Programming Assignments – 2

Programming Languages Essentials
PUCSD – January 2016-May 2016

Exercises in Induction, Recursion and Scope.

1. Some useful Scheme procedures
   (a) Define the length of a list recursively.
   (b) Use your definition from problem 1a to implement a Scheme procedure `my-length` that returns the length of a given list.
   (c) Write a Scheme procedure `my-append` that takes two lists `l1` and `l2` and returns a list that appends the list `l2` to the list `l1`.
   (d) Write a Scheme procedure `flatten` that takes a possibly nested list of numbers and returns an "unnested" list with all the elements. For example: `(flatten (1 (2 3) (4 5)))` returns `(1 2 3 4 5)`.
   (e) Define a Scheme procedure `my-reverse` that accepts a list and returns a list with the elements reversed. For example: `(my-reverse (list 1 2 3 4))` yields the list `(4 3 2 1)`.
   (f) Write a Scheme procedure `list-index` that takes a symbol *s* and a list of symbols *los* and returns a zero-based index of the first occurrence of the symbol *s* in *los*.
   (g) Generalising problem 1f. Write a Scheme procedure `list-indices` that takes a symbol *s* and a list of symbols *los* and returns a list of zero-based indices of all occurrences of *s* in *los*.
   (h) A 2-list is a list with two elements. Let *lst* be a list of 2-lists, i.e. each element of *lst* is a 2-list. Write a Scheme procedure `invert` that accepts a 2-list *lst* as an argument and returns a list of 2-lists with each 2-list of *lst* reversed. For example: `(invert '((a 1) (b 2) (c 3) (d 4)))` yields `((1 a) (2 b) (3 c) (4 d))`.

2. Inductive Specifications
   (a) Write a syntactic derivation that proves `(-7 . (3 . (14. ())))` is a list of numbers.
   (b) Write an inductive specification of a list of numbers.
   (c) Write an inductive specification of a list of characters.
   (d) Write an inductive specification of a list of booleans.
   (e) Write an inductive specification of a list of strings.
   (f) Write an inductive specification of a list of symbols.
   (g) Rewrite the inductive specifications for problems 2b-2f to allow nesting of lists.
   (h) Develop a single inductive specification for the data type based specifications in problem 2g.
   (i) Write an inductive specification for a binary tree of numbers. Does BNF capture all the aspects of a typical binary tree of numbers?

3. Recursively Specified Programs. This section is based on the ideas from the section "Deriving Programs from BNF Data Specifications" of the text.
   (a) Define the addition operation of two natural numbers recursively. Use the "`succ`" operation that yields the next natural number in the definition. (In other words, the `succ` operation is a primitive operation from which the addition operation can be built.) Scheme does not have a standard "`succ`" operation. So you may need to implement it as a helper operation by currying the primitive `+` operation.

(b) Implement your definition from problem 3a as a Scheme procedure.

(c) Define the multiplication operation of two natural numbers recursively. Use the "addition" operation from problem 3a.

(d) Implement your definition from problem 3c as a Scheme procedure.

(e) Define the exponentiation operation of two natural numbers recursively. Use the operations defined in problems 3a and 3c.

(f) Implement your definition from problem 3e as a Scheme procedure.

(g) Use the inductive definition of a binary tree from problem 2i and write Scheme programs for pre-order, in-order, and post-order traversals of trees. Compare your solution with problem 1d.

(h) Write a predicate "`list-of-numbers?`" that accepts a list and returns `#t` if it is a list of numbers, and `#f` otherwise.

(i) Write a predicate "`nth-element`" that takes a list and a zero based index n as arguments and returns the n^{th} element from the list.

(j) Write a Scheme procedure `remove-first` that accepts a symbol s and a list of symbols los as arguments. It returns a list of symbols where the first occurrence of s in los is removed. You might like to use the solution to problem 2f.

(k) Write a Scheme procedure `remove-all` that accepts a symbol s and a list of symbols los as arguments. It returns a list of symbols where all the occurrences of s in los are removed.

(l) Extend the specification in problem 2f to support nesting of symbol lists. For this problem we will call such lists of symbols with possibly nested lists as `slists`. Write a Scheme procedure called `substitute` that takes two symbols `old` and `new` and an slist `sl`, and returns an slist in which every occurrence of symbol `old` in slist `sl` is replaced by symbol `new`. Otherwise no changes are made.