CMPSC 461: Programming Language Concepts
Assignment 3 Solution

Problem 1 [8pt] Write down the set of strings recognized by the following regular expressions. If the set is infinite, write down the first 6 shortest and unique elements.

a) (4pt) a((b|c)| b(d|e))

Solution: {ab, ac, abd, abe}

b) (4pt) (a^*|(ab|c)*) b

Solution:
{b, ab, cb, aab, abb, ccb, . . . }

Problem 2 [10pt]

a) (5pt) Build a regular expression that captures any URL in the following formats:

https://psu.instructure.com/
http://psu.instructure.com/courses/1234567
http://www.unencryptedwebsite1.com/
https://www.eecs.psu.edu/students/undergraduate/Research-Opportunities.aspx
https://bls.gov/

Assume that only numbers, letters, symbol '.' are used in top-level domain. Moreover, you need only match top-level domain ending with com, edu or gov. Note that your solution should make sense in the context of websites (i.e. a URL must contain words before and after every '.'). Any information after the top-level domain can be arbitrary, and hence, can be matched with a wild cast. Use '\.' for symbol '.'.

Solution: https?://(www\.)?([A-Za-z0-9]+\.)+(com|edu|gov)/?.*

b) (5pt) Build a regular expression that captures all non-empty sequences of letters other than "xyz". For your convenience, you may use a "not" operator that takes a set of letters as argument and matches any other letter. For instance, not(xyz) matches any letter other than x, y and z. Use '.' to match any letter.

Solution:

$$(\texttt{not(x)}.*) \mid \texttt{x}\big(\texttt{not(y)}.* \mid \epsilon\big) \mid \texttt{xy}(\epsilon \mid \texttt{z}.+ \mid \texttt{not(z)}.*)\big) \textbf{ OR}$$

$$(....+) \mid (\texttt{not(x)}..) \mid (.\texttt{not(y)}.) \mid (..\texttt{not(z)}) \mid . \mid ..$$

Problem 3 [15pt] Consider the context free grammar:

$$S ::= S\ S + \mid S\ S * \mid \texttt{x}$$

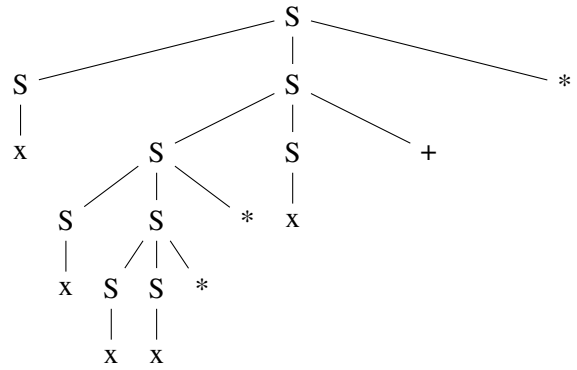
where x is a terminal for letter x, and a string "xxxx**x+*".

a) (5pt) Write down the rightmost derivation for the string.

Solution: S ⇒ S S * ⇒ S S S + * ⇒ S S x + * ⇒ S S S * x + * ⇒ S S S S * * x + *
⇒ S S S x * * x + * ⇒ S S x x * * x + * ⇒ S x x x * * x + * ⇒ x x x x * * x + *

b) (5pt) Draw the *concrete* parse tree for the string.

Solution:



c) (5pt) Is this grammar ambiguous or unambiguous? If it is ambiguous, give an equivalent unambiguous grammar; otherwise, briefly describe how to implement a parser that scans a string once, and then decides if the string is derivable from this grammar.

Solution: It is unambiguous. Multiple answers will be accepted.

One implementation is to check the characters from right to left, and build the parse tree from top to bottom. For each character being scanned, there is only one unique way to expand the current right-most nonterminal node in the tree. The string is accepted when the tree is complete exactly when all characters are scanned.

The parser can also scan from left to right, with a counter. The counter increases by one when "x" is scanned; decreases by one when "+" or "*" occurs. The string is rejected immediately when the counter is < 2 when "+" or "*" occurs, or > 1 when the whole string is scanned.

Problem 4 [10pt] For each of the following grammars state whether it is ambiguous or unambiguous. If it is ambiguous give an equivalent unambiguous grammar. If it is unambiguous, state all the precedences and associativities enforced by the grammar. Assume Id generates identifiers.

a) $S ::= S - T \mid T - S \mid T$
$T ::= Id + T \mid Id \mid (S)$

Solution: ambiguous.
$S ::= S - T \mid T$
$T ::= Id + T \mid Id \mid (S)$

b) $E ::= F \cap E \mid F \cup E \mid F$
$F ::= F \wedge G \mid F \vee G \mid G$
$G ::= Id \mid (E)$

Solution: unambiguous. $\cap, \cup$ are right-associative, $\wedge, \vee$ are left-associative. $\wedge$ and $\vee$ have higher precedence than $\cap$ and $\cup$.

Problem 5 [7pt] Consider the following grammar:
$A ::= \texttt{z}\ B\ \texttt{z}$
$B ::= C\ B \mid \texttt{x}\ \texttt{y} \mid \texttt{y}\ \texttt{x} \mid \epsilon$
$C ::= D\ E \mid E\ D$
$D ::= \texttt{x}\ \texttt{x} \mid \texttt{x}\ B\ \texttt{x} \mid \texttt{x}\ \texttt{x}\ B \mid B\ \texttt{x}\ \texttt{x}$
$E ::= \texttt{y}\ \texttt{y} \mid \texttt{y}\ B\ \texttt{y} \mid \texttt{y}\ \texttt{y}\ B \mid B\ \texttt{y}\ \texttt{y}$
where x, y, and z are terminals for letters x, y, and z respectively. Define in English the language that the grammar generates and explain why so.

Solution: by definition, A generates one B surrounded by z characters. B generates zero or more C's, an x and y, or nothing. C can generate D and E, where D generates exactly two more x characters, and E generates exactly two more y characters. Since each step enforces an equal number of x and y characters prior to a call to B or C, we can say that A generates all strings containing an equal amount of x and y characters that start and end with z.