HW2
50pts + 5pts extra credit
This is a team assignment. Form teams and submit in Submitty!
Posted Tuesday, September 11, 2018
Due Tuesday, September 18, 2018

Problem 1 (16pts). Write a top-down depth-first parser with backtracking for the language

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

Write your parser in Python. Include a function `dfparse` that takes a string, and returns a tuple `(True, Seq)` when the string is in the language, or `(False, [])` when the string is not in the language. `Seq` is the sequence of productions the depth-first parser applies. For example, `dfparse('abab')` yields `(True, [1, 2, 3, 3, 3])`, and `dfparse('abbb')` yields `(False, [])`.

You must name your file `dfparser.py`, and submit it under HW2 Python in Submitty. Our configuration assumes that your Python file includes function `dfparse` as specified above (and no prints!), which we call in interactive mode with various inputs. We run with Python 2.7.

Problem 2 (12pts). [From Scott, modified] 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) (4pts) What is FOLLOW(`Es`)? FOLLOW(`E`)? PREDICT(`Es → ϵ`)?

b) (4pts) 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) (4pts) 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 (6pts). [From Aho, Sethi, Ullman] Can a LL(1) grammar be ambiguous? (You do not need to write a proof, simply answer YES or NO, and briefly justify your answer.)
**Problem 4** (10pts). 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) \[ A \rightarrow B a \mid b B c \mid d c \mid b d c \quad B \rightarrow d \]

Note: You do not need to show tables or automata to justify your answer, just write, e.g., “Grammar XYZ is not LL(1) but is SLR(1).”

**Problem 5** (6pts). [From Aho, Sethi, Ullman] Show that the following grammar

\[
S \rightarrow C a C b \mid D b D a \\
C \rightarrow \epsilon \\
D \rightarrow \epsilon
\]

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

Note: In this problem, you **do need to justify** your answer by showing the LL(1) parsing table and relevant portions of the CFSM.

**Problem 6** (5pts extra credit. NO PARTIAL CREDIT. Only full credit for a complete solution will be assigned.). Construct the CFSM for the following grammar, which generates regular expressions over symbols \(a\) and \(b\):

\[
S \rightarrow R \\
R \rightarrow R \mid R R \mid R^* \mid ( R ) \mid a \mid b
\]

Note that the quoted vertical bar \(\mid\) 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).