Programación Funcional

Prueba Escrita - 2022

Nombre:

CI:

---

1. Dada la siguiente definición:

   $$foo\ a\ b = head\ [fst\ (a + b, \bot), fst\ (\bot, a < b)]$$

   El tipo más general es:

   (a) $foo :: (Num\ a, Ord\ a) \Rightarrow a \to a \to (a, Bool)$
   (b) $foo :: (Num\ a, Ord\ a) \Rightarrow a \to a \to a$
   (c) $foo :: (Num\ a, Ord\ a) \Rightarrow a \to a \to b$
   (d) No tiene

   Respuesta: b)

2. Dada la siguiente definición:

   $$dup\ x = (x, x)$$

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

   (a) $dup\ (4, 4) \equiv dup\ \$\ dup\ 4$
   (b) $fst\ (dup\ dup)\ \$\ fst\ (fst, dup)$ está mal tipada
   (c) El tipo más general de $dup \circ dup$ es $a \to ((a, a), (a, a))$
   (d) El tipo más general de $dup\ dup$ es $(a \to (a, a), a \to (a, a))$

   Respuesta: b)

3. Dada la siguiente definición:

   $$mult4 = map\ f\ (g\ lst)$$

   ¿Cuál de las siguientes implementaciones de f, g y lst **NO** hacen que $mult4$ devuelva la lista infinita de los múltiplos de cuatro a partir del cero? Recuerde que: $iterate\ f\ x = x : iterate\ f\ (f\ x)$

   (a) $f = (2*)$, $g = map\ (\lambda x \to x + x)$, $lst = iterate\ (+1)\ 0$
   (b) $f = id$, $g = map\ (`div`\ 2)$, $lst = iterate\ (+8)\ 0$
   (c) $f = id$, $g = foldl\ (\lambda acc\ x \to acc +\!\!+ [x])\ [\,]$, $lst = iterate\ (+4)\ 0$
   (d) $f = (+4)$, $g = filter\ even$, $lst = iterate\ (+4)\ (-4)$

   Respuesta: c)

4. Dada la siguiente definición:

$$\begin{aligned}&\textbf{data } \mathit{Either}\ a\ b = \mathit{Left}\ a \mid \mathit{Right}\ b\\ &\textbf{data } P = P\ I \mid \mathit{Cero}\\ &\textbf{data } I = I\ P\\ &\mathit{foo}\ (\mathit{Left}\ a) \quad\ = \mathit{Cero}\\ &\mathit{foo}\ (\mathit{Right}\ (I\ a)) = \mathit{foo}\ (\mathit{Left}\ a)\end{aligned}$$

Indique la opción correcta:

(a) El tipo más general de *foo* es $\mathit{Either}\ P\ a \to P$
(b) El tipo más general de *foo* es $\mathit{Either}\ P\ I \to P$
(c) El tipo más general de *foo* es $\mathit{Either}\ a\ I \to P$
(d) El código no compila

Respuesta: b)

5. Dadas las siguientes definiciones:

$$\begin{aligned}&\textbf{data } B\ a\ b = N\ [B\ a\ b] \mid C\ [a]\ [b]\\ &\mathit{bah}\ f\ (N\ bs) \quad\ = \mathit{concat}\ \$\ \mathit{map}\ (\mathit{bah}\ f)\ bs\\ &\mathit{bah}\ f\ (C\ as\ bs) = \mathit{zipWith}\ f\ as\ bs\\ &\mathit{puaj} = N\ [C\ (\mathit{repeat}\ 0)\ [1,2], N\ [C\ (\mathit{repeat}\ 0)\ [1,2]]]\end{aligned}$$

¿Cuál de las siguientes afirmaciones es correcta?

(a) El código no compila
(b) El resultado de evaluar $\mathit{bah}\ (*)\ \mathit{puaj}$ es $[0,0,0,0]$
(c) El resultado de evaluar $\mathit{bah}\ (*)\ \mathit{puaj}$ es $[0,0]$
(d) La evaluación de $\mathit{bah}\ (*)\ \mathit{puaj}$ no termina

Respuesta: b)

6. La función

$$\mathit{lookup} :: \mathit{Eq}\ k \Rightarrow k \to [(k,a)] \to \mathit{Maybe}\ a$$

realiza una búsqueda lineal en una lista de asociaciones de acuerdo a los siguientes ejemplos:
$\mathit{lookup}\ 4\ [(3,\texttt{'a'}),(4,\texttt{'h'}),(1,\texttt{'p'}),(4,\texttt{'m'})]$ retorna $(\mathit{Just}\ \texttt{'h'})$
$\mathit{lookup}\ 4\ [(3,\texttt{'a'}),(6,\texttt{'h'}),(1,\texttt{'p'}),(2,\texttt{'m'})]$ retorna $\mathit{Nothing}$
Se puede implementar *como* un *foldl* de la siguiente forma:

$$\mathit{lookup}\ k = \mathit{foldl}\ (\mathit{pacc}\ k)\ \mathit{Nothing}$$

Indique la opción que permite una implementación correcta de *lookup*.

(a) $\mathit{pacc}\ k\ \mathit{acc}\ (c,v) = \mathit{acc} \gg \textbf{if}\ k == c\ \textbf{then}\ \mathit{Just}\ v\ \textbf{else}\ \mathit{Nothing}$

(b) $\mathit{pacc}\ k\ \mathit{acc}\ (c,v) = \textbf{if}\ k == c\ \textbf{then}\ \mathit{Just}\ v\ \textbf{else}\ \mathit{acc}$

(c) $\mathit{pacc}\ k\ \mathit{Nothing}\ (c,v) = \textbf{if}\ k == c\ \textbf{then}\ \mathit{Just}\ v\ \textbf{else}\ \mathit{Nothing}$
$\mathit{pacc}\ k\ (\mathit{Just}\ u)\ (c,v) = \mathit{Just}\ u$

(d) $\mathit{pacc}\ k\ \mathit{acc}\ (k,v) = \mathit{Just}\ v$
$\mathit{pacc}\ k\ \mathit{acc}\ (c,v) = \mathit{acc}$

Respuesta: c)

7. Dada la siguiente definición:

$$cs = [1,1] : map\ (\lambda(x : y : xs) \to x + y : x : y : xs)\ cs$$

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) $head\ (foldl\ ((:) \circ head)\ []\ (tail\ cs))$ — diverge

(b) $head\ (foldr\ ((:) \circ head)\ []\ (tail\ cs))$ — 2

(c) $take\ 2\ \$\ filter\ ((>4) \circ head)\ cs$ — [[5,3,2,1,1],[8,5,3,2,1,1]]

(d) $take\ 2\ \$\ filter\ ((<2) \circ length)\ cs$ — diverge

(e) $take\ 4\ \$\ map\ head\ cs$ — [1,2,3,5]

(f) $map\ reverse\ \$\ take\ 4\ cs$ — [[1,1],[1,1,2],[1,1,2,3],[1,1,2,3,5]]

(g) $length \circ head \circ tail \circ tail\ \$\ cs$ — 4

(h) $length \circ tail \circ tail\ \$\ cs$ — diverge

8. Consideramos la siguiente definición de expresiones con valores de un tipo arbitrario.

$$\begin{aligned}
&\textbf{data}\ Exp\ val = O \mid V\ val \mid S\ (Exp\ val)\ (Exp\ val)\\
&foldE :: (a \to a \to a) \to (b \to a) \to a \to Exp\ b \to a\\
&foldE\ s\ v\ o\ O \quad\quad\quad = o\\
&foldE\ s\ v\ o\ (V\ c) \quad\quad = v\ c\\
&foldE\ s\ v\ o\ (S\ e1\ e2) = s\ (foldE\ s\ v\ o\ e1)\ (foldE\ s\ v\ o\ e2)
\end{aligned}$$

¿Cuál de las siguientes afirmaciones **NO** es correcta?

(a) Dada una expresión con tipo $Exp\ Int$, la aplicación $foldE\ min\ id\ 0$ retorna el mínimo de sus valores

(b) Dada una expresión con tipo $Exp\ a$, la aplicación $foldE\ (+\!\!+)\ (\lambda x \to [x])\ []$ devuelve una lista con sus valores, de izquierda a derecha

(c) Dada una expresión con tipo $Exp\ Int$, la aplicación $foldE\ (+)\ id\ 0$ suma sus valores

(d) Dada una expresión con tipo $Exp\ a$, la aplicación $foldE\ (+)\ (const\ 0)\ 1$ cuenta la cantidad de Os

Respuesta: a)

9. Implemente usando *recursión explícita* la función:

   $filterDup :: Eq\ a \Rightarrow [a] \to ([a], Int)$

   que dada una lista de elementos comparables se devuelve esa lista sin elementos repetidos y la cantidad de elementos eliminados. Si un elemento aparece múltiples veces se queda con la última aparición. Por ejemplo, *filterDup* $[1,2,3,4,5,2,3,4,3]$ devuelve el par $([1,5,2,4,3],4)$. Puede resultarle de utilidad la función

   $elem :: Eq\ a \Rightarrow a \to [a] \to Bool$

```
filterDup [] = ([], 0)
filterDup (x : xs) | elem x rs = (rs, n + 1)
                   | otherwise = (x : rs, n)
  where (rs, n) = filterDup xs
```

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

```
filterDup = foldr (λx (rs, n) → if elem x rs then (rs, n + 1) else (x : rs, n)) ([], 0)
```

10. Implemente, sin usar recursión, el operador

    $(\$>) :: [a \to b] \to a \to [b]$

    que aplica un argumento dado a cada función de una lista. Por ejemplo, $[(+5),(*2),(*0)]\$>8$ devuelve la lista $[13,16,0]$.

```
fs $> x = map ($ x) fs
```