Introduction to the Theory of Computation

XU Ming

School of Software Engineering, East China Normal University

May 11, 2024

OUTLINE

- Parse Trees for Context-Free Grammars
- Ambiguity in Context-Free Grammars

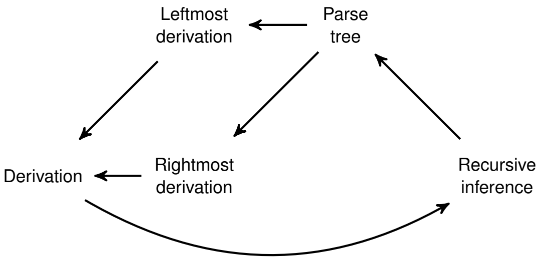
Parse Trees for Context-Free Grammars

Inferences, Derivations, and Parse Trees

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

- w is recursively infered to be in the language of A.
- $A \overset{*}{\Rightarrow} w$.
- $A \underset{lm}{\overset{*}{\Longrightarrow}} w$, and $A \underset{rm}{\overset{*}{\Longrightarrow}} w$.
- There is a parse tree of G with root A and yield w.

To prove the equivalences, we use the following plan.



Note that two of the arcs (leftmost/rightmost derivation → derivation) are very simple and will not be proved, since both leftmost and rightmost derivations are derivation.

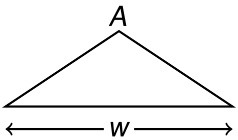
From Inferences to Trees

Theorem 5.1

Let $G = (V, T, P, S)$ be a CFG, and suppose w is shown to be in the language of a variable A by the recursive inference procedure. Then there is a parse tree for G with root A and yield w.

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

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

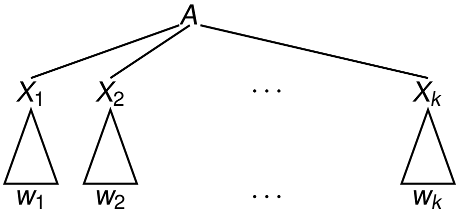


Inductive step: Suppose w is inferred in $n+1$ steps. Consider the last step of the inference. This inference uses some production for A, say $A \to X_1 X_2 \cdots X_k$, where $X_i \in V \cup T$.

We break w up as $w_1 w_2 \cdots w_k$, in which:

- if $X_i \in T$, then $w_i = X_i$;
- if $X_i \in V$, then w_i has been inferred being in X_i within n steps.

By the induction hypothesis 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:



□

Example

Consider a CFG that represent simple expressions in a typical programming language. Operators are $+$ and $\times$, and arguments are identifiers, i.e. strings in

$$L((\mathbf{a}+\mathbf{b})(\mathbf{a}+\mathbf{b}+\mathbf{0}+\mathbf{1})^*).$$

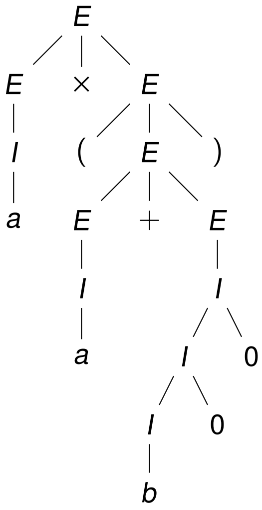
The expressions are defined by the grammar $G = (\{E, I\}, T, P, E)$ where $T = \{+, \times, (,), a, b, 0, 1\}$ and P is the following set of productions:

1. $E \to I$,
2. $E \to E + E$,
3. $E \to E \times E$,
4. $E \to (E)$,
5. $I \to a$,
6. $I \to b$,
7. $I \to Ia$,
8. $I \to Ib$,
9. $I \to I0$,
10. $I \to I1$.

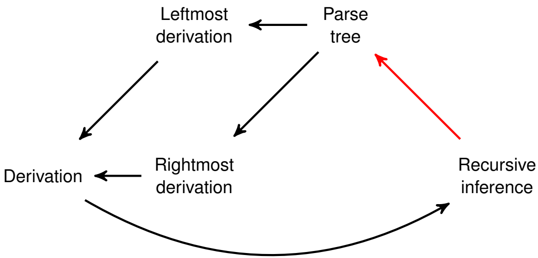
We have seen that $w = a \times (a + b00)$ is in the language of a variable E, and it is inferred in 9 steps. The last step of the inference is $E \to E \times E$.

	String	Language	Production	String(s) used
(i)	a	I	5	–
(ii)	b	I	6	–
(iii)	$b0$	I	9	(ii)
(iv)	$b00$	I	9	(iii)
(v)	a	E	1	(i)
(vi)	$b00$	E	1	(iv)
(vii)	$a + b00$	E	2	$(v), (vi)$
$(viii)$	$(a + b00)$	E	4	(vii)
(ix)	$a \times (a + b00)$	E	3	$(v), (viii)$

So, the parse tree for G with root E and yield $a \times (a + b00)$ is



To prove the equivalences, we use the following plan.



Note that two of the arcs (leftmost/rightmost derivation → derivation) are very simple and will not be proved, since both leftmost and rightmost derivations are derivation.

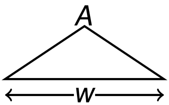
From Trees to Derivations

Theorem 5.2

Let $G = (V, T, P, S)$ be a CFG, and suppose there is a parse tree for G with root A and yield w. Then we can construct a leftmost derivation from this parse tree: $A \underset{lm}{\overset{*}{\Longrightarrow}} w$ in G.

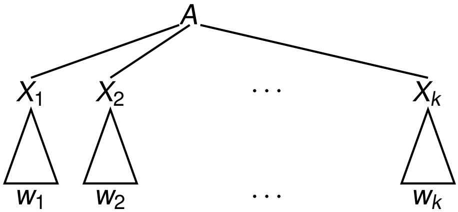
Proof We do an induction on the height of the parse tree.

Basis step: Height is 1. The parse tree must look like



Consequently $A \to w \in P$, and $A \underset{lm}{\Longrightarrow} w$.

Inductive step: Height is $n + 1$. The parse 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 $X_i \underset{lm}{\overset{*}{\Longrightarrow}} w_i$ in G by the induction hypothesis.

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

$$A \underset{lm}{\overset{*}{\Longrightarrow}} w_1 w_2 \cdots w_i X_{i+1} X_{i+2} \cdots X_k \text{ exists for each } i \geq 0.$$

When $i = k$, the result is a leftmost derivation of w from A.

Basis step: Let $i = 0$. We already know that $A \underset{lm}{\Longrightarrow} X_1 X_2 \cdots X_k$.

Inductive step: Make the induction hypothesis that

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

(Case 1): $X_i \in T$. Do nothing, since $X_i = w_i$, we have

$$A \underset{lm}{\overset{*}{\Longrightarrow}} w_1 w_2 \cdots w_{i-1} w_i X_{i+1} \cdots X_k$$

(Case 2): $X_i \in V$. By the induction hypothesis there is derivation

$$X_i \underset{lm}{\Longrightarrow} \alpha_1 \underset{lm}{\Longrightarrow} \alpha_2 \cdots \underset{lm}{\Longrightarrow} w_i$$

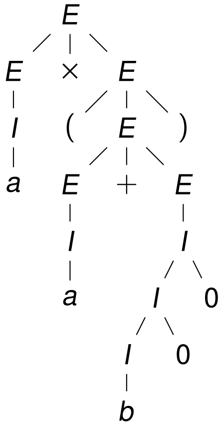
By the context-free property of derivations we can proceed with

$$
\begin{aligned}
A &\overset{*}{\underset{lm}{\Longrightarrow}} w_1 w_2 \cdots w_{i-1} X_i X_{i+1} \cdots X_k \\
&\underset{lm}{\Longrightarrow} w_1 w_2 \cdots w_{i-1} \alpha_1 X_{i+1} \cdots X_k \\
&\underset{lm}{\Longrightarrow} w_1 w_2 \cdots w_{i-1} \alpha_2 X_{i+1} \cdots X_k \qquad \cdots \\
&\underset{lm}{\Longrightarrow} w_1 w_2 \cdots w_{i-1} w_i X_{i+1} \cdots X_k
\end{aligned}
$$

In fact, it is this property that gives rise originally to the term “context-free”. There are more powerful classes of grammars, called “context-sensitive”, which do not play a major role in practice today. □

Example

Let's construct the leftmost derivation for the tree



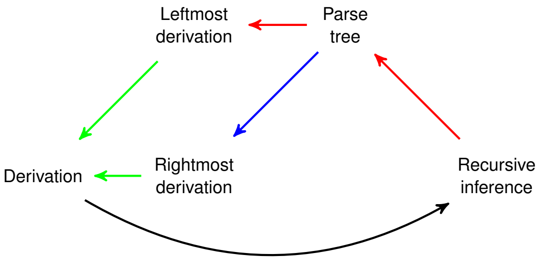
Suppose we have inductively constructed the leftmost derivation $E \underset{lm}{\Longrightarrow} I \underset{lm}{\Longrightarrow} a$ corresponding to the leftmost subtree; and the leftmost derivation $E \underset{lm}{\Longrightarrow} (E) \underset{lm}{\Longrightarrow} (E+E) \underset{lm}{\Longrightarrow} (I+E) \underset{lm}{\Longrightarrow} (a+E) \underset{lm}{\Longrightarrow} (a+I) \underset{lm}{\Longrightarrow} (a+I0) \underset{lm}{\Longrightarrow} (a+I00) \underset{lm}{\Longrightarrow} (a+b00)$ corresponding to the rightmost subtree.

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

$$E \underset{lm}{\Longrightarrow} E \times E \underset{lm}{\Longrightarrow} I \times E \underset{lm}{\Longrightarrow} a \times E \underset{lm}{\Longrightarrow} a \times (E) \underset{lm}{\Longrightarrow} a \times (E+E) \underset{lm}{\Longrightarrow} a \times (I+E) \underset{lm}{\Longrightarrow}$$

$$a \times (a+E) \underset{lm}{\Longrightarrow} a \times (a+I) \underset{lm}{\Longrightarrow} a \times (a+I0) \underset{lm}{\Longrightarrow} a \times (a+I00) \underset{lm}{\Longrightarrow} a \times (a+b00).$$

To prove the equivalences, we use the following plan.



Note that two of the arcs (leftmost/rightmost derivation → derivation) are very simple and will not be proved, since both leftmost and rightmost derivations are derivation.

From Derivations to Inferences

Theorem 5.3

Let $G = (V, T, P, S)$ be a CFG. Suppose $A \underset{lm}{\overset{*}{\Longrightarrow}} w$, 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 $A \underset{G}{\overset{*}{\Longrightarrow}} w$.

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

Observation: Suppose that $A \Rightarrow X_1 X_2 \cdots X_k \overset{*}{\Rightarrow} w$. Then $w = w_1 w_2 \cdots w_k$, where $X_i \overset{*}{\Rightarrow} w_i$. The factor w_i can be extracted from $A \overset{*}{\Rightarrow} w$ by looking at the expansion of X_i only.

For example, $E \Rightarrow \underbrace{E}_{X_1} \underbrace{\times}_{X_2} \underbrace{E}_{X_3} \underbrace{+}_{X_4} \underbrace{E}_{X_5} \overset{*}{\Rightarrow} a \times b + a$.

We have $E \Rightarrow E \times E + E \Rightarrow I \times E + E \Rightarrow I \times I + E \Rightarrow I \times I + I \Rightarrow a \times I + I \Rightarrow a \times b + I \Rightarrow a \times b + a$.

By looking at the expansion of $X_3 = E$ only, we can extract $E \Rightarrow I \Rightarrow b$.

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

$$A \Rightarrow X_1 X_2 \cdots X_k \overset{*}{\underset{G}{\Rightarrow}} w$$

Then as noted on the previous slide we can break w as $w_1 w_2 \cdots w_k$ 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 $A \rightarrow X_1 X_2 \cdots X_k$, and we know by the induction hypothesis that we can infer w_i to be the language of X_i.

Therefore we can infer $w_1 w_2 \cdots w_k$ to be in the language of A. $\square$

Ambiguity in Context-Free Grammars

Ambiguous Grammars

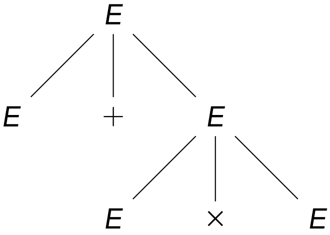
In the grammar

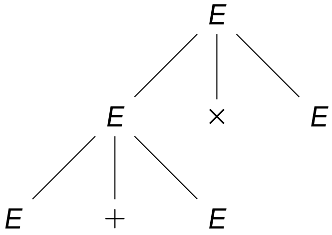
1. $E \to I$,	5. $I \to a$,
2. $E \to E + E$,	6. $I \to b$,
3. $E \to E \times E$,	7. $I \to Ia$,
4. $E \to (E)$,	8. $I \to Ib$,
	9. $I \to I0$,
	10. $I \to I1$.

the sentential form $E + E \times E$ has two derivations:

$$E \Rightarrow E + E \Rightarrow E + E \times E \text{ and } E \Rightarrow E \times E \Rightarrow E + E \times E$$

This gives us two parse trees:





This grammar is not a good one for providing unique structure. To use this expression grammar in a compiler, we would have to modify it to provide only the correct groupings.

In the same grammar 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.

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

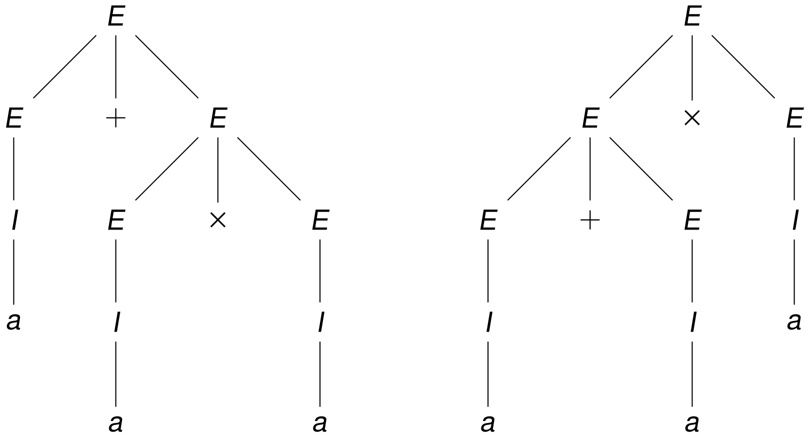
The two examples above suggest that it is not a multiplicity of derivations that cause ambiguity, but rather the existence of two or more parse trees.

Let $G = (V, T, P, S)$ be a CFG. We say that G is ambiguous if there is at least one string w in T^* for which we can find two different parse trees, each with root labeled S and yield w.

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

Example

In above grammar, the terminal string $a + a \times a$ has two parse trees:



Removing Ambiguity from Grammars

- Good news: Sometimes we can remove ambiguity from CFG's "by hand".
- Bad news: There is no algorithm to do it.
- Worse news: There are context-free languages that have nothing but ambiguous CFG's; for these languages, removal of ambiguity is impossible.

Fortunately, in practice, for the sorts of constructs that appear in common programming languages, there are well-known techniques for eliminating ambiguity.

As an important illustration, we are studying the grammar

$$E \to I \mid E + E \mid E \times E \mid (E), \quad I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1.$$

There are two causes of ambiguity:

1. There is no precedence between $\times$ and $+$.
2. There is no grouping of sequences of operators, e.g. is $E + E + E$ meant to be $E + (E + E)$ or $(E + E) + E$.

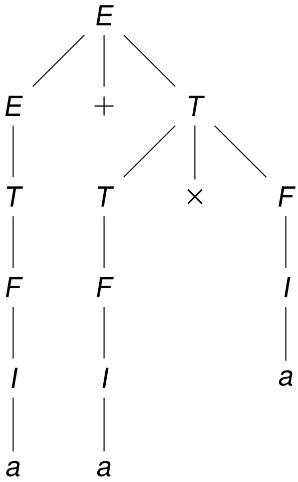
The solution to the problem of enforcing precedence is to introduce more variables, each representing expressions of same "binding strength".

1. A *factor* is an expression that cannot be broken apart by an adjacent $\times$ or $+$. Our factors are identifiers and a parenthesized expression.
2. A *term* is an expression that cannot be broken by $+$. For instance $a \times b$ can be broken by $a1\times$ or $\times a1$, $a1 \times a \times b = (a1 \times a) \times b$. It cannot be broken by $+$, since e.g. $a1 + a \times b$ is (by precedence rules) same $a1 + (a \times b)$, and $a \times b + a1$ is same as $(a \times b) + a1$.
3. The rest are *expressions*, i.e. they can be broken apart with $\times$ 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. $T \rightarrow F \mid T \times F$
4. $E \rightarrow T \mid E + T$

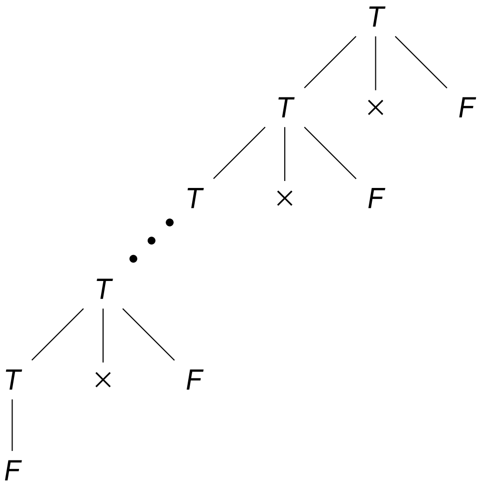
Now the only parse tree for $a + a \times a$ will be



Why is the new grammar unambiguous?

Here is an intuitive explanation:

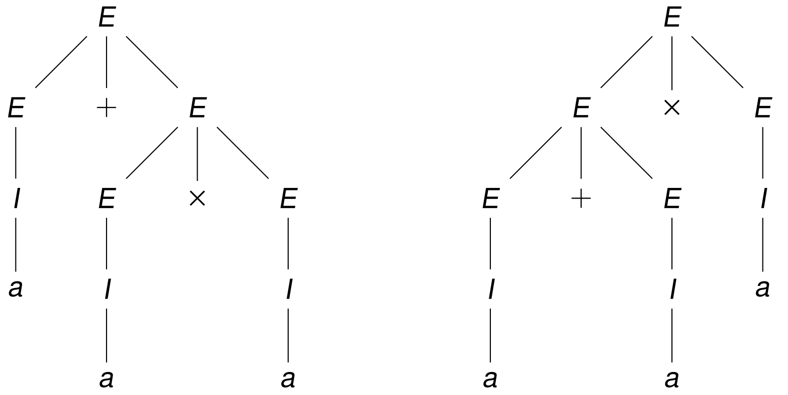
- A factor is either an identifier or (E), for some expression E.
- The only parse tree for a sequence $f_1 \times f_2 \times \cdots \times f_{n-1} \times f_n$ of factors is the one that gives $f_1 \times f_2 \times \cdots \times f_{n-1}$ as a term and f_n as a factor, as in the parse tree on the next 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

Example

The terminal string $a + a \times a$ has two parse trees:



That gives rise to two derivations

$$
\begin{aligned}
E &\underset{lm}{\Longrightarrow} E + E \underset{lm}{\Longrightarrow} I + E \underset{lm}{\Longrightarrow} a + E \underset{lm}{\Longrightarrow} a + E \times E \underset{lm}{\Longrightarrow} a + I \times E \\
&\underset{lm}{\Longrightarrow} a + a \times E \underset{lm}{\Longrightarrow} a + a \times I \underset{lm}{\Longrightarrow} a + a \times a
\end{aligned}
$$

and

$$
\begin{aligned}
E &\underset{lm}{\Longrightarrow} E \times E \underset{lm}{\Longrightarrow} E + E \times E \underset{lm}{\Longrightarrow} I + E \times E \underset{lm}{\Longrightarrow} a + E \times E \underset{lm}{\Longrightarrow} a + I \times E \\
&\underset{lm}{\Longrightarrow} a + a \times E \underset{lm}{\Longrightarrow} a + a \times I \underset{lm}{\Longrightarrow} a + a \times a
\end{aligned}
$$

In general, there may be many derivations for one parse tree. But the diversity of leftmost/rightmost derivations implies that of parse trees.

Theorem 5.4

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.

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

(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. □

Inherent Ambiguity

A CFL L is inherent ambiguous if all grammars for L are ambiguous.

Example

Consider $L = \{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$.

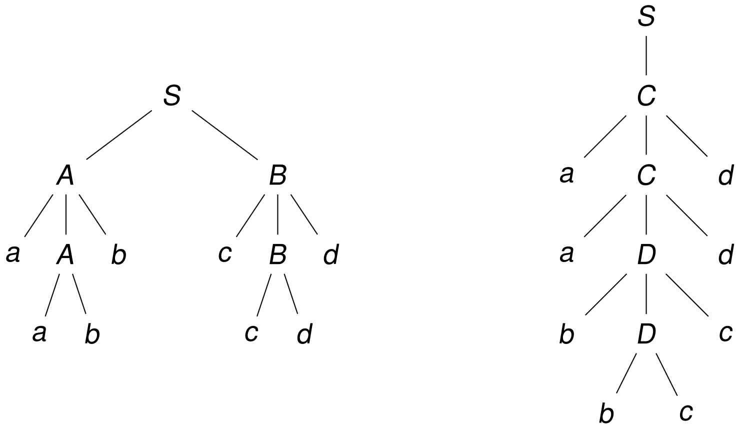
A grammar for L is

$$G = (\{A, B, C, D, S\}, \{a, b, c, d\}, P, S)$$

where P is as follows:

$$S \to AB \mid C,\ A \to aAb \mid ab,\ B \to cBd \mid cd,\ C \to aCd \mid aDd,\ D \to bDc \mid bc$$

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



From this we see that there are two leftmost derivations:

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

and

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

It can be shown that every grammar for L behaves like the one above. The proof is complex.

The language L is inherently ambiguous.

Homework

Exercises 5.2.2, 5.4.3