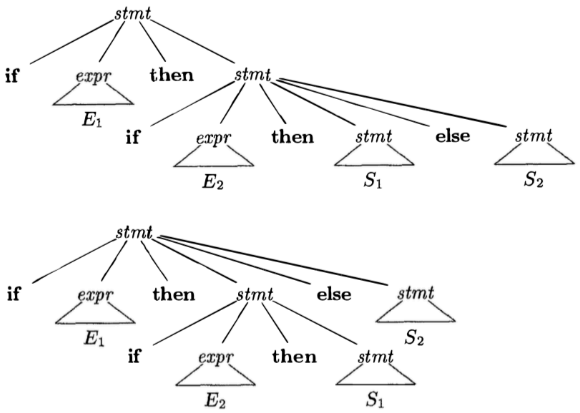
Grammar Notes

Ambiguity happens when there are more than 1 parse trees for an expression



Ambiguous grammar has to be eliminated

Elimination of left recursion

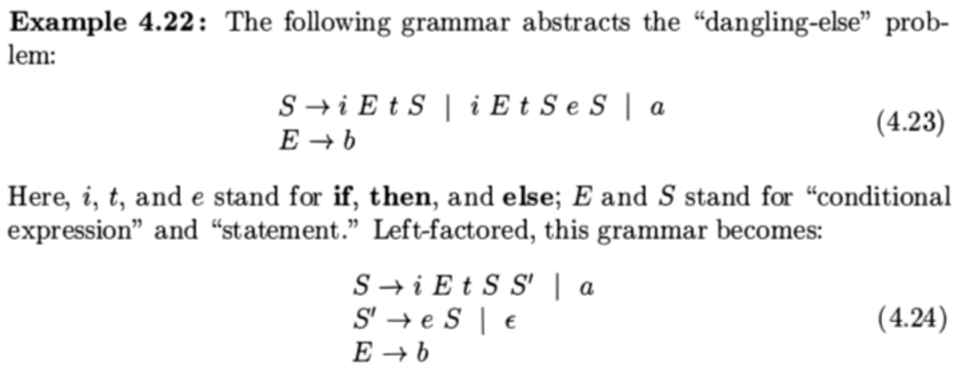
Elimination of left recursion is key to construct a top-down parsing tree
Immediate left recurssion can be eliminated as follows:

$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

$$\begin{array}{l} A \to \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A' \\ A' \to \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon \end{array}$$

Left Factoring

Useful for a grammar suitable for predictive or top-down parsing.
Used when a choice between two alternative A-productions is not clear.



Example 4.22: The following grammar abstracts the "dangling-else" problem:

$$\begin{array}{l} S \to i\ E\ t\ S \mid i\ E\ t\ S\ e\ S \mid a \\ E \to b \end{array} \tag{4.23}$$

Here, i, t, and e stand for **if**, **then**, and **else**; E and S stand for "conditional expression" and "statement." Left-factored, this grammar becomes:

$$\begin{array}{l} S \to i\ E\ t\ S\ S' \mid a \\ S' \to e\ S \mid \epsilon \\ E \to b \end{array} \tag{4.24}$$

Top-down parsing

Consists in constructing a parse tree for the input string, striating from the root and creating nodes of the parse tree in preorder.

Rightmost(botton-up).-Figures out expression while substituting ids

Leftmost(top-down)-Figures out expression first, then substitutes ids