4.2 Context-Free Grammars

Grammars were introduced in Section 2.2 to systematically describe the syntax of programming language constructs like expressions and statements. Using a syntactic variable *stmt* to denote statements and variable *expr* to denote expressions, the production

$$stmt \rightarrow \text{if} \ (\ expr \ ) \ stmt \ \text{else} \ stmt \tag{4.4}$$

specifies the structure of this form of conditional statement. Other productions then define precisely what an *expr* is and what else a *stmt* can be.

This section reviews the definition of a context-free grammar and introduces terminology for talking about parsing. In particular, the notion of derivations is very helpful for discussing the order in which productions are applied during parsing.

4.2.1 The Formal Definition of a Context-Free Grammar

From Section 2.2, a context-free grammar (grammar for short) consists of terminals, nonterminals, a start symbol, and productions.

1. *Terminals* are the basic symbols from which strings are formed. The term "token name" is a synonym for "terminal" and frequently we will use the word "token" for terminal when it is clear that we are talking about just the token name. We assume that the terminals are the first components of the tokens output by the lexical analyzer. In (4.4), the terminals are the keywords if and else and the symbols "(" and ")."

2. *Nonterminals* are syntactic variables that denote sets of strings. In (4.4), *stmt* and *expr* are nonterminals. The sets of strings denoted by nonterminals help define the language generated by the grammar. Nonterminals impose a hierarchical structure on the language that is key to syntax analysis and translation.

3. In a grammar, one nonterminal is distinguished as the *start symbol,* and the set of strings it denotes is the language generated by the grammar. Conventionally, the productions for the start symbol are listed first.

4. The productions of a grammar specify the manner in which the terminals and nonterminals can be combined to form strings. Each *production* consists of:

   (a) A nonterminal called the *head* or *left side* of the production; this production defines some of the strings denoted by the head.

   (b) The symbol -K Sometimes ::= has been used in place of the arrow.

   (c) A *body* or *right side* consisting of zero or more terminals and nonterminals. The components of the body describe one way in which strings of the nonterminal at the head can be constructed.

Example 4.5: The grammar in Fig. 4.2 defines simple arithmetic expressions. In this grammar, the terminal symbols are

id + - * / ( )

The nonterminal symbols are *expression, term* and *factor,* and *expression* is the start symbol •

expression -> *expression* + *term*
expression *expression* - *term*
expression -> *term*
term -> *term* * *factor*
term -» *term* / *factor*
term -» *factor*
factor (*expression*)
factor -> **id**

Figure 4.2: Grammar for simple arithmetic expressions

4.2.2 Notational Conventions

To avoid always having to state that "these are the terminals," "these are the nonterminals," and so on, the following notational conventions for grammars will be used throughout the remainder of this book.

1. These symbols are terminals:

   (a) Lowercase letters early in the alphabet, such as *a, b, c.*

   (b) Operator symbols such as +, *, and so on.

   (c) Punctuation symbols such as parentheses, comma, and so on.

   (d) The digits 0,1,... ,9.

   (e) Boldface strings such as id or if, each of which represents a single terminal symbol.

2. These symbols are nonterminals:

   (a) Uppercase letters early in the alphabet, such as *A, B, C.*

   (b) The letter *S,* which, when it appears, is usually the start symbol.

   (c) Lowercase, italic names such as *expr* or *stmt.*

   (d) When discussing programming constructs, uppercase letters may be used to represent nonterminals for the constructs. For example, nonterminals for expressions, terms, and factors are often represented by *E, T,* and *F,* respectively.

3. Uppercase letters late in the alphabet, such as *X,* 7, *Z,* represent *grammar symbols;* that is, either nonterminals or terminals.

4. Lowercase letters late in the alphabet, chiefly *u,v,..., z,* represent (possibly empty) strings of terminals.

5. Lowercase Greek letters, *a,* ***0,*** 7 for example, represent (possibly empty) strings of grammar symbols. Thus, a generic production can be written as *A* ->• *a,* where *A* is the head and *a* the body.

6. A set of productions *A* -» *ai, A* -»• a_2,... , *A* -> a_k with a common head *A* (call them *A-productions),* may be written *A* ->• *a* | 0:2 | • • • | 0^. Call ai, «2_5 • • • ? cKfc the *alternatives* for A.

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

Example 4.6: Using these conventions, the grammar of Example 4.5 can be rewritten concisely as

E *E* + *T* \ *E* — *T* \ *T*
T -> *T* * *F* \ *T* / *F* \ *F*
F -» *(E)* I id

The notational conventions tell us that *E,* T, and F are nonterminals, with *E* the start symbol. The remaining symbols are terminals. •

4.2.3 Derivations

The construction of a parse tree can be made precise by taking a derivational view, in which productions are treated as rewriting rules. Beginning with the start symbol, each rewriting step replaces a nonterminal by the body of one of its productions. This derivational view corresponds to the top-down construction of a parse tree, but the precision afforded by derivations will be especially helpful when bottom-up parsing is discussed. As we shall see, bottom-up parsing is related to a class of derivations known as "rightmost" derivations, in which the rightmost nonterminal is rewritten at each step.

For example, consider the following grammar, with a single nonterminal *E,* which adds a production *E* ->• — *E* to the grammar (4.3):

E -> *E* + *E* I *E* * *E* I - *E* \ *(E)* \ id (4.7)

The production *E* ->• - *E* signifies that if *E* denotes an expression, then - *E* must also denote an expression. The replacement of a single *E* by - *E* will be described by writing

E -*E*

which is read, *"E* derives *—E."* The production *E -> (E)* can be applied to replace any instance of *E* in any string of grammar symbols by *(E),* e.g., *E* E (E) *E or E*E=>E* (E).* We can take a single *E* and repeatedly apply productions in any order to get a sequence of replacements. For example,

E=>-E=>-(E) => **-(id)**

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

For a general definition of derivation, consider a nonterminal *A* in the middle of a sequence of grammar symbols, as in *aA(3,* where.a and *(3* are arbitrary strings of grammar symbols. Suppose *A* ->• 7 is a production. Then, we write *aAj3 aj(3.* The symbol means, "derives in one step." When a sequence of derivation steps *a* =>• *a*$_2$ => • • • *a*$_n$ rewrites *a*± to *a*$_n$, we say *at derives a*$_n$. Often, we wish to say, "derives in zero or more steps." For this purpose, we can use the symbol . Thus,

1. *a =k- a,* for any string *a,* and

2. If *a* =^ *(3* and *(3* 7, then cu =4> 7.

Likewise, ^ means, "derives in one or more steps."

If *S =$> a,* where *S* is the start symbol of a grammar *G,* we say that *a* is a *sentential form* of G. Note that a sentential form may contain both terminals and nonterminals, and may be empty. A *sentence* of *G* is a sentential form with no nonterminals. The *language generated by* a grammar is its set of sentences. Thus, a string of terminals *w* is in *L(G),* the language generated by *G,* if and only if *w* is a sentence of *G* (or *S* 4> *w).* 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.*

The string **-(id + id)** is a sentence of grammar (4.7) because there is a derivation

E —E =>• *—(E) -{E + E)* **-(id +** *E)* **-(id + id)** (4.8)

The strings *E, -E, -(E),...* , **-(id + id)** are all sentential forms of this grammar. We write *E* ^ **-(id + id)** to indicate that **-(id + id)** can be derived from *E.*

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, we must pick a production with that nonterminal as head. For example, the following alternative derivation of **-(id + id)** differs from derivation (4.8) in the last two steps:

E =>• *—E -(E) -(E + E) -(E +* **id)** **-(id + id)** (4.9)

Each nonterminal is replaced by the same body in the two derivations, but the order of replacements is different.

To understand how parsers work, we shall consider derivations in which the nonterminal to be replaced at each step is chosen as follows:

1. In *leftmost* derivations, the leftmost nonterminal in each sentential is always chosen. If *a* is a step in which the leftmost nonterminal in *a* is replaced, we write *a* ***0.***
   Im

2. In *rightmost* derivations, the rightmost nonterminal is always chosen; we write *a* *(3* in this case.
   rm

Derivation (4.8) is leftmost, so it can be rewritten as

E =>• —*E* -*{E)*=> -*{E* + *E)*=> **-(id** + *E)* => **-(id + id)**

Im **Im** **Im** **Im** **Im**

Note that (4.9) is a rightmost derivation.

Using our notational conventions, every leftmost step can be written as *wAj* =>• *w8y,* where *w* consists of terminals only, *A* -> *S* is the production
Im
applied, and 7 is a string of grammar symbols. To emphasize that *a* derives *(3* by a leftmost derivation, we write *a* => *(3.* If *S* *a,* then we say that a; is a
Im **Im**
left-sentential form of the grammar at hand.

Analogous definitions hold for rightmost derivations. Rightmost derivations are sometimes called *canonical* derivations.

4.2.4 Parse Trees and Derivations

A parse tree is a graphical representation of a derivation that filters out the order in which productions are applied to replace nonterminals. Each interior node of a parse tree represents the application of a production. The interior node is labeled with the nonterminal *A* in the head of the production; the children of the node are labeled, from left to right, by the symbols in the body of the production by which this *A* was replaced during the derivation.

For example, the parse tree for **-(id + id)** in Fig. 4.3, results from the derivation (4.8) as well as derivation (4.9).

The leaves of a parse tree are labeled by nonterminals or terminals and, read from left to right, constitute a sentential form, called the *yield* or *frontier* of the tree.

To see the relationship between derivations and parse trees, consider any derivation ai a_2 • • • a_n, where *a*± is a single nonterminal *A.* For each sentential form *at* in the derivation, we can construct a parse tree whose yield is *ai.* The process is an induction on *i.*

BASIS: The tree for *a* = *A* is a single node labeled *A.*