MicroJava Grammar - Removing Left Recursion and Left Factoring

1. Left Recursion in Expressions and Terms

Original Grammar:

Expr → Expr + Term | Expr - Term | Term Term → Term * Factor | Term / Factor | Factor

Problem: Both *Expr* and *Term* are immediately left recursive.

Solution (After Removing Left Recursion):

Expr → Term Expr' Expr' → + Term Expr' | - Term Expr' | ε Term → Factor Term' Term' → * Factor Term' | / Factor Term' | ε

2. Left Recursion in Designator

Original Grammar:

Designator → Designator . ident | Designator [Expr] | ident

Problem: The nonterminal *Designator* is left recursive.

Solution (After Removing Left Recursion):

Designator → ident Designator' Designator' → . ident Designator' | [Expr] Designator' | ε

3. Left Factoring in Factor (and resolving ambiguity)

Original Grammar:

Factor → Designator | Designator ActPars | number | charConst | (Expr) | new ident [Expr]

Problem: The first two alternatives share the common prefix *Designator*.

Solution (After Left Factoring):

Factor → Designator FactorSuffix | number | charConst | (Expr) | new ident NewSuffix FactorSuffix → ActPars | ε NewSuffix → [Expr] | ε

4. Optional Left Factoring of if-else Statement

Original Grammar:

Statement → if (Expr) Statement | if (Expr) Statement else Statement

Problem: Both alternatives begin with *if (Expr) Statement* (dangling else problem).

Solution (After Left Factoring):

Statement → if (Expr) Statement StmtSuffix StmtSuffix → else Statement | ε

Conclusion: After removing left recursion and applying left factoring, the MicroJava grammar becomes suitable for LL(1) predictive parsing.