Métodos de Programação I

2.º Ano da LMCC (7003N5) + LESI (5303O7)
Ano Lectivo de 2003/04

Exame (Época Especial) — 14 de Setembro 2004
17h00

NB*: Esta prova consta de **8** alíneas que valem, cada uma, 2.5 valores.*

PROVA SEM CONSULTA (2 horas)

Questão 1 A seguinte função codifica um algoritmo de ordenação clássico, normalmente conhecido pelo nome de *heapsort*.

$$\mathsf{hsort} = [\mathsf{id}, \mathsf{cons} \circ (\mathsf{id} \times (\mathsf{merge} \circ (\mathsf{hsort} \times \mathsf{hsort})))) \circ \mathsf{aux} \circ \mathsf{cons}] \circ \mathsf{out}_{\mathsf{RList}}$$

em que merge é a usual função de fusão de listas ordenadas; e aux é definida como se segue:

```
aux [h]   = (h,([],[]))
aux (h:t) = let (y,(l,r)) = aux t
            in if h<y then (h,(y:r,l))
                      else (y,(h:r,l))
```

1. Escreva uma definição de hsort no estilo *pointwise*, justificando todos os passos para a sua obtenção.
2. Defina hsort como um hilomorfismo e desenhe o diagrama correspondente.
3. Declare em Haskell o tipo intermédio deste hilomorfismo e calcule o valor deste tipo que corresponde ao cálculo de $\mathsf{hsort}[10, 5, 9, 1, 8, 2, 4, 6, 7]$?

Questão 2 Considere a função que realiza a partição de uma lista s em duas outras listas que recolhem, respectivamente, os elementos de s que verificam ou falham um determinado predicado p:

$$\mathsf{partition}\,\mathsf{p} \;=\; \langle \mathsf{filter}\ \mathsf{p}, \mathsf{filter}\ \neg\mathsf{p} \rangle$$

Exprima esta função como um catamorfismo, por aplicação da respectiva lei de fusão.

Questão 3 Considere a seguinte função que utiliza um parâmetro de acumulação para calcular a média de uma lista de inteiros:

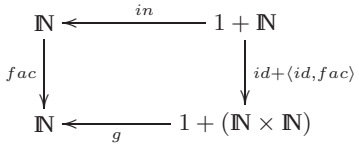
```
media' :: [Int] -> (Int, Int) -> Int
media' [] (a,b) = a/b
media' (h:t) (a,b) = media' t (h+a,b+1)
```

Note que a média de uma lista l é calculada como $\mathsf{media}'\ l\ (0,0)$. Defina esta função media' como um catamorfismo.

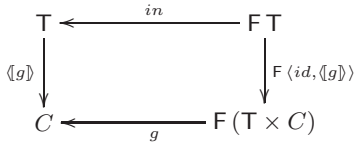
Questão 4 A habitual definição da função factorial

$$\begin{array}{rcl} fac\,0 & = & 1 \\ fac(n+1) & = & (n+1) \times fac\,n \end{array}$$

pode ser calculada a partir do diagrama que se segue



onde $in = [\underline{0}\,, succ]$, $g = [\underline{1}\,, mul \cdot (succ \times id)]$ e $succ\, n \stackrel{\text{def}}{=} n+1$. Repare que o diagrama é um tudo nada mais elaborado do que o habitual catamorfismo sobre $\mathbb{N}$. De facto, é um caso particular de *paramorfismo*. No caso geral, dado um tipo indutivo $\mathsf{T} \xleftarrow{in} \mathsf{F}\,\mathsf{T}$, o paramorfismo de g relativamente ao functor F, designado por $(\!|g|\!)$ é tal que



que traduz a seguinte propriedade universal:

$$h = (\!|f|\!) \quad \Leftrightarrow \quad h \cdot in = f \cdot \mathsf{F}\,\langle id, h\rangle \tag{1}$$

1. A partir de (1) deduza a regra de **reflexão-para**:
$$id = (\!|in \cdot \mathsf{F}\pi_2|\!)$$
2. Sabendo que uma das formas de calcular n^2, o quadrado de um número natural n, é somar os n primeiros ímpares — *cf.* $1^2 = 1$, $2^2 = 1+3$, $3^2 = 1+3+5$, *etc.*, $n^2 = (2n-1)+(n-1)^2$ — exprima a função $\mathsf{sq}\, n \stackrel{\text{def}}{=} n^2$ sob a forma de um paramorfismo em $\mathbb{N}$.

Questão 5 Considere o problema de se construir, a partir de uma lista arbitrária de tipo `a`, uma lista com o mesmo comprimento de tipo `[(a,Bool,Int)]` em que o booleano corresponde à verificação (ou não) de um predicado, e o inteiro corresponde a uma contagem dos elementos que verificaram (ou não) o predicado, *do fim para o início da lista*. Por exemplo:

```
> ocorrencias (<5) [1,3,5,7,2,5,8,9,4]
[(1,True,4),(3,True,3),(5,False,5),(7,False,4),(2,True,2),(5,False,3),
(8,False,2),(9,False,1),(4,True,1)]
```

O problema pode ser resolvido recorrendo a um par auxiliar `(Int,Int)` em que se vai contando o número de elementos que verificaram ou não o predicado:

```
ocorrencias :: (a->Bool) -> [a] -> [(a,Bool,Int)]
ocorrencias p l = let (s,l') = ocorr p l (0,0) in l'

ocorr :: (a->Bool) -> [a] -> (Int,Int) -> ((Int,Int) , [(a,Bool,Int)])
ocorr p [] s = (s,[])
ocorr p (h:t) s = let ((st,sf), l) = ocorr p t s
                  in if (p h) then ((st+1,sf) , (h,True,st+1):l)
                              else ((st,sf+1) , (h,False,sf+1):l)
```

Considere uma variante deste problema em que a contabilização dos elementos que verificam / não verificam o predicado é feita do início para o fim da lista. Implemente-a em HASKELL recorrendo a uma mónade de estado:

```
data Trans a = Trans ((Int,Int) -> ((Int,Int) , a))
instance Monad Trans where ... -- [assuma definicao habitual]
```
