Chapter 5

Context-Free Grammars and Languages

We now turn our attention away from the regular languages to a larger class of languages, called the "context-free languages." These languages have a natural, recursive notation, called "context-free grammars." Context-free grammars have played a central role in compiler technology since the 1960's; they turned the implementation of parsers (functions that discover the structure of a program) from a time-consuming, ad-hoc implementation task into a routine job that can be done in an afternoon. More recently, the context-free grammar has been used to describe document formats, via the so-called document-type definition (DTD) that is used in the XML (extensible markup language) community for information exchange on the Web.

In this chapter, we introduce the context-free grammar notation, and show how grammars define languages. We discuss the "parse tree," a picture of the structure that a grammar places on the strings of its language. The parse tree is the product of a parser for a programming language and is the way that the structure of programs is normally captured.

There is an automaton-like notation, called the "pushdown automaton," that also describes all and only the context-free languages; we introduce the pushdown automaton in Chapter 6. While less important than finite automata, we shall find the pushdown automaton, especially its equivalence to context-free grammars as a language-defining mechanism, to be quite useful when we explore the closure and decision properties of the context-free languages in Chapter 7.

5.1 Context-Free Grammars

We shall begin by introducing the context-free grammar notation informally. After seeing some of the important capabilities of these grammars, we offer formal definitions. We show how to define a grammar formally, and introduce the process of "derivation," whereby it is determined which strings are in the language of the grammar.

5.1.1 An Informal Example

Let us consider the language of palindromes. A *palindrome* is a string that reads the same forward and backward, such as `otto` or `madamimadam` ("Madam, I'm Adam," allegedly the first thing Eve heard in the Garden of Eden). Put another way, string w is a palindrome if and only if $w = w^R$. To make things simple, we shall consider describing only the palindromes with alphabet $\{0, 1\}$. This language includes strings like 0110, 11011, and ϵ, but not 011 or 0101.

It is easy to verify that the language L_{pal} of palindromes of 0's and 1's is not a regular language. To do so, we use the pumping lemma. If L_{pal} is a regular language, let n be the associated constant, and consider the palindrome $w = 0^n10^n$. If L_{pal} is regular, then we can break w into $w = xyz$, such that y consists of one or more 0's from the first group. Thus, xz, which would also have to be in L_{pal} if L_{pal} were regular, would have fewer 0's to the left of the lone 1 than there are to the right of the 1. Therefore xz cannot be a palindrome. We have now contradicted the assumption that L_{pal} is a regular language.

There is a natural, recursive definition of when a string of 0's and 1's is in L_{pal}. It starts with a basis saying that a few obvious strings are in L_{pal}, and then exploits the idea that if a string is a palindrome, it must begin and end with the same symbol. Further, when the first and last symbols are removed, the resulting string must also be a palindrome. That is:

BASIS: ϵ, 0, and 1 are palindromes.

INDUCTION: If w is a palindrome, so are $0w0$ and $1w1$. No string is a palindrome of 0's and 1's, unless it follows from this basis and induction rule.

A context-free grammar is a formal notation for expressing such recursive definitions of languages. A grammar consists of one or more variables that represent classes of strings, i.e., languages. In this example we have need for only one variable P, which represents the set of palindromes; that is the class of strings forming the language L_{pal}. There are rules that say how the strings in each class are constructed. The construction can use symbols of the alphabet, strings that are already known to be in one of the classes, or both.

Example 5.1 : The rules that define the palindromes, expressed in the context-free grammar notation, are shown in Fig. 5.1. We shall see in Section 5.1.2 what the rules mean.

The first three rules form the basis. They tell us that the class of palindromes includes the strings ϵ, 0, and 1. None of the right sides of these rules (the portions following the arrows) contains a variable, which is why they form a basis for the definition.

The last two rules form the inductive part of the definition. For instance, rule 4 says that if we take any string w from the class P, then $0w0$ is also in class P. Rule 5 likewise tells us that $1w1$ is also in P. □

1.	P	$\rightarrow$	ϵ
2.	P	$\rightarrow$	0
3.	P	$\rightarrow$	1
4.	P	$\rightarrow$	$0P0$
5.	P	$\rightarrow$	$1P1$

Figure 5.1: A context-free grammar for palindromes

5.1.2 Definition of Context-Free Grammars

There are four important components in a grammatical description of a language:

1. There is a finite set of symbols that form the strings of the language being defined. This set was $\{0, 1\}$ in the palindrome example we just saw. We call this alphabet the *terminals*, or *terminal symbols*.

2. There is a finite set of *variables*, also called sometimes *nonterminals* or *syntactic categories*. Each variable represents a language; i.e., a set of strings. In our example above, there was only one variable, P, which we used to represent the class of palindromes over alphabet $\{0, 1\}$.

3. One of the variables represents the language being defined; it is called the *start symbol*. Other variables represent auxiliary classes of strings that are used to help define the language of the start symbol. In our example, P, the only variable, is the start symbol.

4. There is a finite set of *productions* or *rules* that represent the recursive definition of a language. Each production consists of:

   (a) A variable that is being (partially) defined by the production. This variable is often called the *head* of the production.

   (b) The production symbol $\rightarrow$.

   (c) A string of zero or more terminals and variables. This string, called the *body* of the production, represents one way to form strings in the language of the variable of the head. In so doing, we leave terminals unchanged and substitute for each variable of the body any string that is known to be in the language of that variable.

   We saw an example of productions in Fig. 5.1.

The four components just described form a *context-free grammar*, or just *grammar*, or *CFG*. We shall represent a CFG G by its four components, that is, $G = (V, T, P, S)$, where V is the set of variables, T the terminals, P the set of productions, and S the start symbol.

Example 5.2 : The grammar G_{pal} for the palindromes is represented by

$$G_{pal} = (\{P\}, \{0,1\}, A, P)$$

where A represents the set of five productions that we saw in Fig. 5.1. $\square$

Example 5.3 : Let us explore a more complex CFG that represents (a simplification of) expressions in a typical programming language. First, we shall limit ourselves to the operators $+$ and $*$, representing addition and multiplication. We shall allow arguments to be identifiers, but instead of allowing the full set of typical identifiers (letters followed by zero or more letters and digits), we shall allow only the letters a and b and the digits 0 and 1. Every identifier must begin with a or b, which may be followed by any string in $\{a, b, 0, 1\}^*$.

We need two variables in this grammar. One, which we call E, represents expressions. It is the start symbol and represents the language of expressions we are defining. The other variable, I, represents identifiers. Its language is actually regular; it is the language of the regular expression

$$(\mathbf{a}+\mathbf{b})(\mathbf{a}+\mathbf{b}+\mathbf{0}+\mathbf{1})^*$$

However, we shall not use regular expressions directly in grammars. Rather, we use a set of productions that say essentially the same thing as this regular expression.

$$\begin{array}{lllll}
1. & E & \rightarrow & I \\
2. & E & \rightarrow & E+E \\
3. & E & \rightarrow & E*E \\
4. & E & \rightarrow & (E) \\
\\
5. & I & \rightarrow & a \\
6. & I & \rightarrow & b \\
7. & I & \rightarrow & Ia \\
8. & I & \rightarrow & Ib \\
9. & I & \rightarrow & I0 \\
10. & I & \rightarrow & I1
\end{array}$$

Figure 5.2: A context-free grammar for simple expressions

The grammar for expressions is stated formally as $G = (\{E, I\}, T, P, E)$, where T is the set of symbols $\{+, *, (,), a, b, 0, 1\}$ and P is the set of productions shown in Fig. 5.2. We interpret the productions as follows.

Rule (1) is the basis rule for expressions. It says that an expression can be a single identifier. Rules (2) through (4) describe the inductive case for expressions. Rule (2) says that an expression can be two expressions connected by a plus sign; rule (3) says the same with a multiplication sign. Rule (4) says

Compact Notation for Productions

It is convenient to think of a production as "belonging" to the variable of its head. We shall often use remarks like "the productions for A" or "A-productions" to refer to the productions whose head is variable A. We may write the productions for a grammar by listing each variable once, and then listing all the bodies of the productions for that variable, separated by vertical bars. That is, the productions $A \to \alpha_1,\ A \to \alpha_2, \ldots, A \to \alpha_n$ can be replaced by the notation $A \to \alpha_1 | \alpha_2 | \cdots | \alpha_n$. For instance, the grammar for palindromes from Fig. 5.1 can be written as $P \to \epsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$.

that if we take any expression and put matching parentheses around it, the result is also an expression.

Rules (5) through (10) describe identifiers I. The basis is rules (5) and (6); they say that a and b are identifiers. The remaining four rules are the inductive case. They say that if we have any identifier, we can follow it by a, b, 0, or 1, and the result will be another identifier. □

5.1.3 Derivations Using a Grammar

We apply the productions of a CFG to infer that certain strings are in the language of a certain variable. There are two approaches to this inference. The more conventional approach is to use the rules from body to head. That is, we take strings known to be in the language of each of the variables of the body, concatenate them, in the proper order, with any terminals appearing in the body, and infer that the resulting string is in the language of the variable in the head. We shall refer to this procedure as *recursive inference*.

There is another approach to defining the language of a grammar, in which we use the productions from head to body. We expand the start symbol using one of its productions (i.e., using a production whose head is the start symbol). We further expand the resulting string by replacing one of the variables by the body of one of its productions, and so on, until we derive a string consisting entirely of terminals. The language of the grammar is all strings of terminals that we can obtain in this way. This use of grammars is called *derivation*.

We shall begin with an example of the first approach — recursive inference. However, it is often more natural to think of grammars as used in derivations, and we shall next develop the notation for describing these derivations.

Example 5.4: Let us consider some of the inferences we can make using the grammar for expressions in Fig. 5.2. Figure 5.3 summarizes these inferences. For example, line (i) says that we can infer string a is in the language for I by using production 5. Lines (ii) through (iv) say we can infer that $b00$

is an identifier by using production 6 once (to get the b) and then applying production 9 twice (to attach the two 0's).

	String Inferred	For language of	Production used	String(s) used
(i)	a	I	5	—
(ii)	b	I	6	—
(iii)	$b0$	I	9	(ii)
(iv)	$b00$	I	9	(iii)
(v)	a	E	1	(i)
(vi)	$b00$	E	1	(iv)
(vii)	$a + b00$	E	2	$(v), (vi)$
$(viii)$	$(a + b00)$	E	4	(vii)
(ix)	$a * (a + b00)$	E	3	$(v), (viii)$

Figure 5.3: Inferring strings using the grammar of Fig. 5.2

Lines (v) and (vi) exploit production 1 to infer that, since any identifier is an expression, the strings a and $b00$, which we inferred in lines (i) and (iv) to be identifiers, are also in the language of variable E. Line (vii) uses production 2 to infer that the sum of these identifiers is an expression; line $(viii)$ uses production 4 to infer that the same string with parentheses around it is also an expression, and line (ix) uses production 3 to multiply the identifier a by the expression we had discovered in line $(viii)$. $\Box$

The process of deriving strings by applying productions from head to body requires the definition of a new relation symbol $\Rightarrow$. Suppose $G = (V, T, P, S)$ is a CFG. Let $\alpha A \beta$ be a string of terminals and variables, with A a variable. That is, α and β are strings in $(V \cup T)^*$, and A is in V. Let $A \rightarrow \gamma$ be a production of G. Then we say $\alpha A \beta \underset{G}{\Rightarrow} \alpha\gamma\beta$. If G is understood, we just say $\alpha A \beta \Rightarrow \alpha\gamma\beta$. Notice that one derivation step replaces any variable anywhere in the string by the body of one of its productions.

We may extend the $\Rightarrow$ relationship to represent zero, one, or many derivation steps, much as the transition function δ of a finite automaton was extended to $\hat{\delta}$. For derivations, we use a * to denote "zero or more steps," as follows:

BASIS: For any string α of terminals and variables, we say $\alpha \overset{*}{\underset{G}{\Rightarrow}} \alpha$. That is, any string derives itself.

INDUCTION: If $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$ and $\beta \underset{G}{\Rightarrow} \gamma$, then $\alpha \overset{*}{\underset{G}{\Rightarrow}} \gamma$. That is, if α can become β by zero or more steps, and one more step takes β to γ, then α can become γ. Put another way, $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$ means that there is a sequence of strings $\gamma_1, \gamma_2, \ldots, \gamma_n$, for some $n \geq 1$, such that

1. $\alpha = \gamma_1$,

2. $\beta = \gamma_n$, and

3. For $i = 1, 2, \ldots, n-1$, we have $\gamma_i \Rightarrow \gamma_{i+1}$.

If grammar G is understood, then we use $\overset{*}{\Rightarrow}$ in place of $\underset{G}{\overset{*}{\Rightarrow}}$.

Example 5.5 : The inference that $a * (a + b00)$ is in the language of variable E can be reflected in a derivation of that string, starting with the string E. Here is one such derivation:

$$E \Rightarrow E * E \Rightarrow I * E \Rightarrow a * E \Rightarrow$$

$$a * (E) \Rightarrow a * (E + E) \Rightarrow a * (I + E) \Rightarrow a * (a + E) \Rightarrow$$

$$a * (a + I) \Rightarrow a * (a + I0) \Rightarrow a * (a + I00) \Rightarrow a * (a + b00)$$

At the first step, E is replaced by the body of production 3 (from Fig. 5.2). At the second step, production 1 is used to replace the first E by I, and so on. Notice that we have systematically adopted the policy of always replacing the leftmost variable in the string. However, at each step we may choose which variable to replace, and we can use any of the productions for that variable. For instance, at the second step, we could have replaced the second E by (E), using production 4. In that case, we would say $E * E \Rightarrow E * (E)$. We could also have chosen to make a replacement that would fail to lead to the same string of terminals. A simple example would be if we used production 2 at the first step, and said $E \Rightarrow E + E$. No replacements for the two E's could ever turn $E + E$ into $a * (a + b00)$.

We can use the $\overset{*}{\Rightarrow}$ relationship to condense the derivation. We know $E \overset{*}{\Rightarrow} E$ by the basis. Repeated use of the inductive part gives us $E \overset{*}{\Rightarrow} E*E$, $E \overset{*}{\Rightarrow} I*E$, and so on, until finally $E \overset{*}{\Rightarrow} a * (a + b00)$.

The two viewpoints — recursive inference and derivation — are equivalent. That is, a string of terminals w is inferred to be in the language of some variable A if and only if $A \overset{*}{\Rightarrow} w$. However, the proof of this fact requires some work, and we leave it to Section 5.2. $\Box$

5.1.4 Leftmost and Rightmost Derivations

In order to restrict the number of choices we have in deriving a string, it is often useful to require that at each step we replace the leftmost variable by one of its production bodies. Such a derivation is called a *leftmost derivation*, and we indicate that a derivation is leftmost by using the relations $\underset{lm}{\Rightarrow}$ and $\underset{lm}{\overset{*}{\Rightarrow}}$, for one or many steps, respectively. If the grammar G that is being used is not obvious, we can place the name G below the arrow in either of these symbols.

Similarly, it is possible to require that at each step the rightmost variable is replaced by one of its bodies. If so, we call the derivation *rightmost* and use

Notation for CFG Derivations

There are a number of conventions in common use that help us remember the role of the symbols we use when discussing CFG's. Here are the conventions we shall use:

1. Lower-case letters near the beginning of the alphabet, a, b, and so on, are terminal symbols. We shall also assume that digits and other characters such as + or parentheses are terminals.

2. Upper-case letters near the beginning of the alphabet, A, B, and so on, are variables.

3. Lower-case letters near the end of the alphabet, such as w or z, are strings of terminals. This convention reminds us that the terminals are analogous to the input symbols of an automaton.

4. Upper-case letters near the end of the alphabet, such as X or Y, are either terminals or variables.

5. Lower-case Greek letters, such as α and β, are strings consisting of terminals and/or variables.

There is no special notation for strings that consist of variables only, since this concept plays no important role. However, a string named α or another Greek letter might happen to have only variables.

the symbols $\underset{rm}{\Rightarrow}$ and $\overset{*}{\underset{rm}{\Rightarrow}}$ to indicate one or many rightmost derivation steps, respectively. Again, the name of the grammar may appear below these symbols if it is not clear which grammar is being used.

Example 5.6: The derivation of Example 5.5 was actually a leftmost derivation. Thus, we can describe the same derivation by:

$$E \underset{lm}{\Rightarrow} E * E \underset{lm}{\Rightarrow} I * E \underset{lm}{\Rightarrow} a * E \underset{lm}{\Rightarrow}$$

$$a * (E) \underset{lm}{\Rightarrow} a * (E + E) \underset{lm}{\Rightarrow} a * (I + E) \underset{lm}{\Rightarrow} a * (a + E) \underset{lm}{\Rightarrow}$$

$$a * (a + I) \underset{lm}{\Rightarrow} a * (a + I0) \underset{lm}{\Rightarrow} a * (a + I00) \underset{lm}{\Rightarrow} a * (a + b00)$$

We can also summarize the leftmost derivation by saying $E \overset{*}{\underset{lm}{\Rightarrow}} a * (a + b00)$, or express several steps of the derivation by expressions such as $E * E \overset{*}{\underset{lm}{\Rightarrow}} a * (E)$.

There is a rightmost derivation that uses the same replacements for each variable, although it makes the replacements in different order. This rightmost derivation is:

$$E \underset{rm}{\Rightarrow} E * E \underset{rm}{\Rightarrow} E * (E) \underset{rm}{\Rightarrow} E * (E + E) \underset{rm}{\Rightarrow}$$

$$E * (E + I) \underset{rm}{\Rightarrow} E * (E + I0) \underset{rm}{\Rightarrow} E * (E + I00) \underset{rm}{\Rightarrow} E * (E + b00) \underset{rm}{\Rightarrow}$$

$$E * (I + b00) \underset{rm}{\Rightarrow} E * (a + b00) \underset{rm}{\Rightarrow} I * (a + b00) \underset{rm}{\Rightarrow} a * (a + b00)$$

This derivation allows us to conclude $E \overset{*}{\underset{rm}{\Rightarrow}} a * (a + b00)$. □

Any derivation has an equivalent leftmost and an equivalent rightmost derivation. That is, if w is a terminal string, and A a variable, then $A \overset{*}{\Rightarrow} w$ if and only if $A \overset{*}{\underset{lm}{\Rightarrow}} w$, and $A \overset{*}{\Rightarrow} w$ if and only if $A \overset{*}{\underset{rm}{\Rightarrow}} w$. We shall also prove these claims in Section 5.2.

5.1.5 The Language of a Grammar

If $G = (V, T, P, S)$ is a CFG, the *language* of G, denoted $L(G)$, is the set of terminal strings that have derivations from the start symbol. That is,

$$L(G) = \{w \text{ in } T^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w\}$$

If a language L is the language of some context-free grammar, then L is said to be a *context-free language*, or CFL. For instance, we asserted that the grammar of Fig. 5.1 defined the language of palindromes over alphabet $\{0, 1\}$. Thus, the set of palindromes is a context-free language. We can prove that statement, as follows.

Theorem 5.7: $L(G_{pal})$, where G_{pal} is the grammar of Example 5.1, is the set of palindromes over $\{0, 1\}$.

PROOF: We shall prove that a string w in $\{0, 1\}^*$ is in $L(G_{pal})$ if and only if it is a palindrome; i.e., $w = w^R$.

(If) Suppose w is a palindrome. We show by induction on $|w|$ that w is in $L(G_{pal})$.

BASIS: We use lengths 0 and 1 as the basis. If $|w| = 0$ or $|w| = 1$, then w is ϵ, 0, or 1. Since there are productions $P \to \epsilon$, $P \to 0$, and $P \to 1$, we conclude that $P \overset{*}{\Rightarrow} w$ in any of these basis cases.

INDUCTION: Suppose $|w| \geq 2$. Since $w = w^R$, w must begin and end with the same symbol That is, $w = 0x0$ or $w = 1x1$. Moreover, x must be a palindrome; that is, $x = x^R$. Note that we need the fact that $|w| \geq 2$ to infer that there are two distinct 0's or 1's, at either end of w.

If $w = 0x0$, then we invoke the inductive hypothesis to claim that $P \overset{*}{\Rightarrow} x$. Then there is a derivation of w from P, namely $P \Rightarrow 0P0 \overset{*}{\Rightarrow} 0x0 = w$. If $w = 1x1$, the argument is the same, but we use the production $P \to 1P1$ at the first step. In either case, we conclude that w is in $L(G_{pal})$ and complete the proof.

(Only-if) Now, we assume that w is in $L(G_{pal})$; that is, $P \overset{*}{\Rightarrow} w$. We must conclude that w is a palindrome. The proof is an induction on the number of steps in a derivation of w from P.

BASIS: If the derivation is one step, then it must use one of the three productions that do not have P in the body. That is, the derivation is $P \Rightarrow \epsilon$, $P \Rightarrow 0$, or $P \Rightarrow 1$. Since ϵ, 0, and 1 are all palindromes, the basis is proven.

INDUCTION: Now, suppose that the derivation takes $n+1$ steps, where $n \geq 1$, and the statement is true for all derivations of n steps. That is, if $P \overset{*}{\Rightarrow} x$ in n steps, then x is a palindrome.

Consider an $(n+1)$-step derivation of w, which must be of the form

$$P \Rightarrow 0P0 \overset{*}{\Rightarrow} 0x0 = w$$

or $P \Rightarrow 1P1 \overset{*}{\Rightarrow} 1x1 = w$, since $n + 1$ steps is at least two steps, and the productions $P \to 0P0$ and $P \to 1P1$ are the only productions whose use allows additional steps of a derivation. Note that in either case, $P \overset{*}{\Rightarrow} x$ in n steps.

By the inductive hypothesis, we know that x is a palindrome; that is, $x = x^R$. But if so, then $0x0$ and $1x1$ are also palindromes. For instance, $(0x0)^R = 0x^R0 = 0x0$. We conclude that w is a palindrome, which completes the proof. □

5.1.6 Sentential Forms

Derivations from the start symbol produce strings that have a special role. We call these "sentential forms." That is, if $G = (V, T, P, S)$ is a CFG, then any string α in $(V \cup T)^*$ such that $S \overset{*}{\Rightarrow} \alpha$ is a *sentential form*. If $S \underset{lm}{\overset{*}{\Rightarrow}} \alpha$, then α is a *left-sentential form*, and if $S \underset{rm}{\overset{*}{\Rightarrow}} \alpha$, then α is a *right-sentential form*. Note that the language $L(G)$ is those sentential forms that are in T^*; i.e., they consist solely of terminals.

Example 5.8: Consider the grammar for expressions from Fig. 5.2. For example, $E * (I + E)$ is a sentential form, since there is a derivation

$$E \Rightarrow E * E \Rightarrow E * (E) \Rightarrow E * (E + E) \Rightarrow E * (I + E)$$

However this derivation is neither leftmost nor rightmost, since at the last step, the middle E is replaced.

As an example of a left-sentential form, consider $a * E$, with the leftmost derivation

The Form of Proofs About Grammars

Theorem 5.7 is typical of proofs that show a grammar defines a particular, informally defined language. We first develop an inductive hypothesis that states what properties the strings derived from each variable have. In this example, there was only one variable, P, so we had only to claim that its strings were palindromes.

We prove the "if" part: that if a string w satisfies the informal statement about the strings of one of the variables A, then $A \overset{*}{\Rightarrow} w$. In our example, since P is the start symbol, we stated "$P \overset{*}{\Rightarrow} w$" by saying that w is in the language of the grammar. Typically, we prove the "if" part by induction on the length of w. If there are k variables, then the inductive statement to be proved has k parts, which must be proved as a mutual induction.

We must also prove the "only-if" part, that if $A \overset{*}{\Rightarrow} w$, then w satisfies the informal statement about the strings derived from variable A. Again, in our example, since we had to deal only with the start symbol P, we assumed that w was in the language of G_{pal} as an equivalent to $P \overset{*}{\Rightarrow} w$. The proof of this part is typically by induction on the number of steps in the derivation. If the grammar has productions that allow two or more variables to appear in derived strings, then we shall have to break a derivation of n steps into several parts, one derivation from each of the variables. These derivations may have fewer than n steps, so we have to perform an induction assuming the statement for all values n or less, as discussed in Section 1.4.2.

$$E \underset{lm}{\Rightarrow} E * E \underset{lm}{\Rightarrow} I * E \underset{lm}{\Rightarrow} a * E$$

Additionally, the derivation

$$E \underset{rm}{\Rightarrow} E * E \underset{rm}{\Rightarrow} E * (E) \underset{rm}{\Rightarrow} E * (E + E)$$

shows that $E * (E + E)$ is a right-sentential form. □

5.1.7 Exercises for Section 5.1

Exercise 5.1.1: Design context-free grammars for the following languages:

* a) The set $\{0^n1^n \mid n \geq 1\}$, that is, the set of all strings of one or more 0's followed by an equal number of 1's.

*! b) The set $\{a^ib^jc^k \mid i \neq j \text{ or } j \neq k\}$, that is, the set of strings of a's followed by b's followed by c's, such that there are either a different number of a's and b's or a different number of b's and c's, or both.

! c) The set of all strings of a's and b's that are *not* of the form ww, that is, not equal to any string repeated.

!! d) The set of all strings with twice as many 0's as 1's.

Exercise 5.1.2: The following grammar generates the language of regular expression $\mathbf{0}^*\mathbf{1}(\mathbf{0}+\mathbf{1})^*$:

$$\begin{array}{lcl} S & \rightarrow & A1B \\ A & \rightarrow & 0A \mid \epsilon \\ B & \rightarrow & 0B \mid 1B \mid \epsilon \end{array}$$

Give leftmost and rightmost derivations of the following strings:

***** a) 00101.

b) 1001.

c) 00011.

! **Exercise 5.1.3:** Show that every regular language is a context-free language. *Hint*: Construct a CFG by induction on the number of operators in the regular expression.

! **Exercise 5.1.4:** A CFG is said to be *right-linear* if each production body has at most one variable, and that variable is at the right end. That is, all productions of a right-linear grammar are of the form $A \rightarrow wB$ or $A \rightarrow w$, where A and B are variables and w some string of zero or more terminals.

a) Show that every right-linear grammar generates a regular language. *Hint*: Construct an ϵ-NFA that simulates leftmost derivations, using its state to represent the lone variable in the current left-sentential form.

b) Show that every regular language has a right-linear grammar. *Hint*: Start with a DFA and let the variables of the grammar represent states.

***!** **Exercise 5.1.5:** Let $T = \{0, 1, (,), +, *, \emptyset, e\}$. We may think of T as the set of symbols used by regular expressions over alphabet $\{0, 1\}$; the only difference is that we use e for symbol ϵ, to avoid potential confusion in what follows. Your task is to design a CFG with set of terminals T that generates exactly the regular expressions with alphabet $\{0, 1\}$.

Exercise 5.1.6: We defined the relation $\overset{*}{\Rightarrow}$ with a basis "$\alpha \Rightarrow \alpha$" and an induction that says "$\alpha \overset{*}{\Rightarrow} \beta$ and $\beta \Rightarrow \gamma$ imply $\alpha \overset{*}{\Rightarrow} \gamma$. There are several other ways to define $\overset{*}{\Rightarrow}$ that also have the effect of saying that "$\overset{*}{\Rightarrow}$ is zero or more $\Rightarrow$ steps." Prove that the following are true:

a) $\alpha \overset{*}{\Rightarrow} \beta$ if and only if there is a sequence of one or more strings

$$\gamma_1, \gamma_2, \ldots, \gamma_n$$

such that $\alpha = \gamma_1$, $\beta = \gamma_n$, and for $i = 1, 2, \ldots, n-1$ we have $\gamma_i \Rightarrow \gamma_{i+1}$.

b) If $\alpha \overset{*}{\Rightarrow} \beta$, and $\beta \overset{*}{\Rightarrow} \gamma$, then $\alpha \overset{*}{\Rightarrow} \gamma$. *Hint*: use induction on the number of steps in the derivation $\beta \overset{*}{\Rightarrow} \gamma$.

! **Exercise 5.1.7:** Consider the CFG G defined by productions:

$$S \rightarrow aS \mid Sb \mid a \mid b$$

a) Prove by induction on the string length that no string in $L(G)$ has ba as a substring.

b) Describe $L(G)$ informally. Justify your answer using part (a).

!! **Exercise 5.1.8:** Consider the CFG G defined by productions:

$$S \rightarrow aSbS \mid bSaS \mid \epsilon$$

Prove that $L(G)$ is the set of all strings with an equal number of a's and b's.

5.2 Parse Trees

There is a tree representation for derivations that has proved extremely useful. This tree shows us clearly how the symbols of a terminal string are grouped into substrings, each of which belongs to the language of one of the variables of the grammar. But perhaps more importantly, the tree, known as a "parse tree" when used in a compiler, is the data structure of choice to represent the source program. In a compiler, the tree structure of the source program facilitates the translation of the source program into executable code by allowing natural, recursive functions to perform this translation process.

In this section, we introduce the parse tree and show that the existence of parse trees is tied closely to the existence of derivations and recursive inferences. We shall later study the matter of ambiguity in grammars and languages, which is an important application of parse trees. Certain grammars allow a terminal string to have more than one parse tree. That situation makes the grammar unsuitable for a programming language, since the compiler could not tell the structure of certain source programs, and therefore could not with certainty deduce what the proper executable code for the program was.

5.2.1 Constructing Parse Trees

Let us fix on a grammar $G = (V, T, P, S)$. The *parse trees* for G are trees with the following conditions:

1. Each interior node is labeled by a variable in V.

2. Each leaf is labeled by either a variable, a terminal, or ϵ. However, if the leaf is labeled ϵ, then it must be the only child of its parent.

Review of Tree Terminology

We assume you have been introduced to the idea of a tree and are familiar with the commonly used definitions for trees. However, the following will serve as a review.

- Trees are collections of *nodes*, with a *parent-child* relationship. A node has at most one parent, drawn above the node, and zero or more children, drawn below. Lines connect parents to their children. Figures 5.4, 5.5, and 5.6 are examples of trees.
- There is one node, the *root*, that has no parent; this node appears at the top of the tree. Nodes with no children are called *leaves*. Nodes that are not leaves are *interior nodes*.
- A child of a child of a $\cdots$ node is a *descendant* of that node. A parent of a parent of a $\cdots$ is an *ancestor*. Trivially, nodes are ancestors and descendants of themselves.
- The children of a node are ordered "from the left," and drawn so. If node N is to the left of node M, then all the descendants of N are considered to be to the left of all the descendants of M.

3. If an interior node is labeled A, and its children are labeled

$$X_1, X_2, \ldots, X_k$$

respectively, from the left, then $A \rightarrow X_1X_2\cdots X_k$ is a production in P. Note that the only time one of the X's can be ϵ is if that is the label of the only child, and $A \rightarrow \epsilon$ is a production of G.

Example 5.9 : Figure 5.4 shows a parse tree that uses the expression grammar of Fig. 5.2. The root is labeled with the variable E. We see that the production used at the root is $E \rightarrow E + E$, since the three children of the root have labels E, $+$, and E, respectively, from the left. At the leftmost child of the root, the production $E \rightarrow I$ is used, since there is one child of that node, labeled I. $\Box$

Example 5.10 : Figure 5.5 shows a parse tree for the palindrome grammar of Fig. 5.1. The production used at the root is $P \rightarrow 0P0$, and at the middle child of the root it is $P \rightarrow 1P1$. Note that at the bottom is a use of the production $P \rightarrow \epsilon$. That use, where the node labeled by the head has one child, labeled ϵ, is the only time that a node labeled ϵ can appear in a parse tree. $\Box$

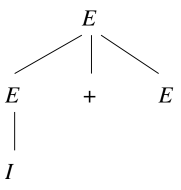

Figure 5.4: A parse tree showing the derivation of $I + E$ from E

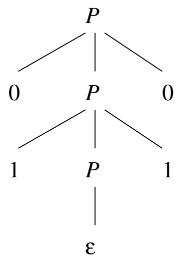

Figure 5.5: A parse tree showing the derivation $P \overset{*}{\Rightarrow} 0110$

5.2.2 The Yield of a Parse Tree

If we look at the leaves of any parse tree and concatenate them from the left, we get a string, called the *yield* of the tree, which is always a string that is derived from the root variable. The fact that the yield is derived from the root will be proved shortly. Of special importance are those parse trees such that:

1. The yield is a terminal string. That is, all leaves are labeled either with a terminal or with ϵ.

2. The root is labeled by the start symbol.

These are the parse trees whose yields are strings in the language of the underlying grammar. We shall also prove shortly that another way to describe the language of a grammar is as the set of yields of those parse trees having the start symbol at the root and a terminal string as yield.

Example 5.11: Figure 5.6 is an example of a tree with a terminal string as yield and the start symbol at the root; it is based on the grammar for expressions that we introduced in Fig. 5.2. This tree's yield is the string $a * (a + b00)$ that was derived in Example 5.5. In fact, as we shall see, this particular parse tree is a representation of that derivation. $\square$

5.2.3 Inference, Derivations, and Parse Trees

Each of the ideas that we have introduced so far for describing how a grammar works gives us essentially the same facts about strings. That is, given a grammar $G = (V, T, P, S)$, we shall show that the following are equivalent:

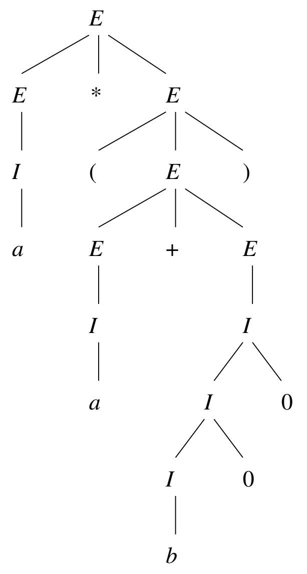

Figure 5.6: Parse tree showing $a * (a + b00)$ is in the language of our expression grammar

1. The recursive inference procedure determines that terminal string w is in the language of variable A.
2. $A \overset{*}{\Rightarrow} w$.
3. $A \underset{lm}{\overset{*}{\Rightarrow}} w$.
4. $A \underset{rm}{\overset{*}{\Rightarrow}} w$.
5. There is a parse tree with root A and yield w.

In fact, except for the use of recursive inference, which we only defined for terminal strings, all the other conditions — the existence of derivations, leftmost or rightmost derivations, and parse trees — are also equivalent if w is a string that has some variables.

We need to prove these equivalences, and we do so using the plan of Fig. 5.7. That is, each arc in that diagram indicates that we prove a theorem that says if w meets the condition at the tail, then it meets the condition at the head of the arc. For instance, we shall show in Theorem 5.12 that if w is inferred to be in the language of A by recursive inference, then there is a parse tree with root A and yield w.

Note that two of the arcs are very simple and will not be proved formally. If w has a leftmost derivation from A, then it surely has a derivation from A, since a leftmost derivation *is* a derivation. Likewise, if w has a rightmost derivation,

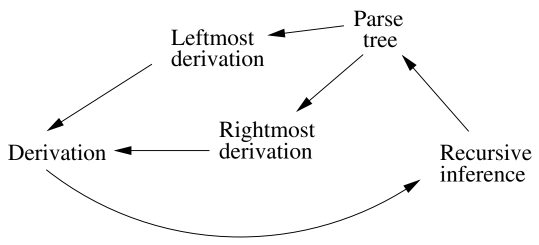

Figure 5.7: Proving the equivalence of certain statements about grammars

then it surely has a derivation. We now proceed to prove the harder steps of this equivalence.

5.2.4 From Inferences to Trees

Theorem 5.12: Let $G = (V, T, P, S)$ be a CFG. If the recursive inference procedure tells us that terminal string w is in the language of variable A, then there is a parse tree with root A and yield w.

PROOF: The proof is an induction on the number of steps used to infer that w is in the language of A.

BASIS: One step. Then only the basis of the inference procedure must have been used. Thus, there must be a production $A \to w$. The tree of Fig. 5.8, where there is one leaf for each position of w, meets the conditions to be a parse tree for grammar G, and it evidently has yield w and root A. In the special case that $w = \epsilon$, the tree has a single leaf labeled ϵ and is a legal parse tree with root A and yield w.

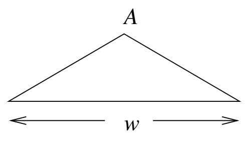

Figure 5.8: Tree constructed in the basis case of Theorem 5.12

INDUCTION: Suppose that the fact w is in the language of A is inferred after $n+1$ inference steps, and that the statement of the theorem holds for all strings x and variables B such that the membership of x in the language of B was inferred using n or fewer inference steps. Consider the last step of the inference that w is in the language of A. This inference uses some production for A, say $A \to X_1X_2 \cdots X_k$, where each X_i is either a variable or a terminal.

We can break w up as $w_1w_2 \cdots w_k$, where:

1. If X_i is a terminal, then $w_i = X_i$; i.e., w_i consists of only this one terminal from the production.

2. If X_i is a variable, then w_i is a string that was previously inferred to be in the language of X_i. That is, this inference about w_i took at most n of the $n+1$ steps of the inference that w is in the language of A. It cannot take all $n+1$ steps, because the final step, using production $A \to X_1X_2\cdots X_k$, is surely not part of the inference about w_i. Consequently, we may apply the inductive hypothesis to w_i and X_i, and conclude that there is a parse tree with yield w_i and root X_i.

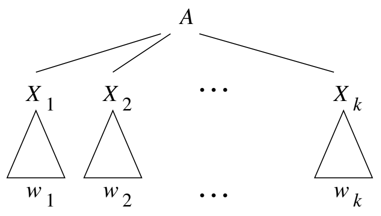

Figure 5.9: Tree used in the inductive part of the proof of Theorem 5.12

We then construct a tree with root A and yield w, as suggested in Fig. 5.9. There is a root labeled A, whose children are $X_1, X_2, \ldots, X_k$. This choice is valid, since $A \to X_1X_2\cdots X_k$ is a production of G.

The node for each X_i is made the root of a subtree with yield w_i. In case (1), where X_i is a terminal, this subtree is a trivial tree with a single node labeled X_i. That is, the subtree consists of only this child of the root. Since $w_i = X_i$ in case (1), we meet the condition that the yield of the subtree is w_i.

In case (2), X_i is a variable. Then, we invoke the inductive hypothesis to claim that there is some tree with root X_i and yield w_i. This tree is attached to the node for X_i in Fig. 5.9.

The tree so constructed has root A. Its yield is the yields of the subtrees, concatenated from left to right. That string is $w_1w_2\cdots w_k$, which is w. $\square$

5.2.5 From Trees to Derivations

We shall now show how to construct a leftmost derivation from a parse tree. The method for constructing a rightmost derivation uses the same ideas, and we shall not explore the rightmost-derivation case. In order to understand how derivations may be constructed, we need first to see how one derivation of a string from a variable can be embedded within another derivation. An example should illustrate the point.

Example 5.13: Let us again consider the expression grammar of Fig. 5.2. It is easy to check that there is a derivation

$$E \Rightarrow I \Rightarrow Ib \Rightarrow ab$$

As a result, for any strings α and β, it is also true that

$$\alpha E\beta \Rightarrow \alpha I\beta \Rightarrow \alpha Ib\beta \Rightarrow \alpha ab\beta$$

The justification is that we can make the same replacements of production bodies for heads in the context of α and β as we can in isolation.[1]

For instance, if we have a derivation that begins $E \Rightarrow E + E \Rightarrow E + (E)$, we could apply the derivation of ab from the second E by treating "$E + ($" as α and "$)$" as β. This derivation would then continue

$$E + (E) \Rightarrow E + (I) \Rightarrow E + (Ib) \Rightarrow E + (ab)$$

□

We are now able to prove a theorem that lets us convert a parse tree to a leftmost derivation. The proof is an induction on the *height* of the tree, which is the maximum length of a path that starts at the root, and proceeds downward through descendants, to a leaf. For instance, the height of the tree in Fig. 5.6 is 7. The longest root-to-leaf path in this tree goes to the leaf labeled b. Note that path lengths conventionally count the edges, not the nodes, so a path consisting of a single node is of length 0.

Theorem 5.14: Let $G = (V, T, P, S)$ be a CFG, and suppose there is a parse tree with root labeled by variable A and with yield w, where w is in T^*. Then there is a leftmost derivation $A \underset{lm}{\overset{*}{\Rightarrow}} w$ in grammar G.

PROOF: We perform an induction on the height of the tree.

BASIS: The basis is height 1, the least that a parse tree with a yield of terminals can be. In this case, the tree must look like Fig. 5.8, with a root labeled A and children that read w, left-to-right. Since this tree is a parse tree, $A \to w$ must be a production. Thus, $A \underset{lm}{\Rightarrow} w$ is a one-step, leftmost derivation of w from A.

INDUCTION: If the height of the tree is n, where $n > 1$, it must look like Fig 5.9. That is, there is a root labeled A, with children labeled $X_1, X_2, \ldots, X_k$ from the left. The X's may be either terminals or variables.

1. If X_i is a terminal, define w_i to be the string consisting of X_i alone.

2. If X_i is a variable, then it must be the root of some subtree with a yield of terminals, which we shall call w_i. Note that in this case, the subtree is of height less than n, so the inductive hypothesis applies to it. That is, there is a leftmost derivation $X_i \underset{lm}{\overset{*}{\Rightarrow}} w_i$.

Note that $w = w_1 w_2 \cdots w_k$.

[1]In fact, it is this property of being able to make a string-for-variable substitution regardless of context that gave rise originally to the term "context-free." There is a more powerful classes of grammars, called "context-sensitive," where replacements are permitted only if certain strings appear to the left and/or right. Context-sensitive grammars do not play a major role in practice today.

We construct a leftmost derivation of w as follows. We begin with the step $A \underset{lm}{\Rightarrow} X_1X_2\cdots X_k$. Then, for each $i = 1, 2, \ldots, k$, in order, we show that

$$A \overset{*}{\underset{lm}{\Rightarrow}} w_1w_2\cdots w_iX_{i+1}X_{i+2}\cdots X_k$$

This proof is actually another induction, this time on i. For the basis, $i = 0$, we already know that $A \underset{lm}{\Rightarrow} X_1X_2\cdots X_k$. For the induction, assume that

$$A \overset{*}{\underset{lm}{\Rightarrow}} w_1w_2\cdots w_{i-1}X_iX_{i+1}\cdots X_k$$

a) If X_i is a terminal, do nothing. However, we shall subsequently think of X_i as the terminal string w_i. Thus, we already have

$$A \overset{*}{\underset{lm}{\Rightarrow}} w_1w_2\cdots w_iX_{i+1}X_{i+2}\cdots X_k$$

b) If X_i is a variable, continue with a derivation of w_i from X_i, in the context of the derivation being constructed. That is, if this derivation is

$$X_i \underset{lm}{\Rightarrow} \alpha_1 \underset{lm}{\Rightarrow} \alpha_2 \cdots \underset{lm}{\Rightarrow} w_i$$

we proceed with

$$\begin{array}{l} w_1w_2\cdots w_{i-1}X_iX_{i+1}\cdots X_k \underset{lm}{\Rightarrow} \\ w_1w_2\cdots w_{i-1}\alpha_1X_{i+1}\cdots X_k \underset{lm}{\Rightarrow} \\ w_1w_2\cdots w_{i-1}\alpha_2X_{i+1}\cdots X_k \underset{lm}{\Rightarrow} \\ \cdots \\ w_1w_2\cdots w_iX_{i+1}X_{i+2}\cdots X_k \end{array}$$

The result is a derivation $A \overset{*}{\underset{lm}{\Rightarrow}} w_1w_2\cdots w_iX_{i+1}\cdots X_k$.

When $i = k$, the result is a leftmost derivation of w from A. □

Example 5.15 : Let us construct the leftmost derivation for the tree of Fig. 5.6. We shall show only the final step, where we construct the derivation from the entire tree from derivations that correspond to the subtrees of the root. That is, we shall assume that by recursive application of the technique in Theorem 5.14, we have deduced that the subtree rooted at the first child of the root has leftmost derivation $E \underset{lm}{\Rightarrow} I \underset{lm}{\Rightarrow} a$, while the subtree rooted at the third child of the root has leftmost derivation

$$E \underset{lm}{\Rightarrow} (E) \underset{lm}{\Rightarrow} (E+E) \underset{lm}{\Rightarrow} (I+E) \underset{lm}{\Rightarrow} (a+E) \underset{lm}{\Rightarrow}$$

$$(a+I) \underset{lm}{\Rightarrow} (a+I0) \underset{lm}{\Rightarrow} (a+I00) \underset{lm}{\Rightarrow} (a+b00)$$

To build a leftmost derivation for the entire tree, we start with the step at the root: $E \underset{lm}{\Rightarrow} E * E$. Then, we replace the first E according to its derivation, following each step by $*E$ to account for the larger context in which that derivation is used. The leftmost derivation so far is thus

$$E \underset{lm}{\Rightarrow} E * E \underset{lm}{\Rightarrow} I * E \underset{lm}{\Rightarrow} a * E$$

The * in the production used at the root requires no derivation, so the above leftmost derivation also accounts for the first two children of the root. We complete the leftmost derivation by using the derivation of $E \overset{*}{\underset{lm}{\Rightarrow}} (a + b00)$, in a context where it is preceded by $a*$ and followed by the empty string. This derivation actually appeared in Example 5.6; it is:

$$E \underset{lm}{\Rightarrow} E * E \underset{lm}{\Rightarrow} I * E \underset{lm}{\Rightarrow} a * E \underset{lm}{\Rightarrow}$$

$$a * (E) \underset{lm}{\Rightarrow} a * (E + E) \underset{lm}{\Rightarrow} a * (I + E) \underset{lm}{\Rightarrow} a * (a + E) \underset{lm}{\Rightarrow}$$

$$a * (a + I) \underset{lm}{\Rightarrow} a * (a + I0) \underset{lm}{\Rightarrow} a * (a + I00) \underset{lm}{\Rightarrow} a * (a + b00)$$

□

A similar theorem lets us convert a tree to a rightmost derivation. The construction of a rightmost derivation from a tree is almost the same as the construction of a leftmost derivation. However, after starting with the step $A \underset{rm}{\Rightarrow} X_1 X_2 \cdots X_k$, we expand X_k first, using a rightmost derivation, then expand X_{k-1}, and so on, down to X_1. Thus, we shall state without further proof:

Theorem 5.16: Let $G = (V, T, P, S)$ be a CFG, and suppose there is a parse tree with root labeled by variable A and with yield w, where w is in T^*. Then there is a rightmost derivation $A \overset{*}{\underset{rm}{\Rightarrow}} w$ in grammar G. □

5.2.6 From Derivations to Recursive Inferences

We now complete the loop suggested by Fig. 5.7 by showing that whenever there is a derivation $A \overset{*}{\Rightarrow} w$ for some CFG, then the fact that w is in the language of A is discovered in the recursive inference procedure. Before giving the theorem and proof, let us observe something important about derivations.

Suppose that we have a derivation $A \Rightarrow X_1 X_2 \cdots X_k \overset{*}{\Rightarrow} w$. Then we can break w into pieces $w = w_1 w_2 \cdots w_k$ such that $X_i \overset{*}{\Rightarrow} w_i$. Note that if X_i is a terminal, then $w_i = X_i$, and the derivation is zero steps. The proof of this observation is not hard. You can show by induction on the number of steps of the derivation, that if $X_1 X_2 \cdots X_k \overset{*}{\Rightarrow} \alpha$, then all the positions of α that come from expansion of X_i are to the left of all the positions that come from expansion of X_j, if $i < j$.

If X_i is a variable, we can obtain the derivation of $X_i \overset{*}{\Rightarrow} w_i$ by starting with the derivation $A \overset{*}{\Rightarrow} w$, and stripping away:

a) All the positions of the sentential forms that are either to the left or right of the positions that are derived from X_i, and

b) All the steps that are not relevant to the derivation of w_i from X_i.

An example should make this process clear.

Example 5.17: Using the expression grammar of Fig. 5.2, consider the derivation

$$E \Rightarrow E * E \Rightarrow E * E + E \Rightarrow I * E + E \Rightarrow I * I + E \Rightarrow$$

$$I * I + I \Rightarrow a * I + I \Rightarrow a * b + I \Rightarrow a * b + a$$

Consider the third sentential form, $E * E + E$, and the middle E in this form.[2]

Starting from $E * E + E$, we may follow the steps of the above derivation, but strip away whatever positions are derived from the $E*$ to the left of the central E or derived from the $+E$ to its right. The steps of the derivation then become E, E, I, I, I, b, b. That is, the next step does not change the central E, the step after that changes it to I, the next two steps leave it as I, the next changes it to b, and the final step does not change what is derived from the central E.

If we take only the steps that change what comes from the central E, the sequence of strings E, E, I, I, I, b, b becomes the derivation $E \Rightarrow I \Rightarrow b$. That derivation correctly describes how the central E evolves during the complete derivation. □

Theorem 5.18: Let $G = (V, T, P, S)$ be a CFG, and suppose there is a derivation $A \underset{G}{\overset{*}{\Rightarrow}} w$, where w is in T^*. Then the recursive inference procedure applied to G determines that w is in the language of variable A.

PROOF: The proof is an induction on the length of the derivation $A \overset{*}{\Rightarrow} w$.

BASIS: If the derivation is one-step, then $A \rightarrow w$ must be a production. Since w consists of terminals only, the fact that w is in the language of A will be discovered in the basis part of the recursive inference procedure.

INDUCTION: Suppose the derivation takes $n + 1$ steps, and assume that for any derivation of n or fewer steps, the statement holds. Write the derivation as $A \Rightarrow X_1 X_2 \cdots X_k \overset{*}{\Rightarrow} w$. Then, as discussed prior to the theorem, we can break w as $w = w_1 w_2 \cdots w_k$, where:

[2]Our discussion of finding subderivations from larger derivations assumed we were concerned with a variable in the second sentential form of some derivation. However, the idea applies to a variable in any step of a derivation.

a) If X_i is a terminal, then $w_i = X_i$.

b) If X_i is a variable, then $X_i \overset{*}{\Rightarrow} w_i$. Since the first step of the derivation $A \overset{*}{\Rightarrow} w$ is surely not part of the derivation $X_i \overset{*}{\Rightarrow} w_i$, we know that this derivation is of n or fewer steps. Thus, the inductive hypothesis applies to it, and we know that w_i is inferred to be in the language of X_i.

Now, we have a production $A \to X_1X_2 \cdots X_k$, with w_i either equal to X_i or known to be in the language of X_i. In the next round of the recursive inference procedure, we shall discover that $w_1w_2 \cdots w_k$ is in the language of A. Since $w_1w_2 \cdots w_k = w$, we have shown that w is inferred to be in the language of A. $\square$

5.2.7 Exercises for Section 5.2

Exercise 5.2.1: For the grammar and each of the strings in Exercise 5.1.2, give parse trees.

! **Exercise 5.2.2:** Suppose that G is a CFG without any productions that have ϵ as the right side. If w is in $L(G)$, the length of w is n, and w has a derivation of m steps, show that w has a parse tree with $n + m$ nodes.

! **Exercise 5.2.3:** Suppose all is as in Exercise 5.2.2, but G may have some productions with ϵ as the right side. Show that a parse tree for a string w other than ϵ may have as many as $n + 2m - 1$ nodes, but no more.

! **Exercise 5.2.4:** In Section 5.2.6 we mentioned that if $X_1X_2 \cdots X_k \overset{*}{\Rightarrow} \alpha$, then all the positions of α that come from expansion of X_i are to the left of all the positions that come from expansion of X_j, if $i < j$. Prove this fact. *Hint*: Perform an induction on the number of steps in the derivation.

5.3 Applications of Context-Free Grammars

Context-free grammars were originally conceived by N. Chomsky as a way to describe natural languages. That promise has not been fulfilled. However, as uses for recursively defined concepts in Computer Science have multiplied, so has the need for CFG's as a way to describe instances of these concepts. We shall sketch two of these uses, one old and one new.

1. Grammars are used to describe programming languages. More importantly, there is a mechanical way of turning the language description as a CFG into a parser, the component of the compiler that discovers the structure of the source program and represents that structure by a parse tree. This application is one of the earliest uses of CFG's; in fact it is one of the first ways in which theoretical ideas in Computer Science found their way into practice.