Ambiguity

Def: A grammar is ambiguous if there is a string w that can be derived using the grammar using two different derivation trees.

E.g. \( \Sigma = \{a, b, c, (, ), *, +\} \)

\( L = \{ \text{algebra expressions w/a's, b's, c's as variables} \} \)

Grammar:

\( S \rightarrow a|b|c|(S)|S*S|S+S \)

Consider the string

\( a*b+c \)

This has 2 derivation trees:
We would like to have a grammar that is unambiguous - has only one parsing (i.e. one derivation tree) (and we'd like the one the forces the parsing that agrees w/ usual semantics)

Variables:

Expression (start = variable)
Factor
Term (an expression is a sum of terms - term = summand)
Factor
\[ E \rightarrow T \mid T + E \]
\[ T \rightarrow F \mid F * T \]
\[ F \rightarrow a \mid b \mid c \mid (E) \]

Parse \( a * b + c \)
Unfortunately, there are some context-free languages that have no unambiguous grammars — every grammar for it will be ambiguous. These languages are said to be inherently ambiguous.

A property of a language, not of a grammar.
(Note — it is possible to unnecessarily add ambiguity to a grammar — so it makes no sense to talk about languages where every grammar is unambiguous.)

E.g.: \{am \, bm \, cm\} \cup \{ak \, bm \, cm\}

(either \#a's = \#b's or \#b's = \#c's)

Grammar:

\[
\begin{align*}
S & \rightarrow \text{ } \text{ } XC \midAY \quad \text{Any string} \\
X & \rightarrow aXb \mid \text{ab} \\
Y & \rightarrow bYc \mid bc \\
C & \rightarrow cC \mid c \\
A & \rightarrow aA \mid a
\end{align*}
\]
Modifying grammars (in ways that don't change the language)

Goal: We want to study some algorithms that tell us things about a context-free language. To make it easier to describe the alg, we want to have only certain kinds of productions allowed. We want modifications to grammars to let us replace unallowed kinds of productions with allowed productions, (w/o changing what the grammar produces)

General idea: if we have productions

\[ A \rightarrow B \]

and \[ B \rightarrow \text{all the prods for } B \]
We can replace

\[ A \rightarrow _{-} B _{-} \]

with

\[ A \rightarrow _{-} _{-} _{-} _{-} _{-} _{-} _{-} \]

Removing useless productions/variables.

E.g. \( S \rightarrow aA | baB | abc \)
\( A \rightarrow bB | aS \)
\( B \rightarrow caB | baA | bS | cb \)
\( C \rightarrow abA | bC | aD \)
\( D \rightarrowSacB \)

Anything you can't reach from the start state is useless.

(We can use depth-first or breadth-first search to mark all the useful vars starting at \( S \) - the rest are useless.)
Another kind of uselessness:

\[ S \rightarrow AB \mid bA \mid BCa \mid ab \]
\[ A \rightarrow aS \mid bB \mid aAB \mid bc \]
\[ B \rightarrow aB \mid bC \mid aBC \]
\[ C \rightarrow bB \mid aC \mid BCAa \]

It's not possible to make B's or C's all go away.