Solutions to Written Assignment 2

1. Give a context-free grammar (CFG) for each of the following languages over the alphabet $\Sigma = \{a,b\}$:

(a) All strings in the language $L : \{a^n b^m a^{2n} | n, m \geq 0\}$

$$S \rightarrow aSaa \mid B$$
$$B \rightarrow bB \mid \epsilon$$

(b) All nonempty strings that start and end with the same symbol.

$$S \rightarrow aXa \mid bXb \mid a \mid b$$
$$X \rightarrow aX \mid bX \mid \epsilon$$

(c) All strings with more a’s than b’s.

$$S \rightarrow Aa \mid MS \mid SMA$$
$$A \rightarrow Aa \mid \epsilon$$
$$M \rightarrow \epsilon \mid MM \mid bMa \mid aMb$$

(d) All palindromes (a palindrome is a string that reads the same forwards and backwards).

$$S \rightarrow aSa \mid bSb \mid a \mid b \mid \epsilon$$

2. A history major taking CS143 decided to write a rudimentary CFG to parse the roman numerals 1-99 (i,ii,iii,iv,v,...,ix,x,...,xl,...,lxxx,...,xc,...,xcix). If you are unfamiliar with roman numerals, please have a look at http://en.wikipedia.org/wiki/Roman_numerals and http://literacy.kent.edu/Minigrants/Cinci/romanchart.htm.

Consider the grammar below, with terminals \{c,l,x,v,i\}. $c = 100, l = 50, x = 10, v = 5, i = 1$. Notice that we use lowercase characters here to represent the numerals, to distinguish them from the non-terminals.

$$S \rightarrow xTU \mid 1X \mid X$$
$$T \rightarrow c \mid l$$
$$X \rightarrow xX \mid U$$
$$U \rightarrow iY \mid vI \mid I$$
$$Y \rightarrow x \mid v$$
$$I \rightarrow iI \mid \epsilon$$

(a) Draw a parse tree for 47: “xlvii”.

See Figure 1.

(b) Is this grammar ambiguous?

No
(c) Write semantic actions for each of the 14 rules in the grammar (remember $X \rightarrow A \mid B$ is short for $X \rightarrow A$ and $X \rightarrow B$) to calculate the decimal value of the input string. You can associate a synthesized attribute $val$ to each of the non-terminals to store its value. The final value should be returned in $S.val$. Hint: have a look at the calculator examples presented in class.

$$S \rightarrow lX \quad \{S.val = X.val + 50\}$$
$$S \rightarrow xTU \quad \{S.val = T.val - 10 + U.val\}$$
$$S \rightarrow c \quad \{T.val = 100\}$$
$$S \rightarrow l \quad \{T.val = 50\}$$
$$S \rightarrow X \quad \{S.val = X.val\}$$
$$T \rightarrow \epsilon \quad \{I.val = 0\}$$
$$T \rightarrow i \quad \{U.val = I.val\}$$
$$T \rightarrow v \quad \{U.val = I.val + 5\}$$
$$X_1 \rightarrow xX_2 \quad \{X_{1.val} = X_{2.val} + 10\}$$
$$X \rightarrow U \quad \{X.val = U.val\}$$
$$U \rightarrow iY \quad \{U.val = Y.val - 1\}$$
$$U \rightarrow vI \quad \{U.val = I.val + 5\}$$
$$U \rightarrow I \quad \{U.val = I.val\}$$
$$Y \rightarrow x \quad \{Y.val = 10\}$$
$$Y \rightarrow v \quad \{Y.val = 5\}$$
$$I_1 \rightarrow iI_2 \quad \{I_{1.val} = I_{2.val} + 1\}$$
$$I \rightarrow \epsilon \quad \{I.val = 0\}$$

3. (a) Left factor the following grammar:

$$E \rightarrow \text{int} \mid \text{int} + E \mid \text{int} - E \mid E - (E)$$

Solution:

$$E \rightarrow \text{int} E' \mid E - (E)$$
$$E' \rightarrow \epsilon \mid + E \mid - E$$

(b) Eliminate left-recursion from the following grammar:

$$A \rightarrow A + B \mid B$$
$$B \rightarrow \text{int} \mid (A)$$
Solution:

\[ A \rightarrow BA' \]
\[ A' \rightarrow +BA' \mid \epsilon \]
\[ B \rightarrow \text{int} \mid (A) \]

4. Consider the following LL(1) grammar, which has the set of terminals \( T = \{ a, b, \text{ep}, +, *, () \} \). This grammar generates regular expressions over \( \{a, b\} \), with + meaning the RegExp OR operator, and \( \text{ep} \) meaning the \( \epsilon \) symbol. (Yes, this is a context free grammar for generating regular expressions!)

\[
\begin{align*}
E &\rightarrow TE' \\
E' &\rightarrow +E \mid \epsilon \\
T &\rightarrow FT' \\
T' &\rightarrow T \mid \epsilon \\
F &\rightarrow PF' \\
F' &\rightarrow *F' \mid \epsilon \\
P &\rightarrow (E) \mid a \mid b \mid \text{ep}
\end{align*}
\]

(a) Give the first and follow sets for each non-terminal.

The First and Follow sets of the non-terminals are as follows.

<table>
<thead>
<tr>
<th></th>
<th>First</th>
<th>Follow</th>
</tr>
</thead>
<tbody>
<tr>
<td>First(( E ))</td>
<td>( { (, a, b, \text{ep} } )</td>
<td>( { }, $ )</td>
</tr>
<tr>
<td>First(( E' ))</td>
<td>( { +, \epsilon } )</td>
<td>( { }, $ )</td>
</tr>
<tr>
<td>First(( T ))</td>
<td>( { (, a, b, \text{ep} } )</td>
<td>( { +, }, $ )</td>
</tr>
<tr>
<td>First(( T' ))</td>
<td>( { (, a, b, \text{ep}, \epsilon } )</td>
<td>( { +, }, $ )</td>
</tr>
<tr>
<td>First(( F ))</td>
<td>( { (, a, b, \text{ep} } )</td>
<td>( { (, a, b, \text{ep}, +, }, $ )</td>
</tr>
<tr>
<td>First(( F' ))</td>
<td>( { *, \epsilon } )</td>
<td>( { (, a, b, \text{ep}, +, }, $ )</td>
</tr>
<tr>
<td>First(( P ))</td>
<td>( { (, a, b, \text{ep} } )</td>
<td>( { (, a, b, \text{ep}, +, }, *, $ )</td>
</tr>
</tbody>
</table>

(b) Construct an LL(1) parsing table for the left-factored grammar.

Here is an LL(1) parsing table for the grammar.

<table>
<thead>
<tr>
<th></th>
<th>( )</th>
<th>a</th>
<th>b</th>
<th>ep</th>
<th>+</th>
<th>*</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E )</td>
<td>( TE' )</td>
<td>( TE' )</td>
<td>( TE' )</td>
<td>( TE' )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( E' )</td>
<td>( +E )</td>
<td>( \epsilon )</td>
<td>( \epsilon )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( T )</td>
<td>( FT' )</td>
<td>( FT' )</td>
<td>( FT' )</td>
<td>( FT' )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( T' )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( \epsilon )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( F )</td>
<td>( PF' )</td>
<td>( PF' )</td>
<td>( PF' )</td>
<td>( PF' )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( F' )</td>
<td>( \epsilon )</td>
<td>( \epsilon )</td>
<td>( \epsilon )</td>
<td>( \epsilon )</td>
<td>( \epsilon )</td>
<td>( *F' )</td>
<td>( \epsilon )</td>
</tr>
<tr>
<td>( P )</td>
<td>(E)</td>
<td>( a )</td>
<td>( b )</td>
<td>( \text{ep} )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(c) Show the operation of an LL(1) parser on the input string ab*.

Stack | Input | Action
--- | --- | ---
$E$ | $ab*$ | $TE'$
$TE'$ | $ab*$ | $FT'$
$FT'E'$ | $ab*$ | $PF'$
$PF'T'E'$ | $ab*$ | $a$
$aF'T'E'$ | $ab*$ | terminal
$F'T'E'$ | $b*$ | $\epsilon$
$T'E'$ | $b*$ | $T$
$TE'$ | $b*$ | $FT'$
$FT'E'$ | $b*$ | $PF'$
$PFT'E'$ | $b*$ | $b$
$bF'T'E'$ | $b*$ | terminal
$F'T'E'$ | $*$ | $F'$
$F'T'E'$ | $*$ | terminal
$F'T'E'$ | $\epsilon$
$T'E'$ | $\epsilon$
$E'$ | $\epsilon$
$\epsilon$ | $\epsilon$ | ACCEPT

5. Consider the following CFG, which has the set of terminals $T = \{\text{stmt}, \{\}, ;\}$. This grammar describes the organization of statements in blocks for a fictitious programming language. Blocks can have zero or more statements and other nested blocks, separated by semicolons, where the last semicolon is optional. (P is the start symbol here.)

$$
P \rightarrow S
$$

$$
S \rightarrow \text{stmt} \mid \{B
$$

$$
B \rightarrow \} \mid S \} \mid S;B
$$

(a) Construct a DFA for viable prefixes of this grammar using LR(0) items.

Figure 2 shows a DFA for viable prefixes of the grammar. Note that for simplicity we omitted adding an extra state $S' \rightarrow P$.

(b) Identify any shift-reduce and reduce-reduce conflicts in this grammar under the SLR(1) rules.

There are no conflicts.

(c) Assuming that an SLR(1) parser resolves shift-reduce conflicts by choosing to shift, show the operation of such a parser on the input string $\{\text{stmt;}\}$. 

Figure 2: A DFA for viable prefixes of the grammar in Question 5a

<table>
<thead>
<tr>
<th>Configuration</th>
<th>DFA Halt State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fall 2010/2011