Ambiguity in Grammars and Languages

In the grammar

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$

$\cdots$

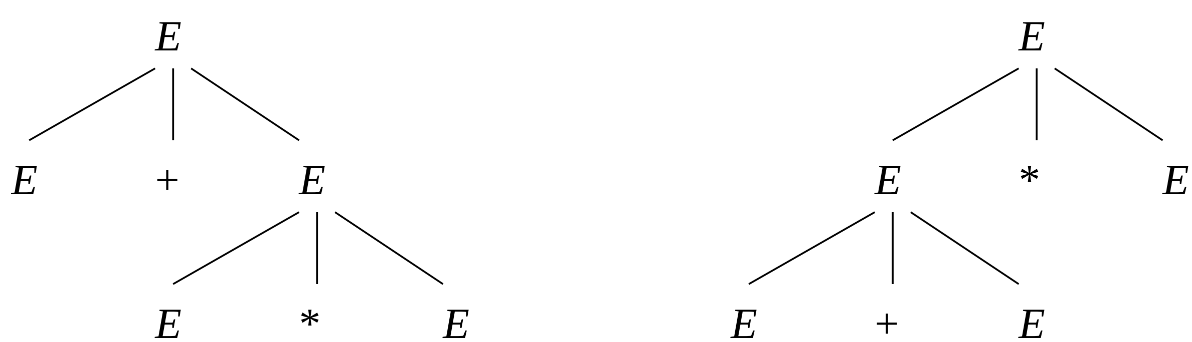
the sentential form $E + E * E$ has two derivations:

$$E \Rightarrow E + E \Rightarrow E + E * E$$

and

$$E \Rightarrow E * E \Rightarrow E + E * E$$

This gives us two parse trees:



(a) (b)

The mere existence of several *derivations* is not dangerous, it is the existence of several parse trees that ruins a grammar.

Example: In the same grammar

$$\begin{aligned} 5.\quad & I \rightarrow a \\ 6.\quad & I \rightarrow b \\ 7.\quad & I \rightarrow Ia \\ 8.\quad & I \rightarrow Ib \\ 9.\quad & I \rightarrow I0 \\ 10.\quad & I \rightarrow I1 \end{aligned}$$

the string $a+b$ has several derivations, e.g.

$$E \Rightarrow E+E \Rightarrow I+E \Rightarrow a+E \Rightarrow a+I \Rightarrow a+b$$

and

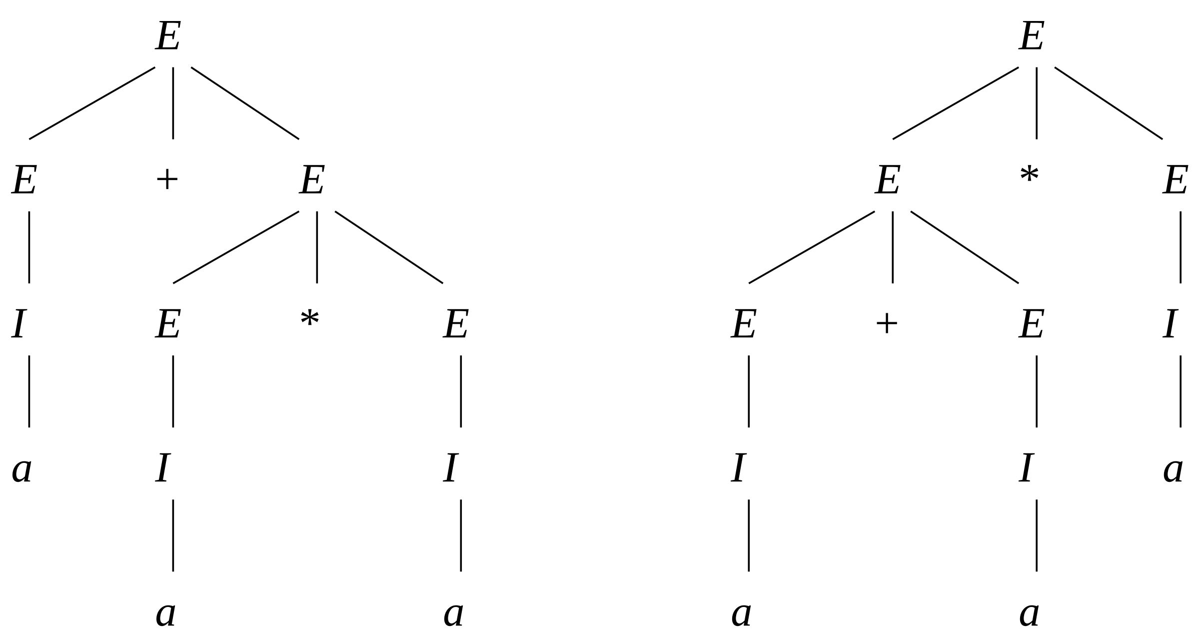
$$E \Rightarrow E+E \Rightarrow E+I \Rightarrow I+I \Rightarrow I+b \Rightarrow a+b$$

However, their parse trees are the same, and the structure of $a+b$ is unambiguous.

Definition: Let $G = (V, T, P, S)$ be a CFG. We say that G is *ambiguous* is there is a string in T^* that has more than one parse tree.

If every string in $L(G)$ has at most one parse tree, G is said to be *unambiguous*.

Example: The terminal string $a+a*a$ has two parse trees:



(a) (b)

Removing Ambiguity From Grammars

Good news: Sometimes we can remove ambiguity "by hand"

Bad news: There is no algorithm to do it

More bad news: Some CFL's have only ambiguous CFG's

We are studying the grammar

$$E \rightarrow I \mid E + E \mid E * E \mid (E)$$
$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

There are two problems:

1. There is no precedence between * and +

2. There is no grouping of sequences of operators, e.g. is $E + E + E$ meant to be $E + (E + E)$ or $(E + E) + E$.

Solution: We introduce more variables, each representing expressions of same "binding strength."

1. A *factor* is an expresson that cannot be broken apart by an adjacent * or $+$. Our factors are

   (a) Identifiers

   (b) A parenthesized expression.

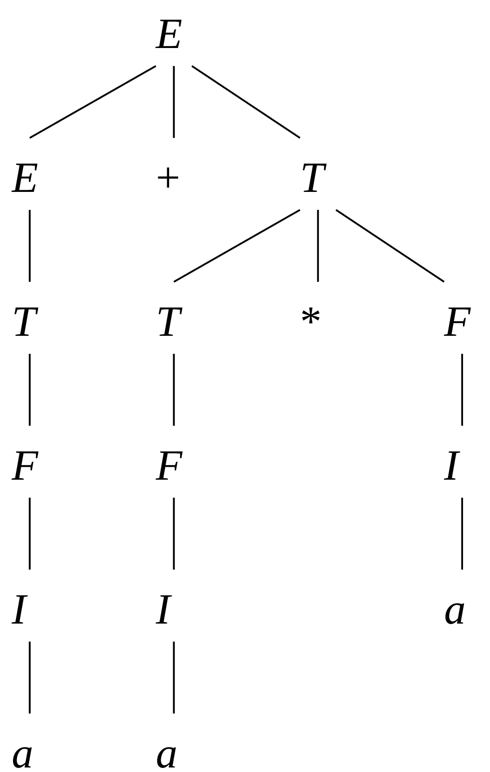
2. A *term* is an expresson that cannot be broken by $+$. For instance $a * b$ can be broken by $a1*$ or $*a1$. It cannot be broken by $+$, since e.g. $a1 + a * b$ is (by precedence rules) same as $a1 + (a * b)$, and $a * b + a1$ is same as $(a * b) + a1$.

3. The rest are *expressions*, i.e. they can be broken apart with * or $+$.

We'll let F stand for factors, T for terms, and E for expressions. Consider the following grammar:

1. $I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
2. $F \rightarrow I \mid (E)$
3. $T \rightarrow F \mid T * F$
4. $E \rightarrow T \mid E + T$

Now the only parse tree for $a + a * a$ will be



Why is the new grammar unambiguous?

Intuitive explanation:

- A factor is either an identifier or (E), for some expression E.

- The only parse tree for a sequence

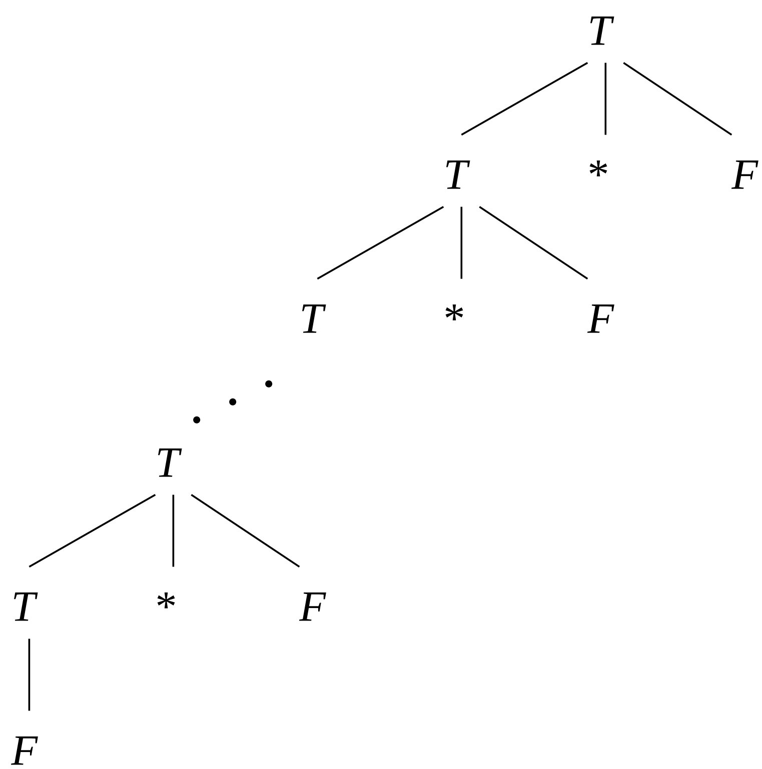
$$f_1 * f_2 * \cdots * f_{n-1} * f_n$$

of factors is the one that gives $f_1 * f_2 * \cdots * f_{n-1}$ as a term and f_n as a factor, as in the parse tree on the next slide.

- An expression is a sequence

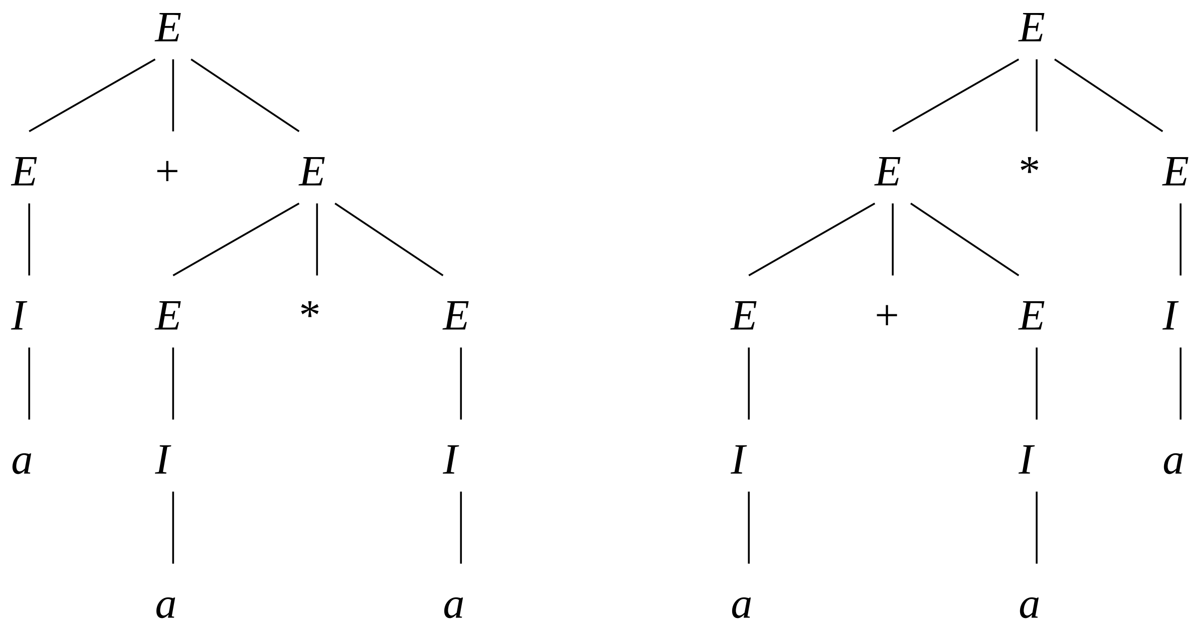
$$t_1 + t_2 + \cdots + t_{n-1} + t_n$$

of terms t_i. It can only be parsed with $t_1 + t_2 + \cdots + t_{n-1}$ as an expression and t_n as a term.


T
T
*
F
T
*
F
T
T
*
F
F

Leftmost derivations and Ambiguity

The two parse trees for $a + a * a$



(a) (b)

give rise to two derivations:

$$E \underset{lm}{\Rightarrow} E + E \underset{lm}{\Rightarrow} I + E \underset{lm}{\Rightarrow} a + E \underset{lm}{\Rightarrow} a + E * E$$
$$\underset{lm}{\Rightarrow} a + I * E \underset{lm}{\Rightarrow} a + a * E \underset{lm}{\Rightarrow} a + a * I \underset{lm}{\Rightarrow} a + a * a$$

and

$$E \underset{lm}{\Rightarrow} E * E \underset{lm}{\Rightarrow} E + E * E \underset{lm}{\Rightarrow} I + E * E \underset{lm}{\Rightarrow} a + E * E$$
$$\underset{lm}{\Rightarrow} a + I * E \underset{lm}{\Rightarrow} a + a * E \underset{lm}{\Rightarrow} a + a * I \underset{lm}{\Rightarrow} a + a * a$$

In General:

- One parse tree, but many derivations

- Many *leftmost* derivation implies many parse trees.

- Many *rightmost* derivation implies many parse trees.

Theorem 5.29: For any CFG G, a terminal string w has two distinct parse trees if and only if w has two distinct leftmost derivations from the start symbol.

Sketch of Proof: *(Only If.)* If the two parse trees differ, they have a node a which different productions, say $A \rightarrow X_1 X_2 \cdots X_k$ and $B \rightarrow Y_1 Y_2 \cdots Y_m$. The corresponding leftmost derivations will use derivations based on these two different productions and will thus be distinct.

(If.) Let's look at how we construct a parse tree from a leftmost derivation. It should now be clear that two distinct derivations gives rise to two different parse trees.

Inherent Ambiguity

A CFL L is *inherently ambiguous* if *all* grammars for L are ambiguous.

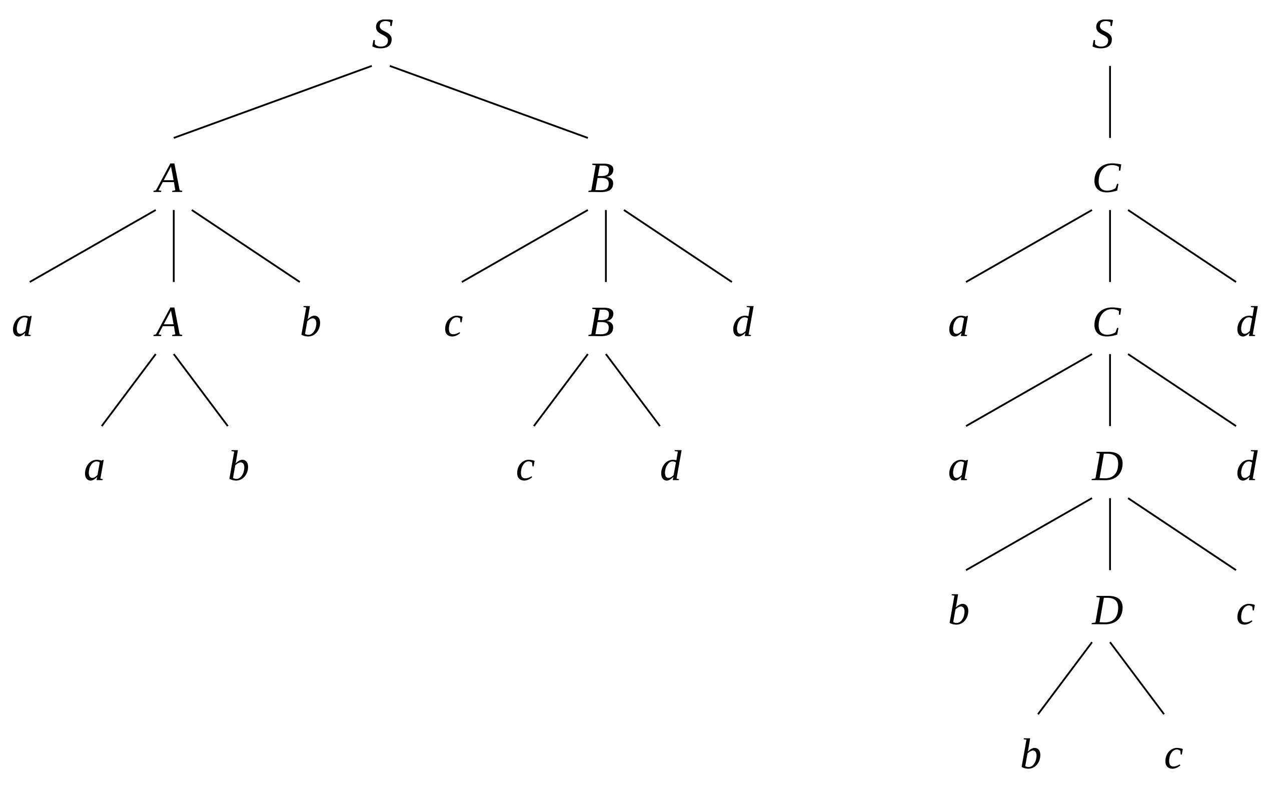
Example: Consider $L =$

$$\{a^n b^n c^m d^m : n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n : n \geq 1, m \geq 1\}.$$

A grammar for L is

$$\begin{aligned} S &\rightarrow AB \mid C \\ A &\rightarrow aAb \mid ab \\ B &\rightarrow cBd \mid cd \\ C &\rightarrow aCd \mid aDd \\ D &\rightarrow bDc \mid bc \end{aligned}$$

Let's look at parsing the string $aabbccdd$.



(a) (b)

From this we see that there are two leftmost derivations:

$$S \underset{lm}{\Rightarrow} AB \underset{lm}{\Rightarrow} aAbB \underset{lm}{\Rightarrow} aabbB \underset{lm}{\Rightarrow} aabbcBd \underset{lm}{\Rightarrow} aabbccdd$$

and

$$S \underset{lm}{\Rightarrow} C \underset{lm}{\Rightarrow} aCd \underset{lm}{\Rightarrow} aaDdd \underset{lm}{\Rightarrow} aabDcdd \underset{lm}{\Rightarrow} aabbccdd$$

It can be shown that *every* grammar for L behaves like the one above. The language L is inherently ambiguous.