Logic and Functional Programming

Labwork 4

March 21, 2021

Labworks related to lecture 4.

1. Define `foldr` with `foldl` and `reverse`, and indicate the runtime complexity of this definition.

2. Define `filter` with `foldr`.

3. Define `length` with `foldl`.

4. Define the following higher-order functions:

   (a) `(nest` f n`)` which takes as input a function $f : A \to A$ and $n \in \mathbb{N}$, and returns the function that maps $x \in A$ to the value of $\underbrace{f(\ldots f}_{n \text{ times}}(x)\cdots)$. If $n = 0$ then `(nest f 0)` should return the identity function `(lambda (x) x)`.

   (b) `(nestwhile` f v p`)` which takes as inputs a function $f : A \to A$, a predicate $p : A \to \textit{bool}$ and a value $v \in A$, and returns the value $w = f^n(v)$ for the smallest $n \in \mathbb{N}$ such that `(`p w`)` is `#f`.

5. Use `foldr` to define the variadic function

   `(comp` f_1 `...` f_n`)`

   which takes as inputs $n \geq 0$ unary functions $f_1, \ldots, f_n$ and returns the function that maps x to the value of

   `(`f_1 `... (`f_n x`)...)`

6. Define the function `(list->set lst)`, which drops the duplicate occurrences of elements from a list `lst`.

   Suggestion: express the computation of `(list->set lst)` as `(foldr` f `null lst)` with a suitable function f. You can use the built-in function `(member e l)` which is true if `e` is an element of list `l` and `#f` otherwise.

7. Consider the problem of counting the number of occurrences of every word in a document `d`. More precisely, let `d` be a list of symbols (the words of document `d`). We wish to define `(count-words d)` which returns the list of pairs `(cons` w n`)` where w is a string in `d`, and n is the number of occurrences of w in `d`. For example

```
> (count-words '(a b a b b c x z z x))
'((a . 2) (b . 3) (c . 1) (z . 2) (x . 2))
```

8. Give recursive definitions to the following variadic functions:

   (a) `(fmap-2` a b f_1 ... f_n`)` which computes `(list` w_1 ... w_n`)` where w_i is the value of `(`f_i a b`)` for every $1 \leq i \leq n$. For example:

   ```
   > (fmap-2 4 2 + * /)    > (fmap-2 4 2)
   '(6 8 2)                '()
   ```

   (b) `(inc?` a_1 ... a_n`)` which takes as inputs $n \geq 0$ integers and returns `#t` if and only if $a_1 < a_2 < \ldots < a_n$. For example:

   ```
   > (inc? 1 4 3)  > (inc?)  > (inc? 1)   > (inc? 4 7 8 9)
   #f              #t        #t            #t
   ```

   (c) `(dec?` a_1 ... a_n`)` which takes as inputs $n \geq 0$ integers and returns `#t` if and only if $a_1 > a_2 > \ldots > a_n$. For example:

   ```
   > (inc? 1 4 3)  > (dec?)  > (dec? 1)   > (dec? 9 7 5 0)
   #f              #t        #t           #t
   ```

   (d) Find the common pattern of computation of `inc?` and `dec?` and define

   `(sorted?` *cmp* a_1 ... a_n`)`

   which returns `#t` if and only if `(`*cmp* a_i a_{i+1}`)` is true for all $1 \leq i < n$.

   (e) `(monotone?` a_1 ... a_n`)` which takes as inputs $n \geq 0$ integers and returns `#t` if and only if

   $$a_1 < a_2 < \ldots < a_n \quad \text{or} \quad a_1 > a_2 > \ldots > a_n.$$

9. Define `(f-inc` n`)` which takes an input $n \in \mathbb{N}$ and computes the list of functions `(list` f_1 ... f_n`)` where, for all $k \geq 1$, `(`f_k x`)` returns the value of $x + k$.