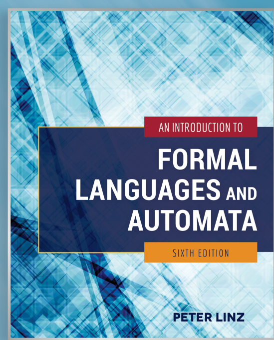
Chapter 5

CONTEXT-FREE LANGUAGES



Learning Objectives

At the conclusion of the chapter, the student will be able to:

- Identify whether a particular grammar is context-free
- Discuss the relationship between regular languages and context-free languages
- Construct context-free grammars for simple languages
- Produce leftmost and rightmost derivations of a string generated by a context-free grammar
- Construct derivation trees for strings generated by a context-free grammar
- Show that a context-free grammar is ambiguous
- Rewrite a grammar to remove ambiguity

Context-Free Grammars

- Many useful languages are not regular
- Context-free grammars are very useful for the definition and processing of programming languages
- A context-free grammar has no restrictions on the right side of its productions, while the left side must be a single variable
- A *language* is context-free if it is generated by a context-free grammar
- Since regular grammars are context-free, the family of regular languages is a proper subset of the family of context-free languages

Context-Free Languages (Example 5.1)

- Consider the grammar

  $V = \{ S \}$, $T = \{ a, b \}$, and productions

  $S \to aSa \mid bSb \mid \lambda$
- Sample derivations:

  $S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aabSbaa \Rightarrow aabbaa$

  $S \Rightarrow bSb \Rightarrow baSab \Rightarrow baab$
- The language generated by the grammar is

  $\{ ww^R : w \in \{ a, b \}^* \}$

  (in other words, even-length palindromes in $\{ a, b \}^*$)

Context-Free Languages (Example 5.4)

- Consider the grammar

  V = { S }, T = { a, b }, and productions

  $S \rightarrow aSb \mid SS \mid \lambda$
- Sample derivations:

  $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

  $S \Rightarrow SS \Rightarrow aSbS \Rightarrow abS \Rightarrow abaSb \Rightarrow abab$
- The language generated by the grammar is

  $\{ w \in \{ a, b \}^*: n_a(w) = n_b(w) \text{ and } n_a(v) \geq n_b(v) \}$

  (where v is any prefix of w)

Leftmost and Rightmost Derivations

- In a *leftmost derivation*, the leftmost variable in a sentential form is replaced at each step
- In a *rightmost derivation*, the rightmost variable in a sentential form is replaced at each step
- Consider the grammar from example 5.5:

  V = { S, A, B }, T = { a, b }, and productions

  $S \rightarrow aAB$

  $A \rightarrow bBb$

  $B \rightarrow A \mid \lambda$
- The string abb has two distinct derivations:
  - Leftmost: $S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abbB \Rightarrow abb$
  - Rightmost: $S \Rightarrow aAB \Rightarrow aA \Rightarrow abBb \Rightarrow abb$

Derivation Trees

- In a *derivation tree* or *parse tree*,
  - the root is labeled S
  - internal nodes are labeled with a variable occurring on the left side of a production
  - the children of a node contain the symbols on the corresponding right side of a production
- For example, given the production $A \rightarrow abABc$, Figure 5.1 shows the corresponding partial derivation tree

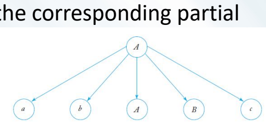


FIGURE 5.1

Derivation Trees (Cont.)

- The yield of a derivation tree is the string of terminals produced by a leftmost depth-first traversal of the tree
- For example, using the grammar from example 5.5, the derivation tree below yields the string abbbb

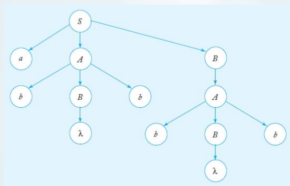


FIGURE 5.3

Sentential Forms and Derivation Trees

- Theorem 5.1 states that, given a context-free grammar G, for every string w in L(G), there exists a derivation tree whose yield is w
- The converse is also true: the yield of any derivation tree formed with productions from G is in L(G)
- Derivation trees show which productions are used in obtaining a sentence, but do not give the order of their application

Parsing and Membership

- The *parsing* problem: given a grammar G and a string w, find a sequence of derivations using the productions in G to produce w
- Can be solved in an exhaustive, top-down, but not very efficient fashion
- Theorem 5.2: Exhaustive parsing is guaranteed to yield all strings eventually, but may fail to stop for strings not in L(G), unless we restrict the productions in the grammar to avoid the forms $A \to \lambda$ and $A \to B$

Parsing and Ambiguity

- A grammar G is *ambiguous* if there is some string w in L(G) for which more than one derivation tree exists
- The grammar with productions S → aSb | SS | λ is ambiguous, since the string aabb has two derivation trees, as shown below

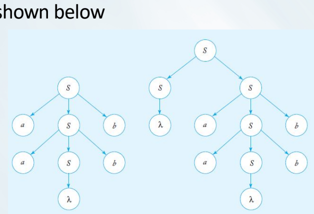


FIGURE 5.4

Ambiguity in Programming Languages

- Consider the grammar below, designed to generate simple arithmetic expressions such as (a+b)*c and a*b+c

  V = { E, I }, T = { a, b, c, +, *, (,) }, and productions

  $E \to I$

  $E \to E+E$

  $E \to E*E$

  $E \to (E)$

  $I \to a \mid b \mid c$
- The grammar is ambiguous because strings such as a+b*c have more than one derivation tree, as shown in Figure 5.5

Derivation Trees from Ambiguous Grammar

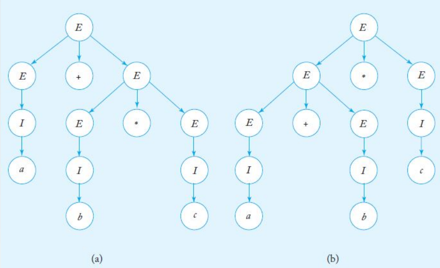


FIGURE 5.5 Two derivation trees for $a + b*c$.

Resolving Ambiguity

- Ambiguity can often be removed by rewriting the grammar so that only one parsing is possible
- Consider the grammar

  V = { E, T, F, I }, T = { a, b, c, +, *, (,) }, and productions

  $E \to T$
  $T \to F$
  $F \to I$
  $E \to E+T$
  $T \to T*F$
  $F \to (E)$
  $I \to a \mid b \mid c$
- As shown in Figure 5.6, only one derivation tree yields the string a+b*c

Derivation Tree for a+b*c Using Unambiguous Grammar

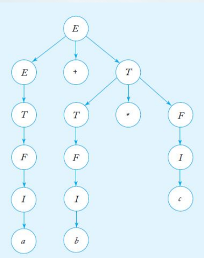


FIGURE 5.6

Ambiguous Languages

- For some languages, it is always possible to find an unambiguous grammar, as shown in the previous example
- However, there are *inherently ambiguous* languages, for which every possible grammar is ambiguous
- Consider the language $\{ a^nb^nb^m \} \cup \{ a^nb^mb^m \}$, which is generated by the grammar

  $S \to S_1 \mid S_2$
  $S_1 \to S_1c \mid A$
  $A \to aAb \mid \lambda$
  $S_2 \to aS_2 \mid B$
  $B \to bBc \mid \lambda$
- The grammar above (and every other equivalent grammar) is ambiguous, because any string of the form $a^nb^nb^m$ has two distinct derivations