LL GRAMMARS

LL grammars are a subset of BNF (context-free) grammars. **LL** grammars impose restrictions on BNF grammars to simplify parsing. **LL** grammars may be implemented in top-down predictive parsing algorithms with complexity O(n), where n is the length of the input string. This includes both recursive-descent and table-driven algorithms based on syntax graphs. **LL** parsers operate by scanning the input from **L**eft to right and producing a **L**eftmost derivation of the input string.

The conditions for an **LL** grammar are stated in terms of the **FIRST** and **FOLLOW** sets applied to the grammar rules:

1. If <X> <V_1> | <V_2> | ⋯ | <V_n>, then FIRST(V_i) ∩ FIRST(V_j) = φ, i≠j.
2. If <Z> {X} (zero or more X's), FIRST(X) ∩ FOLLOW(Z) = φ.
3. No left recursion is allowed.

Briefly, FIRST(X) is the set of all terminals which can appear at the start of a string derived from X. FOLLOW(Z) is the union of FIRST(V) for all grammar rules of the form: X α ZV. In case X α Z, FOLLOW(Z) FOLLOW(X).

Example 3.1: Modified for LL Grammar

The FIRST condition for an LL grammar is violated by the RHS rules in Example 3.1 for <stmt_list> and <expression>. An LL grammar can be obtained by left-factoring these rules and converting the right recursion in <stmt_list> to repetition. Using EBNF, the modified rules are:

<stmt_list> → <stmt> { ; <stmt> }
<expression> → <var> [(+ | -) <var>]