HW 2: Syntax Analysis

CSC 4351, Spring 2014

Due: 26 February 2014

1. Context free grammars, LL and LR parsing
   Consider the following simple context free grammars:

Grammar G_1	Grammar G_2
$G \rightarrow A\$$	$G \rightarrow A\$$
$A \rightarrow \epsilon$	$A \rightarrow \epsilon$
$A \rightarrow bAb$	$A \rightarrow Abb$

   The start symbols are G, the nonterminals are G and A, and the terminal symbols are b and \$ (end of file). Note that these grammars generate the same language: strings consisting of even numbers of b symbols (including zero of them).

   (a) Attempt to show a shift-reduce parse of the input string *bbbb* for a parser for grammar G_1. Show the contents of the stack, the input, and the actions (in the style of Figure 3.18 on Page 58 but without the subscripts for the parse states).

   Indicate any conflicts and describe why they are conflicts. Is G_1 LR(1)? Is it LR(0)?

   (b) Attempt to show a shift-reduce parse of the input string *bbbb* for a parser for grammar G_2. Show the contents of the stack, the input, and the actions (in the style of Figure 3.18 on Page 58 but without the subscripts for the parse states).

   Indicate any conflicts and describe why they are conflicts. Is G_2 LR(1)? Is it LR(0)?

   (c) Indicate whether G_1 and G_2 are LL(1). You don't need to construct their LL(1) parse tables, but you may argue from other properties.

   (d) Of the language classes we have discussed in class, what is the *smallest* category into which $L(G_1)$ fits? Justify your answer. [Hint: This is a trick question!]

2. Context free grammars, LL parsing
   Consider the following grammar:

   $E \to E + T \mid T$
   $T \to \texttt{id} \mid \texttt{id}() \mid \texttt{id}(L)$
   $L \to E, L \mid E$

   The nonterminals are E, T, and L. The terminals are $+$, `id`, $($, $)$, ;. The start symbol is E.

   (a) Modify the grammar such that it can be parsed by an LL(1) parser.

   (b) Show Nullable, FIRST, and FOLLOW and derive the LL(1) parse table for the modified grammar.

   (c) Give the (non-abstract) parse tree produced by your grammar for the input

   ```
   a + b(c, d())
   ```