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

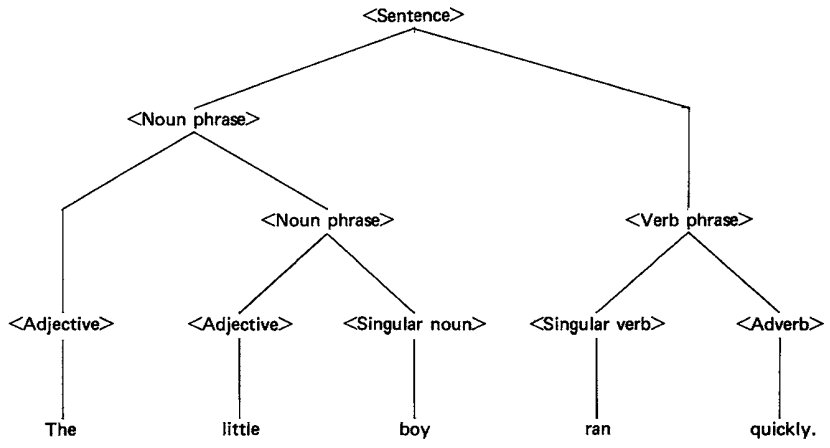


Fig. 2.1. A diagram of the sentence "The little boy ran quickly."

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 $\underset{G}{\Rightarrow}$ and $\overset{*}{\underset{G}{\Rightarrow}}$ between strings in V^*. Specifically, if $\alpha \to \beta$ is a production of P and γ and δ are any strings in V^*, then $\gamma\alpha\delta \underset{G}{\Rightarrow} \gamma\beta\delta\ddagger$. We say that the production $\alpha \to \beta$ is applied to the string $\gamma\alpha\delta$ to obtain $\gamma\beta\delta$. Thus $\underset{G}{\Rightarrow}$ 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 $\alpha_1 \underset{G}{\Rightarrow} \alpha_2$, $\alpha_2 \underset{G}{\Rightarrow} \alpha_3$, $\ldots$, $\alpha_{m-1} \underset{G}{\Rightarrow} \alpha_m$. Then we say $\alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_m$.† In simple terms, we say for two strings α and β that $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$ if we can obtain β from α by application of some number of productions of P. By convention, $\alpha \overset{*}{\underset{G}{\Rightarrow}} \alpha$ for each string α.

We define the *language generated by G* [denoted $L(G)$] to be $\{w | w$ is in V_T^* and $S \overset{*}{\underset{G}{\Rightarrow}} w\}$.‡ 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 $S \overset{*}{\Rightarrow} \alpha$. (Usually, if it is clear which grammar G is involved, we use $\Rightarrow$ for $\underset{G}{\Rightarrow}$ and $\overset{*}{\Rightarrow}$ for $\overset{*}{\underset{G}{\Rightarrow}}$.)

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

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

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, $\{ww | w$ 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. $S \rightarrow aSBC$
2. $S \rightarrow aBC$
3. $CB \rightarrow BC$
4. $aB \rightarrow ab$
5. $bB \rightarrow bb$
6. $bC \rightarrow bc$
7. $cC \rightarrow cc$

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

$$aaaBCBCBC \Rightarrow aaaBBCCBC \Rightarrow aaaBBCBCC \Rightarrow aaaBBBCCC.$$

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

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

Now, let us show that the words $a^nb^nc^n$ for $n \geqq 1$ 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 $n \geqq 1$, 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, $L(G) = \{a^nb^nc^n | n \geqq 1\}$.

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, $|\beta| \geqq |\alpha|$.† 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 $\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$, 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 $\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$ 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.

$$\begin{array}{ll} S \to aB & A \to bAA \\ S \to bA & B \to b \\ A \to a & B \to bS \\ A \to aS & B \to aBB \end{array}$$

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. $S \overset{*}{\Longrightarrow} w$ if and only if w consists of an equal number of a's and b's.
2. $A \overset{*}{\Longrightarrow} w$ if and only if w has one more a than it has b's.
3. $B \overset{*}{\Longrightarrow} w$ 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 $A \overset{*}{\Rightarrow} a$, $B \overset{*}{\Rightarrow} b$, 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 $S \overset{*}{\Rightarrow} w$, 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 $B \overset{*}{\Rightarrow} w_1$. 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 $S \overset{*}{\Rightarrow} w$. 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, $B \overset{*}{\Rightarrow} w_1$. But then $S \Rightarrow aB \overset{*}{\Rightarrow} aw_1 = w$. 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 $S \overset{*}{\underset{G}{\Rightarrow}} w$. Let the first production used be $S \to \alpha$. Then we can write $S \underset{G}{\Rightarrow} \alpha \overset{*}{\underset{G}{\Rightarrow}} w$. By definition of P_1, $S_1 \to \alpha$ is in P_1, so $S_1 \underset{G_1}{\Rightarrow} \alpha$. Also, since P_1 contains all productions of P, $\alpha \overset{*}{\underset{G_1}{\Rightarrow}} w$. Thus $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} w$. 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 $S_1 \to aBC$. $L(G_1) = L(G) = \{a^n b^n c^n | n \geqq 1\}$. 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

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

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 $S \overset{*}{\underset{G_1}{\Longrightarrow}} w$. Let this derivation be $S \Longrightarrow \alpha_1 \Longrightarrow \alpha_2 \cdots \Longrightarrow \alpha_m$, where $\alpha_m = w$. We have observed that $|\alpha_1| \leqq |\alpha_2| \leqq \cdots \leqq |\alpha_m|$. Suppose that $\alpha_i, \alpha_{i+1}, \ldots, \alpha_{i+j}$ are all of the same length, say length p. Also, suppose that $j \geqq k^p$. 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 $S \Longrightarrow \alpha_1 \Longrightarrow \cdots \Longrightarrow \alpha_r \Longrightarrow \alpha_{s+1} \Longrightarrow \cdots \Longrightarrow \alpha_m = w$ 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 $S \overset{*}{\Longrightarrow} \alpha$ 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,

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

Also, if $S \overset{*}{\Longrightarrow} \alpha$, and $|\alpha| \leqq n$, 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 $m \geqq 1$. 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 $S \overset{*}{\Longrightarrow} w$.

We have now to show that for some m $T_m = T_{m-1}$. Recall that for each $i \geqq 1$, $T_i \supseteq T_{i-1}$. 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 $m \leqq (k + 1)^{n+1}$. Our procedure, which is to calculate T_i for all $i \geqq 1$ 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. $S \rightarrow aSBC$
2. $S \rightarrow aBC$
3. $CB \rightarrow BC$
4. $aB \rightarrow ab$
5. $bB \rightarrow bb$
6. $bC \rightarrow bc$
7. $cC \rightarrow cc$

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

$$A \to A_1 A_2 \ldots A_k$$

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:

$$\begin{array}{ll} S \to aAS & S \to a \\ A \to SbA & A \to ba \\ A \to SS & \end{array}$$

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.

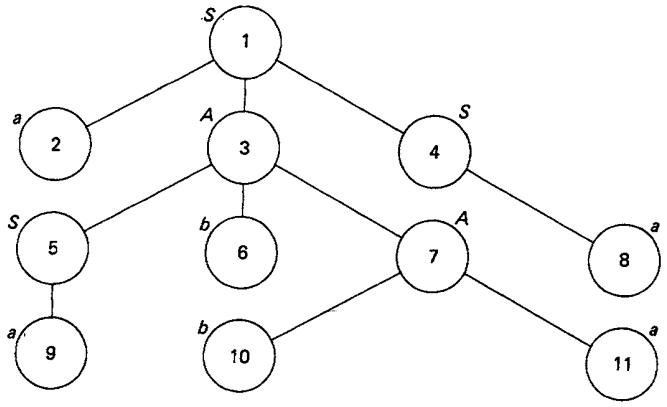


Fig. 2.2. Example of a derivation tree.

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 $S \overset{*}{\underset{G}{\Rightarrow}} \alpha$.

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 $S \overset{*}{\underset{G}{\Rightarrow}} aabbaa$ by the derivation

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

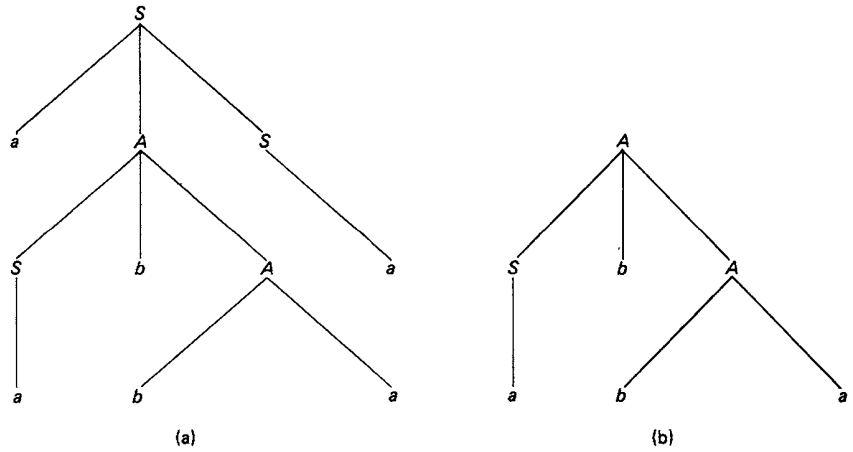


Fig. 2.3. Derivation trees and subtrees.

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 $A \overset{*}{\Rightarrow} abba$. 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 $\alpha \neq \epsilon$, $S \overset{*}{\Rightarrow} \alpha$ 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, $A \overset{*}{\Rightarrow} \alpha$ 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 $A \overset{*}{\underset{G_A}{\Rightarrow}} \alpha$ is equivalent to saying $A \overset{*}{\underset{G_B}{\Rightarrow}} \alpha$, for any B in V_N. Also, since $G_S = G$, it is the same as saying $A \overset{*}{\underset{G}{\Rightarrow}} \alpha$.

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 $A \overset{*}{\underset{G_A}{\Rightarrow}} \alpha$. If there is only one node that is not a leaf of the tree, the tree

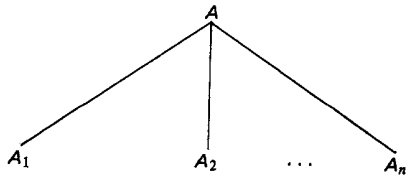


Fig. 2.4. Tree with one nonleaf.

must look like the one in Fig. 2.4. In that case, $A_1A_2 \ldots A_n$ must be α, and $A \rightarrow \alpha$ 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, $S \rightarrow A_1A_2 \ldots A_n$ 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, $A_i \overset{*}{\underset{G_{A_i}}{\Rightarrow}} \alpha_i$. Thus $A_i \overset{*}{\underset{G_A}{\Rightarrow}} \alpha_i$. If $A_i = \alpha_i$, then surely $A_i \overset{*}{\underset{G_A}{\Rightarrow}} \alpha_i$. We can put all these partial derivations together, to see that

$$A \underset{G_A}{\Rightarrow} A_1A_2 \ldots A_n \overset{*}{\underset{G_A}{\Rightarrow}} \alpha_1A_2 \ldots A_n \overset{*}{\underset{G_A}{\Rightarrow}} \alpha_1\alpha_2A_3 \ldots A_n \overset{*}{\underset{G_A}{\Rightarrow}} \cdots \overset{*}{\underset{G_A}{\Rightarrow}} \alpha_1\alpha_2 \ldots \alpha_n = \alpha.$$

Thus $A \overset{*}{\underset{G_A}{\Rightarrow}} \alpha$.

---

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

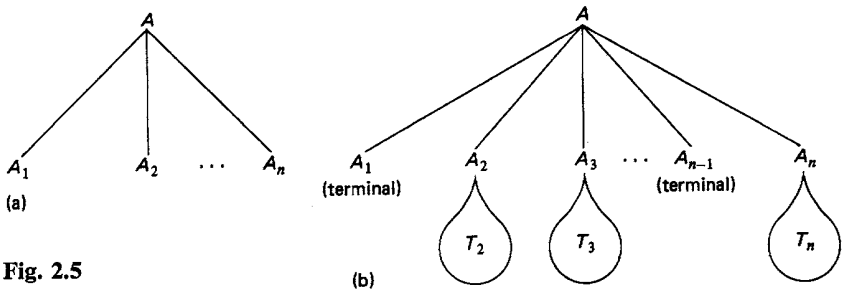


Fig. 2.5

Now, suppose that $A \overset{*}{\underset{G_A}{\Longrightarrow}} \alpha$. We must show that there is a derivation tree with result α in grammar G_A. If $A \overset{*}{\underset{G_A}{\Longrightarrow}} \alpha$ 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 $A \overset{*}{\underset{G_A}{\Longrightarrow}} \alpha$ by a derivation of less than k steps, then there is a derivation tree in grammar G_A with result α. Suppose that $A \overset{*}{\underset{G_A}{\Longrightarrow}} \alpha$ by a derivation of k steps. Let the first step be $A \Longrightarrow A_1A_2 \ldots A_n$. 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, $A_i \overset{*}{\underset{G_{A_i}}{\Longrightarrow}} \alpha_i$.

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.

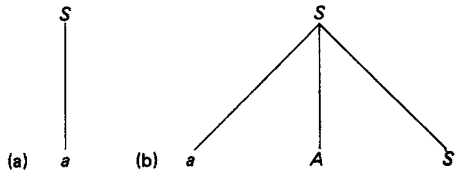


Fig. 2.6

Example 2.9. Consider the derivation $S \overset{*}{\Longrightarrow} aabbaa$ 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 $S \Rightarrow aAS$. 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

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

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 $L(G)$?

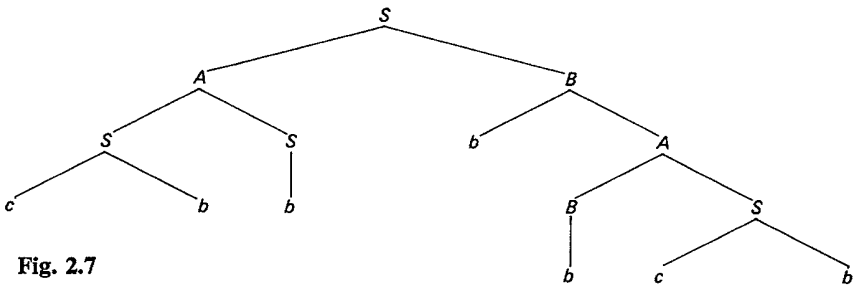


Fig. 2.7

2.12 Let G be a grammar where all productions are of the form $A \rightarrow xB$ and $A \rightarrow x$, 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].

CHAPTER 3

FINITE AUTOMATA AND REGULAR GRAMMARS

3.1 THE FINITE AUTOMATON

In Chapter 2, we were introduced to a generating scheme—the grammar. Grammars are finite specifications for languages. In this chapter we shall see another method of finitely specifying infinite languages—the recognizer. We shall consider what is undoubtedly the simplest recognizer, called a finite automaton. The finite automaton (fa) cannot define all languages defined by grammars, but we shall show that the languages defined are exactly the type 3 languages. In later chapters, the reader will be introduced to recognizers for type 0, 1, and 2 languages. Here we shall define a finite automaton as a formal system, then give the physical meaning of the definition.

A *finite automaton* M over an alphabet Σ is a system $(K, \Sigma, \delta, q_0, F)$, where K is a finite, nonempty set of *states*, Σ is a finite *input alphabet*, δ is a mapping of $K \times \Sigma$ into K, q_0 in K is the *initial state*, and $F \subseteq K$ is the set of *final states*.

Our model in Fig. 3.1 represents a finite control which reads symbols from a linear input tape in a sequential manner from left to right. The set of states K consists of the states of the finite control. Initially, the finite control is in state q_0 and is scanning the leftmost symbol of a string of symbols in Σ which appear on the input tape. The interpretation of $\delta(q, a) = p$, for q and p in K and a in Σ, is that M, in state q and scanning the input symbol a, moves its input head one cell to the right and goes to state p.

The mapping δ is from $K \times \Sigma$ to K. We can extend δ to domain† $K \times \Sigma^*$ by defining a mapping $\hat{\delta}$ as follows:

$$\hat{\delta}(q, \epsilon) = q$$
$$\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a) \qquad \text{for each } x \text{ in } \Sigma^* \text{ and } a \text{ in } \Sigma.$$

Thus the interpretation of $\hat{\delta}(q, x) = p$ is that M, starting in state q with the string x written on the input tape, will be in state p when the input head moves right from the portion of the input tape containing x. Since δ and $\hat{\delta}$

† The *domain* of a mapping is the set of valid arguments for the mapping. The set of values which the mapping could take is called the *range*.

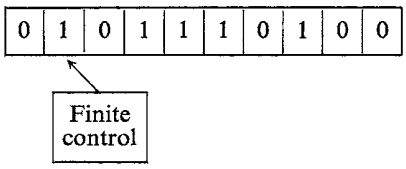


Fig. 3.1. A finite automaton.

agree wherever δ is defined, no confusion will arise if we fail to distinguish between δ and $\hat{\delta}$. Thus, for the remainder of the book, we shall use δ for both δ and $\hat{\delta}$.

A sentence x is said to be *accepted* by M if $\delta(q_0, x) = p$ for some p in F. The set of all x accepted by M is designated $T(M)$. That is,

$$T(M) = \{x | \delta(q, x) \text{ is in } F\}.$$

Any set of strings accepted by a finite automaton is said to be *regular*.

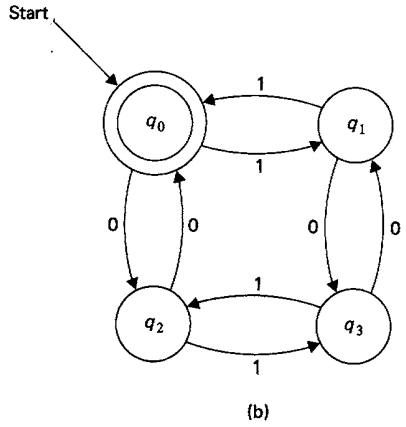
Example 3.1. The specifications for a finite automaton are given in Fig. 3.2(a). A *state diagram* for the automaton is shown in Fig. 3.2(b). The state diagram consists of a node for every state and a directed line from state q to state p with label a (in Σ) if the finite automaton, in state q, scanning the input symbol a, would go to state p. Final states, i.e., states in F, are indicated by a double circle. The initial state is marked by an arrow labeled start.

Consider the state diagram of Fig. 3.2(b). Suppose that 110101 is the input to M. Since $\delta(q_0, 1) = q_1$ and $\delta(q_1, 1) = q_0$, $\delta(q_0, 11) = q_0$. We might comment that thus, 11 is in $T(M)$, but we are interested in 110101. Now $\delta(q_0, 0) = q_2$, so $\delta(q_0, 110) = q_2$. Next $\delta(q_2, 1) = q_3$, so $\delta(q_0, 1101) = q_3$. Finally, $\delta(q_3, 0) = q_1$ and $\delta(q_1, 1) = q_0$, so $\delta(q_0, 110101) = q_0$, and thus 110101 is in $T(M)$. It is easily shown that $T(M)$ is the set of all sentences in $\{0, 1\}^*$ containing both an even number of 0's and an even number of 1's.

$M = (K, \Sigma, \delta, q_0, F)$
$\Sigma = \{0, 1\}$
$K = \{q_0, q_1, q_2, q_3\}$
$F = \{q_0\}$

$\delta(q_0, 0) = q_2 \quad \delta(q_0, 1) = q_1$
$\delta(q_1, 0) = q_3 \quad \delta(q_1, 1) = q_0$
$\delta(q_2, 0) = q_0 \quad \delta(q_2, 1) = q_3$
$\delta(q_3, 0) = q_1 \quad \delta(q_3, 1) = q_2$

(a)



(b)

Fig. 3.2. A finite automaton accepting the set of strings with an even number of 0's and an even number of 1's. (a) A finite automaton. (b) State diagram of the finite automaton.

3.2 EQUIVALENCE RELATIONS AND FINITE AUTOMATA

A *binary relation* R on a set S is a set of pairs of elements in S. If (a, b) is in R, then we are accustomed to seeing this fact written as aRb.

Example 3.2. For a familiar example, consider the relation "less than" usually denoted by the symbol $<$ on the set of integers. In the formal sense, this relation is the set: $\{(i, j) \mid i \text{ is less than } j\}$. Thus $3 < 4$, $2 < 17$, etc.

We are going to be concerned with some relations on sets of strings over a finite alphabet.

A binary relation R over a set S is said to be:

1. *reflexive* if for each s in S, sRs,
2. *symmetric* if for s and t in S, sRt implies tRs,
3. *transitive* if for s, t, and u in S, sRt and tRu imply sRu.

A relation which is reflexive, symmetric, and transitive is called an *equivalence relation.* An example of an equivalence relation over the set of positive integers is the relation E, given by: iEj if and only if $|i - j|$ is divisible by 3.

An important property of equivalence relations is that if R is an equivalence relation on the set S then we can divide S into k disjoint subsets, called *equivalence classes*, for some k between 1 and infinity, inclusive, such that aRb if and only if a and b are in the same subset.

The proof is simple. Define $[a]$ to be $\{b|aRb\}$. For any a and b in S, either $[a] = [b]$, or $[a]$ and $[b]$ are disjoint. Otherwise, let c be in $[a]$ and $[b]$, and d be in $[b]$ but not $[a]$. That is, aRc, bRc, and bRd, but not aRd. By symmetry, we have cRb. By transitivity, we can show cRd and aRd. The latter statement is a contradiction. The distinct sets that are $[a]$ for some a in S are the equivalence classes. Clearly, a and b are in the same set if and only if they are equivalent.

Example 3.3. The relation E given by iEj if and only if $|i - j|$ is divisible by 3 divides the set of positive integers into three classes $\{1, 4, 7, 10, \ldots\}$, $\{2, 5, 8, 11, \ldots\}$, and $\{3, 6, 9, 12, \ldots\}$. Any two elements from the same class are equivalent ($1E4$, $3E6$, etc.), and any two elements from different classes fail to satisfy the equivalence relation (not $7E9$, $1E5$, etc.).

The *index* of an equivalence relation is the number of equivalence classes generated. Thus the equivalence relation E has index 3.

Consider the finite automaton of Example 3.1. For x and y in $\{0, 1\}^*$, let (x, y) be in R if and only if $\delta(q_0, x) = \delta(q_0, y)$. The relation R is reflexive, symmetric, and transitive, since "$=$" has these properties, and thus, R is an equivalence relation. R divides the set $\{0, 1\}^*$ into four equivalence classes corresponding to the four states. In addition, if xRy, then $xz\ R\ yz$ for all z in $\{0, 1\}^*$, since

$$\delta(q_0, xz) = \delta(\delta(q_0, x), z) = \delta(\delta(q_0, y), z) = \delta(q_0, yz).$$

Such an equivalence relation is said to be *right invariant*. We see that every finite automaton induces a right invariant equivalence relation defined as R was defined, on its set of input strings. This result is formalized in the following theorem.

Theorem 3.1. The following three statements are equivalent:

1. The set $L \subseteq \Sigma^*$ is accepted by some finite automaton.
2. L is the union of some of the equivalence classes of a right invariant equivalence relation of finite index.
3. Let equivalence relation R be defined by: xRy if and only if for all z in Σ^*, xz is in L exactly when yz is in L. Then R is of finite index.

Proof. (1) $\Rightarrow$ (2). Assume that L is accepted by some fa $M = (K, \Sigma, \delta, q_0, F)$. Let R' be the equivalence relation $xR'y$ if and only if $\delta(q_0, x) = \delta(q_0, y)$. R' is right invariant since, for any z, if $\delta(q_0, x) = \delta(q_0, y)$, then

$$\delta(q_0, xz) = \delta(q_0, yz).$$

The index of R' is finite since the index is at most the number of states in K. Furthermore, L is the union of those equivalence classes which include an element x such that $\delta(q_0, x)$ is in F.

(2) $\Rightarrow$ (3). We show that any equivalence relation R' satisfying (2) is a refinement of R; that is, every equivalence class of R' is entirely contained in some equivalence class of R. Thus the index of R cannot be greater than the index of R' and so is finite. Assume that $xR'y$. Then since R' is right invariant, for each z in Σ^*, $xzR'yz$, and thus yz is in L if and only if xz is in L. Thus xRy, and hence, the equivalence class of x in R' is contained in the equivalence class of x in R. We conclude that each equivalence class of R' is contained within some equivalence class of R.

(3) $\Rightarrow$ (1). Assume that xRy. Then for each w and z in Σ^*, xwz is in L if and only if ywz is in L. Thus $xwRyw$, and R is right invariant. Now let K' be the finite set of equivalence classes of R and $[x]$ the element of K' containing x. Define $\delta'([x], a) = [xa]$. The definition is consistent, since R is right invariant. Let $q_0' = [\epsilon]$ and let $F' = \{[x] \mid x \in L\}$. The finite automaton $M' = (K', \Sigma, \delta', q_0', F')$ accepts L since $\delta'(q_0', x) = [x]$, and thus x is in $T(M')$ if and only if $[x]$ is in F'.

Theorem 3.2. The minimum state automaton accepting L is unique up to an isomorphism (i.e., a renaming of the states) and is given by M' of Theorem 3.1.

Proof. In the proof of Theorem 3.1 we saw that any fa $M = (K, \Sigma, \delta, q_0, F)$ accepting L defines an equivalence relation which is a refinement of R. Thus the number of states of M is greater than or equal to the number of states of M' of Theorem 3.1. If equality holds, then each of the states of M can be identified with one of the states of M'. That is, let q be a state of M. There

must be some x in Σ^*, such that $\delta(q_0, x) = q$, otherwise q could be removed from K, and a smaller automaton found. Identify q with the state $\delta'(q_0', x)$, of M'. This identification will be consistent. If $\delta(q_0, x) = \delta(q_0, y) = q$, then, by Theorem 3.1, x and y are in the same equivalence class of R. Thus $\delta'(q_0', x) = \delta'(q_0', y)$.

3.3 NONDETERMINISTIC FINITE AUTOMATA

We now introduce the notion of a nondeterministic finite automaton. It will turn out that any set accepted by a nondeterministic finite automaton can also be accepted by a deterministic finite automaton.

However, the nondeterministic finite automaton is a useful concept in proving theorems. Also, the concept of a nondeterministic device is not an easy one to grasp. It is well to begin with a simple device. Later we deal with nondeterministic devices that are not equivalent to their deterministic counterparts. It is hoped that the study of nondeterministic finite automata will help in the understanding of those devices.

A *nondeterministic finite automaton* M is a system $(K, \Sigma, \delta, q_0, F)$, where K is a finite nonempty set of states, Σ the finite input alphabet, δ is a mapping of $K \times \Sigma$ into subsets of K, q_0 in K is the initial state, and $F \subseteq K$ is the set of final states.

The important difference between the deterministic and nondeterministic case is that $\delta(q, a)$ is a (possibly empty) set of states rather than a single state. The interpretation of $\delta(q, a) = \{p_1, p_2, \ldots, p_k\}$ is that M, in state q, scanning a on its input tape, moves its head one cell to the right and chooses any one of $p_1, p_2, \ldots, p_k$ as the next state.

The mapping δ can be extended to domain $K \times \Sigma^*$ by defining

$$\delta(q, \epsilon) = \{q\} \qquad \text{and} \qquad \delta(q, xa) = \bigcup_{p \text{ in } \delta(q,x)} \delta(p, a),$$

for each x in Σ^*, and a in Σ.

The mapping δ can be further extended to domain $2^K \times \Sigma^*$† by defining

$$\delta(\{p_1, p_2, \ldots, p_k\}, x) = \bigcup_{i=1}^{k} \delta(p_i, x).$$

A sentence x is *accepted* by M if there is a state p in both F and $\delta(q_0, x)$. The set of all x accepted by M is denoted $T(M)$.

Example 3.4. A nondeterministic fa which accepts the set of all sentences with either two consecutive 0's or two consecutive 1's is given in Fig. 3.3. The fa will make many choices upon reading an input string. Thus, suppose that 010110 is the input. After reading the first 0, M may stay in state q_0 or go to q_3. Next, with a 1 input, M can go nowhere from q_3, but from q_0 can

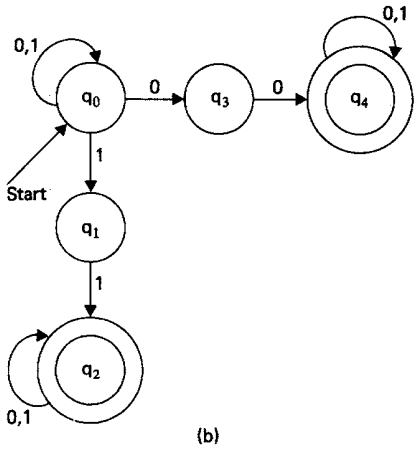
† 2^K, for any set K, denotes the *power set* or set of all subsets of K.

go to q_0 or q_1. Similarly, by the time the fourth input symbol is read, M can still be in only q_0 or q_1. When the fifth symbol, a 1, is read, M can go from q_1 to q_2 and from q_0 to q_0 or q_1. Thus M may be in state q_0, q_1, or q_2. Since there is a sequence of states leading to q_2, 01011 is accepted. Likewise, after the sixth symbol is read, M can be in state q_0, q_2, or q_3. Thus 010110 is also accepted.

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \delta, q_0, \{q_2, q_4\})$$

$$\begin{aligned}
&\delta(q_0, 0) = \{q_0, q_3\};\\
&\delta(q_1, 0) = \varphi;\\
&\delta(q_2, 0) = \{q_2\};\\
&\delta(q_3, 0) = \{q_4\};\\
&\delta(q_4, 0) = \{q_4\};\\
&\\
&\delta(q_0, 1) = \{q_0, q_1\}.\\
&\delta(q_1, 1) = \{q_2\}.\\
&\delta(q_2, 1) = \{q_2\}.\\
&\delta(q_3, 1) = \varphi.\\
&\delta(q_4, 1) = \{q_4\}.
\end{aligned}$$

(a)



(b)

Fig. 3.3. A nondeterministic finite automaton which accepts the set of all sentences containing either two consecutive 0's or two consecutive 1's. (a) Specification. (b) State diagram.

Theorem 3.3. Let L be a set accepted by a nondeterministic finite automaton. Then there exists a deterministic finite automaton that accepts L.

Proof. Let $M = (K, \Sigma, \delta, q_0, F)$ be a nondeterministic fa accepting L. Define a deterministic fa, $M' = (K', \Sigma, \delta', q_0', F')$ as follows. The states of M' are all the subsets of the set of states of M. That is, $K' = 2^K$. M' will keep track of all the states M could be in at any given time. F' is the set of all states in K' containing a state of F. An element of K' will be denoted by $[q_1, q_2, \ldots, q_i]$, where $q_1, q_2, \ldots, q_i$ are in K. Note that $q_0' = [q_0]$.

We define

$$\delta'([q_1, q_2, \ldots, q_i], a) = [p_1, p_2, \ldots, p_j]$$

if and only if

$$\delta(\{q_1, q_2, \ldots, q_i\}, a) = \{p_1, p_2, \ldots, p_j\}.$$

That is, δ' applied to an element Q of K' is computed by applying δ to each state of K represented by $Q = [q_1, q_2, \ldots, q_i]$. On applying δ to each of $q_1, q_2, \ldots, q_i$ and taking the union, we get some new set of states, $p_1, p_2, \ldots, p_j$. This new set of states has a representative, $[p_1, p_2, \ldots, p_j]$ in K', and that element is the value of $\delta'([q_1, q_2, \ldots, q_i], a)$.

It is easy to show by induction on the length of the input string x that

$$\delta'(q_0', x) = [q_1, q_2, \ldots, q_i]$$

if and only if

$$\delta(q_0, x) = \{q_1, q_2, \ldots, q_i\}.$$

The result is trivial for $|x| = 0$, since $q_0' = [q_0]$. Suppose that it is true for $|x| \leqq l$. Then, for a in Σ,

$$\delta'(q_0', xa) = \delta'(\delta'(q_0', x), a).$$

By the inductive hypothesis,

$$\delta'(q_0', x) = [p_1, p_2, \ldots, p_j]$$

if and only if

$$\delta(q_0, x) = \{p_1, p_2, \ldots, p_j\}.$$

But by definition,

$$\delta'([p_1, p_2, \ldots, p_j], a) = [r_1, r_2, \ldots, r_k]$$

if and only if

$$\delta(\{p_1, p_2, \ldots, p_j\}, a) = \{r_1, r_2, \ldots, r_k\}.$$

Thus,

$$\delta'(q_0', xa) = [r_1, r_2, \ldots, r_k]$$

if and only if

$$\delta(q_0, xa) = \{r_1, r_2, \ldots, r_k\}.$$

To complete the proof, we have only to add that $\delta'(q_0', x)$ is in F' exactly when $\delta(q_0, x)$ contains a state of K which is in F. Thus $T(M) = T(M')$.

Since the deterministic and nondeterministic finite automata accept the same sets, we shall not distinguish between them unless it becomes necessary, but shall simply refer to both as finite automata.

Example 3.5. Let $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ be a nondeterministic fa, where:

$$\delta(q_0, 0) = \{q_0, q_1\} \qquad \delta(q_0, 1) = \{q_1\} \qquad \delta(q_1, 0) = \varphi \qquad \delta(q_1, 1) = \{q_0, q_1\}.$$

We can construct a deterministic fa, $M' = (K, \{0, 1\}, \delta', [q_0], F)$, accepting $T(M)$ as follows. K consists of all subsets of $\{q_0, q_1\}$. We denote the elements of K by $[q_0]$, $[q_1]$, $[q_0, q_1]$ and φ. Since $\delta(q_0, 0) = \{q_0, q_1\}$,

$$\delta'([q_0], 0) = [q_0, q_1].$$

Likewise,

$$\delta'([q_0], 1) = [q_1], \ \delta'([q_1], 0) = \varphi \qquad \text{and} \qquad \delta'([q_1], 1) = [q_0, q_1].$$

Naturally, $\delta'(\varphi, 0) = \delta'(\varphi, 1) = \varphi$. Lastly,

$$\delta'([q_0, q_1], 0) = [q_0, q_1],$$

since

$$\delta(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \varphi = \{q_0, q_1\};$$

and

$$\delta'([q_0, q_1], 1) = [q_0, q_1],$$

since

$$\delta(\{q_0, q_1\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_1\} \cup \{q_0, q_1\} = \{q_0, q_1\}.$$

The set F of final states is $\{[q_1], [q_0, q_1]\}$.

3.4 FINITE AUTOMATA AND TYPE 3 LANGUAGES

We now turn to the relationship between the languages generated by type 3 grammars and the sets accepted by finite automata.

Theorem 3.4. Let $G = (V_N, V_T, P, S)$ be a type 3 grammar. Then there exists a finite automaton $M = (K, V_T, \delta, S, F)$ with $T(M) = L(G)$.

Proof. M will be a nondeterministic fa. The states of M are the variables of G, plus an additional state A, not in V_N. Thus, $K = V_N \cup \{A\}$. The initial state of M is S. If P contains the production $S \to \epsilon$, then $F = \{S, A\}$. Otherwise, $F = \{A\}$. Recall that S will not appear on the right of any production if $S \to \epsilon$ is in P. The state A is in $\delta(B, a)$ if $B \to a$ is in P. In addition, $\delta(B, a)$ contains all C such that $B \to aC$ is in P. $\delta(A, a) = \varphi$ for each a in V_T.

The fa M, when accepting a sentence x, simulates a derivation of x by the grammar G. We shall show that $T(M) = L(G)$. Let $x = a_1a_2 \ldots a_n$ be in $L(G)$, $n \geqq 1$. Then

$$S \Rightarrow a_1A_1 \Rightarrow \cdots \Rightarrow a_1a_2 \ldots a_{n-1}A_{n-1} \Rightarrow a_1a_2 \ldots a_{n-1}a_n$$

for some sequence of variables $A_1, A_2, \ldots, A_{n-1}$. From the definition of δ, we can see that $\delta(S, a_1)$ contains A_1, that $\delta(A_1, a_2)$ contains A_2, etc., and that $\delta(A_{n-1}, a_n)$ contains A. Thus x is in $T(M)$, since $\delta(S, x)$ contains A, and A is in F. If ϵ is in $L(G)$, then S is in F, so ϵ is in $T(M)$.

Likewise, if x is in $T(M)$, $|x| \geqq 1$, then there exists a sequence of states $S, A_1, A_2, \ldots, A_{n-1}, A$ such that $\delta(S, a_1)$ contains A_1, $\delta(A_1, a_2)$ contains A_2, and so forth. Thus, P contains rules $S \to a_1A_1$, $A_1 \to a_2A_2, \ldots$ and $A_{n-1} \to a_n$. Therefore, $S \Rightarrow a_1A_1 \Rightarrow a_1a_2A_2 \Rightarrow \cdots \Rightarrow a_1a_2 \ldots a_{n-1}A_{n-1} \Rightarrow a_1a_2 \ldots a_n$ is a derivation in G and x is in $L(G)$. If ϵ is in $T(M)$, then S is in F, so $S \to \epsilon$ is a production in P, and ϵ is in $L(G)$.

Theorem 3.5. Given a finite automaton M, there exists a type 3 grammar G, such that $L(G) = T(M)$.

Proof. Without loss of generality let $M = (K, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton. Define a type 3 grammar $G = (K, \Sigma, P, q_0)$ as follows.

1. $B \to aC$ is in P if $\delta(B, a) = C$.
2. $B \to a$ is in P if $\delta(B, a) = C$ and C is in F.

The proof that $q_0 \overset{*}{\underset{G}{\Rightarrow}} w$ if and only if $\delta(q_0, w)$ is in F, for $|w| \geqq 1$, is similar to the proof of Theorem 3.4, and will be left to the reader. If q_0 is in F, then ϵ is in $T(M)$. In that case, $L(G) = T(M) - \{\epsilon\}$. By Theorem 2.1, we can obtain from G, a new type 3 grammar G_1, where

$$L(G_1) = L(G) \cup \{\epsilon\} = T(M).$$

If q_0 is not in F, then ϵ is not in $T(M)$, so $L(G) = T(M)$.

Example 3.6. Consider the following regular grammar, $G = (\{S, B\}, \{0, 1\}, P, S)$, where P consists of: $S \to 0B$, $B \to 0B$, $B \to 1S$, $B \to 0$.

We can construct a nondeterministic finite automaton $M = (\{S, B, A\}, \{0, 1\}, \delta, S, \{A\})$, where δ is given by:

1. $\delta(S, 0) = \{B\}$, since $S \to 0B$ is the only production in P with S on the left and 0 on the right.
2. $\delta(S, 1) = \varphi$, since no production has S on the left and 1 on the right.
3. $\delta(B, 0) = \{B, A\}$, since $B \to 0B$ and $B \to 0$ are in P.
4. $\delta(B, 1) = \{S\}$, since $B \to 1S$ is in P.
5. $\delta(A, 0) = \delta(A, 1) = \varphi$.

By Theorem 3.4, $T(M) = L(G)$, as one can easily verify.

We now use the construction of Theorem 3.3 to find a deterministic finite automaton M_1 equivalent to M. Then, we use the construction of Theorem 3.5 to find a grammar G_1, generating $L(G)$.

Let $M_1 = (K, \{0, 1\}, \delta', [S], F)$.

$$K = \{\varphi, [S], [A], [B], [A, S], [A, B], [B, S], [A, B, S]\}.$$

$$F = \{[A], [A, S], [A, B], [A, B, S]\}.$$

$$\begin{aligned} \delta'([S], 0) &= [B] & \qquad \delta'([S], 1) &= \varphi \\ \delta'([B], 0) &= [A, B] & \qquad \delta'([B], 1) &= [S] \\ \delta'([A, B], 0) &= [A, B] & \qquad \delta'([A, B], 1) &= [S] \\ \delta'(\varphi, 0) &= \delta'(\varphi, 1) = \varphi \end{aligned}$$

There are other rules of δ'. However, no states other than φ, $[S]$, $[B]$, and $[A, B]$ will ever be entered by M_1, and the other states can be removed from K and F.

Now, let us construct grammar $G_1 = (K, \{0, 1\}, P_1, [S])$ from M_1. From $\delta'([S], 0) = [B]$ we get the production $[S] \to 0[B]$. From $\delta'([B], 0) = [A, B]$, we get $[B] \to 0[A, B]$ and, since $[A, B]$ is a final state of M_1, we

place production $[B] \to 0$ in P_1, and so on. A complete list of the productions of P_1 is:

$$\begin{array}{lll} [S] \to 0[B] & [S] \to 1\varphi & \\ [B] \to 0[A, B] & [B] \to 1[S] & [B] \to 0 \\ [A, B] \to 0[A, B] & [A, B] \to 1[S] & [A, B] \to 0 \\ \varphi \to 0\varphi & \varphi \to 1\varphi & \end{array}$$

The grammar G_1 is much more complicated than is G, but $L(G_1) = L(G)$. The reader can simplify grammar G_1 so that its equivalence to G is readily observable.

3.5 PROPERTIES OF TYPE 3 LANGUAGES

Since the class of languages generated by type 3 grammars is equivalent to the class of sets accepted by finite automata, we shall use both formulations in establishing the properties of the class of type 3 languages. First we intend to show that the type 3 languages form a Boolean algebra† of sets.

Lemma 3.1. The class of type 3 languages is closed under union.

Proof. Two proofs are possible. One involves the use of nondeterministic finite automata. We leave this proof to the reader. A proof using grammars is also easy, and is given here.

Let L_1 and L_2 be type 3 languages generated by type 3 grammars

$$G_1 = (V_N^{(1)}, V_T^{(1)}, P_1, S_1) \quad \text{and} \quad G_2 = (V_N^{(2)}, V_T^{(2)}, P_2, S_2),$$

respectively. By renaming symbols, if necessary, we can assume that $V_N^{(1)}$ and $V_N^{(2)}$ contain no symbols in common, and that S is in neither. We construct a new grammar,

$$G_3 = (V_N^{(1)} \cup V_N^{(2)} \cup \{S\}, V_T^{(1)} \cup V_T^{(2)}, P_3, S),$$

where P_3 consists of the productions of P_1 and P_2 except for $S_1 \to \epsilon$ or $S_2 \to \epsilon$, plus all productions of the form $S \to \alpha$ such that either $S_1 \to \alpha$ is in P_1 or $S_2 \to \alpha$ is in P_2.

It should be obvious that $S \underset{G_3}{\Rightarrow} \alpha$ if and only if $S_1 \underset{G_1}{\Rightarrow} \alpha$ or $S_2 \underset{G_2}{\Rightarrow} \alpha$. In the first case, only strings in alphabet $V_N^{(1)} \cup V_T^{(1)}$ can be derived from α. In the second case, only strings in $V_N^{(2)} \cup V_T^{(2)}$ can be derived from α. Formally, if $S_1 \underset{G_1}{\Rightarrow} \alpha$, then $\alpha \overset{*}{\underset{G_3}{\Rightarrow}} w$ if and only if $\alpha \overset{*}{\underset{G_1}{\Rightarrow}} w$, and if $S_2 \underset{G_2}{\Rightarrow} \alpha$, then $\alpha \overset{*}{\underset{G_3}{\Rightarrow}} w$ if and only if $\alpha \overset{*}{\underset{G_2}{\Rightarrow}} w$. Putting the above together, $S \overset{*}{\underset{G_3}{\Rightarrow}} w$ if and only if either $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} w$ or $S_2 \overset{*}{\underset{G_2}{\Rightarrow}} w$. That is, $L(G_3) = L(G_1) \cup L(G_2)$.

---

† For our purposes a Boolean algebra of sets is a collection of sets closed under union, complement, and intersection. By the complement $\bar{L}$ of a language L, we mean $\Sigma^* - L$, for a finite set of symbols Σ, such that $L \subseteq \Sigma^*$.

Lemma 3.2. The class of sets accepted by finite automata (generated by type 3 grammars) is closed under complement.

Proof. Let $M_1 = (K, \Sigma_1, \delta_1, q_0, F)$ be a deterministic fa accepting a set S_1. Let Σ_2 be a finite alphabet containing Σ_1 and let d be a new state not in K. We construct M_2 to accept $\Sigma_2^* - S_1$. Let

$$M_2 = (K \cup \{d\}, \Sigma_2, \delta_2, q_0, (K - F) \cup \{d\}),$$

where $\delta_2(q, a) = \delta_1(q, a)$ for each q in K and a in Σ_1, $\delta_2(q, a) = d$ for each q in K and a in $\Sigma_2 - \Sigma_1$, and $\delta_2(d, a) = d$ for each a in Σ_2. Intuitively, M_2 is obtained by extending the input alphabet of M_1 to Σ_2, adding the "trap" state d and then interchanging final and nonfinal states. Clearly, M_2 accepts $\Sigma_2^* - S_1$.

Theorem 3.6. The class of sets accepted by finite automata forms a Boolean algebra.

Proof. Immediate from Lemmas 3.1 and 3.2 and the fact that

$$L_1 \cap L_2 = \overline{\overline{L}_1 \cup \overline{L}_2}.$$

We now give some additional theorems which will culminate in the characterization of the type 3 languages.

Theorem 3.7. All finite sets are accepted by finite automata.

Proof. Consider the set containing only the sentence $x = a_1 a_2 \ldots a_n$. We can design a finite automaton M with $n + 2$ states $q_0, q_1, q_2, \ldots, q_n$, and p. The initial state is q_0, and q_n is the only final state. As M sees successive symbols of x, it moves to successively higher-numbered states. If M sees a symbol which is not the next symbol of x, M goes to state p which is a "trap state" with no exit. Formally,

$$\delta(q_{i-1}, a_i) = q_i, \qquad 1 \leqq i \leqq n,$$

$$\delta(q_{i-1}, a) = p, \qquad 1 \leqq i \leqq n, \qquad \text{if } a \neq a_i$$

and

$$\delta(q_n, a) = \delta(p, a) = p \qquad \text{for all } a.$$

The reader should be able to supply the steps necessary to show that M accepts the sentence x. The set containing only the empty sentence is accepted by $M = (\{q_0, p\}, \Sigma, \delta, q_0, \{q_0\})$ where $\delta(q_0, a) = \delta(p, a) = p$ for each a in Σ. The empty set is accepted by $M = (\{q_0\}, \Sigma, \delta, q_0, \varphi)$ where $\delta(q_0, a) = q_0$ for each a in Σ.

The theorem follows immediately from the closure of type 3 languages under union.

We now define the *product*† UV of two languages U and V by

$$UV = \{x | x = uv, u \text{ is in } U \text{ and } v \text{ is in } V\}.$$

That is, each string in the set UV is formed by concatenating a string in U with a string in V. As an example, if $U = \{01, 11\}$ and $V = \{1, 0, 101\}$, then the set UV is $\{011, 010, 01101, 111, 110, 11101\}$.

Theorem 3.8. The class of sets accepted by finite automata (generated by type 3 grammars) is closed under product.

Proof. Let $M_1 = (K_1, \Sigma_1, \delta_1, q_1, F_1)$ and $M_2 = (K_2, \Sigma_2, \delta_2, q_2, F_2)$ be deterministic finite automata accepting languages L_1 and L_2, respectively. Assume that K_1 and K_2 are disjoint. Furthermore, without loss of generality, we can assume that $\Sigma_1 = \Sigma_2 = \Sigma$. (Otherwise, we can add "dead" states to K_1 and K_2 as in the proof of Lemma 3.2.) We construct a nondeterministic finite automaton M_3, accepting L_1L_2, which operates as follows. If the input string is x, M_3 behaves as M_1 until some initial portion (possibly ϵ) of x has been scanned. At this point, if M_1 would accept, M_3 guesses whether the end of the string from L_1 has been reached, or whether a longer initial portion is the string from L_1. In the former case, M_3 acts subsequently as M_2, and in the latter case, M_3 continues to behave as M_1.

Formally, let $M_3 = (K_1 \cup K_2, \Sigma, \delta_3, q_1, F)$. For each a in Σ let:

1. $\delta_3(q, a) = \{\delta_1(q, a)\}$ for each q in $K_1 - F_1$.
2. $\delta_3(q, a) = \{\delta_1(q, a), \delta_2(q_2, a)\}$ for each q in F_1.
3. $\delta_3(q, a) = \{\delta_2(q, a)\}$ for each q in K_2.

The purpose of Rule 1 is to allow M_3 to act like M_1 for some initial segment of the input (possibly ϵ). Rule 2 allows M_3 to continue the simulation of M_1 or to guess that a given symbol starts a word in L_2, provided that the previous symbol completed a word in L_1. Rule 3 allows only the simulation of M_2 after M_3 has guessed that the word from L_2 has been started.

If ϵ is not in L_2, then $F = F_2$. If ϵ is in L_2, then $F = F_1 \cup F_2$.

The *closure* of a language L, denoted by L^*, is the set consisting of the empty string and all finite-length strings formed by concatenating words in L. Thus, if $L = \{01, 11\}$, then $L^* = \{\epsilon, 01, 11, 0101, 0111, 1101, 1111, 010101, \ldots\}$. An alternative definition is $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$, where $L^0 = \{\epsilon\}$ and $L^i = L^{i-1}L$, for $i > 0$.

Theorem 3.9. The class of sets accepted by finite automata is closed under set closure.

---

† Also known as *concatenation* of sets.

Proof. Let $M = (K, \Sigma, \delta, q_0, F)$ be a finite automaton accepting L. We construct a nondeterministic finite automaton M', which behaves as M until an initial portion of a sentence x takes M to a final state. At this time, M' will guess whether or not this point corresponds to a point where a new string from L starts. Formally,

$$M' = (K \cup \{q_0'\}, \Sigma, \delta', \{q_0'\}, F \cup \{q_0'\}),$$

where q_0' is a new state, and

$$\begin{aligned} \delta'(q_0', a) &= \{\delta(q_0, a), q_0\}, && \text{if } \delta(q_0, a) \text{ is in } F, \\ &= \{\delta(q_0, a)\}, && \text{otherwise.} \\ \delta'(q, a) &= \{\delta(q, a), q_0\}, && \text{if } \delta(q, a) \text{ is in } F, \\ &= \{\delta(q, a)\}, && \text{otherwise, for all } q \text{ in } K. \end{aligned}$$

The purpose of the new initial state q_0' is to accept the empty string. If q_0 is not in F, we cannot simply make q_0 a final state since M may come back to q_0 for some input strings. Since the proof is somewhat more difficult than that of the previous theorems, we give a formal proof.

Assume that x is in L^*. Then either $x = \epsilon$, or $x = x_1x_2 \ldots x_n$, where x_i is in L for all i between 1 and n. Clearly M' accepts ϵ. Now x_i in L implies $\delta(q_0, x_i)$ is in F. Thus $\delta'(q_0', x_i)$ and $\delta'(q_0, x_i)$ each contain q_0 and some p (possibly $p = q_0$) in F. Hence, $\delta'(q_0', x)$ contains some state in F, and x is in $T(M')$.

Now assume that $x = a_1a_2 \ldots a_m$ is in $T(M')$. Then there exists some sequence of states $q_1, q_2, \ldots, q_m$ such that $\delta'(q_0', a_1)$ contains q_1, and $\delta'(q_i, a_{i+1})$ contains q_{i+1}, $1 \leqq i < m$, and q_m is in F. Thus, for each i, either $q_{i+1} = q_0$ and $\delta(q_i, a_{i+1})$ is in F or $\delta(q_i, a_{i+1}) = q_{i+1}$. Thus x can be written as $x_1x_2 \ldots x_n$, so that $\delta(q_0, x_i)$ is in F for $1 \leqq i \leqq n$, implying that x_i is in L.

Theorem 3.10. The class of sets accepted by finite automata is the smallest class containing all finite sets and closed under union, product, and closure.

Proof. That the class of sets accepted by finite automata contains the smallest class containing all finite sets and closed under union, product, and closure, is an immediate consequence of Lemma 3.1 and Theorems 3.7, 3.8, and 3.9. It remains to show that the smallest class containing all finite sets and closed under union, product, and closure contains the class of sets accepted by finite automata.

Let L_1 be a set accepted by some finite automaton,

$$M = (\{q_1, \ldots, q_n\}, \Sigma, \delta, q_1, F).$$

Let R_{ij}^k denote the set of all strings x such that $\delta(q_i, x) = q_j$, and if $\delta(q_i, y) = q_l$, for any y which is an initial segment of x other than x or ϵ, then $l \leqq k$.

That is, R_{ij}^k is the set of all strings which take the finite automaton from state q_i to state q_j without going through any state q_l, $l > k$. Note that by "going through a state," we mean both entering and leaving. Thus i or j may be greater than k. We can define R_{ij}^k recursively:

$$R_{ij}^k = R_{ik}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1} \cup R_{ij}^{k-1}$$
$$R_{ij}^0 = \{a \mid \delta(q_i, a) = q_j\}.$$

Informally, the definition of R_{ij}^k above means that the inputs that cause M to go from q_i to q_j without passing through a state higher than q_k are either:

1. in R_{ij}^{k-1}, that is, they never reach a state as high as q_k.
2. composed of a string in R_{ik}^{k-1} (which takes M to q_k for the first time) followed by some number of strings in R_{kk}^{k-1} (which take M from q_k back to q_k without passing through q_k otherwise) followed by a string in R_{kj}^{k-1} (which takes M from state q_k to q_j).

We can show, by induction on k, that R_{ij}^k, $0 \leqq k \leqq l$, is, for all i and j, within the smallest class containing all finite sets and closed under union, complex product, and closure. The induction hypothesis is true for $l = 0$, since all R_{ij}^0 are finite sets. If true for all $k \leqq l$, then it is true for $k = l + 1$ since we can express R_{ij}^{l+1} in terms of union, concatenation, and closure of various sets of the form R_{mn}^l, each of which is presumed to be in the smallest class of sets containing the finite sets and closed under union, concatenation, and closure. Now $L_1 = \bigcup_{q_j \text{ in } F} R_{1j}^n$. Thus L_1 is in the smallest class of sets containing the finite sets, and closed under union, concatenation, and closure.

As a result of Theorem 3.10, we know that any expression made up of finite subsets of Σ^* for some finite alphabet Σ, and a finite number of the operators $\cup$, $\cdot$† and $*$, with parentheses to determine the order of operations, denotes a set that is accepted by a finite automaton. Furthermore, every set accepted by some fa can be so expressed. This provides us with a good notation for describing regular sets. For example, $\{0, 1\}^*\{000\}\{0, 1\}^*$ denotes the set of all strings with three consecutive 0's, and $(\{0, 1\}\{0, 1\})^* \cup (\{0, 1\}\{0,1\}\{0, 1\})^*$ denotes the set of all strings whose length is divisible by two or three.

3.6 SOLVABLE PROBLEMS CONCERNING FINITE AUTOMATA

In this section we show that there are algorithms to answer many questions concerning finite automata and type 3 languages. In Chapter 14 we shall see that no such algorithms can possibly exist to answer some of these questions for the other types of languages discussed in Chapter 2.

---

† $S_1 \cdot S_2$ is the product $S_1 S_2$.

Theorem 3.11. The set of sentences accepted by a finite automaton with n states is:

1. nonempty if and only if the finite automaton accepts a sentence of length less than n.
2. infinite if and only if the automaton accepts a sentence of length l, $n \leqq l < 2n$.

Thus, there is an algorithm to determine if a finite automaton accepts zero, a finite number, or an infinite number of sentences.

Proof. (1) The "if" portion is obvious. Suppose that a finite automaton $M = (K, \Sigma, \delta, q_0, F)$, with n states accepts some word. Let w be a word as short as any other word accepted. We might as well assume that $|w| \geqq n$, else the result is proven for M. Since there are but n states, M must pass through the same state twice in accepting w. Formally, we can find q in K such that we can write $w = w_1w_2w_3$, with $w_2 \neq \epsilon$, $\delta(q_0, w_1) = q$, $\delta(q, w_2) = q$, and $\delta(q, w_3)$ in F. Then w_1w_3 is in $T(M)$, since

$$\delta(q_0, w_1w_3) = \delta(q_0, w_1w_2w_3).$$

But $|w_1w_3| < |w|$, contradicting the assumption that w is as short as any word in $T(M)$.

(2) We leave most of this part to the reader. We merely observe that if w is in $T(M)$ and $n \leqq |w| < 2n$, then we can write $w = w_1w_2w_3$, $w_2 \neq \epsilon$, and for all i, $w_1w_2^iw_3$ is in $T(M)$. Next, if M accepts an infinity of words, and none is of length between n and $2n - 1$, then let w be of length at least $2n$, but as short as any word in $T(M)$ whose length is $\geqq 2n$. Then, we can write $w = w_1w_2w_3$, with $1 \leqq |w_2| \leqq n$, and w_1w_3 in $T(M)$, thus deriving a contradiction.

In part (1), the algorithm to decide if $T(M)$ is empty is: "See if any word of length up to n is in $T(M)$." Clearly there is such a procedure which is guaranteed to halt. In part (2), the algorithm to decide if $T(M)$ is infinite is: "See if any word of length between n and $2n - 1$ is in $T(M)$." Again, clearly there is such a procedure which is guaranteed to halt.

We now show that there is an algorithm to determine if two type 3 grammars generate the same language. As we shall see later, no such algorithm exists for type 0, 1, or 2 grammars.

Theorem 3.12. There is an algorithm to determine if two finite automata are equivalent (i.e., if they accept the same language).

Proof. Let M_1 and M_2 be fa, accepting L_1 and L_2, respectively. By Theorem 3.6, $(L_1 \cap \bar{L}_2) \cup (\bar{L}_1 \cap L_2)$ is accepted by some finite automaton, M_3. It is easy to see that M_3 accepts a word if and only if $L_1 \neq L_2$. Hence, by Theorem 3.11, there is an algorithm to determine if $L_1 = L_2$.

3.7 TWO-WAY FINITE AUTOMATA

We now turn our attention to finite automata that can move two ways on their input tapes. Our reason for studying these is twofold. First it is easier to introduce the concept of moving two ways on the input with finite automata than with more complicated automata. Second, we wish to introduce the concept of a finite table being stored in a finite control, a tool we shall find useful later on.

A two-way finite automaton M will be represented by a 5-tuple $(K, \Sigma, \delta, q_0, F)$, where K is a set of states, Σ is the set of input symbols, δ is a mapping from $K \times \Sigma$ to $K \times \{L, R, S\}$, q_0 in K is the initial state, and $F \subseteq K$ is the set of final states. The interpretation of $\delta(q, a) = (p, D)$, p in K and D in $\{L, R, S\}$ is that M, in state q, scanning the input symbol a, will move its input head one cell to the left, to the right, or not move its input head at all, depending on whether D equals L, R, or S, respectively. The state of M will also be changed to state p. Note that δ cannot be extended from $K \times \Sigma$ to $K \times \Sigma^*$ in the obvious manner, since one must keep track of the net change in input position.

We define a *configuration* of M to be a state and a number (q, i), where q is the present state of M and i is the location of the input head. That is, i is the number of cells the input head is from the left end of the input.

A two-way finite automaton will start in state q_0 with its input head scanning the leftmost cell on the input tape. Should M ever move off either end of x, M halts. An input word will be *accepted* by M if and only if M eventually moves off the right end of x at the same time it enters a final state.

M can reject a word x by:

1. moving off the left end of x.
2. moving off the right end of x in a nonfinal state.
3. looping.

Theorem 3.13. The class of sets accepted by two-way finite automata is the same as the class of sets accepted by one-way finite automata.

Proof. Let $M = (K, \Sigma, \delta, q_0, F)$ be a two-way finite automaton and $x = a_1 a_2 \ldots a_n$ be the input to M. We can associate with each initial segment of x, $a_1 a_2 \ldots a_i$, a mapping Δ_i of K to $K \cup \{R\}$. We say that Δ_i is *associated* with $a_1 a_2 \ldots a_i$. The interpretation of $\Delta_i(q) = p$, p in K, is that if M is started in state q, scanning the ith cell, M will eventually move right to the $i + 1$st cell. On the move which takes M to the $i + 1$st cell for the first time, M enters state p. Note that before reaching the $i + 1$st cell M may move its input head back and forth on cells 1 through i many times.

The interpretation of $\Delta_i(q) = R$ is that if M is started in state q scanning the ith cell, M will reject x without ever having moved right from the ith cell to the $i + 1$st cell. That is, M will either enter a loop or move off the input tape to the left.

The number of distinct mappings of K to $K \cup \{R\}$ is $(s + 1)^s$, where s is the number of elements in K. We can define a one-way finite automaton $M' = (K', \Sigma, \delta', q_0', F')$, whose states except one are ordered pairs $[q, \Delta]$ where q is in K and Δ is a mapping of K to $K \cup \{R\}$.† Also in K' is a "trap state" t, having the property that $\delta'(t, a) = t$ for all a in Σ. The interpretation of the state $[q, \Delta]$ is that the one-way fa M' will be in state $[q, \Delta]$ after reading an input x if and only if the two-way finite automaton M would be in state q the first time M moves right from string x and the mapping Δ is the mapping associated with the string x. Thus M' carries in its finite control a table which contains the information as to the eventual outcome if M moves left into x in any state.

We define the mapping δ' as follows.

$$\delta'([q_1, \Delta_1], a) = [q_2, \Delta_2]$$

exactly when we can compute q_2 and Δ_2 from q_1 and Δ_1 by:

1. $\Delta_2(p_1) = p$ if there exists a sequence of states $p_2, p_3, \ldots, p_n, p$, such that either
$$\delta(p_i, a) = (p_{i+1}, S)$$
or
$$\delta(p_i, a) = (p_i', L)$$
and
$$\Delta_1(p_i') = p_{i+1}$$
for all i, where $1 \leqq i < n$. Finally, $\delta(p_n, a) = (p, R)$.
2. $\Delta_2(p_1) = R$ if there is no finite sequence satisfying (1). Note that if any sequence $p_1, p_2, \ldots, p_n, p$ satisfies (1), then there is a sequence satisfying (1) such that no state appears twice in the sequence.
3. $q_2 = \Delta_2(q_1)$. If $\Delta_2(q_1) = R$, however, there is no such q_2, therefore $\delta'([q_1, \Delta_1], a)$ is t, the trap state.

Informally, to compute $\Delta_2(p_1)$, assume that Δ_1 is associated with the first $j - 1$ symbols. We begin constructing a sequence of states $p_1, p_2, \ldots$ by adding to the sequence, each time M scans the jth cell, the state of M at that time. Hopefully, from one of these states, M will move to the right and enter some state p. Then we can end the sequence, since we have $\Delta_2(p_1) = p$. Suppose that we have constructed the sequence $p_1, p_2, \ldots, p_i$. Then the fa M will be in configuration (p_i, j), scanning an a on its input. Three cases arise:

1. $\delta(p_i, a) = (q, S)$. Here M will again scan cell j, this time in state q. So $p_{i+1} = q$.

† Do not confuse R in the range of Δ, which means reject, with R in the range of δ which means move right.

2. $\delta(p_i\, a) = (q, R)$. Here, we have our answer. M has moved to cell $j + 1$ for the first time and entered state q. Thus, q has the role of p in the sequence, i.e., $\Delta_2(p_1) = q$.
3. $\delta(p_i, a) = (q, L)$. Here, M next moves left to cell $j - 1$. We consult Δ_1 to see what happens next. If $\Delta_1(q) = R$, then M will never again scan cell j, so it cannot scan $j + 1$ either. We therefore let $\Delta_2(p_1) = R$. If $\Delta_1(q) = q'$, then $p_{i+1} = q'$.

The above process will produce a value for $\Delta_2(p_1)$ in all cases, provided that the sequence $p_1, p_2, \ldots$ never repeats a state. If a state is repeated, M is in a loop and will never reach cell $j + 1$. In this case, $\Delta_2(p_1) = R$. We have now covered all contingencies. We leave it to the reader to see that if Δ_1 is the table associated with input x, then Δ_2 will be the table associated with xa.

If M' is to be a one-way finite automaton simulating M, we must let its initial state, q_0', be $[q_0, \Delta_0]$, where Δ_0 is the table associated with ϵ, i.e., $\Delta_0(q) = R$ for all q. Also,

$$F' = \{[q, \Delta] | q \text{ is in } F\}.$$

We must now prove by induction on the length of input x that M' moves right in state $[q, \Delta]$ (for some Δ) from x if and only if M moves right from x in q. Thus M' accepts x if and only if M does. If x is of length 1, the result follows from the way $\delta'([q_0, \Delta_0], x)$ is constructed. That is, if and only if $\delta'([q_0, \Delta_0], x) = [q, \Delta]$, will M eventually move to the right from its first input symbol and enter state q at that time. Note $[q, \Delta]$ is accepting if and only if q is accepting for M.

Suppose that the result is true for $|x| < k$, and let xa be an input of length k. Then M will scan the final symbol, a, of xa for the first time in state q if and only if $\delta'([q_0, \Delta_0], x) = [q, \Delta_1]$ for some Δ_1. But then M will move to the right from xa and enter state p if and only if $\delta'([q, \Delta_1], a) = [p, \Delta_2]$, for the proper Δ_2. This follows from the construction of Δ_2 from Δ_1 just given. The acceptance of xa by M occurs if and only if p is in F. But then $[p, \Delta_2]$ is in F', so M' accepts xa.

Example 3.7. Consider the two-way finite automaton.

$$M = (\{q_0, q_1\}, \{a, b\}, \delta, q_0, \{q_1\})$$

where δ is defined by:

$$\delta(q_0, a) = (q_0, R) \quad \delta(q_0, b) = (q_1, S) \quad \delta(q_1, a) = (q_0, L) \quad \delta(q_1, b) = (q_0, L).$$

We can construct a one-way finite automaton $M' = (K, \{a, b\}, \delta', q_0', F)$ equivalent to M. Here K is the set of objects of the form $[q, \Delta]$, where $q = q_0$ or q_1 and Δ is a map from $\{q_0, q_1\}$ to $\{q_0, q_1, R\}$, plus a trap state. There are nine possible values of Δ. The set of all $[q_1, \Delta]$ is F.

$$q_0' = [q_0, \Delta_0], \qquad \text{where } \Delta_0(q_0) = \Delta_0(q_1) = R.$$

Since M' has 19 states, we shall not construct δ' for all of these, but simply give one case. We compute

$$\delta'([q_0, \Delta_1], a) \quad \text{and} \quad \delta'([q_0, \Delta_1], b),$$

where $\Delta_1(q_0) = q_0$ and $\Delta_1(q_1) = q_1$. Let

$$\delta'([q_0, \Delta_1], a) = [q_a, \Delta_a] \quad \text{and} \quad \delta'([q_0, \Delta_1], b) = [q_b, \Delta_b].$$

To compute $\Delta_a(q_0)$, we note that $\delta(q_0, a) = (q_0, R)$. Thus M immediately moves to the right from a and enters state q_0. We have $\Delta_a(q_0) = q_0$. For $\Delta_a(q_1)$, we note that $\delta(q_1, a) = (q_0, L)$. Thus, we must consult $\Delta_1(q_0)$ to see if M will ever return to the symbol a. We have $\Delta_1(q_0) = q_0$, so M does return in state q_0. From a, in state q_0, M next moves to the right, remaining in q_0, since $\delta(q_0, a) = (q_0, R)$. Hence

$$\Delta_a(q_1) = q_0 \quad \text{and} \quad q_a = \Delta_a(q_0) = q_0.$$

To compute $\Delta_b(q_0)$, note that $\delta(q_0, b) = (q_1, S)$, so M would remain scanning the b, but in state q_1. Next, $\delta(q_1, b) = (q_0, L)$. Now, $\Delta_1(q_0) = q_0$, so M would return to a in state q_0, where it started. M is in a loop, so $\Delta_b(q_0) = R$. Likewise, $\Delta_b(q_1) = R$. But $\Delta_b(q_0) = R$, so $\delta'([q_0, \Delta_1], b)$ is the trap state.

PROBLEMS

3.1 Find a one-way, deterministic finite automaton accepting all strings in $\{0, 1\}^*$ such that every 0 has a 1 immediately to its right.

3.2 From the finite automaton of Problem 3.1, construct a type 3 grammar generating the language of that problem.

3.3 Give an example of a relation which is:

a) reflexive and symmetric, but not transitive.
b) symmetric and transitive, but not reflexive.
c) reflexive and transitive, but not symmetric.

3.4 Let

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$$

be a nondeterministic finite automaton, with

$$\delta(q_0, a) = \{q_1, q_2\} \qquad \delta(q_1, a) = \{q_0, q_1\} \qquad \delta(q_2, a) = \{q_0, q_2\}$$
$$\delta(q_0, b) = \{q_0\} \qquad \delta(q_1, b) = \varphi \qquad \delta(q_2, b) = \{q_1\}.$$

Find a deterministic finite automaton accepting $T(M)$.

3.5 Complete the specification of the one-way finite automaton in Example 3.7.

3.6 Use the notion of a nondeterministic finite automaton to show that if L is a type 3 language, then

$$L^R = \{w | w \text{ reversed is in } L\}$$

is a type 3 language.

3.7 From Problem 3.6, show that grammars where all productions are of the form $A \to Bb$ or $A \to b$, A and B variables, b terminal, generate all and only the type 3 languages.

3.8 A nondeterministic two-way finite automaton is denoted by $M = (K, \Sigma, \delta, q_0, F)$ as the two-way deterministic finite automaton, except that $\delta(q, a)$, for q in K and a in Σ, is a subset of $K \times \{L, R, S\}$. Each (p, D) in $\delta(q, a)$ represents a possible move of M when the automaton is in state q scanning a on its input. If any sequence of choices of moves causes M to move off the right end of its input in an accepting state, M accepts. Show that only type 3 languages are accepted by nondeterministic, two-way finite automata.

3.9 Let L be a type 3 language. Let Init (L) be the set of words x, such that for some word y, xy is in L. Show that Init (L) is a type 3 language.

3.10 Let R be a type 3 language consisting only of words whose length is divisible by 3. Consider the language formed by taking the first third of each sentence in R. Is this language a type 3 language? What about the last third? Middle third? What about the language formed by concatenating the first and last third of each word in R?

3.11 Some people allow a two-way finite automaton to have an end marker, $\not\in$, at the left end of each tape. This finite automaton is said to accept w if it moves off the right end of $\not\in w$ while entering a final state. Show that under this definition, it is still only regular sets which are accepted.

REFERENCES

The notion of a finite state device is usually attributed to McCulloch and Pitts [1943]. The formalism we have used was suggested in Moore [1956] and is found in Rabin and Scott [1959].

Theorem 3.1 is from Nerode [1958] and was proven in a slightly weaker form by Myhill. The minimization of finite automata (Theorem 3.2) appeared originally in Huffman [1954] and Moore [1956]. Nondeterministic finite automata and Theorem 3.3 are from Rabin and Scott [1959]; Theorem 3.13 and two-way finite automata are found in Rabin and Scott [1959] and Shepherdson [1959]. Theorem 3.10 is from Kleene [1956]. The description of regular languages that follows Theorem 3.10 is known as a "regular expression," and is from Kleene [1956]. Many results concerning regular expressions can be found in Brzozowski [1962] and McNaughton and Yamada [1960]. The algorithms in Section 3.6 are from Moore [1956], and the results of Section 3.4 relating type 3 grammars and finite automata are from Chomsky and Miller [1958].

Many books have been written on the subject of finite automata. Among them are Gill [1962] and Ginsburg [1962]. Finite automata are also covered extensively by Harrison [1965], Booth [1967], and Minsky [1967].

CHAPTER 4

CONTEXT-FREE GRAMMARS

4.1 SIMPLIFICATION OF CONTEXT-FREE GRAMMARS

In this chapter we describe some of the basic simplifications of context-free grammars and prove several important normal-form theorems. One of these will be the *Chomsky Normal-Form Theorem*, which states that every context-free language is generated by a grammar for which all productions are of the form $A \to BC$ or $A \to b$.† Here A, B, and C are variables, and b is a terminal.

Another is the *Greibach Normal-Form Theorem*, which states that every context-free language is generated by a grammar for which all productions are of the form $A \to b\alpha$, where b is a terminal and α is a string of variables.

We also show that there exist algorithms to determine whether the language generated by a context-free grammar is empty, finite, or infinite. We define a property of certain context-free grammars, called the self-embedding property, and show that a context-free language is nonregular if and only if every type 2 grammar generating the language has the self-embedding property. Finally we consider certain special types of restricted context-free grammars such as sequential grammars and linear grammars.

The formal definition of a context-free grammar allows for certain structures which are in a sense "wasteful." For example, the vocabulary could include variables that can never be used in the derivation of a terminal string, or there might be a production of the form $A \to A$ for some variable, A. Thus we prove several theorems to show that every context-free language can be generated by a context-free grammar of a specified form. Furthermore, we show that algorithms exist which, for any context-free grammar, will find an equivalent context-free grammar in one of the specified forms. First, we prove a result which is quite important in its own right.

Theorem 4.1. There is an algorithm for determining if the language generated by a given context-free grammar is empty.

---

† Until Section 4.6, we shall revert to the original definition of a cfg and not allow ϵ to be in any cfl. The reader can easily supply the appropriate modification to include the case where $S \to \epsilon$ can be a production.

Proof. Let $G = (V_N, V_T, P, S)$ be a context-free grammar. Suppose that $S \overset{*}{\Rightarrow} w$ for some terminal string w. Consider a derivation tree of w in the grammar G. Suppose that there is a path in the tree with two nodes, n_1 and n_2, having the same label A, with n_1 higher on the path than n_2.† Here we can refer to Fig. 4.1. The subtree with root at n_1 represents the generation of a word w_1, such that $A \overset{*}{\Rightarrow} w_1$. The subtree with root at n_2 likewise represents the generation of a word w_2, such that $A \overset{*}{\Rightarrow} w_2$. (Note that w_2 must be a subword of w_1, perhaps all of w_1.)

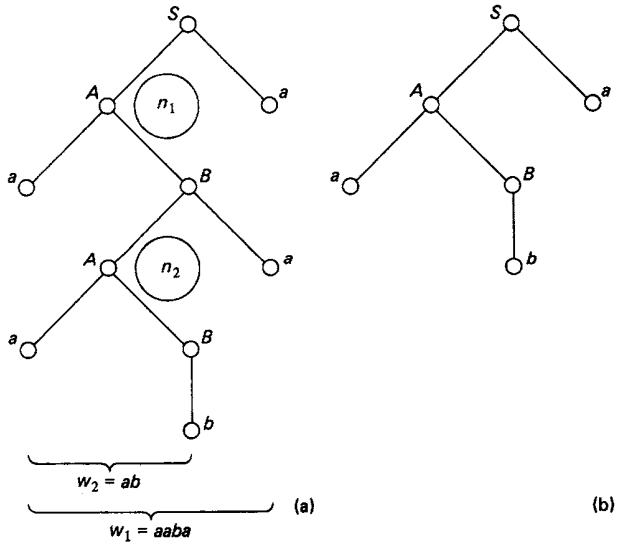


Fig. 4.1. Obtaining the tree for the derivation of $w_3w_2w_4 = aba$ from the tree for the derivation of $w = aabaa$.

Now the word w can be written in the form $w_3w_1w_4$ where w_3 or w_4, or both, may be ϵ. If we replace the subtree of n_1 by that of n_2, we have a new word, $w_3w_2w_4$ (possibly the same word), such that $S \overset{*}{\Rightarrow} w_3w_2w_4$. In Fig. 4.1, $w_3 = \epsilon$ and $w_4 = a$. A tree for $S \overset{*}{\Rightarrow} w_3w_2w_4$ is shown in Fig. 4.1(b). However, we have eliminated at least one node, n_1, from the tree. If the new tree has a path with two identically labeled nodes, the process may be repeated with $w_3w_2w_4$ instead of w. In fact, the process may be repeated until

† A *path* is a connected sequence of directed edges. The *length* of a path is the number of edges in the path.

there are no paths in the tree with two nodes labeled identically. Since each iteration eliminates one or more nodes, the process must eventually terminate.

Now consider the tree which is ultimately produced. If there are m variables in the grammar G, then there can be no path of length greater than m, lest some variable would be repeated in this path. We conclude that if G generates any word at all, then there is a derivation of a word whose tree contains no path of length greater than m. Thus the following algorithm will determine if $L(G)$ is empty.

Form a collection of trees corresponding to derivations in G as follows. Start the collection with the tree containing the single node labeled S. Repeatedly add to the collection any tree that can be obtained from a tree already in the collection by application of a single production and that:

1. is not already in the collection, and
2. does not have any path of length greater than m.

Since there are a finite number of trees corresponding to derivations with no path length greater than m, the process must eventually terminate. Now $L(G)$ is nonempty if and only if at least one of the trees in the collection corresponds to the derivation of a terminal string.

The existence of an algorithm to determine whether a given cfl is empty is very important. We shall use this fact extensively in simplifying context-free grammars. As we shall see later, for more complex types of grammars, such as the context-sensitive grammars, no such algorithm exists.

Theorem 4.2. Given any context-free grammar $G = (V_N, V_T, P, S)$, generating a nonempty language, it is possible to find an equivalent grammar G_1, such that for every variable A of G_1 there is a terminal string w, such that $A \overset{*}{\Rightarrow} w$.

Proof. For each variable A in V_N, consider the grammar $G_A = (V_N, V_T, P, A)$. If the language $L(G_A)$ is empty, then we can remove A from V_N, and we can remove all productions involving A, either on the right or left, from P.

After deleting from G all occurrences of variables A such that $L(G_A)$ is empty, we have a new grammar $G_1 = (V_N', V_T, P', S)$ where V_N' and P' are the remaining variables and productions. Clearly $L(G_1) \subseteq L(G)$, since a derivation in G_1 is a derivation in G. Suppose that there is a word w in $L(G)$ which is not in $L(G_1)$. Then some derivation of w must involve a sentential form $\alpha_1 A \alpha_2$, where A is in $V_N - V_N'$ and $S \overset{*}{\underset{G}{\Rightarrow}} \alpha_1 A \alpha_2 \overset{*}{\underset{G}{\Rightarrow}} w$. Then, however, there must be some w_1 in V_T^* such that $A \overset{*}{\underset{G}{\Rightarrow}} w_1$, a fact that contradicts the requirement that A be in $V_N - V_N'$.

In addition to removing variables from which no terminal string can be derived, we can also remove variables which are useless in the sense that they can never appear in a derivation.

Theorem 4.3. Given any context-free grammar generating a nonempty cfl L, it is possible to find a grammar G, generating L, such that for each variable A there is a derivation

$$S \overset{*}{\Rightarrow} w_1 A w_3 \overset{*}{\Rightarrow} w_1 w_2 w_3,$$

where w_1, w_2, and w_3 are in V_T^*.

Proof. Let $G_1 = (V_N, V_T, P, S)$ be any grammar generating L that satisfies Theorem 4.2. If $S \overset{*}{\Rightarrow} \alpha_1 A \alpha_2$, α_1 and α_2 in V^*, then there exists a derivation $S \overset{*}{\Rightarrow} w_1 A w_2 \overset{*}{\Rightarrow} w_1 w_2 w_3$, since terminal strings can be derived from A and from all variables appearing in α_1 and α_2. We can effectively construct the set V_N' of all variables A, such that $S \overset{*}{\Rightarrow} \alpha_1 A \alpha_2$, as follows. Start by placing S in the set. Add to the set any variable which appears on the right-hand side of any production $A \to \alpha$, if A is in the set. The procedure stops when no new members can be added to the set.

Let $G_2 = (V_N', V_T, P', S)$ where P' is the set of productions remaining after removing all productions from P which have variables in $V_N - V_N'$ on either the left or right. Now $L(G) = L(G')$, as one can easily show, and G_2 satisfies the condition of the theorem.

Before the next theorem, let us introduce the concept of a *leftmost derivation.* We say that a derivation is leftmost if, at every step, the variable replaced has no variable to its left in the sentential form from which the replacement is made. That is, if $S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$ is a leftmost derivation in some grammar $G = (V_N, V_T, P, S)$, then for $1 \leqq i < n$, we can write α_i as $x_i A_i \beta_i$, where the string x_i is in the set V_T^*, β_i is an arbitrary string in V^*, A_i is a variable and $A_i \to \gamma_i$ is a production of P. Finally, $x_i \gamma_i \beta_i$ is α_{i+1}, and α_{i+1} was derived from α_i by replacing A_i by γ_i.

Lemma 4.1. Given a context-free grammar $G = (V_N, V_T, P, S)$, if $S \overset{*}{\underset{G}{\Rightarrow}} w$, then there is a leftmost derivation of w in G.

Proof. We prove by induction on the number of steps in the derivation that if A is any variable and $A \overset{*}{\underset{G}{\Rightarrow}} w$, w in V_T^*, then $A \overset{*}{\underset{G}{\Rightarrow}} w$ by a leftmost derivation. The statement is trivially true for one-step derivations. Suppose that it is true for derivations of k or fewer steps. Let $A \Rightarrow \alpha_1 \overset{*}{\Rightarrow} w$ be a $k+1$ step derivation in G and suppose that $\alpha_1 = B_1 B_2 \ldots B_m$, where B_i is in V, $1 \leqq i \leqq m$. The first step of the derivation is clearly $A \Rightarrow B_1 B_2 \ldots B_m$. We can write w as $u_1 u_2 \ldots u_m$, where $B_i \overset{*}{\Rightarrow} u_i$, for $1 \leqq i \leqq m$. By the inductive hypothesis, there exist leftmost derivations of u_i from B_i, $1 \leqq i \leqq m$. Note

that B_i may be a terminal, in which case $B_i = u_i$ and the derivation takes no steps. Thus the first step of the derivation of w is followed by a leftmost derivation of u_1 from B_1, yielding a leftmost derivation of $u_1B_2B_3 \ldots B_m$ from A. Now u_1 is in V_T^*, so a leftmost derivation of u_2 from B_2 will not violate the definition of a leftmost derivation of w from A. In turn, we replace each B_i which is not a terminal by u_i according to a leftmost derivation. It is easy to see that the definition of a leftmost derivation for $A \overset{*}{\Rightarrow} w$ is never violated.

Theorem 4.4. Given a context-free grammar G, we can find an equivalent grammar G_1 with no productions of the form $A \to B$, where A and B are variables.

Proof. Let G be the grammar (V_N, V_T, P, S). Call the productions in P of the form $A \to B$, A and B in V_N, "type x" productions and all other productions "type y."

We construct a new set of productions P_1 from P by first including all type y productions of P. Then, suppose that $A \overset{*}{\underset{G}{\Rightarrow}} B$, for A and B in V_N. We add to P_1 all productions of the form $A \to \alpha$, where $B \to \alpha$ is a type y production of P.

Observe that we can easily test if $A \overset{*}{\underset{G}{\Rightarrow}} B$, since if

$$A \underset{G}{\Rightarrow} B_1 \underset{G}{\Rightarrow} B_2 \underset{G}{\Rightarrow} \cdots \Rightarrow B_m \underset{G}{\Rightarrow} B,$$

and some variable appears twice in the sequence, we can find a shorter sequence of type x productions which will result in $A \overset{*}{\underset{G}{\Rightarrow}} B$. Thus it is sufficient to consider only those sequences of type x productions whose length is less than the number of variables of G.

We now have a modified grammar, $G_1 = (V_N, V_T, P_1, S)$. Surely, if $A \to \alpha$ is a production of P_1, then $A \overset{*}{\underset{G}{\Rightarrow}} \alpha$. Thus if there is a derivation of w in G_1, then there is a derivation of w in G.

Now suppose that w is in $L(G)$ and consider a leftmost derivation of w in G, say $S = \alpha_0 \underset{G}{\Rightarrow} \alpha_1 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} \alpha_n = w$. If, for $0 \leqq i < n$, $\alpha_i \underset{G}{\Rightarrow} \alpha_{i+1}$ by a type y production, then $\alpha_i \underset{G_1}{\Rightarrow} \alpha_{i+1}$. Suppose that $\alpha_i \underset{G}{\Rightarrow} \alpha_{i+1}$ by a type x production, but that $\alpha_{i-1} \underset{G}{\Rightarrow} \alpha_i$ by a type y production unless $i = 0$. Also, suppose that $\alpha_{i+1} \underset{G}{\Rightarrow} \alpha_{i+2} \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} \alpha_j$ all by type x productions and $\alpha_j \underset{G}{\Rightarrow} \alpha_{j+1}$ by a type y production. Then $\alpha_i, \alpha_{i+1}, \ldots, \alpha_j$ are all of the same length, and since the derivation is leftmost, the symbol replaced in each of these must be at the same position. But then $\alpha_i \underset{G_1}{\Rightarrow} \alpha_{j+1}$ by one of the productions of $P_1 - P$. Hence $L(G_1) = L(G)$.

4.2 CHOMSKY NORMAL FORM

We now prove the first of two normal-form theorems. These each state that all context-free grammars are equivalent to grammars with restrictions on the forms of productions.

Theorem 4.5. (*Chomsky Normal Form.*) Any context-free language can be generated by a grammar in which all productions are of the form $A \to BC$ or $A \to a$. Here A, B, and C are variables and a is a terminal.

Proof. Let G be a context-free grammar. By Theorem 4.4, we can find an equivalent grammar, $G_1 = (V_N, V_T, P, S)$, such that P contains no productions of the form $A \to B$, where A and B are variables. Thus, if a production has a single symbol on the right, that symbol is a terminal, and the production is already in an acceptable form.

Now consider a production in P, of the form $A \to B_1B_2 \ldots B_m$, where $m \geqq 2$. Each terminal B_i is replaced by a new variable C_i, which appears on the right of no other production. We then create a new production $C_i \to B_i$ which is of allowable form, since B_i is a terminal. The production $A \to B_1B_2 \ldots B_m$ is replaced by $A \to C_1C_2 \ldots C_m$ where $C_i = B_i$ if B_i is a variable.

Let the new set of variables be V_N', and the new set of productions, P'. Consider the grammar $G_2 = (V_N', V_T, P', S)$.† If $\alpha \underset{G_1}{\Longrightarrow} \beta$, then $\alpha \overset{*}{\underset{G_2}{\Longrightarrow}} \beta$. Thus $L(G_1) \subseteq L(G_2)$. Now we show by induction on the number of steps in a derivation that if $A \overset{*}{\underset{G_2}{\Longrightarrow}} w$, A in V_N and w in V_T^*, then $A \overset{*}{\underset{G_1}{\Longrightarrow}} w$. The result is trivial for one-step derivations. Suppose that it is true for derivations of up to k steps. Let $A \overset{*}{\underset{G_2}{\Longrightarrow}} w$ by a $k + 1$ step derivation. The first step must be of the form

$$A \Longrightarrow C_1C_2 \ldots C_m, \qquad m \geqq 2.$$

We can write

$$w = w_1w_2 \ldots w_m, \qquad \text{where } C_i \overset{*}{\underset{G_2}{\Longrightarrow}} w_i, \quad 1 \leqq i \leqq m.$$

If C_i is in $V_N' - V_N$, then there is only one production of P' we may use, namely $C_i \to a_i$ for some a_i in V_T. In this case, $a_i = w_i$. By the construction of P', there is a production $A \to B_1B_2 \ldots B_m$ of P where $B_i = C_i$ if C_i is in V_N and $B_i = a_i$ if C_i is in $V_N' - V_N$. For those C_i in V_N, we know that the derivation $C_i \overset{*}{\underset{G_2}{\Longrightarrow}} w_i$ takes no more than k steps, so by the inductive hypothesis, $B_i \overset{*}{\underset{G_1}{\Longrightarrow}} w_i$. Hence $A \overset{*}{\underset{G_1}{\Longrightarrow}} w$.

---

† Note that G_2 is not yet in Chomsky normal form.

We have now proved the intermediate result that any context-free language can be generated by a grammar for which every production is either of the form $A \to a$ or $A \to B_1B_2 \ldots B_m$, for $m \geqq 2$. Here A and B_1, $B_2, \ldots, B_m$ are variables and a is a terminal.

Let us consider such a grammar $G_2 = (V_N', V_T, P', S)$. We modify G_2 by adding some additional symbols to V_N' and replacing some productions of P'. For each production $A \to B_1B_2 \ldots B_m$ of P' for $m \geqq 3$ we create new variables $D_1, D_2, \ldots, D_{m-2}$ and replace $A \to B_1B_2 \ldots B_m$ by the set of productions

$$\{A \to B_1D_1, D_1 \to B_2D_2, \ldots, D_{m-3} \to B_{m-2}D_{m-2}, D_{m-2} \to B_{m-1}B_m\}.$$

Let V_N'' be the new nonterminal vocabulary and P'' the new set of productions. Let $G_3 = (V_N'', V_T, P'', S)$. It is clear that if $A \overset{*}{\underset{G_2}{\Rightarrow}} \beta$, then $A \overset{*}{\underset{G_3}{\Rightarrow}} \beta$ so $L(G_2) \subseteq L(G_3)$. But it is also true that $L(G_3) \subseteq L(G_2)$, as can be shown in essentially the same manner as it was shown that $L(G_2) \subseteq L(G_1)$. The proof will be left to the reader.

Example 4.1. Let us consider the grammar $(\{S, A, B\}, \{a, b\}, P, S)$ which has the productions:

$$\begin{array}{ll} S \to bA & S \to aB \\ A \to a & B \to b \\ A \to aS & B \to bS \\ A \to bAA & B \to aBB \end{array}$$

and find an equivalent grammar in Chomsky normal form.

First, the only productions already in proper form are $A \to a$ and $B \to b$. There are no productions of the form $C \to D$, where C and D are variables, so we may begin by replacing terminals on the right by variables, except in the case of the productions $A \to a$ and $B \to b$. $S \to bA$ is replaced by $S \to C_1A$ and $C_1 \to b$. Similarly, $A \to aS$ is replaced by $A \to C_2S$ and $C_2 \to a$. $A \to bAA$ is replaced by $A \to C_3AA$ and $C_3 \to b$. $S \to aB$ is replaced by $S \to C_4B$ and $C_4 \to a$. $B \to bS$ is replaced by $B \to C_5S$ and $C_5 \to b$. $B \to aBB$ is replaced by $B \to C_6BB$ and $C_6 \to a$.

In the next stage, the production $A \to C_3AA$ is replaced by $A \to C_3D_1$ and $D_1 \to AA$, and the production $B \to C_6BB$ is replaced by $B \to C_6D_2$ and $D_2 \to BB$. The productions for the grammar in Chomsky normal form are shown below.

$$\begin{array}{llll} S \to C_1A & S \to C_4B & C_1 \to b & C_4 \to a \\ A \to C_2S & B \to C_5S & C_2 \to a & C_5 \to b \\ A \to C_3D_1 & B \to C_6D_2 & C_3 \to b & C_6 \to a \\ D_1 \to AA & D_2 \to BB & A \to a & B \to b \end{array}$$

4.3 GREIBACH NORMAL FORM

We now develop a normal-form theorem which uses productions whose right-hand sides each start with a terminal symbol, perhaps followed by some variables. First we prove two lemmas which say we can modify the productions of a cfg in certain ways without affecting the language generated.

Lemma 4.2. Define an *A-production* to be a production with a variable A on the left. Let $G = (V_N, V_T, P, S)$ be a context-free grammar. Let $A \to \alpha_1 B \alpha_2$ be a production in P and $\{B \to \beta_1, B \to \beta_2, \ldots, B \to \beta_r\}$ be the set of all B-productions. Let $G_1 = (V_N, V_T, P_1, S)$ be obtained from G by deleting the production $A \to \alpha_1 B \alpha_2$ from P and adding the productions $A \to \alpha_1\beta_1\alpha_2$, $A \to \alpha_1\beta_2\alpha_2, \ldots, A \to \alpha_1\beta_r\alpha_2$. Then $L(G) = L(G_1)$.

Proof. Obviously $L(G_1) \subseteq L(G)$, since if $A \to \alpha_1\beta_i\alpha_2$ is used in a derivation of G_1, then

$$A \underset{G}{\Rightarrow} \alpha_1 B \alpha_2 \underset{G}{\Rightarrow} \alpha_1\beta_i\alpha_2$$

can be used in G. To show that $L(G) \subseteq L(G_1)$, one simply notes that $A \to \alpha_1 B \alpha_2$ is the only production in G not in G_1. Whenever $A \to \alpha_1 B \alpha_2$ is used in a derivation by G, the variable B must be rewritten at some later step using a production of the form $B \to \beta_i$. These two steps can be replaced by the single step $A \underset{G}{\Rightarrow} \alpha_1\beta_i\alpha_2$.

Lemma 4.3. Let $G = (V_N, V_T, P, S)$ be a context-free grammar. Let

$$\{A \to A\alpha_1, A \to A\alpha_2, \ldots, A \to A\alpha_r\}$$

be the set of A-productions for which A is the leftmost symbol of the right-hand side. Let

$$A \to \beta_1, A \to \beta_2, \ldots, A \to \beta_s$$

be the remaining productions with A on the left. Let $G_1 = (V_N \cup \{Z\}, V_T, P_1, S)$ be the cfg formed by adding the variable Z to V_N, and replacing all the A-productions by the productions:

$$(1)\quad \left.\begin{array}{l} A \to \beta_i, \\ A \to \beta_i Z, \end{array}\right\} \quad 1 \leqq i \leqq s \qquad (2)\quad \left.\begin{array}{l} Z \to \alpha_i, \\ Z \to \alpha_i Z, \end{array}\right\} \quad 1 \leqq i \leqq r.$$

Then $L(G_1) = L(G)$.

Proof. Before proving the lemma, we point out that the A-productions alone, by leftmost derivations, generate the regular set

$$\{\beta_1, \beta_2, \ldots, \beta_s\}\{\alpha_1, \alpha_2, \ldots, \alpha_r\}^*,$$

and this is precisely the set generated by the productions in G_1 with A or Z on the left.

Let x be in $L(G)$. From a leftmost derivation of x by G we can construct a derivation of x by G_1 as follows: Whenever there occurs in the leftmost derivation a sequence of steps

$$tA\gamma \underset{G}{\Rightarrow} tA\alpha_{j_1}\gamma \underset{G}{\Rightarrow} tA\alpha_{j_2}\alpha_{j_1}\gamma \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} tA\alpha_{j_p}\cdots\alpha_{j_2}\alpha_{j_1}\gamma \underset{G}{\Rightarrow} t\beta_i\alpha_{j_p}\cdots\alpha_{j_2}\alpha_{j_1}\gamma,$$

replace the entire sequence by

$$tA\gamma \underset{G_1}{\Rightarrow} t\beta_i Z\gamma \underset{G_1}{\Rightarrow} t\beta_i\alpha_{j_p}Z\gamma \underset{G_1}{\Rightarrow} \cdots \underset{G_1}{\Rightarrow} t\beta_i\alpha_{j_p}\cdots\alpha_{j_2}Z\gamma \underset{G_1}{\Rightarrow} t\beta_i\alpha_{j_p}\cdots\alpha_{j_2}\alpha_{j_1}\gamma.$$

The resulting derivation is a derivation of x in G_1, although not a leftmost derivation. Thus $L(G) \subseteq L(G_1)$.

Now consider a leftmost derivation of x in G_1. Whenever a Z is introduced into the sentential form, reorder the derivation by immediately applying the productions that cause the Z to disappear. That is, for some instance of Z, a production $Z \to \alpha Z$ may be used. Then, in the leftmost derivation, α will derive a terminal string, and another production involving Z will be used. It should be clear that α could be left, temporarily, and the productions with Z on the left used immediately. Of course, the derivation will no longer be leftmost. Finally, a production $Z \to \beta$ will be used, where β has no Z. Then, the α's generated, as well as β, can be expanded normally. The result of the revised order of derivation will be the same as the original leftmost derivation.

Replace the resulting sequence of steps involving Z, namely:

$$tA\gamma \underset{G_1}{\Rightarrow} t\beta_i Z\gamma \underset{G_1}{\Rightarrow} t\beta_i\alpha_{j_p}Z\gamma \underset{G_1}{\Rightarrow} \cdots \underset{G_1}{\Rightarrow} t\beta_i\alpha_{j_p}\cdots\alpha_{j_2}Z\gamma \underset{G_1}{\Rightarrow} t\beta_i\alpha_{j_p}\cdots\alpha_{j_2}\alpha_{j_1}\gamma.$$

by

$$tA\gamma \underset{G}{\Rightarrow} tA\alpha_{j_1}\gamma \underset{G}{\Rightarrow} tA\alpha_{j_2}\alpha_{j_1}\gamma \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} tA\alpha_{j_p}\cdots\alpha_{j_2}\alpha_{j_1}\gamma \underset{G}{\Rightarrow} t\beta_i\alpha_{j_p}\cdots\alpha_{j_2}\alpha_{j_1}\gamma.$$

The result is a derivation of x in G. Thus $L(G_1) \subseteq L(G)$.

Theorem 4.6. (*Greibach Normal Form.*) Every context-free language L can be generated by a grammar for which every production is of the form $A \to a\alpha$, where A is a variable, a is a terminal, and α is a (possibly empty) string of variables.

Proof. Let $G = (V_N, V_T, P, S)$ be a Chomsky normal-form grammar generating the cfl L. Assume that $V_N = \{A_1, A_2, \ldots, A_m\}$. The first step in the construction is to modify the productions so that if $A_i \to A_j\gamma$ is a production, then $j > i$. This will be done as follows, starting with A_1 and proceeding to A_m. Assume that the productions have been modified so that, for $1 \leqq i \leqq k$, $A_i \to A_j\gamma$ is a production only if $j > i$. We now modify the A_{k+1}-productions.

If $A_{k+1} \to A_j\gamma$ is a production, with $j < k + 1$, we generate a new set of productions by substituting for A_j the right-hand side of each A_j-produc-

tion according to Lemma 4.2. By repeating the process $k - 1$ times at most, we obtain productions of the form $A_{k+1} \to A_l\gamma$, $l \geqq k + 1$. The productions with $l = k + 1$ are then replaced according to Lemma 4.3, introducing a new variable Z_{k+1}.

By repeating the above process for each original variable, we have only productions of the forms:

1. $A_k \to A_l\gamma$, $\quad l > k$
2. $A_k \to a\gamma$, $\quad a$ in V_T
3. $Z_k \to \gamma$, $\quad \gamma$ in $(V_N \cup \{Z_1, Z_2, \ldots, Z_m\})^*$.

Note that the leftmost symbol on the right-hand side of any production for A_m must be a terminal, since A_m is the highest-numbered variable. The leftmost symbol on the right-hand side of any production for A_{m-1} must be either A_m or a terminal symbol. When it is A_m, we can generate new productions by replacing A_m by the right-hand side of the productions for A_m according to Lemma 4.2. These productions must have right-hand sides that start with a terminal symbol. We then proceed to the productions for $A_{m-2}, \ldots, A_2, A_1$ until the right-hand side of each production for an A_i starts with a terminal symbol.

As the last step we examine the productions for the new variables, $Z_1, Z_2, \ldots, Z_m$. These productions start with either a terminal symbol or an original variable. Thus one more application of Lemma 4.2 for each Z_i production completes the construction.

Example 4.2. Convert to Greibach normal form, the grammar

$$G = (\{A_1, A_2, A_3\}, \{a, b\}, P, A_1),$$

where P consists of the following.

$$\begin{array}{ll} A_1 \to A_2A_3 & A_3 \to A_1A_2 \\ A_2 \to A_3A_1 & A_3 \to a \\ A_2 \to b & \end{array}$$

Step 1. Since the right-hand side of the productions for A_1 and A_2 start with terminals or higher-numbered variables, we begin with the production $A_3 \to A_1A_2$ and substitute the string A_2A_3 for A_1. Note that $A_1 \to A_2A_3$ is the only production with A_1 on the left.

The resulting set of productions is:

$$\begin{array}{ll} A_1 \to A_2A_3 & A_3 \to A_2A_3A_2 \\ A_2 \to A_3A_1 & A_3 \to a. \\ A_2 \to b & \end{array}$$

Since the right-hand side of the production $A_3 \to A_2A_3A_2$ begins with a lower-numbered variable, we substitute for the first occurrence of A_2 either

A_3A_1 or b. Thus $A_3 \to A_2A_3A_2$ is replaced by $A_3 \to A_3A_1A_3A_2$ and $A_3 \to bA_3A_2$. The new set is

$$\begin{array}{ll} A_1 \to A_2A_3 & A_3 \to A_3A_1A_3A_2 \\ A_2 \to A_3A_1 & A_3 \to bA_3A_2 \\ A_2 \to b & A_3 \to a \end{array}$$

We now apply Lemma 4.3 to the productions

$$A_3 \to A_3A_1A_3A_2, \qquad A_3 \to bA_3A_2, \qquad \text{and} \qquad A_3 \to a.$$

Symbol Z_3 is introduced, and the production $A_3 \to A_3A_1A_3A_2$ is replaced by

$$A_3 \to bA_3A_2Z_3, \quad A_3 \to aZ_3, \quad Z_3 \to A_1A_3A_2, \quad \text{and} \quad Z_3 \to A_1A_3A_2Z_3.$$

The resulting set is

$$\begin{array}{ll} A_1 \to A_2A_3 & A_3 \to bA_3A_2Z_3 \\ A_2 \to A_3A_1 & A_3 \to aZ_3 \\ A_2 \to b & Z_3 \to A_1A_3A_2Z_3 \\ A_3 \to bA_3A_2 & Z_3 \to A_1A_3A_2. \\ A_3 \to a & \end{array}$$

Step 2. Now all the productions with A_3 on the left have right-hand sides that start with terminals. These are used to replace A_3 in the production $A_2 \to A_3A_1$ and then the productions with A_2 on the left are used to replace A_2 in the production $A_1 \to A_2A_3$. The result is the following.

$$\begin{array}{ll} A_3 \to bA_3A_2 & A_3 \to bA_3A_2Z_3 \\ A_3 \to a & A_3 \to aZ_3 \\ A_2 \to bA_3A_2A_1 & A_2 \to bA_3A_2Z_3A_1 \\ A_2 \to aA_1 & A_2 \to aZ_3A_1 \\ A_2 \to b & \\ A_1 \to bA_3A_2A_1A_3 & A_1 \to bA_3A_2Z_3A_1A_3 \\ A_1 \to aA_1A_3 & A_1 \to aZ_3A_1A_3 \\ A_1 \to bA_3 & \\ Z_3 \to A_1A_3A_2Z_3 & Z_3 \to A_1A_3A_2 \end{array}$$

Step 3. The two Z_3 productions are converted to proper form resulting in ten more productions. That is, the productions

$$Z_3 \to A_1A_3A_2 \qquad \text{and} \qquad Z_3 \to A_1A_3A_2Z_3$$

are altered by substituting the right-hand side of each of the five productions with A_1 on the left for the first occurrences of A_1. Thus, $Z_3 \to A_1A_3A_2$ becomes

$$Z_3 \to bA_3A_3A_2, \quad Z_3 \to bA_3A_2A_1A_3A_3A_2, \quad Z_3 \to aA_1A_3A_3A_2,$$
$$Z_3 \to bA_3A_2Z_3A_1A_3A_3A_2, \quad \text{and} \quad Z_3 \to aZ_3A_1A_3A_3A_2.$$

The other production for Z_3 is replaced similarly. The final set of productions is:

$A_3 \to bA_3A_2$ $\quad$ $A_3 \to bA_3A_2Z_3$
$A_3 \to a$ $\quad$ $A_3 \to aZ_3$

$A_2 \to bA_3A_2A_1$ $\quad$ $A_2 \to bA_3A_2Z_3A_1$
$A_2 \to aA_1$ $\quad$ $A_2 \to aZ_3A_1$
$A_2 \to b$

$A_1 \to bA_3A_2A_1A_3$ $\quad$ $A_1 \to bA_3A_2Z_3A_1A_3$
$A_1 \to aA_1A_3$ $\quad$ $A_1 \to aZ_3A_1A_3$
$A_1 \to bA_3$

$Z_3 \to bA_3A_3A_2$ $\quad$ $Z_3 \to bA_3A_3A_2Z_3$
$Z_3 \to bA_3A_2A_1A_3A_3A_2$ $\quad$ $Z_3 \to bA_3A_2A_1A_3A_3A_2Z_3$
$Z_3 \to aA_1A_3A_3A_2$ $\quad$ $Z_3 \to aA_1A_3A_3A_2Z_3$
$Z_3 \to bA_3A_2Z_3A_1A_3A_3A_2$ $\quad$ $Z_3 \to bA_3A_2Z_3A_1A_3A_3A_2Z_3$
$Z_3 \to aZ_3A_1A_3A_3A_2$ $\quad$ $Z_3 \to aZ_3A_1A_3A_3A_2Z_3$

4.4 SOLVABILITY OF FINITENESS AND THE "*uvwxy* THEOREM"

In Theorem 4.2 we showed that we could eliminate from a grammar those variables generating no terminal strings. In fact, we can do more. We can test if a language generated by a given symbol is finite or infinite and eliminate those variables, other than the sentence symbol, from which only a finite number of terminal strings can be derived. In proving this result, we shall show two results (Theorems 4.7 and 4.8) quite interesting in their own right.

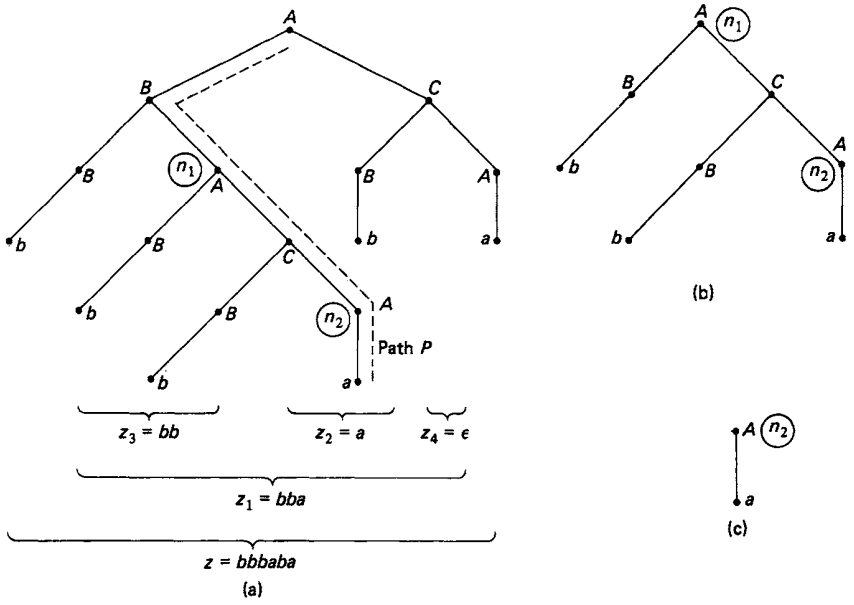
Theorem 4.7. Let L be any context-free language. There exist constants p and q depending only on L, such that if there is a word z in L, with $|z| > p$, then z may be written as $z = uvwxy$, where $|vwx| \leqq q$ and v and x are not both ϵ, such that for each integer $i \geqq 0$, uv^iwx^iy is in L.

Proof. Let $G = (V_N, V_T, P, S)$ be any Chomsky normal-form grammar for L. If G has k variables, then let $p = 2^{k-1}$ and let $q = 2^k$. It is easy to see that, for a Chomsky normal-form grammar, if a derivation tree has no path of length greater than j, then the terminal string derived is of length no greater than 2^{j-1}. The proof is left to the reader.

Hence, if z is in L and $|z| > p$, then the tree for any derivation of z by the grammar G contains a path of length greater than k. We consider a path P, of longest length, and observe that there must be two nodes, n_1 and n_2, satisfying the following conditions.

1. The nodes n_1 and n_2 both have the same label, say A.
2. Node n_1 is closer to the root than node n_2.
3. The portion of path P from n_1 to the leaf is of length at most $k + 1$.†

† Clearly, a path of longest length includes a leaf.



$z_1 = z_3 z_2 z_4$, where $z_3 = bb$ and $z_4 = \epsilon$.

$G = (\{A,B,C\},\{a,b\},\{A \to BC, B \to BA, C \to BA, A \to a, B \to b\},A)$

Fig. 4.2. Illustration of subtrees T_1 and T_2 of Theorem 4.7. (a) Tree. (b) Subtree T_1. (c) Subtree T_2.

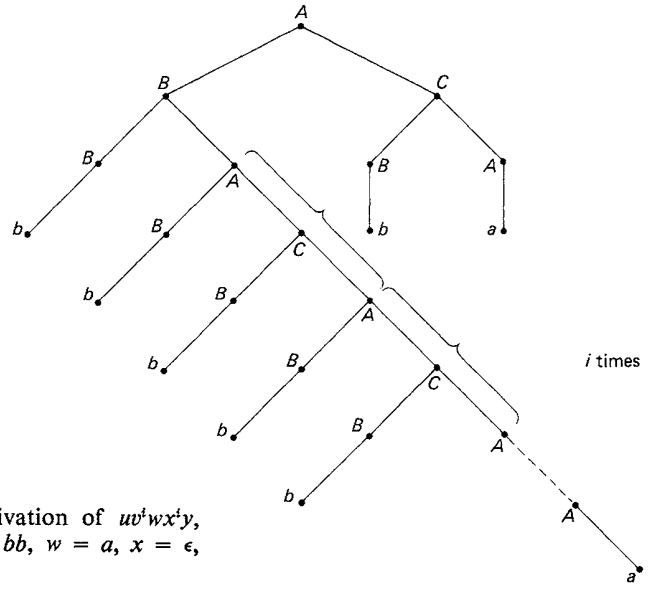


Fig. 4.3. The derivation of $uv^i wx^i y$, where $u = b$, $v = bb$, $w = a$, $x = \epsilon$, $y = ba$.

To see that n_1 and n_2 can always be found, just proceed up path P from the leaf, keeping track of the labels encountered. Of the first $k + 2$ nodes, only the leaf has a terminal label. The remaining $k + 1$ nodes cannot have distinct variable labels.

Now the subtree T_1 with root n_1 represents the derivation of a subword of length at most 2^k (and hence, of length less than or equal to q). This is true because there can be no path in T_1 of length greater than $k + 1$, since P was a path of longest length in the entire tree. Let z_1 be the result of the subtree T_1. If T_2 is the subtree generated by node n_2 and z_2 is the result of the subtree T_2, then we can write z_1 as $z_3z_2z_4$. Furthermore, z_3 and z_4 cannot both be ϵ, since the first production used in the derivation of z_1 must be of the form $A \to BC$ for some variables B and C. The subtree T_2 must be completely within either the subtree generated by B or the subtree generated by C. The above is illustrated in Fig. 4.2.

We now know that

$$A \overset{*}{\underset{G}{\Rightarrow}} z_3Az_4 \overset{*}{\underset{G}{\Rightarrow}} z_3z_2z_4, \qquad \text{where } |z_3z_2z_4| \leqq q.$$

But it follows that $A \overset{*}{\underset{G}{\Rightarrow}} z_3^i z_2 z_4^i$ for each $i \geqq 0$. See Fig. 4.3. The string z can clearly be written as $uz_3z_2z_4y$, for some u and y. We let $z_3 = v$, $z_2 = w$, and $z_4 = x$, to complete the proof.

Theorem 4.8. There is an algorithm to determine if a given context-free grammar G generates a finite or infinite number of words.

Proof. Let p and q be the constants defined in Theorem 4.7. Thus, if z is in $L(G)$ and $|z| > p$, then z can be written as $uvwxy$ where for each $i \geqq 0$, uv^iwx^iy is in $L(G)$. Also $|v| + |x| > 0$. Hence if there is a word in $L(G)$ of length greater than or equal to p, then $L(G)$ is infinite.

Suppose that $L = L(G)$ is infinite. Then there are arbitrarily long words in $L(G)$ and, in particular, a word of length greater than $p + q$. This word may be written as

$$uvwxy, \quad \text{where } |vwx| \leqq q, \quad |v| + |x| > 0,$$

and uv^iwx^iy is in L for all $i \geqq 0$. In particular, uwy is in L, and $|uwy| < |uvwxy|$. Also $|uwy| > p$. If $|uwy| > p + q$, we repeat the procedure until we eventually find a word in L of length l, $p < l \leqq p + q$. Thus L is infinite if and only if it contains a word of length l, $p < l \leqq p + q$.

Since we may test whether a given word is in a given context-free language (Theorem 2.2), we have merely to test all words of length between p and $p + q$ for membership in $L(G)$. If there is such a word, then L is clearly infinite; if not, then there are no words of length greater than p in L, so L is finite.

Theorem 4.9. Given a context-free grammar G_1, we can find an equivalent grammar G_2 for which, if A is a variable of G_2 other than the sentence symbol, there are an infinity of terminal strings derivable from A.

Proof. If $L(G_1)$ is finite, the theorem is trivial, so assume that $L(G_1)$ is infinite. If $G_1 = (V_N, V_T, P_1, S)$, then for each A in V_N, we know from Theorem 4.8 that by considering the context-free grammar $G_A = (V_N, V_T, P_1, A)$ we can determine whether there is an infinity of terminal strings w, such that $A \overset{*}{\underset{G_1}{\Rightarrow}} w$. Suppose that $A_1, A_2, \ldots, A_k$ are exactly the variables generating an infinity of terminal strings, and that $B_1, B_2, \ldots, B_m$ are exactly those generating a finite number of terminal strings. We create a new set of productions, P_2, from P_1 as follows.

Suppose that $C_0 \to C_1C_2 \ldots C_r$ is a production of P_1; C_0 is among $A_1, A_2, \ldots, A_k$. Then every production of the form $C_0 \to u_1u_2 \ldots u_r$ is in P_2, where for $1 \leqq i \leqq r$,

1. If C_i is terminal, $u_i = C_i$.
2. If C_i is among $A_1, A_2, \ldots, A_k$, then $u_i = C_i$.
3. If C_i is among $B_1, B_2, \ldots, B_m$, u_i is one of the finite number of terminal words such that $C_i \overset{*}{\underset{G_1}{\Rightarrow}} u_i$.

We know that P_2 contains no productions with any of $B_1, B_2, \ldots, B_m$ on the left. We consider the new grammar $G_2 = (V_N', V_T, P_2, S)$, where $V_N' = \{A_1, A_2, \ldots, A_k\}$. Note that S must be in V_N', since L is assumed to be infinite. Surely, if $\alpha \underset{G_2}{\Rightarrow} \beta$, then $\alpha \overset{*}{\underset{G_1}{\Rightarrow}} \beta$, so $L(G_2) \subseteq L(G_1)$.

As usual, to show that $L(G_1) \subseteq L(G_2)$, we prove by induction on the number of steps in the derivation that if

$$A_i \overset{*}{\underset{G_1}{\Rightarrow}} w, \qquad 1 \leqq i \leqq k,$$

where w is a terminal string, then $A_i \overset{*}{\underset{G_2}{\Rightarrow}} w$. The result is trivial for one-step derivations, so assume that it is true for up to j steps. Now suppose that in a derivation of $j + 1$ steps, the first production used is $A_i \to C_1C_2 \ldots C_r$. We can write w as

$$w_1w_2 \ldots w_r, \qquad \text{where } C_i \overset{*}{\underset{G_1}{\Rightarrow}} w_i, \quad 1 \leqq i \leqq r.$$

There is a production $A_i \to u_1u_2 \ldots u_r$ in P_2, where $u_p = w_p$ if either C_p is a terminal or if C_p is among $B_1, B_2, \ldots, B_m$ and $u_p = C_p$ if C_p is among $A_1, A_2, \ldots, A_k$. The inductive step follows immediately.

Example 4.3. Consider the grammar $G = (\{S, A, B\}, \{a, b, c, d\}, \{S \to ASB, S \to AB, A \to a, A \to b, B \to c, B \to d\}, S)$. It is easy to see that A generates only the strings a and b and B generates only the strings c and d. How-

ever, S generates an infinity of strings. The only productions with S on the left are $S \to ASB$ and $S \to AB$. The production $S \to ASB$ is replaced by $S \to aSc$, $S \to aSd$, $S \to bSc$, and $S \to bSd$. Likewise, the production $S \to AB$ is replaced by $S \to ac$, $S \to ad$, $S \to bc$, and $S \to bd$. The new grammar is

$$G_2 = (\{S\}, \{a, b, c, d\}, P, S)$$

where

$$P = \{S \to aSc, S \to aSd, S \to bSc, S \to bSd, S \to ac, S \to ad, S \to bc, S \to bd\}.$$

4.5 THE SELF-EMBEDDING PROPERTY

A context-free grammar G is said to be *self-embedding* if there is a variable A with the property that $A \overset{*}{\underset{G}{\Rightarrow}} \alpha_1 A \alpha_2$ where α_1 and α_2 are nonempty strings. The variable A is also said to be *self-embedding*. Note that it is the self-embedding property that gives rise to sentences of the form uv^iwx^iy. One gets the feeling that it is the self-embedding property that distinguishes a strictly context-free language from a regular set. One should note that simply because a grammar is self-embedding does not mean that the language generated is not regular. For example, the grammar

$$G = (\{S\}, \{a, b\}, P, S),$$

where

$$P = \{S \to aSa, S \to aS, S \to bS, S \to a, S \to b\}$$

generates a regular set. In fact, $L(G) = \{a, b\}^+$.

In this section we shall see that a context-free grammar that is not self-embedding generates a regular set. Consequently, a context-free language is nonregular if and only if all of its grammars are self-embedding.

Theorem 4.10. Let G be a non-self-embedding context-free grammar. Then $L(G)$ is a regular set.

Proof. In examining the constructions for the normal forms developed in this chapter, we note that each of the constructions has the property that if the original grammar was non-self-embedding, then the normal-form grammar was non-self-embedding. In particular this is true of the Greibach normal form. Thus, if G is non-self-embedding, we can find a grammar $G_1 = (V_N, V_T, P_1, S_1)$ in Greibach normal form, equivalent to G, which is non-self-embedding.† Moreover, by Theorem 4.2 a terminal string can be derived from each variable in V_N.

† Although the statement is not obvious, it is easy to prove. Clearly the application of Lemma 4.2 does not introduce self-embedding. In Lemma 4.3 one must show that Z is self-embedding only if A is self-embedding.

Consider a leftmost derivation in G_1. If G_1 has m variables, and l is the length of the longest right-hand side of any production, then no sentential form can have more than ml variables appearing in it. To see this, assume that more than ml variables appear in some sentential form α of a leftmost derivation. In the derivation tree for α, consider those nodes on the path from the root to the leftmost variable of α. In particular, consider those nodes where variables are introduced to the right of the path. Since the maximum number of variables coming directly from any node is $l - 1$, and since a variable to the right of the above path has not been rewritten, there must be at least $m + 1$ such nodes. Thus some variable A must appear twice among the labels of these nodes. Since we are considering only nodes where new variables are introduced to the right of the path, and since each production introduces a terminal as the leftmost character, the variable A must be self-embedding.

Now if there are at most ml variables in any sentential form, we can design a type 3 grammar $G_2 = (V_N', V_T, P_2, S)$, generating $L(G)$ as follows. The variables of G_2 correspond to strings of variables of G_1 of length less than or equal to ml. That is, $V_N' = \{[\alpha] \mid |\alpha| \leqq ml \text{ and } \alpha \text{ in } V_N^+\}$. S is $[S_1]$. If $A \to b\alpha$ is in P_1, then for all variables of V_N' corresponding to strings starting with A we have $[A\beta] \to b[\alpha\beta]$ in P_2 provided that $|\alpha\beta| \leqq ml$. It should be obvious from the construction that G_2 simulates all leftmost derivations in G_1, so $L(G_2) = L(G_1)$. Thus $L(G)$ is regular.

4.6 ϵ-RULES IN CONTEXT-FREE GRAMMARS

Earlier we showed that several restrictions can be placed on the productions of context-free grammars without limiting the class of languages that can be generated. Now we consider an extension of context-free grammars to include productions of the form $A \to \epsilon$ for any variable A. Such a production is called an *ϵ-rule*. Many descriptions of context-free languages allow these productions. We shall show that a language generated by a cfg with ϵ-rules is always a cfl.

The concepts concerning trees for context-free grammars carry over directly to these augmented grammars. One simply allows ϵ to be the label of a node. Clearly, that node must be a leaf.

Theorem 4.11. If L is a language generated by a grammar $G = (V_N, V_T, P, S)$ and every production in P is of the form $A \to \alpha$, where A is a variable and α is a string (possibly ϵ) in V^*, then L can be generated by a grammar in which every production is either of the form $A \to \alpha$, with A a variable and α in V^+, or $S \to \epsilon$, and further, S does not appear on the right of any production.

Proof. By a trivial extension of Lemma 2.1, we can assume that S does not appear on the right-hand side of any production in P. For any variable A

of G we can decide whether $A \overset{*}{\underset{G}{\Rightarrow}} \epsilon$. For if so, then there is a derivation whose tree has no path longer than the number of variables of G. (This argument was used in Theorem 4.1.)

Let $A_1, A_2, \ldots, A_k$ be those variables of V from which ϵ can be derived and $B_1, B_2, \ldots, B_m$ be those from which it cannot. We construct a new set of productions P_1 according to the following rules.

1. If $S \overset{*}{\underset{G}{\Rightarrow}} \epsilon$, then $S \to \epsilon$ is in P_1.
2. No other production of the form $A \to \epsilon$ appears in P_1.
3. If
$$A \to C_1C_2 \ldots C_r, \qquad r \geqq 1,$$
is in P, then each production of the form $A \to \alpha_1\alpha_2 \ldots \alpha_r$ is in P_1, where if C_i is in $V_T \cup \{B_1, B_2, \ldots, B_m\}$, then $C_i = \alpha_i$, and if C_i is in $\{A_1, A_2, \ldots, A_k\}$, then α_i may be C_i or ϵ. However, not all α_i's may be ϵ.

As usual, it should be clear that if $G_1 = (V_N, V_T, P_1, S)$, then $L(G_1) \subseteq L(G)$. We must show by induction on the number of steps in the derivation, that if $A \overset{*}{\underset{G}{\Rightarrow}} w$, $w \neq \epsilon$, then $A \overset{*}{\underset{G_1}{\Rightarrow}} w$, for A in V_N. For one step, the result is obvious, so assume that it is true for up to k steps. Suppose that $A \overset{*}{\underset{G}{\Rightarrow}} w$ by a $k+1$-step derivation and suppose that $A \to C_1C_2 \ldots C_r$ is the first production used. We can write w as $w_1w_2 \ldots w_r$ where for $1 \leqq i \leqq r$, $C_i \overset{*}{\underset{G}{\Rightarrow}} w_i$. If $w_i \neq \epsilon$, then by induction we know that $C_i \overset{*}{\underset{G_1}{\Rightarrow}} w_i$. Now, there is a production of P_1 of the form $A \to \alpha_1\alpha_2 \ldots \alpha_r$, where $\alpha_i = C_i$ if $w_i \neq \epsilon$ and $\alpha_i = \epsilon$ if $w_i = \epsilon$. Hence

$$A \overset{*}{\underset{G_1}{\Rightarrow}} w.$$

It follows immediately from Theorem 4.11 that the only difference between context-free grammars with productions of the form $A \to \epsilon$ and those with no such productions is that the former may include ϵ as a word in the language. From here on we call a context-free grammar with ϵ-rules simply a context-free grammar, knowing that an equivalent context-free grammar without ϵ productions (except for $S \to \epsilon$, possibly) can be found.

4.7 SPECIAL TYPES OF CONTEXT-FREE LANGUAGES AND GRAMMARS

At this point, we mention several restricted classes of context-free languages. If every production of a cfg is of the form $A \to uBv$ or $A \to u$, A and B variables, u and v terminal strings, then we say that the grammar is *linear*. A language that can be generated by a linear grammar is called a *linear language*. Not all context-free languages are linear languages. Observe that no string derivable in a linear grammar has more than one variable.

Example 4.4. The grammar

$$G = (\{S\}, \{0, 1\}, \{S \to 0S1, S \to \epsilon\}, S)$$

is a linear grammar which generates $\{0^n1^n | n \geqq 0\}$.

A grammar $G = (V_N, V_T, P, S)$ is said to be *sequential* if the variables in V_N can be ordered $A_1, A_2, \ldots, A_k$ such that if $A_i \to \alpha$ is a production in P, then α contains no A_j with $j < i$. A language generated by a sequential grammar is called a *sequential language.*

Example 4.5. The grammar

$$G = (\{A_1, A_2\}, \{0, 1\}, \{A_1 \to A_2A_1, A_1 \to A_2, A_2 \to 0A_21, A_2 \to \epsilon\}, A_1)$$

is a sequential grammar which generates the language $\{0^n1^n | n \geqq 0\}^*$.

If a context-free language L over an alphabet V_T is a subset of the language $w_1^*w_2^* \ldots w_k^*$ † for some k, where w_i is in V_T^*, $1 \leqq i \leqq k$, then we say that L is a *bounded language.*

Example 4.6. The language

$$\{(ab)^n c^n (dd)^* | n \geqq 1\}$$

is a bounded language. Here $k = 3$ and $w_1 = ab$, $w_2 = c$ and $w_3 = d$.

A context-free grammar $G = (V_N, V_T, P, S)$ is said to be *ambiguous* if there is a word in $L(G)$ with two or more distinct leftmost derivations. If every grammar generating a context-free language is ambiguous, we say that the language is *inherently ambiguous.*

There exist inherently ambiguous context-free languages. An example is the language $L = \{a^ib^jc^k | i = j \text{ or } j = k\}$. Essentially the reason that L is inherently ambiguous is that any context-free grammar generating L must generate those words for which $i = j$ by a process different from that used to generate those words for which $j = k$. It is impossible not to generate some of those words for which $i = j = k$ by both processes.

Example 4.7. Consider the grammar G of Example 4.1, which had productions $S \to bA$, $S \to aB$, $A \to a$, $B \to b$, $A \to aS$, $B \to bS$, $A \to bAA$, $B \to aBB$. The word *aabbab* has the following two leftmost derivations:

$$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aabB \Rightarrow aabbS \Rightarrow aabbaB \Rightarrow aabbab$$

$$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aabSB \Rightarrow aabbAB \Rightarrow aabbaB \Rightarrow aabbab.$$

Hence G is ambiguous. However, the language

$$L(G) = \{w | w \text{ consists of an equal number of } a\text{'s and } b\text{'s}\}$$

---

† Strictly speaking $w_1^*w_2^* \ldots w_k^*$ should be written $\{w_1\}^*\{w_2\}^* \ldots \{w_k\}^*$. No confusion should result.

is not inherently ambiguous. For example, $L(G)$ is generated by the unambiguous grammar

$$G_1 = (\{S, A, B\}, \{a, b\}, P, S),$$

where P consists of:

$S \to aBS, S \to aB, S \to bAS, S \to bA, A \to bAA, A \to a, B \to aBB, B \to b.$

PROBLEMS

4.1 Given a context-free grammar with m variables, for which the right side of no production is longer than l, provide an upper bound on the number of trees with no path longer than $m + 1$?

4.2 Give a simple algorithm to determine if the language generated by a cfg is empty. Note that if a grammar generates a nonempty language, at least one variable must have a production whose right-hand side contains only terminals.

4.3 Given two strings α_1 and α_2 and a context-free grammar G, give an algorithm to determine if $\alpha_1 \overset{*}{\underset{G}{\Longrightarrow}} \alpha_2$.

4.4 Complete the proof of Theorem 4.5.

4.5 Consider the grammar

$$G = (\{S, T, L\}, \{a, b, +, -, \times, /, [,], \}, P, S),$$

where P consists of the productions:

$S \to T + S$	$T \to L \times T$	$L \to [S]$
$S \to T - S$	$T \to L/T$	$L \to a$
$S \to T$	$T \to L$	$L \to b.$

Informally describe $L(G)$. Find a grammar in Chomsky normal form generating $L(G)$.

4.6 Consider the grammar $G = (\{A, B, C\}, \{0, 1\}, P, A)$, where P consists of the productions:

$A \to 0A1$	$B \to 1C0$
$A \to 0AC$	$B \to AC$
$A \to 0$	$C \to 1CB$
$A \to 1B0$	$C \to AB$

Find an equivalent grammar G_1, such that if D is a variable of G_1, then $D \Longrightarrow w$ for some terminal string w.

4.7 If G is a Chomsky normal-form grammar, where w is in $L(G)$, and there is a derivation of w using p steps, how long is w? Prove your answer.

4.8 Show how to put the productions of a Chomsky normal-form grammar into the form $A \to BC$, where $B \neq C$, and if $A \to \alpha_1 B \alpha_2$ and $A \to \gamma_1 B \gamma_2$ are productions then $\alpha_1 = \gamma_1$ and $\alpha_2 = \gamma_2$.

4.9 Several times we have modified the productions of a grammar and then proved that the resulting grammar was equivalent to the original, one such case being Lemma 4.3. Can you think of a general type of modification that will include most of the special cases? Prove that your modification results in an equivalent grammar.

4.10 Consider the grammar $G = (\{S\}, \{p, [,], \sim, \supset\}, P, S)$ where P is the set of productions:

$$S \to p \qquad S \to \sim S \qquad S \to [S \supset S].$$

Describe $L(G)$ informally. Find a Chomsky normal-form grammar for $L(G)$. Number the variables, giving S the highest number. Find a Greibach normal-form grammar for $L(G)$ from the Chomsky normal-form grammar you have obtained. Can you find a simpler Greibach normal-form grammar for $L(G)$?

4.11 Show that every context-free language can be generated by a grammar in which every production is of the form $A \to a$, $A \to aB$, or $A \to aBC$, where a is terminal, and A, B, and C are variables.

4.12 Show that every context-free language can be generated by a grammar in which every production is of the form $A \to a$ or $A \to a\alpha b$, where a and b are terminals and α is a string of variables.

4.13 Use Theorem 4.7 to show that $\{a^i | i \text{ a prime}\}$ is not a context-free language.

4.14 Show that $\{a^i | i \text{ is a perfect square}\}$ is not a context-free language.

4.15 Use Theorem 4.7 to show that $\{a^i b^i c^i | i \geqq 1\}$ is not a context-free language.

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

$$\begin{array}{ll} S \to AB & B \to 0A1 \\ A \to BSB & B \to \epsilon \\ A \to BB & B \to 0 \\ & A \to 1. \end{array}$$

Find an equivalent grammar for which S does not appear on the right of any production and $S \to \epsilon$ is the only production with ϵ on the right.

4.17 Find a cfl that cannot be generated by a linear grammar.

4.18 Find a cfl that is not a bounded language.

4.19 Find a cfl that is not a sequential language.

4.20 Show that the grammar G_1 of Example 4.7 is unambiguous.

4.21 Which of the following grammars are self-embedding? Find finite automata accepting those languages which have non-self-embedding grammars.

a) $G = (\{A, B, C\}, \{a, b\}, P, A)$, where P contains the productions

$$\begin{array}{ll} A \to CB & C \to AB \\ A \to b & C \to a \\ B \to CA & \end{array}$$

b) $G = (\{A, B, C\}, \{a, b\}, P, A)$, where P contains the productions

$$\begin{array}{ll} A \to CB & A \to Ca \\ C \to AB & \\ B \to bC & C \to b \end{array}$$

4.22 Show that every cfl over a one-symbol alphabet is regular.

REFERENCES

The original work on context-free languages appears in Chomsky [1956], Chomsky [1959], and Bar-Hillel, Perles, and Shamir [1961]. Theorems 4.1, 4.7 and 4.8 are from the latter paper. Theorem 4.5 appears in Chomsky [1959] and Theorem 4.6 in Greibach [1965]. A simple proof of the latter result can be found in Rosenkrantz [1967]. Theorem 4.10 is from Chomsky [1959]. Ginsburg [1966] is a good reference on the properties of context-free languages. For results on linear languages, see Greibach [1963], Gross [1964], Haines [1964], and Greibach [1966]. For sequential languages, see Ginsburg and Rice [1962], Ginsburg and Rose [1963(a) and (b)], and Shamir [1965]. For bounded languages, see Ginsburg and Spanier [1964]. Ambiguity and inherent ambiguity are treated more extensively in Chapter 14. For the application of context-free languages to the area of programming, see Samelson and Bauer [1960], Irons [1961], Floyd [1962(a) and (b)], [1963], and [1964(a) and (c)], and Lewis and Stearns [1966].

There are two interesting theorems concerning context-free languages which we have not covered. We have not, in fact, developed the notation even to state them formally, but they deserve mention. The first is Parikh's Theorem (Parikh [1961]) which essentially states that if L is a cfl contained in Σ^*, then for each w in L, the numbers of instances of each symbol of Σ found in w satisfy one of a finite number of nontrivial sets of simultaneous linear equations. For example, $\{a^n b^{n^2} | n \geqq 1\}$ is not a context-free language, since the numbers of a's and b's in each word satisfy only quadratic equations.

The second theorem is the characterization of cfl's in terms of "Dyck languages." A Dyck language is a cfl generated by a grammar

$$G_k = (\{S\}, \{a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_k\}, P, S),$$

where P consists of the productions $S \to SS$, $S \to \epsilon$, and $S \to a_i S b_i$ for $1 \leqq i \leqq k$. $L(G_k)$ can be thought of as being composed of strings of balanced parentheses of k types. The corresponding left and right parentheses for each i are a_i and b_i. The theorem states that every cfl can be expressed as a homomorphism (see Chapter 9) of the intersection of a Dyck language and a regular set. The theorem was first proved in Chomsky [1962]. Alternative proofs appear in Stanley [1965] and Ginsburg [1966].

CHAPTER 5

PUSHDOWN AUTOMATA

5.1 INFORMAL DESCRIPTION

We shall now consider a device which is quite important in the study of formal languages—the pushdown automaton. This device is essentially a finite automaton with control of both an input tape and a pushdown store. The pushdown store is a "first in–last out" list. That is, symbols may be entered or removed only at the top of the list. When a symbol is entered at the top, the symbol previously at the top becomes second from the top, the symbol previously second from the top becomes third, etc. Similarly, when a symbol is removed from the top of the list, the symbol previously second from the top becomes the top symbol, the symbol previously third from the top becomes second, and so on.

A familiar example of a pushdown store is the stack of plates on a spring which we often see in cafeterias. There is a spring below the plates with just enough strength so that only one plate appears above the level of the counter. When that top plate is removed, the load on the spring is lightened, and the plate directly below appears above the level of the counter. If a plate is then put on top of the stack, the pile is pushed down, and that plate appears above the counter. For our purposes, we make the assumption that the spring is arbitrarily long so that we may add as many plates as we desire.

Let us see how we can use the stack of plates, coupled with a finite control, to recognize a nonregular set. The set $L = \{wcw^R | w \text{ in } \{0, 1\}^*\}$† is a context-free language, generated by the grammar

$$G = (\{S\}, \{0, 1, c\}, \{S \to 0S0, S \to 1S1, S \to c\}, S)$$

It is not hard to show that L cannot be accepted by any finite automaton. To accept L, we shall make use of a finite control with two states, q_1 and q_2, and a pushdown store on which we place blue, green, and red plates. The device will operate by the following rules.

1. The machine starts with one red plate on the stack and with the finite control in state q_1.

† w^R denotes w reversed.

2. If the input to the device is 0 and the device is in state q_1, a blue plate is placed on the stack. If the input to the device is 1 and the device is in state q_1, a green plate is placed on the stack. In both cases the finite control remains in state q_1.
3. If the input is c and the device is in state q_1, it changes state to q_2 without adding or removing any plates.
4. If the input is 0 and the device is in state q_2 with a blue plate on top of the stack, the plate is removed. If the input is 1 and the device is in state q_2 with a green plate on top of the stack, the plate is removed. In both cases the finite control remains in state q_2.
5. If the device is in state q_2 and a red plate is on top of the stack, the plate is removed without waiting for the next input.
6. For all cases other than those described above, the device can make no move.

The preceding moves are summarized in Fig. 5.1.

We say that the device described above accepts an input string if, on processing the last symbol of the string, the stack of plates becomes completely empty. Note that, once the stack is completely empty, no further moves are possible.

Essentially, the device operates in the following way. In state q_1, the device makes an image of its input by placing a blue plate on top of the stack of plates each time a 0 appears in the input and a green plate each time a 1

		INPUT		
Top plate	*State*	*0*	*1*	*c*
Blue	q_1	Add blue plate; stay in state q_1.	Add green plate; stay in state q_1.	Go to state q_2.
	q_2	Remove top plate; stay in state q_2.	—	—
Green	q_1	Add blue plate; stay in state q_1.	Add green plate; stay in state q_1.	Go to state q_2.
	q_2	—	Remove top plate; stay in state q_2.	—
Red	q_1	Add blue plate; stay in state q_1.	Add green plate; stay in state q_1.	Go to state q_2.
	q_2	Without waiting for next input, remove top plate.	Without waiting for next input, remove top plate.	Without waiting for next input, remove top plate

Fig. 5.1. Finite control for pushdown machine accepting $\{wcw^R | w \text{ in } \{0, 1\}^*\}$.

appears in the input. When c is the input, the device transfers to state q_2. Next, the remaining input is compared with the stack by removing a blue plate from the top of stack each time the input symbol is a 0 and a green plate each time the input symbol is a 1. Should the top plate be of the wrong color, the device halts and no further processing of the input is possible. If all plates match the inputs, eventually the red plate at the bottom of the stack is exposed. The red plate is immediately removed and the device is said to accept the input string. All plates can be removed only in the case where the string that enters the device after the c is the reversal of what entered before the c.

5.2 DEFINITIONS

We shall now formalize the concept of a pushdown automaton (pda). The pda will have an input tape, a finite control, and a pushdown store. The pushdown store is a string of symbols in some alphabet. The leftmost symbol will be considered to be at the "top" of the store. The device will be nondeterministic, having some finite number of choices of moves in each situation. The moves will be of two types. In the first type of move, an input symbol is scanned. Depending on the input symbol, the top symbol on the pushdown store, and the state of the finite control, a number of choices are possible. Each choice consists of a next state for the finite control and a (possibly empty) string of symbols to replace the top pushdown store symbol. After selecting a choice, the input head is advanced one symbol.

The second type of move (called an *ϵ-move*) is similar to the first, except that the input symbol is not used, and the input head is not advanced after the move. This type of move allows the pda to manipulate the pushdown store without reading input symbols.

Finally, we must define the language accepted by a pushdown automaton. There are two natural ways to do this. The first, which we have already seen, is to define the language accepted to be the set of all inputs for which some sequence of moves causes the pushdown automaton to empty its pushdown store. This language is referred to as the language accepted by empty store.

The second way of defining the language accepted is similar to the way a finite automaton accepts tapes. That is, we could designate some states as final states and define the accepted language as the set of all inputs for which some choice of moves causes the pushdown automaton to enter a final state.

As we shall see, the two definitions of acceptance are equivalent in the sense that if a set can be accepted by empty store by some pda, it can be accepted by final state by some other pda, and vice-versa.

Acceptance by final state is the more common notion, but it is easier to prove the basic theorem of pushdown automata by using acceptance by empty store. The basic theorem is that a language is accepted by a pushdown automaton if and only if it is a context-free language.

A *pushdown automaton* M is a system $(K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where

1. K is a finite set of *states.*
2. Σ is a finite alphabet called the *input alphabet.*
3. Γ is a finite alphabet, called the *pushdown alphabet.*
4. q_0 in K is the *initial state.*
5. Z_0 in Γ is a particular pushdown symbol called the *start symbol.* Z_0 initially appears on the pushdown store.
6. $F \subseteq K$ is the set of *final states.*
7. δ is a mapping from $K \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to finite subsets of $K \times \Gamma^*$.

We use lower-case letters near the front of the alphabet to denote input symbols and lower-case letters near the end of the alphabet to denote strings of input symbols. Capital letters usually denote pushdown symbols and Greek letters indicate strings of pushdown symbols.

The interpretation of

$$\delta(q, a, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \ldots, (p_m, \gamma_m)\}$$

where q and p_i, $1 \leqq i \leqq m$, are in K, a is in Σ, Z is in Γ, and γ_i is in Γ^*, $1 \leqq i \leqq m$, is that the pda in state q, with input symbol a and Z the top symbol on the pushdown store, can, for any i, enter state p_i, replace Z by γ_i, and advance the input head one symbol. We adopt the convention that the leftmost symbol of γ_i will be placed highest on the store and the rightmost symbol lowest on the store.†

The interpretation of

$$\delta(q, \epsilon, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \ldots, (p_m, \gamma_m)\}$$

is that the pda in state q, independent of the input symbol being scanned and with Z the top symbol on the pushdown store, can enter state p_i and replace Z by γ_i for any i, $1 \leqq i \leqq m$. In this case, the input head is not advanced.

Example 5.1. Figure 5.2 gives a formal pushdown automaton which accepts $\{wcw^R | w \text{ in } \{0, 1\}^*\}$ by empty store. Note that for a move in which the pda writes a symbol on the top of the store δ has a value (q, γ) where $|\gamma| = 2$. For example, $\delta(q_1, 0, R) = \{(q_1, BR)\}$. If γ were of length one, the pda would simply replace the top symbol by a new symbol and not increase the length of the pushdown store. This allows us to let γ equal ϵ for the case in which we wish to erase the top symbol, thereby shortening the pushdown store.

Note that the rule $\delta(q_2, \epsilon, R) = \{(q_2, \epsilon)\}$ means that the pda, in state q_2 with R the top pushdown symbol, can erase the R independent of the input symbol. In this case the input head is not advanced.

---

† This convention is opposite that used by some other writers. We prefer it since it simplifies notation in what follows.

$$M = (\{q_1, q_2\}, \{0, 1, c\}, \{R, B, G\}, \delta, q_1, R, \varphi)\dagger$$

$$\begin{aligned}
&\delta(q_1, 0, R) = \{(q_1, BR)\} \qquad \delta(q_1, 1, R) = \{(q_1, GR)\}\\
&\delta(q_1, 0, B) = \{(q_1, BB)\} \qquad \delta(q_1, 1, B) = \{(q_1, GB)\}\\
&\delta(q_1, 0, G) = \{(q_1, BG)\} \qquad \delta(q_1, 1, G) = \{(q_1, GG)\}\\
&\delta(q_1, c, R) = \{(q_2, R)\}\\
&\delta(q_1, c, B) = \{(q_2, B)\}\\
&\delta(q_1, c, G) = \{(q_2, G)\}\\
&\delta(q_2, 0, B) = \{(q_2, \epsilon)\} \qquad \delta(q_2, 1, G) = \{(q_2, \epsilon)\}\\
&\delta(q_2, \epsilon, R) = \{(q_2, \epsilon)\}
\end{aligned}$$

Fig. 5.2. Formal pushdown automaton accepting $\{wcw^R | w \text{ in } \{0, 1\}^*\}$ by empty tape.

A *configuration* of a pda is a pair (q, γ) where q is a state in K and γ is a string of pushdown symbols. We say that a pda M is in configuration (q, γ) if M is in state q with γ on the pushdown store, the leftmost symbol of γ being the top symbol on the pushdown store. If a is in $\Sigma \cup \{\epsilon\}$, γ and β are in Γ^*, and Z is in Γ, and further, if the pair (p, β) is in $\delta(q, a, Z)$, then we write

$$a{:}(q, Z\gamma) \vdash_M (p, \beta\gamma).$$

The above means that according to the rules of the pda the input a may cause M to go from configuration $(q, Z\gamma)$ to configuration $(p, \beta\gamma)$.

If for $a_1, a_2, \ldots, a_n$, each in $\Sigma \cup \{\epsilon\}$, states $q_1, q_2, \ldots, q_{n+1}$ and pushdown strings $\gamma_1, \gamma_2, \ldots, \gamma_{n+1}$ we have:

$$a_i{:}(q_i, \gamma_i) \vdash_M (q_{i+1}, \gamma_{i+1})$$

for all i between 1 and n, then we write

$$a_1 a_2 \ldots a_n{:}(q_1, \gamma_1) \vdash_M^* (q_{n+1}, \gamma_{n+1}).\ddagger$$

Recall that many of the a_i's may be ϵ. The subscript M will be dropped from $\vdash_M^*$ whenever the meaning remains clear.

For a pda M we define $T(M)$, the language *accepted by final state*, to be

$$\{w | w{:}(q_0, Z_0) \vdash_M^* (q, \gamma) \text{ for any } \gamma \text{ in } \Gamma^* \text{ and } q \text{ in } F\}.$$

Also, we define $N(M)$, the language *accepted by empty store*, to be

$$\{w | w{:}(q_0, Z_0) \vdash_M^* (q, \epsilon) \text{ for any } q \text{ in } K\}.$$

---

† φ denotes the empty set.

‡ By convention, we always have $\epsilon{:}(q, \gamma) \vdash_M^* (q, \gamma)$.

When accepting by empty store, the set of final states is irrelevant. Thus when accepting by empty store we usually let the set of final states be the empty set.

The pda of Example 5.1 is deterministic in the sense that at most one move is possible from any configuration. Formally, we say that a pda, $M = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, is *deterministic* if:

1. For each q in K and Z in Γ, whenever $\delta(q, \epsilon, Z)$ is nonempty, then $\delta(q, a, Z)$ is empty for all a in Σ.
2. For no q in K, Z in Γ, and a in $\Sigma \cup \{\epsilon\}$ does $\delta(q, a, Z)$ contain more than one element.

Condition 1 prevents the possibility of a choice between a move independent of the input symbol (ϵ-move) and a move involving an input symbol. Condition 2 prevents a choice of move for any (q, a, Z) or (q, ϵ, Z).

Example 5.2. Figure 5.3 gives a nondeterministic pda that accepts $\{ww^R | w \text{ in } \{0, 1\}^*\}$. Rules 1 through 6 allow M to store the input on the pushdown store. In Rules 3 and 6 M has a choice of moves. If M decides that the middle of the input string has been reached, then the second choice is selected. M goes to state q_2 and tries to match the remaining input symbols with the contents of the pushdown store. If M guessed right, and if the input is of the form ww^R, then the inputs will match, M will empty its pushdown store, and thus accept the input string.

$$M = (\{q_1, q_2\}, \{0, 1\}, \{R, B, G\}, \delta, q_1, R, \varphi)$$

1. $\delta(q_1, 0, R) = \{(q_1, BR)\}$
2. $\delta(q_1, 1, R) = \{(q_1, GR)\}$
3. $\delta(q_1, 0, B) = \{(q_1, BB), (q_2, \epsilon)\}$
4. $\delta(q_1, 0, G) = \{(q_1, BG)\}$
5. $\delta(q_1, 1, B) = \{(q_1, GB)\}$
6. $\delta(q_1, 1, G) = \{(q_1, GG), (q_2, \epsilon)\}$
7. $\delta(q_2, 0, B) = \{(q_2, \epsilon)\}$
8. $\delta(q_2, 1, G) = \{(q_2, \epsilon)\}$
9. $\delta(q_1, \epsilon, R) = \{(q_2, \epsilon)\}$
10. $\delta(q_2, \epsilon, R) = \{(q_2, \epsilon)\}$

Fig. 5.3. A nondeterministic pda that accepts $\{ww^R | w \text{ in } \{0, 1\}^*\}$ by empty store.

We cannot emphasize too strongly that M accepts an input if any sequence of choices causes M to empty its pushdown store. Thus M always "guesses right," because wrong guesses, in themselves, do not cause an input to be rejected. An input is only rejected if there is no "right guess." Figure 5.4 shows the accessible configurations of M when processing the string 001100.

For finite automata, the deterministic and nondeterministic models were equivalent with respect to the languages accepted. We shall see later that the same is not true for pda. In fact ww^R is accepted by a nondeterministic pda, but not by any deterministic pda.

Input	Configurations
ϵ	$(q_1, R) \rightarrow (q_2, \epsilon)$
0	(q_1, BR)
00	$(q_1, BBR)(q_2, R) \rightarrow (q_2, \epsilon)$
001	$(q_1, GBBR)$
0011	$(q_1, GGBBR)\ (q_2, BBR)$
00110	$(q_1, BGGBBR)\ (q_2, BR)$
001100	$(q_1, BBGGBBR)\ (q_2, GGBBR)\ (q_2, R) \rightarrow (q_2, \epsilon)$

Fig. 5.4. Accessible configurations for the pda of Fig. 5.3 with input 001100.

5.3 NONDETERMINISTIC PUSHDOWN AUTOMATA AND CONTEXT-FREE LANGUAGES

We shall now prove the fundamental result that the class of languages accepted by nondeterministic pda is precisely the class of context-free languages. We first show that the languages accepted by nondeterministic pushdown automata by final state are exactly the languages accepted by nondeterministic pushdown automata by empty store. We then show that the languages accepted by empty store are exactly the context-free languages.

Theorem 5.1. L is $N(M_1)$ for some pda M_1, if and only if L is $T(M_2)$ for some pda, M_2.

Proof (if). Let

$$M_2 = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

be a pda such that $L = T(M_2)$. Let

$$M_1 = (K \cup \{q_e, q_0'\}, \Sigma, \Gamma \cup \{X\}, \delta', q_0', X, \varphi)$$

where δ' is defined as follows.

1. $\delta'(q_0', \epsilon, X)$ contains (q_0, Z_0X).
2. $\delta'(q, a, Z)$ includes the elements of $\delta(q, a, Z)$ for all q in K, a in Σ or $a = \epsilon$, and Z in Γ.
3. For all q in F, and Z in $\Gamma \cup \{X\}$, $\delta'(q, \epsilon, Z)$ contains (q_e, ϵ).
4. For all Z in $\Gamma \cup \{X\}$, $\delta'(q_e, \epsilon, Z)$ contains (q_e, ϵ).

Rule 1 causes M_1 to enter the initial configuration of M_2, except that M_1 will have its own bottom of the stack marker, X, which is below the symbols of M_2's pushdown store. Rule 2 allows M_1 to simulate M_2. Should M_2 ever enter a final state, Rules 3 and 4 allow M_1 the choice of entering state q_e and erasing its store, thereby accepting the input, or continuing to simulate M_2.

One should note that M_2 may possibly erase its entire store for some input x not in $T(M_2)$. This is the reason that M_1 has its own special bottom of the stack marker. Otherwise M_1, in simulating M_2, would also erase its entire store, thereby accepting x when it should not.

Now assume that x is in $T(M_2)$. Then $x{:}(q_0, Z_0) \overset{*}{\underset{M_2}{\vdash}} (q, \gamma)$ for some q in F. Thus

$$\epsilon{:}(q_0', X) \underset{M_1}{\vdash} (q_0, Z_0X) \text{ by Rule 1,}$$

$$x{:}(q_0, Z_0X) \overset{*}{\underset{M_1}{\vdash}} (q, \gamma X) \text{ by Rule 2,}$$

$$\epsilon{:}(q, \gamma X) \overset{*}{\underset{M_1}{\vdash}} (q_e, \epsilon) \text{ by Rules 3 and 4,}$$

and therefore x is in $N(M_1)$. By similar reasoning, if x is in $N(M_1)$, then x is in $T(M_2)$.

Proof (only if). Let

$$M_1 = (K, \Sigma, \Gamma, \delta, q_0, Z_0, \varphi)$$

be a pda such that $L = N(M_1)$. Let

$$M_2 = (K \cup \{q_0', q_f\}, \Sigma, \Gamma \cup \{X\}, \delta', q_0', X, \{q_f\})$$

where δ' is defined as follows.

1. $\delta'(q_0', \epsilon, X)$ contains (q_0, Z_0X).
2. For all q in K, a in $\Sigma \cup \{\epsilon\}$, and Z in Γ, $\delta'(q, a, Z)$ includes the elements of $\delta(q, a, Z)$.
3. For all q in K, $\delta'(q, \epsilon, X)$ contains (q_f, ϵ).

Rule 1 causes M_2 to enter the initial configuration of M_1, except that M_2 will have its own bottom of stack marker X which is below the symbols of M_1's pushdown store. Rule 2 allows M_2 to simulate M_1. Should M_1 ever erase its entire pushdown store, then M_2, in simulating M_1, will erase its entire pushdown store except for the symbol X at the bottom. Rule 3 causes M_2, when the X appears, to enter a final state, thereby accepting the input x. The proof that $T(M_2) = N(M_1)$ is similar to the proof in the *if* part of the theorem and is left as an exercise.

Theorem 5.2. If L is a context-free language, then there exists a pda M, such that $L = N(M)$.

Proof. Let $G = (V_N, V_T, P, S)$ be a context-free grammar in Greibach normal form generating L. (We assume that ϵ is not in $L(G)$. The reader may modify the construction for the case where ϵ is in $L(G)$.) Let

$$M = (\{q_1\}, V_T, V_N, \delta, q_1, S, \varphi),$$

where $\delta(q_1, a, A)$ contains (q_1, γ) whenever $A \to a\gamma$ is in P.

To show that $L(G) = N(M)$, note that $xA\beta \underset{G}{\Rightarrow} xa\alpha\beta$ if and only if $a{:}(q_1, A\beta) \vdash_M (q_1, \alpha\beta)$. It follows immediately by induction on the number of steps of the derivation that $xA\beta \overset{*}{\underset{G}{\Rightarrow}} xy\alpha$, for any x and y in V_T^*, A in V_N, and α and β in V_N^*, if and only if $y{:}(q_1, A\beta) \overset{*}{\vdash}_M (q_1, \alpha)$. Thus $S \overset{*}{\underset{G}{\Rightarrow}} x$ if and only if $x{:}(q_1, S) \overset{*}{\vdash}_M (q_1, \epsilon)$.†

Theorem 5.3. If L is $N(M)$ for some pda M, then L is a context-free language.

Proof. Let M be the pda $(K, \Sigma, \Gamma, \delta, q_0, Z_0, \varphi)$. Let $G = (V_N, \Sigma, P, S)$ be a context-free grammar. V_N is the set of objects of the form $[q, A, p]$, where q and p are in K and A is in Γ, plus the new symbol S. P is the set of productions:

1. $S \to [q_0, Z_0, q]$ for each q in K.
2. $[q, A, p] \to a[q_1, B_1, q_2][q_2, B_2, q_3] \ldots [q_m, B_m, q_{m+1}]$ for each $q, q_1, q_2, \ldots, q_{m+1}$ in K, where $p = q_{m+1}$, each a in $\Sigma \cup \{\epsilon\}$, and $A, B_1, B_2, \ldots, B_m$ in Γ, such that $\delta(q, a, A)$ contains $(q_1, B_1B_2 \ldots B_m)$. (If $m = 0$, then $q_1 = p$, $\delta(q, a, A)$ contains (p, ϵ), and the production is $[q, A, p] \to a$.)

To understand the proof it helps to know that the variables and productions of G have been defined in such a way that a leftmost derivation in G of a sentence x is a simulation of the pda M, when fed the input x. In particular, the variables that appear in any step of a leftmost derivation in G correspond to the symbols on the pushdown store of the pda at a time when the pda has seen as much of the input as the grammar has already generated.

To show that $L(G) = N(M)$, we prove by induction on the number of steps in a derivation of G or number of moves of M, that

$$[q, A, p] \overset{*}{\underset{G}{\Rightarrow}} x \qquad \text{if and only if} \qquad x{:}(q, A) \overset{*}{\vdash}_M (p, \epsilon).$$

Now if x is in $L(G)$, then

$$S \underset{G}{\Rightarrow} [q_0, Z_0, q] \overset{*}{\underset{G}{\Rightarrow}} x,$$

for some state q. Hence, $x{:}(q_0, Z_0) \overset{*}{\vdash}_M (q, \epsilon)$, and therefore, x is in $N(M)$. Similarly x in $N(M)$ implies that $x{:}(q_0, Z_0) \overset{*}{\vdash}_M (q, \epsilon)$. Hence,

$$S \underset{G}{\Rightarrow} [q_0, Z_0, q] \overset{*}{\underset{G}{\Rightarrow}} x,$$

and therefore, x is in $L(G)$.

---

† Note that the pda M makes no ϵ-moves.

First we shall undertake the "if" part of the proof. Suppose that $x:(q, A) \overset{*}{\underset{M}{\vdash}} (p, \epsilon)$ by a process taking k steps. We wish to show that $[q, A, p] \overset{*}{\underset{G}{\Rightarrow}} x$. For $k = 1$, x is either a single symbol or ϵ. Thus $\delta(q, x, A)$ must contain (p, ϵ) and hence $[q, A, p] \to x$ is a production in P. Therefore $[q, A, p] \overset{*}{\underset{G}{\Rightarrow}} x$.

Now we assume that the hypothesis is true for any process of up to $k - 1$ steps and show that it is true for processes of k steps. The first step must be of the form

$$a:(q, A) \underset{M}{\vdash} (q_1, B_1B_2 \ldots B_l), \qquad l \geqq 1,$$

where a is ϵ or the first symbol of x. It must be that x can be written $x = ax_1x_2 \ldots x_l$, such that for each i between 1 and l,

$$x_i:(q_i, B_i) \overset{*}{\underset{M}{\vdash}} (q_{i+1}, \epsilon)$$

by a process of fewer than k steps, where $q_1, q_2, \ldots, q_{l+1}$ are in K, and $q_{l+1} = p$. Therefore, from the inductive hypothesis, $[q_i, B_i, q_{i+1}] \overset{*}{\underset{G}{\Rightarrow}} x_i$. But

$$[q, A, p] \to a[q_1, B_1, q_2][q_2, B_2, q_3] \ldots [q_l, B_l, q_{l+1}]$$

is a production of G, so

$$[q, A, p] \overset{*}{\underset{G}{\Rightarrow}} ax_1x_2 \ldots x_l = x.$$

The "only if" part of the proof follows in a manner similar to the "if" part, by induction on the length of a derivation, and will not be given.

Example 5.3. Let

$$M = (\{q_0, q_1\}, \{0, 1\}, \{X, Z_0\}, \delta, q_0, Z_0, \varphi)$$

where δ is given by:

$$\begin{aligned} \delta(q_0, 0, Z_0) &= \{(q_0, XZ_0)\} & \delta(q_1, 1, X) &= \{(q_1, \epsilon)\} \\ \delta(q_0, 0, X) &= \{(q_0, XX)\} & \delta(q_1, \epsilon, X) &= \{(q_1, \epsilon)\} \\ \delta(q_0, 1, X) &= \{(q_1, \epsilon)\} & \delta(q_1, \epsilon, Z_0) &= \{(q_1, \epsilon)\} \end{aligned}$$

To construct a cfg $G = (V_N, V_T, P, S)$ generating $N(M)$ let

$$V_N = \{S, [q_0, X, q_0], [q_0, X, q_1], [q_1, X, q_0], [q_1, X, q_1], [q_0, Z_0, q_0], [q_0, Z_0, q_1], [q_1, Z_0, q_0], [q_1, Z_0, q_1]\}$$

and $V_T = \{0, 1\}$. To construct the set of productions easily, we must realize that some variables may not appear in any derivation starting from the symbol S. Thus, we can save some effort if we start with the productions for S, then add productions only for those variables that appear on the right

of some production already in the set. The productions for S are

$$S \to [q_0, Z_0, q_0] \qquad S \to [q_0, Z_0, q_1].$$

Next we add productions for the variable $[q_0, Z_0, q_0]$. These are

$$[q_0, Z_0, q_0] \to 0[q_0, X, q_0][q_0, Z_0, q_0],$$
$$[q_0, Z_0, q_0] \to 0[q_0, X, q_1][q_1, Z_0, q_0].$$

These productions are required by

$$\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}.$$

Next, the productions for $[q_0, Z_0, q_1]$ are

$$[q_0, Z_0, q_1] \to 0[q_0, X, q_0][q_0, Z_0, q_1],$$
$$[q_0, Z_0, q_1] \to 0[q_0, X, q_1][q_1, Z_0, q_1].$$

These are also required by $\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$. The productions for the remaining variables and the relevant moves of the pda are:

1. $[q_0, X, q_0] \to 0[q_0, X, q_0][q_0, X, q_0]$
   $[q_0, X, q_0] \to 0[q_0, X, q_1][q_1, X, q_0]$
   $[q_0, X, q_1] \to 0[q_0, X, q_0][q_0, X, q_1]$
   $[q_0, X, q_1] \to 0[q_0, X, q_1][q_1, X, q_1]$
   since $\delta(q_0, 0, X) = \{(q_0, XX)\}$
2. $[q_0, X, q_1] \to 1$ since $\delta(q_0, 1, X) = \{(q_1, \epsilon)\}$
3. $[q_1, Z_0, q_1] \to \epsilon$ since $\delta(q_1, \epsilon, Z_0) = \{(q_1, \epsilon)\}$
4. $[q_1, X, q_1] \to \epsilon$ since $\delta(q_1, \epsilon, X) = \{(q_1, \epsilon)\}$
5. $[q_1, X, q_1] \to 1$ since $\delta(q_1, 1, X) = \{(q_1, \epsilon)\}$

It should be noted that there are no productions for the variables $[q_1, X, q_0]$ and $[q_1, Z_0, q_0]$. Thus no terminal string can be derived from either $[q_0, Z_0, q_0]$ or $[q_0, X, q_0]$. Deleting all productions involving one of these four variables on either the right or left, we end up with the following productions.

$$S \to [q_0, Z_0, q_1]$$
$$[q_0, Z_0, q_1] \to 0[q_0, X, q_1][q_1, Z_0, q_1]$$
$$[q_0, X, q_1] \to 0[q_0, X, q_1][q_1, X, q_1]$$
$$[q_1, Z_0, q_1] \to \epsilon$$
$$[q_0, X, q_1] \to 1$$
$$[q_1, X, q_1] \to \epsilon$$
$$[q_1, X, q_1] \to 1$$

We summarize Theorems 5.1, 5.2, and 5.3 as follows. The subsequent three statements are equivalent:

1. L is a context-free language.
2. $L = N(M_1)$ for some pda M_1.
3. $L = T(M_2)$ for some pda M_2.

PROBLEMS

5.1 Find pushdown automata accepting the following sets by final state.

a) $\{w | w$ in $\{0, 1\}^*$ and w consists of an equal number of 0's and 1's$\}$.

b) $\{a^n b^m | n \leqq m \leqq 2n\}$.

c) The set generated by the grammar

$$G = (\{S, A\}, \{a, b\}, \{S \to aAA, A \to bS, A \to aS, A \to a\}, S).$$

d) The set of well-formed FORTRAN arithmetic expressions. Assume that variable names may be of any length greater than or equal to one.

5.2 Give a grammar for the language which is $N(M)$ where

$$M = (\{q_0, q_1\}, \{0, 1\}, \{Z_0, X\}, \delta, q_0, Z_0, \varphi)$$

and δ is given by:

$$\begin{aligned} \delta(q_0, 1, Z_0) &= \{(q_0, XZ_0)\} & \delta(q_0, \epsilon, Z_0) &= \{(q_0, \epsilon)\} \\ \delta(q_0, 1, X) &= \{(q_0, XX)\} & \delta(q_1, 1, X) &= \{(q_1, \epsilon)\} \\ \delta(q_0, 0, X) &= \{(q_1, X)\} & \delta(q_1, 0, Z_0) &= \{(q_0, Z_0)\} \end{aligned}$$

5.3 Prove the "only if" portion of Theorem 5.3.

5.4 Let $L = N(M)$ for some pda. Show that $L = N(M_1)$ for some one state pda, M_1.

5.5 Let $L = T(M)$ for some pda. Show that $L = T(M_1)$ for some two-state pda, M_1. Under what conditions is $L = T(M_1)$ for some one-state pda, M_1?

5.6 Let $L = N(M)$ for some pda. Show that $L = N(M_1)$ for some pda, $M_1 = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where $\delta(q, \epsilon, Z) = \varphi$ for all q in K and Z in Γ.

5.7 Is the pda of Example 5.1 deterministic? Justify your answer.

5.8 In Example 5.3, why are there no productions for the variable $[q_1, X, q_0]$?

REFERENCES

The pushdown automaton appears as a formal construction in Oettinger [1961] and Schutzenberger [1963]. Its relation to context-free languages was shown independently in Chomsky [1962] and Evey [1963].

Various generalizations of pushdown automata have appeared in the literature. Devices with two or more pushdown tapes are equivalent to Turing machines. (See Chapter 6.) The pushdown transducer is a pushdown automaton which may output symbols at each move. It has been studied in Evey [1963], Fischer [1963], Ginsburg and Rose [1966], and Ginsburg and Greibach [1966b]. The two-way pushdown automaton is a device with a pushdown store, a finite control, and an input tape on which a head can move in either direction. These devices have been studied in Hartmanis, Lewis, and Stearns [1965], Aho, Hopcroft, and Ullman [1968], and Gray, Harrison, and Ibarra [1967].