Práctica 6

Programación Funcional, UNQ

Esquemas de alto orden

Aclaraciones:

- *Los ejercicios fueron pensados para ser resueltos en el orden en que son presentados. No se saltee ejercicios sin consultar antes a un docente.*
- *Recuerde que puede aprovechar en todo momento las funciones que ha definido, tanto las de esta misma práctica como las de prácticas anteriores.*
- *Pruebe todas sus implementaciones, al menos en una consola interactiva.*
- *Es sumamente aconsejable resolver los ejercicios utilizando primordialmente los conceptos y metodologías vistos en clase, dado que los exámenes de la materia evaluarán principalmente este aspecto. Si se encuentra utilizando formas alternativas al resolver los ejercicios consulte a los docentes.*
- *No dude en manifestar observaciones y críticas sobre los ejercicios de esta práctica, que con gusto serán recibidas por los docentes.*
- *Los ejercicios del anexo pueden obviarse, pero recuerde que aportan una compresión más profunda sobre los temas que aborda esta práctica. Considere resolverlos si se encuentra practicando para una instancia de evaluación y ya resolvió todos los anteriores.*

1. Fold

Defina las siguientes funciones sobre listas sin utilizar recursión explícita (i.e., utilizando `foldr`):

a) `length :: [a] -> Int`

b) `reverse :: [a] -> [a]`

c) `elem :: Eq a => a -> [a] -> Bool`

d) `map :: (a -> b) -> [a] -> [b]`

e) `filter :: (a -> Bool) -> [a] -> [a]`

f) `any :: (a -> Bool) -> [a] -> Bool`

g) `partition :: (a -> Bool) -> [a] -> ([a], [a])`

h) `nub :: Eq a => [a] -> [a]`

i) `take :: Int -> [a] -> [a]`

2. Fold1

Defina la función `foldr1 :: (a -> a -> a) -> [a] -> a`.

Luego defina las siguientes funciones sobre sin utilizar recursión explícita (i.e., utilizando `foldr1`):

a) `maximum :: Ord a => [a] -> a`

b) `minimum :: Ord a => [a] -> a`

3. Fold Matrix

Considere el tipo de las matrices representadas como listas de filas:

```
type Matrix a = [[a]]
```

Defina las siguientes funciones sobre matrices sin utilizar recursión explícita:

a) `repOkM :: Matrix a -> Bool`, que indica si se cumple el invariante de representación que indica que todas las filas tienen la misma longitud.

b) `getM :: Matrix a -> Int -> Int -> a`, que retorna el elemento en la coordenada (i, j) asumiendo que los índices están dentro de la dimensión de la matriz.

c) `mulM :: Int -> Matrix Int -> Matrix Int`, que multiplica cada elemento de la matriz por un número dado.

d) `addM :: Matrix Int -> Matrix Int -> Matrix Int`, que suma dos matrices (de iguales dimensiones) elemento a elemento.

4. Fold TipTree

De el tipo y defina la función `foldTT` que generaliza la recursión sobre la estructura `TipTree` (definidos en la práctica 4).

Luego defina las siguientes funciones sobre la estructura sin utilizar recursión explícita (i.e., usando `foldTT`):

a) `sizeTT :: TipTree a -> Int`

b) `heightTT :: TipTree a -> Int`

c) `walkthroughTT :: TipTree a -> [a]`

d) `mirrorTT :: TipTree a -> TipTree a`

e) `mapTT :: (a -> b) -> TipTree a -> TipTree b`

5. Fold Poli

De el tipo y defina la función `foldPoli` que generaliza la recursión sobre la estructura `Poli` (definida en la práctica 4).

Luego redefina las funciones sobre la estructura sin utilizar recursión explícita (i.e., usando `foldPoli`):

a) `eval :: Poli -> Int -> Int`

b) `mEscalar :: Poli -> Int -> Poli`

c) `sOptimize :: Poli -> Poli`

6. Fold Logical

De el tipo y defina la función `foldLogic` que generaliza la recursión sobre la estructura `Logical` (definida en la práctica 4).

Luego redefina las funciones sobre la estructura sin utilizar recursión explícita (i.e., usando `foldLogic`:

a) `eval :: Logical -> Valuation -> Bool`

b) `vars :: Logical -> [Int]`

c) `simp :: Logical -> Logical`

7. Fold Seq

De el tipo y defina la función `foldSeq` que generaliza la recursión sobre la estructura `Seq` (definida en la práctica 4).

Luego redefina las funciones sobre la estructura sin utilizar recursión explícita (i.e., usando `foldSeq`):

a) `lenSeq :: Seq a -> Int`

b) `revSeq :: Seq a -> Seq a`

c) `seq2List :: Seq a -> [a]`