**Parse Trees**

Example.

\[ G = (V, \Sigma, R, s) \]

\[ \Sigma = \{ , \} \]  \[ V - \Sigma = \{ s \} \]

\[ R = \{ s \rightarrow sss | (s) \} \]

Look at some derivations:

\[
\begin{align*}
S & \rightarrow SS \\
SS & \rightarrow SS \\
(s)S & \rightarrow (s)(s) \\
(s) & \rightarrow ( ) \\
( ) & \rightarrow ( )
\end{align*}
\]

All these derivations are in a sense "the same". They use the same rules; the only difference is the "order" in the string where they are applied.
Intuitively we picture them as a **PARSE TREE**

- **ROOT**
- **NODES**
  - Each node has a **LABEL**
  - **LEAVES**—labeled by
  - **TERMINALS**, or possibly **ε**

By **concatenating** the **LABELS** from left to right, we obtain the derived string of **terminals**, called the **YIELD** of the parse tree.

**Derivations**:

\[
S \Rightarrow SS \Rightarrow (s)(s) \Rightarrow (s)s \Rightarrow (s)s \Rightarrow s \Rightarrow (s)(s) \Rightarrow (s)(s)
\]
\[ S \Rightarrow SS \Rightarrow SS(S) \Rightarrow SS(S)(S) \Rightarrow S(S)(S)(S) \Rightarrow (S)(S)(S)(S) \]

\[ (S)(S)(S)(S) \Rightarrow (S)(S)(S)(S) \Rightarrow (S)(S)(S)(S) \]

You can read all possible derivations of \((S)(S)(S)(S)\) from the parse tree.
Example

\[ G = (\{ S, a \}, \{ a \}, \{ S \rightarrow SS | aS, S \}, S) \]

Two derivations of \( aa + L(G) \)

\[ S \rightarrow SS \rightarrow aS \rightarrow aa \]
\[ S \rightarrow SS \rightarrow Sa \rightarrow aa \]

**Parse Tree**

Derivation

\[ S \rightarrow SS \rightarrow Sa \rightarrow Ssa \rightarrow aSa \rightarrow aaa \]

**Parse Tree**

Exercise: Read all possible derivations of \( aaa \)
Example

\[ G = \{ \{ S \rightarrow SS \}, \{ S \rightarrow (S) \}, \{ S \rightarrow (S) | (S) \rightarrow e \} \} \]

\[ D_1 = S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow ((S))(S) \Rightarrow ((S))(S) \Rightarrow ((S))(S) \Rightarrow (())(()) \]

\[ D_2 = S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow ((S))(S) \Rightarrow ((S))(S) \Rightarrow ((S))(S) \Rightarrow (())(()) \]

\[ D_3 = S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow ((S))(S) \Rightarrow ((S))(S) \Rightarrow ((S))(S) \Rightarrow (())(()) \]

**Parse Tree**

\[ D_1, D_2, D_3 \text{ have the same parse tree} \]
More derivations described by the parse tree (of $D, D_2, D_3$)

```
D_4 = S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow ((S))(S) \Rightarrow (())(S) \Rightarrow (())()
D_5 = S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow ((S))(S) \Rightarrow (())() \Rightarrow (())()
D_6 = S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow (S)(()) \Rightarrow ((S))(()) \Rightarrow (())()
D_7 = S \Rightarrow SS \Rightarrow S(S) \Rightarrow (S)(S) \Rightarrow ((S))(S) \Rightarrow (())(S) \Rightarrow (())()
D_8 = S \Rightarrow SS \Rightarrow S(S) \Rightarrow (S)(S) \Rightarrow ((S))(S) \Rightarrow (())() \Rightarrow (())()
D_9 = S \Rightarrow SS \Rightarrow S(S) \Rightarrow (S)(S) \Rightarrow (S)() \Rightarrow ((S))() \Rightarrow (())()
D_{10} = S \Rightarrow SS \Rightarrow S(S) \Rightarrow (S)(S) \Rightarrow (S)() \Rightarrow ((S))() \Rightarrow (())()
```
Derivation

\[ s \rightarrow ss \rightarrow sss \rightarrow s(s)s \rightarrow s((s)s) \rightarrow s((1)s) \rightarrow s((1))(1) \rightarrow s((1))(1) \]

has a parse tree

ID Note: Parse trees are ways of representing derivations, so that representing differences due to the order of application of rules are suppressed.
Pare trees represent equivalence classes of derivations.

**Def.** Two derivations are equivalent if they can be transformed into another via a sequence of "switchings" in the order in which rules applied. Such "switching" can replace a derivation either by one that precedes it, or by the one that it precedes.

[\mathcal{D}(1)(1)] = \{D_1, D_2, \ldots, D_{10}\} = \{D_i\}

The all represent applications of the same rules at the same positions in the strings, only differing in the relative order of these applications.

- Define: \(D < D'\) (use \(\leq\) defined).
Each equivalence class of derivations (ex: $D_{(11)}(1)$) that is to say, **EACH PARSE TREE** contains a derivation, that is **MAXIMAL** under order $\leq$.

In our example, it is $D_{11}$.

Then, **THIS MAXIMAL DERIVATION EXISTS IN EACH PARSE TREE** and is called a **LEFT MOST DERIVATION**.

Algorithm for **LMD**:

Start at the root label, repeatedly replace the leftmost nonterminal in the current string according to the rule defined by the parse tree.
Parse tree

RMD:
- Start at root
- Replace rightmost nonterminal
- Repeat

LMD:
- LMD - maximal (MAX)
- RMD - minimal (MIN)
- Both are unique

D1:
- We define similarly the rightmost derivation

RMD
Theorem

Let $G = (V, \Sigma, R, S)$ be a CFG and let $A \in V - \Sigma$, $w \in \Sigma^*$.

The following conditions are equivalent:

(a) $A \xrightarrow{*} w$

(b) There is a PARSE TREE with root $A$ and yield $w$.

(c) There is a LEFTMOST derivation

$$A \xrightarrow{L} w$$

(d) There is a RIGHTMOST derivation

$$A \xrightarrow{R} w$$
Ambiguity

We had two parse trees generating derivations of (())():

T₁

S → S S
S → (S)
S → ε

D₁...
Dₙ

T₂

S → S S
S → (S)
S → ε

WRITE all derivations a T₂

Grammars such that they have strings with two or more distinct parse trees are called ambiguous.
Def: $G$ is ambiguous if there is a $w \in L(G)$ such that $w$ has two at least two different parse trees.
Example

\[ G_1 = (\{+,-,(), id, E\}, \{+,-,(), id\}, E, \{E \rightarrow E+E \mid E \cdot E \mid (E) \mid id\}) \]

This is a simpler grammar, which generates the same language as

\[ G = \{E \rightarrow E+T | T \rightarrow T*F | T \rightarrow F | F \rightarrow (E) | id\} \]

\[ L(G) = LCG_1 \]

\[ G \text{ is NOT AMBIGUOUS} \]

\[ G_1 \text{ is AMBIGUOUS} \]

There are two parse trees "like" corresponding to

\[ T_1 \quad \text{id + id \cdot id} \]

\[ T_2 \quad (id + id) \cdot id \]

\[ \text{These rules are more natural.} \]
What ambiguity mean?

We can assign different meanings to our expressions by different parse trees.

\[ n + (km) \]
PARSING a string - assigning a parse tree to a given string of a language is an important step towards understanding the structure of the string - which defines why the string belongs to a language. Parse tree defines "a meaning" of a string.

Ambiguous grammars are of no help in parsing - no unique meaning. Some grammars like $G_1$ can be made unambiguous ($G$). Our derivation grammar is ambiguous, but can be made unambiguous for programming languages, we want unambiguous grammars.

$L$ is inherently ambiguous if all $G$, such $L = L(G)$ are ambiguous.

Programming languages are never inherently ambiguous.