BIBLIOGRAPHIC NOTES

The minimization of finite automata was first studied by Huffman [1954] and Moore [1956]. The closure properties of regular sets and decidability results for finite automata are from Rabin and Scott [1959].

The Exercises contain some of the many results concerning finite automata and regular sets. Algorithm 2.6 is from Hopcroft [1971]. Exercise 2.3.22 has been proved by Kosaraju [1970]. The derivative of a regular expression was defined by Brzozowski [1964].

There are many techniques to minimize incompletely specified finite automata (Exercise 2.3.27). Ginsburg [1962] and Prather [1969] consider this problem. Kameda and Weiner [1968] give a partial solution to Exercise 2.3.32.

The books by Gill [1962], Ginsburg [1962], Harrison [1965], Minsky [1967], Booth [1967], Ginzburg [1968], Arbib [1969], and Salomaa [1969a] cover finite automata in detail.

Thompson [1968] outlines a useful programming technique for constructing a recognizer from a regular expression.

2.4. CONTEXT-FREE LANGUAGES

Of the four classes of grammars in the Chomsky hierarchy, the context-free grammars are the most important in terms of application to programming languages and compiling. A context-free grammar can be used to specify most of the syntactic structure of a programming language. In addition, a context-free grammar can be used as the basis of various schemes for specifying translations.

During the compiling process itself, we can use the syntactic structure imparted to an input program by a context-free grammar to help produce the translation for the input. The syntactic structure of an input sentence can be determined from the sequence of productions used to derive that input string. Thus in a compiler the syntactic analyzer can be viewed as a device which attempts to determine if there is a derivation of the input string according to some context-free grammar. However, given a CFG G and an input string w, it is a nontrivial task to determine whether w is in $L(G)$ and, if so, what is a derivation for w in G. We shall treat this question in detail in Chapters 4–7.

In this section we shall build the foundation on which we shall base our study of parsing. In particular, we shall define derivation trees and study some transformations which can be applied to context-free grammars to make their representation more convenient.

2.4.1. Derivation Trees

In a grammar it is possible to have several derivations that are equivalent, in the sense that all derivations use the same productions at the same places, but in a different order. The definition of when two derivations are equivalent is a complex matter for unrestricted grammars (see the Exercises for Section 2.2), but for context-free grammars we can define a convenient graphical representative of an equivalence class of derivations called a derivation tree.

A derivation tree for a context-free grammar $G = (N, \Sigma, P, S)$ is a labeled ordered tree in which each node is labeled by a symbol from $N \cup \Sigma \cup \{e\}$. If an interior node is labeled A and its direct descendants are labeled $X_1, X_2, \ldots, X_n$, then $A \longrightarrow X_1X_2 \cdots X_n$ is a production in P.

DEFINITION

A labeled ordered tree D is a *derivation tree* (or *parse tree*) for a context-free grammar $G(A) = (N, \Sigma, P, A)$ if

(1) The root of D is labeled A.

(2) If $D_1, \ldots, D_k$ are the subtrees of the direct descendants of the root and the root of D_i is labeled X_i, then $A \longrightarrow X_1 \cdots X_k$ is a production in P. D_i must be a derivation tree for $G(X_i) = (N, \Sigma, P, X_i)$ if X_i is a nonterminal, and D_i is a single node labeled X_i if X_i is a terminal.

(3) Alternatively, if D_1 is the only subtree of the root of D and the root of D_1 is labeled e, then $A \longrightarrow e$ is a production in P.

Example 2.19

The trees in Fig. 2.8 are derivation trees for the grammar $G = G(S)$ defined by $S \longrightarrow aSbS \mid bSaS \mid e$. □

We note that there is a natural ordering on the nodes of an ordered tree. That is, the direct descendants of a node are ordered "from the left" as defined

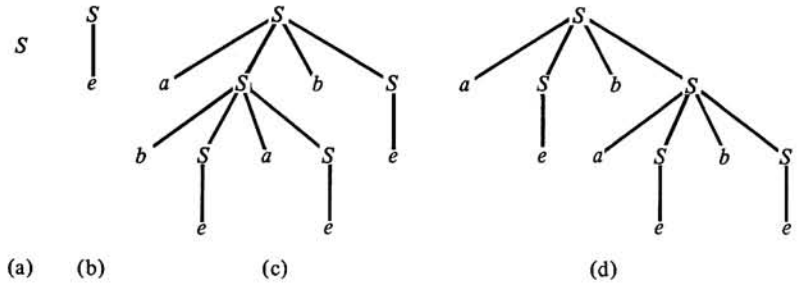


Fig. 2.8 Derivation trees.

in Section 0.5.4. We extend the from-the-left ordering as follows. Suppose that n is a node and $n_1, \ldots, n_k$ are its direct descendants. Then if $i < j$, n_i and all its descendants are to the left of n_j and all its descendants. It is left for the Exercises to show that this ordering is consistent. All that needs to be shown is that given any two nodes of an ordered tree, they are either on a path or one is to the left of the other.

DEFINITION

The *frontier* of a derivation tree is the string obtained by concatenating the labels of the leaves (in order from the left). For example, the frontiers of the derivation trees in Fig. 2.8 are (a) S, (b) e, (c) *abab*, and (d) *abab*.

We shall now show that a derivation tree is an adequate representation for derivations by showing that for every derivation of a sentential form α in a CFG G there is a derivation tree of G with frontier α, and conversely. To do so we introduce a few more terms. Let D be a derivation tree for a CFG $G = (N, \Sigma, P, S)$.

DEFINITION

A *cut* of D is a set C of nodes of D such that

(1) No two nodes in C are on the same path in D, and
(2) No other node of D can be added to C without violating (1).

Example 2.20

The set of nodes consisting of only the root is a cut. Another cut is the set of leaves. The set of circled nodes in Fig. 2.9 is a cut. □

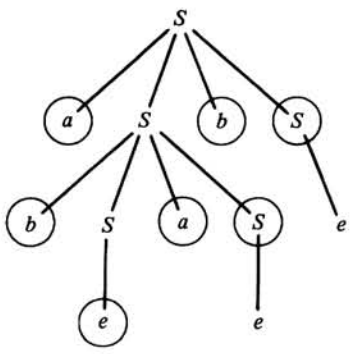


Fig. 2.9 Example of a cut.

DEFINITION

Let us define an *interior frontier* of D as the string obtained by concatenating (in order from the left) the labels of the nodes of a cut of D. For example, *abaSbS* is an interior frontier of the derivation tree shown in Fig. 2.9.

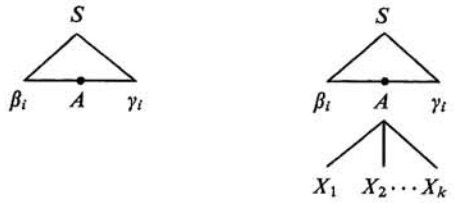
LEMMA 2.12

Let $S = \alpha_0, \alpha_1, \ldots, \alpha_n$ be a derivation of α_n from S in CFG $G = (N, \Sigma, P, S)$. Then there is a derivation tree D for G such that D has frontier α_n and interior frontiers $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ (among others).

Proof. We shall construct a sequence of derivation trees D_i, $0 \leq i \leq n$, such that the frontier of D_i is α_i.

Let D_0 be the derivation tree consisting of the single node labeled S.

Suppose that $\alpha_i = \beta_i A \gamma_i$ and this instance of A is rewritten to obtain $\alpha_{i+1} = \beta_i X_1 X_2 \cdots X_k \gamma_i$. Then the derivation tree D_{i+1} is obtained from D_i by adding k direct descendants to the leaf labeled with this instance of A (i.e., the node which contributes the $|\beta_i| + 1$st symbol to the frontier of D_i) and labeling these direct descendants $X_1, X_2, \ldots, X_k$ respectively. It should be evident that the frontier of D_{i+1} is α_{i+1}. The construction of D_{i+1} from D_i is shown in Fig. 2.10.



(a) D_i (b) D_{i+1}

Fig. 2.10 Alteration of Trees

D_n will then be the desired derivation tree D. □

We will now obtain the converse of Lemma 2.12. That is, for every derivation tree for G there is at least one derivation in G.

LEMMA 2.13

Let D be a derivation tree for a CFG $G = (N, \Sigma, P, S)$ with frontier α. Then $S \overset{*}{\Rightarrow} \alpha$.

Proof. Let $C_0, C_1, C_2, \ldots, C_n$ be any sequence of cuts of D such that

(1) C_0 contains only the root of D.

(2) C_{i+1} is obtained from C_i by replacing one interior node in C_i by its direct descendants, for $0 \leq i < n$.

(3) C_n is the set of leaves of D.

Clearly at least one such sequence exists.

If α_i is the interior frontier associated with C_i, then $\alpha_0, \alpha_1, \ldots, \alpha_n$ is a derivation of α_n from α_0 in G. □

There are two derivations that can be constructed from a derivation tree which will be of particular interest to us.

DEFINITION

In the proof of Lemma 2.13, if C_{i+1} is obtained from C_i by replacing the leftmost nonleaf in C_i by its direct descendants, then the associated derivation $\alpha_0, \alpha_1, \ldots, \alpha_n$ is called a *leftmost* derivation of α_n from α_0 in G. We define a *rightmost* derivation analogously by replacing "leftmost" by "rightmost" above. Notice that the leftmost (or rightmost) derivation associated with a derivation tree is unique.

If $S = \alpha_0, \alpha_1, \ldots, \alpha_n = w$ is a leftmost derivation of the terminal string w, then each α_i, $0 \leq i < n$, is of the form $x_i A_i \beta_i$ with $x_i \in \Sigma^*$, $A_i \in \mathrm{N}$, and $\beta_i \in (\mathrm{N} \cup \Sigma)^*$. The leftmost nonterminal A_i is rewritten to obtain each succeeding sentential form. The reverse situation holds for rightmost derivations.

Example 2.21

Let G_0 be the CFG

$$E \longrightarrow E + T \mid T$$
$$T \longrightarrow T * F \mid F$$
$$F \longrightarrow (E) \mid a$$

The derivation tree shown in Fig. 2.11 represents ten equivalent derivations

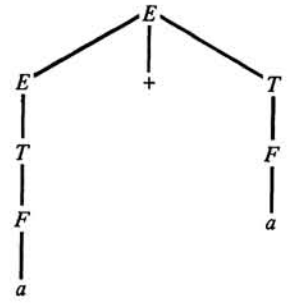


Fig. 2.11 Example of a tree.

of the sentence $a + a$. The leftmost derivation is

$$E \Longrightarrow E + T \Longrightarrow T + T \Longrightarrow F + T \Longrightarrow a + T \Longrightarrow a + F \Longrightarrow a + a$$

and the rightmost derivation is

$$E \Longrightarrow E + T \Longrightarrow E + F \Longrightarrow E + a \Longrightarrow T + a \Longrightarrow F + a \Longrightarrow a + a$$

□

DEFINITION

If $S = \alpha_0, \alpha_1, \ldots, \alpha_n$ is a leftmost derivation in grammar G, then we shall write $S \overset{*}{\underset{G\,\mathrm{lm}}{\Longrightarrow}} \alpha_n$ or $S \overset{*}{\underset{\mathrm{lm}}{\Longrightarrow}} \alpha_n$, if G is clear, to indicate the leftmost derivation. We call α_n a *left sentential form*. Likewise, if $S = \alpha_0, \alpha_1, \ldots, \alpha_n$ is a rightmost derivation, we shall write $S \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} \alpha_n$, and call α_n a *right sentential form*. We use $\underset{\mathrm{lm}}{\Longrightarrow}$ and $\underset{\mathrm{rm}}{\Longrightarrow}$ to indicate single-step leftmost and rightmost derivations.

We can combine Lemmas 2.12 and 2.13 into the following theorem.

THEOREM 2.11

Let $G = (\mathrm{N}, \Sigma, P, S)$ be a CFG. Then $S \overset{*}{\Longrightarrow} \alpha$ if and only if there is a derivation tree for G with frontier α.

Proof. Immediate from Lemmas 2.12 and 2.13. □

Notice that we have been careful not to say that given a derivation $S \overset{*}{\Longrightarrow} \alpha$ in a CFG G we can find a unique derivation tree for G with frontier α. The reason for this is that there are context-free grammars which have several distinct derivation trees with the same frontier. The grammar in Example 2.19 is an example of such a grammar. Derivation trees (c) and (d) (Fig. 2.8) in that example have equal frontiers but are not the same trees.

DEFINITION

We say that a CFG G is *ambiguous* if there is at least one sentence w in $L(G)$ for which there is more than one distinct derivation tree with frontier w. This is equivalent to saying that G is ambiguous if there is a sentence w in $L(G)$ with two or more distinct leftmost (or rightmost) derivations (Exercise 2.4.4).

We shall consider ambiguity in more detail in Section 2.6.5.

2.4.2. Transformations on Context-Free Grammars

Given a grammar it is often desirable to modify the grammar so that a certain structure is imposed on the language generated. For example, let us consider $L(G_0)$. This language can be generated by the grammar G with productions

$$E \longrightarrow E + E \mid E * E \mid (E) \mid a$$

But there are two features of G which are not desirable. First of all, G is ambiguous because of the productions $E \longrightarrow E + E \mid E * E$. This ambiguity can be removed by using the grammar G_1 with productions

$$E \longrightarrow E + T \mid E * T \mid T$$
$$T \longrightarrow (E) \mid a$$

The other drawback to G, which is shared by G_1, is that the operators $+$ and $*$ have the same precedence. That is to say, in the expressions $a + a * a$ and $a * a + a$, the operators would associate from the left as in $(a + a) * a$ and $(a * a) + a$, respectively.

In going to the grammar G_0 we can obtain the conventional precedence of $+$ and $*$.

In general, there is no algorithmic method to impose an arbitrary structure on a given language. However, there are a number of useful transformations which can be used to modify a grammar without disturbing the language generated. In this section and in Sections 2.4.3–2.4.5 we shall consider a number of transformations of this nature.

We shall begin by considering some very obvious but important transformations. In certain situations, a CFG may contain useless symbols and productions. For example, consider the grammar $G = (\{S, A\}, \{a, b\}, P, S)$, where $P = \{S \to a, A \to b\}$. In G, the nonterminal A and the terminal b cannot appear in any sentential form. Thus these two symbols are irrelevant insofar as $L(G)$ is concerned and can be removed from the specification of G without affecting $L(G)$.

DEFINITION

We say that a symbol $X \in \mathrm{N} \cup \Sigma$ is *useless* in a CFG $G = (\mathrm{N}, \Sigma, P, S)$ if there does not exist a derivation of the form $S \overset{*}{\Rightarrow} wXy \overset{*}{\Rightarrow} wxy$. Note that w, x, and y are in Σ^*.

To determine whether a nonterminal A is useless, we first provide an algorithm to determine whether a nonterminal can generate any terminal strings; i.e., is $\{w \mid A \overset{*}{\Rightarrow} w, w \text{ in } \Sigma^*\} = \varnothing$? The existence of such an algorithm implies that the emptiness problem is solvable for context-free grammars.

ALGORITHM 2.7

Is $L(G)$ nonempty?

Input. CFG $G = (\mathrm{N}, \Sigma, P, S)$.

Output. "YES" if $L(G) \neq \varnothing$, "NO" otherwise.

Method. We construct sets $\mathrm{N}_0, \mathrm{N}_1, \ldots$ recursively as follows:

(1) Let $\mathrm{N}_0 = \varnothing$ and set $i = 1$.

(2) Let $\mathrm{N}_i = \{A \mid A \to \alpha \text{ is in } P \text{ and } \alpha \in (\mathrm{N}_{i-1} \cup \Sigma)^*\} \cup \mathrm{N}_{i-1}$.

(3) If $\mathrm{N}_i \neq \mathrm{N}_{i-1}$, then set $i = i + 1$ and go to step 2. Otherwise, let $\mathrm{N}_e = \mathrm{N}_i$.

(4) If S is in N_e, output "YES"; otherwise, output "NO." □

Since $\mathrm{N}_e \subseteq \mathrm{N}$, Algorithm 2.7 must terminate after at most $n + 1$ iterations of step (2) if N has n members. We shall prove the correctness of Algorithm 2.7. The proof is simple and will serve as a model for several similar proofs.

THEOREM 2.12

Algorithm 2.7 says "YES" if and only if $S \overset{*}{\Longrightarrow} w$ for some w in Σ^*.

Proof. We first prove the following statement by induction on i:

(2.4.1) If A is in N_i, then $A \overset{*}{\Longrightarrow} w$ for some w in Σ^*

The basis, $i = 0$, holds vacuously, since $N_0 = \varnothing$. Assume that (2.4.1) is true for i, and let A be in N_{i+1}. If A is also in N_i, the inductive step is trivial. If A is in $N_{i+1} - N_i$, then there is a production $A \longrightarrow X_1 \cdots X_k$, where each X_j is either in Σ or a nonterminal in N_i. Thus we can find a string w_j such that $X_j \overset{*}{\Longrightarrow} w_j$ for each j. If X_j is in Σ, $w_j = X_j$, and otherwise the existence of w_j follows from (2.4.1). It is simple to see that

$$A \Longrightarrow X_1 \cdots X_k \overset{*}{\Longrightarrow} w_1 X_2 \cdots X_k \overset{*}{\Longrightarrow} \cdots \overset{*}{\Longrightarrow} w_1 \cdots w_k.$$

The case $k = 0$ (i.e., production $A \longrightarrow e$) is not ruled out. The inductive step is complete.

The definition of N_i assures us that if $N_i = N_{i-1}$, then $N_i = N_{i+1} = \cdots$. We must show that if $A \overset{*}{\Longrightarrow} w$ for some $w \in \Sigma^*$, then A is in N_e. By the above comment, all we need to show is that A is in N_i for some i. We show the following by induction on n:

(2.4.2) If $A \overset{n}{\Longrightarrow} w$, then A is in N_i for some i

The basis, $n = 1$, is trivial; $i = 1$ in this case. Assume that (2.4.2) is true for n, and let $A \overset{n+1}{\Longrightarrow} w$. Then we can write $A \Longrightarrow X_1 \cdots X_k \overset{n}{\Longrightarrow} w$, where $w = w_1 \cdots w_k$ such that $X_j \overset{n_j}{\Longrightarrow} w_j$ for each j, where $n_j \leq n$.†

By (2.4.2), if X_j is in N, then X_j is in N_{i_j} for some i_j. If X_j is in Σ, let $i_j = 0$. Let $i = 1 + \max(i_1, \ldots, i_k)$. Then by definition, A is in N_i. The induction is complete. Letting $A = S$ in (2.4.1) and (2.4.2), we have the theorem. □

COROLLARY

It is decidable, for CFG G, if $L(G) = \varnothing$. □

DEFINITION

We say that a symbol X in $N \cup \Sigma$ is *inaccessible* in a CFG $G = (N, \Sigma, P, S)$ if X does not appear in any sentential form.

†This is an "obvious" comment that requires a little thought. Think about the derivation tree for the derivation $A \overset{n+1}{\Longrightarrow} w$. w_j is the frontier of the subtree with root X_j.

The following algorithm, which is an adaptation of Algorithm 0.3, can be used to remove inaccessible symbols from a CFG.

ALGORITHM 2.8

Removal of inaccessible symbols.

Input. CFG $G = (N, \Sigma, P, S)$.

Output. CFG $G' = (N', \Sigma', P', S)$ such that

(i) $L(G') = L(G)$.

(ii) For all X in $N' \cup \Sigma'$ there exist α and β in $(N' \cup \Sigma')^*$ such that $S \overset{*}{\underset{G'}{\Rightarrow}} \alpha X \beta$.

Method.

(1) Let $V_0 = \{S\}$ and set $i = 1$.

(2) Let $V_i = \{X \mid \text{some } A \longrightarrow \alpha X \beta \text{ is in } P \text{ and } A \text{ is in } V_{i-1}\} \cup V_{i-1}$.

(3) If $V_i \neq V_{i-1}$, set $i = i + 1$ and go to step (2). Otherwise, let

$N' = V_i \cap N$
$\Sigma' = V_i \cap \Sigma$
P' be those productions in P which involve only symbols in V_i
$G' = (N', \Sigma', P', S)$ □

There is a great deal of similarity between Algorithms 2.7 and 2.8. Note that in Algorithm 2.8, since $V_i \subseteq N \cup \Sigma$, step (2) of the algorithm can be repeated at most a finite number of times. Moreover, a straightforward proof by induction on i shows that $S \overset{*}{\underset{G}{\Rightarrow}} \alpha X \beta$ if and only if X is in V_i for some i.

We are now in a position to remove all useless symbols from a CFG.

ALGORITHM 2.9

Useless symbol removal.

Input. CFG $G = (N, \Sigma, P, S)$, such that $L(G) \neq \varnothing$.

Output. CFG $G' = (N', \Sigma', P', S)$ such that $L(G') = L(G)$ and no symbol in $N' \cup \Sigma'$ is useless.

Method.

(1) Apply Algorithm 2.7 to G to obtain N_e. Let $G_1 = (N \cap N_e, \Sigma, P_1, S)$, where P_1 contains those productions of P involving only symbols in $N_e \cup \Sigma$.

(2) Apply Algorithm 2.8 to G_1 to obtain $G' = (N', \Sigma', P', S)$. □

Step (1) of Algorithm 2.9 removes from G all nonterminals which cannot generate a terminal string. Step (2) then proceeds to remove all symbols which are not accessible. Each symbol X in the resulting grammar must

appear in at least one derivation of the form $S \overset{*}{\Rightarrow} wXy \overset{*}{\Rightarrow} wxy$. Note that applying Algorithm 2.8 first and then applying Algorithm 2.7 will not always result in a grammar with no useless symbols.

THEOREM 2.13

G' of Algorithm 2.9 has no useless symbols, and $L(G') = L(G)$.

Proof. We leave it for the Exercises to show that $L(G') = L(G)$. Suppose that $A \in \mathrm{N}'$ is useless. From the definition of useless, there are two cases to consider.

Case 1: $S \overset{*}{\underset{G'}{\Rightarrow}} \alpha A \beta$ is false for all α and β. In this case, A would have been removed in step (2) of Algorithm 2.9.

Case 2: $S \overset{*}{\underset{G'}{\Rightarrow}} \alpha A \beta$ for some α and β, but $A \overset{*}{\underset{G'}{\Rightarrow}} w$ is false for all w in Σ'^*. Then A is not removed in step (2), and, moreover, if $A \overset{*}{\underset{G}{\Rightarrow}} \gamma B \delta$, then B is not removed in step (2). Thus, if $A \overset{*}{\underset{G}{\Rightarrow}} w$, it would follow that $A \overset{*}{\underset{G'}{\Rightarrow}} w$. We conclude that $A \overset{*}{\underset{G}{\Rightarrow}} w$ is also false for all w, and A is eliminated in step (1).

The proof that no terminal of G' is useless is handled similarly and is left for the Exercises. □

Example 2.22

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

$$S \longrightarrow a \mid A$$
$$A \longrightarrow AB$$
$$B \longrightarrow b$$

Let us apply Algorithm 2.9 to G. In step (1), $\mathrm{N}_e = \{S, B\}$ so that $G_1 = (\{S, B\}, \{a, b\}, \{S \longrightarrow a, B \longrightarrow b\}, S)$. Applying Algorithm 2.8, we have $V_2 = V_1 = \{S, a\}$. Thus, $G' = (\{S\}, \{a\}, \{S \longrightarrow a\}, S)$.

If we apply Algorithm 2.8 first to G, we find that all symbols are accessible, so the grammar does not change. Then applying Algorithm 2.7 gives $\mathrm{N}_e = \{S, B\}$, so the resulting grammar is G_1 above, not G'. □

It is often convenient to eliminate e-productions, that is, productions of the form $A \longrightarrow e$, from a CFG G. However, if e is in $L(G)$, then clearly it is impossible to have no productions of the form $A \longrightarrow e$.

DEFINITION

We say that a CFG $G = (\mathrm{N}, \Sigma, P, S)$ is *e-free* if either

(1) P has no e-productions, or

(2) There is exactly one e-production $S \longrightarrow e$ and S does not appear on the right side of any production in P.

ALGORITHM 2.10

Conversion to an e-free grammar.

Input. CFG $G = (N, \Sigma, P, S)$.

Output. Equivalent e-free CFG $G' = (N', \Sigma, P', S')$.

Method.

(1) Construct $N_e = \{A \mid A \in N \text{ and } A \overset{+}{\underset{G}{\Longrightarrow}} e\}$. The algorithm is similar to that used in Algorithms 2.7 and 2.8 and is left for the Exercises.

(2) Let P' be the set of productions constructed as follows:

(a) If $A \longrightarrow \alpha_0 B_1 \alpha_1 B_2 \alpha_2 \cdots B_k \alpha_k$ is in P, $k \geq 0$, and for $1 \leq i \leq k$ each B_i is in N_e but no symbols in any α_j are in N_e, $0 \leq j \leq k$, then add to P' all productions of the form

$$A \longrightarrow \alpha_0 X_1 \alpha_1 X_2 \alpha_2 \cdots X_k \alpha_k$$

where X_i is either B_i or e, without adding $A \longrightarrow e$ to P'. (This could occur if all $\alpha_i = e$.)

(b) If S is in N_e, add to P' the productions

$$S' \longrightarrow e \mid S$$

where S' is a new symbol, and let $N' = N \cup \{S'\}$. Otherwise, let $N' = N$ and $S' = S$.

(3) Let $G' = (N', \Sigma, P', S')$. □

Example 2.23

Consider the grammar of Example 2.19 with productions

$$S \longrightarrow aSbS \mid bSaS \mid e$$

Applying Algorithm 2.10 to this grammar, we would obtain the grammar with the following productions:

$$S' \longrightarrow S \mid e$$
$$S \longrightarrow aSbS \mid bSaS \mid aSb \mid abS \mid ab \mid bSa \mid baS \mid ba$$ □

THEOREM 2.14

Algorithm 2.10 produces an e-free grammar equivalent to its input grammar.

Proof. By inspection, G' of Algorithm 2.10 is e-free. To prove that

$L(G) = L(G')$, we can prove the following statement by induction on the length of w:

(2.4.3) $A \overset{*}{\underset{G'}{\Longrightarrow}} w$ if and only if $w \neq e$ and $A \overset{*}{\underset{G}{\Longrightarrow}} w$

The proof of (2.4.3) is left for the Exercises. Substituting S for A in (2.4.3), we see that for $w \neq e$, $w \in L(G)$ if and only if $w \in L(G')$. The fact that $e \in L(G)$ if and only if $e \in L(G')$ is evident. Thus, $L(G) = L(G')$. □

Another transformation on grammars which we find useful is the removal of productions of the form $A \longrightarrow B$, which we shall call *single productions.*

ALGORITHM 2.11

Removal of single productions.

Input. An e-free CFG G.

Output. An equivalent e-free CFG G' with no single productions.

Method.

(1) Construct for each A in N the set $\mathrm{N}_A = \{B \mid A \overset{*}{\Longrightarrow} B\}$ as follows:
   (a) Let $\mathrm{N}_0 = \{A\}$ and set $i = 1$.
   (b) Let $\mathrm{N}_i = \{C \mid B \longrightarrow C \text{ is in } P \text{ and } B \in \mathrm{N}_{i-1}\} \cup \mathrm{N}_{i-1}$.
   (c) If $\mathrm{N}_i \neq \mathrm{N}_{i-1}$, set $i = i + 1$ and repeat step (b). Otherwise, let $\mathrm{N}_A = \mathrm{N}_i$.

(2) Construct P' as follows: If $B \longrightarrow \alpha$ is in P and not a single production, place $A \longrightarrow \alpha$ in P' for all A such that $B \in \mathrm{N}_A$.

(3) Let $G' = (\mathrm{N}, \Sigma, P', S)$. □

Example 2.24

Let us apply Algorithm 2.11 to the grammar G_0 with productions

$$E \longrightarrow E + T \mid T$$
$$T \longrightarrow T * F \mid F$$
$$F \longrightarrow (E) \mid a$$

In step (1), $\mathrm{N}_E = \{E, T, F\}$, $\mathrm{N}_T = \{T, F\}$, $\mathrm{N}_F = \{F\}$. After step (2), P' becomes

$$E \longrightarrow E + T \mid T * F \mid (E) \mid a$$
$$T \longrightarrow T * F \mid (E) \mid a$$
$$F \longrightarrow (E) \mid a \quad \square$$

THEOREM 2.15

In Algorithm 2.11, G' has no single productions, and $L(G) = L(G')$.

Proof. By inspection, G' has no single productions. We shall first show that $L(G') \subseteq L(G)$. Let w be in $L(G')$. Then there exists in G' a derivation $S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_n = w$. If the production applied going from α_i to α_{i+1} is $A \to \beta$, then there is some B in N (possibly, $A = B$) such that $A \overset{*}{\underset{G}{\Rightarrow}} B$ and $B \underset{G}{\Rightarrow} \beta$. Thus, $A \overset{*}{\underset{G}{\Rightarrow}} \beta$ and $\alpha_i \overset{*}{\underset{G}{\Rightarrow}} \alpha_{i+1}$. It follows that $S \overset{*}{\underset{G}{\Rightarrow}} w$, and w is in $L(G)$. Thus, $L(G') \subseteq L(G)$.

To show that $L(G') = L(G)$, we must show that $L(G) \subseteq L(G')$. Thus let w be in $L(G)$ and $S = \alpha_0 \underset{\text{lm}}{\Rightarrow} \alpha_1 \underset{\text{lm}}{\Rightarrow} \cdots \underset{\text{lm}}{\Rightarrow} \alpha_n = w$ be a leftmost derivation of w in G. We can find a sequence of subscripts $i_1, i_2, \ldots, i_k$ consisting of exactly those j such that $\alpha_{j-1} \underset{\text{lm}}{\Rightarrow} \alpha_j$ by an application of a production other than a single production. In particular, since the derivation of a terminal string cannot end with a single production, $i_k = n$.

Since the derivation is leftmost, consecutive uses of single productions replace the symbol at the same position in the left sentential forms involved. Thus we see that $S \underset{G'}{\Rightarrow} \alpha_{i_1} \underset{G'}{\Rightarrow} \alpha_{i_2} \underset{G'}{\Rightarrow} \cdots \underset{G'}{\Rightarrow} \alpha_{i_k} = w$. Thus, w is in $L(G')$. We conclude that $L(G') = L(G)$. □

DEFINITION

A CFG $G = (\text{N}, \Sigma, P, S)$ is said to be *cycle-free* if there is no derivation of the form $A \overset{+}{\Rightarrow} A$ for any A in N. G is said to be *proper* if it is cycle-free, is e-free, and has no useless symbols.

Grammars which have cycles or e-productions are sometimes more difficult to parse than grammars which are cycle-free and e-free. In addition, in any practical situation useless symbols increase the size of a parser unnecessarily. Throughout this book we shall assume a grammar has no useless symbols. For some of the parsing algorithms to be discussed in this book we shall insist that the grammar at hand be proper. The following theorem shows that this requirement still allows us to consider all context-free languages.

THEOREM 2.16

If L is a CFL, then $L = L(G)$ for some proper CFG G.

Proof. Use Algorithms 2.8–2.11. □

DEFINITION

An *A-production* in a CFG is a production of the form $A \to \alpha$ for some α. (Do not confuse an "A-production" with an "e-production," which is one of the form $B \to e$.)

Next we introduce a transformation which can be used to eliminate from a grammar a production of the form $A \to \alpha B \beta$. To eliminate this production we must add to the grammar a set of new productions formed by replacing the nonterminal B by all right sides of B-productions.

LEMMA 2.14

Let $G = (N, \Sigma, P, S)$ be a CFG and $A \longrightarrow \alpha B \beta$ be in P for some $B \in N$ and α and β in $(N \cup \Sigma)^*$. Let $B \longrightarrow \gamma_1 | \gamma_2 | \cdots | \gamma_k$ be all the B-productions in P. Let $G' = (N, \Sigma, P', S)$ where

$$P' = P - \{A \longrightarrow \alpha B \beta\} \cup \{A \longrightarrow \alpha\gamma_1\beta | \alpha\gamma_2\beta | \cdots | \alpha\gamma_k\beta\}.$$

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

Proof. Exercise. □

Example 2.25

Let us replace the production $A \longrightarrow aAA$ in the grammar G having the two productions $A \longrightarrow aAA | b$. Applying Lemma 2.14, assuming that $\alpha = a$, $B = A$, and $\beta = A$, we would obtain G' having productions

$$A \longrightarrow aaAAA | abA | b.$$

Derivation trees corresponding to the derivations of *aabbb* in G and G' are shown in Fig. 2.12(a) and (b). Note that the effect of the transformation is to "merge" the root of the tree in Fig. 2.12(a) with its second direct descendant. □

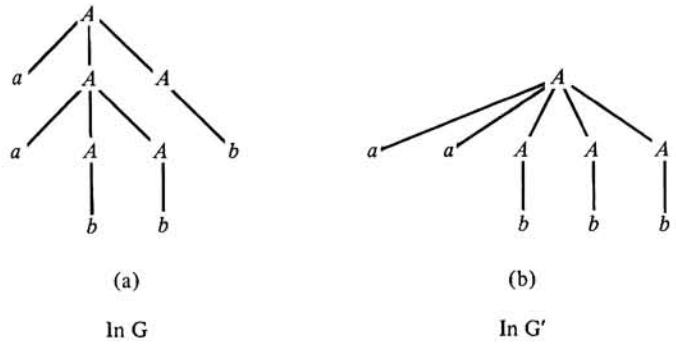


Fig. 2.12 Derivation trees in G and G'.

2.4.3. Chomsky Normal Form

DEFINITION

A CFG $G = (N, \Sigma, P, S)$ is said to be in *Chomsky normal form* (CNF) if each production in P is of one of the forms

(1) $A \longrightarrow BC$ with A, B, and C in N, or
(2) $A \longrightarrow a$ with $a \in \Sigma$, or