HW1
50pts
Posted Friday, September 4, 2020
Due Friday, September 18, 2020

Note: Lecture 2 covers material for problems 1-3, Lectures 3 and 4 cover problems 4-5.

Problem 1 (10pts). Describe in English the languages denoted by the following regular expressions.

(a) (2pts) `0(0|1)*0`
(b) (2pts) `((`ϵ`|0)1*)*`
(c) (2pts) `(0|1)*0(0|1)(0|1)`
(d) (2pts) `0*10*10*10*`
(e) (2pts) `(00|11)*((01|10)(00|11)*(01|10)(00|11)*)*`

Note: Your description should be a high level characterization — one that would still make sense if we were using a different regular expression for the same language. For example, `(10*10*)*` should be described as "All strings of `0`'s and `1`'s, beginning with `1` and having even number of `1`'s." not as, for example, "The string of `1` followed by any number of `0`'s followed by a `1` followed by any number of `0`'s, repeated any number of times.".

Problem 2 (10pts). Consider the grammar

$$S \to \texttt{0}\ S\ \texttt{1}\ S \mid \texttt{1}\ S\ \texttt{0}\ S \mid \epsilon$$

(a) (2pts) Show that this grammar is ambiguous by constructing two different leftmost derivations for the sentence `0101`.
(b) (2pts) Construct the corresponding rightmost derivations.
(c) (3pts) Construct the corresponding parse trees.
(d) (3pts) The grammar generates strings with equal number of `0`'s and `1`'s. Does it generate *all* such strings? (You are not required to write a formal proof. If you answer YES, write a brief justification. If you answer NO, show a string that cannot be generated by the grammar.)

Problem 3 (10pts). The following is an ambiguous expression grammar with one unary operator * and n binary infix operators, $\theta_1, \theta_2, ..., \theta_n$, at n different levels of precedence:

$$E \to E\ \theta_1\ E \mid E\ \theta_2\ E \mid ... \mid E\ \theta_n\ E \mid E^* \mid (E) \mid \texttt{id}$$

(a) (2pts) Show that the grammar is ambiguous.
(b) (8pts) Construct an equivalent unambiguous grammar such that binary operators $\theta_1, \theta_2, ..., \theta_n$ are all *right-associative*. Unary operator * has the highest precedence. θ_n takes precedence over θ_{n-1}, θ_{n-1} takes precedence over θ_{n-2}, etc.

Problem 4 (10pts). Consider the following LL(1) grammar for a simplified subset of Lisp:

$$
\begin{array}{lcl}
P & \rightarrow & E \texttt{ \$\$} \\
E & \rightarrow & \texttt{atom} \\
E & \rightarrow & \texttt{'} \; E \\
E & \rightarrow & \texttt{(} \; E \; Es \; \texttt{)} \\
Es & \rightarrow & E \; Es \\
Es & \rightarrow & \epsilon
\end{array}
$$

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

(a) (3pts) What is FOLLOW(Es)? FOLLOW(E)? PREDICT($Es \rightarrow \epsilon$)?

(b) (3pts) Give a parse tree for the string `(cdr '(a b c)) $$`. Note: keyword `cdr` and identifiers `a`, `b`, and `c` are `atom`s.

(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 5 (10pts). For each grammar below, determine if it is LL(1) or not and if it is ambiguous or not. You do not need to justify your answer, just write YES or NO. As an example of the format for answers we are looking for: (x) LL(1): YES, Ambiguous: YES.

(a) (2pts) $A \rightarrow 0 \; A \; 1 \mid 0 \; 1$

(b) (2pts) $A \rightarrow + \; A \; A \mid * \; A \; A \mid \texttt{a}$

(c) (2pts) $A \rightarrow A \; (\; A \;) \; A \mid \epsilon$

(d) (2pts) $A \rightarrow B \; \texttt{a} \mid \texttt{b} \; B \; \texttt{c} \mid \texttt{d c} \mid \texttt{b d c} \qquad B \rightarrow \texttt{d}$

(e) (2pts) $S \rightarrow C\texttt{a}C\texttt{b} \mid D\texttt{b}D\texttt{a} \qquad C \rightarrow \epsilon \qquad D \rightarrow \epsilon$

Some additional problems if you are looking for an extra challenge. You DO NOT have to solve these problems. These problems won't be graded and won't bring credit even if you submit a solution.

Problem 1. The following grammar generates numbers in binary notation. C is the start symbol.

(1) $C \to C$ `0` $\mid A$ `1` $\mid$ `0`
(2) $A \to B$ `0` $\mid C$ `1` $\mid$ `1`
(3) $B \to A$ `0` $\mid B$ `1`

(a) Warmup: Construct a derivation that generates the binary notation of 21.
(b) Prove that the generated numbers are multiples of 3. (*Hint: Structural induction.*)
(c) Prove that all such numbers (i.e., nonnegative numbers that are multiples of 3) are generated by the grammar. (*Hint: Strong induction on string length, contradiction.*)
(d) Write a regular grammar G such that the numbers generated by G are multiples of 7 and all such numbers (i.e., nonnegative multiples of 7) are generated by G.

Problem 2. Write a top-down depth-first parser with backtracking for the language

(1) $S \to$ `a`S`b`S
(2) $S \to$ `b`S`a`S
(3) $S \to \epsilon$

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, [])`.