CHAPTER 2

GRAMMARS

2.1 MOTIVATION

There is one class of generating systems of primary interest to us—systems known as grammars. The concept of a grammar was originally formalized by linguists in their study of natural languages. Linguists were concerned not only with defining precisely what is or is not a valid sentence of a language, but also with providing structural descriptions of the sentences. One of their goals was to develop a formal grammar capable of describing English.

It was hoped that if, for example, one had a formal grammar to describe the English language, one could use the computer in ways that require it to "understand" English. Such a use might be language translation or the computer solution of word problems.

To date, this goal is for the most part unrealized. We still do not have a definitive grammar for English, and there is even disagreement as to what types of formal grammar are capable of describing English. However, in describing computer languages, better results have been achieved. For

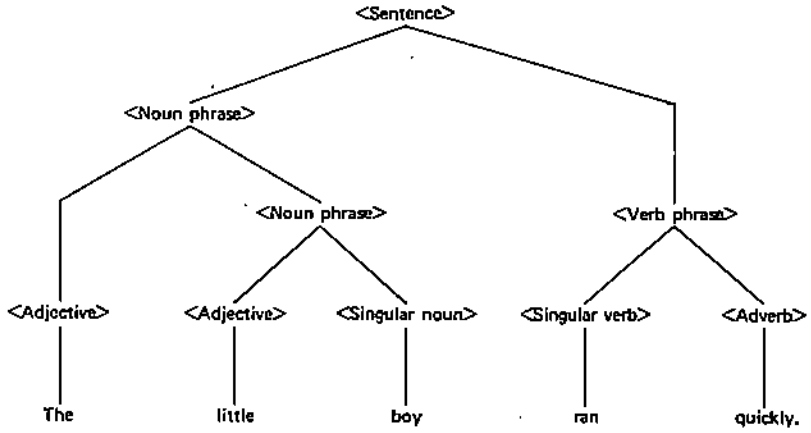

Fig. 2.1. A diagram of the sentence "The little boy ran quickly."

example, the Backus Normal Form used to describe ALGOL is a "context-free grammar," a type of grammar with which we shall deal.

We are all familiar with the idea of diagramming or parsing an English sentence. For example, the sentence "The little boy ran quickly" is parsed by noting that the sentence consists of the noun phrase "The little boy" followed by the verb phrase "ran quickly." The noun phrase is then broken down into the singular noun "boy" modified by the two adjectives "The" and "little." The verb phrase is broken down into the singular verb "ran" modified by the adverb "quickly." This sentence structure is indicated in the diagram of Fig. 2.1. We recognize the sentence structure as being grammatically correct. If we had a complete set of rules for parsing all English sentences, then we would have a technique for determining whether or not a sentence is grammatically correct. However, such a set does not exist. Part of the reason for this stems from the fact that there are no clear rules for determining precisely what constitutes a sentence.

The rules we applied to parsing the above sentence can be written in the following form:

⟨sentence⟩ → ⟨noun phrase⟩ ⟨verb phrase⟩
⟨noun phrase⟩ → ⟨adjective⟩ ⟨noun phrase⟩
⟨noun phrase⟩ → ⟨adjective⟩ ⟨singular noun⟩
⟨verb phrase⟩ → ⟨singular verb⟩ ⟨adverb⟩
⟨adjective⟩ → The
⟨adjective⟩ → little
⟨singular noun⟩ → boy
⟨singular verb⟩ → ran
⟨adverb⟩ → quickly

The arrow in the above rules indicates that the item to the left of the arrow can generate the items to the right of the arrow. Note that we have enclosed the names of the parts of the sentence such as noun, verb, verb phrase, etc., in brackets to avoid confusion with the English words and phrases "noun," "verb," "verb phrase," etc.

One should note that we cannot only test sentences for their grammatical correctness, but can also generate grammatically correct sentences by starting with the quantity ⟨sentence⟩ and replacing ⟨sentence⟩ by ⟨noun phrase⟩ followed by ⟨verb phrase⟩. Next we select one of the two rules for ⟨noun phrase⟩ and apply it, and so on, until no further application of the rules is possible. In this way any one of an infinite number of sentences can be derived—that is, any sentence consisting of a string of occurrences of "the" and "little" followed by "boy ran quickly" such as "little the the boy ran quickly" can be generated. Most of the sentences do not make sense but, nevertheless, are grammatically correct in a broad sense.

2.2 THE FORMAL NOTION OF A GRAMMAR

Let us formalize the partial grammar for English which was mentioned in Section 2.1. Four concepts were present. First, there were certain syntactic categories—⟨singular noun⟩, ⟨verb phrase⟩, ⟨sentence⟩, etc., from which strings of words could be derived. The objects corresponding to syntactic categories we call "nonterminals" or "variables." Second, there were the words themselves. The objects which play the role of words we shall call "terminals."

The third concept is the relation that exists between various strings of variables and terminals. These relationships we call "productions." Examples of productions are ⟨noun phrase⟩ → ⟨adjective⟩ ⟨noun phrase⟩ or ⟨singular noun⟩ ⟨singular predicate⟩ → ⟨singular noun⟩ ⟨adverb⟩ ⟨singular verb⟩. Finally, one nonterminal is distinguished, in that it generates exactly those strings of terminals that are deemed in the language. In our example, ⟨sentence⟩ is distinguished. We call the distinguished nonterminal the "sentence" or "start" symbol.

Formally, we denote a *grammar* G by (V_N, V_T, P, S). The symbols V_N, V_T, P, and S are, respectively, the *variables*, *terminals*, *productions*, and *start symbol*. V_N, V_T, and P are finite sets. We assume that V_N and V_T contain no elements in common; that is,

$$V_N \cap V_T = \varphi\dagger.$$

We conventionally denote $V_N \cup V_T$ by V.

The set of productions P consists of expressions of the form $\alpha \to \beta$, where α is a string in V^+ and β is a string in V^*. Finally, S is always a symbol in V_N.

Customarily, we shall use capital Latin-alphabet letters for variables. Lower case letters at the beginning of the Latin alphabet are used for terminals. Strings of terminals are denoted by lower case letters near the end of the Latin alphabet, and strings of variables and terminals are denoted by lower case Greek letters.

We have presented a grammar, $G = (V_N, V_T, P, S)$, but have not yet defined the language it generates. To do so, we need the relations $\underset{G}{\to}$ and $\overset{*}{\underset{G}{\to}}$ between strings in V^*. Specifically, if $\alpha \to \beta$ is a production of P and γ and δ are any strings in V^*, then $\gamma\alpha\delta \underset{G}{\to} \gamma\beta\delta\ddagger$. We say that the production $\alpha \to \beta$ is applied to the string $\gamma\alpha\delta$ to obtain $\gamma\beta\delta$. Thus $\underset{G}{\to}$ relates two strings exactly when the second is obtained from the first by the application of a single production.

† φ denotes the empty set.

‡ Say $\gamma\alpha\delta$ *directly derives* $\gamma\beta\delta$ in grammar G.

Suppose that $\alpha_1, \alpha_2, \ldots, \alpha_m$ are strings in V^*, and $\alpha_1 \underset{G}{\to} \alpha_2, \alpha_2 \underset{G}{\to} \alpha_3, \ldots, \alpha_{m-1} \underset{G}{\to} \alpha_m$. Then we say $\alpha_1 \overset{*}{\underset{G}{\to}} \alpha_m$.† In simple terms, we say for two strings α and β that $\alpha \overset{*}{\underset{G}{\to}} \beta$ if we can obtain β from α by application of some number of productions of P. By convention, $\alpha \overset{*}{\underset{G}{\to}} \alpha$ for each string α.

We define the *language generated by G* [denoted $L(G)$] to be $\{w | w \text{ is in } V_T^* \text{ and } S \overset{*}{\underset{G}{\to}} w\}$.‡ That is, a string is in $L(G)$ if:

1. The string consists solely of terminals.
2. The string can be derived from S.

A string of terminals and nonterminals α is called a *sentential form* if $S \overset{*}{\to} \alpha$. (Usually, if it is clear which grammar G is involved, we use $\to$ for $\underset{G}{\to}$ and $\overset{*}{\to}$ for $\overset{*}{\underset{G}{\to}}$.)

We define grammars G_1 and G_2 to be *equivalent* if $L(G_1) = L(G_2)$.

Example 2.1. Let us consider a grammar $G = (V_N, V_T, P, S)$, where $V_N = \{S\}$, $V_T = \{0, 1\}$, $P = \{S \to 0S1, S \to 01\}$. Here, S is the only variable, 0 and 1 are terminals. There are two productions, $S \to 0S1$ and $S \to 01$. By applying the first production $n - 1$ times, followed by an application of the second production, we have

$$S \to 0S1 \to 00S11 \to 0^3S1^3 \to \cdots \to 0^{n-1}S1^{n-1} \to 0^n1^n.\S$$

Furthermore, these are the only strings in $L(G)$. After using the second production, we find that the number of S's in the sentential form decreases by one. Each time the first production is used, the number of S's remains the same. Thus, after using $S \to 01$, no S's remain in the resulting string. Since both productions have an S on the left, the only order in which the productions can be applied is $S \to 0S1$ some number of times followed by one application of $S \to 01$. Thus, $L(G) = \{0^n1^n | n \geqq 1\}$.

Example 2.1 was a simple example of a grammar. It was relatively easy to determine which words were derivable and which were not. In general, it may be exceedingly hard to determine what is generated by the grammar. Here is another, more difficult, example.

† Say α_1 *derives* α_m in grammar G.

‡ We shall often use the notation $L = \{x | \varphi(x)\}$, where $\varphi(x)$ is some statement about x, to define languages. It stands for "the set of all x such that $\varphi(x)$ is true." Sometimes, x itself will have some special form. For example, $\{ww | w \text{ is in } V^*\}$ is the set of words of V^* whose first half and second half are the same.

§ If w is any string, w^i will stand for w repeated i times. So $0^3 = 000$. Note: $w^0 = \epsilon$.

Example 2.2. Let $G = (V_N, V_T, P, S)$, $V_N = \{S, B, C\}$, $V_T = \{a, b, c\}$. P consists of the following productions:

1. $S \to aSBC$
2. $S \to aBC$
3. $CB \to BC$
4. $aB \to ab$
5. $bB \to bb$
6. $bC \to bc$
7. $cC \to cc$

The language $L(G)$ contains the word $a^nb^nc^n$ for each $n \geq 1$, since we can use production (1) $n - 1$ times to get $S \xrightarrow{*} a^{n-1}S(BC)^{n-1}$. Then, we use production (2) to get $S \xrightarrow{*} a^n(BC)^n$. Production (3) enables us to arrange the B's and C's so that all B's precede all C's. For example, if $n = 3$,

$$aaaBCBCBC \to aaaBBCCBC \to aaaBBCBCC \to aaaBBBCCC.$$

Thus, $S \xrightarrow{*} a^nB^nC^n$.

Next we use production (4) once to get $S \xrightarrow{*} a^nbB^{n-1}C^n$. Then use production (5) $n - 1$ times to get $S \xrightarrow{*} a^nb^nC^n$. Finally, use production (6) once and production (7) $n - 1$ times to get $S \xrightarrow{*} a^nb^nc^n$.

Now, let us show that the words $a^nb^nc^n$ for $n \geq 1$ are the only terminal strings in $L(G)$. In any derivation beginning with S, until we use production (2), we cannot use (4), (5), (6), or (7), for each of productions (4) through (7) requires a terminal immediately to the left of a B or C. Until production (2) is used, all strings derived consist of a's followed by an S, followed by B's and C's.

After (2) is used, the string consists of n a's, for some $n \geq 1$, followed by n B's and n C's in some order. Now no S's appear in the string, so productions (1) and (2) may no longer be used. Note that the form of the string is all terminals followed by all variables. After applying any of productions (3) through (7), we see that the string will still have that property. Note that (4) through (7) are only applicable at the boundary between terminals and variables. Each has the effect of converting one B to b or one C to c. Production (3) causes B's to migrate to the left, and C's to the right.

Suppose that a C is converted to c before all B's are converted to b's. Then the string can be written as $a^nb^ic\alpha$, where $i < n$ and α is a string of B's and C's, but not all C's. Now, only productions (3) and (7) may be applied; (7) at the interface between terminals and variables, and (3) among the variables. We may use (3) to reorder the B's and C's of α, but not to remove any B's. Production (7) can convert C's to c's at the interface, but eventually, a B will be the leftmost variable. There is no production that can change the B, so this string can never result in a string with no variables.

We conclude that all B's must be converted to b's at the interface between terminals and variables before any C's are converted to c's. Thus, from a^n followed by n B's and n C's in any order, $a^nb^nc^n$ is the only derivable terminal string. Therefore, $L(G) = \{a^nb^nc^n | n \geq 1\}$.

2.3 THE TYPES OF GRAMMARS

We call the type of grammar we have defined a *type 0 grammar*. Certain restrictions can be made on the nature of the productions of a grammar to give three other types of grammars, sometimes called types 1, 2, and 3.

Let $G = (V_N, V_T, P, S)$ be a grammar. Suppose that for every production $\alpha \to \beta$ in P, $|\beta| \geq |\alpha|$.† Then the grammar G is *type 1* or *context sensitive*. We shall use the latter name more often than the former.

As an example, consider the grammar discussed in Example 2.2. Each of the seven productions of the grammar has at least as many symbols on the right as on the left. So, this grammar is context sensitive. Likewise the grammar in Example 2.1 is also context sensitive.

Some authors require that the productions of a context-sensitive grammar be of the form $\alpha_1A\alpha_2 \to \alpha_1\beta\alpha_2$, with α_1, α_2 and β in V^*, $\beta \neq \epsilon$ and A in V_N. It can be shown that this restriction does not change the class of languages generated. However, it does motivate the name context sensitive since the production $\alpha_1A\alpha_2 \to \alpha_1\beta\alpha_2$ allows A to be replaced by β whenever A appears in the context of α_1 and α_2.

Let $G = (V_N, V_T, P, S)$. Suppose that for every production $\alpha \to \beta$ in P,

1. α is a single variable.
2. β is any string other than ϵ.

Then the grammar is called *type 2* or *context free*. Note that a production of the form $A \to \beta$ allows the variable A to be replaced by the string β independent of the context in which the A appears. Hence the name context free.

Example 2.3. Let us consider an interesting context-free grammar. It is $G = (V_N, V_T, P, S)$, where $V_N = \{S, A, B\}$, $V_T = \{a, b\}$ and P consists of the following.

$S \to aB$ $\quad$ $A \to bAA$
$S \to bA$ $\quad$ $B \to b$
$A \to a$ $\quad$ $B \to bS$
$A \to aS$ $\quad$ $B \to aBB$

The grammar G is context free since for each production, the left-hand side is a single variable and the right-hand side is a nonempty string of terminals and variables.

The language $L(G)$ is the set of all words in V_T^+ consisting of an equal number of a's and b's. We shall prove this statement by induction on the length of a word.

Inductive Hypothesis. For w in V_T^+,

1. $S \xrightarrow{*} w$ if and only if w consists of an equal number of a's and b's.
2. $A \xrightarrow{*} w$ if and only if w has one more a than it has b's.
3. $B \xrightarrow{*} w$ if and only if w has one more b than it has a's.

† We use $|x|$ to stand for the length, or number of symbols in the string x.

The inductive hypothesis is certainly true if $|w| = 1$, since $A \xrightarrow{*} a$, $B \xrightarrow{*} b$, and no terminal string of length one is derivable from S. Also, no strings of length one, other than a and b are derivable from A and B, respectively.

Suppose that the inductive hypothesis is true for all w of length $k - 1$ or less. We shall show that it is true for $|w| = k$. First, if $S \xrightarrow{*} w$, then the derivation must begin with either $S \to aB$ or $S \to bA$. In the first case, w is of the form aw_1, where $|w_1| = k - 1$ and $B \xrightarrow{*} w_1$. By the inductive hypothesis, the number of b's in w_1 is one more than the number of a's, so w consists of an equal number of a's and b's. A similar argument prevails if the derivation begins with $S \to bA$.

We must now prove the "only if" of part (1), that is, if $|w| = k$ and w consists of an equal number of a's and b's, then $S \xrightarrow{*} w$. Either the first symbol of w is a or it is b. Assume that $w = aw_1$. Now $|w_1| = k - 1$, and w_1 has one more b than a. By the inductive hypothesis, $B \xrightarrow{*} w_1$. But then $S \to aB \xrightarrow{*} aw_1 = w$. A similar argument prevails if the first symbol of w is b.

Our task is not done. To complete the proof, we must show parts (2) and (3) of the inductive hypothesis for w of length k. These parts are proved in a manner similar to our method of proof for part (1). They will be left to the reader.

Let $G = (V_N, V_T, P, S)$ be a grammar. Suppose that every production in P is of the form $A \to aB$ or $A \to a$, where A and B are variables and a is a terminal. Then G is called a *type 3* or *regular* grammar. In Chapter 3, we shall introduce the finite state machine and see that the languages generated by type 3 grammars are precisely the sets accepted by finite-state machines.

Example 2.4. Consider the grammar $G = (\{S, A, B\}, \{0, 1\}, P, S)$, where P consists of the following:

$$\begin{array}{ll} S \to 0A & B \to 1B \\ S \to 1B & B \to 1 \\ A \to 0A & B \to 0 \\ A \to 0S & S \to 0 \\ A \to 1B & \end{array}$$

Clearly G is a regular grammar. We shall not describe $L(G)$, but rather leave it to the reader to determine what is generated and prove his conclusion.

It should be clear that every regular grammar is context free; every context-free grammar is context sensitive; every context-sensitive grammar is type 0. We shall call a language that can be generated by a type 0 grammar

a *type 0 language*. A language generated by a context-sensitive, context-free, or regular grammar is a *context-sensitive*, *context-free*, or *regular language*, respectively.

We shall abbreviate context-sensitive, context-free, and regular grammar by csg, cfg, and rg,† respectively. Context-sensitive and context-free languages are abbreviated csl and cfl, respectively. In line with current practice, a type 3 or regular language will often be called a *regular set*. A type 0 language is abbreviated r.e. set, for *recursively enumerable set*. It shall be seen later that the languages generated by type 0 grammars correspond, intuitively to the languages which can be enumerated by finitely described procedures.

2.4 THE EMPTY SENTENCE

We might note that, as defined here, ϵ can be in no csl, cfl, or regular set. Recalling that our motivation for thinking of grammars was to find finite descriptions for languages, we would have to agree that if L had a finite description, $L_1 = L \cup \{\epsilon\}$ would likewise have a finite description. We could add "ϵ is also in L_1" to the description of L to get a finite description of L_1.

We shall extend our definition of csg, cfg, and rg to allow productions of the form $S \to \epsilon$, where S is the start symbol, provided that S does not appear on the right-hand side of any production. In this case, it is clear that the production $S \to \epsilon$ can only be used as the first step in a derivation. We shall use the following lemma.

Lemma 2.1. If $G = (V_N, V_T, P, S)$ is a context-sensitive grammar, then there is another csg G_1 generating the same language as G, for which the start symbol of G_1 does not appear on the right of any production of G_1. Also, if G is a cfg, then such a cfg G_1 can be found. If G is an rg, then such an rg G_1 can be found.

Proof. Let S_1 be a symbol not in V_N or V_T. Let $G_1 = (V_N \cup \{S_1\}, V_T, P_1, S_1)$. P_1 consists of all the productions of P, plus all productions of the form $S_1 \to \alpha$ where $S \to \alpha$ is a production of P. Note that S_1 is not a symbol of V_N or V_T, so it does not appear on the right of any production of P_1.

We claim that $L(G) = L(G_1)$. For suppose that $S \xrightarrow[G]{*} w$. Let the first production used be $S \to \alpha$. Then we can write $S \xrightarrow[G]{} \alpha \xrightarrow[G]{*} w$. By definition of P_1, $S_1 \to \alpha$ is in P_1, so $S_1 \xrightarrow[G_1]{} \alpha$. Also, since P_1 contains all productions of P, $\alpha \xrightarrow[G_1]{*} w$. Thus $S_1 \xrightarrow[G_1]{*} w$. We can conclude that $L(G) \subseteq L(G_1)$.

---

† In most cases, we shall abbreviate the names of commonly used devices without periods to conform to current convention.

If we show that $L(G_1) \subseteq L(G)$, we prove that $L(G) = L(G_1)$. Suppose that $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} w$. The first production used is $S_1 \to \alpha$, for some α. Then, $S \to \alpha$ is a production of P, so $S \underset{G}{\Rightarrow} \alpha$. Now, $\alpha \overset{*}{\underset{G_1}{\Rightarrow}} w$, but α cannot have S_1 among its symbols. Since S_1 does not appear on the right of any production of P_1, no sentential form in the derivation $\alpha \overset{*}{\underset{G_1}{\Rightarrow}} w$ can involve S_1. Thus the derivation is also a derivation in grammar G; that is, $\alpha \overset{*}{\underset{G}{\Rightarrow}} w$. We conclude that $S \overset{*}{\underset{G}{\Rightarrow}} w$, and $L(G) = L(G_1)$.

It is easy to see that if G is a csg, cfg, or rg, G_1 will be likewise.

Theorem 2.1. If L is context sensitive, context free, or regular, then $L \cup \{\epsilon\}$ and $L - \{\epsilon\}$ are csl's, cfl's, or regular sets, respectively.

Proof. Given a csg, we can find by Lemma 2.1 an equivalent csg G, whose start symbol does not appear on the right of any production. Let $G = (V_N, V_T, P, S)$. Define $G_1 = (V_N, V_T, P_1, S)$, where P_1 is P plus the production $S \to \epsilon$. Note that S does not appear on the right of any production of P_1. Thus $S \to \epsilon$ cannot be used, except as the first and only production in a derivation. Any derivation of G_1 not involving $S \to \epsilon$ is a derivation in G, so $L(G_1) = L(G) \cup \{\epsilon\}$.

If the csg $G = (V_N, V_T, P, S)$ generates L, and ϵ is in L, then P must contain the production $S \to \epsilon$. Also S does not appear on the right of any production in P. Form grammar $G_1 = (V_N, V_T, P_1, S)$ where P_1 is $P - \{S \to \epsilon\}$. Since $S \to \epsilon$ cannot be used in the derivation of any word but ϵ, $L(G_1) = L - \{\epsilon\}$.

If L is context free or regular, the proof is analogous.

Example 2.5. Consider the grammar G of Example 2.2. We can find a grammar $G_1 = (\{S, S_1, B, C\}, \{a, b, c\}, P_1, S_1)$ generating $L(G)$ by defining P_1 to have the seven productions of P (see Example 2.2) plus the productions $S_1 \to aSBC$ and $S_1 \to aBC$. $L(G_1) = L(G) = \{a^n b^n c^n | n \geq 1\}$. We can add ϵ to $L(G_1)$ by defining grammar $G_2 = (\{S, S_1, B, C\}, \{a, b, c\}, P_2, S_1)$, where $P_2 = P_1 \cup \{S_1 \to \epsilon\}$. Then

$$L(G_2) = L(G_1) \cup \{\epsilon\} = \{a^n b^n c^n | n \geq 0\}.$$

2.5 RECURSIVENESS OF CONTEXT-SENSITIVE GRAMMARS

We say that a grammar G is *recursive* if there is an algorithm which will determine for any word w, whether w is generated by G. To say that a grammar is recursive is a stronger statement than to say that there is a procedure for enumerating sentences in the language generated by the grammar.†

† There is, of course, always such a procedure for any grammar.

Let $G = (V_N, V_T, P, S)$ be a csg. The sentence ϵ is in $L(G)$ if and only if P contains the production $S \to \epsilon$. Thus we have a test to see if ϵ is in $L(G)$. By removing $S \to \epsilon$ from P if it is there, we can form a new csg

$$G_1 = (V_N, V_T, P_1, S)$$

generating $L(G) - \{\epsilon\}$. Every production of P_1 satisfies the original restriction on a csg. That is, the right-hand side is at least as long as the left-hand side. As a consequence, in every derivation in G_1, the successive sentential forms are nondecreasing in length.

Let $V = V_N \cup V_T$ have k symbols. Suppose that $w \neq \epsilon$, and that there is a derivation $S \overset{*}{\underset{G_1}{\Rightarrow}} w$. Let this derivation be $S \to \alpha_1 \to \alpha_2 \cdots \to \alpha_m$, where $\alpha_m = w$. We have observed that $|\alpha_1| \leq |\alpha_2| \leq \cdots \leq |\alpha_m|$. Suppose that $\alpha_i, \alpha_{i+1}, \ldots, \alpha_{i+j}$ are all of the same length, say length p. Also, suppose that $j \geq k^p$. Then two of $\alpha_i, \alpha_{i+1}, \ldots, \alpha_{i+j}$ must be the same, for there are only k^p strings of length p in V^*. In this case, we can omit at least one step in the derivation. For, let $\alpha_r = \alpha_s$, where $r < s$. Then $S \to \alpha_1 \to \cdots \to \alpha_r \to \alpha_{s+1} \to \cdots \to \alpha_m = w$ is a shorter derivation of w in grammar G_1.

Intuitively, then, if there is a derivation of w, there is one which is "not too long." We shall give an algorithm in the next theorem which essentially incorporates this idea.

Theorem 2.2. If $G = (V_N, V_T, P, S)$ is a context-sensitive grammar, then G is recursive.

Proof. In the preceding paragraphs we saw that one could determine by inspection if ϵ was in $L(G)$ and then remove $S \to \epsilon$ from the productions if ϵ was there. We assume that P does not contain $S \to \epsilon$ and let w be a string in V_T^+. Suppose that $|w| = n$. Define the set T_m as the set of strings α in V^+, of length at most n, such that $S \overset{*}{\Rightarrow} \alpha$ by a derivation of at most m steps. Clearly, $T_0 = \{S\}$.

It is easy to see that we can calculate T_m from T_{m-1} by seeing what strings of length less than or equal to n can be derived from strings in T_{m-1} by a single application of a production. Formally,

$$T_m = T_{m-1} \cup \{\alpha | \text{ for some } \beta \text{ in } T_{m-1}, \beta \to \alpha \text{ and } |\alpha| \leq n\}.$$

Also, if $S \overset{*}{\Rightarrow} \alpha$, and $|\alpha| \leq n$, then α will be in T_m for some m; if S does not derive α, or $|\alpha| > n$, then α will not be in T_m for any m.

It should also be evident that $T_m \supseteq T_{m-1}$ for all $m \geq 1$. Since T_m depends only on T_{m-1}, if $T_m = T_{m-1}$, then $T_m = T_{m+1} = T_{m+2} = \cdots$. Our algorithm will be to calculate $T_1, T_2, T_3, \ldots$ until for some m, $T_m = T_{m-1}$. If w is not in T_m, then it is not in $L(G)$, because for $j > m$, $T_j = T_m$. Of course, if w is in T_m, then $S \overset{*}{\Rightarrow} w$.

We have now to show that for some m $T_m = T_{m-1}$. Recall that for each $i \geq 1$, $T_i \supseteq T_{i-1}$. If $T_i \neq T_{i-1}$, then the number of elements in T_i is at least one greater than the number in T_{i-1}. But, let V have k elements. Then the number of strings in V^+ of length less than or equal to n is $k + k^2 + \cdots + k^n$, which is less than or equal to $(k + 1)^{n+1}$. These are the only strings that may be in any T_i. Thus $T_m = T_{m-1}$ for some $m \leq (k + 1)^{n+1}$. Our procedure, which is to calculate T_i for all $i \geq 1$ until two equal sets are found, is thus guaranteed to halt. Therefore, it is an algorithm.

It should need no mention that the algorithm of Theorem 2.2 also applies to context-free and regular grammars.

Example 2.6. Consider the grammar G of Example 2.2, with productions:

1. $S \rightarrow aSBC$	5. $bB \rightarrow bb$
2. $S \rightarrow aBC$	6. $bC \rightarrow bc$
3. $CB \rightarrow BC$	7. $cC \rightarrow cc$
4. $aB \rightarrow ab$	

We determine if $w = abac$ is in $L(G)$, using the algorithm of Theorem 2.2.

$$T_0 = \{S\}.$$
$$T_1 = \{S, aSBC, aBC\}.$$

The first of these strings is in T_0, the second comes from S by application of production (1), the third, by application of (2).

$$T_2 = \{S, aSBC, aBC, abC\}.$$

The first three sentences of T_2 come from T_1, the fourth comes from aBC by application of (4). Note that although $aaSBCBC$ and $aaBCBC$ can be derived from $aSBC$ by productions (1) and (2), they are not in T_2, since their lengths are greater than $|w|$, which is 4. Similarly,

$$T_3 = \{S, aSBC, aBC, abC, abc\}.$$

We can easily see that $T_4 = T_3$. Since $abac$ is not in T_3, it is not in $L(G)$.

2.6 DERIVATION TREES FOR CONTEXT-FREE GRAMMARS

We now consider a visual method of describing any derivation in a context-free grammar. A *tree* is a finite set of *nodes* connected by directed *edges*, which satisfy the following three conditions (if an edge is directed from node 1 to node 2, we say the edge *leaves* node 1 and *enters* node 2):

1. There is exactly one node which no edge enters. This node is called the *root*.
2. For each node in the tree there exists a sequence of directed edges from the root to the node. Thus the tree is connected.

3. Exactly one edge enters every node except the root. As a consequence, there are no loops in the tree.

The set of all nodes n, such that there is an edge leaving a given node m and entering n, is called the set of *direct descendants* of m. A node n is called a *descendant* of node m if there is a sequence of nodes $n_1, n_2, \ldots, n_k$ such that $n_k = n$, $n_1 = m$, and for each i, n_{i+1} is a direct descendant of n_i. We shall, by convention, say a node is a descendant of itself.

For each node in the tree, we can order its direct descendants. Let n_1 and n_2 be direct descendants of node n, with n_1 appearing earlier in the ordering than n_2. Then we say that n_1 and all the descendants of n_1 are to the *left of* n_2 and all the descendants of n_2. Note that every node is a descendant of the root. If n_1 and n_2 are nodes, and neither is a descendant of the other, then they must both be descendants of some node. (This may not be obvious, but a little thought should suffice to make it clear.) Thus, one of n_1 and n_2 is to the left of the other.

Let $G = (V_N, V_T, P, S)$ be a cfg. A tree is a *derivation tree for* G if:

1. Every node has a *label*, which is a symbol of V.
2. The label of the root is S.
3. If a node n has at least one descendant other than itself, and has label A, then A must be in V_N.
4. If nodes $n_1, n_2, \ldots, n_k$ are the direct descendants of node n, in order from the left, with labels $A_1, A_2, \ldots, A_k$, respectively, then

$$A \rightarrow A_1 A_2 \ldots A_k$$

must be a production in P.

These ideas may be confusing, but an example should clarify things.

Example 2.7. Consider the grammar $G = (\{S, A\}, \{a, b\}, P, S)$, where P consists of:

$S \rightarrow aAS$	$S \rightarrow a$
$A \rightarrow SbA$	$A \rightarrow ba$
$A \rightarrow SS$	

We draw a tree, just this once with circles instead of points for the nodes. The nodes will be numbered for reference. The labels will be adjacent to the nodes. Edges are assumed to be directed downwards. See Fig. 2.2.

Some general comments will illustrate the definitions we have made. The label of node 1 is S. Node 1 is the root of the tree. Nodes 2, 3, and 4 are the direct descendants of node 1. Node 2 is to the left of nodes 3 and 4. Node 3 is to the left of node 4. Node 10 is a descendant of node 3, although not a direct descendant. Node 5 is to the left of node 10. Node 11 is to the left of node 4, for surely node 3 is to the left of node 4, and 11 is a descendant of node 3.

The nodes with direct descendants are 1, 3, 4, 5, and 7. Node 1 has label S, and its direct descendants, from the left, have labels a, A, and S. Note that $S \rightarrow aAS$ is a production. Likewise, node 3 has label A, and the labels of its direct descendants are S, b, and A from the left. $A \rightarrow SbA$ is also a production. Nodes 4 and 5 each have label S. Their only direct descendants each have label a, and $S \rightarrow a$ is a production. Lastly, node 7 has label A and its direct descendants, from the left, have labels b and a. $A \rightarrow ba$ is also a production. Thus, the conditions that Fig. 2.2 represent a derivation tree for G have been met.

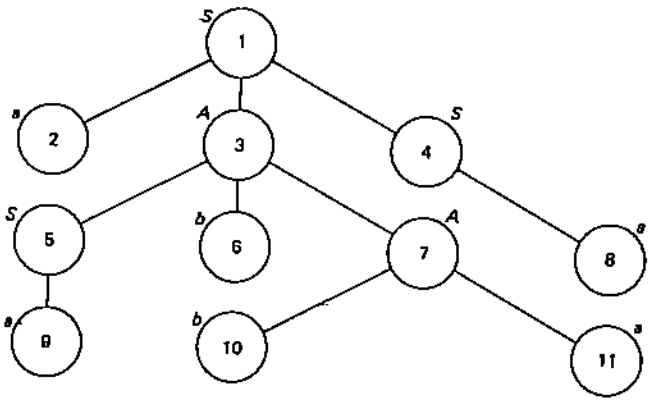

Fig. 2.2. Example of a derivation tree.

We shall see that a derivation tree is a very natural description of the derivation of a particular sentential form of the grammar G. Some of the nodes in any tree have no descendants. These nodes we shall call *leaves*. Given any two leaves, one is to the left of the other, and it is easy to tell which is which. Simply backtrack along the edges of the tree, toward the root, from each of the two leaves, until the first node of which both leaves are descendants is found.

If we read the labels of the leaves from left to right, we have a sentential form. We call this string the *result* of the derivation tree. Later, we shall see that if α is the result of some derivation tree for grammar $G = (V_N, V_T, P, S)$, then $S \xrightarrow[G]{*} \alpha$.

We need one additional concept, that of a *subtree*. A subtree of a derivation tree is a particular node of the tree together with all its descendants, the edges connecting them, and their labels. It looks just like a derivation tree, except that the label of the root may not be the start symbol of the grammar.

Example 2.8. Let us consider the grammar and derivation tree of Example 2.7. The derivation tree of Fig. 2.2 is reproduced without numbered nodes as Fig. 2.3(a). The result of the tree in Fig. 2.3(a) is *aabbaa*. Referring to Fig. 2.2 again, we see that the leaves are the nodes numbered 2, 9, 6, 10, 11, and 8, in that order, from the left. These nodes have labels a, a, b, b, a, a, respectively. Note that in this case all leaves had terminals for labels, but there is no reason why this should always be so. Note that $S \xrightarrow[G]{*} aabbaa$ by the derivation

$$S \rightarrow aAS \rightarrow aSbAS \rightarrow aabAS \rightarrow aabbaS \rightarrow aabbaa.$$

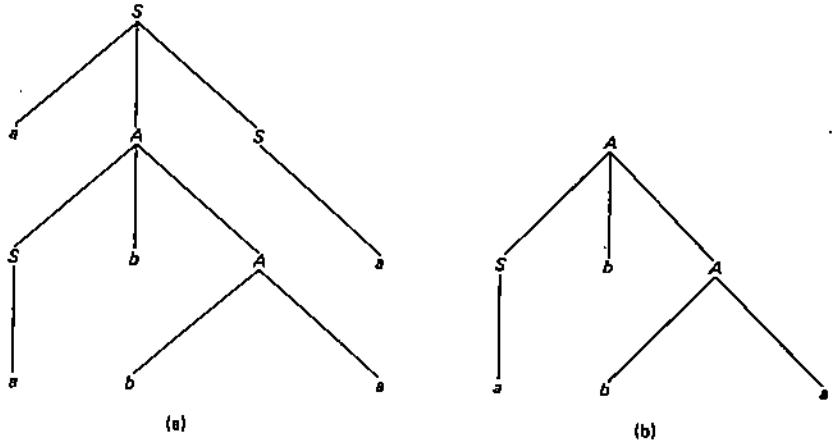

Fig. 2.3. Derivation trees and subtrees.

In part (b) of Fig. 2.3 is a subtree of the tree illustrated in part (a). It is node 3 of Fig. 2.2, together with its descendants. The result of the subtree is *abba*. The label of the root of the subtree is A, and $A \xrightarrow{*} abba$. The derivation in this case is:

$$A \rightarrow SbA \rightarrow abA \rightarrow abba.$$

We shall now prove a useful theorem about derivation trees for context-free grammars and, since every regular grammar is context free, for regular grammars also.

Theorem 2.3. Let $G = (V_N, V_T, P, S)$ be a context-free grammar. Then, for $\alpha \neq \epsilon$, $S \xrightarrow{*} \alpha$ if and only if there is a derivation tree in grammar G with result α.

Proof. We shall find it easier to prove something in excess of the theorem. What we shall prove is that if we define G_A to be the grammar (V_N, V_T, P, A) (i.e., G with the variable A chosen as the start symbol), then for any A in

V_N, $A \xrightarrow{*} \alpha$ if and only if there is a tree in grammar G_A with α as the result.† Note that for all grammars mentioned, the productions are the same. Therefore $A \xrightarrow[G_A]{*} \alpha$ is equivalent to saying $A \xrightarrow[G_B]{*} \alpha$, for any B in V_N. Also, since $G_S = G$, it is the same as saying $A \xrightarrow[G]{*} \alpha$.

Suppose, first, that α is the result of a derivation tree for grammar G_A. We prove, by induction on the number of nodes in the tree that are not leaves, that $A \xrightarrow[G_A]{*} \alpha$. If there is only one node that is not a leaf of the tree, the tree

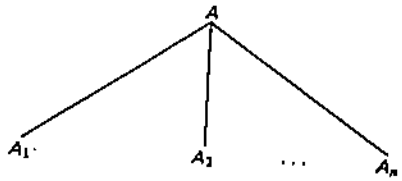

Fig. 2.4. Tree with one nonleaf.

must look like the one in Fig. 2.4. In that case, $A_1A_2 \ldots A_n$ must be α, and $A \to \alpha$ must be a production of P by definition of a derivation tree.

Now, suppose that the result is true for trees with up to $k-1$ nodes which are not leaves. Also, suppose that α is the result of a tree with root labeled A, and suppose that that tree has k nodes which are not leaves, $k > 1$. Consider the direct descendants of the root. These could not all be leaves. Let the labels of the direct descendants be $A_1, A_2, \ldots, A_n$ in order from the left. Number these nodes $1, 2, \ldots, n$. Then, surely, $S \to A_1A_2 \ldots A_n$ is a production in P. Note that n may be any integer greater than or equal to one in the argument that follows.

If the node i is not a leaf, it is the root of a subtree. Also, A_i must be a variable. The subtree is a tree in grammar G_{A_i}, and has some result α_i. If node i is a leaf, let $A_i = \alpha_i$. It is easy to see that if $j < i$, node j and all of its descendants are to the left of node i and all of its descendants. Thus $\alpha = \alpha_1\alpha_2 \ldots \alpha_n$. A subtree must have fewer nodes that are not leaves than its tree does, unless the subtree is the entire tree. By the inductive hypothesis, for each node i which is not a leaf, $A_i \xrightarrow[G_{A_i}]{*} \alpha_i$. Thus $A_i \xrightarrow[G_A]{*} \alpha_i$. If $A_i = \alpha_i$, then surely $A_i \xrightarrow[G_A]{*} \alpha_i$. We can put all these partial derivations together, to see that

$$A \xrightarrow[G_A]{} A_1A_2 \ldots A_n \xrightarrow[G_A]{*} \alpha_1A_2 \ldots A_n \xrightarrow[G_A]{*} \alpha_1\alpha_2A_3 \ldots A_n \xrightarrow[G_A]{*} \cdots \xrightarrow[G_A]{*} \alpha_1\alpha_2 \ldots \alpha_n = \alpha.$$

Thus $A \xrightarrow[G_A]{*} \alpha$.

† The introduction of these grammars is necessary only because a tree in grammar G always has a root labeled S.

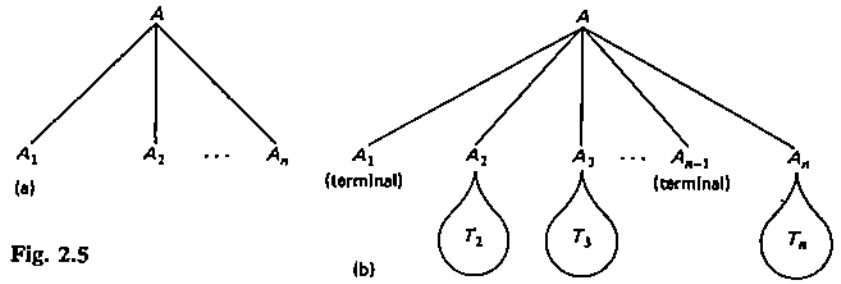

Fig. 2.5

Now, suppose that $A \xrightarrow[G_A]{*} \alpha$. We must show that there is a derivation tree with result α in grammar G_A. If $A \xrightarrow[G_A]{*} \alpha$ by a single step, then $A \to \alpha$ is a production in P, and there is a tree with result α, of the form shown in Fig. 2.4.

Now, assume that if $A \xrightarrow[G_A]{*} \alpha$ by a derivation of less than k steps, then there is a derivation tree in grammar G_A with result α. Suppose that $A \xrightarrow[G_A]{*} \alpha$ by a derivation of k steps. Let the first step be $A \to A_1A_2 \ldots A_n$. Now, it should be clear that any symbol in α must either be one of $A_1, A_2, \ldots, A_n$ or be derived from one of these. Also, that portion of α derived from A_i must lie to the left of the symbols derived from A_j, if $i < j$. Thus, we can write α as $\alpha_1\alpha_2 \ldots \alpha_n$, where for each i between 1 and n, $A_i \xrightarrow[G_{A_i}]{*} \alpha_i$.

By the inductive hypothesis, there is a derivation tree for each variable A_i, in grammar G_{A_i}, with result α_i. Let this tree be T_i. We begin by constructing a derivation tree in grammar G_A with root labeled A, and n leaves labeled $A_1, A_2, \ldots, A_n$, and no other nodes. This tree is shown in Fig. 2.5(a). Each node with label A_i, where A_i is not a terminal, is replaced by the tree T_i. If A_i is a terminal, no replacement is made. An example appears in Fig. 2.5(b). In a straightforward manner, it can be shown that the result of this tree is α.

Fig. 2.6

Example 2.9. Consider the derivation $S \xrightarrow{*} aabbaa$ of Example 2.8. The first step is $S \to aAS$. If we follow the derivation, we see that A eventually is replaced by SbA, then by abA, and finally, by $abba$. Part (b) of Fig. 2.3 is a derivation tree for this derivation. The only symbol derived from S in aAS is a. (This replacement is the last step.) Part (a) of Fig. 2.6 is a tree for the latter derivation.

Part (b) of Fig. 2.6 is the derivation tree for $S \to aAS$. If we replace the node with label A in Fig. 2.6(b) by the tree of Fig. 2.3(b), and the node with label S in Fig. 2.6(b) with the tree of Fig. 2.6(a), we get the tree of Fig. 2.3(a), whose result is *aabbaa*.

PROBLEMS

2.1 Give a regular grammar generating

$L = \{w | w \text{ is in } \{0, 1\}^*, \text{ and } w \text{ does not contain two consecutive 1's}\}$.

2.2 Give a context-free grammar generating

$L = \{w | w \text{ is in } \{a, b\}^* \text{ and } w \text{ consists of twice as many } a\text{'s as } b\text{'s}\}$.

2.3 Give a context-free grammar generating the FORTRAN arithmetic statements.

2.4 Give a context-sensitive grammar generating

$L = \{w | w \text{ in } \{a, b, c\}^*, \text{ and } w \text{ consists of equal numbers of } a\text{'s, } b\text{'s, and } c\text{'s}\}$.

2.5 Give a context-sensitive grammar generating

$$L = \{ww | w \text{ is in } \{0, 1\}^*\}.$$

That is, L is all words in $\{0, 1\}^*$ whose first and last halves are equal.

2.6 Informally describe the words generated by the grammar G of Example 2.7.

2.7 Use the algorithm of Theorem 2.2 to determine if the following words are in $L(G)$, where G is as in Example 2.7.

a) *abaa* b) *abbb* c) *baaba*

2.8 If G is context free, can you improve upon the bound on m in Theorem 2.2? What if G is regular?

2.9 Consider the grammar G of Example 2.3. Draw a derivation tree in G for the following words.

a) *ababab* b) *bbbaabaa* c) *aabbaabb*

2.10 Let $G = (V_N, V_T, P, S)$, where $V_N = \{A, B, S\}$ and $V_T = \{0, 1\}$. P consists of the productions:

$$\begin{array}{ll} S \to 0AB & B \to 01 \\ 1B \to 0 & A1 \to SB1 \\ B \to SA & A0 \to S0B \end{array}$$

Can you prove that $L(G)$ is empty?

2.11 In Fig. 2.7 is a derivation tree of some context-free grammar,

$$G = (V_N, V_T, P, S),$$

for which the productions and symbols are not known. What is the result of the tree? What symbols are necessarily in V_N? What symbols might be in V_T? Disregarding our convention that lower case italic letters denote terminals, do we find that b and c must be in V_T, or could they be in V_N? What productions must be in P? Is the word *bcbbcbb* in $L(G)$?

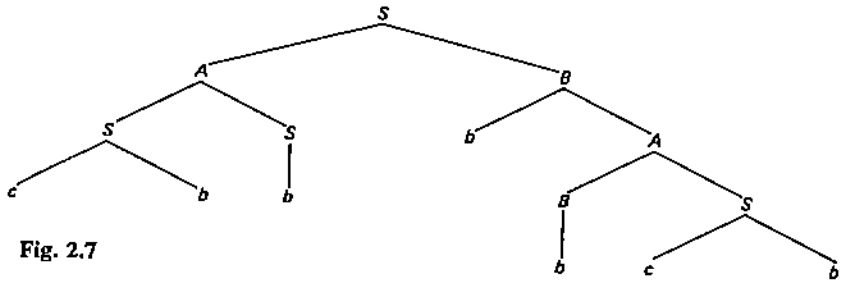

Fig. 2.7

2.12 Let G be a grammar where all productions are of the form $A \to xB$ and $A \to x$, where A and B are single variables and x is a string of terminals. Show that $L(G)$ can be generated by a regular grammar.

REFERENCES

Early works on generating systems are found in Chomsky [1956], Chomsky and Miller [1958], Chomsky [1959], and Bar-Hillel, Gaifman, and Shamir [1960]. The notation of grammar used here and the classification by type is due to Chomsky [1959].

For references on regular, context-free, recursively enumerable, and context-sensitive sets, check the references given at the end of Chapters 3, 4, 6, and 8, respectively. Two survey papers with additional references are Chomsky [1963] and Floyd [1964c].