

* Parsing :

* LL : Left-to-right, Leftmost derivation
(input read)

- simpler, easier to understand & generate
- top-down, predictive parsers

* LR : Left-to-right, Rightmost derivation
(input read)

- larger, more intuitive
- bottom-up parsers
- construct the parse tree from leaves up
- no prediction (shift-reduce parsers)

* Example : pp. 62-63, Figure 2.13

* LL(1), LR(2)



how many tokens of lookahead are required in order to parse.

⇒ most compilers use only one token lookahead.

* Writing an LL(1) grammar :

Two obstacles : 1) Left recursion
2) common prefixes

} Not having these
does not guarantee
LL(1), but required.

- * $A \rightarrow A + B$ $\Leftarrow$ left recursion
 - * $A \rightarrow C + B$
 $C \rightarrow A$ } indirect left recursion
 - * $A \rightarrow aB$
 $A \rightarrow a$ } common prefix
- } problem for top-down
 but not a problem
 for bottom-up

* Left Factoring: (Getting rid of Common prefixes)

$$\begin{array}{l}
 A \rightarrow aB \\
 A \rightarrow a \\
 \hline
 A \rightarrow aB \mid a \\
 \hline
 A \rightarrow a(B \mid \epsilon) \\
 \hline
 A \rightarrow a \quad A_{\text{new}} \\
 A_{\text{new}} \rightarrow B \mid \epsilon
 \end{array}$$

* Eliminating Left Recursion:

$$\begin{array}{l}
 A \rightarrow A + B \\
 \text{ ~~} A \rightarrow A + B \text{ } \end{array}
 \quad
 \left.
 \begin{array}{l}
 A \rightarrow a + B \\
 B \rightarrow b + B \\
 B \rightarrow b
 \end{array}
 \right\}~~$$

$$\begin{array}{l}
 A \rightarrow a \\
 B \rightarrow b
 \end{array}$$

* Example : pp. 64-65, Figure 2.14