Parse Trees

• If $w \in L(G)$, for some CFG, then w has a *parse tree*, which tells us the (syntactic) structure of w

• w could be a program, a SQL-query, an XML-document, etc.

• Parse trees are an alternative representation to derivations and recursive inferences.

• There can be several parse trees for the same string

• Ideally there should be only one parse tree (the "true" structure) for each string, i.e. the language should be *unambiguous*.

• Unfortunately, we cannot always remove the ambiguity.

Constructing Parse Trees

Let $G = (V, T, P, S)$ be a CFG. A tree is a *parse tree* for G if:

1. Each interior node is labelled by a variable in V.

2. Each leaf is labelled by a symbol in $V \cup T \cup \{\epsilon\}$. Any ϵ-labelled leaf is the only child of its parent.

3. If an interior node is lablelled A, and its children (from left to right) labelled

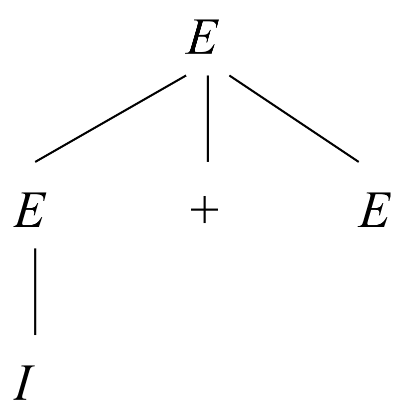
$$X_1, X_2, \ldots, X_k,$$

then $A \rightarrow X_1 X_2 \ldots X_k \in P$.

Example: In the grammar

$$
\begin{aligned}
&1.\ E \to I \\
&2.\ E \to E + E \\
&3.\ E \to E * E \\
&4.\ E \to (E) \\
&\quad\vdots
\end{aligned}
$$

the following is a parse tree:

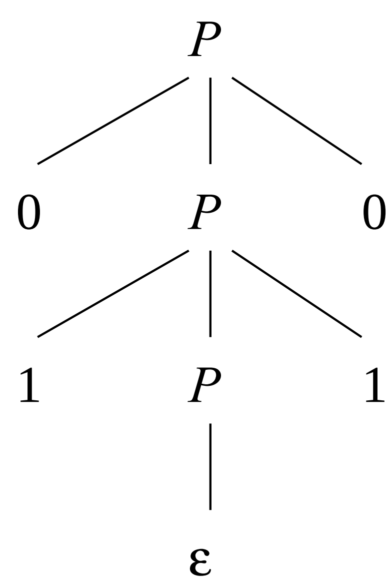


This parse tree shows the derivation $E \stackrel{*}{\Rightarrow} I + E$

Example: In the grammar

1. $P \rightarrow \epsilon$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

the following is a parse tree:



It shows the derivation of $P \stackrel{*}{\Rightarrow} 0110$.

The Yield of a Parse Tree

The *yield* of a parse tree is the string of leaves from left to right.

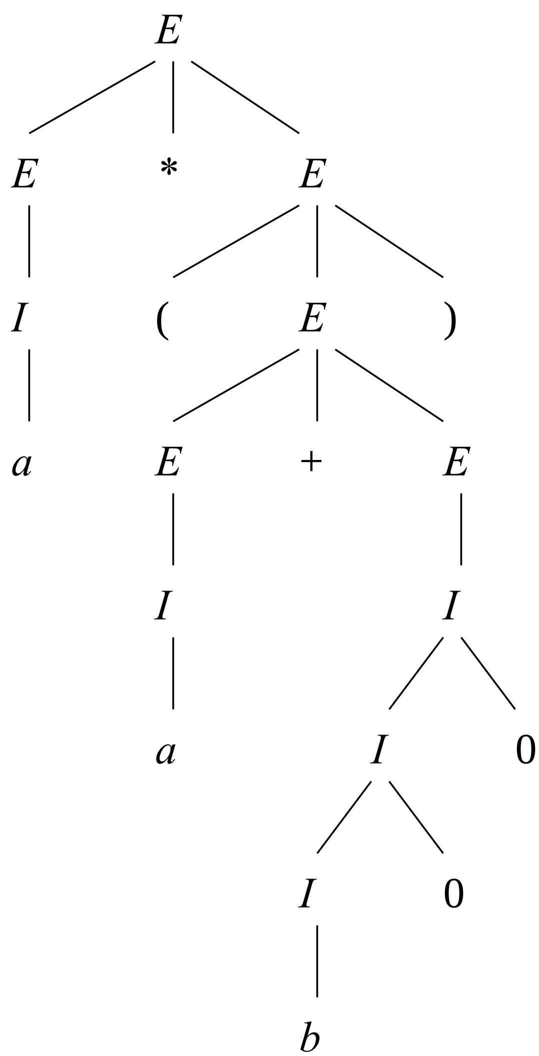
Important are those parse trees where:

1. The yield is a terminal string.

2. The root is labelled by the start symbol

We shall see the the set of yields of these important parse trees is the language of the grammar.

Example: Below is an important parse tree



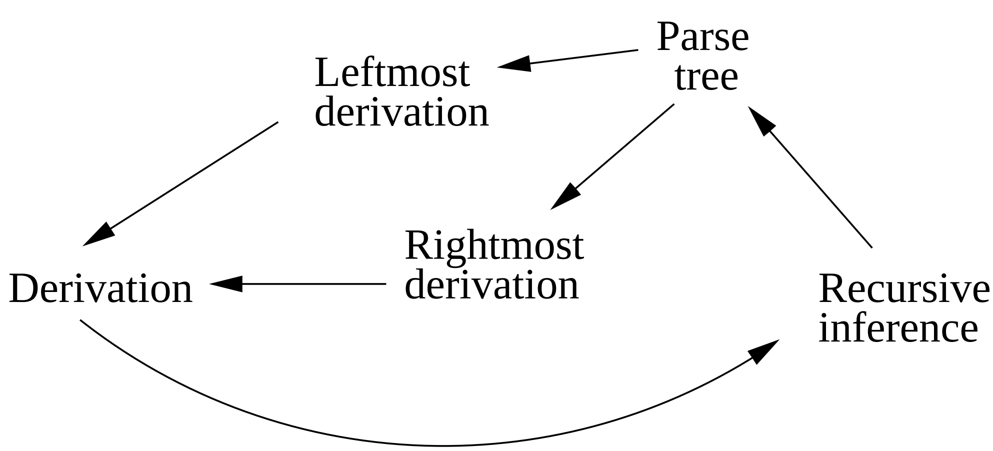
The yield is $a * (a + b00)$.

Compare the parse tree with the derivation on slide 141.

Let $G = (V, T, P, S)$ be a CFG, and $A \in V$. We are going to show that the following are equivalent:

1. We can determine by recursive inference that w is in the language of A
2. $A \overset{*}{\Rightarrow} w$
3. $A \underset{lm}{\overset{*}{\Rightarrow}} w$, and $A \underset{rm}{\overset{*}{\Rightarrow}} w$
4. There is a parse tree of G with root A and yield w.

To prove the equivalences, we use the following plan.



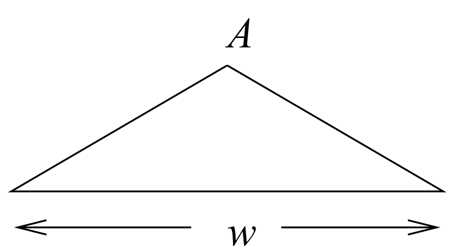
From Inferences to Trees

Theorem 5.12: Let $G = (V, T, P, S)$ be a CFG, and suppose we can show w to be in the language of a variable A. Then there is a parse tree for G with root A and yield w.

by inference

Proof: We do an induction of the length of the inference.

Basis: One step. Then we must have used a production $A \rightarrow w$. The desired parse tree is then



Induction: w is inferred in $n+1$ steps. Suppose the last step was based on a production

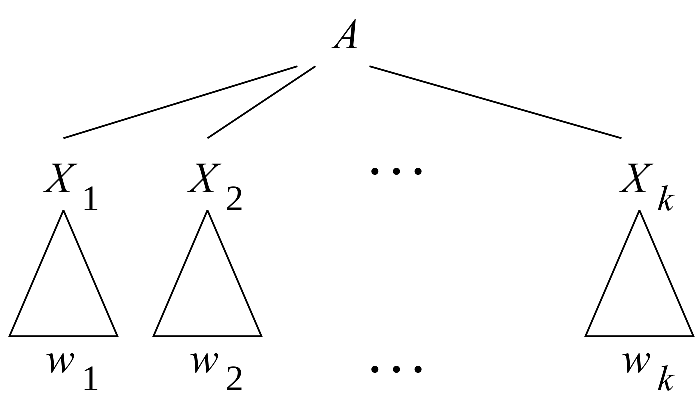
$$A \to X_1 X_2 \cdots X_k,$$

where $X_i \in V \cup T$. We break w up as

$$w_1 w_2 \cdots w_k,$$

where $w_i = X_i$, when $X_i \in T$, and when $X_i \in V$, then w_i was previously inferred being in $L(X_i)$, in at most n steps.

By the IH there are parse trees i with root X_i and yield w_i. Then the following is a parse tree for G with root A and yield w:



From trees to derivations

We'll show how to construct a leftmost derivation from a parse tree.

Example: In the grammar of slide 6 there clearly is a derivation

$$E \Rightarrow I \Rightarrow Ib \Rightarrow ab.$$

Then, for any α and β there is a derivation

$$\alpha E\beta \Rightarrow \alpha I\beta \Rightarrow \alpha Ib\beta \Rightarrow \alpha ab\beta.$$

For example, suppose we have a derivation

$$E \Rightarrow E + E \Rightarrow E + (E).$$

The we can choose $\alpha = E + ($ and $\beta =)$ and continue the derivation as

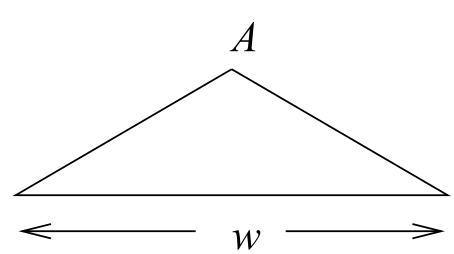
$$E + (E) \Rightarrow E + (I) \Rightarrow E + (Ib) \Rightarrow E + (ab).$$

This is why CFG's are called context-free.

Theorem 5.14: Let $G = (V, T, P, S)$ be a CFG, and suppose there is a parse tree with root labelled A and yield w. Then $A \underset{lm}{\overset{*}{\Rightarrow}} w$ in G.

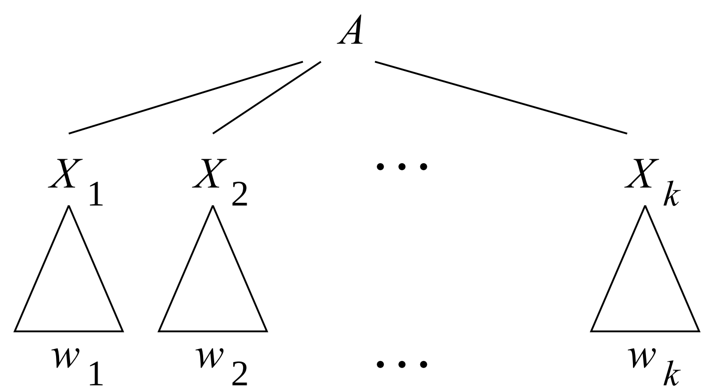
Proof: We do an induction on the height of the parse tree.

Basis: Height is 1. The tree must look like



Consequently $A \rightarrow w \in P$, and $A \underset{lm}{\Rightarrow} w$.

Induction: Height is $n + 1$. The tree must look like



Then $w = w_1 w_2 \cdots w_k$, where

1. If $X_i \in T$, then $w_i = X_i$.

2. If $X_i \in V$, then $X_i \underset{lm}{\overset{*}{\Rightarrow}} w_i$ in G by the IH.

Now we construct $A \overset{*}{\underset{lm}{\Rightarrow}} w$ by an (inner) induction by showing that

$$\forall i : A \overset{*}{\underset{lm}{\Rightarrow}} w_1 w_2 \cdots w_i X_{i+1} X_{i+2} \cdots X_k.$$

Basis: Let $i = 0$. We already know that $A \underset{lm}{\Rightarrow} X_1 X_{i+2} \cdots X_k$.

Induction: Make the IH that

$$A \overset{*}{\underset{lm}{\Rightarrow}} w_1 w_2 \cdots w_{i-1} X_i X_{i+1} \cdots X_k.$$

(*Case 1:*) $X_i \in T$. Do nothing, since $X_i = w_i$ gives us

$$A \overset{*}{\underset{lm}{\Rightarrow}} w_1 w_2 \cdots w_i X_{i+1} \cdots X_k.$$

(*Case 2:*) $X_i \in V$. By the IH there is a derivation $X_i \underset{lm}{\Rightarrow} \alpha_1 \underset{lm}{\Rightarrow} \alpha_2 \underset{lm}{\Rightarrow} \cdots \underset{lm}{\Rightarrow} w_i$. By the contex-free property of derivations we can proceed with

$$A \overset{*}{\underset{lm}{\Rightarrow}}$$

$$w_1 w_2 \cdots w_{i-1} X_i X_{i+1} \cdots X_k \underset{lm}{\Rightarrow}$$

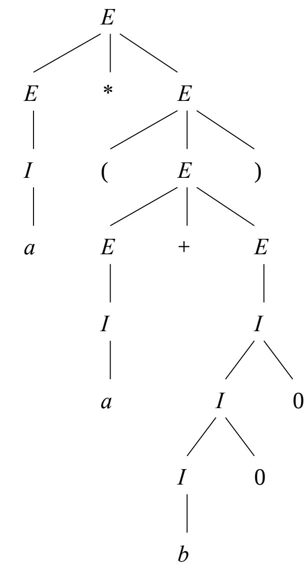
$$w_1 w_2 \cdots w_{i-1} \alpha_1 X_{i+1} \cdots X_k \underset{lm}{\Rightarrow}$$

$$w_1 w_2 \cdots w_{i-1} \alpha_2 X_{i+1} \cdots X_k \underset{lm}{\Rightarrow}$$

$$\cdots$$

$$w_1 w_2 \cdots w_{i-1} w_i X_{i+1} \cdots X_k$$

Example: Let's construct the leftmost derivation for the tree



Suppose we have inductively constructed the leftmost derivation

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

corresponding to the leftmost subtree, and the leftmost derivation

$$E \underset{lm}{\Rightarrow} (E) \underset{lm}{\Rightarrow} (E+E) \underset{lm}{\Rightarrow} (I+E) \underset{lm}{\Rightarrow} (a+E) \underset{lm}{\Rightarrow}$$
$$(a+I) \underset{lm}{\Rightarrow} (a+I0) \underset{lm}{\Rightarrow} (a+I00) \underset{lm}{\Rightarrow} (a+b00)$$

corresponding to the righmost subtree.

For the derivation corresponding to the whole tree we start with $E \underset{lm}{\Rightarrow} E * E$ and expand the first E with the first derivation and the second E with the second derivation:

$$
\begin{aligned}
& E \underset{lm}{\Rightarrow} \\
& E * E \underset{lm}{\Rightarrow} \\
& I * E \underset{lm}{\Rightarrow} \\
& a * E \underset{lm}{\Rightarrow} \\
& a * (E) \underset{lm}{\Rightarrow} \\
& a * (E + E) \underset{lm}{\Rightarrow} \\
& a * (I + E) \underset{lm}{\Rightarrow} \\
& a * (a + E) \underset{lm}{\Rightarrow} \\
& a * (a + I) \underset{lm}{\Rightarrow} \\
& a * (a + I0) \underset{lm}{\Rightarrow} \\
& a * (a + I00) \underset{lm}{\Rightarrow} \\
& a * (a + b00)
\end{aligned}
$$

From Derivations to Recursive Inferences

Observation: Suppose that $A \Rightarrow X_1 X_2 \cdots X_k \stackrel{*}{\Rightarrow} w$. Then $w = w_1 w_2 \cdots w_k$, where $X_i \stackrel{*}{\Rightarrow} w_i$

The factor w_i can be extracted from $A \stackrel{*}{\Rightarrow} w$ by looking at the expansion of X_i only.

Example: $E \stackrel{*}{\Rightarrow} a * b + a$, and

$$E \Rightarrow \underbrace{E}_{X_1} \underbrace{*}_{X_2} \underbrace{E}_{X_3} \underbrace{+}_{X_4} \underbrace{E}_{X_5}$$

We have

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

By looking at the expansion of $X_3 = E$ only, we can extract

$$E \Rightarrow I \Rightarrow b.$$

Theorem 5.18: Let $G = (V, T, P, S)$ be a CFG. Suppose $A \overset{*}{\underset{G}{\Rightarrow}} w$, and that w is a string of terminals. Then we can infer that w is in the language of variable A.

Proof: We do an induction on the length of the derivation $A \overset{*}{\underset{G}{\Rightarrow}} w$.

Basis: One step. If $A \underset{G}{\Rightarrow} w$ there must be a production $A \rightarrow w$ in P. The we can infer that w is in the language of A.

Induction: Suppose $A \overset{*}{\underset{G}{\Rightarrow}} w$ in $n+1$ steps. Write the derivation as

$$A \underset{G}{\Rightarrow} X_1X_2 \cdots X_k \overset{*}{\underset{G}{\Rightarrow}} w$$

The as noted on the previous slide we can break w as $w_1w_2 \cdots w_k$ where $X_i \overset{*}{\underset{G}{\Rightarrow}} w_i$. Furthermore, $X_i \overset{*}{\underset{G}{\Rightarrow}} w_i$ can use at most n steps.

Now we have a production $A \to X_1X_2 \cdots X_k$, and we know by the IH that we can infer w_i to be in the language of X_i.

Therefore we can infer $w_1w_2 \cdots w_k$ to be in the language of A.

Gram Matic (Paul Cernea):
https://itunes.apple.com/ca/app/gram-matic/id914302373?mt=8

Ambiguity in Grammars and Languages

In the grammar

$$\begin{array}{l} 1.\ E \rightarrow I \\ 2.\ E \rightarrow E + E \\ 3.\ E \rightarrow E * E \\ 4.\ E \rightarrow (E) \\ \quad \cdots \end{array}$$

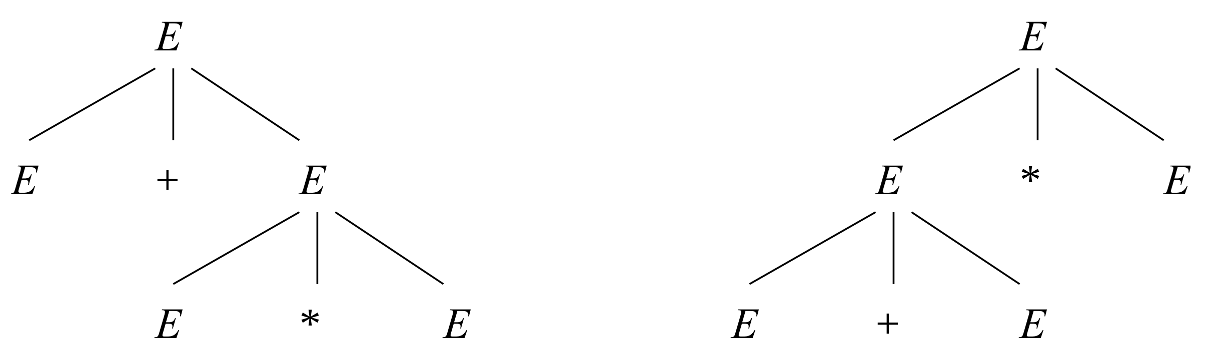
the sentential form $E + E * E$ has two derivations:

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

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.

But, multiple left-most (or right-most) derivations do cause ambiguity.

Example: In the same grammar

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

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

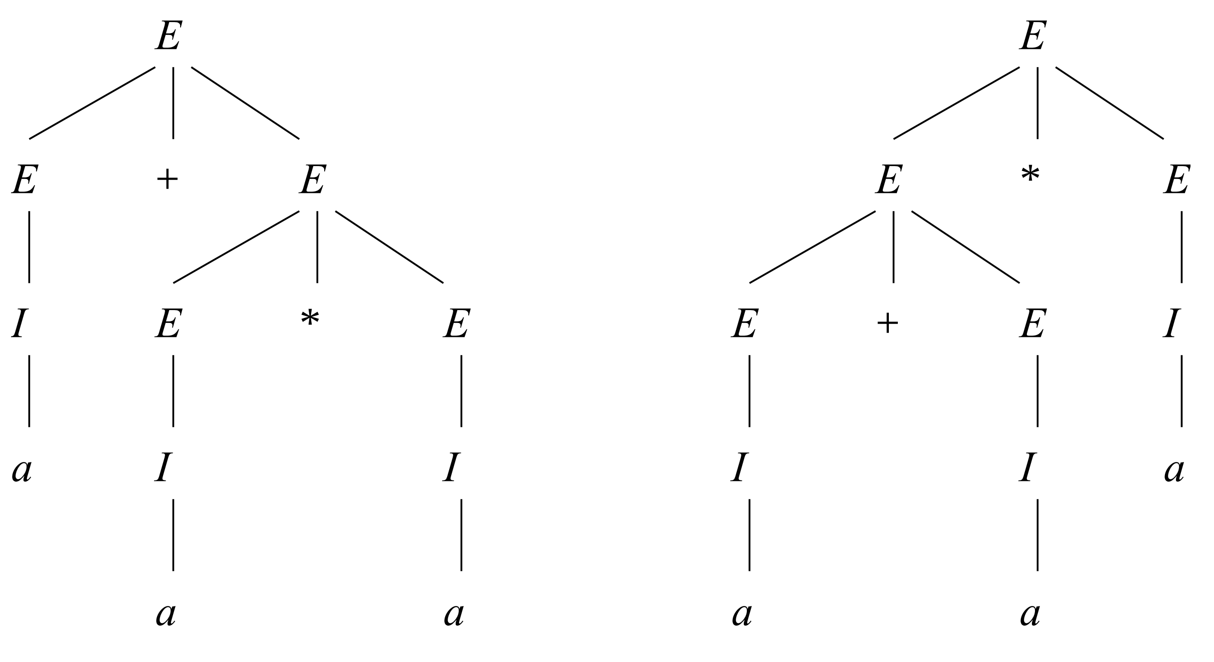
$$E \Rightarrow E + E \Rightarrow E + I \Rightarrow I + I \Rightarrow I + b \Rightarrow a + b$$

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)

Example: Unambiguous Grammar

B -> (RB | ϵ   R ->) | (RR

- Construct a unique leftmost derivation for a given balanced string of parentheses by scanning the string from left to right.
  - If we need to expand B, then use B -> (RB if the next symbol is "(" and ϵ if at the end.
  - If we need to expand R, use R ->) if the next symbol is ")" and (RR if it is "(".

The Parsing Process

Remaining Input:

(())()

Next symbol

Steps of leftmost derivation:

B

B -> (RB | ε   R ->) | (RR

The Parsing Process

Remaining Input:

())()

Next symbol

Steps of leftmost derivation:

B

(RB

B -> (RB | ϵ   R ->) | (RR

The Parsing Process

Remaining Input:

))()

Next symbol

Steps of leftmost derivation:

B

(RB

((RRB

B -> (RB | ε   R ->) | (RR

The Parsing Process

Remaining Input:

)()

Next symbol

Steps of leftmost derivation:

B

(RB

((RRB

(()RB

B -> (RB | ϵ   R ->) | (RR

The Parsing Process

Remaining Input:

()

Next symbol

Steps of leftmost derivation:

B

(RB

((RRB

(()RB

(())B

B -> (RB | ϵ R ->) | (RR

The Parsing Process

Remaining Input:

)

Next
symbol

Steps of leftmost derivation:

B

(RB

((RRB

(()RB

(())B

(())(RB

B -> (RB | ϵ   R ->) | (RR

The Parsing Process

Remaining Input:

Next
symbol

Steps of leftmost derivation:

B

(RB

((RRB

(()RB

(())B

(())(RB

(())()B

B -> (RB | ϵ

R ->) | (RR

The Parsing Process

Remaining Input:

Next
symbol

Steps of leftmost derivation:

B	(())(RB
(RB	(())()B
((RRB	(())()
(()RB	
(())B	

B -> (RB | ϵ   R ->) | (RR

LL(1) Grammars

- As an aside, a grammar such B -> (RB | ϵ R ->) | (RR, where you can always figure out the production to use in a leftmost derivation by scanning the given string left-to-right and looking only at the next one symbol is called LL(1).
  - "Leftmost derivation, left-to-right scan, one symbol of lookahead."

LL(1) Grammars – (2)

- Most programming languages have LL(1) grammars.
- LL(1) grammars are never ambiguous.

Removing Ambiguity From Grammars

Good news: Sometimes we can remove ambiguity "by hand" (without changing the language)

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.

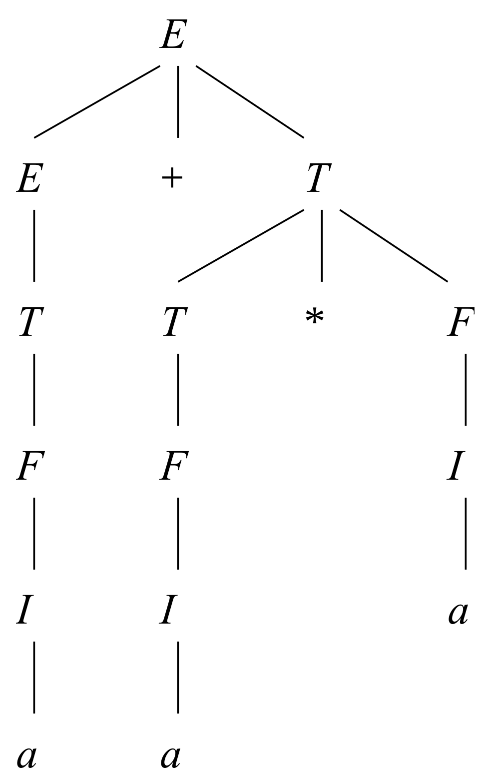
2. A *term* is an expresson that cannot be broken by $+$. For instance $a * b$ can be broken by $a1*$ or $*a1$. It cannot be broken by $+$, since e.g. $a1 + a * b$ is (by precedence rules) same as $a1 + (a * b)$, and $a * b + a1$ is same as $(a * b) + a1$.

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

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

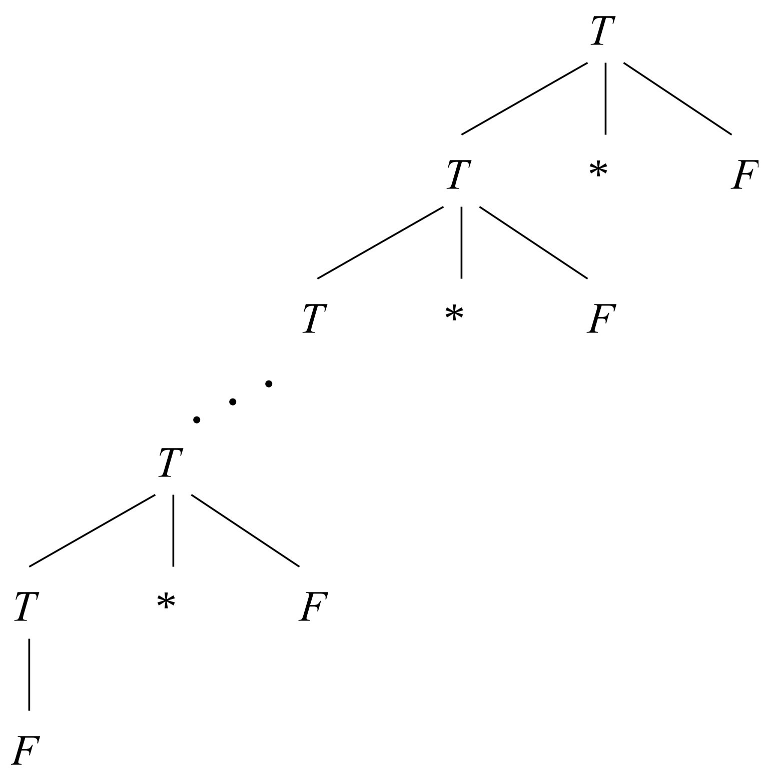
of factors is the one that gives $f_1 * f_2 * \cdots * f_{n-1}$ as a term and f_n as a factor, as in the parse tree on the next slide.

IOW, consecutive multiplications are calculated from left to right.

- An expression is a sequence

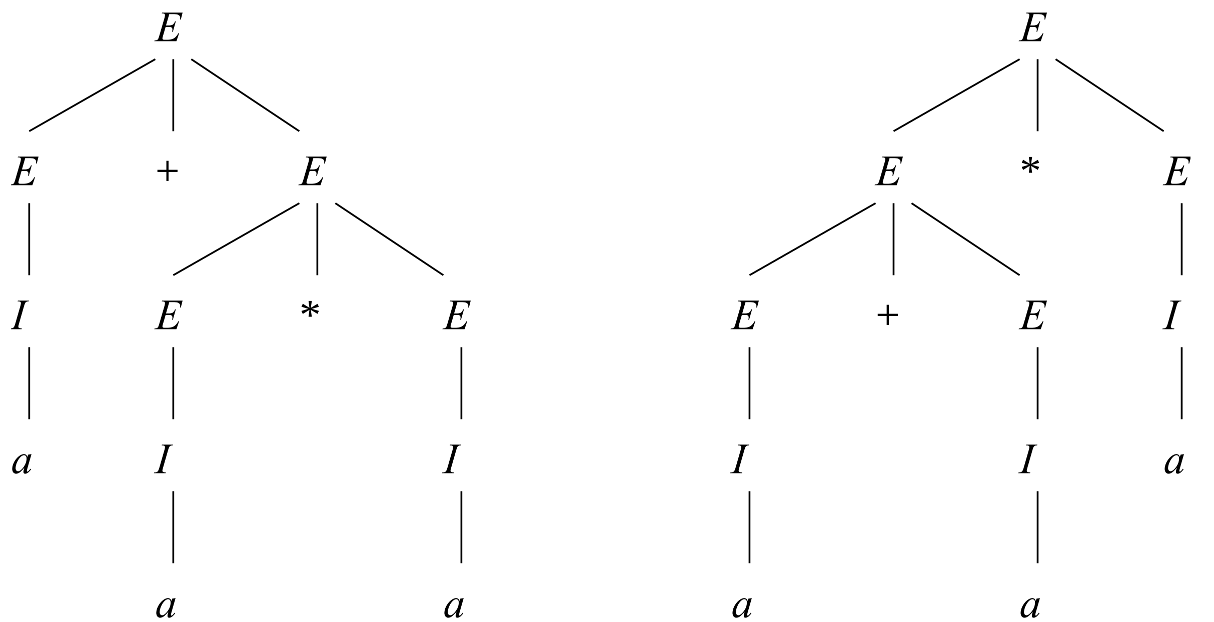
$$t_1 + t_2 + \cdots + t_{n-1} + t_n$$

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 with different productions, say $A \rightarrow X_1 X_2 \cdots X_k$ and $A \rightarrow Y_1 Y_2 \cdots Y_m$. 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.

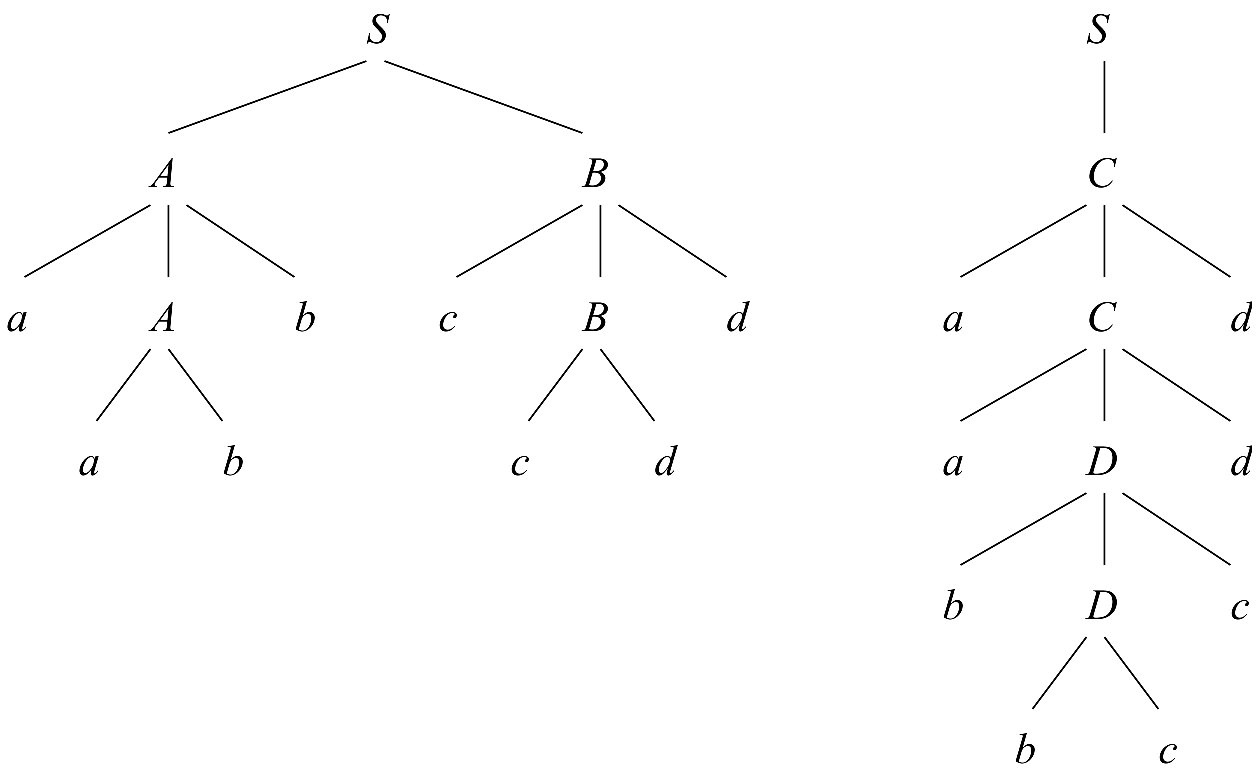
Example: Consider $L =$

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

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.

There is no algorithm to determine if a CFL is inherently ambiguous.
There is no algorithm to determine if a CFG is ambiguous.