Métodos de Programação I

2.º Ano da LESI (5303O7) + LMCC (7003N5)
Ano Lectivo de 2005/06

Exame (época de recurso) — 13 de Fevereiro 2006
09h30
Salas 2202 a 2205

NB*: Esta prova consta de **8** alíneas que valem, cada uma, 2.5 valores. Utilize folhas de resposta diferentes para cada grupo.*

PROVA SEM CONSULTA (2 horas)

GRUPO I

Questão 1 Considere o seguinte isomorfismo.

$$(A + B) \times C \cong (A \times C) + (B \times C)$$

Escreva, em Haskell e no estilo *pointwise*, a função `undistl` que testemunha o isomorfismo da direita para a esquerda. Defina a versão *point-free* dessa função, acompanhando a definição obtida com o diagrama respectivo.

Questão 2 Demonstre a seguinte igualdade:

$$[\langle id, i_1 \cdot !\rangle, (id \times i_2) \cdot swap] \quad = \quad \langle [id, \pi_2], ! + \pi_1 \rangle$$

Qual o isomorfismo que esta função estabelece? Justifique através de um diagrama ilustrativo.

Questão 3 Considere o seguinte tipo de dados indutivo:

```
data From a = First a | Succ (From a)
```

Defina `inFrom`, `outFrom`, `cataFrom` e `anaFrom`. Acompanhe as suas definições com os diagramas respectivos.

GRUPO II

Questão 4 Considere a seguinte versão linear do algoritmo de Fibonacci.

```
fib :: Int -> Int
fib n = snd (aux n)
    where aux 0     = (0,1)
          aux (n+1) = (snd (aux n), add (aux n)
          add = uncurry (+)
```

Demonstre que `aux` pode ser definida no estilo *point-free* como

```
aux = cataNat (either (split (const 0) (const 1)) (split snd add))
```

Entre outras, deverá usar a lei universal de `cataNat`. Relembre que esta função e `inNat` podem ser definidas da seguinte forma:

```
cataNat :: (Either () b -> b) -> Int -> b
cataNat g = g . (id -|- cataNat g) . outNat

inNat :: Either () Int -> Int
inNat = either (const 0) succ
```

Questão 5 Relembre o tipo de listas polimórficas que está definido num módulo da biblioteca da disciplina:

```
data RList a = Nil | Cons (a , RList a)
```

Sobre este tipo é possível definir a função `map` que aplica uma função a todos os elementos de uma lista da seguinte forma:

```
mapRList f = cataRList (inRList . (id -|- f >< id))
```

Recorrendo, entre outras, à lei de fusão para listas, demonstre a seguinte propriedade de absorção para os catamorfismos deste tipo de dados, válida para quaiquer funções `f` e `g`.

```
cataRList g . mapRList f = cataRList (g . (id -|- f >< id))
```

Questão 6 Considere a seguinte definição *tail-recursive* da função factorial.

```
fact :: Int -> Int
fact n = aux (n,1)
    where aux (0,a) = a
          aux (n,a) = aux (n-1,a*n)
```

Defina a função auxiliar `aux` como um hilomorfismo do tipo `From` da questão 3. Apresente o diagrama respectivo.

GRUPO III

Questão 7 Relembre o exercício da primeira chamada, em que se pretendia imprimir o conteúdo de uma árvore binária de procura prefixando cada elemento com o número de ordem em que este ocorre na travassia *in-order*. Assim,

```
1:3                                              8
2:5                                             / \
3:8     (será o resultado para a árvore)       5   12
4:10                                           /    /
5:12                                          3    10
```

Desta vez pretende-se implementar essa função usando `sequence_ :: Monad m => [m a] -> m ()`, a função da biblioteca do Haskell que, dada uma lista de computações monádicas, executa-as sequencialmente ignorando o seu resultado. Implemente esta função e depois complete o seguinte código por forma a resolver o problema proposto.

```
data BTree a = Vazia | Nodo a (BTree a) (BTree a)
mostra :: (Show a) => BTree a -> IO ()
mostra t = sequence_ ...
```

Questão 8 Dê definições Haskell para as funções f_1 a f_9 da seguinte tabela, onde se apresentam três tipos paramétricos de dados que são instâncias da classe `Monad`:

`F` a	u `(return)`	`(>>=)`	μ `(join)`
`Maybe a`	f_1	f_2	f_3
`[a]`	f_4	f_5	f_6
`Error a`	f_7	f_8	f_9

Assuma que que o tipo `Error` se encontra definido da seguinte forma:

```
data Error a = Err String | Ok a
```
