Programación Funcional

Ejercicios de Práctica Nro.2

Sistemas de tipos

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) Indicar los tipos de las siguientes definiciones:

a.
```
first (x,y) = x
```
b.
```
apply f = g
  where g x = f x
```
c.
```
twice f = g
  where g x = f (f x)
```
d.
```
doble x = x + x
```
e.
```
swap (x, y) = (y, x)
```
f.
```
uflip f = g
  where g p = f (swap p)
```

Ejercicio 2) Dadas las definiciones anteriores, indicar el tipo de las siguientes expresiones:

a. `apply first`
b. `first (swap, uflip)`
c. `twice doble`
d. `twice twice`
e. `twice uflip`
f. `twice swap`
g. `uflip swap`
h. `(twice twice) swap`

Ejercicio 3) Dadas las siguientes definiciones y los siguientes tipos, asociar cada tipo con la función correspondiente.

a.
```
const x = g
  where g y = x
```
b.
```
appDup f = g
  where g x = f (x, x)
```

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

I. `(a -> b, c -> d) -> ((a, c) -> (b, d))`
II. `((a, a) -> b) -> (a -> b)`
III. `(a -> (b -> c)) -> (b -> (a -> c))`
IV. `(a -> b) -> ((a, a) -> (b, b))`
V. `(a -> b, a -> c) -> (a -> (b, c))`
VI. `(a -> (b -> c)) -> ((a -> b) -> (a -> c))`
VII. `a -> (b -> a)`

Ejercicio 4) Para cada una de las siguientes expresiones decidir si poseen tipo. Si es así indicar cuál es.

a. `1 && 2 == 2`
b. `1 + if 3 < 5 then 3 else 5`
c.
```
let par = (True, 4)
    in (if first par then first par else second par)
```
d. `(doble doble) 5`
e. `doble (doble 5)`
f. `twice first`
g. `(twice doble) doble`
h. `(twice twice) first`
i. `apply apply`

Ejercicio 5) Dar dos ejemplos de expresiones que tengan cada uno de los siguientes tipos:

a. `Bool`
b. `(Int, Int)`
c. `Char -> Int`
d. `(Int, Char) -> Bool`
e. `(Int -> Int) -> Int`
f. `(Bool -> Bool, Int)`
g. `a -> Bool`

Ejercicio 6) Para cada una de las siguientes expresiones, decir a cuál función del ejercicio 3 es equivalente. Ofrecer argumentos de por qué son equivalentes.

a.
```
\p -> let (f, g) = p
          in \x -> (f x, g x)
```
b.
```
\f -> (\g -> (\x -> f x (g x))
```
c.
```
\f -> (\x -> (\y -> (f y) x)
```
d.
```
\f -> (\px -> let (x, y) = px
                  in (f x, f y))
```
e.
```
\x -> (\y -> x)
```
f.
```
\pf -> let (f, g) = pf
           in \px -> let (x, y) = px
                         in (f x, g y)
```
g.
```
\f -> (\x -> f (x, x))
```

Ejercicio 7) Encontrar cuales de estas expresiones son equivalentes entre sí.

Sugerencia: utilizar funciones anónimas es una forma interesante de encontrar equivalencias entre expresiones que denotan funciones.

a.
```
appFork (id,id)
```
b.
```
\f -> appDup (appDist f)
```
c.
```
appDup id
```
d.
```
appDup appFork
```
e.
```
flip (appDup const)
```
f.
```
const (appDup id)
```