Programación Funcional

Prueba Escrita - 2022

Nombre:

CI:

Número de Prueba:

---

1. Dadas las siguientes definiciones:

$$\begin{aligned} &moo\ f\ k\ [\,] &&= 0 \\ &moo\ f\ k\ (x : xs) &&\mid f\ k == x = 1 + moo\ f\ (f\ k)\ (tail\ xs) \\ & &&\mid otherwise = 0 \end{aligned}$$

   ¿Cuál de las siguientes afirmaciones es correcta?

   (a) El código no compila
   (b) El resultado de evaluar $moo\ (+1)\ 1\ [1\,..\,6]$ es 1
   (c) El resultado de evaluar $moo\ id\ 1\ (take\ 10\ (repeat\ 1))$ es 10
   (d) Al intentar evaluar $moo\ (+2)\ 0\ [2\,..\,6]$ la ejecución da error

   Respuesta: d)

2. Dada la siguiente definición:

$$foo\ a\ b\ c = map\ (show \circ snd)\ \$\ zip\ [a, const\ c\ b, a + c]\ (repeat\ b)$$

   El tipo más general es:

   (a) $foo :: (Num\ a, Show\ a) \Rightarrow a \to a \to a \to [String]$
   (b) $foo :: (Num\ a, Show\ b) \Rightarrow a \to b \to a \to [String]$
   (c) $foo :: (Num\ a, Show\ a) \Rightarrow a \to a \to a \to [a]$
   (d) $foo :: (Num\ a, Show\ b) \Rightarrow a \to b \to a \to [b]$

   Respuesta: b)

3. Dada la siguiente definición:

$$twice\ f = f \circ f$$

   ¿Cuál de las siguientes opciones **NO** es correcta?:

   (a) $(twice\ (twice\ (+1)))$ está mal tipada
   (b) $(twice\ fst)$ está mal tipada
   (c) El tipo más general de $(twice\ twice)$ es $(a \to a) \to a \to a$
   (d) El tipo más general de $(twice \circ twice)$ es $(a \to a) \to a \to a$

   Respuesta: a)

4. Dado el siguiente programa:

$$
\begin{aligned}
main = \textbf{do}\ & x \leftarrow putStr\ \texttt{"hola"} \\
& y \leftarrow putStr\ \texttt{"chau"} \\
& putStr\ \$\ show\ (fst\ (x, y))
\end{aligned}
$$

Al ejecutarlo imprime:

(a) `()`

(b) `hola()`

(c) `hola`

(d) `holachau()`

Respuesta: d)

5. Dada la siguiente definición:

$$
\begin{aligned}
& \textbf{data}\ T\ a = D\ (T\ a)\ a\ (T\ a) \mid E \\
& foo\ E \quad\quad\ \_\ a = a \\
& foo\ (D\ l\ x\ r)\ f\ a = foo\ l\ f\ (foo\ r\ f\ (f\ x\ a))
\end{aligned}
$$

¿Cuál de las siguientes opciones **NO** es correcta?:

(a) Para todo t finito de tipo $T\ Int$, $foo\ t\ const\ 0$ retorna 0

(b) Para todo t finito de tipo $T\ Int$, $foo\ t\ (+)\ 0$ retorna la suma de sus valores

(c) Para todo t finito de tipo $T\ a$, $foo\ t\ (:)\ []$ retorna una lista con sus valores ordenados de acuerdo a una recorrida izquierda-derecha-raiz

(d) El tipo más general de foo es $T\ a \to (a \to b \to b) \to b \to b$

Respuesta: a)

6. Dada la siguiente definición:

$$
\begin{aligned}
& intercalate :: [a] \to [[a]] \to [a] \\
& intercalate\ xs = f \circ g \circ (h\ xs)
\end{aligned}
$$

¿Cuál de las siguientes implementaciones de f, g y h hace que $intercalate$ tome una lista (xs) y una lista de listas (xss), intercale xs entre los elementos de xss y concatene el resultado?

Ejemplos:
$intercalate$ `","` [`"primero"`, `"segundo"`, `"tercero"`] = `"primero,segundo,tercero"`
$intercalate\ [1,2]\ [[3,4],[5,6]] = [3,4,1,2,5,6]$
$intercalate\ [1,2]\ [] = []$

(a) $f = concat$
$g = id$
$h\ xs = map\ (xs{+\!\!+})$

(b) $f = concat$
$g = \lambda ys \to \textbf{if}\ null\ ys\ \textbf{then}\ ys\ \textbf{else}\ tail\ ys$
$h\ xs = foldr\ (\lambda x\ rs \to xs : x : rs)\ []$

(c) $f = id$
$g = concat$
$h\ xs = zipWith\ (+\!\!+)\ (repeat\ xs)$

(d) $f = \lambda ys \to \textbf{if}\ null\ ys\ \textbf{then}\ ys\ \textbf{else}\ tail\ ys$
$g = concat$
$h\ xs = map\ (xs{+\!\!+})$

Respuesta: b)

7. Consideramos la siguiente definición de expresiones enteras con suma y multiplicación.

$$
\begin{aligned}
&\textbf{data } \mathit{Exp} = \mathit{Lit}\ \mathit{Int} \mid \mathit{Add}\ \mathit{Exp}\ \mathit{Exp} \mid \mathit{Mul}\ \mathit{Exp}\ \mathit{Exp}\\
&\mathit{foldE} :: (\mathit{Int} \to a) \to (a \to a \to a) \to (a \to a \to a) \to \mathit{Exp} \to a\\
&\mathit{foldE}\ l\ \_\ \_\ (\mathit{Lit}\ x) \quad\ = l\ \ x\\
&\mathit{foldE}\ l\ a\ m\ (\mathit{Add}\ e1\ e2) = a\ \ (\mathit{foldE}\ l\ a\ m\ e1)\ (\mathit{foldE}\ l\ a\ m\ e2)\\
&\mathit{foldE}\ l\ a\ m\ (\mathit{Mul}\ e1\ e2) = m\ (\mathit{foldE}\ l\ a\ m\ e1)\ (\mathit{foldE}\ l\ a\ m\ e2)
\end{aligned}
$$

Dada una expresión e finita ¿cuál de las siguientes afirmaciones **NO** es correcta?

(a) $\mathit{foldE}\ (\mathit{const}\ 0)\ \mathit{Add}\ \mathit{Mul}\ e$ cambia el valor de todos los literales enteros de e por el valor 0

(b) $\mathit{foldE}\ (\mathit{const}\ 0)\ (\lambda x\ y \to x + y + 1)\ (+)\ e$ cuenta la cantidad de sumas que tiene e

(c) $\mathit{foldE}\ \mathit{id}\ (+)\ (*)\ e$ retorna el resultado de la evaluación de e

(d) $\mathit{foldE}\ \mathit{id}\ \mathit{const}\ \mathit{const}\ e$ retorna el entero de más a la izquierda de e

Respuesta: a)

8. Dada la siguiente definición:

$$
\begin{aligned}
&\mathit{as} = \mathit{iterate}\ (+1)\ 1\\
&\mathit{bs} = \mathit{as} : \mathit{zipWith}\ \mathit{drop}\ \mathit{as}\ \mathit{bs}
\end{aligned}
$$

Para cada una de las siguientes expresiones indique el resultado de su evaluación o si la misma diverge (si pone diverge en todas las opciones anula la pregunta).

(a) $\mathit{take}\ 5\ (\mathit{bs}\ !!\ 2)$ — [4,5,6,7,8]

(b) $\mathit{take}\ 5\ (\mathit{map}\ \mathit{head}\ \mathit{bs})$ — [1,2,4,7,11]

(c) $\mathit{head}\ \$\ \mathit{foldl}\ (\mathit{flip}\ (:))\ []\ (\mathit{map}\ \mathit{sum}\ \mathit{bs})$ — diverge

(d) $\mathit{head}\ \$\ \mathit{foldr}\ (:)\ []\ (\mathit{map}\ \mathit{sum}\ \mathit{bs})$ — diverge

(e) $\mathit{fst} \circ \mathit{head}\ \$\ \mathit{zip}\ \mathit{as}\ \mathit{bs}$ — 1

(f) $(\lambda xs \to \mathit{take}\ (\mathit{head}\ xs)\ xs)\ \$\ (\mathit{head} \circ \mathit{tail})\ \mathit{bs}$ — [2,3]

(g) $\mathit{take}\ 2\ \$\ \mathit{map}\ (\mathit{length} \circ \mathit{filter}\ (<2))\ \mathit{bs}$ — diverge

(h) $\mathit{head} \circ \mathit{snd} \circ \mathit{head}\ \$\ \mathit{zip}\ \mathit{as}\ \mathit{bs}$ — 1

9. Implemente, sin usar recursión, la función *composeN*:

   $composeN :: (b \to b) \to Int \to b \to b$

   que dadas una función f y un entero n, compone f consigo mismo n veces. Por ejemplo, *composeN* $(+1)$ 4 1 devuelve 5.

```
composeN f n = foldr (∘) id $ take n (repeat f)
```

10. Implemente usando *recursión explícita* la función:

    $group :: Eq\ a \Rightarrow [a] \to [[a]]$

    que dada una lista de elementos comparables devuelve una lista de listas con los mismos elementos y en el mismo orden, pero agrupados en listas que contienen los elementos contiguos iguales. Ejemplos:

    group `"Mississippi"` devuelve [”M”,”i”,”ss”,”i”,”ss”,”i”,”pp”,”i”]

    group $[1,2,2,3,4,4,2,5,5,5,1]$ devuelve [[1],[2,2],[3],[4,4],[2],[5,5,5],[1]]

    group $[\,]$ devuelve []

```
group []       = []
group (x : xs) = insGroup x (group xs)
  where insGroup x (ys@(y: _) : yss) | x == y    = (x : ys) : yss
                                     | otherwise = [x] : ys : yss
        insGroup x yss               = [x] : yss
```

    Implemente la misma función, pero *como un foldr*.

```
group = foldr insGroup []
  where insGroup x (ys@(y: _) : yss) | x == y    = (x : ys) : yss
                                     | otherwise = [x] : ys : yss
        insGroup x yss               = [x] : yss
```