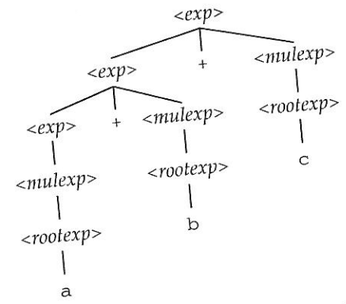
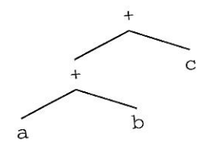


Although a parser would go through all the steps necessary to *find* this parse tree, it would not actually construct an explicit representation of it in memory. Instead, it would probably construct an AST like this:



Many language systems use an AST as an internal representation of a program. Type-checking and other post-parsing steps can be carried out on the AST. Compilers can use the AST as input to the machine-code generator, and interpreters can interpret the program by traversing the AST and carrying out the operation required at each node. Chapter 23 discusses ASTs again, both as input for some little interpreters and as the starting point for some formal definitions of semantics.

3.9 Conclusion

This chapter has shown that a grammar can do more than just define the syntax of a language. By defining a unique parse tree for each program—a parse tree whose structure corresponds to the computation specified by the program—a grammar can begin to define semantics as well. Parse trees and ASTs are where syntax meets semantics.

We must now leave the question of how to define programming languages formally. Defining their syntax is the easy part. After seeing some concepts of programming-language semantics and experiencing them through some ML, Java, and Prolog programming, we will return in Chapter 23 to the question of formally defining programming-language semantics.

Exercises

Exercise 1 Start with the grammar G6, repeated here:

G6:
```
<exp> ::= <exp> + <mulexp> | <mulexp>
<mulexp> ::= <mulexp> * <rootexp> | <rootexp>
<rootexp> ::= ( <exp> )
            | a | b | c
```

Modify it in the following ways:

a. Add subtraction and division operators (- and /) with the customary precedence and associativity.
b. Then add a left-associative operator % between + and * in precedence.
c. Then add a right-associative operator = at lower precedence than any of the other operators.

Exercise 2 Give an EBNF grammar for each of the languages of Exercise 1. Use the EBNF extensions wherever possible to simplify the grammars. Include whatever notes to the reader are required to make the associativity of the operators clear.

Exercise 3 Show that each of the following grammars is ambiguous. (To show that a grammar is ambiguous, you must demonstrate that it can generate two parse trees for the same string.)

a. The grammar G4, repeated here:

G4:
```
<exp> ::= <exp> + <exp>
        | <exp> * <exp>
        | ( <exp> )
        | a | b | c
```

b. This grammar:

```
<person> ::= <woman> | <man>
<woman> ::= wilma | betty | <empty>
<man> ::= fred | barney | <empty>
```

c. The following grammar for strings of balanced parentheses. (A language of any number of different kinds of balanced parentheses is called a Dyck language. This type of language plays an interesting role in the theory of formal languages.)

```
<s> ::= <s> <s> | ( <s> ) | ()
```