Homework Assignment 2 (HW2)
Posted Monday, February 8, 2016
Due Thursday, February 18, 2016

Problem 1 (10pts). Write a recursive-descent parser with backtracking for this grammar:

(1) \( S \rightarrow aSbS \)
(2) \( S \rightarrow bSaS \)
(3) \( S \rightarrow \epsilon \)

Write your parser in Python. You must have a function `backtrack` that takes a string and returns a tuple \((\text{True}, \text{Seq})\) when the string is in the language (\(\text{Seq}\) is the sequence your parser applies), or it returns a tuple \((\text{False}, [])\) when the string is not in the language. For example, `backtrack('abbb')` yields \((\text{False}, [])\) and `backtrack('abab')` yields \((\text{True}, [1, 2, 3, 3, 3])\).

You must submit your solution in the Homework server for autograding or you will receive a grade of 0 on this problem. Your Python file should be called `backtrack.py`. Assuming you want to name your zip file `backtrack.zip`, run `zip -r backtrack.zip backtrack.py`. Submit `backtrack.zip` to the server.

The Homework server runs Python 2.7.6. Problem will be available for submission by the end of this week.

Problem 2 (6pts). [From Scott] Consider the following LL(1) grammar for a simplified subset of Lisp:

\[
\begin{align*}
P & \rightarrow E \$$ \\
E & \rightarrow \text{atom} \\
E & \rightarrow ' E \\
E & \rightarrow ( E Es ) \\
Es & \rightarrow E Es \\
Es & \rightarrow \epsilon
\end{align*}
\]

atom, ', (, ), and $$ are the terminals (tokens), and \(P, E\) and \(Es\) are the nonterminals.

a) What is FOLLOW(\(Es\))? FOLLOW(\(E\))? PREDICT(\(Es \rightarrow \epsilon\))? 

b) Give a parse tree for the string `(cdr '([a b c])) $$`. Note: keyword `cdr` is an atom; identifiers a, b and c are atoms as well.

c) Give a left-most derivation for the string `(cdr '([a b c])) $$`.

d) Show a trace, in the style of Figure 2.20 (in the textbook), of a table-driven top-down parse of this same input.

e) Now consider a recursive descent parser running on the same input. At the point where the quote token ('') is matched, which recursive descent routines will be active (i.e., what routines will have a frame on the run-time stack)?

Problem 3 (10pts). [From Aho, Sethi, Ullman] Show that no LL(1) grammar can be ambiguous.

Problem 4 (6pts). [From Scott, modified] Write top-down and bottom-up grammars for the language consisting of all well-formed regular expressions. Give Kleene star the highest precedence and alternation the lowest precedence.
Problem 5 (6pts). For each grammar below, determine if it is LL(1), SLR(1), both or neither.

a) \( A \rightarrow 0 A 1 \mid 0 1 \)
b) \( A \rightarrow + A A \mid ^* A A \mid a \)
c) \( A \rightarrow A ( A ) A \mid \epsilon \)
d) \( A \rightarrow B a \mid b B c \mid d c \mid b d c \)
\( B \rightarrow d \)

Problem 6 (2pts). [From Aho, Sethi, Ullman] Show that the following grammar

\[
\begin{align*}
S &\rightarrow C a C b \mid D b D a \\
C &\rightarrow \epsilon \\
D &\rightarrow \epsilon 
\end{align*}
\]

is LL(1) but not SLR(1).

Problem 7 (10pts). Construct the CFSM for the following grammar, which generates regular expressions over symbols a and b:

\[
\begin{align*}
S &\rightarrow R \\
R &\rightarrow R ' | ' R \mid R R \mid R^* \mid ( R ) \mid a \mid b 
\end{align*}
\]

Note that the quoted vertical bar ’ | ’ is the “or” symbol, not a separator between alternatives.

Next, identify the parsing conflicts. Resolve the parsing conflicts in such a way that regular expressions will be parsed normally (i.e., * has highest precedence, followed by concatenation, followed by |, and all operators are left-associative).