Métodos de Programação I

2.º Ano da LMCC (7003N5) + LESI (5303O7)
Ano Lectivo de 2002/2003

Exame (época normal, 1.ª chamada) — 16 de Janeiro de 2003
09h30
Salas 2201 a 2208

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 Partindo da definição do "combinador condicional" de McCarthy

$$p \to g, h \quad \stackrel{\text{def}}{=} \quad [g, h] \cdot p? \tag{1}$$

e da propriedade

$$p? \cdot f \quad = \quad (f + f) \cdot (p \cdot f)? \tag{2}$$

prove a validade de

$$(p \to f, g) \cdot h \quad = \quad (p \cdot h) \to (f \cdot h), (g \cdot h) \tag{3}$$

Questão 2 Considere a seguinte declaração de tipo em Haskell

```
data DT a = Vazia | Nodo (a, Bool -> DT a)
```

1. Demonstre que o tipo `DT a` é isomorfo ao tipo `BTree a`.
2. Defina em Haskell as funções que testemunham o isomorfismo `DT a` $\cong$ `BTree a`.
3. Apresente as funções que definiu na alinea anterior como um catamorfismo e um anamorfismo do tipo `BTree`.
4. Considere a função

```
resp :: DT a -> [Bool] -> Maybe a
resp Vazia _ = Nothing
resp (Nodo (x,_)) [] = Just x
resp (Nodo (x,f)) (b:bs) = case (f b) of
                             Vazia -> Nothing
                             _     -> resp (f b) bs
```

   (a) Explique por palavras suas porque é que, para um dado `t` do tipo `DT a`, a função (`resp t`) não pode ser expressa como um catamorfismo sobre listas.

   (b) Defina a função `resposta = uncurry resp` como um hilomorfismo.

Questão 3 A seguir apresenta-se uma versão do famoso algoritmo de Euclides para calcular o máximo divisor comum entre dois números inteiros positivos:

```
mdc (x,y) = if  (x == y) then  x
            else mdc (x',y')
    where x' = (max x y) - y'
          y' = min x y
```

1. Exprima esta função como um hilomorfismo, acompanhando a resposta de um diagrama elucidativo.
2. Defina em Haskell e em **C** o tipo de dados intermédio que obteve na alínea anterior e apresente o elemento desse tipo que é calculado no cálculo de `mdc (12,18)`

Questão 4 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 com o auxílio de 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)
```

Uma solução alternativa para este problema recorre a uma mónade de estado:

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

Complete a definição da versão monádica das funções `ocorr` e `ocorrencias`:

```
mocorr :: (a->Bool) -> [a] -> Trans [(a,Bool,Int)]
mocorr p [] = Trans (\(nt,nf) -> ((nt,nf) , []))
mocorr p (h:t) = do l <- ..........
                    n <- Trans f
                    return ((h, p h, n):l)
                 where f = if (p h) then \(nt,nf) -> ((nt+1,nf),nt+1)
                                    else ..........

mocorrencias :: (a->Bool) -> [a] -> [(a,Bool,Int)]
mocorrencias p l = let (Trans f) = mocorr p l
                   in ..........
```
