2

CONTEXT-FREE LANGUAGES

In Chapter 1 we introduced two different, though equivalent, methods of describing languages: *finite automata* and *regular expressions*. We showed that many languages can be described in this way but that some simple languages, such as $\{0^n1^n | n \geq 0\}$, cannot.

In this chapter we present ***context-free grammars***, a more powerful method of describing languages. Such grammars can describe certain features that have a recursive structure, which makes them useful in a variety of applications.

Context-free grammars were first used in the study of human languages. One way of understanding the relationship of terms such as *noun*, *verb*, and *preposition* and their respective phrases leads to a natural recursion because noun phrases may appear inside verb phrases and vice versa. Context-free grammars help us organize and understand these relationships.

An important application of context-free grammars occurs in the specification and compilation of programming languages. A grammar for a programming language often appears as a reference for people trying to learn the language syntax. Designers of compilers and interpreters for programming languages often start by obtaining a grammar for the language. Most compilers and interpreters contain a component called a ***parser*** that extracts the meaning of a program prior to generating the compiled code or performing the interpreted execution. A number of methodologies facilitate the construction of a parser once a context-free grammar is available. Some tools even automatically generate the parser from the grammar.

The collection of languages associated with context-free grammars are called the ***context-free languages***. They include all the regular languages and many additional languages. In this chapter, we give a formal definition of context-free grammars and study the properties of context-free languages. We also introduce ***pushdown automata***, a class of machines recognizing the context-free languages. Pushdown automata are useful because they allow us to gain additional insight into the power of context-free grammars.

2.1 CONTEXT-FREE GRAMMARS

The following is an example of a context-free grammar, which we call G_1.

$$\begin{aligned} A &\to \texttt{0}A\texttt{1} \\ A &\to B \\ B &\to \texttt{\#} \end{aligned}$$

A grammar consists of a collection of ***substitution rules***, also called ***productions***. Each rule appears as a line in the grammar, comprising a symbol and a string separated by an arrow. The symbol is called a ***variable***. The string consists of variables and other symbols called ***terminals***. The variable symbols often are represented by capital letters. The terminals are analogous to the input alphabet and often are represented by lowercase letters, numbers, or special symbols. One variable is designated as the ***start variable***. It usually occurs on the left-hand side of the topmost rule. For example, grammar G_1 contains three rules. G_1's variables are A and B, where A is the start variable. Its terminals are 0, 1, and #.

You use a grammar to describe a language by generating each string of that language in the following manner.

1. Write down the start variable. It is the variable on the left-hand side of the top rule, unless specified otherwise.
2. Find a variable that is written down and a rule that starts with that variable. Replace the written down variable with the right-hand side of that rule.
3. Repeat step 2 until no variables remain.

For example, grammar G_1 generates the string 000#111. The sequence of substitutions to obtain a string is called a ***derivation***. A derivation of string 000#111 in grammar G_1 is

$$A \Rightarrow \texttt{0}A\texttt{1} \Rightarrow \texttt{00}A\texttt{11} \Rightarrow \texttt{000}A\texttt{111} \Rightarrow \texttt{000}B\texttt{111} \Rightarrow \texttt{000\#111}.$$

You may also represent the same information pictorially with a ***parse tree***. An example of a parse tree is shown in Figure 2.1.

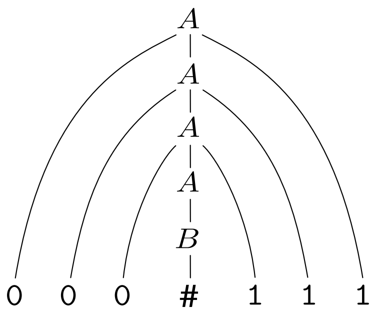

FIGURE 2.1
Parse tree for 000#111 in grammar G_1

All strings generated in this way constitute the ***language of the grammar***. We write $L(G_1)$ for the language of grammar G_1. Some experimentation with the grammar G_1 shows us that $L(G_1)$ is $\{\texttt{0}^n\texttt{\#1}^n|\ n \geq 0\}$. Any language that can be generated by some context-free grammar is called a ***context-free language*** (CFL). For convenience when presenting a context-free grammar, we abbreviate several rules with the same left-hand variable, such as $A \to \texttt{0}A\texttt{1}$ and $A \to B$, into a single line $A \to \texttt{0}A\texttt{1} \mid B$, using the symbol " | " as an "or".

The following is a second example of a context-free grammar, called G_2, which describes a fragment of the English language.

$$\begin{aligned}
\langle\text{SENTENCE}\rangle &\to \langle\text{NOUN-PHRASE}\rangle\langle\text{VERB-PHRASE}\rangle\\
\langle\text{NOUN-PHRASE}\rangle &\to \langle\text{CMPLX-NOUN}\rangle \mid \langle\text{CMPLX-NOUN}\rangle\langle\text{PREP-PHRASE}\rangle\\
\langle\text{VERB-PHRASE}\rangle &\to \langle\text{CMPLX-VERB}\rangle \mid \langle\text{CMPLX-VERB}\rangle\langle\text{PREP-PHRASE}\rangle\\
\langle\text{PREP-PHRASE}\rangle &\to \langle\text{PREP}\rangle\langle\text{CMPLX-NOUN}\rangle\\
\langle\text{CMPLX-NOUN}\rangle &\to \langle\text{ARTICLE}\rangle\langle\text{NOUN}\rangle\\
\langle\text{CMPLX-VERB}\rangle &\to \langle\text{VERB}\rangle \mid \langle\text{VERB}\rangle\langle\text{NOUN-PHRASE}\rangle\\
\langle\text{ARTICLE}\rangle &\to \texttt{a} \mid \texttt{the}\\
\langle\text{NOUN}\rangle &\to \texttt{boy} \mid \texttt{girl} \mid \texttt{flower}\\
\langle\text{VERB}\rangle &\to \texttt{touches} \mid \texttt{likes} \mid \texttt{sees}\\
\langle\text{PREP}\rangle &\to \texttt{with}
\end{aligned}$$

Grammar G_2 has 10 variables (the capitalized grammatical terms written inside brackets); 27 terminals (the standard English alphabet plus a space character); and 18 rules. Strings in $L(G_2)$ include:

```
a boy sees
the boy sees a flower
a girl with a flower likes the boy
```

Each of these strings has a derivation in grammar G_2. The following is a derivation of the first string on this list.

$$\begin{aligned}
\langle\text{SENTENCE}\rangle &\Rightarrow \langle\text{NOUN-PHRASE}\rangle\langle\text{VERB-PHRASE}\rangle \\
&\Rightarrow \langle\text{CMPLX-NOUN}\rangle\langle\text{VERB-PHRASE}\rangle \\
&\Rightarrow \langle\text{ARTICLE}\rangle\langle\text{NOUN}\rangle\langle\text{VERB-PHRASE}\rangle \\
&\Rightarrow \texttt{a}\ \langle\text{NOUN}\rangle\langle\text{VERB-PHRASE}\rangle \\
&\Rightarrow \texttt{a boy}\ \langle\text{VERB-PHRASE}\rangle \\
&\Rightarrow \texttt{a boy}\ \langle\text{CMPLX-VERB}\rangle \\
&\Rightarrow \texttt{a boy}\ \langle\text{VERB}\rangle \\
&\Rightarrow \texttt{a boy sees}
\end{aligned}$$

FORMAL DEFINITION OF A CONTEXT-FREE GRAMMAR

Let's formalize our notion of a context-free grammar (CFG).

DEFINITION 2.2

A ***context-free grammar*** is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the ***variables***,
2. Σ is a finite set, disjoint from V, called the ***terminals***,
3. R is a finite set of ***rules***, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

If u, v, and w are strings of variables and terminals, and $A \rightarrow w$ is a rule of the grammar, we say that uAv ***yields*** uwv, written $uAv \Rightarrow uwv$. Say that u ***derives*** v, written $u \overset{*}{\Rightarrow} v$, if $u = v$ or if a sequence $u_1, u_2, \ldots, u_k$ exists for $k \geq 0$ and

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v.$$

The ***language of the grammar*** is $\{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$.

In grammar G_1, $V = \{A, B\}$, $\Sigma = \{\texttt{0}, \texttt{1}, \texttt{\#}\}$, $S = A$, and R is the collection of the three rules appearing on page 102. In grammar G_2,

$$\begin{aligned}
V = \big\{&\langle\text{SENTENCE}\rangle, \langle\text{NOUN-PHRASE}\rangle, \langle\text{VERB-PHRASE}\rangle, \\
&\langle\text{PREP-PHRASE}\rangle, \langle\text{CMPLX-NOUN}\rangle, \langle\text{CMPLX-VERB}\rangle, \\
&\langle\text{ARTICLE}\rangle, \langle\text{NOUN}\rangle, \langle\text{VERB}\rangle, \langle\text{PREP}\rangle\big\},
\end{aligned}$$

and $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}, \ldots, \texttt{z}, \text{“ ”}\}$. The symbol “ ” is the blank symbol, placed invisibly after each word (a, boy, etc.), so the words won't run together.

Often we specify a grammar by writing down only its rules. We can identify the variables as the symbols that appear on the left-hand side of the rules and the terminals as the remaining symbols. By convention, the start variable is the variable on the left-hand side of the first rule.

EXAMPLES OF CONTEXT-FREE GRAMMARS

EXAMPLE 2.3

Consider grammar $G_3 = (\{S\}, \{\texttt{a}, \texttt{b}\}, R, S)$. The set of rules, R, is

$$S \to \texttt{a}S\texttt{b} \mid SS \mid \varepsilon.$$

This grammar generates strings such as abab, aaabbb, and aababb. You can see more easily what this language is if you think of a as a left parenthesis "(" and b as a right parenthesis ")". Viewed in this way, $L(G_3)$ is the language of all strings of properly nested parentheses. Observe that the right-hand side of a rule may be the empty string ε. ■

EXAMPLE 2.4

Consider grammar $G_4 = (V, \Sigma, R, \langle\text{EXPR}\rangle)$.

V is $\{\langle\text{EXPR}\rangle, \langle\text{TERM}\rangle, \langle\text{FACTOR}\rangle\}$ and Σ is $\{\texttt{a}, \texttt{+}, \texttt{x}, \texttt{(}, \texttt{)}\}$. The rules are

$$\begin{aligned}\langle\text{EXPR}\rangle &\to \langle\text{EXPR}\rangle\texttt{+}\langle\text{TERM}\rangle \mid \langle\text{TERM}\rangle\\ \langle\text{TERM}\rangle &\to \langle\text{TERM}\rangle\texttt{x}\langle\text{FACTOR}\rangle \mid \langle\text{FACTOR}\rangle\\ \langle\text{FACTOR}\rangle &\to \texttt{(}\,\langle\text{EXPR}\rangle\,\texttt{)} \mid \texttt{a}\end{aligned}$$

The two strings a+axa and (a+a)xa can be generated with grammar G_4. The parse trees are shown in the following figure.

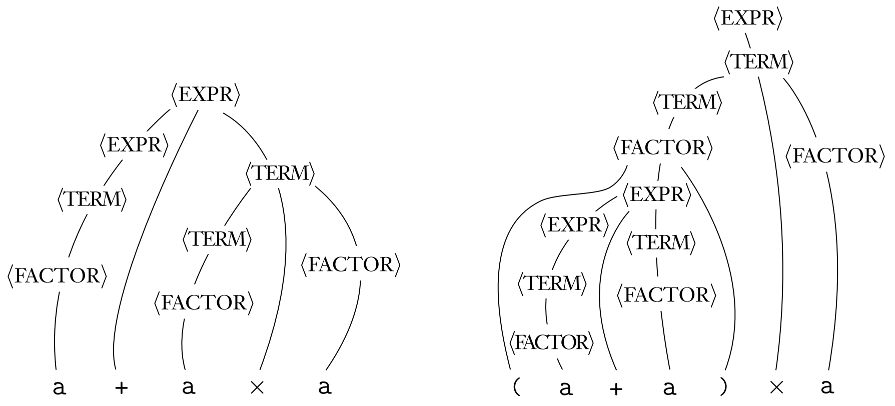

FIGURE 2.5
Parse trees for the strings a+axa and (a+a)xa

A compiler translates code written in a programming language into another form, usually one more suitable for execution. To do so, the compiler extracts the meaning of the code to be compiled in a process called ***parsing***. One representation of this meaning is the parse tree for the code, in the context-free grammar for the programming language. We discuss an algorithm that parses context-free languages later in Theorem 7.16 and in Problem 7.45.

Grammar G_4 describes a fragment of a programming language concerned with arithmetic expressions. Observe how the parse trees in Figure 2.5 "group" the operations. The tree for a+a×a groups the × operator and its operands (the second two a's) as one operand of the + operator. In the tree for (a+a)×a, the grouping is reversed. These groupings fit the standard precedence of multiplication before addition and the use of parentheses to override the standard precedence. Grammar G_4 is designed to capture these precedence relations. ■

DESIGNING CONTEXT-FREE GRAMMARS

As with the design of finite automata, discussed in Section 1.1 (page 41), the design of context-free grammars requires creativity. Indeed, context-free grammars are even trickier to construct than finite automata because we are more accustomed to programming a machine for specific tasks than we are to describing languages with grammars. The following techniques are helpful, singly or in combination, when you're faced with the problem of constructing a CFG.

First, many CFLs are the union of simpler CFLs. If you must construct a CFG for a CFL that you can break into simpler pieces, do so and then construct individual grammars for each piece. These individual grammars can be easily merged into a grammar for the original language by combining their rules and then adding the new rule $S \to S_1 \mid S_2 \mid \cdots \mid S_k$, where the variables S_i are the start variables for the individual grammars. Solving several simpler problems is often easier than solving one complicated problem.

For example, to get a grammar for the language $\{0^n1^n|n \geq 0\} \cup \{1^n0^n|n \geq 0\}$, first construct the grammar

$$S_1 \to 0S_11 \mid \varepsilon$$

for the language $\{0^n1^n|\, n \geq 0\}$ and the grammar

$$S_2 \to 1S_20 \mid \varepsilon$$

for the language $\{1^n0^n|\, n \geq 0\}$ and then add the rule $S \to S_1 \mid S_2$ to give the grammar

$$\begin{aligned} S &\to S_1 \mid S_2 \\ S_1 &\to 0S_11 \mid \varepsilon \\ S_2 &\to 1S_20 \mid \varepsilon. \end{aligned}$$

Second, constructing a CFG for a language that happens to be regular is easy if you can first construct a DFA for that language. You can convert any DFA into an equivalent CFG as follows. Make a variable R_i for each state q_i of the DFA. Add the rule $R_i \rightarrow aR_j$ to the CFG if $\delta(q_i, a) = q_j$ is a transition in the DFA. Add the rule $R_i \rightarrow \varepsilon$ if q_i is an accept state of the DFA. Make R_0 the start variable of the grammar, where q_0 is the start state of the machine. Verify on your own that the resulting CFG generates the same language that the DFA recognizes.

Third, certain context-free languages contain strings with two substrings that are "linked" in the sense that a machine for such a language would need to remember an unbounded amount of information about one of the substrings to verify that it corresponds properly to the other substring. This situation occurs in the language $\{0^n1^n \mid n \geq 0\}$ because a machine would need to remember the number of 0s in order to verify that it equals the number of 1s. You can construct a CFG to handle this situation by using a rule of the form $R \rightarrow uRv$, which generates strings wherein the portion containing the u's corresponds to the portion containing the v's.

Finally, in more complex languages, the strings may contain certain structures that appear recursively as part of other (or the same) structures. That situation occurs in the grammar that generates arithmetic expressions in Example 2.4. Any time the symbol a appears, an entire parenthesized expression might appear recursively instead. To achieve this effect, place the variable symbol generating the structure in the location of the rules corresponding to where that structure may recursively appear.

AMBIGUITY

Sometimes a grammar can generate the same string in several different ways. Such a string will have several different parse trees and thus several different meanings. This result may be undesirable for certain applications, such as programming languages, where a program should have a unique interpretation.

If a grammar generates the same string in several different ways, we say that the string is derived *ambiguously* in that grammar. If a grammar generates some string ambiguously, we say that the grammar is *ambiguous*.

For example, consider grammar G_5:

$$\langle\text{EXPR}\rangle \rightarrow \langle\text{EXPR}\rangle\texttt{+}\langle\text{EXPR}\rangle \mid \langle\text{EXPR}\rangle\texttt{x}\langle\text{EXPR}\rangle \mid \texttt{(}\,\langle\text{EXPR}\rangle\,\texttt{)} \mid \texttt{a}$$

This grammar generates the string a+axa ambiguously. The following figure shows the two different parse trees.