COS 441 Assignment 1

The main goal of the assignment is to get used to working with syntax, inductive definitions, denotational semantics and simple proofs by induction.

Assume that b, h, n, i, and j are metavariables for the binary, hex, mixed, unary and natural numbers defined in lecture. Assume that e is the metavariable for arithmetic expressions defined in lecture.Assume binsem, mixsem, etc. are defined as in lecture.

Your work should be done on a printout of this assignment. Turn in your printout filled in.

Part I

Consider each of the functions and assume that the function argument has one of the types we defined in class (binary, hex, mixed, unary, arithmetic expression or natural number). Fill in the blanks with "yes" or "no" or "it's ambiguous".

a. evenbin(#) = true
   evenbin(b0) = true

   evenbin is total: _____ evenbin is inductive: _____

b. oddu(#) = false
   oddu(#S) = true
   oddu(bSS) = oddbin(b)

   oddu is total: _____ oddu is inductive: _____

c. iszero(#) = true

   iszero is total: _____ iszero is inductive: _____

d. grow(#) = 0
   grow(b0) = grow(b1) - 1
   grow(b1) = 1 + grow(b0)

   grow is total: _____ grow is inductive: _____

e. trumpet(add(e_1,e_2)) = drums(e_1) + trumpet(e_2)
   trumpet(mult(e_1,e_2)) = drums(e_1) - trumpet(e_2)
   trumpet(num(n)) = mixsem(n) + mixsem(n)
   drums(e) = expsem(e)
   trumpet is inductive: _____ drums is inductive: _____

Part II

This question is about Roman numerals. Refresh your memory (or learn about) Roman numerals here:

http://en.wikipedia.org/wiki/Roman_numerals

a. Give a BNF definition for the Roman numbers from 1 to 100 inclusive. The easiest way to do this to simply list them all (you can do that because there are only finitely many between 1 and 100). ie:

   r ::= I | II | III | IV | V | VI | … | C

   Obviously, that takes a long time. Your job is to give the most compact BNF definition that you can. Avoid repeating symbols or patterns whenever possible but be sure you represent every number and only numbers between 1 and 100.

b. Give a denotational semantics for your definition. In other words, define a total, inductive function "semr" from the syntax of roman numerals to the natural numbers.

Part III

Recall the unary numbers:

i ::= # | iS

Define even(i) as follows:

even (#) = true

even (#S) = false

even (iSS) = even(i)

Theorem: For every i, either even(i) or even(iS).

Proof (Fill in the blank parts to complete the proof): By induction on the syntax of i.

case #:

case #S:

case iSS:

Part IV

a. Define a function that reverses the order of the elements of a list:

reverse :: List -> List

reverse ([ ]) = _____

reverse (j : l) = _____

b. Prove this theorem:

theorem: for all l, reverse (l ++ [j]) = j : (reverse (l))

(recall that "[j]" is just a shorter way of writing "j : [ ]")

Proof: by induction on the structure of lists

case l = [ ]

case l = (j' : l')

IH: reverse ((j' : l') ++ [j]) = j : (reverse (j' : l'))

Part V

String patterns. Let u be a meta variable that ranges over uppercase letters A, B, C:

Upper case characters: u ::= A | B | C

Consider the following language of string patterns.

String patterns: r ::= nothing | emptystr | single c | concat(r, r) | or(r, sr)

The semantics of individual characters u is a string containing just the itself:

semchar(A) = "A"
semchar(B) = "B"
semchar(C) = "C"

The semantics of string patterns will be given as a function from syntax to the set of strings that the pattern matches. In the definition, assume that the operation s_1 ++ s_2 concatentates the two strings s_1 and s_2. We write s ϵ S when string s is a member of the set of strings S.

semre (nothing) = { }
semre(emptystr) = { "" }
semre(single c) = { semchar c }
semre(concat (r_1, r_2)) = { s_1 ++ s_2 | s_1 ϵ semre(r_1) and s_2 ϵ semre(r_2) }
semre(or (r_1, r_2)) = { s | s ϵ semre(r_1) or s ϵ semre(r_2) }

Examples:

semre (or (single A, single B)) = { "A", "B" }
semre (concat (single A, single B)) = { "AB" }
semre (concat (or (single A, single B), or (single B, single C))) = { "AB", "AC", "BB", BC" }

Compute the semantics of the following expressions.

a. semre(concat(single A, nothing)) = _____

b. semre(concat(or (single A, single B), or (single C, nothing))) = _____

c. semre(concat(or(single A, single B), or (single C, emptystr))) = _____

Part VI

Consider the following definition:

isempty (nothing) = true
isempty(emptystr) = false
isempty(single c) = false
isempty(concat (r_1, r_2)) = isempty (r_1) or isempty(r_2)
isempty(or(r_1, r_2)) = isempty (r_1) and isempty(r_2)

Theorem: For all r, if isempty(r) then semr(r) = { }.

Proof: By induction on the syntax of r.
Complete the proof here. Carry over to blank page if necessary – don't scrunch it up and make it hard to read!
Format the proof like you have been shown in 2-column style.
Be sure you have the correct number of cases.
Be sure you write down the induction hypothesis you are allowed to use in each case.
Even if you cannot figure out the details, showing that you know what the general form of the proof should look like will give you partial credit.

(Extra page)