Parsing

Parsing is finding a derivation of $w$ from a grammar $G$, if it exists.

CFGs have $O(n^3)$ parsing algorithm, $n = |w|$. Later, we will discuss the CYK algorithm.

Some types of CFGs are more efficient.
An s-grammar has productions $A \rightarrow ax$, where $a \in T$, $x \in V^*$, and it is illegal to have both $A \rightarrow ax$ and $A \rightarrow ay$.

Example: Balanced Parentheses within ()

\[
S \rightarrow (R \\
R \rightarrow (RR \\
R \rightarrow )
\]

S-grammars have an $O(n)$ parsing algorithm. In a leftmost derivation, only one choice for substituting leftmost variable.
Ambiguity

A CFG is *ambiguous* if there is some string \( w \) that has more than one derivation tree.

Example: Arithmetic Expressions

\[
E \rightarrow V \\
E \rightarrow EOE \\
E \rightarrow (E) \\
V \rightarrow a \mid b \mid c \\
O \rightarrow + \mid - \mid * \mid /
\]

\[
E \rightarrow T \mid EPT \\
T \rightarrow F \mid TMF \\
F \rightarrow (E) \mid V \\
V \rightarrow a \mid b \mid c \\
P \rightarrow + \mid - \\
M \rightarrow * \mid /
\]

A CFL \( L \) is *unambiguous* if some unambiguous CFG generates \( L \). Otherwise, \( L \) is *inherently ambiguous*.

How do you show that a CFG is ambiguous? How do you show that a CFG is unambiguous?
CYK Algorithm

CYK is a general algorithm for parsing CFGs.

For a grammar $G$ in Chomsky normal form, the CYK algorithm is $O(l + m^2n^3)$, where $l = |G|$, $m = |V|$, $n = |w|$.

In Chomsky normal form, all productions have the form $A \rightarrow a$ or $A \rightarrow BC$, where $A, B, C$ are variables and $a$ is a terminal.

Any CFG that does not derive $\lambda$ can be converted to Chomsky normal form.

Logic of CYK Algorithm

The CYK algorithm is an example of dynamic programming.

Goal: $V[i, i + d] = \{ A : A \Rightarrow w[i] \ldots w[i + d] \}$.

To determine $V[i, i]$, find the variable symbols such that $A \rightarrow w[i]$.

To determine $V[i, i + d]$, find $j$, $A$, $B$, and $C$ such that $A \rightarrow BC$, $B \in V[i, j]$, and $C \in V[i + j + 1, i + d]$. 
procedure CYK(G, w)
    Initialize all entries in V to ∅
    for i ← 1 to |w|
        V[i, i] ← {A : A → w[i]}
    for d ← 1 to |w| − 1
        for i ← 1 to |w| − d
            for j ← 0 to d − 1
                for B in V[i, i + j]
                    for C in V[i + j + 1, i + d]
                        V[i, i + d] ← V[i, i + d] ∪ {A : A → BC}
    return V