Programación Funcional

Prueba Escrita - 2022

Nombre:

CI:

---

1. Dada la siguiente definición:

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

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

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

   Respuesta: d)

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

   Respuesta: d)

3. 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
   ```

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

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

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

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

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

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

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

Respuesta: c)

5. Dadas las siguientes definiciones:

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

¿Cuál de las siguientes afirmaciones es correcta?

(a) El código no compila

(b) La evaluación de $\mathit{bah}\ (*)\ \mathit{puaj}$ no termina

(c) El resultado de evaluar $\mathit{bah}\ (*)\ \mathit{puaj}$ es $[0,0,0,0]$

(d) El resultado de evaluar $\mathit{bah}\ (*)\ \mathit{puaj}$ es $[0,0]$

Respuesta: c)

6. Dada la siguiente definición:

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

Indique la opción correcta:

(a) El código no compila

(b) El tipo más general de *foo* es $\mathit{Either\ a\ I} \to P$

(c) El tipo más general de *foo* es $\mathit{Either\ P\ a} \to P$

(d) El tipo más general de *foo* es $\mathit{Either\ P\ I} \to P$

Respuesta: d)

7. La función

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

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

$$lookup\ k = foldl\ (pacc\ k)\ Nothing$$

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

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

(b) $pacc\ k\ acc\ (k, v) = Just\ v$
$pacc\ k\ acc\ (c, v) = acc$

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

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

Respuesta: c)

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

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

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

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

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

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

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

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

9. Dada la siguiente definición:

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

El tipo más general es:

(a) No tiene

(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) $foo :: (Num\ a, Ord\ a) \Rightarrow a \to a \to (a, Bool)$

Respuesta: b)

10. 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)
```