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)-{ε}.Since the algorithm of Theorem 4.3 tests whether $S \xrightarrow{*} \varepsilon$, we need not concern ourselves with the case $x=\varepsilon$. Thus assume $x\neq\varepsilon$, 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 $1\leq i<n-j+1$, 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 most n apart. Step (5) takes constant time at each execution, so the aggregate time spent at step (5) is $O(n^2)$. 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 O(n) time. As they are executed $O(n^2)$ times, the total time spent in step (7) is $O(n^3)$. Thus the entire algorithm is $O(n^3)$.

```
        Begin
1)          for i := 1 to n do
2)              Vi1 := {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)                  Vij = ∅;
6)                  for k := 1 to j-1 do
7)                      Vij := Vij ∪ {A|A→BC is a production, B is in Vik and
                                      C is in Vi+k,j-k }
                end
        end
```

Fig. 6.8. The CYK algorithm.

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

	b	a	a	b	a
j ↓ \ i →	1	2	3	4	5
1	B	A,C	A.C	B	A,C
2	S,A	B	S,C	S,A	
3	∅	B	B		
4	∅	S,A,C			
5	S,A,C				

Fig. 6.9. Table of V_{ij}'s.

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, …, j-1, seeding 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, …, 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.

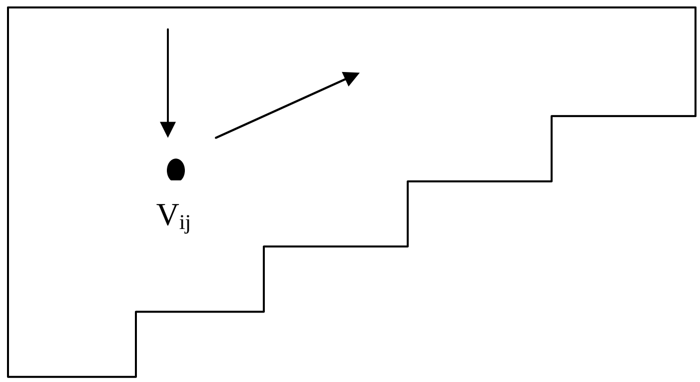

Fig. 6.10. Traversal pattern for computation of V_{ij}.

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 $\rightarrow$ AB and C $\rightarrow$ 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.

The above is from "*Introduction to Automata Theory, Languages, and Computation*" by John E. Hopcroft and Jeffrey D. Ullman, First Edition, Addison Wesley.