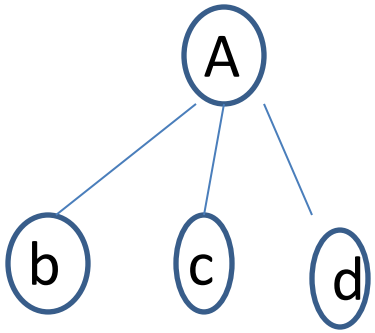
Top-Down and Bottom-Up Parsing

We usually display a derivation as a *parse tree*, where a rule such as A ::= bcd is displayed as



Top-down parsing builds the parse tree from the top (start symbol) down; most top-down methods are LL.

Bottom-up parsing builds the parse tree from the leaves (terminal symbols) up; most methods are LR.

We will look primarily at 2 parsing techniques: recursive descent, which is top-down and LL, and table-driven LALR parsers, which are bottom-up and LR.

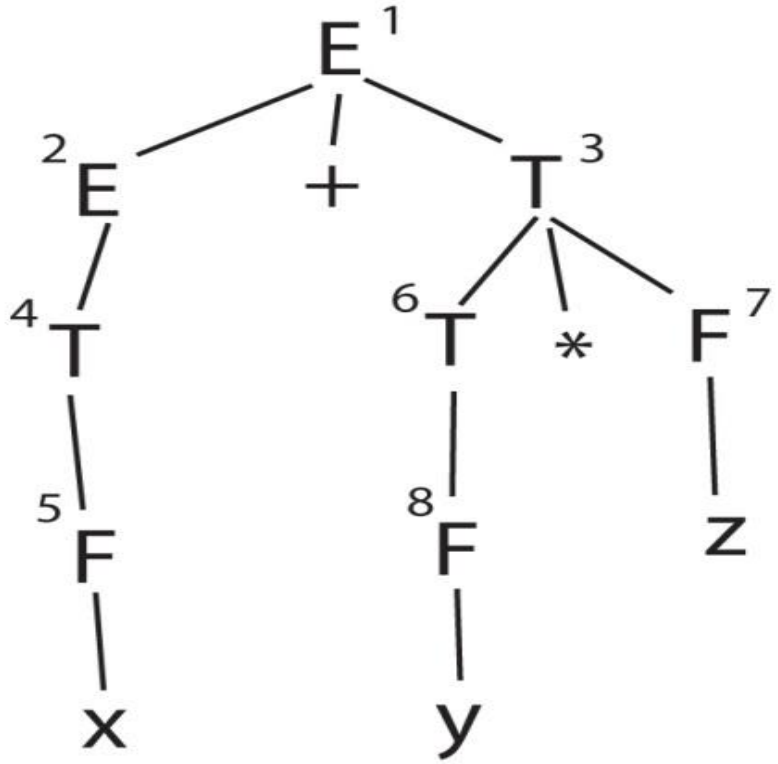
For example, consider the grammar

E ::= E+T | T (E is the start symbol)

T ::= T*F | F

F ::= id (assume some finite set of possible ids)

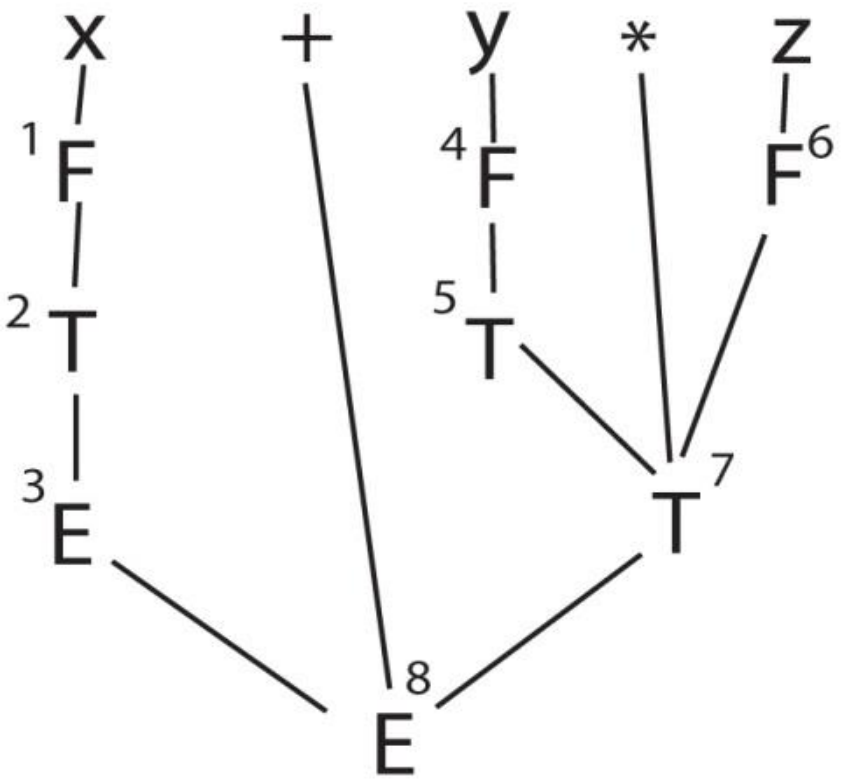
Here is a top-down parse of the sentence x+y*z. Note that it is LL: we always expand the leftmost nonterminal node.



The top-down parse corresponds to the derivation

E
E+T
T+T
F+T
x+T
x+T*F
x+F*F
x+y*F
x+y*z

Here is a bottom-up parse of the same sentence: x+y*z. Note that both parses produce the same tree; it is the order in which the nodes are produced that changes. To read this as a derivation we have to go from the bottom to the top, and then it is LR: we always expand the rightmost node.



Here is the bottom-up parse written as a derivation:

E
E+T
E+T*F
E+T*z
E+F*z
E+y*z
T+y*z
F+y*z
x+y*z

Some people are confused by the idea of reading from the left and still expanding from the right. This possible because we are producing the tree rather than the derivation, and we only get the derivation by turning the tree upside down. I find that "bottom-up" and "top-down" are more effective descriptors than "LL" and "LR" but the latter names are frequently used.