CMPSC 461: Programming Language Concepts
Assignment 3 Solution

Problem 1 [10pt]

a) (5pt) Build a regular expression that captures any number in the following formats:
3.1452926
-212.45
126
1.9e10
For simplicity, assume only a nonnegative integer may appear after 'e'. Use '\.' for symbol '.'.

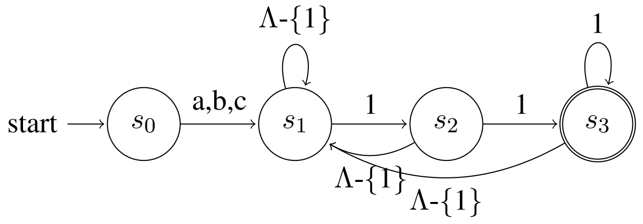
Solution: (-?)([1-9][0-9]*| 0)(\. [0-9]+)?(e[0-9]+)?

b) (5pt) Build a regular expression that captures all non-empty sequences of letters other than "for". 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`(abc) matches any letter other than a, b and c. Use '.' to match any letter.

Solution: (`not`(f).*) | f $\big($`not`(o).* | $\epsilon\big)$ | fo (ϵ | r.+ | `not`(r).*$\big)$

Problem 2 [10pt] Draw a deterministic finite state automaton (DFSA) which only accepts strings with at least three characters. The first character must be a letter, and the string must end with two '1's. Assume the alphabet Λ is $\{0,1,2,a,b,c\}$ for this problem.

Solution:



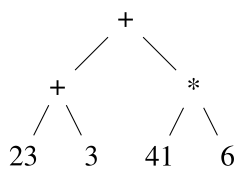
Problem 3 [15pt] Reverse Polish Notation (the syntax used by some early HP calculators) is a mathematical notation in which every operator follows all of its operands. It is also known as postfix notation. For example, `(1 2 +)` means `(1 + 2)` in the infix form used by C and Java, or `(+ 1 2)` in the prefix form used by Scheme. `(1 2 3 + +)` means `(1 + (2 + 3))` in the infix form. The following strings are all valid expressions:

```
33
1 3 /
23 3 + 41 6 * +
1 2 3 4 + + +
```

a) (5pt) Give a context-free grammar for valid expressions in the Reverse Polish Notation. You can assume the operators are just $\{+, -, *, /\}$ and that N is the terminal symbol for numbers.

$E ::= E\ E\ op \mid N$
$op ::= + \mid - \mid * \mid /$

b) (5pt) Draw the abstract syntax tree for string "`23 3 + 41 6 * +`" based on your grammar.



c) (5pt) Give the rightmost derivation for string “`23 3 + 41 6 * +`” based on your grammar.

$$\begin{aligned}
E &= E\ E\ op = E\ E\ + \\
&= E\ E\ E\ op + = E\ E\ E\ * \ + \\
&= E\ E\ N\ * \ + = E\ E\ 6\ * \ + \\
&= E\ N\ 6\ * \ + = E\ 41\ 6\ * \ + \\
&= E\ E\ op\ 41\ 6\ * \ + \\
&= E\ E\ +\ 41\ 6\ * \ + \\
&= E\ N\ +\ 41\ 6\ * \ + \\
&= E\ 3\ +\ 41\ 6\ * \ + \\
&= N\ 3\ +\ 41\ 6\ * \ + \\
&= 23\ 3\ +\ 41\ 6\ * \ +
\end{aligned}$$

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) $E ::= E + F \mid F$
$F ::= F * F \mid Id \mid (E)$

Solution: ambiguous.
$E ::= E + F \mid F$
$F ::= F * G \mid G$
$G ::= Id \mid (E)$

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 [5pt] Consider the following grammar:
$G ::= S$
$S ::= A\ M$
$A ::= \mathtt{a}\ E \mid \mathtt{b}\ A\ A$
$M ::= S \mid \epsilon$
$E ::= \mathtt{a}\ B \mid \mathtt{b}\ A \mid \epsilon$
$B ::= \mathtt{b}\ E \mid \mathtt{a}\ B\ B$
where `a` and `b` are terminals for letters a and b respectively. Define in English the language that the grammar generates.

Solution: by definition, G generates one to multiple A’s. A generates a string where `a` appears one more time than `b`. Hence, G generates all strings (of `a` and `b`) that contain more `a` than `b`.