Top-Down Parsing

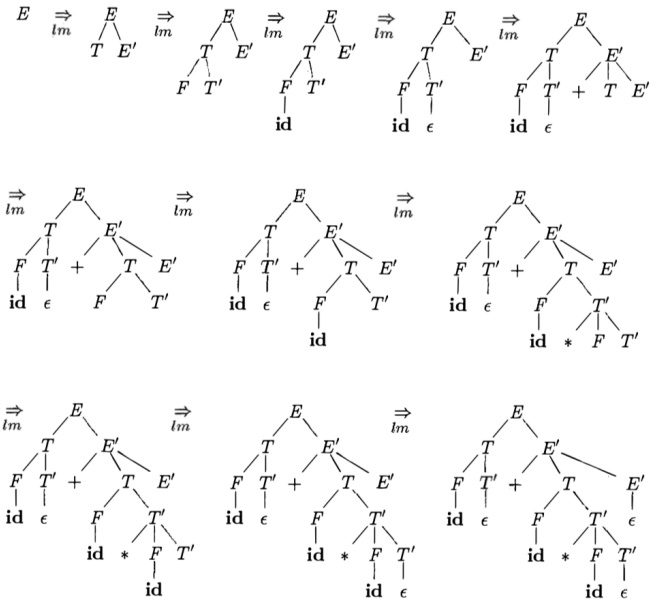
-Constructs a parse tree for the input string, starting from the root and creating nodes of the parse tree in preorder(leftmost derivation)

-At each step of a top-down parse, the key problem Is that of determining the production to be applied for a nonterminal, say A. Once an A-production is chosen, the rest of the parsing process consists of “matching “ the terminal symbols in the production body with the input string.

Ex:
Grammar:

$$
\begin{array}{lcl}
E & \rightarrow & T\ E' \\
E' & \rightarrow & +\ T\ E' \mid \epsilon \\
T & \rightarrow & F\ T' \\
T' & \rightarrow & *\ F\ T' \mid \epsilon \\
F & \rightarrow & (\ E\ ) \mid \mathbf{id}
\end{array}
$$

Parse Tree step by step:



Recursive-Descent Parsing

-A recursive-descent parsing program consists of a set of procedures, one for each nonterminal.
-General recursive-descent may require backtracking so it may test every variant until it finds one(the algorithm in lectures forced a variable)

First and Follow

-The construction of both top-down and bottom-up parsers is aided by two functions, FIRST and FOLLOW, associated with a grammar G.
During top-down parsing, FIRSt and FOLLOW allow us to chose which production to apply, based on the next input symbol.(precedence)

LL(1) Grammars

-Predictive parsers, that is, recursive-descent parsers needing no backtracking, can be constructed for a class of grammars called LL(1).
The first “L” in LL(1) stands for scanning the input from left to right, the second “L” for producing a leftmost derivation, and the “1” for using one input symbol of lookahead at each step to make parsing action decisions.

-A grammar G is LL(1) if and only if whenever A-> alpha|beta are two distinct productions of G, the following conditions hold:
1.- For no terminal alpha do both alpha and beta derive strings beginning with alpha
2.- At most one of alpha and Beta can derive the empty string
3.-

3. If $\beta \stackrel{*}{\Rightarrow} \epsilon$, then α does not derive any string beginning with a terminal in FOLLOW(A). Likewise, if $\alpha \stackrel{*}{\Rightarrow} \epsilon$, then β does not derive any string beginning with a terminal in FOLLOW(A).

COMPARISON

	Pros	Cons
LL(1)	No need of backtracking	No ambiguous grammar allowed
	Faster	No left-recursive allowed
	Produces Left most derivation	Prepare grammar to be unambiguous
FIRST and FOLLOW	Better flow-control of tokens	Need of Backtracking
	Allows left recursive(solved with FIRST or FOLLOW)	Slower
	Allows ambiguous grammr(solved with FIRSt and FOLLOW)	