All these ten derivations are similar, because, informally, they represent applications of the same rules at the same positions in the strings, only differing in the relative order of these applications; equivalently, one can go from any one of them to any other by repeatedly following either a $\prec$, or an inverted $\prec$. There are no other derivations similar to these.

There are, however, other derivations of $(())()$ that are not similar to the ones above —and thus are not captured by the parse tree shown in Figure 3-4. An example is the following derivation: $S \Rightarrow SS \Rightarrow SSS \Rightarrow S(S)S \Rightarrow S((S))S \Rightarrow S(())S \Rightarrow S(())(S) \Rightarrow S(())() \Rightarrow (())()$. Its parse tree is shown in Figure 3-6 (compare with Figure 3-4).◇

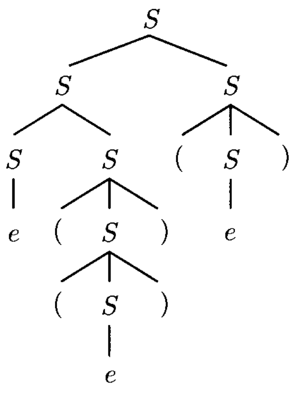

Figure 3-6

Each equivalence class of derivations under similarity, that is to say, each parse tree, contains a derivation that is *maximal* under $\prec$; that is, it is not preceded by any other derivation. This derivation is called a **leftmost derivation**. A leftmost derivation exists in every parse tree, and it can be obtained as follows. Starting from the label of the root A, repeatedly replace the *leftmost* nonterminal in the current string according to the rule suggested by the parse tree. Similarly, a **rightmost derivation** is one that does not precede any other derivation; it is obtained from the parse tree by always expanding the rightmost nonterminal in the current string. Each parse tree has exactly one leftmost and exactly one rightmost derivation. This is so because the leftmost derivation of a parse tree is uniquely determined, since at each step there is one nonterminal to replace: the leftmost one. Similarly for the rightmost derivation. In the example above, D_1 is a leftmost derivation, and D_{10} is a rightmost one.

It is easy to tell when a step of a derivation can be a part of a leftmost derivation: the leftmost nonterminal must be replaced. We write $x \overset{L}{\Rightarrow} y$ if and only if $x = wA\beta$, $y = w\alpha\beta$, where $w \in \Sigma^*$, $\alpha, \beta \in V^*$, $A \in V - \Sigma$, and $A \to \alpha$ is a rule of G. Thus, if $x_1 \Rightarrow x_2 \Rightarrow \cdots \Rightarrow x_n$ is a leftmost derivation, then in fact $x_1 \overset{L}{\Rightarrow} x_2 \overset{L}{\Rightarrow} \cdots \overset{L}{\Rightarrow} x_n$. Similarly for rightmost derivations (the notation is $x \overset{R}{\Rightarrow} y$).

To summarize our insights into parse trees and derivations in this section, we state without formal proof the following theorem.

Theorem 3.2.1: *Let $G = (V, \Sigma, R, S)$ be a context-free grammar, and let $A \in V - \Sigma$, and $w \in \Sigma^*$. Then the following statements are equivalent:*

(a) $A \Rightarrow^* w$.
(b) There is a parse tree with root A and yield w.
(c) There is a leftmost derivation $A \overset{L}{\Rightarrow}^* w$.
(d) There is a rightmost derivation $A \overset{R}{\Rightarrow}^* w$.

Ambiguity

We saw in Example 3.2.2 that there may be a string in the language generated by a context-free grammar with two derivations that are *not* similar —that is to say, *with two distinct parse trees*, or, equivalently, with two distinct rightmost derivations (and two distinct leftmost derivations). For a more substantial example, recall the grammar G that generates all arithmetic expressions over id in Example 3.1.3, and consider another grammar, G', that generates the same language, with these rules:

$$E \to E + E, \quad E \to E * E, \quad E \to (E), \quad E \to \mathsf{id}.$$

It is not hard to see that $L(G') = L(G)$. Still, there are important differences between G and G'. Intuitively, the variant G', by "blurring the distinction" between factors (F) and terms (T) "risks getting wrong" the precedence of multiplication over addition. Indeed, there are two parse trees for the expression $\mathsf{id} + \mathsf{id} * \mathsf{id}$ in G', both shown in Figure 3-7. One of them, 3-7(a), corresponds to the "natural" meaning of this expression (with $*$ taking precedence over $+$), the other is "wrong."

Grammars such as G', with strings that have two or more distinct parse trees, are called **ambiguous**. As we shall see extensively in Section 3.7, assigning a parse tree to a given string in the language —that is to say, *parsing* the string— is an important first step towards understanding the structure of the string, the reasons why it belongs to the language —ultimately its "meaning." This is of course of special importance in the case of grammars such as G and G' above that generate fragments of programming languages. Ambiguous grammars such

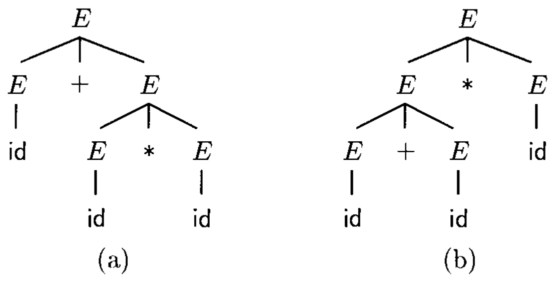

Figure 3-7

as G' are of no help in parsing, since they assign no unique parse tree —no unique "meaning"— to each string in the language.

Fortunately, in this case there is a way to "disambiguate" G' by introducing the new nonterminals T and F (recall the grammar G of Example 3.1.3). That is, there is an *unambiguous* grammar that generates the same language (namely, the grammar G defined in Example 3.1.3; for a proof that the grammar G is indeed unambiguous see Problem 3.2.1). Similarly, the grammar given in Example 3.1.4 for generating balanced strings of parentheses, which is also ambiguous (as discussed at the end of Example 3.2.2, can be easily made unambiguous (see Problem 3.2.2). In fact, there are context-free languages with the property that *all* context-free grammars that generate them must be ambiguous. Such languages are called **inherently ambiguous**. Fortunately, programming languages are never inherently ambiguous.

Problems for Section 3.2

3.2.1. Show that the context-free grammar G given in Example 3.1.3, which generates all arithmetic expressions over id, is unambiguous.

3.2.2. Show that the context-free grammar given in Example 3.1.4, which generates all strings of balanced parentheses is ambiguous. Give an equivalent unambiguous grammar.

3.2.3. Consider the grammar of Example 3.1.3. Give two derivations of the string $\mathsf{id} * \mathsf{id} + \mathsf{id}$, one which is leftmost and one which is not leftmost.

3.2.4. Draw parse trees for each of the following.

(a) The grammar of Example 3.1.2 and the string "big Jim ate green cheese."

(b) The grammar of Example 3.1.3 and the strings $\mathsf{id} + (\mathsf{id} + \mathsf{id}) * \mathsf{id}$ and $(\mathsf{id} * \mathsf{id} + \mathsf{id} * \mathsf{id})$.