Parcial Llenguatges de Programació

Grau en Enginyeria Informàtica

13 Maig 2014

Per accedir al racó aneu a `https://examens.fib.upc.es`
Cal que lliureu via racó el codi amb els comentaris que considereu necessaris en un arxiu ".hs" executable en l'entorn ghci i que solucioni els problemes que es llisten a continuació.

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 2.1
```

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 ha de caldre cap import. Podeu definir funcions auxiliars si us calen.

Problema 1 (3 punts): *Llistes.* Heu de crear les operacions següents:

Apartat 1.1: `inflists::[[Integer]]`
Que retorna la llista infinita de llistes infinites d'enters que comencen successivament per 1, per 2, per 3, etc. És a dir, la primera és `[1,2,3,...]`, la segona `[2,3,4,...]`, etc.

Apartat 1.2: `takeLESum::Integer -> [Integer] -> [Integer]`
Que retorna el prefix més llarg de la llista que suma igual o menys que l'enter donat.

```
takeLESum  8 [8,9,10,11] = [8]
takeLESum  7 [2,3,5,8] = [2,3]
takeLESum  18 [4,5,9,3] = [4,5,9]
```

Podeu assumir que tots els enters són positius.

Apartat 1.3: Usant les dues funcions anteriors crea l'operació
`consecutSum::Integer -> [Integer]`
que retorna la llista amb els nombres positius més petits consecutius que sumen l'enter positiu donat.

```
consecutSum 8  = [8]
consecutSum 33 = [3,4,5,6,7,8]
consecutSum 24 = [7,8,9]
```

Problema 2 (3.5 punts): *Divideix i vençe.* Heu de crear les operacions següents:

Apartat 2.1: Una operació d'ordre superior
`dc::(a -> Bool) -> (a -> b) -> (a -> [a]) -> (a -> [b] -> b) -> a -> b`
que donades (1) una funció que indica si estem en un cas *trivial*, (2) una funció que *resol* els casos trivials, (3) una funció que *parteix* un problema en una llista de subproblemes, (4) una funció que *combina* el problema i la llista de solucions dels subproblemes per obtenir la solució i (5) un *problema*, resol el problema mitjançant l'esquema de *divideix i vençe*. És a dir,

`dc trivial resol parteix combina problema`

Noteu que en general `parteix` ha de retornar una llista, encara que en la majoria de casos és parteix només en dues parts. Igualment, noteu que en general, l'operació `combina` usa les solucions als subproblemes, però també tot o part del problema original.

Apartat 2.1: Utilitzant l'operació `dc`, creeu la funció
`mergesort::Ord a => [a] -> [a]`
que implementi l'algorisme d'ordenació per fusió. Per això, heu de definir totes les funcions auxiliars que us calguin per a fer la crida a `dc`.

Problema 3 (3.5 punts): *Expressions*
Considereu el tipus genèric `Expressio`, que permet emmagatzemar expressions sobre un tipus `a`, construïdes amb valors d'aquest tipus com a fulles i amb operacions binàries (amb tipus `a -> a -> a`) i unàries (amb tipus `a -> a`). Per exemple, si `a` és `Int` tenim

```
Unari (+1) (Binari (*) (Unari (abs) (Binari (-) (Fulla 3) (Fulla 5))) (Fulla 2))
```

i si `a` és `String` tenim

```
Unari (reverse) (Binari (++) (Fulla "ani") (Fulla "mal"))
```

com valors del tipus `Expressio Int` i `Expressio String` respectivament.

Apartat 3.1: Definiu el tipus genèric `Expressio` que permeti crear expressions generals amb operadors binaris i unaris com les donades.
Creeu l'operació `aval::Expressio a -> a`, que avalua una expressió. Per exemple,

```
aval (Unari (reverse) (Binari (++) (Fulla "ani") (Fulla "mal")))
```

retorna `"lamina"`.
Tenint en compte que considerem que dues expressions són iguals si s'avaluen igual, indiqueu que `Expressio` és una `instance` de la classe `Eq` on (==) és aquesta igualtat.

Apartat 3.2: Considereu ara el tipus genèric `NExpressio`, en que tenim `NFulla`, `NUnari` i `Nari` com a constructors. Els dos primers són com els d'abans (`Fulla` i `Unari`) però sobre `NExpressio`. El tercer, en canvi, usa una funció binària (amb tipus `a -> a -> a`) per acumular el resultat d'una llista d'expressions. Per exemple, si `a` és `Int` tenim

```
NUnari (+1) (Nari (*) [(Nari (+) [(NFulla 3),(NFulla 5),(NFulla 6)]),(NFulla 2)])
```

com valor del tipus `NExpressio Int`.
Definiu el tipus genèric `NExpressio` i la funció `naval::NExpressio a -> a`. Així

```
naval (NUnari(+1)(Nari(*)[(Nari(+)[(NFulla 3),(NFulla 5),(NFulla 6)]),(NFulla 2)]))
```

retorna `29`.
Podeu assumir que la llista d'expressions del constructor Nari no serà mai buida i que si té un únic element l'expressió avalua com aquest element (independentment de l'operació).
Finalment indiqueu que `NExpressio` és una `instance` de la classe `Eq` on (==) és la mateixa que abans, és a dir, dues expressions són iguals si s'avaluen igual.