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 $L : \{a^n b^m a^{2n} | n, m \geq 0\}$

   $$\begin{aligned} S &\rightarrow aSaa \mid B \\ B &\rightarrow bB \mid \epsilon \end{aligned}$$

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

   $$\begin{aligned} S &\rightarrow aXa \mid bXb \mid a \mid b \\ X &\rightarrow aX \mid bX \mid \epsilon \end{aligned}$$

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

   $$\begin{aligned} S &\rightarrow Aa \mid MS \mid SMA \\ A &\rightarrow Aa \mid \epsilon \\ M &\rightarrow \epsilon \mid MM \mid bMa \mid aMb \end{aligned}$$

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

   $$\begin{aligned} S &\rightarrow \mathbf{x}TU \mid \mathbf{l}X \mid X \\ T &\rightarrow \mathbf{c} \mid \mathbf{l} \\ X &\rightarrow \mathbf{x}X \mid U \\ U &\rightarrow \mathbf{i}Y \mid \mathbf{v}I \mid I \\ Y &\rightarrow \mathbf{x} \mid \mathbf{v} \\ I &\rightarrow \mathbf{i}I \mid \epsilon \end{aligned}$$

   (a) Draw a parse tree for 47: "xlvii".
   See Figure 1.

   (b) Is this grammar ambiguous?
   No

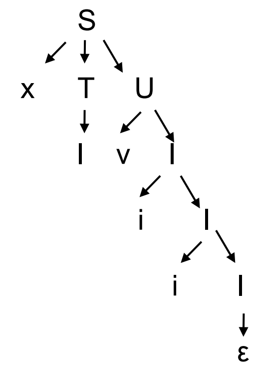

Figure 1: Question 2a: Parse tree for 47: "xlvii"

(c) Write semantic actions for each of the 14 rules in the grammar (remember $X \to A \mid B$ is short for $X \to A$ and $X \to B$) 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.

$$
\begin{array}{rll}
S & \to \mathbf{x}TU & \{S.val = T.val - 10 + U.val\} \\
S & \to \mathbf{l}X & \{S.val = X.val + 50\} \\
S & \to X & \{S.val = X.val\} \\
T & \to \mathbf{c} & \{T.val = 100\} \\
T & \to \mathbf{l} & \{T.val = 50\} \\
X_1 & \to \mathbf{x}X_2 & \{X_1.val = X_2.val + 10\} \\
X & \to U & \{X.val = U.val\} \\
U & \to \mathbf{i}Y & \{U.val = Y.val - 1\} \\
U & \to \mathbf{v}I & \{U.val = I.val + 5\} \\
U & \to I & \{U.val = I.val\} \\
Y & \to \mathbf{x} & \{Y.val = 10\} \\
Y & \to \mathbf{v} & \{Y.val = 5\} \\
I_1 & \to \mathbf{i}I_2 & \{I_1.val = I_2.val + 1\} \\
I & \to \epsilon & \{I.val = 0\}
\end{array}
$$

3. (a) Left factor the following grammar:

$$E \to int \mid int + E \mid int - E \mid E - (E)$$

Solution:

$$
\begin{array}{l}
E \to int\ E' \mid E - (E) \\
E' \to \epsilon \mid\ + E \mid - E
\end{array}
$$

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

$$
\begin{array}{l}
A \to A + B \mid B \\
B \to int \mid (A)
\end{array}
$$

Solution:

$$A \to BA'$$
$$A' \to +BA' \mid \epsilon$$
$$B \to int \mid (A)$$

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

$$\begin{aligned} E &\to TE' \\ E' &\to +E \mid \epsilon \\ T &\to FT' \\ T' &\to T \mid \epsilon \\ F &\to PF' \\ F' &\to *F' \mid \epsilon \\ P &\to (E) \mid \mathbf{a} \mid \mathbf{b} \mid \mathbf{ep} \end{aligned}$$

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

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

$\text{First}(E) = \{(, a, b, ep\}$	$\text{Follow}(E) = \{), \$\}$
$\text{First}(E') = \{+, \epsilon\}$	$\text{Follow}(E') = \{), \$\}$
$\text{First}(T) = \{(, a, b, ep\}$	$\text{Follow}(T) = \{+,), \$\}$
$\text{First}(T') = \{(, a, b, ep, \epsilon\}$	$\text{Follow}(T') = \{+,), \$\}$
$\text{First}(F) = \{(, a, b, ep\}$	$\text{Follow}(F) = \{(, a, b, ep, +,), \$\}$
$\text{First}(F') = \{*, \epsilon\}$	$\text{Follow}(F') = \{(, a, b, ep, +,), \$\}$
$\text{First}(P) = \{(, a, b, ep\}$	$\text{Follow}(P) = \{(, a, b, ep, +,), *, \$\}$

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

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

	(	)	a	b	ep	$+$	$*$	\$
E	TE'		TE'	TE'	TE'			
E'		ϵ				$+E$		ϵ
T	FT'		FT'	FT'	FT'			
T'	T	ϵ	T	T	T	ϵ		ϵ
F	PF'		PF'	PF'	PF'			
F'	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	$*F'$	ϵ
P	(E)		a	b	ep			

(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*
$F'T'E'\$$	$b*\$$	ϵ
$T'E'\$$	$b*\$$	T
$TE'\$$	$b*\$$	FT'
$FT'E'\$$	$b*\$$	PF'
$PF'T'E'\$$	$b*\$$	b
$bF'T'E'\$$	$b*\$$	*terminal*
$F'T'E'\$$	$*\$$	$*F'$
$*F'T'E'\$$	$*\$$	*terminal*
$F'T'E'\$$	$\$$	ϵ
$T'E'\$$	$\$$	ϵ
$E'\$$	$\$$	ϵ
$\$$	$\$$	*ACCEPT*

5. Consider the following CFG, which has the set of terminals $T = \{\textbf{stmt}, \textbf{\{}, \textbf{\}}, \textbf{;}\}$. 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.)

$$
\begin{aligned}
P &\rightarrow S \\
S &\rightarrow \textbf{stmt} \mid \textbf{\{}B \\
B &\rightarrow \textbf{\}} \mid S\textbf{\}} \mid S\textbf{;}B
\end{aligned}
$$

(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 **{stmt;}**.

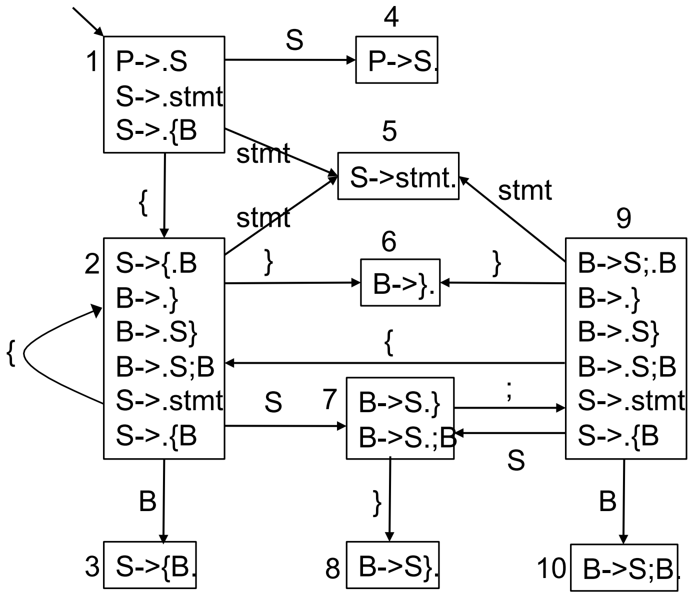

Figure 2: A DFA for viable prefixes of the grammar in Question 5a

Configuration	DFA Halt State	Action
$\mid \{stmt;\}\$$	1	$shift$
$\{\mid stmt;\}\$$	2	$shift$
$\{stmt \mid ;\}\$$	5 $\;';' \in Follow(S)$	$reduce\ S \rightarrow stmt$
$\{S \mid ;\}\$$	7	$shift$
$\{S; \mid\}\$$	9	$shift$
$\{S;\} \mid \$$	6 $\;'\$' \in Follow(B)$	$reduce\ B \rightarrow \}$
$\{S;B \mid \$$	10 $\;'\$' \in Follow(B)$	$reduce\ B \rightarrow S;B$
$\{B \mid \$$	3 $\;'\$' \in Follow(S)$	$reduce\ S \rightarrow \{B$
$S \mid \$$	4 $\;'\$' \in Follow(S')$	$reduce\ P \rightarrow S$
$P \mid \$$		$ACCEPT$