The G' obtained in this way using Theorem 2.14 is in Chomsky normal form and is equivalent to all the above grammars, including G_0. This completes the algorithm to find a Chomsky normal form cfg equivalent to G_0.

To make the grammar easier to read, we will go one step further and simplify the notation. In order to get a more readable grammar, start with this last G' and replace $<AS>$ by X, replace S by Y, and then replace S_2 by S. Call the grammar so obtained G_c. G_c can be described as follows:

start symbol $= S$
productions: $S \to \Lambda$, $A \to a$, $B \to b$, $Y \to XB$, $X \to AY$,
$S \to XB$, $Y \to AB$, $S \to AB$

If we now group together those productions of G_c that have the same symbol on the left-hand side, we get the following, more compact description of G_c:

start symbol $= S$
productions: $S \to \Lambda \mid AB \mid XB$
$X \to AY$
$Y \to AB \mid XB$
$A \to a$, $B \to b$

So G_c is exactly the cfg of Example 2.10.2, which we have already seen to be in Chomsky normal form and equivalent to G_0, the grammar of Example 2.2.1.

This example worked out particularly well. In general, the Chomsky normal form grammar produced by this algorithm need not be so clean. It can turn out to have many more productions than are needed and can even have productions that are useless. The grammar produced will always be in Chomsky normal form though and will always be equivalent to the grammar we started out with. If a simple grammar is desired, it is often necessary to perform some additional transformations on the grammar obtained by simply applying the algorithm.

The next result shows that a cfg description of a language can be converted to an effective procedure for testing strings to see whether or not they are in the language.

2.16 Theorem If G is a cfg, then there is an algorithm that can tell for any string w whether or not w is in $L(G)$.

PROOF This is left as an exercise. *Hint*: First find a Chomsky normal form grammar G' equivalent to G. Then compute a bound on the size of the parse tree for a string of length n. Use this to show that for any length n, we can compute a finite number of derivations such that if w has length n and w is in $L(G')$, then a derivation of w is in this set. □

THE PUMPING LEMMA

Our next result provides a method whereby, given a suitably long string in a cfl, we can produce infinitely many other strings which are also in the language.

This does not mean that all cfl's are infinite. If the cfl is finite, then there will not be any suitably long strings. This result will be useful in a number of contexts. One of its most common applications is to use it to show that certain languages are not cfl's.

2.17 Theorem (Pumping Lemma for CFL's). For each context-free language L, we can find an integer k, depending only on L, and such that the following holds: If z is in L and length$(z) > k$, then z may be written as $z = uvwxy$ where

1. length $(vwx) \leqslant k$
2. at least one of v and x is nonempty, and
3. $uv^i wx^i y$ is in L for all $i \geqslant 0$

PROOF Since L is a context-free language, there is a context-free grammar G such that $L = L(G)$. By Theorem 2.14, we may assume that G is in Chomsky normal form. First note that, since G is in Chomsky normal form, we can derive a relationship between the length of a string z in $L(G) = L$ and the longest path from S to a terminal symbol in a parse tree for z. Specifically,

Claim 1a: If the longest path contains at most m nodes ($m-1$ arcs), then z has length at most 2^{m-2}.

To see this, note that the longest string is obtained if all paths are as long as the longest path. In that case, we start with one node; that node produces two nodes, each of those produce two nodes, and so forth. After $m-2$ iterations of this doubling, we have produced a tree with 2^{m-2} nodes at the growing ends. To get a terminal string, each of these 2^{m-2} nonterminal nodes must produce a terminal node. So, after $m-1$ arcs we can, in this way, produce a terminal string of length 2^{m-2}. Clearly, this is the longest string we can produce under the constraint that no path has more than $m-1$ arcs. Another way of stating this relationship is as follows:

Claim 1b: If the length of a terminal string z is greater than 2^n, then some path in the parse tree must contain at least one more than $n+2$ nodes.

Let $k = 2^n$, where n is the number of nonterminals of G and suppose length$(z) > k$. Consider a fixed parse tree for z. By Claim 1b, some path, from S to a terminal, has $n+2$ or more nodes. Consider the longest such path. Going from S to a terminal along this path, we pass through all the nodes of this path. Consider the last $n+2$ such nodes. One is labeled with a terminal and $n+1$ are labeled with nonterminals. There are only n nonterminals; therefore, two such nodes must be labeled with the same nonterminal. Call this nonterminal A. The situation is diagrammed in Figure 2.5(a). One way to write a derivation corresponding to this tree is

4. $S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uvAxy \overset{*}{\Rightarrow} uvwxy = z$

where the two A's shown are the ones under consideration. Consider the subtree for $A \overset{*}{\Rightarrow} vAx \overset{*}{\Rightarrow} vwx$. Since the first A was chosen to be among the last $n+2$ nodes of the longest path, it follows that no path from the first A to a terminal has length greater than $n+2$. So, by Claim 1a, the string derived, namely vwx, has length at most 2^n. So we have

5. $\text{length}(vwx) \leqslant 2^n = k$

Now, by 4, it follows that for all $i \geqslant 0$,

6. $S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uv^iAx^iy \overset{*}{\Rightarrow} uv^iwx^iy$

This is diagrammed in Figure 2.5(b), for the case of $i = 2$.

Finally, it remains only to show that either x or v is nonempty. Again, consider the subtree $A \overset{*}{\Rightarrow} vAx \overset{*}{\Rightarrow} vwx$. Since the grammar is in Chomsky normal form, it can be written as $A \Rightarrow BC \overset{*}{\Rightarrow} vAx \overset{*}{\Rightarrow} vwx = z_1z_2$, where B and C are nonterminals such that $B \overset{*}{\Rightarrow} z_1$ and $C \overset{*}{\Rightarrow} z_2$. Since all Chomsky normal form grammars are nonerasing, it follows that both z_1 and z_2 are nonempty. Now the second A must be derived from either the B or the C. If it is derived from the B, then z_2 lies completely to the right of w. So z_2 is a substring of x and so x is nonempty. This situation is diagrammed in Figure 2.5(a). If A is derived from C, then a similar argument shows that, in this case, v is nonempty. In any case, either x or v is nonempty and the proof is complete. □

We conclude this section with two applications of the pumping lemma. When applying the pumping lemma, it is important to remember that, while we know for certain that any z satisfying the hypothesis of the pumping lemma can be written as $z = uvwxy$ and that this decomposition has the properties listed in the pumping lemma, we do not get to choose how z is divided into u, v, w, x, and y. We only know that there is at least one such decomposition. It is also important to note that we can "pump down" as well as "pump up." In the notation of the pumping lemma, we can take $i = 0$ and conclude that uwy is in L. So we can use the pumping lemma to produce either longer or shorter strings which are in the given cfl.

2.18 Theorem $L = \{a^nb^nc^n \mid n \geqslant 0\}$ is not a context-free language.

PROOF Suppose L were a cfl. We will derive a contradiction. Let k be as in the pumping lemma for cfl's. The string $a^kb^kc^k$ is in L. By the pumping lemma $a^kb^kc^k = uvwxy$, where

1. uv^2wx^2y is in L and
2. either v or x (or both) are nonempty

First note that v must be either all a's, all b's, or all c's. Otherwise, uv^2wx^2y would not have only a's followed by b's followed by c's, and so would not be in L. Similarly, x must be all a's, all b's, or all c's. Say v is all a's and x is all b's (the other cases are similar). If v is nonempty, then uv^2wx^2y contains more a's than c's. If x is non-empty, then it contains more b's than c's. In

any case, we conclude that uv^2wx^2y is not of the form $a^n b^n c^n$, and so we have contradicted (1). □

2.19 Theorem There is an algorithm to determine if a given context-free grammar generates a finite or infinite number of words.

PROOF Let G be a cfg, let $L = L(G)$, and let Σ be an alphabet such that L is a subset of Σ^*. We will first give an algorithm that produces a finite set F of words such that L is infinite if and only if there is some word w in both F and L. Then to test if L is infinite, we test whether or not w is in L for each w in F. If any such w is in L, then L is infinite. If no such w is in L, then L is finite. Let $F = \{z \mid z \in \Sigma^* \text{ and } k < \text{length}(z) \leqslant 2k\}$, where k is the constant associated with G by the pumping lemma 2.17. The next claim tells us that this F is just the sort of set we need for our plan.

Claim 1: L is infinite if and only if there is some word z in F which is also in L.

In order to prove Claim 1, it will suffice to prove two things: (1) If there is a z in $L \cap F$, then L is infinite and (2) if L is infinite then $L \cap F$ is not empty. We first prove (1). Suppose z is in $L \cap F$. Then $\text{length}(z) > k$. By the pumping lemma, $z = uvwxy$ where $vx \neq \Lambda$ and $uv^i wx^i y$ is in L for all $i = 0, 1, 2, \ldots$. So there are an infinite number of strings in L, namely, the $uv^i wx^i y$. Thus (1) is shown.

Next we show (2). Suppose L is infinite. We must show that $L \cap F$ is not empty. Since L is infinite, it follows that there are arbitrarily long strings in L. So there must be some string z in L such that $\text{length}(z) > k$. Now if $\text{length}(z) \leqslant 2k$, then z is in F. Thus z is $L \cap F$ and (2) is true. Unfortunately, all we know is that $\text{length}(z) > k$. It could be true that $\text{length}(z) > 2k$. Assume the worst, namely, that $\text{length}(z) > 2k$. We will find another word z' such that z' is in L and $k < \text{length}(z') \leqslant 2k$. The string z' is obtained from z by using the pumping lemma. By the pumping lemma, $z = uvwxy$ where $vx \neq \Lambda$ and $\text{length}(vwx) \leqslant k$ and uwy is in L. Now $\text{length}(z) > 2k$ and, since $\text{length}(vwx) \leqslant k$, $\text{length}(uwy) \geqslant \text{length}(z) - k$. Therefore, $\text{length}(uwy) > k$. But $vx \neq \Lambda$. Thus uwy is shorter than z. So, starting with any z such that z is in L and $\text{length}(z) > 2k$, we can produce a string t, namely, $t = uwy$, such that $\text{length}(t) > k$ and $\text{length}(t) < \text{length}(z)$. If $\text{length}(t) \leqslant 2k$, then set $z' = t$ and we have our z' in $F \cap L$. If $\text{length}(t) > 2k$, then just as we produced a shorter string in L from z, we can produce a shorter string in L from t. We repeat this process again and again, and get a list of strings in L each shorter than the previous one. We continue to do this so long as the strings produced are greater than $2k$. Since each string produced is shorter than the previous string, we eventually get a string z' in L such that $\text{length}(z') \leqslant 2k$ and, by the way we are producing these strings, we also know $\text{length}(z') > k$. Therefore, z' is in $F \cap L$. Thus $F \cap L$ is nonempty and (2) is proven.

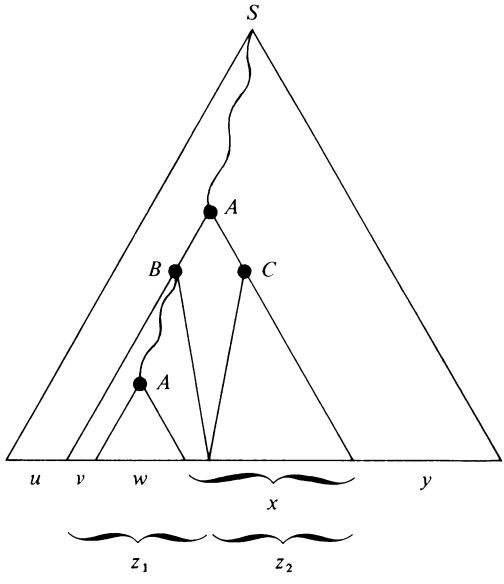

Figure 2.5(a)

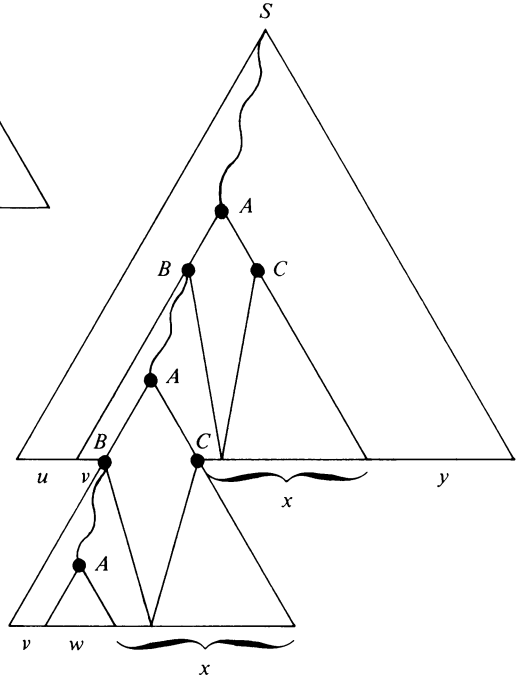

Figure 2.5(b)

Procedure 2.1

Algorithm to tell if L is finite or infinite.

1. Produce $F = \{z \mid k < \text{length}(z) \leqslant 2k\}$.
2. For each z in F, test if z is in L.
3. If one such test gives an affirmative answer output, "L is infinite."
4. If all tests in 2 give a negative answer output, "L is finite."
