In order to obtain our pumping lemma, we must show that this duplication of variables occurs in the derivation of every sufficiently long string in $L(G)$. It will also be helpful to impose some restrictions on the strings v, w, x, y, and z, just as we did on the strings u, v, and w in the simpler case.

The discussion will be a little easier if we can assume that the tree representing a derivation is a *binary* tree, which means simply that no node has more than two children. We can guarantee this by putting our grammar into Chomsky normal form (Section 6.6). The resulting loss of the null string will not matter, because the result we want involves only long strings.

Let us say that a *path* in a nonempty binary tree consists either of a single node or of a node, one of its descendants, and all the nodes in between. We will say that the *length* of a path is the number of nodes it contains, and the *height* of a binary tree is the length of the longest path. In any derivation whose tree has a sufficiently long path, some variable must reoccur. Lemma 8.1 shows that any binary tree will have a long path if the number of leaf nodes is sufficiently large. (In the case we are interested in, the binary tree is a derivation tree, and because there are no Λ-productions the number of leaf nodes is simply the length of the string being derived.)

Lemma 8.1 For any $h \geq 1$, a binary tree having more than 2^{h-1} leaf nodes must have height greater than h.

Proof We prove, by induction on h, the contrapositive statement: If the height is no more than h, the number of leaf nodes is no greater than 2^{h-1}. For the basis step, we observe that a binary tree with height ≤ 1 has no more than one node and therefore no more than one leaf node.

In the induction step, suppose that $k \geq 1$ and that any binary tree of height $\leq k$ has no more than 2^{k-1} leaf nodes. Now let T be a binary tree with height $\leq k+1$. If T has no more than one node, the result is clear. Otherwise, the left and right subtrees of T both have height $\leq k$, and so by the induction hypothesis each has 2^{k-1} or fewer leaf nodes. The number of leaf nodes in T is the sum of the numbers in the two subtrees and is therefore no greater than $2^{k-1} + 2^{k-1} = 2^k$. ■

Theorem 8.1
Let $G = (V, \Sigma, S, P)$ be a context-free grammar in Chomsky normal form, with a total of p variables. Any string u in $L(G)$ with $|u| \geq 2^{p+1}$ can be written as $u = vwxyz$, for some strings v, w, x, y, and z satisfying

$$|wy| > 0$$
$$|wxy| \leq 2^{p+1}$$
$$\text{for any } m \geq 0,\ vw^m xy^m z \in L(G)$$

Proof
Lemma 8.1 shows that any derivation tree for u must have height at least $p+2$. (It has more than 2^p leaf nodes, and therefore its height is $> p+1$.) Let us consider a path of maximum length and look at the bottom portion, consisting of a leaf node and the $p+1$ nodes above it. Each of these $p+1$ nodes corresponds to a variable, and since there are only p distinct variables, some variable A must appear twice in this portion of the path. Let x be the portion of u derived from the A closest to the leaf, and let $t = wxy$ be the portion of u derived from the other A. If v and z represent the beginning and ending portions of u, we have $u = vwxyz$. An illustration of what this might look like is provided in Figure 8.1.

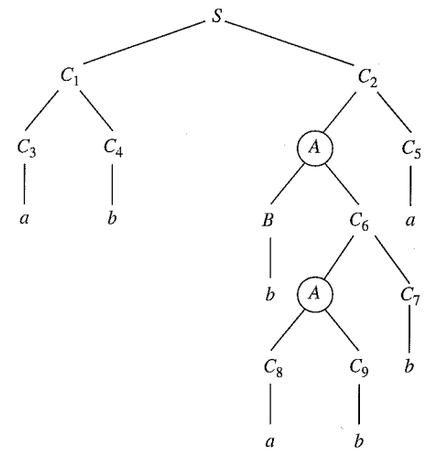
The A closest to the root in this portion of the path is the root of a binary derivation tree for wxy. Since we began with a path of maximum length, this tree has height $\leq p+2$, and so by the lemma $|wxy| \leq 2^{p+1}$. The node containing this A has two children, both corresponding to variables. If we let B denote the one that is not an ancestor of the other A, then since x is derived from that other A, the string of terminals derived from B does not overlap x. It follows that either w or y is nonnull, and therefore $|wy| > 0$.

Finally,

$$S \Rightarrow^* vAz \Rightarrow^* vwAyz \Rightarrow^* vwxyz$$

(the first A is the one in the higher node, the second the one in the lower), and so the third conclusion of the theorem follows from our preliminary discussion.

Just as in the case of the earlier pumping lemma, it is helpful to restate the result so as to emphasize the essential features.



$u = (ab)(b)(ab)(b)(a)$
$= v\ w\ x\ y\ z$

Figure 8.1 |

Theorem 8.1a The Pumping Lemma for Context-Free Languages
Let L be a CFL. Then there is an integer n so that for any $u \in L$ satisfying $|u| \geq n$, there are strings v, w, x, y, and z satisfying

$$u = vwxyz \tag{8.1}$$

$$|wy| > 0 \tag{8.2}$$

$$|wxy| \leq n \tag{8.3}$$

$$\text{For any } m \geq 0,\ vw^m xy^m z \in L \tag{8.4}$$

Proof
We can find a context-free grammar in Chomsky normal form that generates $L - \{\Lambda\}$. If we let p be the number of variables in this grammar and $n = 2^{p+1}$, the result is an immediate consequence of Theorem 8.1.

Using the pumping lemma for context-free languages requires the same sorts of precautions as in the case of the earlier pumping lemma for regular languages. In order to show that L is not context-free, we assume it is and try to derive a contradiction. Theorem 8.1a says only that there exists an integer n, nothing about its value; because we can apply the theorem only to strings u with length $\geq n$, the u we choose must be defined in terms of n. Once we have chosen u, the theorem tells us only that there exist strings v, w, x, y, and z satisfying the four properties; the only way to guarantee a contradiction is to show that *every* choice of v, w, x, y, z satisfying the properties leads to a contradiction.

According to Chapter 7, a finite-state machine with an auxiliary memory in the form of a stack is enough to accept a CFL. An example of a language for which a single stack is sufficient is the language of strings of the form $(^i)^i$ (see Example 7.3), which could just as easily have been called $a^i b^i$. The a's are saved on the stack so that the number of a's can be compared to the number of b's that follow. By the time all the b's have been matched with a's, the stack is empty, which means that the machine has forgotten how many there were. This approach, therefore, would not allow us to recognize strings of the form $a^i b^i c^i$. Rather than trying to show directly that no other approach using a single stack can work, we choose this language for our first proof using the pumping lemma.

EXAMPLE 8.1 The Pumping Lemma Applied to $\{a^i b^i c^i\}$

Let

$$L = \{a^i b^i c^i \mid i \geq 1\}$$

Suppose for the sake of contradiction that L is context-free, and let n be the integer in Theorem 8.1a. An obvious choice for u with $|u| \geq n$ is $u = a^n b^n c^n$. Suppose v, w, x, y, and z are any strings satisfying conditions (8.1)–(8.4). Since $|wxy| \leq n$, the string wxy can contain at most two distinct types of symbol (a's, b's, and c's), and since $|wy| > 0$, w and y together contain at least one. The string vw^2xy^2z contains additional occurrences of the symbols in w and y;

therefore, it cannot contain equal numbers of all three symbols. On the other hand, according to (8.4), $vw^2xy^2z \in L$. This is a contradiction, and our assumption that L is a CFL cannot be correct.

Note that to get the contradiction, we started with $u \in L$ and showed that vw^2xy^2z fails to be an element, not only of L but also of the bigger language

$$L_1 = \{u \in \{a, b, c\}^* \mid n_a(u) = n_b(u) = n_c(u)\}$$

Therefore, our proof is also a proof that L_1 is not a context-free language.

EXAMPLE 8.2 The Pumping Lemma Applied to $\{ss \mid s \in \{a, b\}^*\}$

Let

$$L = \{ss \mid s \in \{a, b\}^*\}$$

This language is similar in one obvious respect to the language of even-length palindromes, which is a context-free language: In both cases, in order to recognize a string in the language, a machine needs to remember the first half so as to be able to compare it to the second. For palindromes, the last-in, first-out principle by which a stack operates is obviously appropriate. In the case of L, however, when we encounter the first symbol in the second half of the string (even assuming that we know when we encounter it), the symbol we need to compare it to is the *first* in, not the last—in other words, it is buried at the bottom of the stack. Here again, arguing that the obvious approach involving a PDA fails does not prove that a PDA cannot be made to work. Instead, we apply the pumping lemma.

Suppose L is a CFL, and let n be the integer in Theorem 8.1a. This time the choice of u is not as obvious; we try $u = a^n b^n a^n b^n$. Suppose that v, w, x, y, and z are any strings satisfying (8.1)–(8.4). We must derive a contradiction from these facts, without making any other assumptions about the five strings.

As in Example 8.1, condition (8.2) tells us that wxy can overlap at most two of the four contiguous groups of symbols. We consider several cases.

First, suppose w or y contains at least one a from the first group of a's. Since $|wxy| \leq n$, neither w nor y can contain any symbols from the second half of u. Consider $m = 0$ in condition (8.4). Omitting w and y causes at least one initial a to be omitted, and does not affect the second half. In other words,

$$vw^0xy^0z = a^i b^j a^n b^n$$

where $i < n$ and $1 \leq j \leq n$. The midpoint of this string is somewhere within the substring a^n, and therefore it is impossible for it to be of the form ss.

Next, suppose wxy contains no a's from the first group but that w or y contains at least one b from the first group of b's. Again we consider $m = 0$; this time we can say

$$vw^0xy^0z = a^n b^i a^j b^n$$

where $i < n$ and $1 \leq j \leq n$. The midpoint is somewhere in the substring $b^i a^j$, and as before, the string cannot be in L.

We can get by with two more cases: case 3, in which wxy contains no symbols from the first half of u but at least one a, and case 4, in which w or y contains at least one b from

the second group of b's. The arguments in these cases are similar to those in cases 2 and 1, respectively. We leave the details to you.

Just as in Example 8.1, there are other languages for which essentially the same proof works. Two examples in this case are $\{a^ib^ia^ib^i \mid i \geq 0\}$ and $\{a^ib^ja^ib^j \mid i, j \geq 0\}$. A similar proof shows that $\{scs \mid s \in \{a, b\}^*\}$ also fails to be context-free. Although the marker in the middle may appear to remove the need for nondeterminism, the basic problem that prevents a PDA from recognizing this language is still present.

In proofs of this type, there are several potential trouble-spots. It may not be obvious what string u to choose. In Example 8.2, although $a^nb^na^nb^n$ is not the only choice that works, there are many that do not (Exercise 8.2).

Once u is chosen, deciding on cases to consider can be a problem. A straightforward way in Example 8.2 might have been to consider seven cases:

1. wy contains only a's from the first group;
2. wy contains a's from the first group and b's from the first group;
3. wy contains only b's from the first group;

   ...

7. wy contains only b's from the last group.

There is nothing wrong with this, except that all other things being equal, four cases is better than seven. If you find yourself saying "Cases 6, 8, and 10 are handled exactly like cases 1, 3, and 5," perhaps you should try to reduce the number of cases. (Try to rephrase the proof in Example 8.2 so that there are only two cases.) The important thing in any case is to make sure your cases cover all the possibilities and that you do actually obtain a contradiction in each case.

Finally, for a specific case, you must choose the value of m to use in order to obtain a contradiction. In the first case in Example 8.2, it was not essential to choose $m = 0$, but the string vw^0xy^0z is probably easier to describe exactly than the string vw^2xy^2z. In some situations, choosing $m = 0$ will not work but choosing $m > 1$ will, and in some other situations the opposite is true.

EXAMPLE 8.3 A Third Application of the Pumping Lemma

Let

$$L = \{x \in \{a, b, c\}^* \mid n_a(x) < n_b(x) \text{ and } n_a(x) < n_c(x)\}$$

The intuitive reason that no PDA can recognize L is similar to the reason in Example 8.1; a stack allows the machine to compare the number of a's to either the number of b's or the number of c's, but not both. Suppose L is a CFL, and let n be the integer in Theorem 8.1a. Let $u = a^nb^{n+1}c^{n+1}$. If (8.1)–(8.4) hold for strings v, w, x, y, and z, the string wxy can contain at most two distinct symbols. This time, two cases are sufficient.

If w or y contains at least one a, then wy cannot contain any c's. Therefore, vw^2xy^2z contains at least as many a's as c's and cannot be in L. If neither w nor y contains an a, then vw^0xy^0z still contains n a's; since wy contains one of the other two symbols, vw^0xy^0z contains fewer occurrences of that symbol than u does and therefore is not in L. We have obtained a contradiction and may conclude that L is not context-free.

The language $\{a^ib^jc^k \mid i < j \text{ and } i < k\}$ can be shown to be non-context-free by exactly the same argument.

EXAMPLE 8.4 The Set of Legal C Programs Is Not a CFL

The feature of the C programming language that we considered in Section 5.5, which prevents the language from being regular, can be taken care of by context-free grammars. In Chapter 6 we saw other examples of the way in which CFGs can describe much of the syntax of such high-level languages. CFGs cannot do this completely, however: There are some rules in these languages that depend on context, and Theorem 8.1a allows us to show that the set L of all legal C programs is not a context-free language. (Very little knowledge of C is required for this example.)

A basic rule in C is that a variable must be declared before it is used. Checking that this rule is obeyed is essentially the same as determining whether a certain string has the form xcx, where x is the identifier and c is the string appearing between the declaration of x and its use. As we observed in Example 8.2, the language $\{xcx \mid x \in \{a, b\}^*\}$ is not a CFL. Although we are now using a larger alphabet, and c is no longer a single symbol, the basic problem is the same, provided identifiers are allowed to be arbitrarily long. Let us try to use the pumping lemma to show that L is not a CFL.

Assume that L is a CFL, and let n be the integer in Theorem 8.1a. We want to choose a string u in L whose length is at least n, containing both a declaration of a variable and a separate, subsequent reference to the variable. The following will (barely) qualify:

```
main(){int aa...a;aa...a;}
```

where both identifiers have n a's. However, for a technical reason that will be mentioned in a minute, we complicate the program by including two subsequent references to the variable instead of one:

```
main(){int aa...a;aa...a;aa...a;}
```

Here it is assumed that all three identifiers are a^n. There is one blank in the program, after `int`, and it is necessary as a separator. About all that can be said for this program is that it will make it past the compiler, possibly with a warning. It declares an integer variable; then, twice, it evaluates the expression consisting of that identifier (the value is probably garbage, since the program has not initialized the variable), and does nothing with the value.

According to the pumping lemma, $u = vwxyz$, where (8.2)–(8.4) are satisfied. In particular, vw^0xy^0z is supposed to be a valid C program. However, this is impossible. If wy contains the blank or any of the symbols before it, then vxz still contains at least most of the first occurrence of the identifier, and without `main(){int` and the blank, it cannot be syntactically correct. We are left with the case in which wxy is a substring of "a^n; a^n; a^n;". If wy contains either the final semicolon or bracket, the string vxz is also illegal. If it contains one of the two intermediate semicolons, and possibly portions of one or both of the identifiers on either

side, then vxz has two identifiers, which are now not the same. Finally, if wy contains only a portion of one of the identifiers and nothing else, then vxz still has a variable declaration and two subsequent expressions consisting of an identifier, but the three identifiers are not all the same. In either of these last two situations, the declaration-before-use principle is violated. We conclude that vxz is not a legal C program, and therefore that L is not a context-free language.

(The argument would almost work with the shorter program having only two occurrences of the identifier, but not quite. The case in which it fails is the case in which wy contains the first semicolon and nothing else. Deleting it still leaves a valid program, consisting simply of a declaration of a variable with a longer name. Adding multiple copies is also legal because they are interpreted as harmless "empty" statements.)

There are other examples of syntax rules whose violation cannot be detected by a PDA. We noted in Example 8.2 that $\{a^n b^m a^n b^m\}$ is not a context-free language, and we can imagine a situation in which being able to recognize a string of this type is essentially what is required. Suppose that two functions f and g are defined, having n and m formal parameters, respectively, and then calls on f and g are made. Then the numbers of parameters in the calls must agree with the numbers in the respective definitions.

In the remainder of this section, we discuss a generalization of the pumping lemma, a slightly weakened form of what is known as Ogden's lemma. Although the pumping lemma provides some information about the strings w and y that are pumped, in the form of (8.2) and (8.3), it says very little about the location of these substrings in the string u. Ogden's lemma makes it possible to designate certain positions of u as "distinguished" and to guarantee that the pumped portions include at least some of these distinguished positions. As a result, it is sometimes more convenient than the pumping lemma, and occasionally it can be used when the pumping lemma fails.

Theorem 8.2 Ogden's Lemma

Suppose L is a context-free language. Then there is an integer n so that if u is any string in L of length n or greater, and any n or more positions of u are designated as "distinguished," then there are strings v, w, x, y, and z satisfying

$$u = vwxyz \tag{8.5}$$

the string wy contains at least one distinguished position (8.6)

the string wxy contains no more than n distinguished positions (8.7)

the string x contains at least one distinguished position (8.8)

$$\text{for every } m \geq 0,\ vw^m xy^m z \in L \tag{8.9}$$

Proof

Just as in the proof of the pumping lemma, let $n = 2^{p+1}$, where p is the number of variables in a Chomsky normal form grammar generating $L - \{\Lambda\}$. Suppose u is a string in L in which n or more positions are marked as distinguished. In order to obtain the strings v, w, x, y, and z, we start as

before, by selecting a path in a derivation tree for u on which some variable A occurs twice. Once we identify the two nodes corresponding to A, if we define the five strings exactly as before, it will follow that (8.5) and (8.9) hold. Only the other three properties depend on the particular way we choose the path, and so this is what we need to describe.

Beginning at the top, the path contains the root node, corresponding to S. For each interior node N in the path, we extend the path downward one more step by selecting the child of N having the larger number of distinguished positions among its descendants (breaking ties arbitrarily). Now we say that an interior node on this path is a *branch point* if both its children have at least one distinguished descendant. As we follow the path down the tree, starting at a branch point, the number of distinguished descendants of the current node decreases at the first step and remains constant thereafter until the next branch point is reached. It follows from the way the path is chosen that every branch point below the top one has at least half as many distinguished descendants as the branch point above it in the path. Using this result, we obtain the following lemma (corresponding to Lemma 8.1 in its contrapositive form), for which we interrupt our proof briefly.

Lemma 8.2 If the binary tree consisting of a node on the path and its descendants has h or fewer branch points, its leaf nodes include no more than 2^h distinguished nodes.

Proof The proof is by induction and is virtually identical to that of Lemma 8.1, except that the number of branch points is used instead of the height, and the number of distinguished leaf nodes is used instead of the total number of leaf nodes. The reason the statement involves 2^h rather than 2^{h-1} is that the bottom-most branch point has two distinguished descendants rather than one. ■

Conclusion of proof of Theorem 8.2

Since there are n distinguished leaf nodes in the tree, and $n > 2^p$, Lemma 8.2 implies that there must be more than p branch points in our path. Consider the $p+1$ branch points farthest down in the path, and the subtree whose root is the topmost such node. Since there are only p variables in the grammar, at least two of these branch points are labeled with the same variable A. We now define the five strings v, w, x, y, and z in terms of these two nodes exactly as in the proof of the pumping lemma. Property (8.6) follows from the definition of branch point, and (8.7) is an immediate consequence of Lemma 8.2. Property (8.8) is clearly true because the bottom A is a branch point. As we observed above, the other two properties follow as in the proof of the pumping lemma.

Note that the ordinary pumping lemma is identical to the special case of Theorem 8.2 in which *all* the positions of u are distinguished.

EXAMPLE 8.5 Using Ogden's Lemma on $\{a^i b^i c^j \mid j \neq i\}$

Let $L = \{a^i b^i c^j \mid i, j \geq 0 \text{ and } j \neq i\}$. Suppose L is a context-free language, and let n be the integer in the statement of Theorem 8.2. One choice for the string u is

$$u = a^n b^n c^{n+n!}$$

(The reason for this choice will be clear shortly.) Let us also designate the first n positions of u as the distinguished positions, and let us suppose that v, w, x, y, and z satisfy (8.5)–(8.9).

First, we can see that if either w or y contains two distinct symbols, then we can obtain a contradiction by considering the string vw^2xy^2z, which will no longer have the form $a^*b^*c^*$. Second, we know that because wy contains at least one distinguished position, w or y consists of a's. It follows from these two observations that unless w consists of a's and y consists of the same number of b's, looking at vw^2xy^2z will give us a contradiction, because this string has different numbers of a's and b's. Suppose now that $w = a^j$ and $y = b^j$. Let $k = n!/j$, which is still an integer, and let $m = k + 1$. Then the number of a's in vw^mxy^mz is

$$n + (m - 1) * j = n + k * j = n + n!$$

which is the same as the number of c's. We have our contradiction, and therefore L cannot be context-free.

With just the pumping lemma here, we would be in trouble: There would be no way to rule out the possibility that w and y contained only c's, and therefore no way to guarantee a contradiction.

EXAMPLE 8.6 Using Ogden's Lemma when the Pumping Lemma Fails

Let $L = \{a^p b^q c^r d^s \mid p = 0 \text{ or } q = r = s\}$. It seems clear that L should not be a CFL, because $\{a^q b^q c^q\}$ is not. The first thing we show in this example, however, is that L satisfies the properties of the pumping lemma; therefore, the pumping lemma will not help us to show that L is not context-free.

Suppose n is any positive integer, and u is any string in L with $|u| \geq n$, say $u = a^p b^q c^r d^s$. We must show the existence of strings v, w, x, y, and z satisfying (8.1)–(8.4). We consider two cases. If $p = 0$, then there are no restrictions on the numbers of b's, c's, or d's, and the choices $w = b$, $v = x = y = \Lambda$ work. If $p > 0$, then we know that $q = r = s$, and the choices $w = a$, $v = x = y = \Lambda$ work.

Now we can use Theorem 8.2 to show that L is indeed not a CFL. Suppose L is a CFL, let n be the integer in the theorem, let $u = ab^n c^n d^n$, and designate all but the first position of u as distinguished. Suppose that v, w, x, y, and z satisfy (8.5)–(8.9). Then the string wy must contain one of the symbols b, c, or d and cannot contain all three. Therefore, vw^2xy^2z has one a and does not have equal numbers of b's, c's, and d's, which means that it cannot be in L.

8.2 INTERSECTIONS AND COMPLEMENTS OF CONTEXT-FREE LANGUAGES

According to Theorem 6.1, the set of context-free languages is closed under the operations of union, concatenation, and Kleene *. For regular languages, we can add

the intersection and complement operations to the list. We can now show, however, that for context-free languages this is not possible.

Theorem 8.3
There are CFLs L_1 and L_2 so that $L_1 \cap L_2$ is not a CFL, and there is a CFL L so that L' is not a CFL.

Proof
In Example 8.3 we observed that

$$L = \{a^i b^j c^k \mid i < j \text{ and } i < k\}$$

is not context-free. However, although no PDA can test both conditions $i < j$ and $i < k$ simultaneously, it is easy enough to build two PDAs that test the conditions separately. In other words, although the intersection L of the two languages

$$L_1 = \{a^i b^j c^k \mid i < j\}$$

and

$$L_2 = \{a^i b^j c^k \mid i < k\}$$

is not a CFL, both languages themselves are. Another way to verify that L_1 is a CFL is to check that it is generated by the grammar with productions

$$S \to ABC \quad A \to aAb \mid \Lambda \quad B \to bB \mid b \quad C \to cC \mid \Lambda$$

Similarly, L_2 is generated by the CFG with productions

$$S \to AC \quad A \to aAc \mid B \quad B \to bB \mid \Lambda \quad C \to cC \mid c$$

The second statement in the theorem follows from the first and the formula

$$L_1 \cap L_2 = (L_1' \cup L_2')'$$

If complements of CFLs were always CFLs, then for any CFLs L_1 and L_2, the languages L_1' and L_2' would be CFLs, so would their union, and so would its complement. We know now that this is not the case.

EXAMPLE 8.7 A CFL Whose Complement Is Not a CFL

The second part of the proof of Theorem 8.3 is a proof by contradiction and appears to be a nonconstructive proof. If we examine it more closely, however, we can use it to find an example. Let L_1 and L_2 be the languages defined in the first part of the proof. Then the language $L_1 \cap L_2 = (L_1' \cup L_2')'$ is not a CFL. Therefore, because the union of CFLs is a CFL, at least one of the three languages L_1, L_2, $L_1' \cup L_2'$ is a CFL whose complement is not a CFL. Let us try to determine which.

There are two ways a string can fail to be in L_1. It can fail to be an element of R, the language $\{a\}^*\{b\}^*\{c\}^*$, or it can be a string $a^i b^j c^k$ for which $i \geq j$. In other words,

$$L_1' = R' \cup \{a^i b^j c^k \mid i \geq j\}$$

The language R' is regular because R is, and therefore R' is context-free. The second language involved in the intersection can be expressed as the concatenation

$$\{a^i b^j c^k \mid i \geq j\} = \{a^m \mid m \geq 0\}\ \{a^j b^j \mid j \geq 0\}\ \{c^k \mid k \geq 0\}$$

each factor of which is a CFL. Therefore, L_1' is a CFL. A similar argument shows that L_2' is also a CFL. We conclude that $L_1' \cup L_2'$, or

$$R' \cup \{a^i b^j c^k \mid i \geq j \text{ or } i \geq k\}$$

is a CFL whose complement is not a CFL. (In fact, the second part alone is also an example; see Exercise 8.8.)

At this point it might be interesting to go back to Theorem 3.4, in which the intersection of two regular languages was shown to be regular, and see what goes wrong when we try to use the same construction for CFLs. We began with FAs M_1 and M_2 recognizing the two languages, and we constructed a composite machine M whose states were pairs (p, q) of states in M_1 and M_2, respectively. This allows M to keep track of both machines at once, or to simulate running the two machines in parallel. A string is accepted by M if it is accepted simultaneously by M_1 and M_2.

Suppose we have CFLs L_1 and L_2, and PDAs $M_1 = (Q_1, \Sigma, \Gamma_1, q_1, Z_1, A_1, \delta_1)$ and $M_2 = (Q_2, \Sigma, \Gamma_2, q_2, Z_2, A_2, \delta_2)$ accepting them. We can define states of a new machine M in the same way, by letting $Q = Q_1 \times Q_2$. We might also construct the stack alphabet of M by letting $\Gamma = \Gamma_1 \times \Gamma_2$, because this would let us use the top stack symbol of M to determine those of M_1 and M_2. When we try to define the moves of M, however, things become complicated, even aside from the question of nondeterminism. In the simple case where $\delta_1(p, a, X) = \{(p', X')\}$ and $\delta_2(q, a, Y) = \{(q', Y')\}$, where $X, X' \in \Gamma_1$ and $Y, Y' \in \Gamma_2$, it is reasonable to let

$$\delta((p, q), a, (X, Y)) = \{((p', q'), (X', Y'))\}$$

However, what if $\delta_1(p, a, X) = \{(p', X'X)\}$ and $\delta_2(q, a, Y) = \{(q', \Lambda)\}$? Or, what if $\delta_1(p, a, X) = \{(p', X)\}$ and $\delta_2(q, a, Y) = \{(q', YYY)\}$? There is no obvious way to have M keep track of both the states of M_1 and M_2 and the stacks of M_1 and M_2 and still operate like a PDA. Theorem 8.3 confirms that such a machine is indeed impossible in general.

We can salvage some positive results from this discussion. If M_1 is a PDA and M_2 is a PDA with *no* stack (in particular, a deterministic one—i.e., an FA) there is no obstacle to carrying out this construction. The stack on our new machine is simply the one associated with M_1. The result is the following theorem.

Theorem 8.4
If L_1 is a context-free language and L_2 is a regular language, then $L_1 \cap L_2$ is a context-free language.

Proof
Let $M_1 = (Q_1, \Sigma, \Gamma, q_1, Z_0, A_1, \delta_1)$ be a PDA accepting the language L_1, and let $M_2 = (Q_2, \Sigma, q_2, A_2, \delta_2)$ be an FA recognizing L_2. We define a

PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ as follows:

$$Q = Q_1 \times Q_2 \qquad q_0 = (q_1, q_2) \qquad A = A_1 \times A_2$$

For $p \in Q_1$, $q \in Q_2$, and $Z \in \Gamma$,

(1) $\delta((p, q), a, Z)$
$$= \{((p', q'), \alpha) \mid (p', \alpha) \in \delta_1(p, a, Z) \text{ and } \delta_2(q, a) = q'\}$$
for every $a \in \Sigma$; and

(2) $\delta((p, q), \Lambda, Z) = \{((p', q), \alpha) \mid (p', \alpha) \in \delta_1(p, \Lambda, Z)\}$

We can see that (1) and (2) allow M to keep track of the states of M_1 and M_2. Note that because an FA is deterministic, the only way in which M has a choice of moves is in the first component of the state; similarly, in the case of a Λ-transition of M_1, the second component of M's state, the one corresponding to M_2, is unchanged. The contents of the stack at any time are just what they would be in the machine M_1.

We want to show that these two statements are equivalent: On the one hand, the input string y allows M_1 to reach state p with stack contents α, and causes M_2 to reach state q; on the other hand, by processing y, M can get to state (p, q) with stack contents α. More precisely, for any $n \geq 0$ and any $p \in Q_1$, $q \in Q_2$, $y, z \in \Sigma^*$, and $\alpha \in \Gamma^*$,

$$(q_1, yz, Z_1) \vdash_{M_1}^n (p, z, \alpha) \quad \text{and} \quad \delta_2^*(q_2, y) = q$$

if and only if

$$((q_1, q_2), yz, Z_1) \vdash_M^n ((p, q), z, \alpha)$$

where the notation $\vdash^n$ refers to a sequence of n moves. The "only if" direction shows that $L(M_1) \cap L(M_2) \subseteq L(M)$, and the converse shows the opposite inclusion.

The two parts of the proof are both by induction on n and are very similar. We show the "only if" part and leave the other to you. For the basis step, suppose that

$$(q_1, yz, Z_1) \vdash_{M_1}^0 (p, z, \alpha) \quad \text{and} \quad \delta_2^*(q_2, y) = q$$

This means that $y = \Lambda$, $p = q_1$, $\alpha = Z_1$, and $q = q_2$. In this case, it is clear that

$$((q_1, q_2), yz, Z_1) \vdash_M^0 ((p, q), z, \alpha) = ((q_1, q_2), yz, Z_1)$$

Now suppose that $k \geq 0$ and that the statement is true for $n = k$, and assume that

$$(q_1, yz, Z_1) \vdash_{M_1}^{k+1} (p, z, \alpha) \quad \text{and} \quad \delta_2^*(q_2, y) = q$$

We want to show that

$$((q_1, q_2), yz, Z_1) \vdash_M^{k+1} ((p, q), z, \alpha)$$

Consider the last move in the sequence of $k+1$ moves of M_1. If it is a Λ-transition, then

$$(q_1, yz, Z_1) \vdash_{M_1}^{k} (p', z, \beta) \vdash_{M_1} (p, z, \alpha)$$

for some $p' \in Q_1$ and some $\beta \in \Gamma^*$. In this case the inductive hypothesis implies that

$$((q_1, q_2), yz, Z_1) \vdash_{M}^{k} ((p', q), z, \beta)$$

and from (2) we have

$$((p', q), z, \beta) \vdash_{M} ((p, q), z, \alpha)$$

which implies the result. Otherwise, $y = y'a$ for some $a \in \Sigma$, and

$$(q_1, y'az, Z_1) \vdash_{M_1}^{k} (p', az, \beta) \vdash_{M_1} (p, z, \alpha)$$

Let $q' = \delta_2^*(q_2, y')$. Then the induction hypothesis implies that

$$((q_1, q_2), y'az, Z_1) \vdash_{M}^{k} ((p', q'), az, \beta)$$

and from (1) we have

$$((p', q'), az, \beta) \vdash_{M} ((p, q), z, \alpha)$$

Again the result follows.

It is also worthwhile, in the light of Theorems 8.3 and 8.4, to re-examine the proof that the complement of a regular language is regular. If the finite automaton $M = (Q, \Sigma, q_0, A, \delta)$ recognizes the language L, then the finite automaton $M' = (Q, \Sigma, q_0, Q - A, \delta)$ recognizes $\Sigma^* - L$. We are free to apply the same construction to the pushdown automaton $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ and to consider the PDA $M' = (Q, \Sigma, \Gamma, q_0, Z_0, Q - A, \delta)$. Theorem 8.3 says that even if M accepts the context-free language L, M' does not necessarily accept $\Sigma^* - L$. Why not?

The problem is nondeterminism. It may happen that for some $x \in \Sigma^*$,

$$(q_0, x, Z_0) \vdash_{M}^{*} (p, \Lambda, \alpha)$$

for some state $p \in A$, and

$$(q_0, x, Z_0) \vdash_{M}^{*} (q, \Lambda, \beta)$$

for some other state $q \notin A$. This means that the string x is accepted by M as well as by M', since q is an accepting state in M'. In the case of finite automata, nondeterminism can be eliminated: Every NFA-Λ is equivalent to some FA. The corresponding result about PDAs is false (Theorem 7.1), and the set of CFLs is not closed under the complement operation.

We would expect that if M is a *deterministic* PDA (DPDA) recognizing L, then the machine M' constructed as above would recognize $\Sigma^* - L$. Unfortunately, this is still not quite correct. One reason is that there might be input strings that cause M to enter an infinite sequence of Λ-transitions and are never processed to completion. Any such string would be accepted neither by M nor by M'. However, the result in Exercise 7.46 shows that this difficulty can be resolved and that a DPDA can be constructed that recognizes $\Sigma^* - L$. It follows that the complement of a deterministic context-free language (DCFL) is a DCFL, and in particular that any context-free language whose complement is not a CFL (Theorem 8.3 and Example 8.7) cannot be a DCFL.

8.3 DECISION PROBLEMS INVOLVING CONTEXT-FREE LANGUAGES

At the end of Chapter 5, we considered a number of decision problems involving regular languages, beginning with the membership problem: Given an FA M and a string x, does M accept x? We may formulate the same sorts of questions for context-free languages. For some of the questions, essentially the same algorithms work; for others, the inherent nondeterminism of PDAs requires us to find new algorithms; and for some, *no* algorithm is possible, although proving this requires a more sophisticated model of computation than we now have.

The basic membership problem for regular languages has a simple solution: To decide whether an FA M accepts a string x, run M with input x and see what happens. In the case of PDAs, a solution cannot be this simple, because nondeterminism cannot always be eliminated. If we think of a PDA M as making a move once a second, and choosing arbitrarily whenever it has a choice, then "what happens" may be different from one time to the next. A sequence of moves that ends up with the input x being rejected does not mean there is no other sequence leading to acceptance. From the specifications for a PDA, we can very likely formulate some backtracking algorithm that will be guaranteed to answer the question. Simpler to describe (though probably inefficient to carry out) is the following approach, which depends on the fact that a CFG without Λ-productions or unit productions requires at most $2n - 1$ steps for the derivation of a string of length n.

Decision algorithm for the membership problem. (Given a pushdown automaton M and a string x, does M accept x?) Use the construction in the proof of Theorem 7.4 to find a CFG G generating the language recognized by M. If $x = \Lambda$, use Algorithm FindNull in Section 6.5 to determine whether the start symbol of G is nullable. Otherwise, eliminate Λ-productions and unit productions from G, using Algorithms 6.1 and 6.2. Examine all derivations with one step, all those with two steps, and so on, until either a derivation of x has been found or all derivations of length $2|x| - 1$ have been examined. If no derivation of x has been found, M does not accept x.

Note that since Theorems 7.2 and 7.4 let us go from a CFG to a PDA and vice versa, we can formulate any of these decision problems in terms of either a CFG or a PDA. We consider the decision problems corresponding to problems 2 and 3 in Chapter 5, but stated in terms of CFGs:

1. Given a CFG G, does it generate any strings? (Is $L(G) = \emptyset$?)
2. Given a CFG G, is $L(G)$ finite?

Theorem 8.1 provides us with a way of answering both questions. We transform G into Chomsky normal form. Let G' be the resulting grammar, p the number of variables in G', and $n = 2^{p+1}$. If G' generates any strings, it must generate one of length less than n. Otherwise, apply the pumping lemma to a string u of minimal length ($\geq n$) generated by G'; then $u = vwxyz$, for some strings v, w, x, y, and z with $|wy| > 0$ and $vxz \in L(G')$—and this contradicts the minimality of u. Similarly, if $L(G')$ is infinite, there must be a string $u \in L(G')$ with $n \leq |u| < 2n$; the proof is virtually identical to that of the corresponding result in Chapter 5. We therefore obtain the two decision algorithms that follow.

Decision algorithms for problems 1 and 2. (Given a CFG G, is $L(G) = \emptyset$? Is $L(G)$ finite?) First, test whether Λ can be generated from G, using the algorithm for the membership problem. If it can, then $L(G) \neq \emptyset$. In any case, let G' be a Chomsky-normal-form grammar generating $L(G) - \{\Lambda\}$, and let $n = 2^{p+1}$, where p is the number of variables in G'. For increasing values of i beginning with 1, test strings of length i for membership in $L(G)$. If for no $i < n$ is there a string of length i in $L(G')$, and $\Lambda \notin L(G)$, then $L(G') = L(G) = \emptyset$. If for no i with $n \leq i < 2n$ is there a string of length i in $L(G)$, then $L(G)$ is finite; if there is a string $x \in L(G)$ with $n \leq |x| < 2n$, then $L(G)$ is infinite.

These algorithms are easy to describe, but obviously not so easy to carry out. Fortunately, for problems such as the membership problem for which it is important to find practical solutions, there are considerably more efficient algorithms [two well-known ones are those by Cocke-Younger-Kasami, described in the paper by Younger (*Information and Control* 10(2): 189–208, 1967) and Earley (*Communications of the ACM* 13(2): 94–102, 1970)]. If we go any farther down the list of decision problems in Chapter 5, formulated for CFGs or PDAs instead of regular expressions or FAs, we encounter problems for which there is no possible decision algorithm, even an extremely inefficient one. To take an example, given two CFGs, are there any strings generated by both? The algorithm given in Chapter 5 to solve the corresponding problem for FAs depends on the fact that the set of regular languages is closed under the operations of intersection and complement. Because the set of CFLs is not closed under these operations, we know at least that the earlier approach does not give us an algorithm; we will see in Chapter 11 that the problem is actually "unsolvable."

EXERCISES

8.1. In each case, show using the pumping lemma that the given language is not a CFL.

a. $L = \{a^i b^j c^k \mid i < j < k\}$

b. $L = \{x \in \{a, b\}^* \mid n_b(x) = n_a(x)^2\}$

c. $L = \{a^n b^{2n} a^n \mid n \geq 0\}$

d. $L = \{x \in \{a, b, c\}^* \mid n_a(x) = \max\{n_b(x), n_c(x)\}\}$

e. $L = \{a^n b^m a^n b^{n+m} \mid m, n \geq 0\}$

8.2. In the pumping-lemma proof in Example 8.2, give some examples of choices of strings $u \in L$ with $|u| \geq n$ that would not work.

8.3. In the proof given in Example 8.2 using the pumping lemma, the contradiction was obtained in each case by considering the string vw^0xy^0z. Would it have been possible instead to use vw^2xy^2z in each case?

8.4. In Example 8.4, is it possible with Ogden's lemma rather than the pumping lemma to use the string u mentioned first, with only two occurrences of the identifier?

8.5. Decide in each case whether the given language is a CFL, and prove your answer.

a. $L = \{a^n b^m a^m b^n \mid m, n \geq 0\}$

b. $L = \{xayb \mid x, y \in \{a, b\}^*$ and $|x| = |y|\}$

c. $L = \{xcx \mid x \in \{a, b\}^*\}$

d. $L = \{xyx \mid x, y \in \{a, b\}^*$ and $|x| \geq 1\}$

e. $L = \{x \in \{a, b\}^* \mid n_a(x) < n_b(x) < 2n_a(x)\}$

f. $L = \{x \in \{a, b\}^* \mid n_a(x) = 10n_b(x)\}$

g. $L =$ the set of non-balanced strings of parentheses

8.6. State and prove theorems that generalize Theorems 5.3 and 5.4 to context-free languages. Then give an example to illustrate each of the following possibilities.

a. Theorem 8.1a can be used to show that the language is a CFL, but the generalization of Theorem 5.3 cannot.

b. The generalization of Theorem 5.3 can be used to show the language is not a CFL, but the generalization of Theorem 5.4 cannot.

c. The generalization of Theorem 5.4 can be used to show the language is not a CFL.

8.7. Show that if L is a DCFL and R is regular, then $L \cap R$ is a DCFL.

8.8. In each case, show that the given language is a CFL but that its complement is not. (It follows in particular that the given language is not a DCFL.)

a. $\{a^i b^j c^k \mid i \geq j$ or $i \geq k\}$

b. $\{a^i b^j c^k \mid i \neq j$ or $i \neq k\}$

c. $\{x \in \{a, b\}^* \mid x$ is not ww for any $w\}$

8.9. Use Ogden's lemma to show that the languages below are not CFLs.

a. $\{a^i b^{i+k} a^k \mid k \neq i\}$

b. $\{a^i b^i a^j b^j \mid j \neq i\}$

c. $\{a^i b^j a^i \mid j \neq i\}$

8.10. a. Show that if L is a CFL and F is finite, $L - F$ is a CFL.

b. Show that if L is not a CFL and F is finite, then $L - F$ is not a CFL.

c. Show that if L is not a CFL and F is finite, then $L \cup F$ is not a CFL.

8.11. For each part of the previous exercise, say whether the statement is true if "finite" is replaced by "regular," and give reasons.

8.12. For each part of Exercise 8.10, say whether the statement is true if "CFL" is replaced by "DCFL," and give reasons.

8.13. Give an example of a DPDA M accepting a language L for which the language accepted by the machine obtained from M by reversing accepting and nonaccepting states is not L'.

MORE CHALLENGING PROBLEMS

8.14. If L is a CFL, does it follow that $rev(L) = \{x^r \mid x \in L\}$ is a CFL? Give either a proof or a counterexample.

8.15. Decide in each case whether the given language is a CFL, and prove your answer.

a. $L = \{x \in \{a, b\}^* \mid n_a(x)$ is a multiple of $n_b(x)\}$

b. Given a CFG L, the set of all prefixes of elements of L

c. Given a CFG L, the set of all suffixes of elements of L

d. Given a CFG L, the set of all substrings of elements of L

e. $\{x \in \{a, b\}^* \mid |x|$ is even and the first half of x has more a's than the second$\}$

f. $\{x \in \{a, b, c\}^* \mid n_a(x), n_b(x)$, and $n_c(x)$ have a common factor greater than 1$\}$

8.16. Prove the following variation of Theorem 8.1a. If L is a CFL, then there is an integer n so that for any $u \in L$ with $|u| \geq n$, and any choice of u_1, u_2, and u_3 satisfying $u = u_1u_2u_3$ and $|u_2| \geq n$, there are strings v, w, x, y, and z satisfying the following conditions:

(1) $u = vwxyz$
(2) $wy \neq \Lambda$
(3) Either w or y is a nonempty substring of u_2
(4) For every $m \geq 0$, $vw^ixy^iz \in L$

Hint: Suppose # is a symbol not appearing in strings in L, and let L_1 be the set of all strings that can be formed by inserting two occurrences of # into an element of L. Show that L_1 is a CFL, and apply Ogden's Lemma to L_1.

This result is taken from Floyd and Beigel (1994).

8.17. Show that the result in the previous problem can be used in each part of Exercise 8.9.

8.18. Show that the result in Exercise 16 can be used in both Examples 8.5 and 8.6 to show that the language is not context-free.

8.19. The class of DCFLs is closed under the operation of complement, as discussed in Section 8.2. Under which of the following other operations is this class of languages closed? Give reasons for your answers.

a. Union

b. Intersection

c. Concatenation

d. Kleene *

e. Difference

8.20. Use Exercise 7.40 and Exercise 8.7 to show that the following languages are not DCFLs. This technique is used in Floyd and Beigel (1994), where the language in Exercise 7.40 is referred to as Double-Duty(L).

a. *pal*, the language of palindromes over {0, 1} (Hint: Consider the regular language corresponding to 0*1*0*#1*0*.)

b. $\{x \in \{a, b\}^* \mid n_b(x) = n_a(x)$ or $n_b(x) = 2n_a(x)\}$

c. $\{x \in \{a, b\}^* \mid n_b(x) < n_a(x)$ or $n_b(x) > 2n_a(x)\}$

8.21. (Refer to Exercise 7.40) Consider the following argument to show that if $L \subseteq \Sigma^*$ is a CFL, then so is $\{x\#y \mid x \in L$ and $xy \in L\}$. (# is assumed to be a symbol not in Σ.)

Let M be a PDA accepting L, with state set Q. We construct a new PDA M_1 whose state set contains two states q and q' for every state $q \in Q$. M_1 copies M up to the point where the input symbol # is encountered, but using the primed states rather than the original ones. Once this symbol is seen, if the current state is q' for some $q \in A$ (i.e., if M would have accepted the current string), then the machine switches over to the original states for the rest of the processing. Therefore, it enters an accepting state subsequently if and only if both the substring preceding the # and the entire string read so far, except for the #, would be accepted by M.

Explain why this argument is not correct.

8.22. Show that the result at the beginning of the previous exercise is false. (Find a CFL L so that $\{x\#y \mid x \in L$ and $xy \in L\}$ is not a CFL.)

8.23. Show that if $L \subseteq \{a\}^*$ is a CFL, then L is regular.

8.24. Consider the language $L = \{x \in \{a, b\}^* \mid n_a(x) = f(n_b(x))\}$. Exercise 5.52 is to show that L is regular if and only if f is ultimately periodic; in other words, L is regular if and only if there is a positive integer p so that for each r with $0 \leq r < p$, f is eventually constant on the set $S_{p,r} = \{jp + r \mid j \geq 0\}$. Show that L is a CFL if and only if there is a positive integer p so that for each r with $0 \leq r < p$, f is eventually linear on the set $S_{p,r}$. "Eventually linear" on $S_{p,r}$ means that there are integers N, c, and d so that for every $j \geq N$, $f(jp + r) = cj + d$. (Suggestion: for the "if" direction, show how to construct a PDA accepting L; for the converse, use the pumping lemma.)

8.25. Let

$$f(n) = \begin{cases} 4n + 7 & \text{if } n \text{ is even} \\ 4n + 13 & \text{if } n \text{ is odd} \end{cases}$$

a. Show that the language $\{x \in \{a, b\}^* \mid n_b(x) = f(n_a(x))\}$ is a DCFL.

b. Show that if $4n + 13$ is changed to $5n + 13$, L is not a DCFL.