ii) The letter $S$, which, when it appears, is usually the start symbol.
iii) Lower-case italic names such as *expr* or *stmt*.

3. Upper-case letters late in the alphabet, such as $X$, $Y$, $Z$, represent *grammar symbols*, that is, either nonterminals or terminals.

4. Lower-case letters late in the alphabet, chiefly $u$, $v$, . . . , $z$, represent strings of terminals.

5. Lower-case Greek letters, $\alpha$, $\beta$, $\gamma$, for example, represent strings of grammar symbols. Thus, a generic production could be written as $A \to \alpha$, indicating that there is a single nonterminal $A$ on the left of the arrow (the *left side* of the production) and a string of grammar symbols $\alpha$ to the right of the arrow (the *right side* of the production).

6. If $A \to \alpha_1$, $A \to \alpha_2$, . . . , $A \to \alpha_k$ are all productions with $A$ on the left (we call them *A-productions*), we may write $A \to \alpha_1 | \alpha_2 | \cdots | \alpha_k$. We call $\alpha_1$, $\alpha_2$, . . . , $\alpha_k$ the *alternatives* for $A$.

7. Unless otherwise stated, the left side of the first production is the start symbol.

**Example 4.3.** Using these shorthands, we could write the grammar of Example 4.2 concisely as

$$E \to E\ A\ E \mid (\ E\ ) \mid -\ E \mid \textbf{id}$$
$$A \to + \mid - \mid * \mid / \mid \uparrow$$

Our notational conventions tell us that $E$ and $A$ are nonterminals, with $E$ the start symbol. The remaining symbols are terminals. □

### Derivations

There are several ways to view the process by which a grammar defines a language. In Section 2.2, we viewed this process as one of building parse trees, but there is also a related derivational view that we frequently find useful. In fact, this derivational view gives a precise description of the top-down construction of a parse tree. The central idea here is that a production is treated as a rewriting rule in which the nonterminal on the left is replaced by the string on the right side of the production.

For example, consider the following grammar for arithmetic expressions, with the nonterminal $E$ representing an expression.

$$E \to E + E \mid E * E \mid (\ E\ ) \mid -\ E \mid \textbf{id} \tag{4.3}$$

The production $E \to -\ E$ signifies that an expression preceded by a minus sign is also an expression. This production can be used to generate more complex expressions from simpler expressions by allowing us to replace any instance of an $E$ by $-\ E$. In the simplest case, we can replace a single $E$ by $-\ E$. We can describe this action by writing

$$E \Rightarrow -E$$

which is read "$E$ derives $-E$." The production $E \rightarrow (E)$ tells us that we could also replace one instance of an $E$ in any string of grammar symbols by $(E)$; e.g., $E*E \Rightarrow (E)*E$ or $E*E \Rightarrow E*(E)$.

We can take a single $E$ and repeatedly apply productions in any order to obtain a sequence of replacements. For example,

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(\mathbf{id})$$

We call such a sequence of replacements a *derivation* of $-(\mathbf{id})$ from $E$. This derivation provides a proof that one particular instance of an expression is the string $-(\mathbf{id})$.

In a more abstract setting, we say that $\alpha A \beta \Rightarrow \alpha\gamma\beta$ if $A \rightarrow \gamma$ is a production and $\alpha$ and $\beta$ are arbitrary strings of grammar symbols. If $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$, we say $\alpha_1$ *derives* $\alpha_n$. The symbol $\Rightarrow$ means "derives in one step." Often we wish to say "derives in zero or more steps." For this purpose we can use the symbol $\overset{*}{\Rightarrow}$. Thus,

1. $\alpha \overset{*}{\Rightarrow} \alpha$ for any string $\alpha$, and
2. If $\alpha \overset{*}{\Rightarrow} \beta$ and $\beta \Rightarrow \gamma$, then $\alpha \overset{*}{\Rightarrow} \gamma$.

Likewise, we use $\overset{+}{\Rightarrow}$ to mean "derives in one or more steps."

Given a grammar $G$ with start symbol $S$, we can use the $\overset{+}{\Rightarrow}$ relation to define $L(G)$, the *language generated by* $G$. Strings in $L(G)$ may contain only terminal symbols of $G$. We say a string of terminals $w$ is in $L(G)$ if and only if $S \overset{+}{\Rightarrow} w$. The string $w$ is called a *sentence* of $G$. A language that can be generated by a grammar is said to be a *context-free language*. If two grammars generate the same language, the grammars are said to be *equivalent*.

If $S \overset{*}{\Rightarrow} \alpha$, where $\alpha$ may contain nonterminals, then we say that $\alpha$ is a *sentential form* of $G$. A sentence is a sentential form with no nonterminals.

**Example 4.4.** The string $-(\mathbf{id}+\mathbf{id})$ is a sentence of grammar (4.3) because there is the derivation

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id}+E) \Rightarrow -(\mathbf{id}+\mathbf{id}) \qquad (4.4)$$

The strings $E$, $-E$, $-(E)$, . . . , $-(\mathbf{id}+\mathbf{id})$ appearing in this derivation are all sentential forms of this grammar. We write $E \overset{*}{\Rightarrow} -(\mathbf{id}+\mathbf{id})$ to indicate that $-(\mathbf{id}+\mathbf{id})$ can be derived from $E$.

We can show by induction on the length of a derivation that every sentence in the language of grammar (4.3) is an arithmetic expression involving the binary operators + and *, the unary operator −, parentheses, and the operand **id**. Similarly, we can show by induction on the length of an arithmetic expression that all such expressions can be generated by this grammar. Thus, grammar (4.3) generates precisely the set of all arithmetic expressions involving binary + and *, unary −, parentheses, and the operand **id**. □

At each step in a derivation, there are two choices to be made. We need to choose which nonterminal to replace, and having made this choice, which

alternative to use for that nonterminal. For example, derivation (4.4) of Example 4.4 could continue from $-(E+E)$ as follows

$$-(E+E) \Rightarrow -(E+\mathbf{id}) \Rightarrow -(\mathbf{id}+\mathbf{id}) \tag{4.5}$$

Each nonterminal in (4.5) is replaced by the same right side as in Example 4.4, but the order of replacements is different.

To understand how certain parsers work we need to consider derivations in which only the leftmost nonterminal in any sentential form is replaced at each step. Such derivations are termed *leftmost*. If $\alpha \Rightarrow \beta$ by a step in which the leftmost nonterminal in $\alpha$ is replaced, we write $\alpha \underset{lm}{\Rightarrow} \beta$. Since derivation (4.4) is leftmost, we can rewrite it as:

$$E \underset{lm}{\Rightarrow} -E \underset{lm}{\Rightarrow} -(E) \underset{lm}{\Rightarrow} -(E+E) \underset{lm}{\Rightarrow} -(\mathbf{id}+E) \underset{lm}{\Rightarrow} -(\mathbf{id}+\mathbf{id})$$

Using our notational conventions, every leftmost step can be written $wA\gamma \underset{lm}{\Rightarrow} w\delta\gamma$ where $w$ consists of terminals only, $A \rightarrow \delta$ is the production applied, and $\gamma$ is a string of grammar symbols. To emphasize the fact that $\alpha$ derives $\beta$ by a leftmost derivation, we write $\alpha \overset{*}{\underset{lm}{\Rightarrow}} \beta$. If $S \overset{*}{\underset{lm}{\Rightarrow}} \alpha$, then we say $\alpha$ is a *left-sentential form* of the grammar at hand.

Analogous definitions hold for *rightmost* derivations in which the rightmost nonterminal is replaced at each step. Rightmost derivations are sometimes called *canonical* derivations.

### Parse Trees and Derivations

A parse tree may be viewed as a graphical representation for a derivation that filters out the choice regarding replacement order. Recall from Section 2.2 that each interior node of a parse tree is labeled by some nonterminal $A$, and that the children of the node are labeled, from left to right, by the symbols in the right side of the production by which this $A$ was replaced in the derivation. The leaves of the parse tree are labeled by nonterminals or terminals and, read from left to right, they constitute a sentential form, called the yield or frontier of the tree. For example, the parse tree for $-(\mathbf{id}+\mathbf{id})$ implied by derivation (4.4) is shown in Fig. 4.2.

To see the relationship between derivations and parse trees, consider any derivation $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$, where $\alpha_1$ is a single nonterminal $A$. For each sentential form $\alpha_i$ in the derivation, we construct a parse tree whose yield is $\alpha_i$. The process is an induction on $i$. For the basis, the tree for $\alpha_1 = A$ is a single node labeled $A$. To do the induction, suppose we have already constructed a parse tree whose yield is $\alpha_{i-1} = X_1X_2 \cdots X_k$. (Recalling our conventions, each $X_i$ is either a nonterminal or a terminal.) Suppose $\alpha_i$ is derived from $\alpha_{i-1}$ by replacing $X_j$, a nonterminal, by $\beta = Y_1Y_2 \cdots Y_r$. That is, at the $i$th step of the derivation, production $X_j \rightarrow \beta$ is applied to $\alpha_{i-1}$ to derive $\alpha_i = X_1X_2 \cdots X_{j-1}\beta X_{j+1} \cdots X_k$.

To model this step of the derivation, we find the $j$th leaf from the left in the current parse tree. This leaf is labeled $X_j$. We give this leaf $r$ children, labeled $Y_1, Y_2, \ldots, Y_r$, from the left. As a special case, if $r = 0$, i.e.,

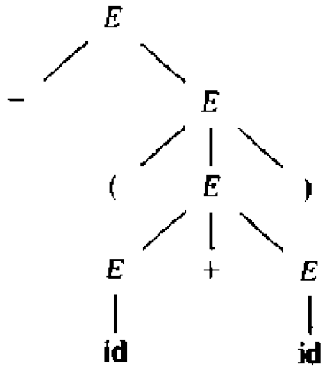


**Fig. 4.2.** Parse tree for $-(\mathbf{id} + \mathbf{id})$.

$\beta = \epsilon$, then we give the $j$th leaf one child labeled $\epsilon$.

**Example 4.5.** Consider derivation (4.4). The sequence of parse trees constructed from this derivation is shown in Fig. 4.3. In the first step of the derivation, $E \Rightarrow -E$. To model this step, we add two children, labeled $-$ and $E$, to the root $E$ of the initial tree to create the second tree.

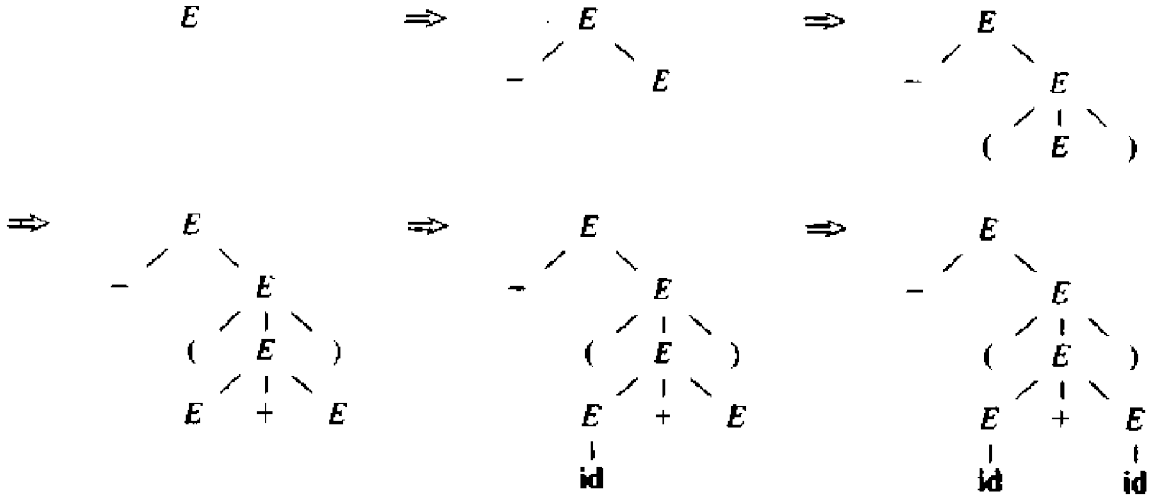


**Fig. 4.3.** Building the parse tree from derivation (4.4).

In the second step of the derivation, $-E \Rightarrow -(E)$. Consequently, we add three children, labeled (, $E$, and ), to the leaf labeled $E$ of the second tree to obtain the third tree with yield $-(E)$. Continuing in this fashion we obtain the complete parse tree as the sixth tree. □

As we have mentioned, a parse tree ignores variations in the order in which symbols in sentential forms are replaced. For example, if derivation (4.4) were continued as in line (4.5), the same final parse tree of Fig. 4.3 would result. These variations in the order in which productions are applied can also be eliminated by considering only leftmost (or rightmost) derivations. It is

not hard to see that every parse tree has associated with it a unique leftmost and a unique rightmost derivation. In what follows, we shall frequently parse by producing a leftmost or rightmost derivation, understanding that instead of this derivation we could produce the parse tree itself. However, we should not assume that every sentence necessarily has only one parse tree or only one leftmost or rightmost derivation.

**Example 4.6.** Let us again consider the arithmetic expression grammar (4.3). The sentence **id**+**id*****id** has the two distinct leftmost derivations:

$$
\begin{aligned}
E &\Rightarrow E+E \\
&\Rightarrow \mathbf{id}+E \\
&\Rightarrow \mathbf{id}+E*E \\
&\Rightarrow \mathbf{id}+\mathbf{id}*E \\
&\Rightarrow \mathbf{id}+\mathbf{id}*\mathbf{id}
\end{aligned}
\qquad\qquad
\begin{aligned}
E &\Rightarrow E*E \\
&\Rightarrow E+E*E \\
&\Rightarrow \mathbf{id}+E*E \\
&\Rightarrow \mathbf{id}+\mathbf{id}*E \\
&\Rightarrow \mathbf{id}+\mathbf{id}*\mathbf{id}
\end{aligned}
$$

with the two corresponding parse trees shown in Fig. 4.4. □

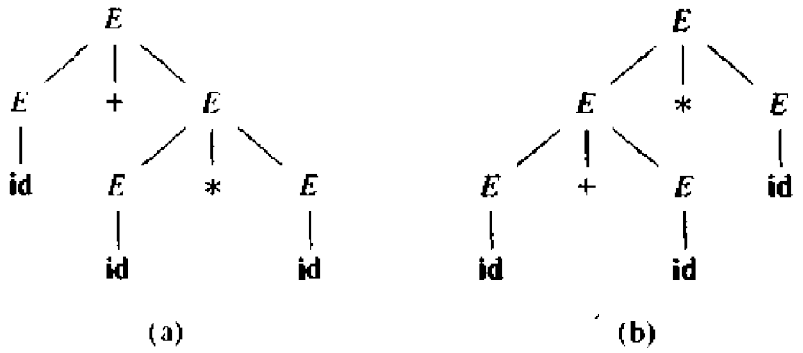


**Fig. 4.4.** Two parse trees for **id**+**id*****id**.

Note that the parse tree of Fig. 4.4(a) reflects the commonly assumed precedence of + and *, while the tree of Fig. 4.4(b) does not. That is, it is customary to treat operator * as having higher precedence than +, corresponding to the fact that we would normally evaluate an expression like $a+b*c$ as $a+(b*c)$, rather than as $(a+b)*c$.

### Ambiguity

A grammar that produces more than one parse tree for some sentence is said to be *ambiguous*. Put another way, an ambiguous grammar is one that produces more than one leftmost or more than one rightmost derivation for the same sentence. For certain types of parsers, it is desirable that the grammar be made unambiguous, for if it is not, we cannot uniquely determine which parse tree to select for a sentence. For some applications we shall also consider methods whereby we can use certain ambiguous grammars, together with *disambiguating rules* that "throw away" undesirable parse trees, leaving us with only one tree for each sentence.