PLP - Recuperatorio Primer Parcial - 1º cuatrimestre de 2006

Este examen consiste de cuatro ejercicios que suman un puntaje máximo de 115 puntos: de los dos primeros, que suman 60 puntos en total, se computará un máximo de 45. El examen se aprueba obteniendo al menos **70 puntos**.

Resolver cada ejercicio en hojas separadas. 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 (15 puntos)

Resolver los siguientes items **sin usar recursión explícita**.

a) **(5 puntos)** Definir la función `groupBy :: (a → a → Bool) → [a] → [[a]]`, que dados un predicado p y una lista l, devuelva una lista de subcadenas maximales de l de la forma $[x_1 \dots x_n]$ tales que su concatenación sea igual a l y que valga $p\ x_i\ x_{i+1}$ para todo i.

`groupBy (<) [1,2,3,2,4,1]` $\leadsto$ `[[1,2,3],[2,4],[1]]`

b) **(5 puntos)** Definir la función `compactar :: Eq a ⇒[a] → [(a, Int)]`, que dada una lista xs devuelva una lista $[(x_1, k_1) \dots (x_m, k_m)]$ tal que la lista original xs sea equivalente a repetir k_i veces x_i para $i = 1, \dots, m$ y tal que $x_i \neq x_{i+1}$ para todo i.

`compactar ‘‘aaabbacco’’` $\leadsto$ `[(’a’, 3), (’b’, 2), (’a’, 1), (’c’, 2), (’o’, 1)]`

c) **(5 puntos)** Definir la función `expandir :: [(a, Int)] → [a]`, que dada una lista de pares $[(x_1, k_1) \dots (x_n, k_n)]$ devuelva la lista $[x_1, \dots, x_1, x_2, \dots, x_2, \dots, x_n, \dots, x_n]$ donde cada x_i aparece repetido k_i veces. Observar que esta función es la “inversa” de `compactar`, es decir que para cualquier `l`, `expandir (compactar l) ==l`

`expandir [(’a’, 3), (’b’, 2), (’a’, 1), (’c’, 2), (’o’, 1)]` $\leadsto$ `‘‘aaabbacco’’`

Ejercicio 2 - Programación funcional (45 puntos)

Dado un conjunto infinito numerable de *variables* $\mathcal{X} = \{x, y, z, \dots\}$ y un conjunto finito de *símbolos de función* (con aridad positiva) $\mathcal{F} = \{f_1, \dots, f_n\}$, definimos al conjunto $\mathcal{T}$ de *términos* inductivamente como:

- Toda variable es un término
- Si $t_1, \dots, t_n$ son términos y f es un símbolo de función de aridad n, entonces $f(t_1, \dots, t_n)$ es un término

Por ejemplo, si $\mathcal{F} = \{a, f, g\}$ con a de aridad 0, f de aridad 1 y g de aridad 2, entonces x, $f(g(x, y))$, $f(f(f(a)))$ son términos.

a) **(5 puntos)** Modelamos los términos en Haskell de la siguiente manera:

```
type Symbol = Char
data Term = Var String | Op Symbol [Term]
```

Definir los siguientes esquemas de recursión y recursión primitiva para términos. Definir `foldTerm` **sin usar recursión explícita** (usando `foldTermPrim`).

```
foldTermPrim :: (Symbol → Term → [b] → b) → (String → b) → Term → b
foldTerm :: (Symbol → [b] → b) → (String → b) → Term → b
```

b) **(5 puntos)** Una *sustitución* es una función $\sigma : \mathcal{X} \to \mathcal{T}$. Las sustituciones pueden aplicarse naturalmente a los términos, reemplazando cada variable x en el término por $\sigma(x)$.

```
type Subst = String → Term
```

Definir **sin usar recursión explícita** la función `applySubst :: Subst → Term → Term`, que aplica una sustitución a un término.

c) **(10 puntos)** Un término p se llama *pattern* si

- no es una variable
- no tiene variables repetidas

Dado un pattern p y un término t decimos que t *matchea* con p si $t = \sigma(p)$ para alguna sustitución σ. Por ejemplo, $f(g(a, x))$ matchea con el pattern $f(x)$ con la sustitución $\sigma = \{x/g(a, x)\}$. Sin embargo, $g(f(a), x)$ no matchea con el pattern $f(x)$ (aunque el subtérmino $f(a)$ sí lo hace con la sustitución $\sigma = \{x/a\}$).

Definir **sin usar pattern matching ni recursión explícita sobre listas** la función `match :: Term → Term → Bool`, que dado un término representando un pattern p y un término t indica si t matchea p.

d) **(15 puntos)** Definir **sin usar recursión explícita** `anyMatch :: Term → Term → Bool`, que dado un término p que representa un pattern y un término t indica si algún subtérmino de t matchea con p.

e) **(15 puntos)** Definir **sin usar pattern matching ni recursión explícita sobre listas** la función `computeSubs :: Term → Term → Subst`, que dado un término representando un pattern p y un término t devuelve la sustitución s tal que `applySubst s p ==t`. Puede asumirse que t matchea con p (observar que hablamos del término t completo, no de uno de sus subtérminos).

Pueden usarse las siguientes funciones:

```
-- Sustitucion identidad
idSubs:: String → Term
idSubs s = Var s

-- Union de sustituciones con dominio disjunto
joinSubs :: Subst → Subst → Subst
joinSubs s t = λx→ case s x of
                   (Var x) → t x
                   _ → s x
```

Ejercicio 3 - PCF (30 puntos)

El presente ejercicio consiste en extender el lenguaje PCF con *registros* para obtener PCF*. Comenzamos extendiendo las expresiones de tipos de PCF. Dada una lista finita de *etiquetas* $\{l_1, \ldots, l_m\}$ las expresiones de tipos de PCF* son:

$$\sigma \quad ::= \quad nat \,|\, bool \,|\, \sigma \to \sigma \,|\, \{l_1 : \sigma_1, \ldots, l_n : \sigma_n\}$$

La expresión $\{l_1 : \sigma_1, \ldots, l_n : \sigma_n\}$ es el tipo de los registros cuyo campo l_i tiene tipo σ_i, $i \in 1..n$. Se asume que el orden de los campos es irrelevante. Por ejemplo

- $\{dni : int, edad : int\}$
- $\{x : int, y : int, distancia : int \to int \to int\}$

Luego extendemos el conjunto de árboles de términos agregando dos nuevas construcciones, una para crear registros y otra para proyectar campos de un registro:

- Construcción de creación de registros:

  Sintaxis Si $M_1, \ldots, M_n$ son árboles de términos de PCF* y $l_1, \ldots, l_n$ son etiquetas distintas, entonces
  $$[l_1 = M_1, \ldots, l_n = M_n]$$
  es un árbol de términos de PCF*.

  Comportamiento
  Un árbol de término de la forma $[l_1 = M_1, \ldots, l_n = M_n]$ se encuentra en forma normal.

  Ejemplos
  - $[dni = 23, edad = 30]$
  - $[x = 2, y = 7, distancia = \lambda u : int.\lambda v : int....]$

  donde ... es una expresión que calcula la distancia.

- Construcción de proyección de campos de un registro:

  Sintaxis Si M es un árbol de términos de PCF* y l_i es una etiqueta, entonces
  $$M.l_i$$
  es un árbol de términos de PCF*.

  Comportamiento
  Se evalúa M hasta que dé un registro $[l_1 = M_1, \ldots, l_n = M_n]$. Luego se evalúa M_i y se retorna el valor del mismo como valor de la expresión entera.

  Ejemplos
  - $(\lambda r : \{x : int, y : int\}.r.x + r.y)\,[x = \underline{2}, y = \underline{3}] \Downarrow \underline{5}$

Para ambas construcciones definir:

a) **(10 puntos)** Las reglas de tipado.

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

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

Ejercicio 4 - Inferencia (25 puntos)

El presente ejercicio consiste en extender el algoritmo de inferencia visto en clase para el lenguaje resultante de incluir una construcción de matching para pares:

- Construcción de matching para pares:

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

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

  es un árbol de términos de PCF*.

  Regla de tipado

  $$\frac{\Gamma \rhd M : \rho \times \sigma \quad \Gamma, x : \rho, y : \sigma \rhd P : \tau}{\Gamma \rhd match\ M\ with\ \langle x, y\rangle\ in\ P : \tau}$$

a) **(15 puntos)** Extender el algoritmo de inferencia acordemente. Asumir que el mismo fue extendido para tipar correctamente pares ordenados $\langle M, N\rangle$, es decir que el tipo producto cartesiano $(\sigma \times \tau)$ ya está contemplado.

b) **(10 puntos)** Dar un ejemplo de un término en el que la inferencia es exitosa y otro donde la inferencia falla. Justificar con el árbol de inferencia en ambos casos.