PLP - Primer Parcial - 1º cuatrimestre de 2006

Este examen se aprueba obteniendo al menos **70 puntos**. Poner nombre, apellido y número de orden en cada hoja, y numerarlas. Se puede utilizar todo lo definido en las prácticas y todo lo que se dio en clase, colocando referencias claras.

Ejercicio 1 - Programación Funcional (10 puntos)

Dos anagramas determinan una *transposición* de los caracteres de una tercer cadena. Por ejemplo, los anagramas "abcde" y "eabcd" definen la transposición en la que el último caracter de una cadena de 5 caracteres es movido al principio.

Definir la función `transpose :: String → String → String → String` que permite transponer los caracteres de una cadena según se especifica a través de dos anagramas. Por ejemplo, `transpose "UVWXYZ" "fedcba" "ecabdf"` debe devolver `"VXZYWU"`.

Ejercicio 2 - Programación funcional (30 puntos)

Consideremos una representación estándar de árboles binarios:

```
data BinTree a = Nil | Branch a (BinTree a) (BinTree a)
```

a) **(5 puntos)** Un operador de *recursión primitiva* es aquel en el que el resultado no depende únicamente de el o los llamados recursivos correspondientes sino también de los elementos que componen la estructura recursiva. Por ejemplo, el recursor primitivo para listas (llamado `recr` en las transparencias de la teórica) es:

```
primFold :: b → ([a] → a → b → b) → [a] → b
primFold z f [] = z
primFold z f (x:xs) = f xs x (primFold z f xs)
```

Definir el operador de recursión primitiva para árboles binarios:

```
primFoldBinTree :: (BinTree a → BinTree a → a → b → b → b)
                                                  → b → BinTree a → b
```

b) **(5 puntos)** Dar el tipo de `foldBinTree`, un esquema de recursión "clásico" (en el sentido que lo es `foldr` para listas) para árboles binarios, y definirlo usando `primFoldBinTree`.

c) **(10 puntos)** Un *árbol binario de búsqueda* es un árbol en el cual cada nodo es mayor o igual que todos los nodos en el subárbol izquierdo, y menor que todos los del subárbol derecho. De este modo, la operación de búsqueda de un elemento es más eficiente que en el caso de un árbol sin esta propiedad.

```
type SearchTree a = BinTree a
```

Definir **sin usar recursión explícita** la función

```
elemSearchTree :: Ord a ⇒ a → SearchTree a → Bool
```

que decida si un elemento está en el árbol o no aprovechando la propiedad de ser un árbol de búsqueda. ¿Esta función es realmente más eficiente que en un árbol ordinario? En otras palabras, ¿efectivamente la búsqueda se hace sólo sobre los subárboles que corresponden? Justificar brevemente.

d) **(10 puntos)** Definir **sin usar recursión explícita** la función

```
insertSearchTree :: Ord a ⇒ a → SearchTree a → SearchTree a
```

que inserta el elemento en el lugar apropiado del árbol, manteniendo la propiedad de ser un árbol binario de búsqueda.

Ejercicio 3 - PCF (35 puntos)

El presente ejercicio consiste en definir una lenguaje llamado PCF* que introduce construcciones que permiten hacer *pattern matching* sobre pares y sobre funciones.

PCF* resulta de modificar la sintaxis de árboles de términos de PCF de la siguiente manera: *se eliminan* las proyecciones de pares $\pi_1(M)$ y $\pi_2(M)$, y la aplicación $M\,N$ (dado que son expresables con las construcciones nuevas que se agregan) y *se agregan* construcciones de matching para pares y funciones:

- Construcción de matching para pares:

  Sintaxis Si M, P son árboles de términos PCF* y x, y son variables, entonces

  $$\mathit{match}\ M\ \mathit{with}\ \langle x, y\rangle\ \mathit{in}\ P$$

  es un árbol de términos de PCF*.

  Comportamiento

  Se evalúa M hasta que dé un par ordenado $\langle Q_1, Q_2\rangle$. Luego se evalúa P donde todas las ocurrencias libres de x e y se reemplazan por Q_1 y Q_2, respectivamente. El valor arrojado por ésta última expresión es el valor de toda la expresión.

  Ejemplos

  - $\mathit{match}\ \langle \underline{2}, \underline{3}\rangle\ \mathit{with}\ \langle x, y\rangle\ \mathit{in}\ x + y \Downarrow \underline{5}$
  - $\mathit{match}\ (\langle \lambda x.\langle x, \underline{3}\rangle)\,(\lambda x.x)\ \mathit{with}\ \langle x, y\rangle\ \mathit{in}\ x\,y \Downarrow \underline{3}$

  Observar que $\pi_1(M)$ puede definirse como syntactic sugar para $\mathit{match}\ M\ \mathit{with}\ \langle x, y\rangle\ \mathit{in}\ x$, y de manera análoga se puede proceder con $\pi_2(M)$.

- Construcción de matching para funciones:

  Sintaxis Si M, N, P son árboles de términos PCF* y x es una variable, entonces

  $$\mathit{match}\ M\ \mathit{using}\ N\ \mathit{with}\ x\ \mathit{in}\ P$$

  es un árbol de término PCF*.

  Comportamiento

  Se evalúa M hasta que dé una función $\lambda z.Q$. Luego se evalúa P donde se reemplazan todas las ocurrencias libres de la variable x por $Q\{z := N\}$.

  Ejemplos

  - $\mathit{match}\ (\lambda x.x + x)\ \mathit{using}\ \underline{2}\ \mathit{with}\ y\ \mathit{in}\ y + \underline{3} \Downarrow \underline{7}$
  - $\mathit{match}\ ((\lambda y.(\lambda x.x\,\underline{2} + \underline{y}))\,\underline{3})\ \mathit{using}\ (\lambda z.z)\ \mathit{with}\ y\ \mathit{in}\ y + \underline{3} \Downarrow \underline{8}$

Observar que la aplicación de PCF $(\lambda x.Q)\ R$ puede definirse como syntactic sugar para *match* $\lambda x.Q$ *using* R *with* y *in* y .

Para ambas construcciones definir:

a) **(15 puntos)** Las reglas de tipado.

b) **(10 puntos)** Las reglas de semántica operacional call-by-name *small-step* $(M \rightarrow_{CBN} N)$.

c) **(10 puntos)** Las reglas de semántica operacional call-by-name *big-step* $(M \Downarrow N)$.

Ejercicio 4 - Inferencia de tipos (25 puntos)
Resolver los siguientes ítems, justificando en todos los casos a través del árbol de inferencia.

a) **(10 puntos)** Mostrar que el algoritmo de inferencia falla con la siguiente expresión, donde id es una constante cuyo tipo es $id :: a \rightarrow a$.

$$(\lambda f.f\ f\ (f\ 3))\ id$$

b) **(10 puntos)** Mostrar que, sin embargo, el algoritmo de inferencia es capaz de inferir el tipo de la expresión

$$id\ id\ (id\ 3)$$

c) **(5 puntos)** ¿Qué sucede con la siguiente expresión cuando es alimentada al algoritmo de inferencia?

$$(\lambda x.x)\ (\lambda x.x)\ ((\lambda x.x)\ 3)$$