Programming Languages and Computation

Week 4: Obtaining LL(1) Grammars

* 1. Consider the following grammar for binary list literals:

$$\begin{array}{rcl} L & \longrightarrow & [\,] \mid [B] \mid [M] \\ M & \longrightarrow & M,\ M \mid B \\ B & \longrightarrow & 0 \mid 1 \end{array}$$

Give an equivalent grammar that avoids left recursion.

* 2. Left factor the following grammar (that is, give an equivalent grammar in which no two productions for the same non-terminal have the same prefix):

$$S \longrightarrow \mathsf{if}\ \mathsf{bexp}\ \mathsf{then}\ S \mid \mathsf{if}\ \mathsf{bexp}\ \mathsf{then}\ S\ \mathsf{else}\ S \mid \mathsf{other}$$

** 3. For each of the following grammars, give an equivalent LL(1) grammar. Typically, you will need to look at examples of strings in the language of the grammars to get an idea of the shape of the sequences generated by left recursive rules in order to give insight into the problem.

(a)

$$S \longrightarrow 0\ S\ 1 \mid 0\ 1$$

(b)

$$\begin{array}{l} S \longrightarrow (\ L\ ) \mid a \\ L \longrightarrow L,\ S \mid S \end{array}$$

(c)

$$S \longrightarrow S\ (\ S\ )\ S \mid \epsilon$$

** 4. The following grammar describes the language of Regular Expressions (regex), which are a standard way of describing pattern matching on strings in programming languages.

$$S \longrightarrow S + S \mid S\ S \mid (\ S\ ) \mid S* \mid a$$

Give an LL(1) grammar for the same language.

*** 5. Consider again the grammar given in answer to Question 2.

(a) Explain why your left-factored grammar is not LL(1).

(b) Rather than giving an equivalent LL(1) grammar, which is impossible in this case, describe a simple modification to the parsing function for the nonterminal which exhibits the rule conflict which will allow for parsing the left-factored grammar.