Práctica 1

Programación Funcional, UNQ

Expresiones, valores y reducción

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.*

Muchas de las definiciones presentadas a continuación pueden ser encontradas en los módulos `Prelude` y `Data.List`. Por lo que algunas se cargan automáticamente al iniciar un entorno. Considere cambiar el nombre de las funciones definidas si desea testear su código y encuentra un problema con el interprete.

1. Booleanos

Dada la siguiente definición para los Booleanos:

```
data Bool = True | False
```

Definir las siguientes funciones:

a) `not :: Bool -> Bool`

b) `and :: Bool -> Bool -> Bool`

c) `or :: Bool -> Bool -> Bool`

d) `(==) :: Bool -> Bool -> Bool`

e) `ifThenElse :: Bool -> a -> a -> a`

2. Pares

Dada la siguiente definición para los Pares:

```
data (a,b) = (a,b)  -- notacion especial de Haskell
```

Definir las siguientes funciones:

a) `fst :: (a,b) -> a`

b) `snd :: (a,b) -> b`

c) `swap :: (a,b) -> (b,a)`

d) `(==) :: Eq a, Eq b => (a,b) -> (a,b) -> Bool`

e) `(<=) :: Ord a, Ord b => (a,b) -> (a,b) -> Bool`

3. Maybe

Dada la siguiente definición para los Maybe:

```
Maybe a = Nothing | Just a
```

Definir las siguientes funciones:

a) `isNothing :: Maybe a -> Bool`

b) `fromJust :: Maybe a -> a`

c) `liftMaybe :: a -> Maybe a`

d) `(==) :: Eq a => Maybe a -> Maybe a -> Bool`

e) `maybeApply :: (a -> b) -> Maybe a -> Maybe b`

4. Números naturales

Dada la siguiente definición para los Nat:

```
data Nat = Zero | Suc Nat
```

Definir las siguientes funciones:

a) `inc :: Nat -> Nat`

b) `add :: Nat -> Nat -> Nat`

c) `sub :: Nat -> Nat -> Maybe Nat`

d) `(==) :: Nat -> Nat -> Bool`

e) `(<=) :: Nat -> Nat -> Bool`

5. Listas

Dada la siguiente definición para las listas:

```
data [a] = [] | a : [a]  -- notacion especial de Haskell
```

Definir las siguientes funciones:

a) `null :: [a] -> Bool` (indica si está vacía)

b) `head :: [a] -> a` (retorna el primer elemento)

c) `tail :: [a] -> [a]` (retorna la lista con todos los elementos menos el primero)

d) `length :: [a] -> Int` (retorna la longitud de la lista)

e) `(++) :: [a] -> [a] -> [a]` (concatena dos listas)

f) `elem :: Eq a => a -> [a] -> Bool` (indica si un elemento pertenece a la lista)

g) `(!!) :: [a] -> Int -> a` (retorna el i-ésimo elemento de la lista indexado desde 0)

h) `reverse :: [a] -> [a]` (retorna la lista al revés)

6. Patrones

Indicar si los siguientes patterns son correctos:

a) `(x, y)`

b) `(1, y)`

c) `(n+1)`

d) `('a',('a',b))`

e) `(a,(a,b))`

f) `([]:[4])`

g) `(x:y:[])`

h) `[x]`

i) `([]:[])`

7. Reducción

Reduzca las siguientes expresiones hasta alcanzar la forma normal (indique la regla utilizada en cada paso):

a) `add (Suc Zero) (Suc Zero)`

b) `isNothing (sub Zero (Suc Zero))`

c) `fst (swap (True, False))`

d) `length [1,2,3]`

e) `[1,2,3] !! 2`

f) `not (elem 2 [1,2,3])`