CS 4410

Automata, Computability, and Formal Language

Dr. Xuejun Liang

Spring 2019

Chapter 5

Context-Free Languages

1. Context-Free Grammars
   - Examples of Context-Free Languages
   - Leftmost and Rightmost Derivations
   - Derivation Tree
   - Relation Between Sentential Forms and Derivation Tree
2. Parsing and Ambiguity
   - Parsing and Membership
   - Ambiguity in Grammars and Languages
3. Context-Free Grammars and Programming 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

Definition 5.1: A grammar G=(V, T, S, P) is said to be context-free if all productions in P have the form

$$A \rightarrow x$$

where $A \in V$ and $x \in (V \cup T)^*$

A language L is said to be context-free if and only if there is a context-free grammar G such that L=L(G).

Example 5.1:
A context-free grammar but not regular

$$S \rightarrow aSa$$
$$S \rightarrow bSb$$
$$S \rightarrow \lambda$$

$$L(G) = \{ww^R : w \in \{a,b\}^*\}$$

Example 5.2:
A context-free grammar

$$S \rightarrow abB$$
$$A \rightarrow aaBb$$
$$B \rightarrow bbAa$$
$$A \rightarrow \lambda$$

$$L(G) = \{ab(bbaa)^n bba(ba)^n : n \geq 0\}$$

Example 5.3: The language $L=\{a^n b^m : n \neq m\}$ is context-free

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

Definition 5.2: A derivation is said to be leftmost if in each step the leftmost variable in the sentential form is replaced. If in the each step the rightmost variable is replaced, we call the derivation rightmost.

Example 5.5: Leftmost derivation

$$S \to aAB$$
$$A \to bBb$$
$$B \to A \mid \lambda$$

$$S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abAbB \Rightarrow abbBbbB \Rightarrow abbbbB \Rightarrow abbbb$$

Rightmost derivation

$$S \Rightarrow aAB \Rightarrow aA \Rightarrow abBb \Rightarrow abAb \Rightarrow abbBbb \Rightarrow abbbb$$

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

$A \to abABc$

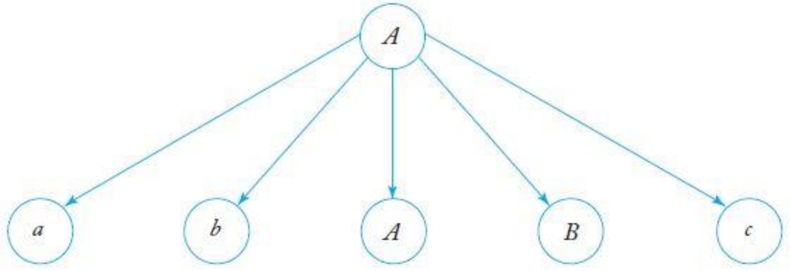


FIGURE 5.1

Leftmost and Rightmost Derivations

A **partial derivation tree** may not be rooted at S and the leaves would be variables, terminals, or λ.

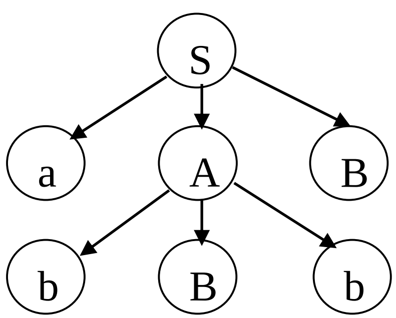
The **yield** of a derivation tree is the string of terminals produced by a leftmost depth-first traversal of the tree.

Example 5.6:

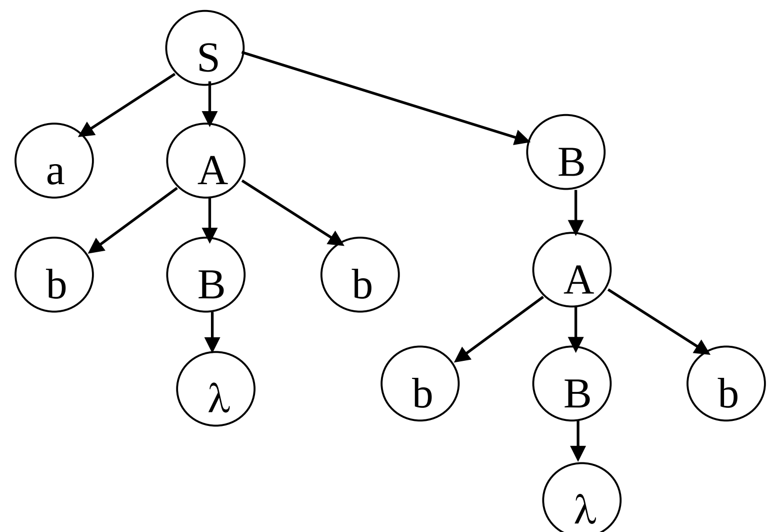
$S \to aAB$

$A \to bBb$

$B \to A \mid \lambda$



Yield is abBbB



Yield is abbbb

Sentential Form and Derivation Tree

Theorem 5.1: Let G=(V, T, S, P) be a context-free grammar. Then for every w ∈ L(G), there exists a derivation tree of G whose yield is w. Conversely, the yield of any derivation tree is in L(G). Also, if t_G is any partial derivation tree for G whose root is labeled S, then the yield of t_G is a sentential form of G.

Parsing and Membership

Membership: Determine whether or not $w \in L(G)$
Parsing: Find a sequence of derivations by which a $w \in L(G)$ is derived.

Example 5.7: Exhaustive search parsing (a top-down approach)
Consider the grammar G: $S \rightarrow SS|aSb|bSa|\lambda$ and the string w=aabb.

Problem: Exhaustive search parsing may not able to terminate for $w \notin L(G)$.

Example 5.8: Consider the grammar G_1: $S \rightarrow SS|aSb|bSa|ab|ba$. We have $L(G_1)=L(G)-\{\lambda\}$ and exhaustive search parsing will terminate for any $w \in \{a,b\}^+$.

Theorem 5.2: Suppose that G=(V, T, S, P) be a context-free grammar which does not have any rules of the form

$$A \rightarrow \lambda \text{ or } A \rightarrow B$$

where $A, B \in V$. Then the exhaustive search parsing method can be made into an algorithm which, for any $w \in \Sigma^*$, either produces a parsing of w, or tells us that no parsing is possible.

Ambiguity in Grammars

Definition 5.5: A context-free grammar G is said to be ambiguous if there exists some $w \in L(G)$ that has at least two distinct derivation trees. Alternatively, ambiguity implies the existence of two or more leftmost or rightmost derivations.

Example 5.10: Grammar G with the products: $S \rightarrow aSb|SS|\lambda$ is ambiguous. The sentence *aabb* has two derivation trees.

Example 5.11: Grammar G=(V,T,E,P) with V={E,I} and T={a,b,c,+,*,(,)}. The products are $E \rightarrow I \mid E+E \mid E*E \mid (E)$, and $I \rightarrow a \mid b \mid c$. Then the string $(a+b)*b$ and $a*b+c$ are in L(G) and the grammar is ambiguous as the string a+b*c has two different derivation trees.

Example 5.12: Grammar in Example 5.11 can be rewritten as V={E,T,F,I} and $E \rightarrow T \mid E+T$, $T \rightarrow F \mid T*F$, $F \rightarrow I \mid (E)$, and $I \rightarrow a \mid b \mid c$. Then the grammar is unambiguous and equivalent to the grammar in Example 5.11.

Ambiguity in Languages

Definition 5.6: If L is context-free language there exists an unambiguous grammar, then L is said to be ambiguous. If every grammar that generates L is ambiguous, then the language is called inherently ambiguous.

Example 5.13: The language

$$L=\{a^n b^n c^m\} \cup \{a^n b^m c^m\}$$

with *n* and *m* non-negative, is an inherently ambiguous context-free language.

Context-Free Grammars and Programming Languages

- Using grammars to specify languages such as the Backrus-Naur form (BNF)

  <expression> ::= <term> | <expression> + <term>

  <term> ::= <factor> | <term> * <factor>

  ……
- Grammatical rules

  Can describe all features of a programming language

  Support efficient parsing
- Detecting and resolving ambiguities in the grammar.