Syntax analysis (Ch. 2 [M])

Takes a list of tokens, gives a syntax tree

<u>context-free grammars</u>

$N \longrightarrow X_1 \ldots X_n$ production (rule, or production rule)

<u>one</u> nonterminal symbol on the LHS

a^+

$A \longrightarrow \mathrm{a}$

terminal symbol

$A \longrightarrow \mathrm{a}A$

a

a a

a a a

a ... a

optionally

a^* $\begin{cases} B \to \varepsilon \\ B \to aB \end{cases}$

$\varepsilon, a, aa, aaa, \ldots$

s^*	$\{\text{``''}\} \cup \{vw \mid v \in \mathrm{L}(s), w \in \mathrm{L}(s^*)\}$	Each string in the language is a concatenation of any number of strings in the language of s.

Up to this point we have given examples of c.f. languages that are also regular languages, for example:

$S \to \varepsilon$

$S \to aSb$

$\{a^n b^n \mid n \geq 0\} = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$

(2)
several nonterminals : T and R

start (nonterminal) symbol

$$
\begin{aligned}
T &\rightarrow R \\
T &\rightarrow \mathrm{a}T\mathrm{a} \\
R &\rightarrow \mathrm{b} \\
R &\rightarrow \mathrm{b}R
\end{aligned}
$$

b^+

disjunction

$$
\begin{aligned}
T &\rightarrow R \mid \mathrm{a}T\mathrm{a} \\
R &\rightarrow \mathrm{b} \mid \mathrm{b}R
\end{aligned}
$$

a shorthand for this

In EBNF:

T -> b+ | a T a

context free grammar, using regexp notation

You may use +, *, ?

Form of s_i	Productions for N_i
ε	$N_i \longrightarrow$
$\texttt{a}$	$N_i \longrightarrow \texttt{a}$
$s_j s_k$	$N_i \longrightarrow N_j N_k$
$s_j \vert s_k$	$N_i \longrightarrow N_j$ $N_i \longrightarrow N_k$
s_j*	$N_i \longrightarrow N_j N_i$ $N_i \longrightarrow$
s_j+	$N_i \longrightarrow N_j N_i$ $N_i \longrightarrow N_j$
$s_j?$	$N_i \longrightarrow N_j$ $N_i \longrightarrow$

Fig. 2.1 From regular expression to context-free grammars

Syntactic category: constructs of a (programming) language that differ in meaning. Three typical ones:

Expressions: are evaluated to yield a value

Commands: are executed to change memory or perform I/O

Declarations: define properties of names used in other parts of the program

expressions without parentheses and precedence; can be described by a regular expression

num((+|-|*|/)**num**)*

an ambiguous grammar

$$
\begin{array}{lcl}
Exp & \rightarrow & Exp + Exp \\
Exp & \rightarrow & Exp - Exp \\
Exp & \rightarrow & Exp * Exp \\
Exp & \rightarrow & Exp \,/\, Exp \\
Exp & \rightarrow & \mathbf{num} \\
Exp & \rightarrow & (Exp)
\end{array}
$$

2, 5+2-3, 3-6*5 4/5*2

✓ (3-6)*5 ✗

Each syntactic category is denoted by a nonterminal, such as Exp in the grammar above.

Stat -> **id** := *Exp* assignment

Stat -> *Stat* ; *Stat* sequence (list) of statements

Stat -> `if` *Exp* `then` *Stat* `else` *Stat* two-way conditional

Stat -> `if` *Exp* `then` *Stat* one-way conditional

A grammar for (simple) statements

Ex. 2,3 [M] (first part)

$$P \to \varepsilon$$
$$P \to (P)$$
$$P \to P\,P$$

ε, $()$, $(())$, $()()$, $(())()$, $()(())$, ...

✗ $(()$

✗ $()())$

Derivation $\Rightarrow$ or, sometimes ::=

"may be rewritten as"

(rewrite)

1. $\alpha N \beta \Rightarrow \alpha\gamma\beta$ if there is a production $N \longrightarrow \gamma$
2. $\alpha \Rightarrow \alpha$ (reflexivity)
3. $\alpha \Rightarrow \gamma$ if there is a β such that $\alpha \Rightarrow \beta$ and $\beta \Rightarrow \gamma$ (transitivity)

Definition 3.1 *Given a context-free grammar G with start symbol S, terminal symbols T and productions P, the language L(G) that G generates is defined to be the set of strings of terminal symbols that can be obtained by derivation from S using the productions P,* i.e., *the set* $\{w \in T^* \mid S \Rightarrow w\}$.

$\Rightarrow$ for one step in a derivation

$\overset{*}{\Rightarrow}$ for what Mogensen calls a derivation.

sentential form

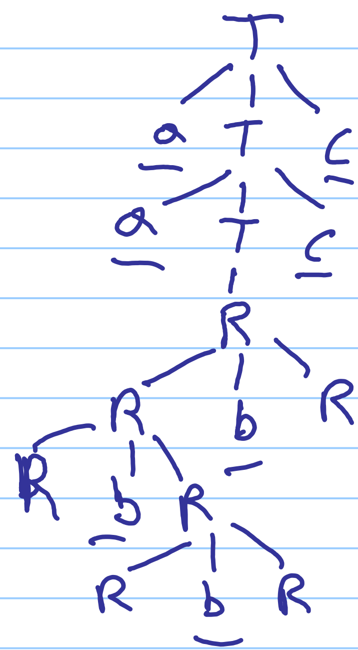
$P \Rightarrow (P) \Rightarrow ((P)) \Rightarrow \underbrace{(())}_{\text{sentence}}$

$$
\begin{aligned}
T &\rightarrow R \\
T &\rightarrow \mathrm{a}T\mathrm{c} \\
R &\rightarrow \\
R &\rightarrow R\mathrm{b}R
\end{aligned}
$$

ac

$T \Rightarrow aTc \Rightarrow aRc \Rightarrow ac$

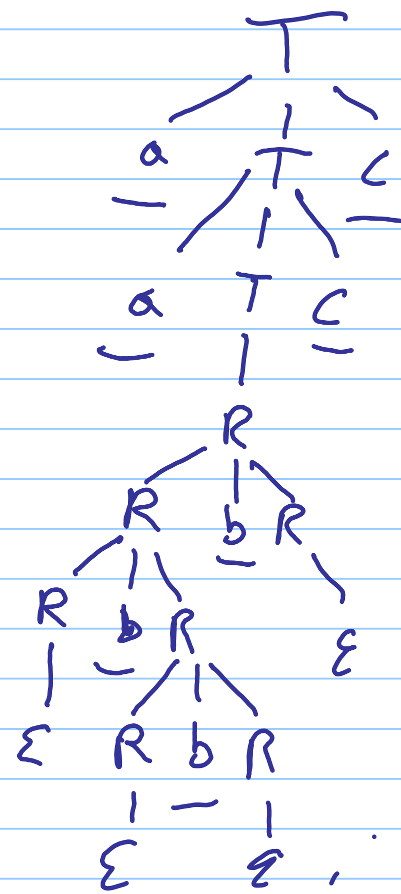
$$
\begin{aligned}
& \underline{T} \\
\Rightarrow\; & \mathrm{a}\underline{T}\mathrm{c} \\
\Rightarrow\; & \mathrm{aa}\underline{T}\mathrm{cc} \\
\Rightarrow\; & \mathrm{aa}\underline{R}\mathrm{cc} \\
\Rightarrow\; & \mathrm{aa}R\mathrm{b}\underline{R}\mathrm{cc} \\
\Rightarrow\; & \mathrm{aa}\underline{R}\mathrm{bcc} \\
\Rightarrow\; & \mathrm{aa}R\mathrm{b}\underline{R}\mathrm{bcc} \\
\Rightarrow\; & \mathrm{aa}R\mathrm{b}\underline{R}\mathrm{b}R\mathrm{bcc} \\
\Rightarrow\; & \mathrm{aa}\underline{R}\mathrm{bb}R\mathrm{bcc} \Leftarrow \\
\Rightarrow\; & \mathrm{aabb}\underline{R}\mathrm{bcc} \\
\Rightarrow\; & \mathrm{aabbbcc}
\end{aligned}
$$



neither a leftmost nor a rightmost derivation

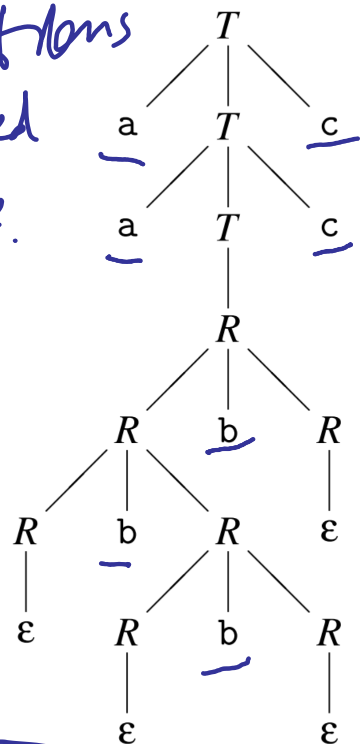
$$
\begin{aligned}
& \underline{T} \\
\Rightarrow\quad & \mathrm{a}\underline{T}\mathrm{c} \\
\Rightarrow\quad & \mathrm{aa}\underline{T}\mathrm{cc} \\
\Rightarrow\quad & \mathrm{aa}\underline{R}\mathrm{cc} \\
\Rightarrow\quad & \mathrm{aa}\underline{R}\mathrm{b}R\mathrm{cc} \\
\Rightarrow\quad & \mathrm{aa}\underline{R}\mathrm{b}R\mathrm{b}R\mathrm{cc} \\
\Rightarrow\quad & \mathrm{aab}\underline{R}\mathrm{b}R\mathrm{cc} \\
\Rightarrow\quad & \mathrm{aab}\underline{R}\mathrm{b}R\mathrm{b}R\mathrm{cc} \\
\Rightarrow\quad & \mathrm{aabb}\underline{R}\mathrm{b}R\mathrm{cc} \\
\Rightarrow\quad & \mathrm{aabbb}\underline{R}\mathrm{cc} \\
\Rightarrow\quad & \mathrm{aabbbcc}
\end{aligned}
$$

a leftmost derivation



Syntax tree

This tree corresponds to two the two different derivations described before.



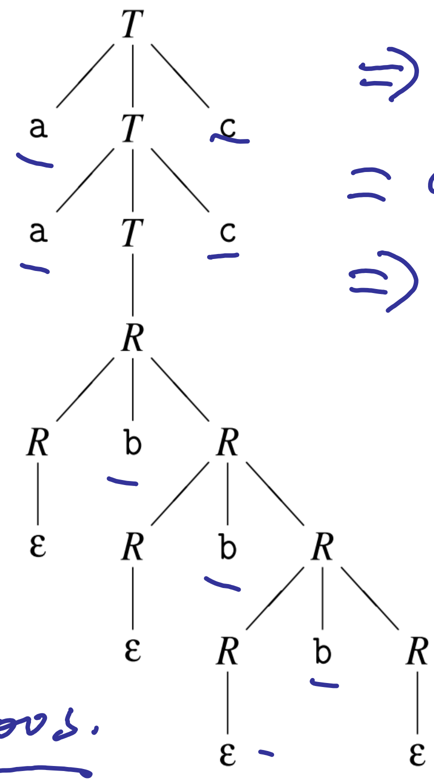
a abbbc

A grammar such that there is exists a string in its language that has two distinct syntax trees is <u>ambiguous</u>.

$\underline{T} \Rightarrow a\underline{T}c \Rightarrow aa\underline{T}cc \Rightarrow aa\underline{R}cc \Rightarrow$

$\Rightarrow aa\underline{R}bRcc \Rightarrow aab\underline{R}cc \Rightarrow$

$\Rightarrow aab\underline{R}bRcc \Rightarrow aabb\underline{R}cc \Rightarrow$



$\Rightarrow abb\underline{R}bRcc \Rightarrow$

$= abbb\underline{R}cc \Rightarrow$

$\Rightarrow abbbcc$

equivalent grammar

This grammar is a non-ambiguous version of

T	$\rightarrow$	R
T	$\rightarrow$	aTc
R	$\rightarrow$	
R	$\rightarrow$	bR

$$T \rightarrow R$$
$$T \rightarrow aTc$$
$$R \rightarrow$$
$$R \rightarrow RbR$$

, which is ambiguous

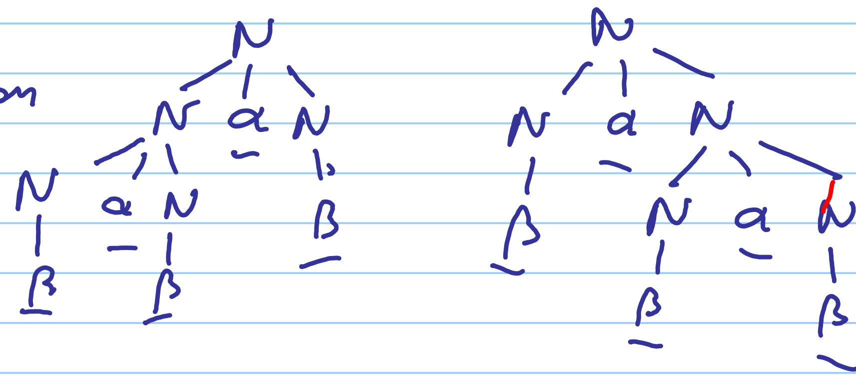
Any grammar with productions like these:

$$\left\{ \begin{array}{l} N \rightarrow N\alpha N \\ N \rightarrow \beta \end{array} \right.$$

is ambiguous

both left & right recursion

$\beta\alpha\beta\alpha\beta$ as a sentential form



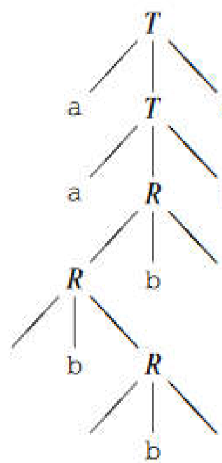
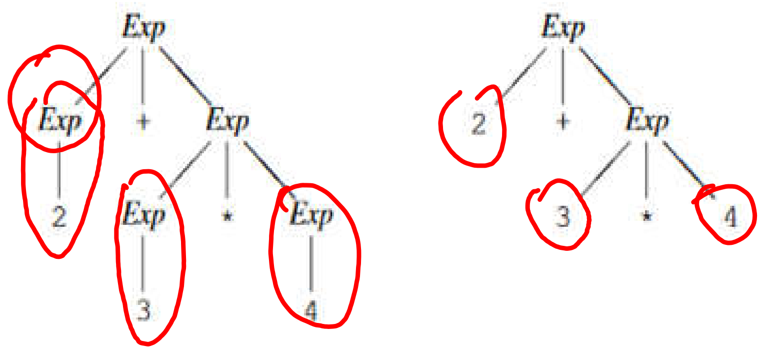
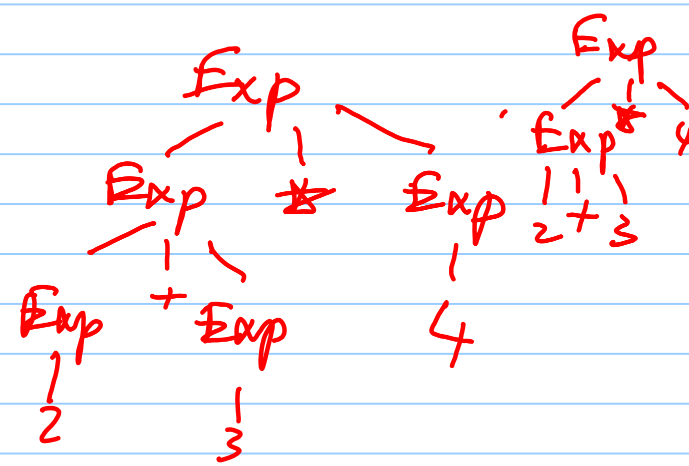


Fig. 2.10 Fully reduced tree for the syntax tree in Fig. 2.7

Fig. 2.11 Preferred syntax tree for 2+3*4 using Grammar 2.2, and the corresponding fully reduced tree



* associates (binds) more tightly than +



$$
\begin{aligned}
Exp &\longrightarrow Exp + Exp \\
Exp &\longrightarrow \mathbf{num}
\end{aligned}
$$

We need to get rid of productions that are both left and right recursive

Operator precedence & associativity

ambiguous
$$\begin{array}{lcl} E & \rightarrow & E \oplus E \\ E & \rightarrow & \textbf{num} \end{array}$$

prod. is both left and right recursion

$5+2-3 = (5+2)-3$? $5+(2-3)$

$5*2/3 = (5*2)/3$? $5*(2/3)$

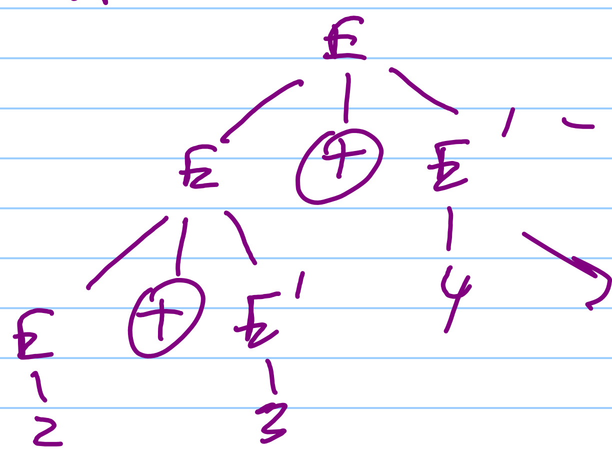
$2-3-4 = 2-(3-4) = 2-(-1) = 3$? $(2-3)-4 = -1-4 = -5$

non-ambiguous assuming $\oplus$ is left-associative
$$\begin{array}{lcl} E & \rightarrow & E \oplus E' \\ E & \rightarrow & E' \\ E' & \rightarrow & \textbf{num} \end{array}$$

production is only left recursive

By convention, $-$ and $/$ are left-associative, so

$2-3-4 = (2-3)-4$



$$\begin{aligned} E &\rightarrow E' \oplus E \\ E &\rightarrow E' \\ E' &\rightarrow \textbf{num} \end{aligned}$$

$\oplus$ is a right associative operator

For example, the list constructor : in Haskell!

$1 : 2 : [\,] = 1 : (2 : [\,]) = 1 : [2] = [1, 2]$

$$\begin{aligned} E &\rightarrow E' \oplus E' \\ E &\rightarrow E' \\ E' &\rightarrow \textbf{num} \end{aligned}$$

$\text{add} :: \text{Int} \rightarrow \underbrace{(\text{Int} \rightarrow \text{Int})}$

$\text{add}\ x\ y = x + y$

"Increment by x"

Non-associative operators

e.g., $<$ in Pascal is non-associative

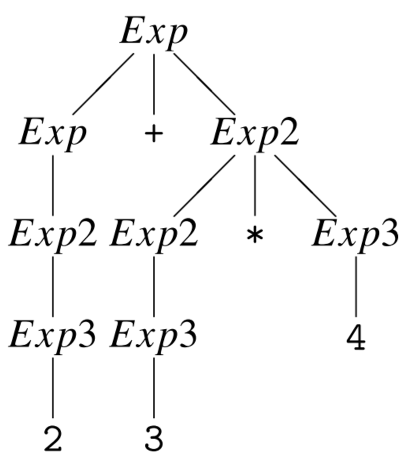
In C, $(3 < 4) < 5 = 1 < 5$ true

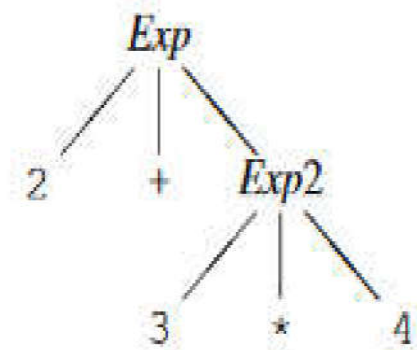
is left associative

$$
\begin{aligned}
E &\longrightarrow E + E' \\
E &\longrightarrow E - E' \\
E &\longrightarrow E' \\
E' &\longrightarrow \mathbf{num}
\end{aligned}
$$

$$
\begin{aligned}
E &\longrightarrow E + E' \\
E &\longrightarrow E' \oplus E \\
E &\longrightarrow E' \\
E' &\longrightarrow \mathbf{num}
\end{aligned}
$$

$$
\begin{array}{lcl}
Exp & \rightarrow & Exp + Exp2 \\
Exp & \rightarrow & Exp - Exp2 \\
Exp & \rightarrow & Exp2 \\
Exp2 & \rightarrow & Exp2 * Exp3 \\
Exp2 & \rightarrow & Exp2 \,/\, Exp3 \\
Exp2 & \rightarrow & Exp3 \\
Exp3 & \rightarrow & \textbf{num} \\
Exp3 & \rightarrow & (\, Exp \,)
\end{array}
$$



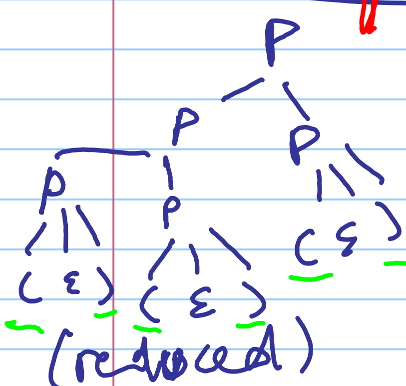


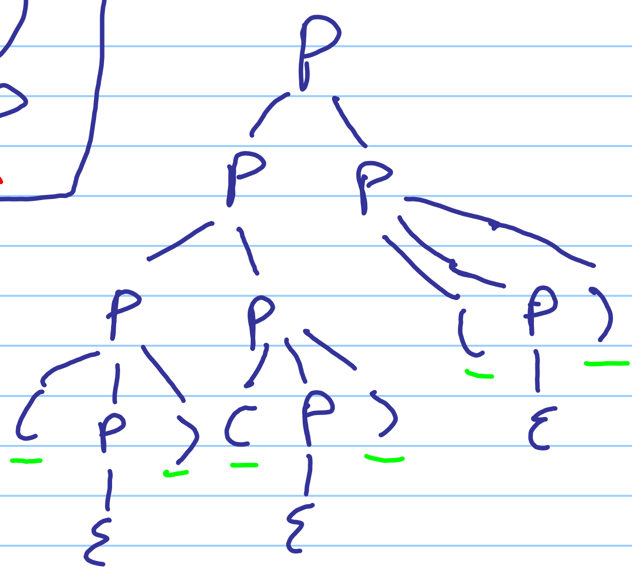
$$
\begin{array}{lcl}
Stat & \longrightarrow & Stat2\ ;\ Stat \\
Stat & \longrightarrow & Stat2 \\
Stat2 & \longrightarrow & Matched \\
Stat2 & \longrightarrow & Unmatched \\
Matched & \longrightarrow & \texttt{if}\ Exp\ \texttt{then}\ Matched\ \texttt{else}\ Matched \\
Matched & \longrightarrow & \mathbf{id} := Exp \\
Unmatched & \longrightarrow & \texttt{if}\ Exp\ \texttt{then}\ Matched\ \texttt{else}\ Unmatched \\
Unmatched & \longrightarrow & \texttt{if}\ Exp\ \texttt{then}\ Stat2
\end{array}
$$

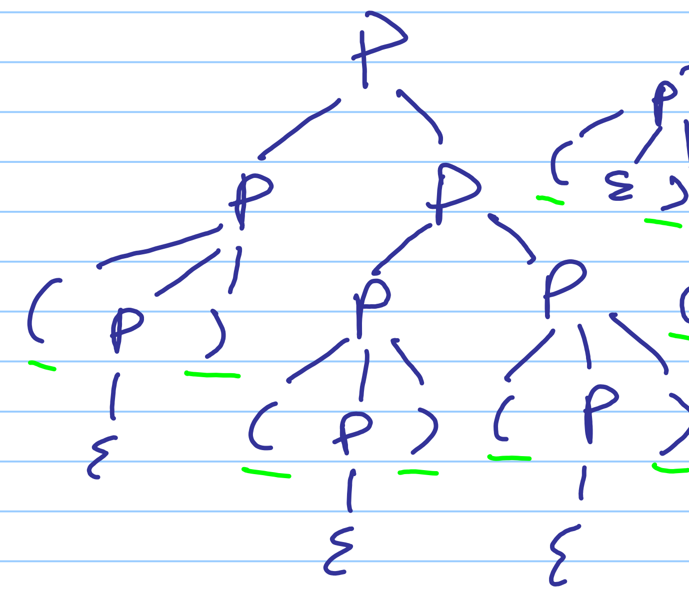
The following grammar (for the language of balanced parentheses) is ambiguous

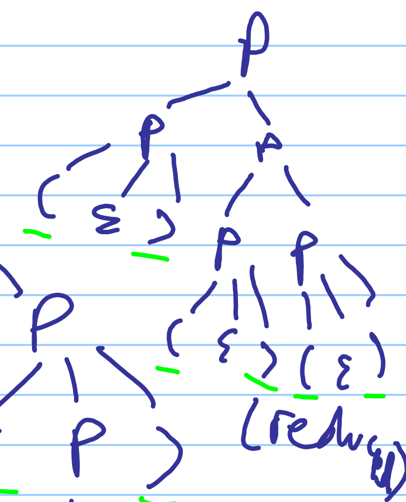
$$P \to \varepsilon$$
$$P \to (P)$$
$$P \to PP$$

Consider the string ()()()









Note that the last production is both left and right recursive.

We can remove left recursion:

$$P \to \varepsilon$$
$$P \to (P)P$$

we removed the left recursion

$$()()()$$