

As  $|wx| > 0$ ,  $y^i w^j x^i z$  cannot equal  $y^j w^i x^j z$  if  $i \neq j$ . Thus the grammar generates an infinite number of strings.

Conversely, suppose the graph has no cycles. Define the rank of a variable  $A$  to be the length of the longest path in the graph beginning at  $A$ . The absence of cycles implies that the rank of  $A$  is finite. We also observe that if  $A \rightarrow BC$  is a production, then the rank of  $B$  and  $C$  must be strictly less than the rank of  $A$ , because for every path from  $B$  or  $C$ , there is a path of length one greater from  $A$ . We show by induction on  $r$  that if  $A$  has rank  $r$ , then no terminal string derived from  $A$  has length greater than  $2^r$ .

**Basis**  $r = 0$ . If  $A$  has rank 0, then its vertex has no edges out. Therefore all  $A$ -productions have terminals on the right, and  $A$  derives only strings of length 1.

**Induction**  $r > 0$ . If we use a production of the form  $A \rightarrow a$ , we may derive only a string of length 1. If we begin with  $A \rightarrow BC$ , then as  $B$  and  $C$  are of rank  $r - 1$  or less, by the inductive hypothesis, they derive only strings of length  $2^{r-1}$  or less. Thus  $BC$  cannot derive a string of length greater than  $2^r$ .

Since  $S$  is of finite rank  $r_0$ , and in fact, is of rank no greater than the number of variables,  $S$  derives strings of length no greater than  $2^{r_0}$ . Thus the language is finite. □

**Example 6.6** Consider the grammar

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow BC|a \\ B &\rightarrow CC|b \\ C &\rightarrow a \end{aligned}$$

whose graph is shown in Fig. 6.7(a). This graph has no cycles. The ranks of  $S, A, B$ , and  $C$  are 3, 2, 1, and 0, respectively. For example, the longest path from  $S$  is  $S, A, B, C$ . Thus this grammar derives no string of length greater than  $2^3 = 8$  and therefore generates a finite language. In fact, a longest string generated from  $S$  is

$$S \Rightarrow AB \Rightarrow BCB \Rightarrow CCCB \Rightarrow CCCC \not\Rightarrow aaaaa.$$

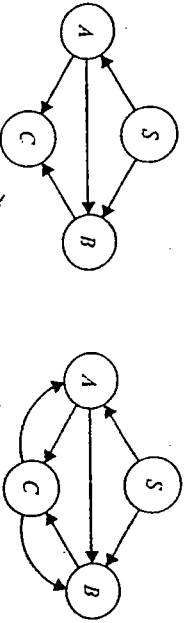


Fig. 6.7 Graphs corresponding to CNF grammars.

If we add production  $C \rightarrow AB$ , we get the graph of Fig. 6.7(b). This new graph has several cycles, such as  $A, B, C, A$ . Thus we can find a derivation  $A \xRightarrow{*} a_3 A b_3$ , in particular  $A \xRightarrow{*} BC \xRightarrow{*} CCC \xRightarrow{*} CAB C$ , where  $a_3 = C$  and  $b_3 = BC$ . Since  $C \xRightarrow{*} a$  and  $BC \xRightarrow{*} ba$ , we have  $A \xRightarrow{*} a^3 b a$ . Then as  $S \xRightarrow{*} Ab$  and  $A \xRightarrow{*} a$ , we now have  $S \xRightarrow{*} a^i b (ba)^j b$  for every  $i$ . Thus the language is infinite.

**Membership**

Another question we may answer is: Given a CFG  $G = (V, T, P, S)$  and string  $x$  in  $T^*$ , is  $x$  in  $L(G)$ ? A simple but inefficient algorithm to do so is to convert  $G$  to  $G' = (V', T, P', S)$ , a grammar in Greibach normal form generating  $L(G) - \{\epsilon\}$ . Since the algorithm of Theorem 4.3 tests whether  $S \xRightarrow{*} \epsilon$ , we need not concern ourselves with the case  $x = \epsilon$ . Thus assume  $x \neq \epsilon$ , so  $x$  is in  $L(G')$  if and only if  $x$  is in  $L(G)$ . Now, as every production of a GNF grammar adds exactly one terminal to the string being generated, we know that if  $x$  has a derivation in  $G'$ , it has one with exactly  $|x|$  steps. If no variable of  $G'$  has more than  $k$  productions, then there are at most  $k^{|x|}$  leftmost derivations of strings of length  $|x|$ . We may try them all systematically.

However, the above algorithm can take time which is exponential in  $|x|$ . There are several algorithms known that take time proportional to the cube of  $|x|$  or even a little less. The bibliographic notes discuss some of these. We shall here present a simple cubic time algorithm known as the Cocke-Younger-Kasami or CYK algorithm. It is based on the dynamic programming technique discussed in the solution to Exercise 3.23. Given  $x$  of length  $n \geq 1$ , and a grammar  $G$ , which we may assume is in Chomsky normal form, determine for each  $i$  and  $j$  and for each variable  $A$ , whether  $A \xRightarrow{*} x_{ij}$ , where  $x_{ij}$  is the substring of  $x$  of length  $j$  beginning at position  $i$ .

We proceed by induction on  $j$ . For  $j = 1$ ,  $A \xRightarrow{*} x_{ij}$  if and only if  $A \rightarrow x_{ij}$  is a production, since  $x_{ij}$  is a string of length 1. Proceeding to higher values of  $j$ , if  $j > 1$ , then  $A \xRightarrow{*} x_{ij}$  if and only if there is some production  $A \rightarrow BC$  and some  $k$ ,  $1 \leq k < j$ , such that  $B$  derives the first  $k$  symbols of  $x_{ij}$  and  $C$  derives the last  $j - k$  symbols of  $x_{ij}$ . That is,  $B \xRightarrow{*} x_{ik}$  and  $C \xRightarrow{*} x_{i+k, j-k}$ . Since  $k$  and  $j - k$  are both less than  $j$ , we already know whether each of the last two derivations exists. We may thus determine whether  $A \xRightarrow{*} x_{ij}$ . Finally, when we reach  $j = n$ , we may determine whether  $S \xRightarrow{*} x_{1n}$ . But  $x_{1n} = x$ , so  $x$  is in  $L(G)$  if and only if  $S \xRightarrow{*} x_{1n}$ .

To state the CYK algorithm precisely, let  $V_{ij}$  be the set of variables  $A$  such that  $A \xRightarrow{*} x_{ij}$ . Note that we may assume  $1 \leq i \leq n - j + 1$ , for there is no string of length greater than  $n - i + 1$  beginning at position  $i$ . Then Fig. 6.8 gives the CYK algorithm formally.

Steps (1) and (2) handle the case  $j = 1$ . As the grammar  $G$  is fixed, step (2) takes a constant amount of time. Thus steps (1) and (2) take  $O(n)$  time. The nested for-loops of lines (3) and (4) cause steps (5) through (7) to be executed at most  $n^2$  times, since  $i$  and  $j$  range in their respective for-loops between limits that are at

