CSE 322
Winter Quarter 2009
Assignment 8
Due Friday, February 27, 2009

All solutions should be neatly written or type set. All major steps in proofs must be justified. Please start each problem solution on a new page and put your name on every page.

1. (10 points) In this problem we explore the "top down" and "bottom up" construction for PDAs from context-free grammars. Consider the grammar $G = (V, \Sigma, R, E)$ where

$$\begin{aligned} V &= \{T, F, E\} \\ \Sigma &= \{+, *, (,), a\} \\ R &= \{E \to E + T, \\ &= E \to T, \\ &= T \to T * F, \\ &= T \to F, \\ &= F \to (E), \\ &= F \to a\} \end{aligned}$$

   (a) Design a PDA M_T by the "top down" construction that accepts $L(G)$. You may use a state diagram. Give a leftmost derivation of $(a + a) * a + a$. Beside it give the sequence of configurations from M_T that corresponds to the leftmost derivation. A configuration shows which symbol of the input is being processed, what state the machine is in, and the contents of the stack.

   (b) Design a (extended) PDA M_B by the "bottom up" construction that accepts $L(G)$. You may use a state diagram. Give a rightmost derivation of $(a + a) * a + a$ in reverse order, top-to-bottom. Beside it give the sequence of configurations from M_B that corresponds to the rightmost derivation.

   The bottom up construction was given in class and is not found in the book. It works as follows. There is a state q_ℓ which has two roles. The first role is to manage *reduce* steps. In a reduce step, if $A \to \alpha$ is a production, then the extended PDA in state q_ℓ can remove α^R from the stack and replace it with A. The second role is to manage *shift* steps. In a shift step, the PDA in state q_ℓ can take an input symbol and push it on to the stack. If $S\$$ ever appears on the stack then the PDA can move from q_ℓ to its only accepting state q_f. In the start state q_0, the symbol \$ is pushed on the stack, and the machine moves to state q_ℓ.

2. (10 points) In this problem you will execute an example of the CYK algorithm for deciding membership in a context-free language. Consider the grammar $G = (V, \Sigma, R, S)$ where

$$V = \{S, A, B, C\}$$

$$
\begin{aligned}
\Sigma &= \{0, 1\} \\
R &= \{S \to AB \mid BC \\
&= A \to BA \mid 0 \\
&= B \to CC \mid 1 \\
&= C \to AB \mid 0\}
\end{aligned}
$$

(a) Show the result (the matrix of sets) of running the CYK algorithm on the input 10001.

(b) If 10001 is generated by the grammar, then use the result of part (a) to construct a parse tree for it.

3. (10 points) In this problem you will design an algorithm to decide if the language generated by a context-free language is finite. First let $G = (V, \Sigma, R, S)$ be a Chomsky normal form context-free grammar with the property that every non-terminal is productive. Recall, that a non-terminal A is productive if $A \Rightarrow^* x$ for some terminal string x. Using a closure algorthm it can be shown that any context-free grammar can converted to an equivalent grammar with this form. Define the relation $D_G \subseteq V \times V$ as follows. A pair $(A, B) \in D_G$ if there is a production of the form $A \to XB$ or $A \to BX$ in R for some X. Design an algorithm for deciding the finiteness of $L(G)$ that uses the reflexive, transitive closure of D_G, called D_G^*, which can be computed using Warshall's algorithm. Expain briefly why your algorithms works.