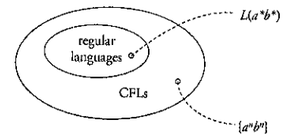
CHAPTER 14

The Context-free Frontier

At this point we have two major language categories, the regular languages and the context-free languages, and we have seen that the CFLs include the regular languages, like this:



Are there languages outside of the CFLs? In this chapter we see that the answer is yes, and we see some simple examples of languages that are not CFLs.

We have already seen that there are many closure properties for regular languages. Given any two regular languages, there are many ways to combine them—intersections, unions, and so on—that are guaranteed to produce another regular language. The context-free languages also have some closure properties, though not as many as the regular languages. If regular languages are a safe and settled territory, context-free languages are more like frontier towns. Some operations like union get you safely to another context-free language; others like complement and intersection just leave you in the wilderness.

14.1 Pumping Parse Trees

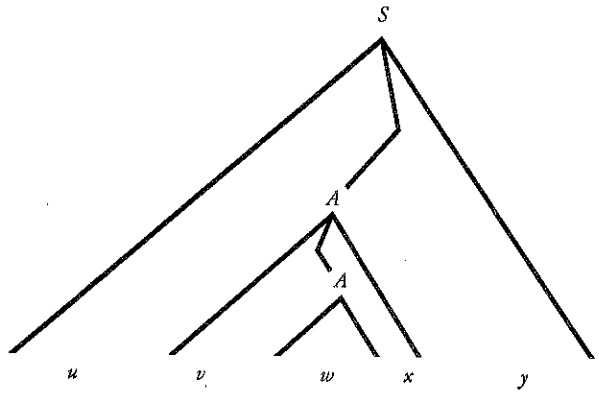
To show that there are some languages that are not regular, we reasoned that any sufficiently long string will force a DFA to repeat a state. For context-free languages there is a parallel argument, again involving repetition. We will see that we can force a parse tree to repeat a nonterminal symbol in a certain way. Such parse trees are called *pumping parse trees.*

A pumping parse tree for a CFG $G = (V, \Sigma, S, P)$ is a parse tree with two properties:

1. There is a node for some nonterminal symbol A, which has that same nonterminal symbol A as one of its descendants.
2. The terminal string generated from the ancestor A is longer than the terminal string generated from the descendant A.

If you can find a pumping parse tree for a grammar, you know a set of strings that must be in the language—not just the string the pumping parse tree yields, but a whole collection of related strings.

Lemma 14.1.1: If a grammar G generates a pumping parse tree with a yield as shown:



then $L(G)$ includes uv^iwx^iy for all i.

Proof: The string $uvwxy$ is the whole terminal string derived by the pumping parse tree. As shown, A is the nonterminal symbol that occurs as its own descendant in the tree. The string vwx is the terminal string derived from the ancestor A, and the string w is the terminal string derived from the

14.1 Pumping Parse Trees

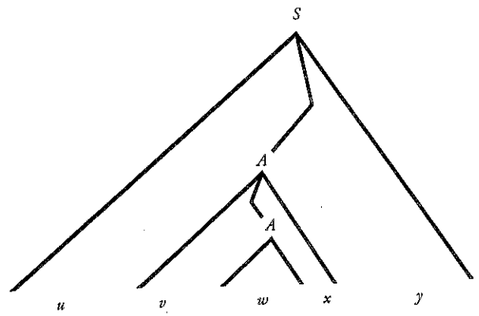
To show that there are some languages that are not regular, we reasoned that any sufficiently long string will force a DFA to repeat a state. For context-free languages there is a parallel argument, again involving repetition. We will see that we can force a parse tree to repeat a nonterminal symbol in a certain way. Such parse trees are called *pumping parse trees.*

A pumping parse tree for a CFG $G = (V, \Sigma, S, P)$ is a parse tree with two properties:

1. There is a node for some nonterminal symbol A, which has that same nonterminal symbol A as one of its descendants.
2. The terminal string generated from the ancestor A is longer than the terminal string generated from the descendant A.

If you can find a pumping parse tree for a grammar, you know a set of strings that must be in the language—not just the string the pumping parse tree yields, but a whole collection of related strings.

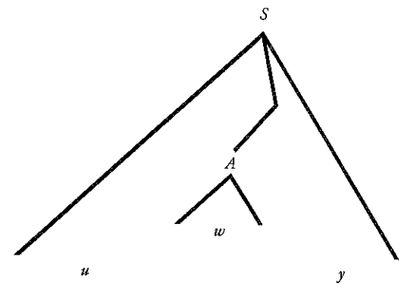
Lemma 14.1.1: If a grammar G generates a pumping parse tree with a yield as shown:



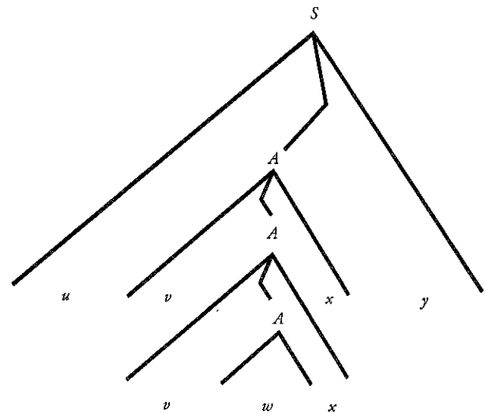
then $L(G)$ includes uv^iwx^iy for all i.

Proof: The string $uvwxy$ is the whole terminal string derived by the pumping parse tree. As shown, A is the nonterminal symbol that occurs as its own descendant in the tree. The string vwx is the terminal string derived from the ancestor A, and the string w is the terminal string derived from the

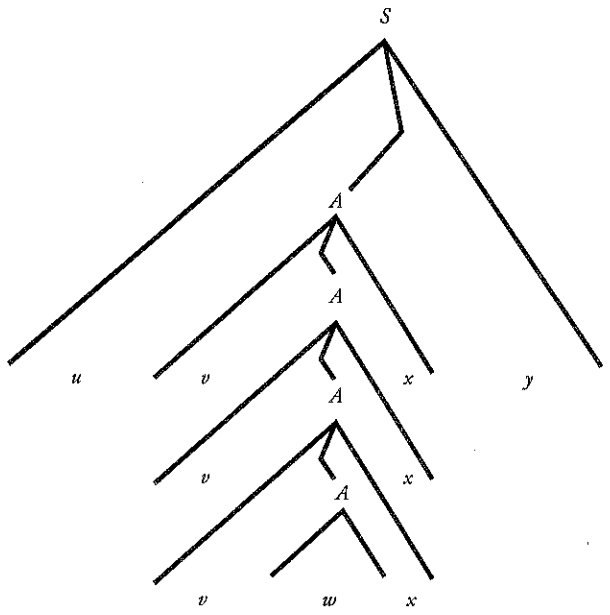
descendant A. (Satisfying the second property of pumping parse trees, v and x are not both ε.) It is the nature of context-free grammars that any A in the tree can be replaced with either of the two subtrees rooted at A, producing other legal parse trees for the grammar. For example, we could replace the vwx subtree with the w subtree, producing this parse tree for uwy:



Or we could replace the w subtree with the vwx subtree, producing this parse tree for uv^2wx^2y:



Then we could again replace the w subtree with the vwx subtree, producing this parse tree for uv^3wx^3y:



Repeating this step any number of times, we can generate a parse tree for uv^iwx^iy for any i.

The previous lemma shows that pumping parse trees are very useful—find one, and you can conclude that not only $uvwxy \in L(G)$, but $uv^iwx^iy \in L(G)$ for all i. The next lemma shows that they are not at all hard to find. To prove it, we will need to refer to the height of a parse tree, which is defined as follows:

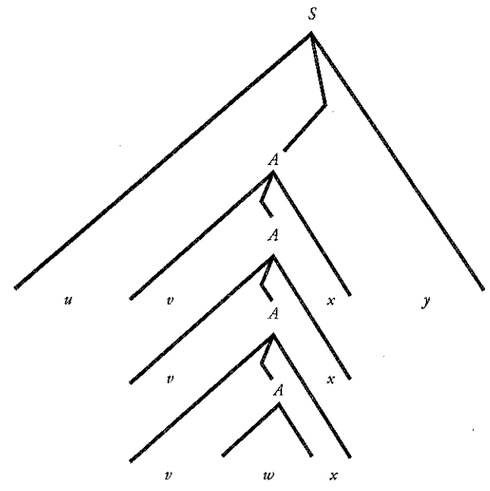
The *height* of a parse tree is the number of edges in the longest path from the start symbol to any leaf.

Consider this (ambiguous) grammar, for example:

$$S \rightarrow S \mid S + S \mid S * S \mid a \mid b \mid c$$

These are parse trees of heights 1, 2, and 3, respectively:

Then we could again replace the w subtree with the vwx subtree, producing this parse tree for uv^3wx^3y:



Repeating this step any number of times, we can generate a parse tree for uv^iwx^iy for any i.

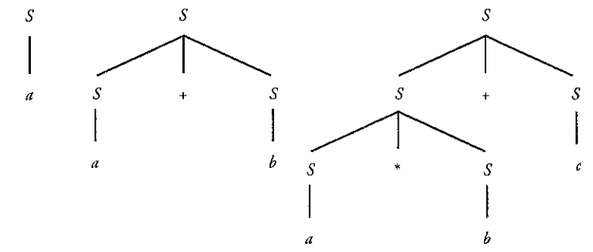
The previous lemma shows that pumping parse trees are very useful—find one, and you can conclude that not only $uvwxy \in L(G)$, but $uv^iwx^iy \in L(G)$ for all i. The next lemma shows that they are not at all hard to find. To prove it, we will need to refer to the height of a parse tree, which is defined as follows:

The *height* of a parse tree is the number of edges in the longest path from the start symbol to any leaf.

Consider this (ambiguous) grammar, for example:

$S \rightarrow S \mid S + S \mid S * S \mid a \mid b \mid c$

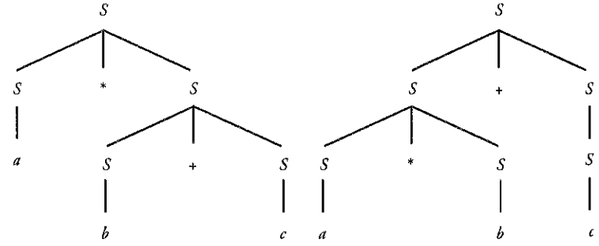
These are parse trees of heights 1, 2, and 3, respectively:



We will also need the idea of a minimum-size parse tree for a given string.

A *minimum-size* parse tree for the string x in a grammar G is a parse tree in G that generates x and has no more nodes than any other parse tree in G that generates x.

The need for this definition arises because an ambiguous grammar can generate the same string using more than one parse tree, and these parse trees might even have different numbers of nodes. For example, the previous grammar generates the string $a*b+c$ with the parse tree shown above, but also with these ones:



All three parse trees generate the same string, but the last one is not minimum size, since it has one more node than the others. Every string in the language generated by a grammar has at least one minimum-size parse tree; for an ambiguous grammar, some strings may have more than one.

Now we're ready to see why pumping parse trees are easy to find.

Lemma 14.1.2: Every CFG $G = (V, \Sigma, S, P)$ that generates an infinite language generates a pumping parse tree.

Proof: Let $G = (V, \Sigma, S, P)$ be any CFG that generates an infinite language. Each string in $L(G)$ has at least one minimum-size parse tree; therefore, G

generates infinitely many minimum-size parse trees. There are only finitely many minimum-size parse trees of height $|V|$ or less; therefore, G generates a minimum-size parse tree of height greater than $|V|$ (indeed, it generates infinitely many). Such a parse tree must satisfy property 1 of pumping parse trees, because on a path with more than $|V|$ edges there must be some nonterminal A that occurs at least twice. And such a parse tree must also satisfy property 2, because it is a minimum-size parse tree; if it did not satisfy property 2, we could replace the ancestor A with the descendant A to produce a parse tree with fewer nodes yielding the same string. Thus, G generates a pumping parse tree.

This proof actually shows that every grammar for an infinite language generates not just one, but infinitely many pumping parse trees. However, one is all we'll need for the next proof.

14.2 The Language $\{a^n b^n c^n\}$

We have seen that the language $\{a^n b^n\}$ is context free but not regular. Now we consider the language $\{a^n b^n c^n\}$: any number of as, followed by the same number of bs, followed by the same number of cs.

Theorem 14.1: $\{a^n b^n c^n\}$ is not a CFL.

Proof: Let $G = (V, \Sigma, S, P)$ be any CFG over the alphabet $\{a, b, c\}$, and suppose by way of contradiction that $L(G) = \{a^n b^n c^n\}$. By Lemma 14.1.2, G generates a pumping parse tree. Using Lemma 14.1.1, this means that for some k we have $a^k b^k c^k = uvwxy$, where v and x are not both ε and uv^2wx^2y is also in $L(G)$. The substrings v and x must each contain only as, only bs, or only cs, because otherwise uv^2wx^2y is not even in $L(a^*b^*c^*)$. But in that case uv^2wx^2y has more than k copies of one or two symbols, but only k of the third. Thus $uv^2wx^2y \notin \{a^n b^n c^n\}$. By contradiction, $L(G) \neq \{a^n b^n c^n\}$.

The tricky part of this proof is seeing that no matter how you break up a string $a^k b^k c^k$ into substrings $uvwxy$, where v and x are not both ε, you must have $uv^2wx^2y \notin \{a^n b^n c^n\}$. If the substrings v and/or x contain a mixture of symbols, as in this example:

a	a	a	a	a	b	b	b	b	b	c	c	c	c		
	u			v			w	x				y			

then the resulting string uv^2wx^2y would have as after bs and/or bs after cs; for the example above it would be *aaaaabbaabbbbbcbbccccc*, clearly not in $\{a^n b^n c^n\}$. On the other hand, if the substrings v and x contain at most one kind of symbol each, as in this example:

generates infinitely many minimum-size parse trees. There are only finitely many minimum-size parse trees of height $|V|$ or less; therefore, G generates a minimum-size parse tree of height greater than $|V|$ (indeed, it generates infinitely many). Such a parse tree must satisfy property 1 of pumping parse trees, because on a path with more than $|V|$ edges there must be some nonterminal A that occurs at least twice. And such a parse tree must also satisfy property 2, because it is a minimum-size parse tree; if it did not satisfy property 2, we could replace the ancestor A with the descendant A to produce a parse tree with fewer nodes yielding the same string. Thus, G generates a pumping parse tree.

This proof actually shows that every grammar for an infinite language generates not just one, but infinitely many pumping parse trees. However, one is all we'll need for the next proof.

14.2 The Language $\{a^n b^n c^n\}$

We have seen that the language $\{a^n b^n\}$ is context free but not regular. Now we consider the language $\{a^n b^n c^n\}$: any number of as, followed by the same number of bs, followed by the same number of cs.

Theorem 14.1: $\{a^n b^n c^n\}$ is not a CFL.

Proof: Let $G = (V, \Sigma, S, P)$ be any CFG over the alphabet $\{a, b, c\}$, and suppose by way of contradiction that $L(G) = \{a^n b^n c^n\}$. By Lemma 14.1.2, G generates a pumping parse tree. Using Lemma 14.1.1, this means that for some k we have $a^k b^k c^k = uvwxy$, where v and x are not both ε and uv^2wx^2y is also in $L(G)$. The substrings v and x must each contain only as, only bs, or only cs, because otherwise uv^2wx^2y is not even in $L(a^*b^*c^*)$. But in that case uv^2wx^2y has more than k copies of one or two symbols, but only k of the third. Thus $uv^2wx^2y \notin \{a^n b^n c^n\}$. By contradiction, $L(G) \neq \{a^n b^n c^n\}$.

The tricky part of this proof is seeing that no matter how you break up a string $a^k b^k c^k$ into substrings $uvwxy$, where v and x are not both ε, you must have $uv^2wx^2y \notin \{a^n b^n c^n\}$. If the substrings v and/or x contain a mixture of symbols, as in this example:

a	a	a	a	a	b	b	b	b	c	c	c	c
	u			v		w	x			y		

then the resulting string uv^2wx^2y would have as after bs and/or bs after cs; for the example above it would be *aaaaabbaabbbbbcbbccccc*, clearly not in $\{a^n b^n c^n\}$. On the other hand, if the substrings v and x contain at most one kind of symbol each, as in this example:

a	a	a	a	a	b	b	b	b	b	c	c	c	c	c
	u			v		w		x			y			

then the resulting string uv^2wx^2y would no longer have the same number of as, bs, and cs; for the example above it would be *aaaaaaabbbbbbccccc*, clearly not in $\{a^n b^n c^n\}$. Either way, $uv^2wx^2y \notin \{a^n b^n c^n\}$.

14.3 Closure Properties for CFLs

Context-free languages are closed for some of the same common operations as regular languages, including union, concatenation, Kleene star, and intersection with regular languages. Closure under union and concatenation can be proved using constructions that we saw, informally, back in Chapter 12.

Theorem 14.2.1: If L_1 and L_2 are any context-free languages, $L_1 \cup L_2$ is also context free.

Proof: By construction. If L_1 and L_2 are context free then, by definition, there exist grammars $G_1 = (V_1, \Sigma_1, S_1, P_1)$ and $G_2 = (V_2, \Sigma_2, S_2, P_2)$ with $L(G_1) = L_1$ and $L(G_2) = L_2$. We can assume without a loss of generality that V_1 and V_2 are disjoint. (Symbols shared between V_1 and V_2 could easily be renamed.) Now consider the grammar $G = (V, \Sigma, S, P)$, where S is a new nonterminal symbol and

$$V = V_1 \cup V_2 \cup \{S\},$$
$$\Sigma = \Sigma_1 \cup \Sigma_2, \text{ and}$$
$$P = P_1 \cup P_2 \cup \{(S \rightarrow S_1), (S \rightarrow S_2)\}$$

Clearly $L(G) = L_1 \cup L_2$, so $L_1 \cup L_2$ is a context-free language.

Theorem 14.2.2: If L_1 and L_2 are any context-free languages, $L_1 L_2$ is also context free.

Proof: By construction. If L_1 and L_2 are context free then, by definition, there exist grammars $G_1 = (V_1, \Sigma_1, S_1, P_1)$ and $G_2 = (V_2, \Sigma_2, S_2, P_2)$ with $L(G_1) = L_1$ and $L(G_2) = L_2$. We can assume without a loss of generality that V_1 and V_2 are disjoint. (Symbols shared between V_1 and V_2 could easily be renamed.) Now consider the grammar $G = (V, \Sigma, S, P)$, where S is a new nonterminal symbol and

$V = V_1 \cup V_2 \cup \{S\}$,
$\Sigma = \Sigma_1 \cup \Sigma_2$, and
$P = P_1 \cup P_2 \cup \{(S \rightarrow S_1 S_2)\}$

Clearly $L(G) = L_1 L_2$, so $L_1 L_2$ is a context-free language.

We can define the Kleene closure of a language in a way that parallels our use of the Kleene star in regular expressions:

The Kleene closure of any language L is $L^* = \{x_1 x_2 \ldots x_n \mid n \geq 0$, with all $x_i \in L\}$.

The fact that the context-free languages are closed under Kleene star can then be proved with a CFG construction, as before:

Theorem 14.2.3: If L is any context-free language, L^* is also context free.

Proof: By construction. If L is context free then, by definition, there exists a grammar $G = (V, \Sigma, S, P)$ with $L(G) = L$. Now consider the grammar $G' = (V', \Sigma, S', P')$, where S' is a new nonterminal symbol and

$V' = V \cup \{S'\}$, and
$P' = P \cup \{(S' \rightarrow SS'), (S' \rightarrow \varepsilon)\}$

Now $L(G') = L^*$, so L^* is a context-free language.

One more closure property: the CFLs are closed under intersection with regular languages:

Theorem 14.2.4: If L_1 is any context-free language and L_2 is any regular language, then $L_1 \cap L_2$ is context free.

Proof sketch: This can be proved by a (somewhat hairy) construction that combines a stack machine M_1 for L_1 with an NFA M_2 for L_2, producing a new stack machine for $L_1 \cap L_2$. It works a bit like the product construction: if M_1's stack alphabet is Γ, and M_2's state set is Q, the new stack machine can use an element of $\Gamma \times Q$ as the symbol on the top of its stack to record both the M_1's current top symbol and M_2's current state.

$V = V_1 \cup V_2 \cup \{S\}$,
$\Sigma = \Sigma_1 \cup \Sigma_2$, and
$P = P_1 \cup P_2 \cup \{(S \rightarrow S_1 S_2)\}$

Clearly $L(G) = L_1 L_2$, so $L_1 L_2$ is a context-free language.

We can define the Kleene closure of a language in a way that parallels our use of the Kleene star in regular expressions:

The Kleene closure of any language L is $L^* = \{x_1 x_2 \ldots x_n \mid n \geq 0, \text{ with all } x_i \in L\}$.

The fact that the context-free languages are closed under Kleene star can then be proved with a CFG construction, as before:

Theorem 14.2.3: If L is any context-free language, L^* is also context free.

Proof: By construction. If L is context free then, by definition, there exists a grammar $G = (V, \Sigma, S, P)$ with $L(G) = L$. Now consider the grammar $G' = (V', \Sigma, S', P')$, where S' is a new nonterminal symbol and

$V' = V \cup \{S'\}$, and
$P' = P \cup \{(S' \rightarrow SS'), (S' \rightarrow \varepsilon)\}$

Now $L(G') = L^*$, so L^* is a context-free language.

One more closure property: the CFLs are closed under intersection with regular languages:

Theorem 14.2.4: If L_1 is any context-free language and L_2 is any regular language, then $L_1 \cap L_2$ is context free.

Proof sketch: This can be proved by a (somewhat hairy) construction that combines a stack machine M_1 for L_1 with an NFA M_2 for L_2, producing a new stack machine for $L_1 \cap L_2$. It works a bit like the product construction: if M_1's stack alphabet is Γ, and M_2's state set is Q, the new stack machine can use an element of $\Gamma \times Q$ as the symbol on the top of its stack to record both the M_1's current top symbol and M_2's current state.

14.4 Nonclosure Properties

In the previous section we saw that the CFLs are closed for some of the same operations for which the regular languages are closed. Not all, however: the CFLs are not closed for intersection or complement.

Theorem 14.3.1: The CFLs are not closed for intersection.

Proof: By counterexample. Consider these two CFGs G_1 and G_2:

$S_1 \rightarrow A_1 B_1$
$A_1 \rightarrow aA_1 b \mid \varepsilon$
$B_1 \rightarrow cB_1 \mid \varepsilon$

$S_2 \rightarrow A_2 B_2$
$A_2 \rightarrow aA_2 \mid \varepsilon$
$B_2 \rightarrow bB_2 c \mid \varepsilon$

Now $L(G_1) = \{a^n b^n c^m\}$, while $L(G_2) = \{a^m b^n c^n\}$. The intersection of the two languages is $\{a^n b^n c^n\}$, which we know is not context free. Thus, the CFLs are not closed for intersection.

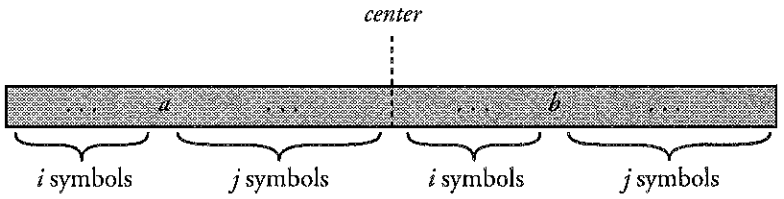
This nonclosure property does *not* mean that every intersection of CFLs fails to be a CFL. In fact, we have already seen many cases in which the intersection of two CFLs is another CFL. For example, we know that any intersection of regular languages is a regular language—and they are all CFLs. We also know that the intersection of a CFL with a regular language is a CFL. All the theorem says is that the intersection of two CFLs is *not always* a CFL. Similarly, the complement of a CFL is sometimes, but not always, another CFL:

Theorem 14.3.2: The CFLs are not closed for complement.

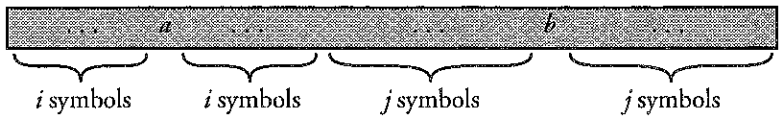
Proof 1: By Theorem 14.2.1 we know that CFLs are closed for union. Suppose by way of contradiction that they are also closed for complement. By DeMorgan's laws we have $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$. This defines intersection in terms of union and complement, so the CFLs are closed for intersection. But this is a contradiction of Theorem 14.3.1. By contradiction, we conclude that the CFLs are not closed for complement.

Proof 2: By counterexample. Let L be the language $\{xx \mid x \in \{a, b\}^*\}$, and consider its complement, $\overline{L} = \{x \in \{a, b\}^* \mid x \notin L\}$. Obviously, all odd-length

strings are in $\overline{L}$. Even-length strings are in $\overline{L}$ if and only if they have at least one symbol in the first half that is not the same as the corresponding symbol in the second half; that is, either there is an a in the first half and a b at the corresponding position in the second half, or the reverse. In the first case the string looks like this, for some i and j:



At first this looks daunting: how could a grammar generate such strings? How can we generate *waxybz*, where $|w| = |y| = i$ and $|x| = |z| = j$? It doesn't look context free, until you realize that because the x and y parts can both be any string in $\{a, b\}^*$, we can swap them, instead picturing it as:



Now we see that this actually is context free; it's $\{way \mid |w| = |y|\}$, concatenated with $\{xbz \mid |x| = |z|\}$. For the case with a b in the first half corresponding to an a in the second half, we can just swap the two parts, getting $\{xbz \mid |x| = |z|\}$ concatenated with $\{way \mid |w| = |y|\}$.

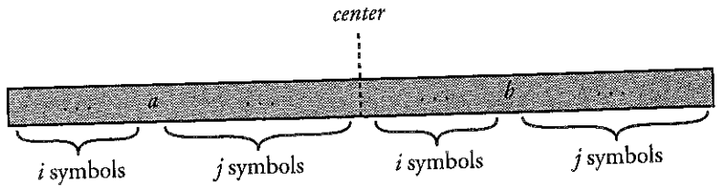
To summarize, we can express $\overline{L}$ as the union of three sets:

$$\begin{aligned}\overline{L} = \{&x \in \{a, b\}^* \mid |x| \text{ is odd}\} \\ &\cup (\{way \mid |w| = |y|\} \text{ concatenated with } \{xbz \mid |x| = |z|\}) \\ &\cup (\{xbz \mid |x| = |z|\} \text{ concatenated with } \{way \mid |w| = |y|\})\end{aligned}$$

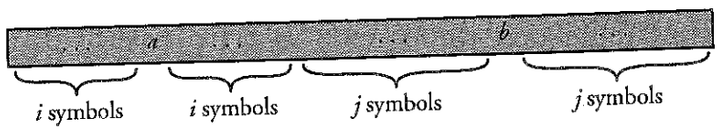
This CFG generates the language: O generates the odd-length part, and AB and BA generate the two even-length parts.

$$\begin{aligned}S &\rightarrow O \mid AB \mid BA \\ A &\rightarrow XAX \mid a\end{aligned}$$

strings are in $\overline{L}$. Even-length strings are in $\overline{L}$ if and only if they have at least one symbol in the first half that is not the same as the corresponding symbol in the second half; that is, either there is an a in the first half and a b at the corresponding position in the second half, or the reverse. In the first case the string looks like this, for some i and j:



At first this looks daunting: how could a grammar generate such strings? How can we generate $waxybz$, where $|w| = |y| = i$ and $|x| = |z| = j$? It doesn't look context free, until you realize that because the x and y parts can both be any string in $\{a, b\}^*$, we can swap them, instead picturing it as:



Now we see that this actually is context free; it's $\{way \mid |w| = |y|\}$, concatenated with $\{xbz \mid |x| = |z|\}$. For the case with a b in the first half corresponding to an a in the second half, we can just swap the two parts, getting $\{xbz \mid |x| = |z|\}$ concatenated with $\{way \mid |w| = |y|\}$.

To summarize, we can express $\overline{L}$ as the union of three sets:

$$\begin{aligned}\overline{L} = \{&x \in \{a, b\}^* \mid |x| \text{ is odd}\} \\ &\cup (\{way \mid |w| = |y|\} \text{ concatenated with } \{xbz \mid |x| = |z|\}) \\ &\cup (\{xbz \mid |x| = |z|\} \text{ concatenated with } \{way \mid |w| = |y|\})\end{aligned}$$

This CFG generates the language: O generates the odd-length part, and AB and BA generate the two even-length parts.

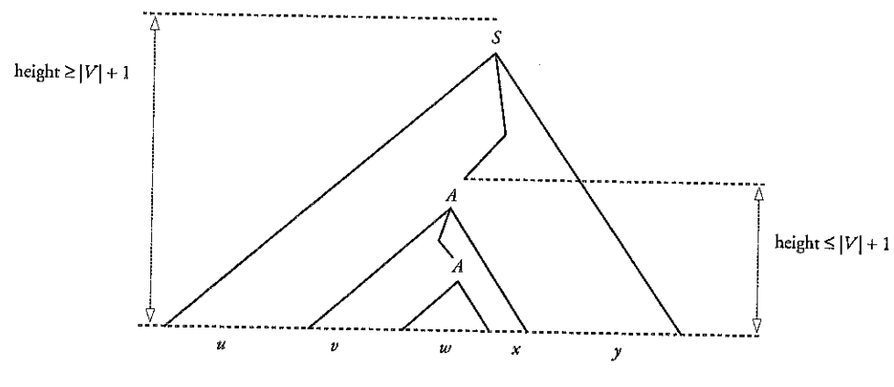
$$\begin{aligned}S &\rightarrow O \mid AB \mid BA \\ A &\rightarrow XAX \mid a \\ B &\rightarrow XBX \mid b \\ O &\rightarrow XXO \mid X \\ X &\rightarrow a \mid b\end{aligned}$$

We conclude that $\overline{L}$ is context free. But as we will show in Section 14.7, the complement of $\overline{L}$, $L = \{xx \mid x \in \{a, b\}^*\}$ is not context free. We conclude that the CFLs are not closed for complement.

14.5 A Pumping Lemma

Let's return to pumping parse trees and try to make a general-purpose pumping lemma for CFLs. We have argued that any minimum-size tree of a height greater than $|V|$ must be a pumping parse tree, since on any path of more than $|V|$ edges there must be some nonterminal A that occurs at least twice. Of course, there may be many different places where a nonterminal occurs as its own descendant. To make a useful pumping lemma for context-free languages, we need to be a little more specific about how we are dividing the tree.

Lemma 14.2.1: For every grammar $G = (V, \Sigma, S, P)$, every minimum-size parse tree of a height greater than $|V|$ can be expressed as a pumping parse tree with the properties shown:



Proof: Choose any path from root to leaf with more than $|V|$ edges. Then, working from the leaf back toward the root on this path, choose the first two nodes that repeat a nonterminal, and let these be the nodes labeled A in the diagram. As shown in the proof of Lemma 14.1.2, the result is a pumping parse tree. Furthermore, the nonterminal A must have repeated within the first $|V| + 1$ edges, so the height of the subtree generating vwx is $\leq |V| + 1$.

Being able to give a bound on the height of a tree or subtree also enables us to give a bound on the length of the string generated.

Lemma 14.2.2: For every CFG $G = (V, \Sigma, S, P)$ there exists some k greater than the length of any string generated by any parse tree or subtree of height $|V| + 1$ or less.

Proof 1: For any given grammar, there are only finitely many parse trees and subtrees of height $|V| + 1$ or less. Let k be one greater than the length of the longest string so generated.

Proof 2: Let b be the length of the longest right-hand side of any production in P. Then b is the maximum branching factor in any parse tree. So a tree or subtree of height $|V| + 1$ can have at most $b^{|V|+1}$ leaves. Let $k = b^{|V|+1} + 1$.

These two proofs give different values for k. That is not a problem, because what matters here is not the value of k, but the existence of some k with the right properties. This was also true of the value k we used in the pumping lemma for regular languages. To prove the pumping lemma, we let k be the number of states in some DFA for the language. But once the pumping lemma for regular languages was proved, the value of k used in the proof was irrelevant, since the lemma itself claims only that some suitable k exists.

Putting these lemmas together with our previous results about pumping parse trees, we get the pumping lemma for context-free languages.

Lemma 14.2.3 *(The Pumping Lemma for Context-free Languages)*: For all context-free languages L there exists some $k \in \mathcal{N}$ such that for all $z \in L$ with $|z| \geq k$, there exist $uvwxy$ such that

1. $z = uvwxy$,
2. v and x are not both ε,
3. $|vwx| \leq k$, and
4. for all i, $uv^iwx^iy \in A$.

Proof: We are given some CFL L. Let G be any CFG with $L(G) = L$, and let k be as given for this grammar by Lemma 14.2.2. We are given some $z \in L$ with $|z| \geq k$. Since all parse trees in G for z have a height greater than $|V| + 1$, that includes a minimum-size parse tree for z. We can express this as a pumping parse tree for z as given in Lemma 14.2.1. Since this is a parse tree for z, property 1 is satisfied; since it is a pumping parse tree, properties 2 and 4 are satisfied; and since the subtree generating vwx is of height $|V| + 1$ or less, property 3 is satisfied.

Being able to give a bound on the height of a tree or subtree also enables us to give a bound on the length of the string generated.

Lemma 14.2.2: For every CFG $G = (V, \Sigma, S, P)$ there exists some k greater than the length of any string generated by any parse tree or subtree of height $|V| + 1$ or less.

Proof 1: For any given grammar, there are only finitely many parse trees and subtrees of height $|V| + 1$ or less. Let k be one greater than the length of the longest string so generated.

Proof 2: Let b be the length of the longest right-hand side of any production in P. Then b is the maximum branching factor in any parse tree. So a tree or subtree of height $|V| + 1$ can have at most $b^{|V|+1}$ leaves. Let $k = b^{|V|+1} + 1$.

These two proofs give different values for k. That is not a problem, because what matters here is not the value of k, but the existence of some k with the right properties. This was also true of the value k we used in the pumping lemma for regular languages. To prove the pumping lemma, we let k be the number of states in some DFA for the language. But once the pumping lemma for regular languages was proved, the value of k used in the proof was irrelevant, since the lemma itself claims only that some suitable k exists.

Putting these lemmas together with our previous results about pumping parse trees, we get the pumping lemma for context-free languages.

Lemma 14.2.3 *(The Pumping Lemma for Context-free Languages)*: For all context-free languages L there exists some $k \in \mathcal{N}$ such that for all $z \in L$ with $|z| \geq k$, there exist $uvwxy$ such that

1. $z = uvwxy$,
2. v and x are not both ε,
3. $|vwx| \leq k$, and
4. for all i, $uv^iwx^iy \in A$.

Proof: We are given some CFL L. Let G be any CFG with $L(G) = L$, and let k be as given for this grammar by Lemma 14.2.2. We are given some $z \in L$ with $|z| \geq k$. Since all parse trees in G for z have a height greater than $|V| + 1$, that includes a minimum-size parse tree for z. We can express this as a pumping parse tree for z as given in Lemma 14.2.1. Since this is a parse tree for z, property 1 is satisfied; since it is a pumping parse tree, properties 2 and 4 are satisfied; and since the subtree generating vwx is of height $|V| + 1$ or less, property 3 is satisfied.

This pumping lemma shows once again how *matching pairs* are fundamental to the context-free languages. Every sufficiently long string in a context-free language contains a matching pair consisting of the two substrings v and x of the lemma. These substrings can be pumped in tandem, always producing another string uv^iwx^iy in the language. Of course, one or the other (but not both) of those substrings may be ε. In that way the pumping lemma for context-free languages generalizes the one for regular languages, since if one of the strings is ε, the other can be pumped by itself.

14.6 Pumping-Lemma Proofs

The pumping lemma is very useful for proving that languages are not context free. For example, here is a pumping-lemma proof showing that $\{a^nb^nc^n\}$ is not context free (a fact we already proved without using the pumping lemma). The steps in the proof are numbered for future reference.

1. Proof is by contradiction using the pumping lemma for context-free languages. Assume that $L = \{a^nb^nc^n\}$ is context free, so the pumping lemma holds for L. Let k be as given by the pumping lemma.
2. Choose $z = a^kb^kc^k$. Now $z \in L$ and $|z| \geq k$ as required.
3. Let u, v, w, x, *and* y be as given by the pumping lemma, so that $uvwxy = a^kb^kc^k$, v and x are not both ε, $|vwx| \leq k$, and for all i, $uv^iwx^iy \in L$.
4. Now consider pumping with $i = 2$. The substrings v and x cannot contain more than one kind of symbol each—otherwise the string uv^2wx^2y would not even be in $L(a^*b^*c^*)$. So the substrings v and x must fall within the string $a^kb^kc^k$ in one of these ways:

	a^k	b^k	c^k
1.	v x		
2.	v	x	
3.		v x	
4.		v	x
5.			v x
6.	v		x

   But in all these cases, since v and x are not both ε, pumping changes the number of one or two of the symbols, but not all three. So $uv^2wx^2y \notin L$.
5. This contradicts the pumping lemma. By contradiction, $L = \{a^nb^nc^n\}$ is not context free.

The structure of this proof matches the structure of the pumping lemma itself. Here is the overall structure of the pumping lemma, using the symbols $\forall$ ("for all") and $\exists$ ("there exists"):

1. $\exists\, k$ ...
2. $\forall\, z$ ...
3. $\exists\, uvwxy$ ...
4. $\forall\, i \geq 0$ ...

The alternating $\forall$ and $\exists$ parts make every pumping-lemma proof into a kind of game, just as we saw using the pumping lemma for regular languages. The $\exists$ parts (the natural number k and the strings u, v, w, x, and y) are merely guaranteed to exist. In effect, the pumping lemma itself makes these moves, choosing k, u, v, w, x, and y any way it wants. In steps 1 and 3 we could only say these values are "as given by the pumping lemma." The $\forall$ parts, on the other hand, are the moves you get to make: you can choose any values for z and i, since the lemma holds for all such values. This pumping lemma offers fewer choices than the pumping lemma for regular languages, and that actually makes the proofs a little harder to do. The final step, showing that a contradiction is reached, is often more difficult, because one has less control early in the proof.

You may have noticed that in that table in the proof—the one that shows how the strings v and x can fall within the string $a^k b^k c^k$—line 6 is unnecessary. It does not hurt to include it in the proof, because all six cases lead to the same contradiction. But the substrings v and x actually cannot be as far apart as shown in line 6, because that would make the combined substring vwx more than k symbols long, while the pumping lemma guarantees that $|vwx| \leq k$. In many applications of the pumping lemma it is necessary to make use of this. The following pumping-lemma proof is an example:

Theorem 14.4: $L = \{a^n b^m c^n \mid m \leq n\}$ is not context free.

Proof: By contradiction using the pumping lemma for context-free languages. Assume that $L = \{a^n b^m c^n \mid m \leq n\}$ is context free, so the pumping lemma holds for L. Let k be as given by the pumping lemma. Choose $z = a^k b^k c^k$. Now $z \in L$ and $|z| \geq k$ as required. Let u, v, w, x, and y be as given by the pumping lemma, so that $uvwxy = a^k b^k c^k$, v and x are not both ε, $|vwx| \leq k$, and for all i, $uv^i wx^i y \in L$.

Now consider pumping with $i = 2$. The substrings v and x cannot contain more than one kind of symbol each—otherwise the string uv^2wx^2y would not even be in $L(a^*b^*c^*)$. So the substrings v and x must fall within the string $a^k b^k c^k$ in one of these ways:

1. $\exists\, k$ ...
2. $\forall\, z$ ...
3. $\exists\, uvwxy$ ...
4. $\forall\, i \geq 0$ ...

The alternating $\forall$ and $\exists$ parts make every pumping-lemma proof into a kind of game, just as we saw using the pumping lemma for regular languages. The $\exists$ parts (the natural number k and the strings u, v, w, x, and y) are merely guaranteed to exist. In effect, the pumping lemma itself makes these moves, choosing k, u, v, w, x, and y any way it wants. In steps 1 and 3 we could only say these values are "as given by the pumping lemma." The $\forall$ parts, on the other hand, are the moves you get to make: you can choose any values for z and i, since the lemma holds for all such values. This pumping lemma offers fewer choices than the pumping lemma for regular languages, and that actually makes the proofs a little harder to do. The final step, showing that a contradiction is reached, is often more difficult, because one has less control early in the proof.

You may have noticed that in that table in the proof—the one that shows how the strings v and x can fall within the string $a^k b^k c^k$—line 6 is unnecessary. It does not hurt to include it in the proof, because all six cases lead to the same contradiction. But the substrings v and x actually cannot be as far apart as shown in line 6, because that would make the combined substring vwx more than k symbols long, while the pumping lemma guarantees that $|vwx| \leq k$. In many applications of the pumping lemma it is necessary to make use of this. The following pumping-lemma proof is an example:

Theorem 14.4: $L = \{a^n b^m c^n \mid m \leq n\}$ is not context free.

Proof: By contradiction using the pumping lemma for context-free languages. Assume that $L = \{a^n b^m c^n \mid m \leq n\}$ is context free, so the pumping lemma holds for L. Let k be as given by the pumping lemma. Choose $z = a^k b^k c^k$. Now $z \in L$ and $|z| \geq k$ as required. Let u, v, w, x, and y be as given by the pumping lemma, so that $uvwxy = a^k b^k c^k$, v and x are not both ε, $|vwx| \leq k$, and for all i, $uv^i wx^i y \in L$.

Now consider pumping with $i = 2$. The substrings v and x cannot contain more than one kind of symbol each—otherwise the string uv^2wx^2y would not even be in $L(a^*b^*c^*)$. So the substrings v and x must fall within the string $a^k b^k c^k$ in one of these ways:

	a^k	b^k	c^k
1.	v x		
2.	v	x	
3.		v x	
4.		v	x
5.			v x
6.	v		x

Because v and x are not both ε, uv^2wx^2y in case 1 has more as than cs. In case 2 there are either more as than cs, more bs than cs, or both. In case 3 there are more bs than as and cs. In case 4 there are more cs than as, more bs than as, or both. In case 5 there are more cs than as. Case 6 contradicts $|vwx| \leq k$. Thus, all cases contradict the pumping lemma. By contradiction, $L = \{a^n b^m c^n \mid m \leq n\}$ is not context free.

This proof is very similar to the previous one, using the same string z and pumping count i. This time, however, we needed to use the fact that $|vwx| \leq k$ to reach a contradiction in case 6. Without that, no contradiction would be reached, since pumping v and x together could make the number of as and the number of cs grow in step, which would still be a string in the language L.

14.7 The Language {xx}

Consider the language of strings that consist of any string over a given alphabet Σ, followed by another copy of the same string: $\{xx \mid x \in \Sigma^*\}$. For $\Sigma = \{a, b\}$, the language contains the strings ε, *aa, bb, abab, baba, aaaa, bbbb*, and so on. We have seen that the related language $\{xx^R\}$ is context free, though not regular for any alphabet with at least two symbols.

Theorem 14.5: $\{xx \mid x \in \Sigma^*\}$ is not a CFL for any alphabet Σ with at least two symbols.

Proof: By contradiction using the pumping lemma for context-free languages. Let Σ be any alphabet containing at least two symbols (which we will refer to as a and b). Assume that $L = \{xx \mid x \in \Sigma^*\}$ is context free, so the pumping lemma holds for L. Let k be as given by the pumping lemma. Choose $z = a^k b^k a^k b^k$. Now $z \in L$ and $|z| \geq k$ as required. Let u, v, w, x, and y be as given by the pumping lemma, so that $uvwxy = a^k b^k a^k b^k$, v and x are not both ε, $|vwx| \leq k$, and for all i, $uv^i wx^i y \in L$.