CS 143 2020 - Written Assignment 2

Due Monday, May 4, 2020 at 11:59 PM

This assignment covers context free grammars and parsing. You may discuss this assignment with other students and work on the problems together. However, your write-up should be your own individual work, and you should indicate in your submission who you worked with, if applicable. Assignments can be submitted electronically through Gradescope as a PDF by 11:59 PM PDT. A LaTeX template for writing your solutions is available on the course website.

1. Give the context-free grammar (CFG) for each of the following languages. Any grammar is acceptable - including ambiguous grammars - as long as it has the correct language.

   (a) The set of all strings over the alphabet $\{2, -, +\}$ representing valid arithmetic expressions where each integer in the expression is a single digit and the expression evaluates to some value ≥ 0.
   Example Strings in the Language:

   2+2   2-2+2   -2-2+2+2

   Strings not in the Language:

   +2   -2   -2+22   2++2-2 ϵ

   (b) The set of all strings over the alphabet $\{string| \ |\%s|arg|"|,\}$ representing valid arguments to the c printf() function. For the purposes of this problem, treat *string* and *arg* as tokens of your language where *string* represents an arbitrary length sequence of characters [A-Z][a-z] and *arg* represents any arbitrary char*. printf() replaces each $\%s$ with the contents of arg. For instance, printf("Test $\%s$ $\%s$", foo, bar) will print "Test (contents of foo) (contents of bar)". See the c printf() documentation for further detail. Although printf() ignores unused args, your grammar should produce strings **with an equal number of %s and arg tokens**. Note that ',' and ' ' are in the alphabet.

   Example Strings in the Language (surrounded by printf() for clarity, do not inclue printf() in the grammar):

   printf("")   printf("*string* $\%s\%s$", arg, arg)
   printf(" $\%s$ *stringstring*   "   , *arg*   )

   Strings not in the Language (surrounded by printf() for clarity, do not inclue printf() in the grammar):

   printf("$\%s$")   printf("*arg string*")   printf(*string* $\%s$, *arg*)

   (c) The set of all strings over the alphabet $\{0, 1\}$ in the language $L : \{0^i 1^j 0^k \mid j \leq i + k\}$.
   Example Strings in the Language:

   00000   000111100 ϵ

   Strings not in the Language:

   1   000111101

(d) The set of all strings over the alphabet $\{[,], \{, \}, ,\}$ which are sets. We define a set to be a collection of zero or more comma-separated arrays enclosed in an open brace and a close brace. Similarly, we define an array to be a collection of zero or more comma-separated sets enclosed in an open bracket and a close bracket. **Note** that "," is in the alphabet.
Example Arrays:

$$[\{\}, \{\}] \qquad [] \qquad [\{[], []\}]$$

Example Sets:

$$\{[]\} \qquad \{\} \qquad \{[\{\}], []\}$$

Example Strings in the Language:

$$\{\} \qquad \{[], [\{[]\}]\} \qquad \{[\{\}, \{\}, \{\}], []\}$$

Strings not in the Language:

$$[] \qquad \{\{\}\} \qquad \{[[]]\}$$

2. (a) Left factor the following grammar:

$$\begin{aligned} S &\to I \mid I - J \mid I + K \\ I &\to (J - K) \mid (J) \\ J &\to K1 \mid K2 \\ K &\to K3 \mid \epsilon \end{aligned}$$

(b) Eliminate left recursion from the following grammar:

$$\begin{aligned} S &\to STS \mid ST \mid T \\ T &\to Ta \mid Tb \mid U \\ U &\to T \mid c \end{aligned}$$

3. Consider the following CFG, where the set of terminals is $\{a, b, \#, \%, !\}$:

$$\begin{aligned} S &\to \%aT \mid U! \\ T &\to aS \mid baT \mid \epsilon \\ U &\to \#aTU \mid \epsilon \end{aligned}$$

(a) Construct the FIRST sets for each of the nonterminals.
(b) Construct the FOLLOW sets for each of the nonterminals.
(c) Construct the LL(1) parsing table for the grammar.
(d) Show the sequence of stack, input and action configurations that occur during an LL(1) parse of the string "#abaa%aba!". At the beginning of the parse, the stack should contain a single S.

4. What advantage does left recursion have over right recursion in shift-reduce parsing?
Hint: Consider left and right recursive grammars for the language a*. What happens if your input has a million a's?

5. Consider the Following Grammar G over the alphabet $\Sigma = \{a, b, c\}$:

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow Aa \\
S &\rightarrow Bb \\
A &\rightarrow Ac \\
A &\rightarrow \epsilon \\
B &\rightarrow Bc \\
B &\rightarrow \epsilon
\end{aligned}
$$

You want to implement G using an SLR(1) parser (note that we have already added the S' $\rightarrow$ S production for you).

(a) Construct the first state of the LR(0) machine, compute the FOLLOW sets of A and B, and point out the conflicts that prevent the grammar from being SLR(1)

(b) Show modifications to production 4 (A $\rightarrow$ Ac) and production 6 (B $\rightarrow$ Bc) that make the grammar SLR(1) while having the same language as the original grammar G. Explain the intuition behind this result.