Programación Funcional

Ejercicios de Práctica Nro.3

Currificación

Aclaraciones:

- Los ejercicios siguen un orden de complejidad creciente, y cada uno puede servir a los siguientes. No se recomienda saltear ejercicios sin consultar antes a un docente.
- Recordar que se pueden aprovechar en todo momento las funciones ya definidas, tanto las de esta misma práctica como las de prácticas anteriores.
- Probar todas las 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 evalúan principalmente este aspecto. Para utilizando formas alternativas al resolver los ejercicios consultar a los docentes.

Ejercicio 1) Definir funciones

```
curry :: ((a,b) -> c) -> a -> b -> c
uncurry :: (a -> b -> c) -> (a,b) -> c
```

de tal manera que

- para toda función `f :: a -> b -> c`, se cumple
  ```
  curry (uncurry f) = f
  ```
- y para toda función `f' :: (a,b) -> c`, se cumple
  ```
  uncurry (curry f') = f'
  ```

Ejercicio 2) Reescribir las siguientes definiciones sin utilizar **`where`**, **`let`** o lambdas, y utilizando la menor cantidad de paréntesis posible.

a.
```
apply f = g
  where g x = f x
```
b.
```
twice f = g
  where g x = f (f x)
```
c.
```
id = \x -> x
```
d.
```
flip f = g
  where g x = h
        h y = (f y) x
```
e.
```
uflip f = g
  where g p = f (swap p)
```
f.
```
const = \x -> (\y -> x)
```
g.
```
compose = \f -> (\g -> (\x -> f (g x)))
```

Ejercicio 3) Indicar el tipo de cada una de las funciones del ejercicio anterior, utilizando también la menor cantidad posible de paréntesis.

Ejercicio 4) En las expresiones que siguen, colocar los paréntesis que están implícitos, manteniendo el significado de cada una de las expresiones, y dar el tipo de cada una de ellas, suponiendo dadas las definiciones de los ejercicios anteriores.

a. `apply apply apply`
b. `twice doble 2`
c. `twice twice twice swap`
d. `flip twice 1 doble`

Ejercicio 5) Reescribir las siguientes definiciones utilizando sólo lambdas (sin **`where`** ni **`let`**).

a.
```
appDup f = g
  where g x = f (x, x)
```
b.
```
appFork (f, g) = h
  where h x = (f x, g x)
```
c.
```
appPar (f, g) = h
  where h (x, y) = (f x, g y)
```
d.
```
appDist f = g
  where g (x, y) = (f x, f y)
```
e.
```
subst f = h
  where h g = k
    where k x = (f x) (g x)
```

Ejercicio 6) Indicar cuáles de las siguientes expresiones tienen tipo según el sistema de tipos de Hindley Milner. En el caso de que alguna sea incorrecta, ¿existirá una expresión que utilice las mismas partes, pero asociadas de forma diferente y que sí tenga significado? En el caso de que sí, escribir tal variante.

a. `compose (fst snd)`
b. `(uncurry curry snd)`
c. `(apply id) ((id apply) apply)`
d. `compose (compose doble doble)`
e. `(compose compose) doble doble`

Ejercicio 7) Dada la siguiente definición, indicar cómo podría reescribirse usando **`compose`** y **`id`**:

```
many :: Int -> (a -> a) -> a -> a
many 0 f x = x
many n f x = f (many (n-1) f x)
```

Ejercicio 8) Quitar de los siguientes tipos la mayor cantidad de paréntesis posible, sin cambiar su significado. En cada caso, escribir en castellano cómo debería leerse el tipo obtenido de forma correcta, y cómo con la convención de leerla como si estuviera no-currificada.

Por ejemplo, para `Int -> (Int -> Int)` las respuestas serían: `Int -> Int -> Int`, en castellano en forma correcta *"es una función que toma un entero y devuelve una función que toma otro entero y devuelve un entero"*, y en castellano usando la convención "*es una función que toma dos enteros y devuelve un entero"*.

a. `(Int -> Int) -> (Int -> Int)`
b. `(a -> (b -> c)) -> (a -> b) -> c`
c. `(a -> b, c -> d) -> ((a, c) -> (b, d))`
d. `((a, a) -> b) -> (a -> b)`
e. `(a -> (b -> c)) -> (b -> (a -> c))`
f. `(a -> b) -> ((a, a) -> (b, b))`
g. `(a -> b, a -> c) -> (a -> (b, c))`
h. `(a -> (b -> c)) -> ((a -> b) -> (a -> c))`
i. `a -> (b -> a)`

Ejercicio 9) Dar expresiones equivalentes a las funciones definidas a continuación utilizando funciones como `compose`, `flip`, etc. (dadas en los ejercicios anteriores) y sin utilizar lambas.

a. `cuadruple x = doble (doble x)`
b. `timesTwoPlusThree x = suma (doble x) 3`
c. `fourTimes f x = f (f (f (f x)))`

Ejercicio 10) Investigar la notación de sección de operadores.