Práctica 3

Programación Funcional, UNQ

Representación de datos con funciones

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. Lambda Booleans

Dada las siguientes definiciones que representan a los booleanos mediante funciones:

```
trueLam  = \x y -> x
falseLam = \x y -> y
```

Por ejemplo, con esta representación podemos escribir la negación (función `notLam`) de la siguiente manera:

```
notLam b = b falseLam trueLam
```

Definir las siguientes operaciones (que se comportan como sus contrapartes booleanas):

a) `orLam`, que toma dos Booleanos representados con funciones y retorna `trueLam` si cualquiera de los dos es `trueLam`.

b) `andLam`, que toma dos Booleanos representados con funciones y retorna `trueLam` sólo si ambos son `trueLam`.

2. Lambda Pairs

Considere una representación de pares mediante funciones, donde la función `mkPairLam` permite construir un par (e.g. `(mkPairLam 1 True)` es una expresión que denota el par `(1,True)`):

```
mkPairLam x y = \f -> f x y
```

Defina las siguientes operaciones:

a) `fstLam`, que dado un par representado con funciones proyecta su primer elemento (e.g. `fstLam (pairLam 1 True) = 1`).

b) `sndLam`, que dado un par representado con funciones proyecta su segundo elemento (e.g. `sndLam (pairLam 1 True) = True`).

c) `swapLam`, que dado un par representado con funciones retorna el par con los elementos invertidos (e.g., `swapLam (mkPairLam 1 True) = mkPair True 1`).

3. Lambda Maps

Considere la siguiente representación de Maps (diccionarios clave valor) como funciones:

```
type MapLam k v = k -> v
```

Defina las siguientes funciones:

a) `mkMapLam :: MapLam k v`, que construye un Map donde toda clave tiene asociado un valor por default.

b) `getMapLam :: k -> MapLam k v -> v`, que retorna el valor asociado a una clave dada.

c) `putMapLam :: Eq k => k -> v -> MapLam k v -> MapLam k v`, que retorna un map donde se sobreescribe el valor asociado a una clave dada.

4. Lambda Sets

Defina las siguientes operaciones para conjuntos representados por extensión (como `[a]`) y alternativamente por comprensión (como predicado `a -> Bool`) cuando sea posible.

a) `union :: Set a -> Set a -> Set a`

b) `intersect :: Set -> Set a -> Set a`

c) `complement :: Set a -> Set a`

d) `cardinal :: Set a -> Nat`

5. Fuzzy Sets

Un conjunto difuso (o fuzzy set en inglés) indica para cada elemento `x` un nivel de certeza de que `x` pertenezca al conjunto (i.e., un número real entre 0 y 1, donde 0 indica que con certeza el elemento no pertenece al conjunto y 1 indica que con certeza el elemento sí pertenece al conjunto). Considere la siguiente representación de fuzzy sets mediante funciones:

```
type Fuzzy a = a -> Float
```

Y defina la siguientes operaciones:

a) `belongs :: Fuzzy a -> a -> Float`

b) `complement :: Fuzzy a -> Fuzzy a`

c) `union :: Fuzzy a -> Fuzzy a -> Fuzzy a`

d) `intersect :: Fuzzy -> Fuzzy a -> Fuzzy a`

6. Church Numerals*

Dada la siguiente representación de números mediante funciones (donde un número `n` se representa mediante `n` composiciones de una función `f`, es decir, la repetición de `n` aplicaciones sucesivas de la función `f`):

```
zero = \f x -> x  -- 0 aplicaciones de la funcion f
succ n = \f x -> f (n f x)  -- 1 aplicacion mas de la funcion f
```

Defina la siguientes operaciones:

a) `church`, que a partir de un `Nat` retorna el numeral que lo representa.

b) `unchurch`, que a partir de un numeral retorna el `Nat` que lo representa.

c) `add`, que retorna el numeral que representa la suma de otros dos.

d) `mul`, que retorna el numeral que representa el producto de otros dos.