As $|wx| > 0$, yw^ivx^iz cannot equal yw^jvx^jz 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 $A \to BC$ 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 $A \to a$, we may derive only a string of length 1. If we begin with $A \to BC$, 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.

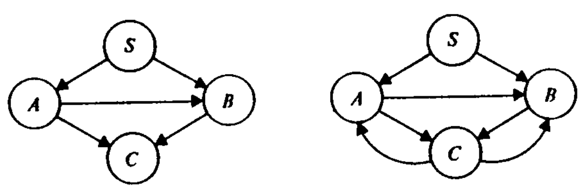
Since S is of finite rank r_0, and in fact, is of rank no greater than the number of variables, S derives strings of length no greater than 2^{r_0}. Thus the language is finite. □

Example 6.6 Consider the grammar

$$S \to AB$$
$$A \to BC \mid a$$
$$B \to CC \mid b$$
$$C \to a$$

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

$$S \Rightarrow AB \Rightarrow BCB \Rightarrow CCCB \Rightarrow CCCCC \overset{*}{\Rightarrow} aaaaa.$$



(a) (b)

Fig. 6.7 Graphs corresponding to CNF grammars.

If we add production $C \to AB$, 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 $A \overset{*}{\Rightarrow} \alpha_3 A \beta_3$, in particular $A \Rightarrow BC \Rightarrow CCC \Rightarrow CABC$, where $\alpha_3 = C$ and $\beta_3 = BC$. Since $C \overset{*}{\Rightarrow} a$ and $BC \overset{*}{\Rightarrow} ba$, we have $A \overset{*}{\Rightarrow} aAba$. Then as $S \overset{*}{\Rightarrow} Ab$ and $A \overset{*}{\Rightarrow} a$, we now have $S \overset{*}{\Rightarrow} a^i a (ba)^i b$ 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 $S \overset{*}{\Rightarrow} \epsilon$, 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 $A \overset{*}{\Rightarrow} x_{ij}$, where x_{ij} is the substring of x of length j beginning at position i.

We proceed by induction on j. For $j = 1$, $A \overset{*}{\Rightarrow} x_{ij}$ if and only if $A \to x_{ij}$ is a production, since x_{ij} is a string of length 1. Proceeding to higher values of j, if $j > 1$, then $A \overset{*}{\Rightarrow} x_{ij}$ if and only if there is some production $A \to BC$ 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, $B \overset{*}{\Rightarrow} x_{ik}$ and $C \overset{*}{\Rightarrow} x_{i+k,j-k}$. 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 $A \overset{*}{\Rightarrow} x_{ij}$. Finally, when we reach $j = n$, we may determine whether $S \overset{*}{\Rightarrow} x_{1n}$. But $x_{1n} = x$, so x is in $L(G)$ if and only if $S \overset{*}{\Rightarrow} x_{1n}$.

To state the CYK algorithm precisely, let V_{ij} be the set of variables A such that $A \overset{*}{\Rightarrow} x_{ij}$. 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

```
    begin
1)     for i:= 1 to n do
2)        V_i1:= {A | A → a is a production and the ith symbol of x is a};
3)     for j:= 2 to n do
4)        for i:= 1 to n − j + 1 do
             begin
5)              V_ij:= ∅;
6)              for k:= 1 to j − 1 do
7)                 V_ij:= V_ij ∪ {A | A → BC is a production, B is in V_ik and C
                      is in V_i+k,j−k}
             end
    end
```

Fig. 6.8. The CYK algorithm.

most n apart. Step (5) takes constant time at each execution, so the aggregate time spent at step (5) is $0(n^2)$. The for-loop of line (6) causes step (7) to be executed n or fewer times. Since step (7) takes constant time, steps (6) and (7) together take $0(n)$ time. As they are executed $0(n^2)$ times, the total time spent in step (7) is $0(n^3)$. Thus the entire algorithm is $0(n^3)$.

Example 6.7 Consider the CFG

$$S \to AB \mid BC$$
$$A \to BA \mid a$$
$$B \to CC \mid b$$
$$C \to AB \mid a$$

and the input string *baaba*. The table of V_{ij}'s is shown in Fig. 6.9. The top row is filled in by steps (1) and (2) of the algorithm in Fig. 6.8. That is, for positions 1 and 4, which are b, we set $V_{11} = V_{41} = \{B\}$, since B is the only variable which derives b.

	b			b	
			3	4	
1	B	A, C	A, C	B	A, C
2	S, A	B	S, C	S, A	
3	$\varnothing$	B	B		
4	$\varnothing$	S, A, C			
5	S, A, C				

Fig. 6.9 Table of V_{ij}'s

Similarly, $V_{21} = V_{31} = V_{51} = \{A, C\}$, since only A and C have productions with a on the right.

To compute V_{ij} for $j > 1$, we must execute the for-loop of steps (6) and (7). We must match V_{ik} against $V_{i+k,j-k}$ for $k = 1, 2, \ldots, j - 1$, seeking variable D in V_{ik} and E in $V_{i+k,j-k}$ such that DE is the right side of one or more productions. The left sides of these productions are adjoined to V_{ij}. The pattern in the table which corresponds to visiting V_{ik} and $V_{i+k,j-k}$ for $k = 1, 2, \ldots, j - 1$ in turn is to simultaneously move down column i and up the diagonal extending from V_{ij} to the right, as shown in Fig. 6.10.

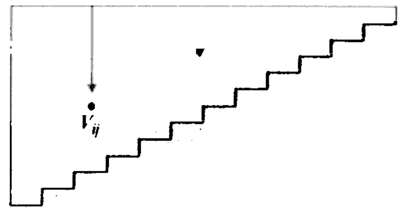


Fig. 6.10 Traversal pattern for computation of V_{ij}.

For example, let us compute V_{24}, assuming that the top three rows of Fig. 6.9 are filled in. We begin by looking at $V_{21} = \{A, C\}$ and $V_{33} = \{B\}$. The possible right-hand sides in $V_{21}V_{33}$ are AB and CB. Only the first of these is actually a right side, and it is a right side of two productions $S \to AB$ and $C \to AB$. Hence we add S and C to V_{24}. Next we consider $V_{22}V_{42} = \{B\}\{S, A\} = \{BS, BA\}$. Only BA is a right side, so we add the corresponding left side A to V_{24}. Finally, we consider $V_{23}V_{51} = \{B\}\{A, C\} = \{BA, BC\}$. BA and BC are each right sides, with left sides A and S, respectively. These are already in V_{24}, so we have $V_{24} = \{S, A, C\}$. Since S is a member of V_{15}, the string *baaba* is in the language generated by the grammar.

EXERCISES

6.1 Show that the following are not context-free languages.

a) $\{a^ib^jc^k \mid i < j < k\}$
b) $\{a^ib^j \mid j = i^2\}$
c) $\{a^i \mid i \text{ is a prime}\}$
d) the set of strings of a's, b's, and c's with an equal number of e
e) $\{a^nb^nc^m \mid n \le m \le 2n\}$

* **6.2** Which of the following are CFL's?

a) $\{a^ib^j \mid i \ne j \text{ and } i \ne 2j\}$
b) $(\mathbf{a} + \mathbf{b})^* - \{(a^nb^n)^r \mid n \ge 1\}$
c) $\{ww^Rw \mid w \text{ is in } (\mathbf{a} + \mathbf{b})^*\}$
d) $\{b_i \# b_{i+1} \mid b_i \text{ is } i \text{ in binary}, i \ge 1\}$