Parcial Llenguatges de Programació

Grau en Enginyeria Informàtica

26 Novembre 2014

Per accedir al racó aneu a `https://examens.fib.upc.edu`
Cal que lliureu via racó el codi amb els comentaris que considereu necessaris en un arxiu "examen.hs" executable en l'entorn ghci sense activar cap opció addicional (només fent `ghci examen.hs`) i que solucioni els problemes que es llisten a continuació.
Imprimirem la vostra solució amb la comanda

```
a2ps -1 -r -f 8 --borders=0 --no-header --header=Examen examen.hs -o examen.ps
```

comproveu que el que envieu té una indentació correcte i no es surt dels límits de la pàgina.

Cal que al començar la solució de cada problema afegiu una línia comentada indicant el problema i subapartat que ve a continuació. Per exemple,

```
-- Problema 3.1
```

Es valorarà l'ús que es faci de funcions d'ordre superior predefinides. Ara bé, en principi només s'han d'usar les de l'entorn Prelude, és a dir, no heu de fer cap import.

Problema 1 (2 punts): *Prefixos i Sufixos.* Implementeu la funció `prefsufs::[a]->[[a]]` que donada una llista retorna la llista amb tots prefixos (no buits) en ordre creixent respecte a la mida seguit dels sufixos (no buits) en ordre decreixent respecte a la mida.

```
prefsufs [1,2,3,4] = [[1],[1,2],[1,2,3],[1,2,3,4],[2,3,4],[3,4],[4]]
```

Problema 2 (2 punts): *Punt fix.* Donada una funció `f :: a -> a` i un punt x_0, podem iterar la seva aplicació i obtenir la seqüència $x_0, f(x_0), f(f(x_0)), f(f(f(x_0))), \ldots$ Es diu que x és un punt fix de f si $f(x) = x$.

Apartat 2.1: Implementeu una funció `fixedPoint`, que donada una funció `f` i un punt `x0`, calculi el punt fix de `f` començant des de `x0`. Per exemple,

```
fixedPoint (`div` 2) 15 = 0
```

Nota 1: No podeu utilitzar recursivitat, però sí funcions d'ordre superior.
Nota 2: Si no hi ha cap punt fix, fixedPoint no acaba.

Apartat 2.2: El mètode de Newton per calcular arrels quadrades és basa en que l'arrel quadrada de y és el punt fix començant en 1.0 de la funció $f(x) = \frac{(\frac{y}{x}+x)}{2.0}$.
Definiu l'arrel quadrada utilitzant el mètode de Newton.

Problema 3 (4 punts): *Polinomis.* Volem representar els polinomis sobre un única variable amb llistes que contenen els coeficient de forma creixent segons el grau. Per exemple, el polinomi sobre els enters $x + 4x^3$ el representarem amb la llista $[0, 1, 0, 4]$ o el polinomi sobre els reals $1.1 - 3.5x^2$ amb la llista $[1.1, 0.0, (-3.5)]$. Noteu que la llista $[1.1, 0, (-3.5), 0, 0]$ també representa el mateix polinomi.

Apartat 3.1: Definiu un nou tipus polimòrfic `Polynomial` seguint la representació indicada al paràgraf anterior per polinomis on el tipus dels coeficients és general. Feu que el tipus `Polynomial` sigui `instance` de la classe `Eq` on (==) és la igualtat dels coeficients sense tenir en compte els sufixes amb tot zeros.

Noteu que, com que usareu el `0`, cal que el tipus dels coeficients sigui de la *class* `Num`, a més d'altres *class* que us puguin caler.

Apartat 3.2: Feu que el tipus Polynomial sigui `instance` de la classe `Num`, amb la condició que el tipus dels coeficients sigui de la class `Num`. Per això heu d'implementar les següents cinc operacions:

```
(+)  :: a -> a -> a                 abs    :: a -> a
(*)  :: a -> a -> a                 signum :: a -> a
fromInteger :: Integer -> a
```

On la suma i el producte de polinomis són les definicions estàndard, el valor absolut d'un polinomi s'obté fent el valor absolut dels seus coeficients, el `signum` d'un polinomi s'obté fent el `signum` del terme independent i el `fromInteger` ha de crear un polinomi que només té el terme independent i que s'obté amb l'`Integer` que ens passen.

Per al producte, encara que no és obligatori, es valorarà que useu la funció del Problema 1, tenint en compte que el producte dels polinomis $a_0 + a_1x + \ldots, a_nx^n$ i $b_0 + b_1x + \ldots, b_nx^n$ és $\Sigma_{i=0}^{2n}(\Sigma_{j=0}^{i} a_j b_{i-j})x^i$.

Nota. Un cop fet l'`instance`, podeu aplicar, per exemple, les operacions `sum` o `product` que tenen tipus `Num a => [a] -> a` a llistes de polinomis.

Problema 4 (2 punts): *Arbres AndOr alternats*

Els arbres andor alternats d'objectes (d'un tipus genèric o polimòrfic) contenen conjuncions o disjuncions al nodes i tenen objectes a les fulles. L'arbre sempre té un node conjunció a l'arrel (o una fulla) i, a partir d'aquest moment, alterna nodes disjunció i conjunció fins arribar a les fulles.

Apartat 4.1: Definiu el tipus arbre `AndOr` de manera que garanteixi l'alternança. Podeu usar un altre data auxiliar, per aconseguir-ho. Per exemple, pel cas particular que les fulles tenen números, tenim que

```
(ALeaf 5)
(Nand [ (Nor [ ALeaf 5, (Nand [(Nor [ALeaf 2, ALeaf 1]), OLeaf 4 ])]), (Nor [ ALeaf 3, ALeaf 4 ]) ])
```

són del tipus, però

```
(Nand [ (Nand [(Nor [ALeaf 2, ALeaf 1]), OLeaf 4 ]), OLeaf 3])
```

no és del tipus.

Apartat 4.2: Feu una funció `eval :: (a -> Bool) -> (AndOr a) -> Bool` que donat un predicat p (que s'aplica a les fulles) avalua a cert o a fals seguint les conjuncions i les disjuncions de l'arbre.

Com a exemples, amb

```
let e1 = (Nand [(Nor [ ALeaf 5,(Nand [(Nor [ALeaf 2, ALeaf 1]), OLeaf 4 ])]),(Nor [ALeaf 3, ALeaf 4])])
eval even e1
eval (==1) e1
```

obtindreu

```
True
False
```