Formal languages and Automata

形式语言与自动机

Chapter4 CONTEXT-FREE LANGUAGES

Beijing University of Posts and Telecommunications

北京邮电大学

刘咏彬 liiuyb@bupt.edu.cn

2023.11

文法类型	文法	所接受的语言	自动机
0型 Type-0	Unrestricted Grammar	递归可枚举语言 Recursively Enumerable Language	图灵机 Turing Machine
1型 Type-1	上下文有关文法 Context Sensitive Grammar $\alpha\to\beta$均有 $\lvert\beta\rvert\geq\lvert\alpha\rvert$	上下文相关语言 Context Sensitive Language G: $S\to aBC\mid aSBC$ $CB\to BC$ $aB\to ab$ $bB\to bb$ $bC\to bc$ $cC\to cc$ $L(G)=\{a^nb^nc^n\mid n\geq 1\}$	线性有界自动机 Linear Bounded Automaton
2型 Type-2	上下文无关文法 Context Free Grammar $\alpha\to\beta$均有 $\lvert\beta\rvert\geq\lvert\alpha\rvert$，并且$\alpha\in V$	上下文无关语言 Context Free language G: $S\to 0A$, $A\to S1\mid 1$ $L(G)=\{0^n1^n\mid n\geq 1\}$	下推自动机 Pushdown Automaton
3型 Type-3	正则文法（左线性文法、右线性文法） Regular Grammar $\alpha\to\beta$均具有形式 $A\to w$ $A\to wB$ A，$B\in V$，$w\in T^+$ 或 $\alpha\to\beta$均具有形式 $A\to w$ $A\to Bw$ A，$B\in V$，$w\in T^+$	正则语言 Regular Language $\{00，01，10，11\}^*$ $\{aa\}^+\{bbb\}^*\#\#\{cc\}^+$	有限状态自动机 Finite State Automaton

About Context-free Languages

- Regular languages is a proper subset of the family of context-free languages.
- The **membership problem** (given a grammar G and a string w, find if w $\in$ L(G)) is much more complicated here than it is for regular languages.
- **Parsing** involves not only membership, but also finding the specific derivation that leads to w.
- Brute force parsing is so inefficient that it is rarely used.

5.1 CONTEXT-FREE GRAMMARS

- The productions in a context-free grammar
  - The left side must be a single variable
  - The right side can be anything

DEFINITION 5.1

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

$$A \to x,$$

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

The grammar $G = (\{S\}, \{a, b\}, S, P)$, with productions

$$\begin{aligned} S &\rightarrow aSa, \\ S &\rightarrow bSb, \\ S &\rightarrow \lambda, \end{aligned}$$

not only context-free, but linear

is context-free. A typical derivation in this grammar is

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

This, and similar derivations, make it clear that

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

The language is context-free, but as shown in Example 4.8, it is not regular.

EXAMPLE 5.2

The grammar G, with productions

$$\begin{aligned} S &\rightarrow abB, \\ A &\rightarrow aaBb, \\ B &\rightarrow bbAa, \\ A &\rightarrow \lambda, \end{aligned}$$

is context-free. We leave it to the reader to show that

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

EXAMPLE 5.3

The language

$$L = \{a^n b^m : n \neq m\}$$

is context-free.

To show this, we need to produce a context-free grammar for the language. The case of $n = m$ is solved in Example 1.11 and we can build on that solution. Take the case $n > m$. We first generate a string with an equal number of a's and b's, then add extra a's on the left. This is done with

$$\begin{aligned} S &\rightarrow AS_1, \\ S_1 &\rightarrow aS_1b|\lambda, \\ A &\rightarrow aA|a. \end{aligned}$$

We can use similar reasoning for the case $n < m$, and we get the answer

$$\begin{aligned} S &\rightarrow AS_1|S_1B, \\ S_1 &\rightarrow aS_1b|\lambda, \\ A &\rightarrow aA|a, \\ B &\rightarrow bB|b. \end{aligned}$$

The resulting grammar is context-free, hence L is a context-free language. However, the grammar is not linear.

EXAMPLE 5.4

Consider the grammar with productions

$$S \to aSb|SS|\lambda.$$

This is another grammar that is context-free, but not linear. Some strings in $L(G)$ are $abaabb$, $aababb$, and $ababab$. It is not difficult to conjecture and prove that

$$\begin{aligned} L \;=\; & \{w \in \{a,b\}^* : n_a(w) = n_b(w) \text{ and } n_a(v) \geq n_b(v), \\ & \text{where } v \text{ is any prefix of } w\}. \end{aligned} \tag{5.1}$$

We can see the connection with programming languages clearly if we replace a and b with left and right parentheses, respectively. The language L includes such strings as $(())$ and $()()()$ and is in fact the set of all properly nested parenthesis structures for the common programming languages.

Leftmost and Rightmost Derivations (最左推导、最右推导)

- G = ({A, B, S} , {a, b} , S, P) with productions

  1. S → AB.
  2. A → aaA.
  3. A → λ.
  4. B → Bb.
  5. B → λ.

yield the same sentence

use the same productions

Consider now the two derivations

$$S \stackrel{1}{\Rightarrow} AB \stackrel{2}{\Rightarrow} aaAB \stackrel{3}{\Rightarrow} aaB \stackrel{4}{\Rightarrow} aaBb \stackrel{5}{\Rightarrow} aab$$

and

$$S \stackrel{1}{\Rightarrow} AB \stackrel{4}{\Rightarrow} ABb \stackrel{2}{\Rightarrow} aaABb \stackrel{5}{\Rightarrow} aaAb \stackrel{3}{\Rightarrow} aab.$$

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 each step the rightmost variable is replaced, we call the derivation **rightmost**.

EXAMPLE 5.5

Consider the grammar with productions

$$\begin{aligned} S &\to aAB, \\ A &\to bBb, \\ B &\to A|\lambda. \end{aligned}$$

Then

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

is a leftmost derivation of the string $abbbb$. A rightmost derivation of the same string is

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

Derivation Trees（推导树）

- A second way of showing derivations, independent of the order in which productions are used, is by a derivation or parse tree（推导树、解析树）.
- **A derivation tree** is an ordered tree（有序数） in which nodes are labeled with the left sides of productions and in which the children of a node represent its corresponding right sides.

$A \rightarrow abABc$

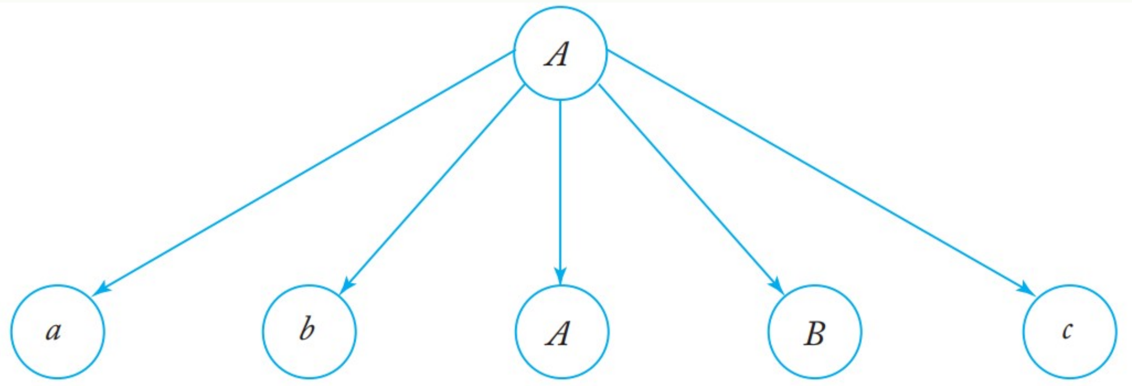

FIGURE 5.1

Derivation Trees（推导树）

DEFINITION 5.3

Let $G = (V, T, S, P)$ be a context-free grammar. An ordered tree is a derivation tree for G if and only if it has the following properties.

1. The root is labeled S.

2. Every leaf has a label from $T \cup \{\lambda\}$.

3. Every interior vertex (a vertex that is not a leaf) has a label from V.

4. If a vertex has label $A \in V$, and its children are labeled (from left to right) $a_1, a_2, ..., a_n$, then P must contain a production of the form

$$A \to a_1 a_2 \cdots a_n.$$

5. A leaf labeled λ has no siblings, that is, a vertex with a child labeled λ can have no other children.

A tree that has properties 3, 4, and 5, but in which 1 does not necessarily hold and in which property 2 is replaced by

2a. Every leaf has a label from $V \cup T \cup \{\lambda\}$,

is said to be a **partial derivation tree**.

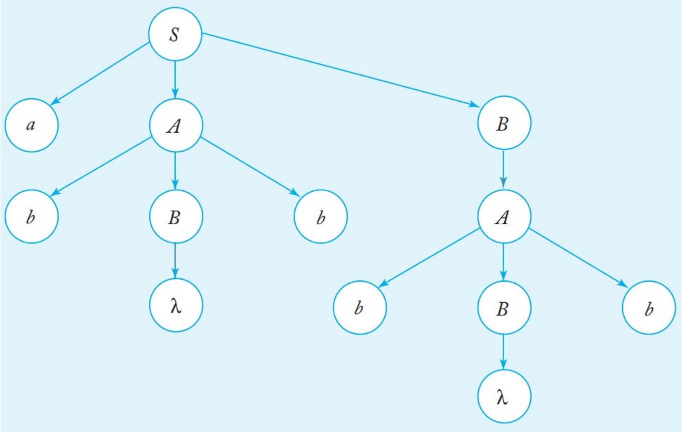

FIGURE 5.3

The string of symbols obtained by reading the leaves of the tree from left to right, omitting any λ' s encountered, is said to be the yield （果、结果） of the tree.

Sentential Form and Derivation Tree

THEOREM 5.1

Let $G = (V, T, S, P)$ be a context-free grammar. Then for every $w \in L(G)$, 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.

Proof by induction on the number of steps in the derivation.

(略)

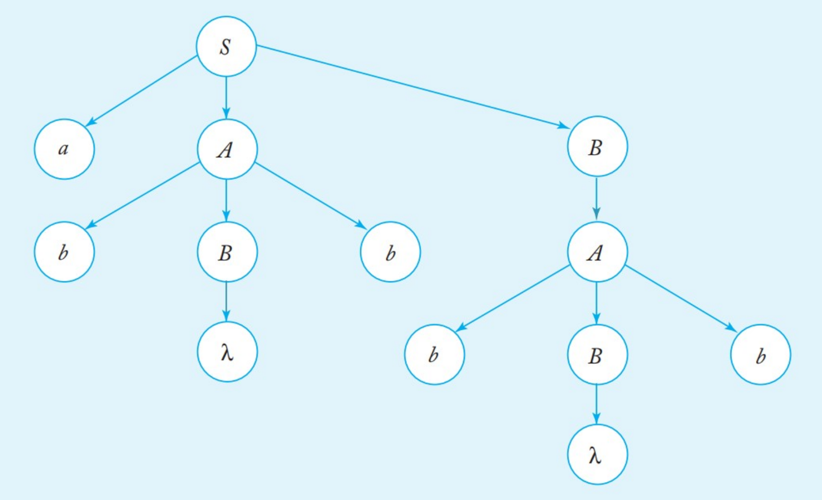

FIGURE 5.3

leftmost derivation

S ⇒ aAB ⇒ abBbB ⇒ abAbB
⇒ abbB ⇒ abbA ⇒ abbbBb
⇒ abbbb

rightmost derivation

S ⇒ aAB ⇒ aAA ⇒ aAbBb
⇒ aAbb ⇒ abBbbb ⇒ abbbb

5.2 PARSING AND AMBIGUITY （分析和二义性）

- An algorithm that can tell us whether w is in L(G) is a membership algorithm（成员资格判定算法）.
- The term parsing（分析） describes finding a sequence of productions by which a $w \in L(G)$ is derived.

EXAMPLE 5.7

Consider the grammar

$$S \to SS\,|aSb|\,bSa|\lambda$$

and the string $w = aabb$. Round one gives us

1. $S \Rightarrow SS,$
2. $S \Rightarrow aSb,$
3. $S \Rightarrow bSa,$
4. $S \Rightarrow \lambda.$

round one
Look at all productions of the form
$S \to x$

The last two of these can be removed from further consideration for obvious reasons. Round two then yields sentential forms

$$S \Rightarrow SS \Rightarrow SSS,$$
$$S \Rightarrow SS \Rightarrow aSbS,$$
$$S \Rightarrow SS \Rightarrow bSaS,$$
$$S \Rightarrow SS \Rightarrow S,$$

round two
Apply all applicable productions to the leftmost variable of sentential form 1

which are obtained by replacing the leftmost S in sentential form 1 with all applicable substitutes. Similarly, from sentential form 2 we with all applicable substitutes. Similarly, from sentential form 2 we get the additional sentential forms

$$S \Rightarrow aSb \Rightarrow aSSb,$$
$$S \Rightarrow aSb \Rightarrow aaSbb,$$
$$S \Rightarrow aSb \Rightarrow abSab,$$
$$S \Rightarrow aSb \Rightarrow ab.$$

round two
Apply all applicable productions to the leftmost variable of sentential form 2

Again, several of these can be removed from contention. On the next round, we find the actual target string from the sequence

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

Therefore, $aabb$ is in the language generated by the grammar under consideration.

- With w = abb, the method will go on producing trial sentential forms indefinitely （无限期地） unless we build into it some way of stopping.
- This difficulty comes from the productions $S \rightarrow \lambda$

Flaws of Exhaustive Search Parsing

- Exhaustive search parsing flaws
  - Tediousness(冗长)
  - Possibly never terminates for strings not in L(G).
- Simplification of CFGs (descripted in Chapter6)
  - Rule out A → λ as well as those of the form A → B.

EXAMPLE 5.8

The grammar

$$S \to SS \,|aSb|\, bSa \,|ab|\, ba$$

satisfies the given requirements. It generates the language in Example 5.7 without the empty string.

Given any $w \in \{a, b\}^+$, the exhaustive search parsing method will always terminate in no more than $|w|$ rounds. This is clear because the length of the sentential form grows by at least one symbol in each round. After $|w|$ rounds we have either produced a parsing or we know that $w \notin L(G)$.

Exhaustive Search Parsing Method (穷举搜索分析析方法)

- **THEOREM 5.2**

  Suppose that $G = (V, T, S, P)$ is a context-free grammar that does not have any rules of the form

  $$A \rightarrow \lambda,$$

  or

  $$A \rightarrow B,$$

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

Proof: For each sentential form, consider both its length and the number of terminal symbols. Each step in the derivation increases at least one of these. Since neither the length of a sentential form nor the number of terminal symbols can exceed |w|, a derivation cannot involve more than 2 |w| rounds, at which time we either have a successful parsing or w cannot be generated by the grammar.

CFG Parse Algorithm Exists

- Complexity of Exhaustive Search Parsing
  - If we restrict ourselves to leftmost derivations, we can have no more than $|P|$ sentential forms after one round, no more than $|P|^2$ sentential forms after the second round, and so on.
  - The total number of sentential forms cannot exceed

$$M = |P| + |P|^2 + \cdots + |P|^{2|w|} = O(P^{2|w|+1}).$$

- **THEOREM 5.3 (略)**

For every context-free grammar there exists an algorithm that parses any $w \in L(G)$ in a number of steps proportional(成正比) to $|w|^3$.

Simple Grammar（简单文法）

- **DEFINITION 5.4**

A context-free grammar G = (V, T, S, P) is said to be a simple grammar or s-grammar if all its productions are of the form

$$A \rightarrow ax,$$

where $A \in V$, $a \in T$, $x \in V^*$, and any pair (A, a) occurs at most once in P.

- Each derivation step produces one terminal symbol
- The whole process must be completed in no more than |w| steps.

EXAMPLE 5.9

The grammar

$$S \rightarrow aS \mid bSS \mid c$$

is an s-grammar. The grammar

$$S \rightarrow aS \mid bSS \mid aSS \mid c$$

is not an s-grammar because the pair (S, a) occurs in the two productions $S \rightarrow aS$ and $S \rightarrow aSS$.

Ambiguous（二义性的）

- **DEFINITION 5.5**

  A context-free grammar G is said to be ambiguous（有二义性的）if there exists some w∈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

The grammar in Example 5.4, with productions $S \to aSb|SS|\lambda$, is ambiguous. The sentence $aabb$ has the two derivation trees shown in Figure 5.4.

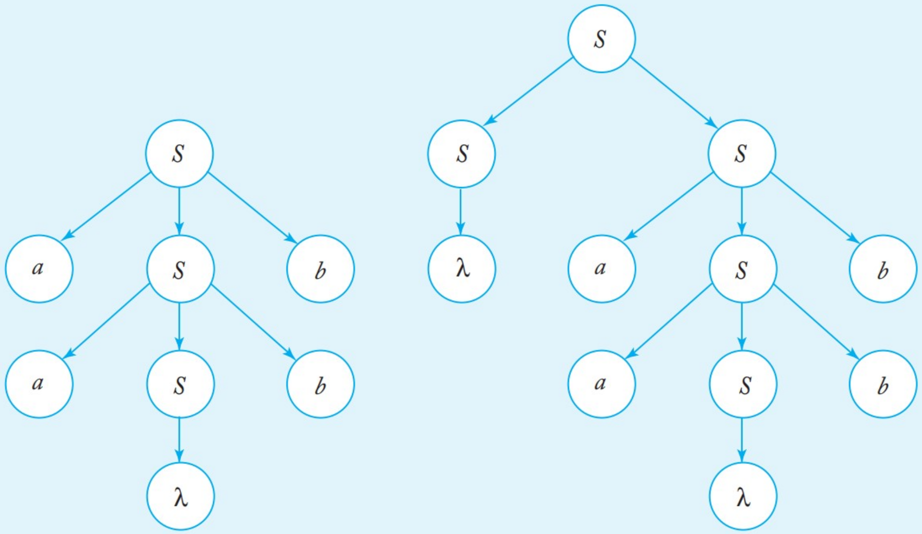

FIGURE 5.4

EXAMPLE 5.11

Consider the grammar $G = (V, T, E, P)$ with

$$V = \{E, I\},$$
$$T = \{a, b, c, +, *, (,)\},$$

and productions

$$E \to I,$$
$$E \to E + E,$$
$$E \to E*E,$$
$$E \to (E),$$
$$I \to a\,|b|c.$$

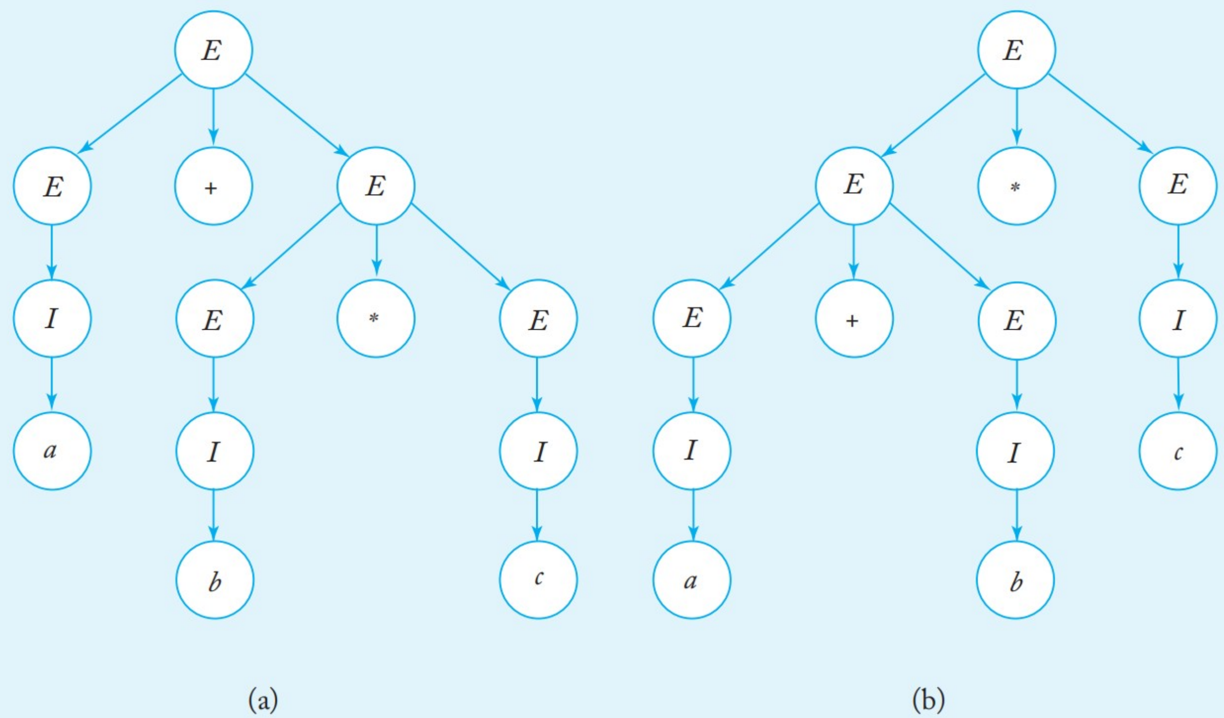

FIGURE 5.5 Two derivation trees for $a + b*c$.

EXAMPLE 5.12

To rewrite the grammar in Example 5.11 we introduce new variables, taking V as $\{E, T, F, I\}$, and replacing the productions with

$$
\begin{aligned}
E &\rightarrow T, \\
T &\rightarrow F, \\
F &\rightarrow I, \\
E &\rightarrow E + T, \\
T &\rightarrow T * F, \\
F &\rightarrow (E), \\
I &\rightarrow a \,|b|\, c.
\end{aligned}
$$

An equivalent unambiguous grammar to EXAMPLE 5.11.

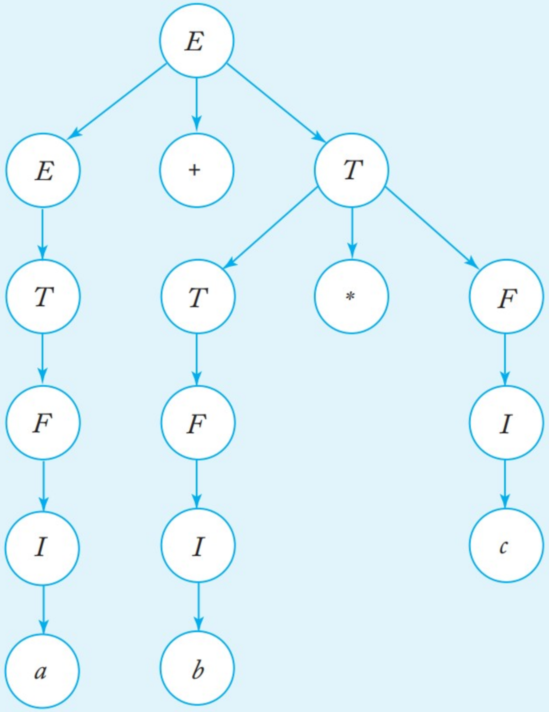

FIGURE 5.6

Inherently Ambiguous(固有二义性的)

- In some instances, removing ambiguity is not possible because the ambiguity is in the language.
- DEFINITION 5.6

  If L is a context-free language for which there exists an unambiguous grammar, then L is said to be unambiguous. 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 nonnegative, is an inherently ambiguous context-free language.

That L is context-free is easy to show. Notice that

$$L = L_1 \cup L_2,$$

where L_1 is generated by

$$\begin{aligned} S_1 &\rightarrow S_1 c|A, \\ A &\rightarrow aAb|\lambda \end{aligned}$$

and L_2 is given by an analogous grammar with start symbol S_2 and productions

$$\begin{aligned} S_2 &\rightarrow aS_2|B, \\ B &\rightarrow bBc|\lambda. \end{aligned}$$

Then L is generated by the combination of these two grammars with the additional production

$$S \rightarrow S_1|S_2.$$

The grammar is ambiguous since the string $a^n b^n c^n$ has two distinct derivations, one starting with $S \Rightarrow S_1$, the other with $S \Rightarrow S_2$.

END