Métodos de Programação I

2.º Ano da LMCC (7003N5) + LESI (5303O7)
Ano Lectivo de 2001/2002

Exame (época normal, 2.ª chamada) — 5 de Fevereiro 2002
14h30
Salas 2201 a 2207

NB: *Esta prova consta de* ***8*** *alíneas que valem, cada uma, 2.5 valores. Responda no enunciado, preenchendo a sua identificação em todas as folhas.*

PROVA SEM CONSULTA (2 horas)

Questão 1 A seguinte função, extraída de `Prelude.hs`,

```
lookup           :: Eq a => a -> [(a,b)] -> Maybe b
lookup k []       = Nothing
lookup k ((x,y):xys)
      | k==x      = Just y
      | otherwise = lookup k xys
```

procura um dado elemento k numa lista de pares $\langle x, y\rangle$ e, se o encontrar, devolve o y correspondente.

1. Escreva $lookup\, k$ sob a forma de um catamorfismo de listas, isto é, calcule o gene g em

$$lookup\, k = (\!|g|\!) \tag{1}$$

Qual o seu comportamento para o caso da chave k estar repetida na lista argumento?

2. Defina a função equivalente a $lookup\, k$ para árvores binárias (em `BTree.hs`). Qual o seu comportamento para o caso da chave k vir repetida na árvore argumento?

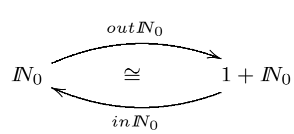
Questão 2 Na biblioteca `RList.hs` a função factorial é apresentada como o hilomorfismo

```
fac = hyloRList (either (const 1) mul) obsNat
      where mul(x,y) = x*y
            obsNat 0 = Left ()
            obsNat (n+1) = Right (n+1,n)
```

1. Mostre que

$$obsNat \quad = \quad (id + \langle succ, id\rangle) \cdot out_{I\!N_0} \tag{2}$$

se verifica, onde $out_{I\!N_0}$ é o isomorfismo inverso de $in_{I\!N_0} = [\underline{0}, succ]$ em



Sugestão: componha ambos os membros da igualdade com $in_{I\!N_0}$.

2. Será a seguinte definição alternativa

```
fac' = hyloRList (either (const 1) g) obsNat'
   where g(n,m) = (n+1) * m
         obsNat' 0 = Left ()
         obsNat' (n+1) = Right (n,n)
```

equivalente a `fac`? Prove que o é, de facto, seguindo e justificando o raciocínio que se segue:

$$
\begin{array}{ll}
 & fac' = [\![\, [\underline{1}, mul \cdot (succ \times id)], (id + \langle id, id\rangle) \cdot out_{I\!N_0} \,]\!] \\
\equiv & \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\
 & fac' = [\underline{1}, mul \cdot (succ \times id)] \cdot (id + id \times fac') \cdot (id + \langle id, id\rangle) \cdot out_{I\!N_0} \\
\equiv & \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\
 & fac' = [\underline{1}, mul] \cdot (id + succ \times id) \cdot (id + id \times fac') \cdot (id + \langle id, id\rangle) \cdot out_{I\!N_0}
\end{array}
$$

$\equiv \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \}$

$fac' = [\underline{1}, mul] \cdot (id + id \times fac') \cdot (id + succ \times id) \cdot (id + \langle id, id \rangle) \cdot out_{I\!N_0}$

$\equiv \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \}$

$fac' = [\underline{1}, mul] \cdot (id + id \times fac') \cdot (id + \langle succ, id \rangle) \cdot out_{I\!N_0}$

$\equiv \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \}$

$fac' = [\underline{1}, mul] \cdot (id + id \times fac') \cdot obsNat$

$\equiv \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \}$

$fac' = [\![\, [\underline{1}, mul], obsNat \,]\!]$

$\equiv \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \}$

$fac' = fac$

Questão 3 Recorde a biblioteca `LTree.hs`, a que se acrescenta a função

```
f = cataLTree (inLTree . (id -|- swap))
```

Que "faz" esta função? Converta-a para notação HASKELL **com** variáveis e indique o seu resultado se a aplicar, no contexto do algoritmo `mSort` ('merge sort'), ao resultado de `anaLTree lsplit [1,2,3,4]`.

Questão 4 Considere a seguinte definição duma função que calcula a média de todos os inteiros que são folhas de uma árvore binária:

```
media :: LTree Integer -> Integer
media = (uncurry div) . calc
```

Defina `calc` como um catamorfismo.

Questão 5 Na interpretação de comandos é vulgar disponibilizar-se um modo de execução dito "verbose" sempre que uma função ou comando, ao executar, "explica" o que está a fazer. A opção "`-v`" é muitas vezes usada para esse efeito.

A execução "verbose" é também muito utilizada em "debug", permitindo ao programador acompanhar o trajecto de uma execução até à ocorrência de um erro, por exemplo. Em programação imperativa (eg. C), a adição de instruções de saída tipo `printf` é uma forma primitiva de introduzir verbosidade. Em HASKELL, esse processo pode ser sistematizado através do recurso a uma mónada particular que prevê "logs" (textos explicativos) anexos ao resultado de funções:

```
data Verbose a = Verb (a, Text)

type Text = [ String ]

vap :: Text -> (a -> b) -> a -> Verbose b        -- verbose apply
vap t f a = Verb(f a,t)
```

1. Com base no seguinte exemplo de composição (monádica) entre duas funções "verbose",

   ```
   Verbose> ((vap ["First succ: ok!"] succ) .!
             (vap ["Second succ: ok!"] succ)) 3
   Verb (5,["Second succ: ok!","First succ: ok!"])
   ```

   complete a seguinte definição da mónada subjacente:

   ```
   instance Monad Verbose where

       return ..........................................................

       .......... >>= ..................................................

       .................................................................

       .................................................................
   ```

2. Defina `Verbose` como instância da classe `Functor`.
