بسم الله الرحمن الرحیم

نظریه زبان‌ها و ماشین‌ها

جلسه ۱۴

مجتبی خلیلی
دانشکده برق و کامپیوتر
دانشگاه صنعتی اصفهان

طراحی CFG

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.

طراحی CFG

- چند تکنیک:

- First, many CFLs are the union of simpler CFLs.

مثال

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

$$S_1 \rightarrow 0S_11 \mid \varepsilon$$

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

$$S_2 \rightarrow 1S_20 \mid \varepsilon$$

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

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

مثال

- CFG برای زبان زیر

$$L = \{0^n 1^n 0^m 1^m \mid n \geqslant 0, m \geqslant 0\}$$

$$L = L_1 L_2$$

$$L_1 = \{0^n 1^n \mid n \geqslant 0\} \longrightarrow L_1 : S_1 \to 0S_1 1 \mid \varepsilon$$

$$L_2 = \{0^m 1^m \mid m \geqslant 0\} \longrightarrow L_2$$ همان

مثال

- CFG برای زبان زیر

$$L = \{0^n 1^n 0^m 1^m \mid n \geqslant 0, m \geqslant 0\}$$

$L_1 : S_1 \rightarrow 0S_11 \mid \varepsilon$

L_2 همان

$$\boxed{\begin{aligned} S &\rightarrow S_1 S_1 \\ S_1 &\rightarrow 0S_11 \mid \varepsilon \end{aligned}}$$

طراحی CFG

- چند تکنیک:

- First, many CFLs are the union of simpler CFLs.
- Second, constructing a CFG for a language that happens to be regular is easy.

رابطه با زبان منظم

- قضیه: هر زبان منظم، یک زبان مستقل از متن است.

Make a variable R_i for each state q_i of the DFA.

Add the rule $R_i \to aR_j$ to the CFG if $\delta(q_i, a) = q_j$ is a transition in the DFA. Add the rule $R_i \to \varepsilon$ 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.

مثال

- CFG برای عبارت منظم زیر

$$(0+1)^*111$$

$E_1 + E_2$	$\Rightarrow$	$S \to S_1 \mid S_2$
$E_1 E_2$		$S \to S_1 S_2$
E_1^*		$S \to S_1 S \mid \varepsilon$

مثال

- CFG برای عبارت منظم زیر

$$(0+1)^*111$$

$$S \rightarrow U111$$

$$U \rightarrow 0U \mid 1U \mid \varepsilon$$

طراحی CFG

- چند تکنیک:

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 $\{0^n1^n | n \geq 0\}$ 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 $R \to uRv$, which generates strings wherein the portion containing the u's corresponds to the portion containing the v's.

$$S \to 0S1$$
$$S \to \epsilon$$
$$\longrightarrow \quad 0^n1^n$$

طراحی CFG

- چند تکنیک:

- First, many CFLs are the union of simpler CFLs.
- Second, constructing a CFG for a language that happens to be regular is easy.
- Third …..
- Finally, in more complex languages, the strings may contain certain structures that appear recursively as part of other (or the same) structures.

مثال

- CFG برای زبان زیر

$$L = \{0^n1^m0^m1^n \mid n \geqslant 0, m \geqslant 0\}$$

0^n1^n

1^m0^m

$\longrightarrow$

$$\boxed{\begin{array}{l} S \rightarrow 0S1 \mid I \\ I \rightarrow 1I0 \mid \varepsilon \end{array}}$$

مثال

- CFG برای زبان زیر

$$L = \{\mathrm{b}^n \mathrm{a}^m \mathrm{b}^{2n} \mid n, m \geq 0\}$$

$$\begin{aligned} S &\to A \mid \mathrm{b}S\mathrm{bb} \\ A &\to \epsilon \mid \mathrm{a}A \end{aligned}$$

ابهام

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*.

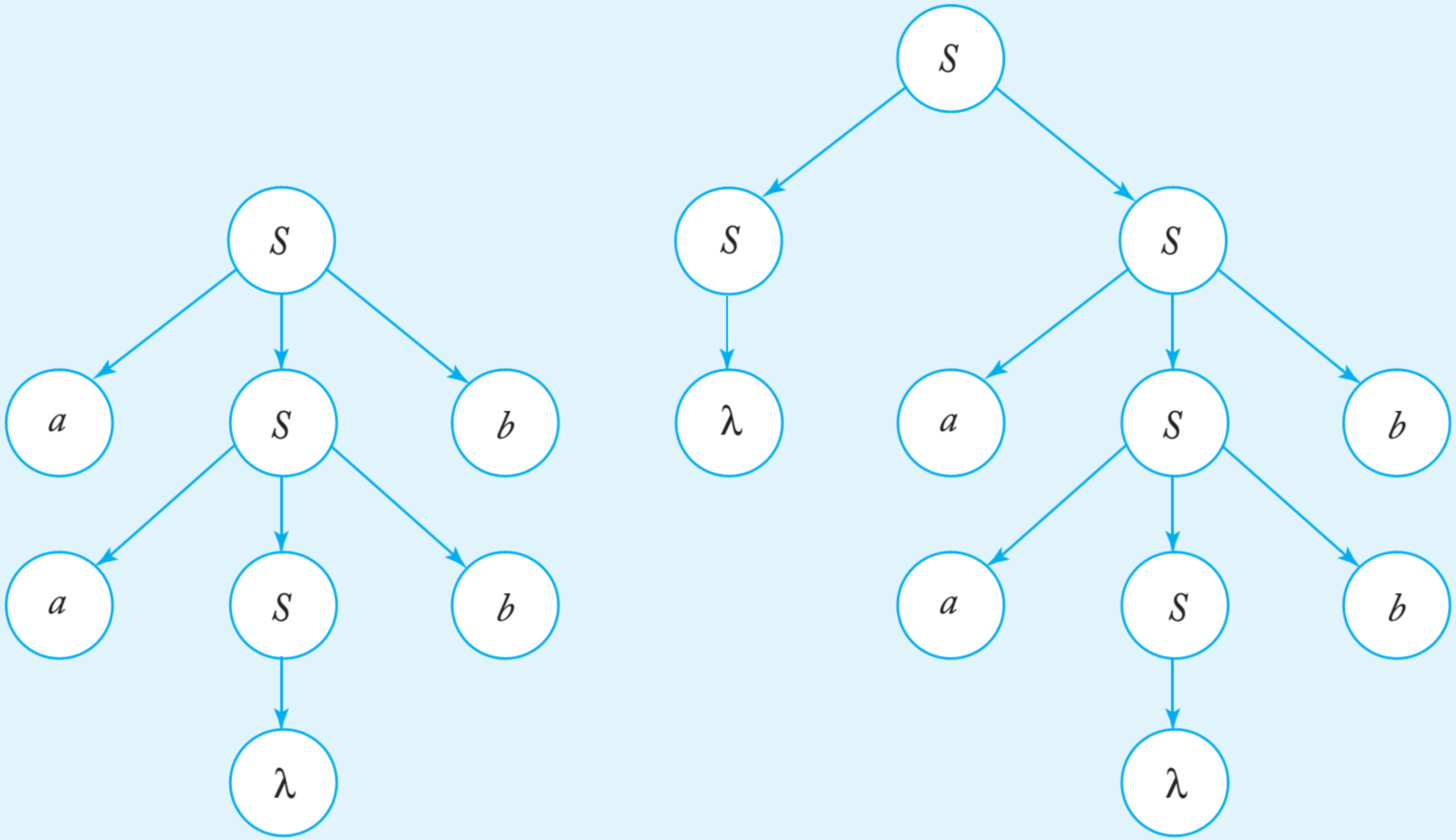
ابهام

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

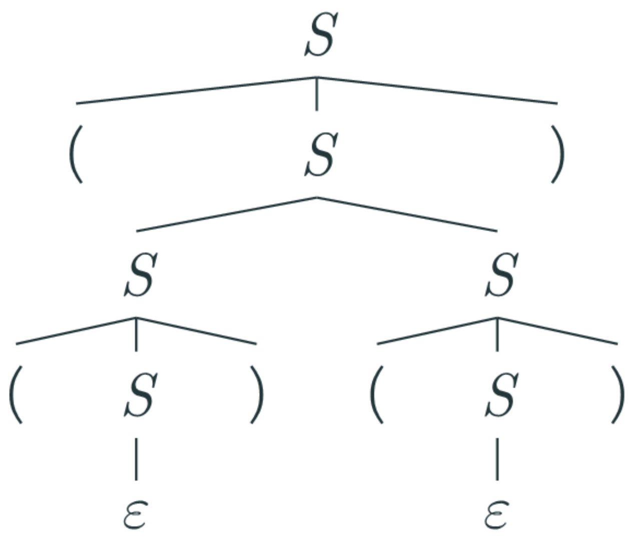
The grammar in Example 5.4, with productions $S \rightarrow aSb\,|SS|\,\lambda$, is ambiguous. The sentence $aabb$ has the two derivation trees shown in Figure 5.4.



مثال

$$
\begin{aligned}
S &\Rightarrow (S)\\
&\Rightarrow (SS)\\
&\Rightarrow ((S)S)\\
&\Rightarrow ((S)(S))\\
&\Rightarrow (()(S))\\
&\Rightarrow (()())
\end{aligned}
$$

$$
\begin{aligned}
S &\Rightarrow (S)\\
&\Rightarrow (SS)\\
&\Rightarrow ((S)S)\\
&\Rightarrow (()S)\\
&\Rightarrow (()(S))\\
&\Rightarrow (()())
\end{aligned}
$$



$$
\begin{aligned}
S &\Rightarrow (S)\\
&\Rightarrow (SS)\\
&\Rightarrow (S(S))\\
&\Rightarrow ((S)(S))\\
&\Rightarrow (()(S))\\
&\Rightarrow (()())
\end{aligned}
$$

$$
\begin{aligned}
S &\Rightarrow (S)\\
&\Rightarrow (SS)\\
&\Rightarrow (S(S))\\
&\Rightarrow (S())\\
&\Rightarrow ((S)())\\
&\Rightarrow (()())
\end{aligned}
$$

مثال

EXAMPLE 5.11

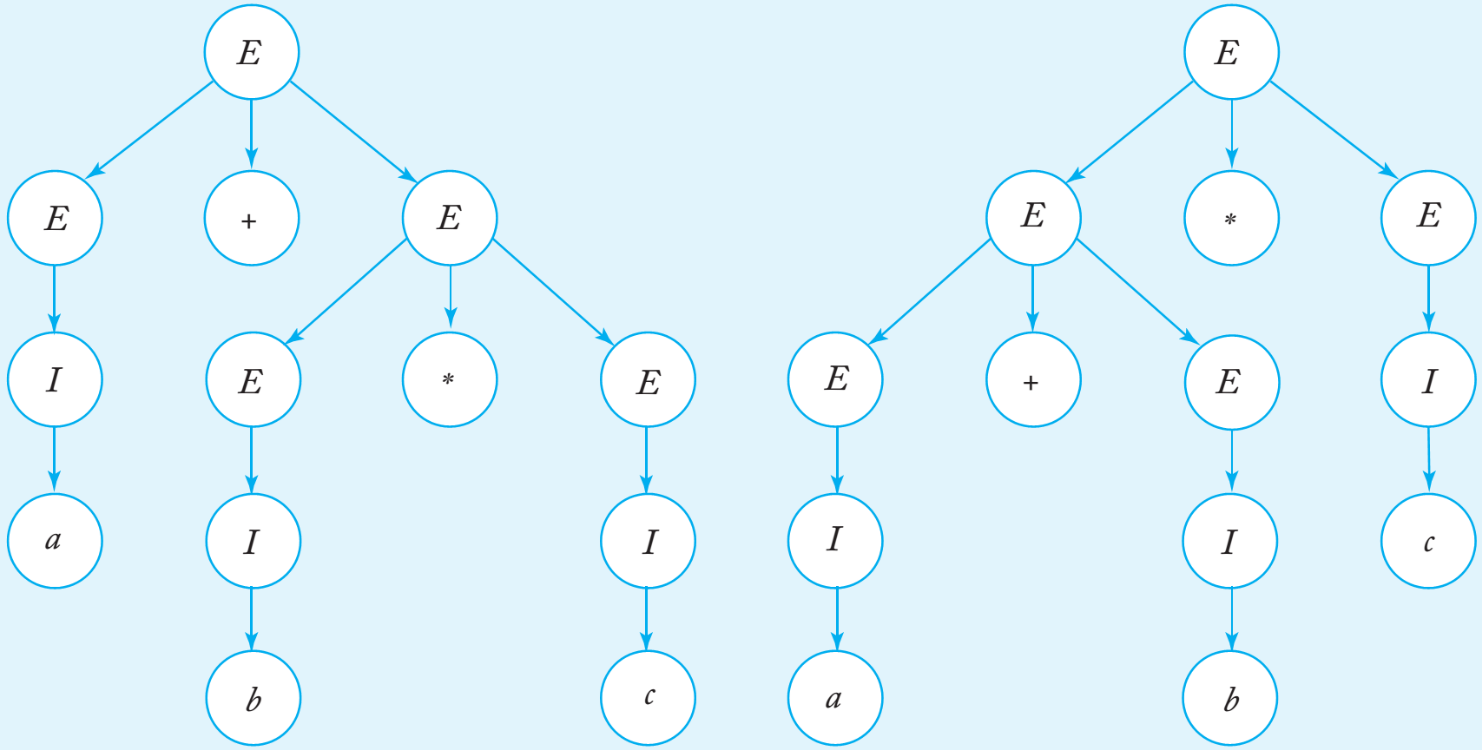
Consider the grammar $G = (V, T, E, P)$ with

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

and productions

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

The strings $(a+b)*c$ and $a*b+c$ are in $L(G)$. It is easy to see that this grammar generates a restricted subset of arithmetic expressions for C-like programming languages. The grammar is ambiguous. For instance, the string $a+b*c$ has two different derivation trees, as shown in Figure 5.5.



(a) (b)

IUT-ECE

Mojtaba Khalili

رفع ابهام

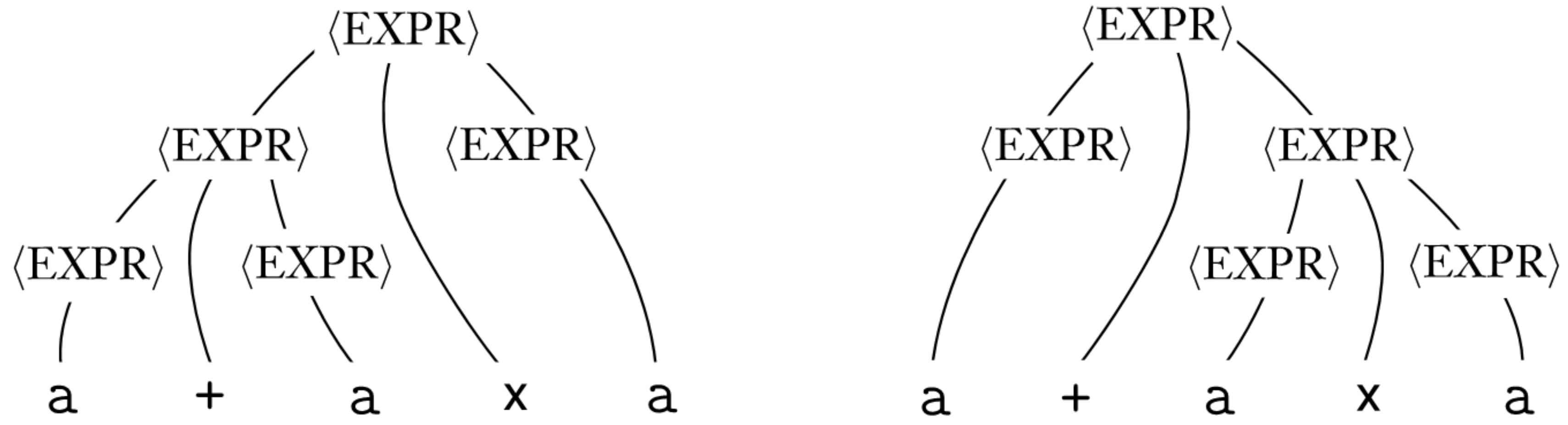
One way to resolve the ambiguity is, as is done in programming manuals, to associate precedence rules with the operators $+$ and $*$. Since $*$ normally has higher precedence than $+$, we would take Figure 5.5(a) as the correct parsing as it indicates that $b*c$ is a subexpression to be evaluated before performing the addition. However, this resolution is completely outside the grammar. It is better to rewrite the grammar so that only one parsing is possible.

رفع ابهام (بازنویسی گرامر)

IUT-ECE

For example, consider grammar G_5:

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



رفع ابهام (بازنویسی گرامر)

IUT-ECE

For example, consider grammar G_5:

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

- اعمال اولویتها در گرامر:

$$E \rightarrow T \mid E + T$$

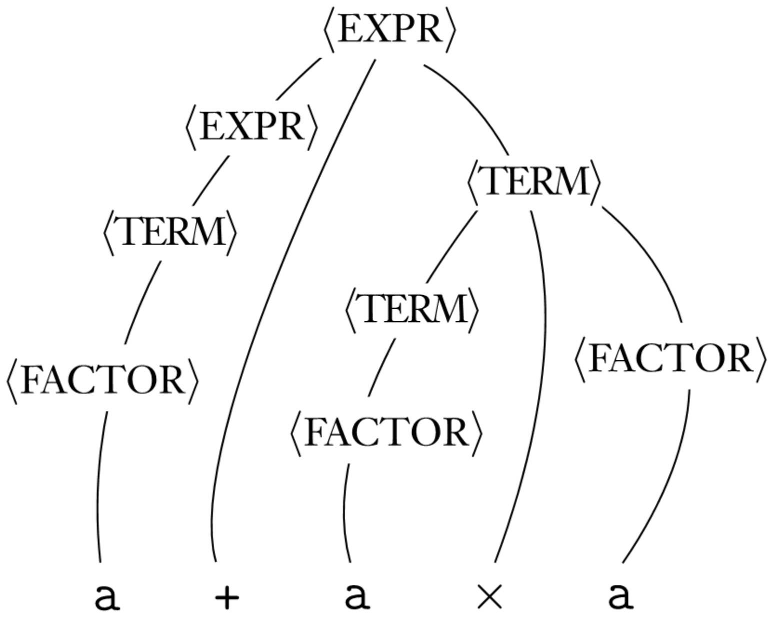
$$T \rightarrow F \mid T * F$$

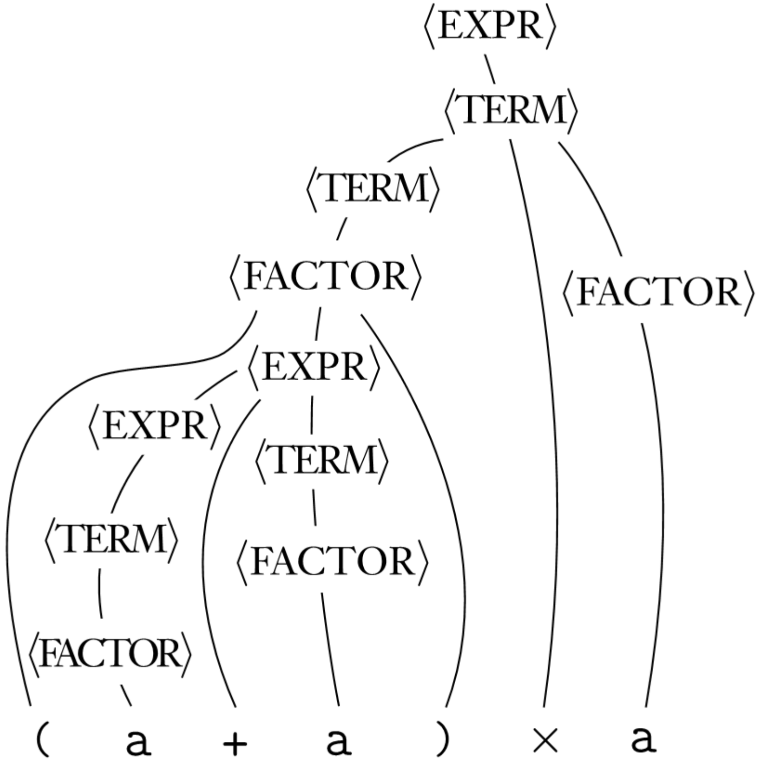
$$F \rightarrow (E) \mid \texttt{a}$$

رفع ابهام (بازنویسی گرامر)

- اعمال اولویتها در گرامر:

$$
\begin{aligned}
\langle \mathrm{EXPR} \rangle &\to \langle \mathrm{EXPR} \rangle + \langle \mathrm{TERM} \rangle \mid \langle \mathrm{TERM} \rangle \\
\langle \mathrm{TERM} \rangle &\to \langle \mathrm{TERM} \rangle \times \langle \mathrm{FACTOR} \rangle \mid \langle \mathrm{FACTOR} \rangle \\
\langle \mathrm{FACTOR} \rangle &\to (\langle \mathrm{EXPR} \rangle) \mid \mathtt{a}
\end{aligned}
$$





مثال

IUT-ECE

EXAMPLE 5.11

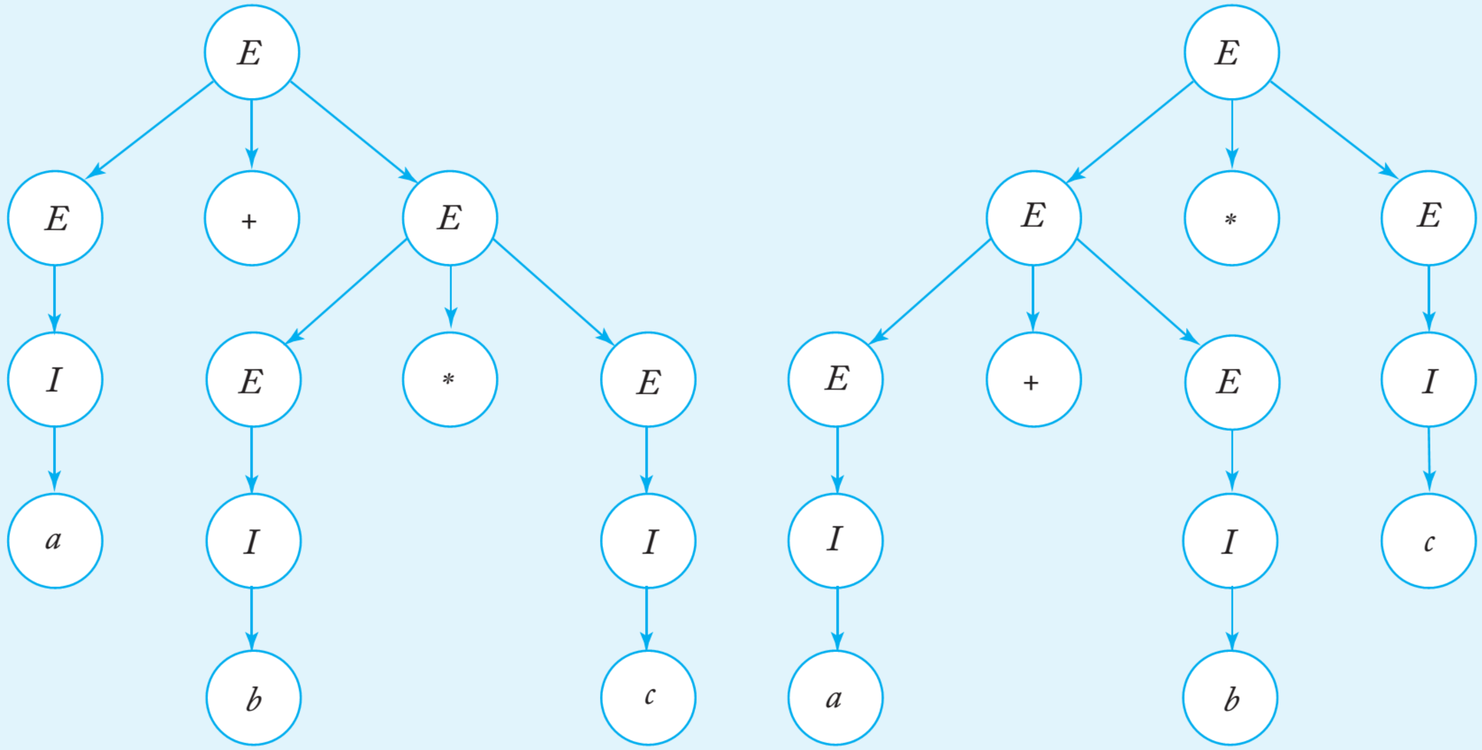
Consider the grammar $G = (V, T, E, P)$ with

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

and productions

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

The strings $(a+b)*c$ and $a*b+c$ are in $L(G)$. It is easy to see that this grammar generates a restricted subset of arithmetic expressions for C-like programming languages. The grammar is ambiguous. For instance, the string $a+b*c$ has two different derivation trees, as shown in Figure 5.5.



(a) (b)

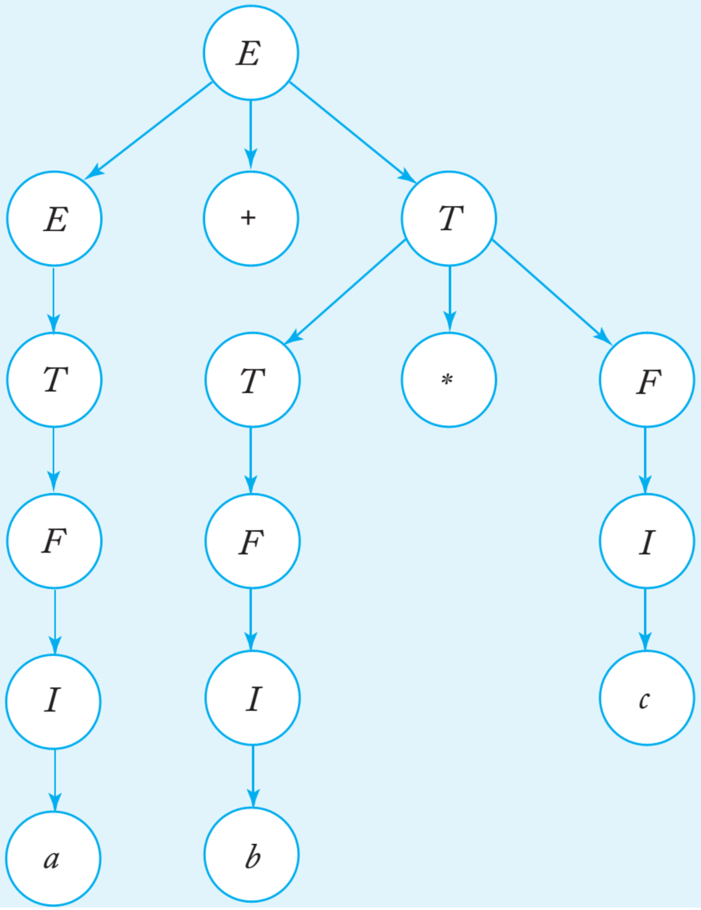
رفع ابهام (بازنویسی گرامر)

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

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

A derivation tree of the sentence $a + b * c$ is shown in Figure 5.6. No other derivation tree is possible for this string: The grammar is unambiguous. It is also equivalent to the grammar in Example 5.11. It is not too hard to justify these claims in this specific instance, but, in general, the questions of whether a given context-free grammar is ambiguous or whether two given context-free grammars are equivalent are very difficult to answer. In fact, we will later show that there are no general algorithms by which these questions can always be resolved.



IUT-ECE

Mojtaba Khalili

گرامر ذاتا مبهم

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**.

مثال

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.

مثال

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

$$L = L_1 \cup L_2,$$

where L_1 is generated by

$$\begin{aligned} S_1 &\to S_1c|A, \\ A &\to aAb|\lambda \end{aligned}$$

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

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

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

$$S \to S_1|S_2.$$

مثال

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 $S \Rightarrow S_2$. It does not, of course, follow from this that L is inherently ambiguous as there might exist some other unambiguous grammars for it. But in some way L_1 and L_2 have conflicting requirements, the first putting a restriction on the number of a's and b's, while the second does the same for b's and c's. A few tries will quickly convince you of the impossibility of combining these requirements in a single set of rules that cover the case $n = m$ uniquely. A rigorous argument, though, is quite technical. One proof can be found in Harrison 1978.