Def: A parse tree for a string $x \in L(G)$ or a sentential form $x \in (\Sigma^*)$ is a tree with internal nodes labeled by variables such that for each internal node with label $A \in \mathcal{V}$, its children have labels $U_1, \ldots, U_m$ such that $A \rightarrow U_1 \ldots U_m$ is a rule in the CGF $G$ and finally the leaves in left-to-right order form $x$. For example, $G = S \rightarrow SS \mid (S) \mid \varepsilon$, $x = (\) (\) (\) (\) (\) (\) (\) (\) (\) (\) (\) (\) (\)}
Defn: A string $x$ (or $X$) is ambiguous (in $G$) if it has two or more distinct parse trees. Else it is unambiguous (in $G$). $E$, $($, $)$, are unambiguous, while $S$ and $SS$, but $SSS$ is ambiguous.

Observation: Every parse tree gives a unique leftmost derivation in which every step expands the leftmost variable. For $x$:

$T_1$: $S \Rightarrow SS \Rightarrow SSS \Rightarrow ((S)S)S \Rightarrow (S)(S)S \Rightarrow ((S)(S)S \Rightarrow (S)(S)(S) \Rightarrow (S)(S)(S) = x$.

LM deriv using $T_2$: $S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow (S)(S)(S) \Rightarrow (S)(S)(S)$

Fact: $x$ is ambiguous in $G$ if and only if $x$ has two distinct leftmost derivations (iff $x$ has two distinct RM derivs).

I.e. Parse trees are in 1-1 correspondence with LM derivations (and in 1-1 correspondence with RM derivs).

And Every derivation gives a unique parse tree.

Final Defn: A CFG $G$ is ambiguous if there is some ambiguous $x \in L(G)$ (or sentential form $x_0 \Rightarrow \ast \Rightarrow \ast \Rightarrow \ast$).

Defn: For every $A \in V$, we can define $L_A = \{x \in \Sigma^* : A \Rightarrow^* x \}$.

$A$ is live if $L_A \neq \emptyset$, else $A$ is dead. If no variables are dead, then these two defns of ambiguity are equivalent. Finally, a language $L$ is inherently ambiguous if every CFG $G$ st. $L(G) = L$ is ambiguous.
A typical expression grammar

Let \( \Sigma = \{a, b, \} \) but let \( \Gamma = \{\}, \), and the rules

\[
G = \{ E \rightarrow (E) | E + E | E \cdot E | E^* | a | b | \epsilon | \emptyset \}
\]

\( L(G) = \{ \text{"liberal" textual representation of regular expressions over } \Sigma \} \)

Note: the grammar allows \((a+b) \cdot a\cdot a\)
but this is a different string \( x \in L(G) \)

\( a + b \cdot a \cdot a \)

We can also get \( x \) by a parse tree that respects the actual reading:

Hence \( G \) is ambiguous (many trees over), but happily, \( L(G) \) is not inherently ambiguous...

An "overkill": \( \forall x \in L(G) \):  

\[
E \rightarrow (E + E) | (E \cdot E) | (E^*) | a | b | \epsilon | \emptyset
\]

\( L(G_2) = \{ \text{fully parenthesized regexes} \} \neq L(G) \).  

Eg: \( x = (a + ((ba) \cdot a)) \)
An unambiguous grammar for regexps the way we write them, using three "categories":

\[
G_3: \quad E \rightarrow T | E + T \\
T \rightarrow F | T \cdot F \\
F \rightarrow a | b | e | \emptyset | (E) | F^* 
\]

Let us try to derive \(a + b a a\) in \(G_3\).

Now try to group \((a + b) - a - a\). Can we do it without parentheses?

\[
\begin{align*}
E + T \\
E &+ T \\
E &+ T \\
F &+ T \\
F &
\end{align*}
\]

\[a + b - a - a = x\]

**Added:** The "\(E \rightarrow T + F\)" design pattern is used to define much wider hierarchies of operators in many programming languages. My lecture stopped short of the "left-associative" vs. "right-associative" issue, which comes into play when - and / (etc.) are operators and which you should see in cases. The big "gotcha" is that \(a/bc\) parses as \((a/b)c\) when you probably intended \(a/(bc)\) as it writing \(a/(bc)\).

**Conclusion:** to group \((a+b)/a a\) the grammar will force the parentheses. That's intuitively why it is unambiguous but gives the same language as \(G_1\).