w in Σ^*, i.e. G' is reduced.

Note: To get a reduced grammar, we must first apply Theorem 6.3 and then Theorem 6.4. For, if we apply Theorem 6.4 first and then Theorem 6.3, we may not get a reduced grammar (refer to Exercise 6.8 at the end of the chapter).

EXAMPLE 6.7

Find a reduced grammar equivalent to the grammar G whose productions are

$$S \to AB \mid CA, \qquad B \to BC \mid AB, \qquad A \to a, \qquad C \to aB \mid b$$

Solution

Step 1 $W_1 = \{A, C\}$ as $A \to a$ and $C \to b$ are productions with a terminal string on R.H.S.

$W_2 = \{A, C\} \cup \{A_1 \mid A_1 \to \alpha$ with $\alpha \in (\Sigma \cup \{A, C\})^*\}$

$= \{A, C\} \cup \{S\}$ as we have $S \to CA$

$W_3 = \{A, C, S\} \cup \{A_1 \mid A_1 \to \alpha$ with $\alpha \in (\Sigma \cup \{S, A, C\})^*\}$

$= \{A, C, S\} \cup \emptyset$

As $W_3 = W_2$,

$V'_N = W_2 = \{S, A, C\}$

$P' = \{A_1 \to \alpha \mid A_1, \alpha \in (V'_N \cup \Sigma)^*\}$

$= \{S \to CA, A \to a, C \to b\}$

Thus,

$G_1 = (\{S, A, C\}, \{a, b\}, \{S \to CA, A \to a, C \to b\}, S)$

Step 2 We have to apply Theorem 6.4 to G_1. Thus,

$$W_1 = \{S\}$$

As we have production $S \to CA$ and $S \in W_1$, $W_2 = \{S\} \cup \{A, C\}$

As $A \to a$ and $C \to b$ are productions with $A, C \in W_2$, $W_3 = \{S, A, C, a, b\}$

As $W_3 = V'_N \cup \Sigma$, $P'' = \{S \to a \mid A_1 \in W_3\} = P'$

Therefore,

$G' = (\{S, A, C\}, \{a, b\}, \{S \to CA, A \to a, C \to b\}, S)$

is the reduced grammar.

EXAMPLE 6.8

Construct a reduced grammar equivalent to the grammar

$$S \to aAa, \qquad A \to Sb \mid bCC \mid DaA, \qquad C \to abb \mid DD,$$
$$E \to aC, \qquad D \to aDA$$

Solution

Step 1 $W_1 = \{C\}$ as $C \to abb$ is the only production with a terminal string on the R.H.S.

$W_2 = \{C\} \cup \{E, A\}$

as $E \to aC$ and $A \to bCC$ are productions with R.H.S. in $(\Sigma \cup \{C\})^*$

$W_3 = \{C, E, A\} \cup \{S\}$

as $S \to aAa$ and aAa is in $(\Sigma \cup W_2)^*$

$W_4 = W_3 \cup \emptyset$

Hence,

$V'_N = W_3 = \{S, A, C, E\}$

$P' = \{A_1 \to \alpha \mid \alpha \in (V_N \cup \Sigma)^*\}$

$= \{S \to aAa, A \to Sb \mid bCC, C \to abb, E \to aC\}$

$G_1 = (V'_N, \{a, b\}, P', S)$

Step 2 We have to apply Theorem 6.4 to G_1. We start with

$$W_1 = \{S\}$$

As we have $S \to aAa$,

$$W_2 = \{S\} \cup \{A, a\}$$

As $A \to Sb \mid bCC$,

$$W_3 = \{S, A, a\} \cup \{S, b, C\} = \{S, A, C, a, b\}$$

As we have $C \to abb$,

$$W_4 = W_3 \cup \{a, b\} = W_3$$

Hence,
$$P'' = \{A_1 \to \alpha \mid A_1 \in W_3\}$$
$$= \{S \to aAa, A \to Sb \mid bCC, C \to abb\}$$

Therefore,
$$G' = (\{S, A, C\}, \{a, b\}, P'', S)$$

is the reduced grammar.

6.3.2 Elimination of Null Productions

A context-free grammar may have productions of the form $A \to \Lambda$. The production $A \to \Lambda$ is just used to erase A. So a production of the form $A \to \Lambda$, where A is a variable, is called a *null production*. In this section we give a construction to eliminate null productions.

As an example, consider G whose productions are $S \to aS \mid aA \mid A$, $A \to \Lambda$. We have two null productions $S \to \Lambda$ and $A \to \Lambda$. We can delete $A \to \Lambda$ provided we erase A whenever it occurs in the course of a derivation of a terminal string. So we can replace $S \to aA$ by $S \to a$. If G_1 denotes the grammar whose productions are $S \to aS \mid a \mid \Lambda$, then $L(G_1) = L(G) = \{a^n \mid n \geq 0\}$. Thus it is possible to eliminate the null production $A \to \Lambda$. If we eliminate $S \to \Lambda$, we cannot generate Λ in $L(G)$. But we can generate $L(G) = \{\Lambda\}$ even if we eliminate $S \to \Lambda$.

Before giving the construction we give a definition.

Definition 6.9 A variable A in a context-free grammar is nullable if $A \overset{*}{\Rightarrow} \Lambda$.

Theorem 6.6 If $G = (V_N, \Sigma, P, S)$ is a context-free grammar, then we can find a context-free grammar G_1 having no null productions such that $L(G_1) = L(G) - \{\Lambda\}$.

Proof We construct $G_1 = (V_N, \Sigma, P', S)$ as follows:

Step 1 *Construction of the set of nullable variables:*

We find the nullable variables recursively:

(i) $W_1 = \{A \in V_N \mid A \to \Lambda \text{ is in } P\}$
(ii) $W_{i+1} = W_i \cup \{A \in V_N \mid$ there exists a production $A \to \alpha$ with $\alpha \in W_i^*\}$.

By definition of W_i, $W_i \subseteq W_{i+1}$ for all i. As V_N is finite, $W_{k+1} = W_k$ for some $k \leq |V_N|$. So, $W_{k+j} = W_k$ for all j. Let $W = W_k$. W is the set of all nullable variables.

Step 2 (i) *Construction of P':*

Any production whose R.H.S. does not have any nullable variable is included in P'.

(ii) If $A \to X_1X_2 \ldots X_k$ is in P, the productions of the form $A \to \alpha_1\alpha_2 \ldots \alpha_k$ are included in P', where $\alpha_j = X_i$ if $X_i \notin W$. $\alpha_i = X_i$ or Λ if $X_i \in W$ and $\alpha_1\alpha_2 \ldots \alpha_k \neq \Lambda$. Actually, (ii) gives several productions in P'. The productions are obtained either by not erasing any nullable variable on the

R.H.S. of $A \to X_1X_2 \ldots X_k$ or by erasing some or all nullable variables provided some symbol appears on the R.H.S. after erasing.

Let $G_1 = (V_N, \Sigma, P', S)$. G_1 has no null productions.

Before proving that G_1 is the required grammar, we apply the construction to an example.

EXAMPLE 6.9

Consider the grammar G whose productions are $S \to aS \mid AB$, $A \to \Lambda$, $B \to \Lambda$, $D \to b$. Construct a grammar G_1 without null productions generating $L(G) - \{\Lambda\}$.

Solution

Step 1 *Construction of the set W of all nullable variables:*

$$W_1 = \{A_1 \in V_N \mid A_1 \to \Lambda \text{ is a production in } G\}$$
$$= \{A, B\}$$
$$W_2 = \{A, B\} \cup \{S\} \text{ as } S \to AB \text{ is a production with } AB \in W_1^*$$
$$= \{S, A, B\}$$
$$W_3 = W_2 \cup \emptyset = W_2$$

Thus,
$$W = W_2 = \{S, A, B\}$$

Step 2 *Construction of P':*

(i) $D \to b$ is included in P'.
(ii) $S \to aS$ gives rise to $S \to aS$ and $S \to a$.
(iii) $S \to AB$ gives rise to $S \to AB$, $S \to A$ and $S \to B$.

(***Note:*** We cannot erase both the nullable variables A and B in $S \to AB$ as we will get $S \to \Lambda$ in that case.)

Hence the required grammar without null productions is

$$G_1 = (\{S, A, B, D\}, \{a, b\}, P, S)$$

where P' consists of

$$D \to b,\ S \to aS,\ S \to AB,\ S \to a,\ S \to A,\ S \to B$$

Step 3 $L(G_1) = L(G) - \{\Lambda\}$. To prove that $L(G) = L(G) - \{\Lambda\}$, we prove an auxiliary result given by the following relation:

For all $A \in V_N$ and $w \in \Sigma^*$,

$$A \overset{*}{\underset{G_1}{\Rightarrow}} w \text{ if and only if } A \overset{*}{\underset{G}{\Rightarrow}} w \text{ and } w \to \Lambda \tag{6.6}$$

We prove the 'if' part first. Let $A \overset{*}{\underset{G}{\Rightarrow}} w$ and $w \neq \Lambda$. We prove that $A \overset{*}{\underset{G_1}{\Rightarrow}} w$ by induction on the number of steps in the derivation $A \overset{*}{\underset{G}{\Rightarrow}} w$. If $A \underset{G}{\Rightarrow} w$ and

$w \neq \Lambda$, $A \to w$ is a production in P', and so $A \underset{G_1}{\Rightarrow} w$. Thus there is basis for induction. Assume the result for derivations in at most i steps. Let $A \overset{i+1}{\underset{G}{\Rightarrow}} w$ and $w \neq \Lambda$. We can split the derivation as $A \underset{G}{\Rightarrow} X_1X_2 \ldots X_k \overset{i}{\underset{G}{\Rightarrow}} w_1w_2 \ldots w_k$, where $w = w_1w_2 \ldots w_k$ and $A_j \overset{*}{\underset{G}{\Rightarrow}} w_j$. As $w \neq \Lambda$, not all w_j's are Λ. If $w_j \neq \Lambda$, then by induction hypothesis, $X_j \overset{i}{\underset{G}{\Rightarrow}} w_j$. If $w_j = \Lambda$, then $X_j \in W$. So using the production $A \to A_1A_2 \ldots A_k$ in P, we construct $A \to \alpha_1\alpha_2 \ldots \alpha_k$ in P', where $\alpha_j = X_j$ if $w_j \neq \Lambda$ and $\alpha_j = \Lambda$ if $w_j = \Lambda$ (i.e. $X_j \in W$). Therefore,

$$A \underset{G_1}{\Rightarrow} \alpha_1\alpha_2 \ldots \alpha_k \overset{*}{\underset{G_1}{\Rightarrow}} w_1\alpha_2 \ldots \alpha_k \overset{*}{\underset{G_1}{\Rightarrow}} \ldots \Rightarrow w_1w_2 \ldots w_k = w$$

By the principle of induction, the 'if' part of (6.6) is proved.

We prove the 'only if' part by induction on the number of steps in the derivation of $A \overset{*}{\underset{G_1}{\Rightarrow}} w$. If $A \underset{G_1}{\Rightarrow} w$, then $A \to w$ is in P_1. By construction of P', $A \to w$ is obtained from some production $A \to X_1X_2 \ldots X_n$ in P by erasing some (or none of the) nullable variables. Hence $A \underset{G}{\Rightarrow} X_1X_2 \ldots X_n \overset{*}{\underset{G}{\Rightarrow}} w$. So there is basis for induction. Assume the result for derivation in at most j steps. Let $A \overset{j+1}{\underset{G}{\Rightarrow}} w$. This can be split as $A \underset{G_1}{\Rightarrow} X_1X_2 \ldots X_k \overset{j}{\underset{G_1}{\Rightarrow}} w_1w_2 \ldots w_k$, where $X_i \overset{*}{\underset{G}{\Rightarrow}} w_i$. The first production $A \to X_1X_2 \ldots X_k$ in P' is obtained from some production $A \to \alpha$ in P by erasing some (or none of the) nullable variables in α. So $A \underset{G}{\Rightarrow} \alpha \overset{*}{\underset{G}{\Rightarrow}} X_1X_2 \ldots X_k$. If $X_i \in \Sigma$ then $X_i \overset{0}{\underset{G}{\Rightarrow}} X_i = w_i$. If $X_i \in V_N$ then by induction hypothesis, $X_i \overset{*}{\underset{G}{\Rightarrow}} w_i$. So, we get $A \overset{*}{\underset{G}{\Rightarrow}} X_1X_2 \ldots X_k \overset{*}{\underset{G}{\Rightarrow}} w_1w_2 \ldots w_k$. Hence by the principle of induction whenever $A \overset{*}{\underset{G_1}{\Rightarrow}} w$, we have $A \overset{*}{\underset{G}{\Rightarrow}} w$ and $w \neq \Lambda$. Thus (6.6) is completely proved.

By applying (6.6) to S, we have $w \in L(G_1)$ if and only if $w \in L(G)$ and $w \neq \Lambda$. This implies $L(G_1) = L(G) - \{\Lambda\}$. ∎

Corollary 1 There exists an algorithm to decide whether $\Lambda \in L(G)$ for a given context-free grammar G.

Proof $\Lambda \in L(G)$ if and only if $S \in W$, i.e. S is nullable. The construction given in Theorem 6.6 is recursive and terminates in a finite number of steps (actually in at most $|V_N|$ steps). So the required algorithm is as follows: (i) construct W; (ii) test whether $S \in W$.

Corollary 2 If $G = (V_N, \Sigma, P, S)$ is a context-free grammar we can find an equivalent context-free grammar $G_1 = (V'_N, \Sigma, P, S_1)$ without null productions except $S_1 \to \Lambda$ when Λ is in $L(G)$. If $S_1 \to \Lambda$ is in P_1, S_1 does not appear on the R.H.S. of any production in P_1.

Proof By Corollary 1, we can decide whether Λ is in $L(G)$.

Case 1 If Λ is not in $L(G)$, G_1 obtained by using Theorem 6.6 is the required equivalent grammar.

Case 2 If Λ is in $L(G)$, construct $G' = (V_N, \Sigma, P', S)$ using Theorem 6.6. $L(G') = L(G) - \{\Lambda\}$. Define $G_1 = (V_N \cup \{S_1\}, \Sigma, P_1, S_1)$, where $P_1 = P' \cup \{S_1 \to S, S_1 \to \Lambda\}$. S_1 does not appear on the R.H.S. of any production in P_1, and so G_1 is the required grammar with $L(G_1) = L(G)$. ∎

6.3.3 Elimination of Unit Productions

A context-free grammar may have productions of the form $A \to B$, $A, B \in V_N$.

Consider, for example, G as the grammar $S \to A$, $A \to B$, $B \to C$, $C \to a$. It is easy to see that $L(G) = \{a\}$. The productions $S \to A$, $A \to B$, $B \to C$ are useful just to replace S by C. To get a terminal string, we need $C \to a$. If G_1 is $S \to a$, then $L(G_1) = L(G)$.

The next construction eliminates productions of the form $A \to B$.

Definition 6.10 A unit production (or a chain rule) in a context-free grammar G is a production of the form $A \to B$, where A and B are variables in G.

Theorem 6.7 If G is a context-free grammar,we can find a context-free grammar G_1 which has no null productions or unit productions such that $L(G_1) = L(G)$.

Proof We can apply Corollary 2 of Theorem 6.6 to grammar G to get a grammar $G' = (V_N, \Sigma, P, S)$ without null productions such that $L(G') = L(G)$. Let A be any variable in V_N.

Step 1 *Construction of the set of variables derivable from A:*

Define $W_i(A)$ recursively as follows:

$$W_0(A) = \{A\}$$

$$W_{i+1}(A) = W_i(A) \cup \{B \in V_N \mid C \to B \text{ is in } P \text{ with } C \in W_i(A)\}$$

By definition of $W_i(A)$, $W_i(A) \subseteq W_{i+1}(A)$. As V_N is finite, $W_{k+1}(A) = W_k(A)$ for some $k \leq |V_N|$. So, $W_{k+j}(A) = W_j(A)$ for all $j \geq 0$. Let $W(A) = W_k(A)$. Then $W(A)$ is the set of all variables derivable from A.

Step 2 *Construction of A-productions in* G_1*:*

The A-productions in G_1 are either (i) the nonunit production in G' or (ii) $A \to \alpha$ whenever $B \to \alpha$ is in G with $B \in W(A)$ and $\alpha \notin V_N$.

(Actually, (ii) covers (i) as $A \in W(A)$). Now, we define $G_1 = (V_N, \Sigma, P_1, S)$, where P_1 is constructed using step 2 for every $A \in V_N$.

Before proving that G_1 is the required grammar, we apply the construction to an example.

EXAMPLE 6.10

Let G be $S \to AB$, $A \to a$, $B \to C \mid b$, $C \to D$, $D \to E$ and $E \to a$. Eliminate unit productions and get an equivalent grammar.

Solution

Step 1 $W_0(S) = \{S\}$, $W_1(S) = W_0(S) \cup \emptyset$

Hence $W(S) = \{S\}$. Similarly,

$W(A) = \{A\}$, $W(E) = \{E\}$

$W_0(B) = \{B\}$, $W_1(B) = \{B\} \cup \{C\} = \{B, C\}$

$W_2(B) = \{B, C\} \cup \{D\}$, $W_3(B) = \{B, C, D\} \cup \{E\}$, $W_4(B) = W_3(B)$

Therefore,

$$W(B) = \{B, C, D, E\}$$

Similarly,

$W_0(C) = \{C\}$, $W_1(C) = \{C, D\}$, $W_2\{C\} = \{C, D, E\} = W_3(C)$

Therefore,

$$W(C) = \{C, D, E\}, \quad W_0(D) = \{D\}$$

Hence,

$$W_1(D) = \{D, E\} = W_2(D)$$

Thus,

$$W(D) = \{D, E\}$$

Step 2 The productions in G_1 are

$$S \to AB, \quad A \to a, \quad E \to a$$

$$B \to b \mid a, \quad C \to a, \quad D \to a$$

By construction, G_1 has no unit productions.

To complete the proof we have to show that $L(G') = L(G_1)$.

Step 3 $L(G') = L(G)$. If $A \to \alpha$ is in $P_1 - P$, then it is induced by $B \to \alpha$ in P with $B \in W(A)$, $\alpha \notin V_N$, $B \in W(A)$ implies $A \overset{*}{\underset{G'}{\Rightarrow}} B$. Hence, $A \overset{*}{\underset{G'}{\Rightarrow}} B \underset{G'}{\Rightarrow} \alpha$. So, if $A \underset{G_1}{\Rightarrow} \alpha$, then $A \overset{*}{\underset{G'}{\Rightarrow}} \alpha$. This proves $L(G_1) \subseteq L(G')$.

To prove the reverse inclusion, we start with a leftmost derivation

$$S \underset{G}{\Rightarrow} \alpha_1 \underset{G}{\Rightarrow} \alpha_2 \ldots \underset{G}{\Rightarrow} \alpha_n = w$$

in G'.

Let i be the smallest index such that $\alpha_i \overset{*}{\underset{G'}{\Rightarrow}} \alpha_{i+1}$ is obtained by a unit production and j be the smallest index greater than i such that $\alpha_j \underset{G}{\Rightarrow} \alpha_{j+1}$ is obtained by a nonunit production. So, $S \overset{*}{\underset{G_1}{\Rightarrow}} \alpha_i$, and $\alpha_j \overset{*}{\underset{G'}{\Rightarrow}} \alpha_{j+1}$ can be written as

$$\alpha_i = w_i A_i \beta_i \Rightarrow w_i A_{i+1} \beta_i \Rightarrow \ldots \Rightarrow w_i A_j \beta_i \Rightarrow w_i \gamma \beta_i = \alpha_{j+1}$$

$A_j \in W(A_i)$ and $A_j \to \gamma$ is a nonunit production. Therefore, $A_j \to \gamma$ is a production in P_1. Hence, $\alpha_j \overset{*}{\underset{G_1}{\Rightarrow}} \alpha_{j+1}$. Thus, we have $S \overset{*}{\underset{G_1}{\Rightarrow}} \alpha_{j+1}$.

Repeating the argument whenever some unit production occurs in the remaining part of the derivation, we can prove that $S \overset{*}{\underset{G_1}{\Rightarrow}} \alpha_n = w$. This proves $L(G') \subseteq L(G)$. ∎

Corollary If G is a context-free grammar, we can construct an equivalent grammar G' which is reduced and has no null productions or unit productions.

Proof We construct G_1 in the following way:

Step 1 Eliminate null productions to get G_1 (Theorem 6.6 or Corollary 2 of this theorem).

Step 2 Eliminate unit productions in G_1 to get G_2 (Theorem 6.7).

Step 3 Construct a reduced grammar G' equivalent to G_1 (Theorem 6.5). G' is the required grammar equivalent to G.

Note: We have to apply the constructions only in the order given in the corollary of Theorem 6.7 to simplify grammars. If we change the order we may not get the grammar in the most simplified form (refer to Exercise 6.11).

6.4 NORMAL FORMS FOR CONTEXT-FREE GRAMMARS

In a context-free grammar, the R.H.S. of a production can be any string of variables and terminals. When the productions in G satisfy certain restrictions, then G is said to be in a 'normal form'. Among several 'normal forms' we study two of them in this section—the Chomsky normal form (CNF) and the Greibach normal form.

6.4.1 CHOMSKY NORMAL FORM

In the Chomsky normal form (CNF), we have restrictions on the length of R.H.S. and the nature of symbols in the R.H.S. of productions.

Definition 6.11 A context-free grammar G is in Chomsky normal form if every production is of the form $A \to a$, or $A \to BC$, and $S \to \Lambda$ is in G if

$\Lambda \in L(G)$. When Λ is in $L(G)$, we assume that S does not appear on the R.H.S. of any production.

For example, consider G whose productions are $S \to AB \mid \Lambda$, $A \to a$, $B \to b$. Then G is in Chomsky normal form.

Remark For a grammar in CNF, the derivation tree has the following property: Every node has atmost two descendants—either two internal vertices or a single leaf.

When a grammar is in CNF, some of the proofs and constructions are simpler.

Reduction to Chomsky Normal Form

Now we develop a method of constructing a grammar in CNF equivalent to a given context-free grammar. Let us first consider an example. Let G be $S \to ABC \mid aC$, $A \to a$, $B \to b$, $C \to c$. Except $S \to aC \mid ABC$, all the other productions are in the form required for CNF. The terminal a in $S \to aC$ can be replaced by a new variable D. By adding a new production $D \to a$, the effect of applying $S \to aC$ can be achieved by $S \to DC$ and $D \to a$. $S \to ABC$ is not in the required form, and hence this production can be replaced by $S \to AE$ and $E \to BC$. Thus, an equivalent grammar is $S \to AE \mid DC$, $E \to BC$, $A \to a$, $B \to b$, $C \to c$, $D \to a$.

The techniques applied in this example are used in the following theorem.

Theorem 6.8 (Reduction to Chomsky normal form). For every context-free grammar, there is an equivalent grammar G_2 in Chomsky normal form.

Proof (Construction of a grammar in CNF)

Step 1 *Elimination of null productions and unit productions:*

We apply Theorem 6.6 to eliminate null productions. We then apply Theorem 6.7 to the resulting grammar to eliminate chain productions. Let the grammar thus obtained be $G = (V_N, \Sigma, P, S)$.

Step 2 *Elimination of terminals on R.H.S.:*

We define $G_1 = (V'_N, \Sigma, P_1, S')$, where P_1 and V'_N are constructed as follows:

(i) All the productions in P of the form $A \to a$ or $A \to BC$ are included in P_1. All the variables in V_N are included in V'_N.

(ii) Consider $A \to X_1X_2 \ldots X_n$ with some terminal on R.H.S. If X_i is a terminal, say a_i, add a new variable C_{a_i} to V'_N and $C_{a_i} \to a_i$ to P_1. In production $A \to X_1X_2 \ldots X_n$, every terminal on R.H.S. is replaced by the corresponding new variable and the variables on the R.H.S. are retained. The resulting production is added to P_1. Thus, we get $G_1 = (V'_N, \Sigma, P_1, S)$.

Step 3 *Restricting the number of variables on R.H.S.:*

For any production in P_1, the R.H.S. consists of either a single terminal (or

Λ in $S \to \Lambda$) or two or more variables. We define $G_2 = (V''_N, \Sigma, P_2, S)$ as follows:

(i) All productions in P_1 are added to P_2 if they are in the required form. All the variables in V'_N are added to V''_N.

(ii) Consider $A \to A_1A_2 \ldots A_m$, where $m \geq 3$. We introduce new productions $A \to A_1C_1$, $C_1 \to A_2C_2$, $\ldots$, $C_{m-2} \to A_{m-1}A_m$ and new variables $C_1, C_2, \ldots, C_{m-2}$. These are added to P'' and V''_N, respectively.

Thus, we get G_2 in Chomsky normal form.

Before proving that G_2 is the required equivalent grammar, we apply the construction to the context-free grammar given in Example 6.11.

EXAMPLE 6.11

Reduce the following grammar G to CNF. G is $S \to aAD$, $A \to aB \mid bAB$, $B \to b$, $D \to d$.

Solution

As there are no null productions or unit productions, we can proceed to step 2.

Step 2 Let $G_1 = (V'_N, \{a, b, d\}, P_1, S)$, where P_1 and V'_N are constructed as follows:

(i) $B \to b$, $D \to d$ are included in P_1.

(ii) $S \to aAD$ gives rise to $S \to C_aAD$ and $C_a \to a$.
$A \to aB$ gives rise to $A \to C_aB$.
$A \to bAB$ gives rise to $A \to C_bAB$ and $C_b \to b$.
$V'_N = \{S, A, B, D, C_a, C_b\}$.

Step 3 P_1 consists of $S \to C_aAD$, $A \to C_aB \mid C_bAB$, $B \to b$, $D \to d$, $C_a \to a$, $C_b \to b$.

$A \to C_aB$, $B \to b$, $D \to d$, $C_a \to a$, $C_b \to b$ are added to P_2

$S \to C_aAD$ is replaced by $S \to C_aC_1$ and $C_1 \to AD$.

$A \to C_bAB$ is replaced by $A \to C_bC_2$ and $C_2 \to AB$.

Let

$$G_2 = (\{S, A, B, D, C_a, C_b, C_1, C_2\}, \{a, b, d\}, P_2, S)$$

where P_2 consists of $S \to C_aC_1$, $A \to C_aB \mid C_bC_2$, $C_1 \to AD$, $C_2 \to AB$, $B \to b$, $D \to d$, $C_a \to a$, $C_b \to b$. G_2 is in CNF and equivalent to G.

Step 4 $L(G) = L(G_2)$. To complete the proof we have to show that $L(G) = L(G_1) = L(G_2)$.

To show that $L(G) \subseteq L(G_1)$, we start with $w \in L(G)$. If $A \to X_1X_2 \ldots X_n$ is used in the derivation of w, the same effect can be achieved by using the corresponding production in P_1 and the productions involving the new variables. Hence, $A \underset{G_1}{\overset{*}{\Rightarrow}} X_1X_2 \ldots X_n$. Thus, $L(G) \subseteq L(G_1)$.

Let $w \in L(G_1)$. To show that $w \in L(G)$, it is enough to prove the following:

$$A \overset{*}{\underset{G}{\Rightarrow}} w \quad \text{if } A \in V_N,\ A \overset{*}{\underset{G_1}{\Rightarrow}} w \tag{6.7}$$

We prove (6.7) by induction on the number of steps in $A \overset{*}{\underset{G_1}{\Rightarrow}} w$.

If $A \underset{G_1}{\Rightarrow} w$, then $A \to w$ is a production in P_1. By construction of P_1, w is a single terminal. So $A \to w$ is in P, i.e. $A \underset{G}{\Rightarrow} w$. Thus there is basis for induction.

Let us assume (6.7) for derivations in at most k steps. Let $A \overset{k+1}{\underset{G_1}{\Rightarrow}} w$. We can split this derivation as $A \underset{G_1}{\Rightarrow} A_1A_2 \ldots A_m \overset{k}{\underset{G_1}{\Rightarrow}} w_1 \cdot w_m = w$ such that $A_i \overset{*}{\underset{G_1}{\Rightarrow}} w_i$. Each A_i is either in V_N or a new variable, say C_{a_i}. When $A_i \in V_N$, $A_i \overset{*}{\underset{G_1}{\Rightarrow}} w_i$ is a derivation in at most k steps, and so by induction hypothesis, $A_i \overset{*}{\underset{G}{\Rightarrow}} w_i$. When $A_i = C_{a_i}$, the production $C_{a_i} \to a_i$ is applied to get $A_i \overset{*}{\Rightarrow} w_i$. The production $A \to A_1A_2 \ldots A_m$ is induced by a production $A \to X_1X_2 \ldots X_m$ in P where $X_i = A_i$ if $A_i \in V_N$ and $X_i = w_i$ if $A_i = C_{a_i}$. So $A \underset{G}{\Rightarrow} X_1X_2 \ldots X_m \overset{*}{\underset{G}{\Rightarrow}} w_1w_2 \ldots w_m$, i.e. $A \overset{*}{\underset{G}{\Rightarrow}} w$. Thus, (6.7) is true for all derivations. Therefore, $L(G) = L(G_1)$.

The effect of applying $A \to A_1A_2 \ldots A_m$ in a derivation for $w \in L(G_1)$ can be achieved by applying the productions $A \to A_1C_1$, $C_1 \to A_2C_2$, $\ldots$, $C_{m-2} \to A_{m-1}A_m$ in P_2. Hence it is easy to see that $L(G_1) \subseteq L(G_2)$.

To prove $L(G_2) \subseteq L(G_1)$, we can prove an auxiliary result

$$A \overset{*}{\underset{G_1}{\Rightarrow}} w \quad \text{if } A \in V'_N,\ A \overset{*}{\underset{G_2}{\Rightarrow}} w \tag{6.8}$$

Condition (6.8) can be proved by induction on the number of steps in $A \overset{*}{\underset{G_2}{\Rightarrow}} w$.

Applying (6.7) to S, we get $L(G_2) \subseteq L(G)$. Thus,

$$L(G) = L(G_1) = L(G_2) \quad \blacksquare$$

EXAMPLE 6.12

Find a grammar in Chomsky normal form equivalent to $S \to aAbB$, $A \to aA \mid a$, $B \to bB \mid b$.

Solution

As there are no unit productions or null productions, we need not carry out step 1. We proceed to step 2.

Step 2 Let $G_1 = (V'_N, \{a, b\}, P_1, S)$, where P_1 and V'_N are constructed as follows:

(i) $A \to a$, $B \to b$ are added to P_1.

(ii) $S \to aAbB$, $A \to aA$, $B \to bB$ yield $S \to C_aAC_bB$, $A \to C_aA$, $B \to C_bB$, $C_a \to a$, $C_b \to b$.

$$V'_N = \{S, A, B, C_a, C_b\}.$$

Step 3 P_1 consists of $S \to C_aAC_bB$, $A \to C_aA$, $B \to C_bB$, $C_a \to a$, $C_b \to b$, $A \to a$, $B \to b$.

$S \to C_aAC_bB$ is replaced by $S \to C_aC_1$, $C_1 \to AC_2$, $C_2 \to C_bB$

The remaining productions in P_1 are added to P_2. Let

$$G_2 = (\{S, A, B, C_a, C_b, C_1, C_2\}, \{a, b\}, P_2, S),$$

where P_2 consists of $S \to C_aC_1$, $C_1 \to AC_2$, $C_2 \to C_bB$, $A \to C_aA$, $B \to C_bB$, $C_a \to a$, $C_b \to b$, $A \to a$, and $B \to b$.

G_2 is in CNF and equivalent to the given grammar.

EXAMPLE 6.13

Find a grammar in CNF equivalent to the grammar

$$S \to \sim S \mid [S \supset S] \mid p \mid q \qquad (S \text{ being the only variable})$$

Solution

As the given grammar has no unit or null productions, we omit step 1 and proceed to step 2.

Step 2 Let $G_1 = (V'_N, \Sigma, P_1, S)$, where P_1 and V'_N are constructed as follows:

(i) $S \to p \mid q$ are added to P_1.

(ii) $S \to \sim S$ induces $S \to AS$ and $A \to \sim$.

(iii) $S \to [S \supset S]$ induces $S \to BSCSD$, $B \to [$, $C \to \supset$, $D \to]$

$$V'_N = \{S, A, B, C, D\}$$

Step 3 P_1 consists of $S \to p \mid q$, $S \to AS$, $A \to \sim$, $B \to [$, $C \to \supset$, $D \to]$, $S \to BSCSD$.

$S \to BSCSD$ is replaced by $S \to BC_1$, $C_1 \to SC_2$, $C_2 \to CC_3$, $C_3 \to SD$.

Let

$$G_2 = (\{S, A, B, C, D, C_1, C_2, C_3\}, \Sigma, P_2, S)$$

where P_2 consists of $S \to p \mid q \mid AS \mid BC_1$, $A \to \sim$, $B \to [$, $C \to \supset$, $D \to]$, $C_1 \to SC_2$, $C_2 \to CC_3$, $C_3 \to SD$. G_2 is in CNF and equivalent to the given grammar.

6.4.2 Greibach Normal Form

Greibach normal form (GNF) is another normal form quite useful in some proofs and constructions. A context-free grammar generating the set accepted by a pushdown automaton is in Greibach normal form as will be seen in Theorem 7.4.

Definition 6.12 A context-free grammar is in Greibach normal form if every production is of the form $A \to a\alpha$, where $\alpha \in V_N^*$ and $a \in \Sigma$ (α may be Λ), and $S \to \Lambda$ is in G if $\Lambda \in L(G)$. When $\Lambda \in L(G)$, we assume that S does not appear on the R.H.S. of any production. For example, G given by $S \to aAB \mid \Lambda$, $A \to bC$, $B \to b$, $C \to c$ is in GNF.

Note: A grammar in GNF is a natural generalisation of a regular grammar. In a regular grammar the productions are of the form $A \to a\alpha$, where $a \in \Sigma$ and $\alpha \in V_N \cup \{\Lambda\}$, i.e. $A \to a\alpha$, with αV_N^* and $|\alpha| \le 1$. So for a grammar in GNF or a regular grammar, we get a (single) terminal and a string of variables (possibly Λ) on application of a production (with the exception of $S \to \Lambda$).

The construction we give in this section depends mainly on the following two technical lemmas:

Lemma 6.1 Let $G = (V_N, \Sigma, P, S)$ be a CFG. Let $A \to B\gamma$ be an A-production in P. Let the B-productions be $B \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_s$. Define

$$P_1 = (P - \{A \to B\gamma\}) \cup \{A \to \beta_i\gamma \mid 1 \le i \le s\}.$$

Then, $G_1 = (V_N, \Sigma, P_1, S)$ is a context-free grammar equivalent to G.

Proof If we apply $A \to B\gamma$ in some derivation for $w \in L(G)$, we have to apply $B \to \beta_i$ for some i at a later step. So $A \overset{*}{\underset{G}{\Rightarrow}} B_i\gamma$. The effect of applying $A \to B\gamma$ and eliminating B in grammar G is the same as applying $A \to \beta_i\gamma$ for some i in grammar G_1. Hence $w \in L(G_1)$, i.e. $L(G) \subseteq L(G_1)$. Similarly, instead of applying $A \to \beta_i\gamma$, we can apply $A \to B\gamma$ and $B \to \beta_i$ to get $A \overset{*}{\underset{G}{\Rightarrow}} \beta_i\gamma$. This proves $L(G_1) \subseteq L(G)$. ∎

Note: Lemma 6.1 is useful for deleting a variable B appearing as the first symbol on the R.H.S. of some A-production, provided no B-production has B as the first symbol on R.H.S.

The construction given in Lemma 6.1 is simple. To eliminate B in $A \to B\gamma$, we simply replace B by the right-hand side of every B-production.

For example, using Lemma 6.1, we can replace $A \to Bab$ by $A \to aAab$, $A \to bBab$, $A \to aaab$, $A \to ABab$ when the B-productions are $B \to aA \mid bB \mid aa \mid AB$.

The lemma is useful to eliminate A from the R.H.S. of $A \to A\alpha$.

Lemma 6.2 Let $G = (V_N, \Sigma, P, S)$ be a context-free grammar. Let the set of A-productions be $A \to A\alpha_1 \mid \ldots A\alpha_r \mid \beta_1 \mid \ldots \mid \beta_s$ (β_i's do not start with A).

Let Z be a new variable. Let $G_1 = (V_N \cup \{Z\}, \Sigma, P_1, S)$, where P_1 is defined as follows:

(i) The set of A-productions in P_1 are $A \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_s$

$$A \to \beta_1 Z \mid \beta_2 Z \mid \ldots \mid \beta_s Z$$

(ii) The set of Z-productions in P_1 are $Z \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_r$

$$Z \to \alpha_1 Z \mid \alpha_2 Z \ldots \mid \alpha_r Z$$

(iii) The productions for the other variables are as in P. Then G_1 is a CFG and equivalent to G.

Proof To prove $L(G) \subseteq L(G)$, consider a leftmost derivation of w in G. The only productions in $P - P_1$ are $A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_r$. If $A \to A\alpha_{i_1}$, $A \to A\alpha_{i_2}$, $\ldots$, $A \to A\alpha_{i_k}$ are used, then $A \to \beta_j$ should be used at a later stage (to eliminate A). So we have $A \overset{*}{\underset{G_1}{\Rightarrow}} \beta_j A\alpha_{i_1} \ldots \alpha_{i_k}$ while deriving w in G. However,

$$A \underset{G_1}{\Rightarrow} \beta_j Z \underset{G_1}{\Rightarrow} \beta_j \alpha_{i_1} Z \ldots \underset{G_1}{\Rightarrow} \beta_j \alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_k}$$

i.e.

$$A \overset{*}{\underset{G_1}{\Rightarrow}} \beta_j A\alpha_{i_1} \ldots \alpha_{i_k}$$

Thus, A can be eliminated by using productions in G_1. Therefore, $w \in L(G_1)$.

To prove $L(G_1) \subseteq L(G)$, consider a leftmost derivation of w in G_1. The only productions in $P_1 - P$ are $A \to \beta_1 Z \mid \beta_2 Z \mid \ldots \mid \beta_s Z$, $Z \to \alpha_1 \mid \ldots \mid \alpha_r$, $Z \to \alpha_1 Z \mid \alpha_2 Z \mid \ldots \mid \alpha_r Z$. If the new variable Z appears in the course of the derivation of w, it is because of the application of $A \to \beta_j Z$ in some earlier step. Also, Z can be eliminated only by a production of the form $Z \to \alpha_j$ or $Z \to \alpha_j Z$ for some i and j in a later step. So we get $A \overset{*}{\underset{G_1}{\Rightarrow}} \beta_j \alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_k}$ in the course of the derivation of w. But, we know that $A \overset{*}{\underset{G}{\Rightarrow}} \beta_j \alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_k}$. Therefore, $w \in L(G)$. ∎

EXAMPLE 6.14

Apply Lemma 6.2 to the following A-productions in a context-free grammar G.

$$A \to aBD \mid bDB \mid c, \qquad A \to AB \mid AD$$

Solution

In this example, $\alpha_1 = B$, $\alpha_2 = D$, $\beta_1 = aBD$, $\beta_2 = bDB$, $\beta_3 = c$. So the new productions are:

(i) $A \to aBD \mid bDB \mid c, \quad A \to aBDZ \mid bDBZ \mid cZ$

(ii) $Z \to B, \; Z \to D, \quad Z \to BZ \mid DZ$

Theorem 6.9 (Reduction to Greibach normal form). Every context-free language L can be generated by a context-free grammar G in Greibach normal form.

Proof We prove the theorem when $\Lambda \notin L$ and then extend the construction to L having Λ.

Case 1 Construction of G (when $\Lambda \in L$):

Step 1 We eliminate null productions and then construct a grammar G in Chomsky normal form generating L. We rename the variables as $A_1, A_2, \ldots, A_n$ with $S = A_1$. We write G as $(\{A_1, A_2, \ldots, A_n\}, \Sigma, P, A_1)$.

Step 2 To get the productions in the form $A_i \to a\gamma$ or $A_i \to A_j\gamma$, where $j > i$, convert the A_i-productions ($i = 1, 2, \ldots, n-1$) to the form $A_i \to A_j\gamma$ such that $j > i$. Prove that such modification is possible by induction on i.

Consider A_1-productions. If we have some A_1-productions of the form $A_1 \to A_1\gamma$, then we can apply Lemma 6.2 to get rid of such productions. We get a new variable, say Z_1, and A_1-productions of the form $A_1 \to a$ or $A_1 \to A_j\gamma'$, where $j > 1$. Thus there is basis for induction.

Assume that we have modified A_1-productions, A_2-productions ... A_i-productions. Consider A_{i+1}-productions. Productions of the form $A_{i+1} \to a\gamma$ required no modification. Consider the first symbol (this will be a variable) on the R.H.S. of the remaining A_{i+1}-productions. Let t be the smallest index among the indices of such symbols (variables). If $t > i + 1$, there is nothing to prove. Otherwise, apply the induction hypothesis to A_t-productions for $t \le i$. So any A_t-production is of the form $A_t \to A_j\gamma$, where $j > t$ or $A_t \to a\gamma'$. Now we can apply Lemma 6.1 to A_{i+1}-production whose R.H.S. starts with A_t. The resulting A_{i+1}-productions are of the form $A_{i+1} \to A_j\gamma$, where $j > t$ (or $A_{i+1} \to a\gamma'$).

We repeat the above construction by finding t for the new set of A_{i+1}-productions. Ultimately, the A_{i+1}-productions are converted to the form $A_{i+1} \to A_j\gamma$; where $j \ge i + 1$ or $A_{i+1} \to a\gamma'$. Productions of the form $A_{i+1} \to A_{i+1}\gamma$ can be modified by using Lemma 6.2. Thus we have converted A_{i+1}-productions to the required form. By the principle of induction, the construction can be carried out for $i = 1, 2, \ldots, n$. Thus for $i = 1, 2, \ldots, n-1$, any A_i-production is of form $A_i \to A_j\gamma$, where $j > i$ or $A_i \to a\gamma'$. Any A_n-production is of the form $A_n \to A_n\gamma$ or $A_n \to a\gamma'$.

Step 3 Convert A_n-productions to the form $A_n \to a\gamma$. Here, the productions of the form $A_n \to A_n\gamma$ are eliminated using Lemma 6.2. The resulting A_n-productions are of the form $A_n \to a\gamma$.

Step 4 Modify the A_i-productions to the form $A_i \to a\gamma$ for $i = 1, 2, \ldots, n-1$. At the end of step 3, the A_n-productions are of the form $A_n \to a\gamma$. The A_{n-1}-productions are of the form $A_{n-1} \to a\gamma'$ or $A_{n-1} \to A_n\gamma$. By applying Lemma 6.1, we eliminate productions of the form $A_{n-1} \to A_n\gamma$. The resulting

A_{n-1}-productions are in the required form. We repeat the construction by considering $A_{n-2}, A_{n-3}, \ldots, A_1$.

Step 5 Modify Z_i-productions. Every time we apply Lemma 6.2, we get a new variable. (We take it as Z_i when we apply the Lemma for A_i-productions.) The Z_i-productions are of the form $Z_i \to \alpha Z_i$ or $Z_i \to \alpha$ (where α is obtained from $A_i \to A_i\alpha$), and hence of the form $Z_i \to a\gamma$ or $Z_i \to A_k\gamma$ for some k. At the end of step 4, the R.H.S. of any A_k-production starts with a terminal. So we can apply Lemma 6.1 to eliminate $Z_i \to A_k\gamma$. Thus at the end of step 5, we get an equivalent grammar G_1 in GNF.

It is easy to see that G_1 is in GNF. We start with G in CNG. In G any A-production is of the form $A \to a$ or $A \to AB$ or $A \to CD$. When we apply Lemma 6.1 or Lemma 6.2 in step 2, we get new productions of the form $A \to a\alpha$ or $A \to \beta$, where $\alpha \in V_N^*$ and $\beta \in V_N^+$ and $a \in \Sigma$. In steps 3–5, the productions are modified to the form $A \to a\alpha$ or $Z \to a'\alpha'$, where $a, a' \in \Sigma$ and $\alpha, \alpha' \in V_N^*$.

Case 2 Construction of G when $\Lambda \in L$:

By the previous construction we get $G' = (V'_N, \Sigma, P_1, S)$ in GNF such that $L(G') = L - \{\Lambda\}$. Define a new grammar G_1 as

$$G_1 = (V'_N \cup \{S'\}, \Sigma, P_1 \cup \{S' \to S, S' \to \Lambda\}, S')$$

$S' \to S$ can be eliminated by using Theorem 6.7. As S-productions are in the required form, S'-productions are also in the required form. So $L(G) = L(G_1)$ and G_1 is in GNF. ∎

Remark Although we convert the given grammar to CNF in the first step, it is not necessary to convert all the productions to the form required for CNF. In steps 2–5, we do not disturb the productions of the form $A \to a\alpha$, $a \in \Sigma$ and $\alpha \in V_N^*$. So such productions can be allowed in G (in step 1). If we apply Lemma 6.1 or 6.2 as in steps 2–5 to productions of the form $A \to \alpha$, where $\alpha \in V_N^*$ and $|\alpha| \ge 2$, the resulting productions at the end of step 5 are in the required form (for GNF). Hence we can allow productions of the form $A \to \alpha$, where $\alpha \in V_N^*$, and $|\alpha| \ge 2$.

Thus we can apply steps 2–5 to a grammar whose productions are either $A \to a\alpha$ where $a \in V_N^*$, or $A \to \alpha \in V_N^*$ where $|\alpha| \ge 2$. To reduce the productions to the form $A \to \alpha \in V_N^*$ where $|\alpha| \ge 2$, we can apply step 2 of Theorem 6.8.

EXAMPLE 6.15

Construct a grammar in Greibach normal form equivalent to the grammar $S \to AA \mid a$, $A \to SS \mid b$.

Solution

The given grammar is in CNF. S and A are renamed as A_1 and A_2, respectively. So the productions are $A_1 \to A_2A_2 \mid a$ and $A_2 \to A_1A_1 \mid b$. As the

given grammar has no null productions and is in CNF we need not carry out step 1. So we proceed to step 2.

Step 2 (i) A_1-productions are in the required form. They are $A_1 \to A_2A_2 \mid a$.

(ii) $A_2 \to b$ is in the required form. Apply Lemma 6.1 to $A_2 \to A_1A_1$. The resulting productions are $A_2 \to A_2A_2A_1$, $A_2 \to aA_1$. Thus the A_2-productions are

$$A_2 \to A_2A_2A_1, \quad A_2 \to aA_1, \quad A_2 \to b$$

Step 3 We have to apply Lemma 6.2 to A_2-productions as we have $A_2 \to A_2A_2A_1$. Let Z_2 be the new variable. The resulting productions are

$$A_2 \to aA_1, \quad A_2 \to b$$
$$A_2 \to aA_1Z_2, \quad A_2 \to bZ_2$$
$$Z_2 \to A_2A_1, \quad Z_2 \to A_2A_1Z_2$$

Step 4 (i) The A_2-productions are $A_2 \to aA_1 \mid b \mid aA_1Z_2 \mid bZ_2$.

(ii) Among the A_1-productions we retain $A_1 \to a$ and eliminate $A_1 \to A_2A_2$ using Lemma 6.1. The resulting productions are $A_1 \to aA_1A_2 \mid bA_2$, $A_1 \to aA_1Z_2A_2 \mid bZ_2A_2$. The set of all (modified) A_1-productions is

$$A_1 \to a \mid aA_1A_2 \mid bA_2 \mid aA_1Z_2A_2 \mid bZ_2A_2$$

Step 5 The Z_2-productions to be modified are $Z_2 \to A_2A_1$, $Z_2 \to A_2A_1Z_2$. We apply Lemma 6.1 and get

$$Z_2 \to aA_1A_1 \mid bA_1 \mid aA_1Z_2A_1 \mid bZ_2A_1$$
$$Z_2 \to aA_1A_1Z_2 \mid bA_1Z_2 \mid aA_1Z_2A_1Z_2 \mid bZ_2A_1Z_2$$

Hence the equivalent grammar is

$$G' = (\{A_1, A_2, Z_2\}, \{a, b\}, P_1, A_1)$$

where P_1 consists of

$$A_1 \to a \mid aA_1A_2 \mid bA_2 \mid aA_1Z_2A_1 \mid bZ_2A_2$$
$$A_2 \to aA_1 \mid b \mid aA_1Z_2 \mid bZ_2$$
$$Z_2 \to aA_1A_1 \mid bA_1 \mid aA_1Z_2A_1 \mid bZ_2A_1$$
$$Z_2 \to aA_1A_1Z_2 \mid bA_1Z_2 \mid aA_1Z_2A_1Z_2 \mid bZ_2A_1Z_2$$

EXAMPLE 6.16

Convert the grammar $S \to AB$, $A \to BS \mid b$, $B \to SA \mid a$ into GNF.

Solution

As the given grammar is in CNF, we can omit step 1 and proceed to step 2 after renaming S, A, B as A_1, A_2, A_3, respectively. The productions are $A_1 \to A_2A_3$, $A_2 \to A_3A_1 \mid b$, $A_3 \to A_1A_2 \mid a$.

Step 2 (i) The A_1-production $A_1 \to A_2A_3$ is in the required form.

(ii) The A_2-productions $A_2 \to A_3A_1 \mid b$ are in the required form.

(iii) $A_3 \to a$ is in the required form.

Apply Lemma 6.1 to $A_3 \to A_1A_2$. The resulting productions are $A_3 \to A_2A_3A_2$. Applying the lemma once again to $A_3 \to A_2A_3A_2$, we get

$$A_3 \to A_3A_1A_3A_2 \mid bA_3A_2.$$

Step 3 The A_3-productions are $A_3 \to a \mid bA_3A_2$ and $A_3 \to A_3A_1A_3A_2$. As we have $A_3 \to A_3A_1A_3A_2$, we have to apply Lemma 6.2 to A_3-productions. Let Z_3 be the new variable. The resulting productions are

$$A_3 \to a \mid bA_3A_2, \quad A_3 \to aZ_3 \mid bA_3A_2Z_3$$
$$Z_3 \to A_1A_3A_2, \quad Z_3 \to A_1A_3A_2Z_3$$

Step 4 (i) The A_3-productions are

$$A_3 \to a \mid bA_3A_2 \mid aZ_3 \mid bA_3A_2Z_3 \tag{6.9}$$

(ii) Among the A_2-productions, we retain $A_2 \to b$ and eliminate $A_2 \to A_3A_1$ using Lemma 6.1. The resulting productions are

$$A_2 \to aA_1 \mid bA_3A_2A_1 \mid aZ_3A_1 \mid bA_3A_2Z_3A_1$$

The modified A_2-productions are

$$A_2 \to b \mid aA_1 \mid bA_3A_2A_1 \mid aZ_3A_1 \mid bA_3A_2Z_3A_1 \tag{6.10}$$

(iii) We apply Lemma 6.1 to $A_1 \to A_2A_3$ to get

$$A_1 \to bA_3 \mid aA_1A_3 \mid bA_3A_2A_1A_3 \mid aZ_3A_1A_3 \mid bA_3A_2Z_3A_1A_3 \tag{6.11}$$

Step 5 The Z_3-productions to be modified are

$$Z_3 \to A_1A_3A_2 \mid A_1A_3A_2Z_3$$

We apply Lemma 6.1 and get

$$Z_3 \to bA_3A_3A_2 \mid bA_3A_2Z_3$$
$$Z_3 \to aA_1A_3A_3A_2 \mid aA_1A_3A_3A_2Z_3$$
$$Z_3 \to bA_3A_2A_1A_3A_3A_2 \mid bA_3A_2A_1A_3A_3A_2Z_3 \tag{6.12}$$
$$Z_3 \to aZ_3A_1A_3A_3A_2 \mid aZ_3A_1A_3A_3A_2Z_3$$
$$Z_3 \to bA_3A_2Z_3A_1A_3A_3A_2 \mid bA_3A_2Z_3A_1A_3A_3A_2Z_3$$

The required grammar in GNF is given by (6.9)–(6.12).

The following example uses the Remark appearing after Theorem 6.9. In this example we retain productions of the form $A \to a\alpha$ and replace the terminals only when they appear as the second or subsequent symbol on R.H.S. (Example 6.17 gives productions to generate arithmetic expressions involving a and operations like +, * and parentheses.)

EXAMPLE 6.17

Find a grammar in GNF equivalent to the grammar

$$E \to E + T | T,\ T \to T * F | F,\ F \to (E) | a$$

Solution

Step 1 We first eliminate unit productions. Hence

$$W_0(E) = \{E\}, \quad W_1(E) = \{E\} \cup \{T\} = \{E, T\}$$
$$W_2(E) = \{E, T\} \cup \{F\} = \{E, T, F\}$$

So,

$$W(E) = \{E, T, F\}$$

So,

$$W_0(T) = \{T\}, \quad W_1(T) = \{T\} \cup \{F\} = \{T, F\}$$

Thus,

$$W(T) = \{T, F\}$$
$$W_0(F) = \{F\}, \quad W_1(F) = \{F\} = W(F)$$

The equivalent grammar without unit productions is, therefore, $G_1 = (V_1, \Sigma, P_1, S)$, where P_1 consists of

(i) $E \to E + T | T * F | (E) | a$
(ii) $T \to T * F | (E) | a$, and
(iii) $F \to (E) | a$.

We apply step 2 of reduction to CNF. We introduce new variables A, B, C corresponding to $+$, $*$, $)$. The modified productions are

(i) $E \to EAT | TBF | (EC | a$
(ii) $T \to TBF | (EC | a$
(iii) $F \to (EC | a$
(iv) $A \to +,\ B \to *,\ C \to)$

The variables A, B, C, F, T and E are renamed as A_1, A_2, A_3, A_4, A_5, A_6. Then the productions become

$$A_1 \to +, \quad A_2 \to *, \quad A_3 \to), \quad A_4 \to (A_6A_3 | a \qquad (6.13)$$
$$A_5 \to A_5A_2A_4 | (A_6A_3 | a$$
$$A_6 \to A_6A_1A_5 | A_5A_2A_4 | (A_6A_3 | a$$

Step 2 We have to modify only the A_5- and A_6-productions. $A_5 \to A_5A_2A_4$ can be modified by using Lemma 6.2. The resulting productions are

$$A_5 \to (A_6A_3 | a, \quad A_5 \to (A_6A_3Z_5 | aZ_5 \qquad (6.14)$$
$$Z_5 \to A_2A_4 | A_2A_4Z_5$$

$A_6 \to A_5A_2A_4$ can be modified by using Lemma 6.1. The resulting productions are

$$A_6 \to (A_6A_3A_2A_4 | aA_2A_4 | (A_6A_3Z_5A_2A_4 | aZ_5A_2A_4$$
$$A_6 \to (A_6A_3 | a \text{ are in the proper form.}$$

Step 3 $A_6 \to A_6A_1A_5$ can be modified by using Lemma 6.2. The resulting productions give all the A_6-productions:

$$A_6 \to (A_6A_3A_2A_4 | aA_2A_4 | (A_6A_3Z_5A_2A_4$$
$$A_6 \to aZ_5A_2A_4 | (A_6A_3 | a \qquad (6.15)$$
$$A_6 \to (A_6A_3A_2A_4Z_6 | aA_2A_4Z_6 | (A_6A_3Z_5A_2A_4Z_6$$
$$A_6 \to aZ_5A_2A_4Z_6 | (A_6A_3Z_6 | aZ_6 \qquad (6.16)$$
$$A_6 \to A_1A_5 | A_1A_5Z_6$$

Step 4 The step is not necessary as A_i–productions for $i = 5, 4, 3, 2, 1$ are in the required form.

Step 5 The Z_5-productions are $Z_5 \to A_2A_4 | A_2A_4Z_5$. These can be modified as

$$Z_5 \to *A_4 | *A_4Z_5 \qquad (6.17)$$

The Z_6-productions are $Z_6 \to A_1A_5 | A_1A_5Z_6$. These can be modified as

$$Z_6 \to +A_5 | +A_5Z_6 \qquad (6.18)$$

The required grammar in GNF is given by (6.13)–(6.18).

6.5 PUMPING LEMMA FOR CONTEXT-FREE LANGUAGES

The pumping lemma for context-free languages gives a method of generating an infinite number of strings from a given sufficiently long string in a context-free language L. It is used to prove that certain languages are not context-free. The construction we make use of in proving pumping lemma yields some decesion algorithms regarding context-free languages.

Lemma 6.3 Let G be a context-free grammar in CNF and T be a derivation tree in G. If the length of the longest path in T is less than or equal to k, then the yield of T is of length less than or equal to 2^{k-1}.

Proof We prove the result by induction on k, the length of the longest path for all A-trees (Recall an A-tree is a derivation tree whose root has label A).

When the longest path in an A-tree is of length 1, the root has only one son whose label is a terminal (when the root has two sons, the labels are variables). So the yield is of length 1. Thus, there is basis for induction.

Assume the result for $k - 1$ ($k > 1$). Let T be an A-tree with a longest path of length less than or equal to k. As $k > 1$, the root of T has exactly two sons with labels A_1 and A_2. The two subtrees with the two sons as roots have the longest paths of length less than or equal to $k - 1$ (see Fig. 6.12).

If w_1 and w_2 are their yields, then by induction hypothesis, $|w_1| \le 2^{k-2}$, $|w_2| \le 2^{k-2}$. So the yield of $T = w_1w_2$, $|w_1w_2| \le 2^{k-2} + 2^{k-2} = 2^{k-1}$. By the principle of induction, the result is true for all A-trees, and hence for all derivation trees.

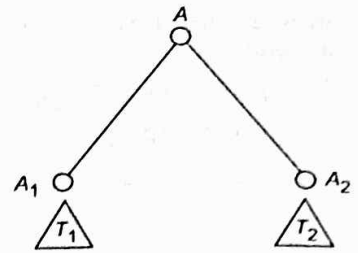

Fig. 6.12 Tree T with subtrees T_1 and T_2.

Theorem 6.10 (Pumping lemma for context-free languages). Let L be a context-free language. Then we can find a natural number n such that:

(i) Every $z \in L$ with $|z| \geq n$ can be written as $uvwxy$ for some strings u, v, w, x, y.
(ii) $|vx| \geq 1$.
(iii) $|vwx| \leq n$.
(iv) $uv^kwx^ky \in L$ for all $k \geq 0$.

Proof By Corollary 1 of Theorem 6.6, we can decide whether or not $\Lambda \in L$. When $\Lambda \in L$, we consider $L - \{\Lambda\}$ and construct a grammar $G = (V_N, \Sigma, P, S)$ in CNF generating $L - \{\Lambda\}$ (when $\Lambda \notin L$, we construct G in CNF generating L).

Let $|V_N| = m$ and $n = 2^m$. To prove that n is the required number, we start with $z \in L$, $|z| \geq 2^m$, and construct a derivation tree T (parse tree) of z. If the length of a longest path in T is at most m, by Lemma 6.3, $|z| \leq 2^{m-1}$ (since z is the yield of T). But $|z| \geq 2^m > 2^{m-1}$. So T has a path, say Γ, of length greater than or equal to $m + 1$. Γ has at least $m + 2$ vertices and only the last vertex is a leaf. Thus in Γ all the labels except the last one are variables. As $|V_N| = m$, some label is repeated.

We choose a repeated label as follows: We start with the leaf of Γ and travel along Γ upwards. We stop when some label, say B, is repeated. (Among several repeated labels, B is the first.) Let v_1 and v_2 be the vertices with label B, v_1 being nearer the root. In Γ, the portion of the path from v_1 to the leaf has only one label, namely B, which is repeated, and so its length is at most $m + 1$.

Let T_1 and T_2 be the subtrees with v_1, v_2 as roots and z_1, w as yields, respectively. As Γ is a longest path in T, the portion of Γ from v_1 to the leaf is a longest path in T_1 and of length at most $m + 1$. By Lemma 6.3, $|z_1| \leq 2^m$ (since z_1 is the yield of T_1).

For better understanding, we illustrate the construction for the grammar whose productions are $S \to AB$, $A \to aB \mid a$, $B \to bA \mid b$, as in Fig. 6.13. In the figure,

$$\Gamma = S \to A \to B \to A \to B \to b$$
$$z = ababb, \quad z_1 = bab, \quad w = b$$
$$v = ba, \quad x = \Lambda, \quad u = a, \quad y = b$$

As z and z_1 are the yields of T and a proper subtree T_1 of T, we can write $z = uz_1y$. As z_1 and w are the yields of T_1 and a proper subtree T_2 of T_1, we can write $z_1 = vwx$. Also, $|vwx| > |w|$. So, $|vx| \geq 1$. Thus, we have $z = uvwxy$ with $|vwx| \leq n$ and $|vx| \geq 1$. This proves the points (i)–(iii) of the theorem.

As T is an S-tree and T_1, T_2 are B-trees, we get $S \overset{*}{\Rightarrow} uBy$, $B \overset{*}{\Rightarrow} vBx$ and $B \overset{*}{\Rightarrow} w$. As $S \overset{*}{\Rightarrow} uBy \Rightarrow uwy$, $uv^0wx^0y \in L$. For $k \geq 1$, $S \overset{*}{\Rightarrow} uBy \overset{*}{\Rightarrow} uv^kBx^ky \overset{*}{\Rightarrow} uv^kwx^ky \in L$. This proves the point (iv) of the theorem. ∎

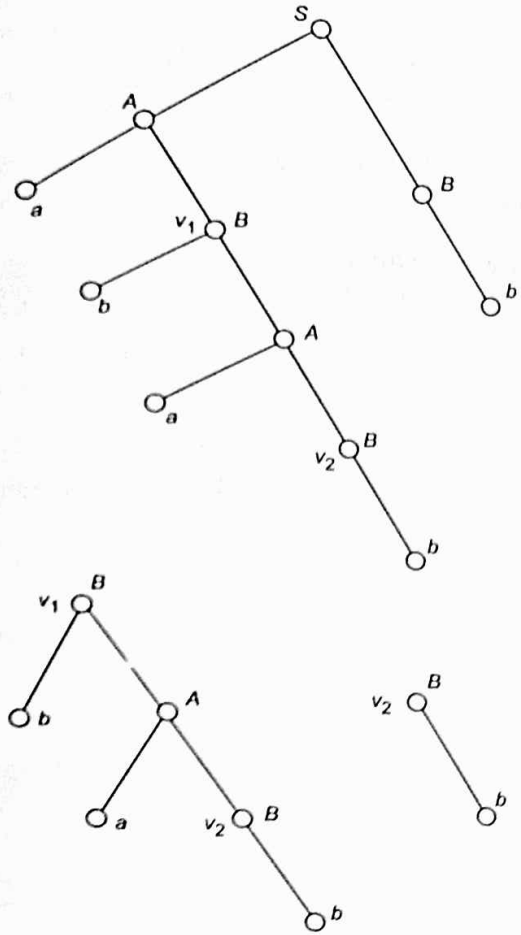

Fig. 6.13 Tree T and its subtrees T_1 and T_2.

Corollary Let L be a context-free language and n be the natural number obtained by using the pumping lemma. Then (i) $L \neq \emptyset$ if and only if there exists $w \in L$ with $|w| < n$, and (ii) L is infinite if and only if there exists $z \in L$ such that $n \leq |z| < 2n$.

Proof (i) We have to prove the 'only if' part. If $z \in L$ with $|z| \geq n$, we apply the pumping lemma to write $z = uvwxy$, where $1 \leq |vx| \leq n$. Also, $uwy \in L$ and $|uwy| < |z|$. Applying the pumping lemma repeatedly, we can get $z' \in L$ such that $|z'| < n$. Thus (i) is proved.

(ii) If $z \in L$ such that $n \leq |z| < 2n$, by pumping lemma we can write $z = uvwxy$. Also, $uv^k wx^k y \in L$ for all $k \geq 0$. Thus we get an infinite number of elements in L. Conversely, if L is infinite, we can find $z \in L$ with $|z| \geq n$. If $|z| < 2n$, there is nothing to prove. Otherwise, we can apply the pumping lemma to write $z = uvwxy$ and get $uwy \in L$. Every time we apply the pumping lemma we get a smaller string and the decrease in length is at most n (being equal to $|vx|$). So, we ultimately get a string z' in L such that $n \leq |z'| < 2n$. This proves (ii). ∎

Note: As the proof of the corollary depends only on the length of vx, we can apply the corollary to regular sets as well (refer to pumping lemma for regular sets).

The corollary given above provides us algorithms to test whether a given context-free language is empty or infinite. But these algorithms are not efficient. We shall give some other algorithms in Section 6.6.

We use the pumping lemma to show that a language L is not a context-free language. We assume that L is context-free. By applying the pumping lemma we get a contradiction.

The procedure can be carried out by using the following steps:

Step 1 Assume L is context-free. Let n be the natural number obtained by using the pumping lemma.

Step 2 Choose $z \in L$ so that $|z| \geq n$. Write $z = uvwxy$ using the pumping lemma.

Step 3 Find a suitable k so that $uv^k wx^k y \notin L$. This is a contradiction, and so L is not context-free.

EXAMPLE 6.18

Show that $L = \{a^n b^n c^n | n \geq 1\}$ is not context-free but context-sensitive.

Solution

We have already constructed a context-sensitive grammar G generating L (see Example 4.11). We note that in every string of L, any symbol appears the same number of times as any other symbol. Also a cannot appear after b, and c cannot appear before b, and so on.

Step 1 Assume L is context-free. Let n be the natural number obtained by using the pumping lemma.

Step 2 Let $z = a^n b^n c^n$. Then $|z| = 3n > n$. Write $z = uvwxy$, where $|vx| \geq 1$, i.e. at least one of v or x is not Λ.

Step 3 $uvwxy = a^n b^n c^n$. As $1 \leq |vx| \leq n$, v or x cannot contain all the three symbols a, b, c. So, (i) v or x is of the form $a^i b^j$ (or $b^i c^j$) for some i, j such that $i + j \leq n$. Or (ii) v or x is a string formed by the repetition of only one symbol among a, b, c.

When v or x is of the form $a^i b^j$, $v^2 = a^i b^j a^i b^j$ (or $x^2 = a^i b^j a^i b^j$). As v^2 is a substring of uv^2wx^2y, we cannot have uv^2wx^2y of the form $a^m b^m c^m$. So, $uv^2wx^2y \notin L$.

When both v and x are formed by the repetition of a single symbol (e.g. $u = a^i$ and $v = b^j$ for some i and j, $i \leq n$, $j \leq n$), the string uwy will contain the remaining symbol, say a_1. Also, a_1^n will be a substring of uwy as a_1 does not occur in v or x. The number of occurrences of one of the other two symbols in uwy is less than n (recall $uvwxy = a^n b^n c^n$), and n is the number of occurrences of a_1. So $uv^0wx^0y = uwy \notin L$.

Thus for any choice of v or x, we get a contradiction. Therefore, L is not context-free.

EXAMPLE 6.19

Show that $L = \{a^p | p \text{ is a prime}\}$ is not a context-free language.

Solution

We use the following property of L: If $w \in L$, then $|w|$ is a prime.

Step 1 Suppose $L = L(G)$ is context-free. Let n be the natural number obtained by using the pumping lemma.

Step 2 Let p be a prime number greater than n. Then $z = a^p \in L$. We write $z = uvwxy$.

Step 3 By pumping lemma, $uv^0wx^0y = uwy \in L$. So $|uwy|$ is a prime number, say q. Let $|vx| = r$. Then, $|uv^q wx^q y| = q + qr$. As $q + qr$ is not a prime, $uv^q wx^q y \notin L$. This is a contradiction. Therefore, L is not context-free.

6.6 DECISION ALGORITHMS FOR CONTEXT-FREE LANGUAGES

In this section we give some decision algorithms for context-free languages and regular sets.

(i) *Algorithm for deciding whether a context-free language L is empty.* We can apply the construction given in Theorem 6.3 for getting $V_N' = W_k$. L is nonempty if and only if $S \in W_k$.

(ii) *Algorithm for deciding whether a context-free language L is finite.* Construct a non-redundant context-free grammar G in CNF generating $L - \{\Lambda\}$. We draw a directed graph whose vertices are variables in G. If $A \to BC$ is a production, there are directed edges from A to B and A to C. L is finite if and only if the directed graph has no cycles.

(iii) *Algorithm for deciding whether a regular language L is empty.* Construct a deterministic finite automaton M accepting L. We construct the set of all states reachable from the initial state q_0. We find the states which are reachable from q_0 by applying a single input symbol. These states are arranged as a row under columns corresponding to every input symbol. The construction is repeated for every state appearing in an earlier row. The construction terminates in a finite number of steps. If a final state appears in this tabular column, then L is nonempty. (Actually, we can terminate the construction as soon as some final state is obtained in the tabular column.) Otherwise, L is empty.

(iv) *Algorithm for deciding whether a regular language L is infinite.* Construct a deterministic finite automaton M accepting L. L is infinite if and only if M has a cycle.

6.7 SUPPLEMENTARY EXAMPLES

EXAMPLE 6.20

Consider a context-free grammar G with the following productions.

$$S \to ASA \mid B$$
$$B \to aCb \mid bCa$$
$$C' \to ACA \mid A$$
$$A \to a \mid b$$

and answer the following questions:

(a) What are the variables and terminals of G?
(b) Give three strings of length 7 in $L(G)$.
(c) Are the following strings in $L(G)$?
  (i) aaa (ii) bbb (iii) aba (iv) abb
(d) True or false: $C \Rightarrow bab$
(e) True or false: $C \overset{*}{\Rightarrow} bab$
(f) True or false: $C \overset{*}{\Rightarrow} abab$
(g) True or false: $C \overset{*}{\Rightarrow} AAA$
(h) Is Λ in $L(G)$?

Solution

(a) $V_N = \{S, A, B, C\}$ and $\Sigma = \{a, b\}$
(b) $S \overset{*}{\Rightarrow} A^2SA^2 \Rightarrow A^2BA^2 \Rightarrow A^2aCbA^2 \Rightarrow A^2aAbA^2 \overset{*}{\Rightarrow} ababbab$

So $ababbab \in L(G)$.

$$S \overset{*}{\Rightarrow} A^2aAbA^2 \text{ (as in the derivation of the first string)}$$
$$\overset{*}{\Rightarrow} aaaabaa$$
$$S \overset{*}{\Rightarrow} A^2aAbA^2 \overset{*}{\Rightarrow} bbabbbb$$

So $ababbab$, $aaaabaa$, $bbabbbb$ are in $L(G)$.

(c) $S \to ASA$. If $B \overset{*}{\Rightarrow} w$, then w starts with a and ends in b or vice versa and $|w| \geq 3$. If aaa is in $L(G)$, then the first two steps in the derivation of aaa should be $S \Rightarrow ASA \Rightarrow ABA$ or $S \overset{*}{\Rightarrow} aBa$. The length of the terminal string thus derived is of length 5 or more. Hence $aaa \notin L(G)$. A similar argument shows that $bbb \notin L(G)$.

$$S \Rightarrow B \Rightarrow acb \Rightarrow aAb \Rightarrow abb. \text{ So } abb \in L(G)$$

(d) False, since the single-step derivations starting with C can only be $C \Rightarrow ACA$ or $C \Rightarrow A$.
(e) $C \Rightarrow ACA \Rightarrow AAA \overset{*}{\Rightarrow} bab$. True
(f) Let $w = abab$. If $C \overset{*}{\Rightarrow} w$, then $C \Rightarrow ACA \overset{*}{\Rightarrow} w$ or $C \Rightarrow A \Rightarrow w$. In the first case $|w| = 3, 5, 7, \ldots$. As $|w| = 4$, and $A \Rightarrow w$ if and only if $w = a$ or b, the second case does not arise. Hence (f) is false.
(g) $C \Rightarrow ACA \Rightarrow AAA$. Hence $C \overset{*}{\Rightarrow} AAA$ is true.
(h) $\Lambda \notin L(G)$.

EXAMPLE 6.21

If G consists of the productions $S \to aSa \mid bSb \mid aSb \mid bSa \mid \Lambda$, show that $L(G)$ is a regular set.

Solution

First of all, we show that $L(G)$ consists of the set L of all strings over $\{a, b\}$, of even length. It is easy to see that $L(G) \subseteq L$. Consider a string w of even length. Then, $w = a_1a_2 \ldots a_{2n-1}a_{2n}$ where each a_i is either a or b. Hence

$$S \Rightarrow a_1Sa_{2n} \Rightarrow a_1a_2Sa_{2n-1}a_{2n} \Rightarrow a_1a_2 \ldots a_nSa_{n+1} \ldots a_{2n} \Rightarrow a_1a_2 \ldots a_{2n}.$$

Hence $L \subseteq L(G)$.

Next we prove that $L = L(G_1)$ for some regular grammar G_1. Define $G_1 = (\{S, S_1, S_2, S_3, S_4\}, \{a, b\}, P, S)$ where P consists of $S \to aS_1$, $S_1 \to aS$, $S \to aS_2$, $S_2 \to bS$, $S \to bS_3$, $S_3 \to bS$, $S \to bS_4$, $S_4 \to aS$, $S \to \Lambda$.

Then $S \overset{*}{\Rightarrow} a_1a_2S$ where $a_1 = a$ or b and $a_2 = a$ or b. It is easy to see that $L(G_1) = L$. As G_1 is regular, $L(G) = L(G_1)$ is a regular set.

EXAMPLE 6.22

Reduce the following grammar to CNF:

$$S \to ASA \mid bA, \qquad A \to B \mid S, \qquad B \to c$$

Solution

Step 1 *Elimination of unit productions:*
The unit productions are $A \to B$, $A \to S$.

$$W_0(S) = \{S\},\ W_1(S) = \{S\} \cup \emptyset = \{S\}$$
$$W_0(A) = \{A\},\ W_1(A) = \{A\} \cup \{S, B\} = \{S, A, B\}$$
$$W_2(A) = \{S, A, B\} \cup \emptyset = \{S, A, B\}$$
$$W_0(B) = \{B\},\ W_1(B) = \{B\} \cup \emptyset = \{B\}$$

The productions for the equivalent grammar without unit productions are

$$S \to ASA \mid bA,\ B \to c$$
$$A \to ASA \mid bA,\ A \to c$$

So, $G_1 = (\{S, A, B\}, \{b, c\}, P, S)$ where P consists of $S \to ASA \mid bA$, $B \to c$, $A \to ASA \mid bA \mid c$.

Step 2 *Elimination of terminals in R.H.S.:*

$S \to ASA$, $B \to c$, $A \to ASA \mid c$ are in proper form. We have to modify $S \to bA$ and $A \to bA$.

Replace $S \to bA$ by $S \to C_bA$, $C_b \to b$ and $A \to bA$ by $A \to C_bA$, $C_b \to b$.

So, $G_2 = (\{S, A, B, C_b\}, \{b, c\}, P_2, S)$ where P_2 consists of

$$S \to ASA \mid C_bA$$
$$A \to ASA \mid c \mid C_bA$$
$$B \to c,\ C_b \to b$$

Step 3 *Restricting the number of variables on R.H.S.:*

$S \to ASA$ is replaced by $S \to AD$, $D \to SA$

$A \to ASA$ is replaced by $A \to AE$, $E \to SA$

So the equivalent grammar in CNF is

$$G_3 = (\{S, A, B, C_b, D, E\}, \{b, c\}, P_3, S)$$

where P_3 consists of

$$S \to C_bA \mid AD$$
$$A \to c \mid C_bA \mid AE$$
$$B \to c,\ C_b \to b,\ D \to SA,\ E \to SA$$

EXAMPLE 6.23

Let $G = (V_N, \Sigma, P, S)$ be a context-free grammar without null productions or unit productions and k be the maximum number of symbols on the R.H.S. of any production of G. Show that there exists an equivalent grammar G_1 in CNF, which has at most $(k - 1)|P| + |\Sigma|$ productions.

Solution

In step 2 (Theorem 6.8), a production of the form $A \to X_1X_2 \ldots X_n$ is replaced by $A \to Y_1Y_2, \ldots, Y_n$ where $Y_i = X_i$ if $X_i \in V_N$ and Y_i is a new variable if $X_i \in \Sigma$. We also add productions of the form $Y_i \to X_i$ whenever $X_i \in \Sigma$. As there are $|\Sigma|$ terminals, we have a maximum of $|\Sigma|$ productions of the form $Y_i \to X_i$ to be added to the new grammar. In step 3 (Theorem 6.8), $A \to A_1A_2 \ldots A_n$ is replaced by $n - 1$ productions, $A \to A_1D_1$, $D_1 \to A_2D_2 \ldots D_{n-2} \to A_{n-1}A_n$. Note that $n \le k$. So the total number of new productions obtained in step 3, is at most $(k - 1)|P|$. Thus the total number of productions in CNF is at most $(k - 1)|P| + |\Sigma|$.

Example 6.24

Reduce the following CFG to GNF:

$$S \to ABb|a, \qquad A \to aaA, \qquad B \to bAb$$

Solution

The valid productions for a grammar in GNF are $A \to a\alpha$, where $a \in \Sigma$, $\alpha \in V_N^*$.

So, $S \to ABb$ can be replaced by $S \to ABC$, $C \to b$.

$A \to aaA$ can be replaced by $A \to aDA$, $D \to a$.

$B \to bAb$ can be replaced by $B \to bAC$, $C \to b$.

So the revised productions are:

$$S \to ABC \mid a, \qquad A \to aDA, \qquad B \to bAC, \qquad C \to b, \qquad D \to a.$$

Name S, A, B, C, D as A_1, A_2, A_3, A_4, A_5.

Now we proceed to step 2.

Step 2 $G_1 = (\{A_1, A_2, A_3, A_4, A_5\}, \{a, b\}, P_1, A_1)$ where P_1 consists of

$$A_1 \to A_2A_3A_4 \mid a,\ A_2 \to aA_5A_2,\ A_3 \to bA_2A_4,\ A_4 \to b,\ A_5 \to a$$

The only production to be modified using step 4 (refer to Theorem 6.9) is $A_1 \to A_2A_3A_4$.

Replace $A_1 \to A_2A_3A_4$ by $A_1 \to aA_5A_2A_3A_4$.

The required grammar in GNF is

$G_2 = (\{A_1, A_2, A_3, A_4, A_5\}, \{a, b\}, P_2, A_1)$ where P_2 consists of

$$A_1 \to aA_5A_3A_4 \mid a$$
$$A_2 \to aA_5A_2$$
$$A_3 \to bA_2A_4, \qquad A_4 \to b, \qquad A_5 \to a$$

EXAMPLE 6.25

If a context-free grammar is defined by the productions

$$S \to a \mid Sa \mid bSS \mid SSb \mid SbS$$

show that every string in $L(G)$ has more a's than b's.

Proof We prove the result by induction on $|w|$, where $w \in L(G)$.

When $|w| = 1$, then $w = a$. So there is basis for induction.

Assume that $S \overset{*}{\Rightarrow} w$, $|w| < n$ implies that w has more a's than b's. Let $|w| = n > 1$. Then the first step in the derivation $S \overset{*}{\Rightarrow} w$ is $S \Rightarrow bSS$ or $S \Rightarrow SSb$ or $S \Rightarrow SbS$. In the first case, $S \Rightarrow bSS \overset{*}{\Rightarrow} bw_1w_2 = w$ for some $w_1, w_2 \in \Sigma^*$ and $S \overset{*}{\Rightarrow} w_1$, $S \overset{*}{\Rightarrow} w_2$. By induction hypothesis each of w_1 and w_2 has more a's than b's. So w_1w_2 has at least two more a's than b's. Hence bw_1w_2 has more a's than b's. The other two cases are similar. By the principle of induction, the result is true for all $w \in L(G)$.

EXAMPLE 6.26

Show that a CFG G with productions $S \to SS \mid (S) \mid \Lambda$ is ambiguous.

Solution

$$S \Rightarrow SS \Rightarrow S(S) \Rightarrow \Lambda(S) \Rightarrow \Lambda(\Lambda) = (\Lambda)$$

Also,

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow (\Lambda)S \Rightarrow (\Lambda)\Lambda = (\Lambda)$$

Hence G is ambiguous.

EXAMPLE 6.27

Is it possible for a regular grammar to be ambiguous?

Solution

Let $G = (V_N, \Sigma, P, S)$ be regular. Then every production is of the form $A \to aB$ or $A \to b$. Let $w \in L(G)$. Let $S \overset{*}{\Rightarrow} w$ be a leftmost derivation. We prove that any leftmost derivation $A \overset{*}{\Rightarrow} w$, for every $A \in V_N$ is unique by induction on $|w|$. If $|w| = 1$, then $w = a \in T$. The only production is $A \to a$. Hence there is basis for induction. Assume that any leftmost derivation of the form $A \overset{*}{\Rightarrow} w$ is unique when $|w| = n - 1$. Let $|w| = n$ and $A \overset{*}{\Rightarrow} w$ be a leftmost derivation.

Take $w = aw_1$, $a \in T$. Then the first step of $A \overset{*}{\Rightarrow} w$ has to be $A \Rightarrow aB$ for some $B \in V_N$. Hence the leftmost derivation $A \overset{*}{\Rightarrow} w$ can be split into $A \Rightarrow aB \overset{*}{\Rightarrow} aw_1$. So, we get a leftmost derivation $B \overset{*}{\Rightarrow} w_1$. By induction hypothesis, $B \overset{*}{\Rightarrow} w_1$ is unique. So, we get a unique leftmost derivation of w. Hence a regular grammar cannot be ambiguous.

SELF-TEST

1. Consider the grammer G which has the productions

$$A \to a \mid Aa \mid bAA \mid AAb \mid AbA$$

and answer the following questions:

(a) What is the start symbol of G?
(b) Is $aaabb$ in $L(G)$?
(c) Is $aaaabb$ in $L(G)$?
(d) Show that abb is not in $L(G)$.
(e) Write the labels of the nodes of the following derivation tree T which are not labelled. It is given that T is the derivation tree whose yield is in $\{a, b\}^*$.

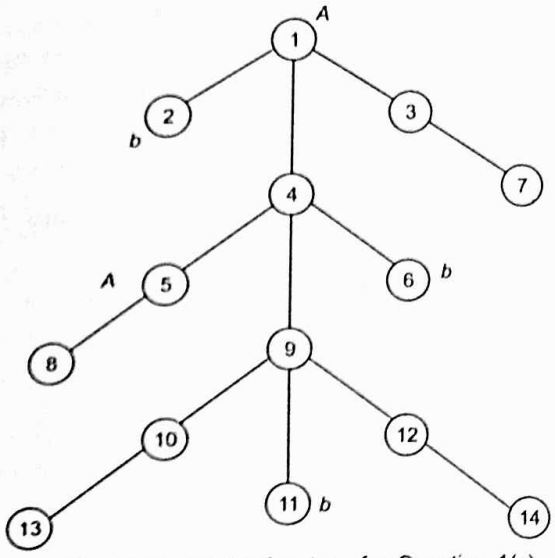

Fig. 6.14 Derivation tree for Question 1(e).

2. Consider the grammar G which has the following productions

$$S \to aB \mid bA,\ A \to aS \mid bAA \mid a,\ B \to bS \mid aBB \mid b.$$

and state whether the following statements are true or false.

(a) $L(G)$ is finite.
(b) $abbbaa \in L(G)$
(c) $aab \notin L(G)$
(d) $L(G)$ has some strings of odd length.
(e) $L(G)$ has some strings of even length.

3. State whether the following statements are true or false.
   (a) A regular language is context-free.
   (b) There exist context-free languages that are not regular.
   (c) The class of context-free languages is closed under union.
   (d) The class of context-free languages is closed under intersection.
   (e) The class of context-free languages is closed under complementation.
   (f) Every finite subset of $\{a, b\}^*$ is a context-free language.
   (g) $\{a^n b^n c^n \mid n \geq 1\}$ is a context-free language.
   (h) Any derivation tree for a regular grammar is a binary tree.

EXERCISES

6.1 Find a derivation tree of $a * b + a * b$ given that $a * b + a * b$ is in $L(G)$, where G is given by $S \to S + S \mid S * S$, $S \to a \mid b$.

6.2 A context-free grammar G has the following productions:

$$S \to 0S0 \mid 1S1 \mid A, \qquad A \to 2B3, \qquad B \to 2B3 \mid 3$$

Describe the language generated by the parameters.

6.3 A derivation tree of a sentential form of a grammar G is given in Fig. 6.15.

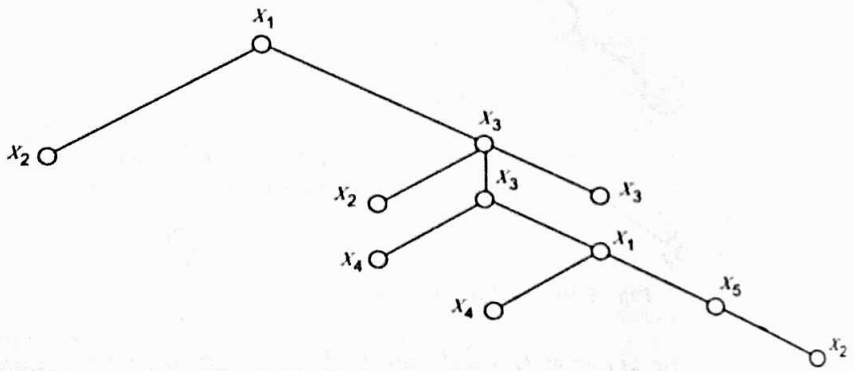

Fig. 6.15 Derivation tree for Exercise 6.3.

   (a) What symbols are necessarily in V_N?
   (b) What symbols are likely to be in Σ?
   (c) Determine if the following strings are sentential forms: (i) X_4X_2, (ii) $X_2X_2X_3X_2X_3X_3$, and (iii) $X_2X_4X_4X_2$.

6.4 Find (i) a leftmost derivation, (ii) a rightmost derivation, and (iii) a derivation which is neither leftmost nor rightmost of *abababa*, given that *abababa* is in $L(G)$, where G is the grammar given in Example 6.4.

6.5 Consider the following productions:

$$S \to aB \mid bA$$
$$A \to aS \mid bAA \mid a$$
$$B \to bS \mid aBB \mid b$$

For the string *aaabbabbba*, find
   (a) the leftmost derivation,
   (b) the rightmost derivation, and
   (c) the parse tree.

6.6 Show that the grammar $S \to a \mid abSb \mid aAb$, $A \to bS \mid aAAb$ is ambiguous.

6.7 Show that the grammar $S \to aB \mid ab$, $A \to aAB \mid a$, $B \to ABb \mid b$ is ambiguous.

6.8 Show that if we apply Theorem 6.4 first and then Theorem 6.3 to a grammar G, we may not get a reduced grammar.

6.9 Find a reduced grammar equivalent to the grammar $S \to aAa$, $A \to bBB$, $B \to ab$, $C \to aB$.

6.10 Given the grammar $S \to AB$, $A \to a$, $B \to C \mid b$, $C \to D$, $D \to E$, $E \to a$, find an equivalent grammar which is reduced and has no unit productions.

6.11 Show that for getting an equivalent grammar in the most simplified form, we have to eliminate unit productions first and then the redundant symbols.

6.12 Reduce the following grammars to Chomsky normal form:
   (a) $S \to 1A \mid 0B$, $A \to 1AA \mid 0S \mid 0$, $B \to 0BB \mid 1S \mid 1$
   (b) $G = (\{S\}, \{a, b, c\}, \{S \to a \mid b \mid cSS\}, S)$
   (c) $S \to abSb \mid a \mid aAb$, $A \to bS \mid aAAb$.

6.13 Reduce the grammars given in Exercises 6.1, 6.2, 6.6, 6.7, 6.9, 6.10 to Chomsky normal form.

6.14 Reduce the following grammars to Greibach normal form:
   (a) $S \to SS$, $S \to 0S1 \mid 01$
   (b) $S \to AB$, $A \to BSB$, $A \to BB$, $B \to aAb$, $B \to a$, $A \to b$
   (c) $S \to A0$, $A \to 0B$, $B \to A0$, $B \to 1$

6.15 Reduce the grammars given in Exercises 6.1, 6.2, 6.6, 6.7, 6.9, 6.10 to Greibach normal form.

6.16 Construct the grammars in Chomsky normal form generating the following:
   (a) $\{wcw^r \mid w \in 0\ \{a, b\}^*\}$,
   (b) the set of all strings over $\{a, b\}$ consisting of equal number of a's and b's,

(c) $\{a^m b^n \mid m \neq n,\ m,\ n \geq 1\}$, and
(d) $\{a^n b^m c^n \mid m,\ n \geq 1\}$.

6.17 Construct grammars in Greibach normal form generating the sets given in Exercise 6.16.

6.18 If $w \in L(G)$ and $|w| = k$, where G is in (i) Chomsky normal form, (ii) Greibach normal form, what can you say about the number of steps in the derivation of w?

6.19 Show that the language $\{a^{n^2} \mid n \geq 1\}$ is not context-free.

6.20 Show that the following are not context-free languages:
(a) The set of all strings over $\{a, b, c\}$ in which the number of occurrences of a, b, c is the same.
(b) $\{a^m b^m c^n \mid m \leq n \leq 2m\}$.
(c) $\{a^m b^n \mid n = m^2\}$.

6.21 A context-free grammar G is called a right-linear grammar if each production is of the form $A \to wB$ or $A \to w$, where A, B are variables and $w \in \Sigma^*$. (G is said to be left-linear if the productions are of the form $A \to Bw$ or $A \to w$. G is linear if the productions are of the form $A \to vBw$ or $A \to w$.) Prove the following:
(a) A right-linear or left-linear grammar is equivalent to a regular grammar.
(b) A linear grammar is not necessarily equivalent to a regular grammar.

6.22 A context-free grammar G is said to be self-embedding if there exists some useful variable A such that $A \overset{*}{\Rightarrow} uAv$, where $u, v \in \Sigma^*$, u, $v \neq \Lambda$. Show that a context-free language is regular iff it is generated by a nonselfembedding grammar.

6.23 Show that every context-free language without Λ is generated by a context-free grammar in which all productions are of the form $A \to a$, $A \to a\alpha b$.