Métodos de Programação I

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

Exame (época normal, 1.ª chamada) — 17 de Janeiro de 2002
09h30
Salas 2201 a 2210

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 O facto

$$(id \times \pi_2) \cdot assocr = \pi_1 \times id$$

verifica-se. Complete as reticências no respectivo processo de cálculo que se segue:

$(id \times \pi_2) \cdot assocr = \pi_1 \times id$

$=$ { definição de $assocr$ }

....

$=$ { absorção-$\times$ }

....

$=$ { natural-id, definição de $\times$ e cancelamento-$\times$ }

....

$=$ { absorção-$\times$ }

....

$=$ { reflexão-$\times$ }

....

$=$ { natural-id }

$\pi_1 \times id = \pi_1 \times id$

$=$ { .... }

V

Questão 2 Partindo da definição do "combinador condicional" de McCarthy e da propriedade

$$p? \cdot f \quad = \quad (f + f) \cdot (p \cdot f)? \tag{1}$$

prove a validade de

$$(p \rightarrow f, g) \cdot h \quad = \quad (p \cdot h) \rightarrow (f \cdot h), (g \cdot h) \tag{2}$$

Questão 3 Considere a seguinte definição de uma função t, em HASKELL:

```
t f g h k =
       [ either (split f g)(split h k),
         split (either f h)(either g k) ]
```

Qual é o tipo de t? Justifique convenientemente a sua resposta.

Questão 4 Considere, em HASKELL, a seguinte definição recursiva da função factorial,

```
fac :: Integral a => a -> a
fac 0 = 1
fac (n+1) = (n+1) * fac n
```

1. Mostre que essa definição pode ser convertida na seguinte definição "pointfree":

$$fac \cdot [\underline{0}, succ] = [\underline{1}, mul \cdot (succ \times id)] \cdot (id + \langle id, fac \rangle) \tag{3}$$

onde mul designa a versão "uncurried" de $*$ na class `Num`.

2. Assumindo as definições

$$\begin{array}{rcl} in & = & [\underline{0}, succ] \\ out & = & in^{-1} \\ g & = & [\underline{1}, mul \cdot (succ \times id)] \end{array}$$

o functor "números naturais"

$$\left\{ \begin{array}{l} \mathsf{F}\, X = 1 + X \\ \mathsf{F}\, f = id + f \end{array} \right.$$

e o functor "listas de naturais"

$$\left\{ \begin{array}{l} \mathsf{G}\, X = 1 + I\!N \times X = \mathsf{F}(I\!N \times X) \\ \mathsf{G}\, f = id + id \times f = \mathsf{F}(id \times f) \end{array} \right.$$

complete as reticências no seguinte processo de transformação de (3) num hilomorfismo de listas:

$$\begin{array}{cl}
& fac \cdot in = g \cdot \mathsf{F}\,\langle id, fac \rangle \\
= & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\
& fac = g \cdot \mathsf{F}\,\langle id, fac \rangle \cdot out \\
= & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\
& fac = g \cdot \mathsf{F}\,((id \times fac) \cdot \langle id, id \rangle) \cdot out \\
= & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\
& fac = g \cdot \mathsf{F}\,(id \times fac) \cdot \mathsf{F}\,\langle id, id \rangle \cdot out \\
= & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\
& fac = g \cdot (\mathsf{G}\, fac) \cdot \mathsf{F}\,\langle id, id \rangle \cdot out \\
= & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\
& fac = [\![\, g, \mathsf{F}\,\langle id, id \rangle \cdot out \,]\!] \\
= & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\
& fac = (\!|g|\!) \cdot [\!(\mathsf{F}\,\langle id, id \rangle \cdot out)\!]
\end{array}$$

Questão 5 Pretende-se uma função que some os elementos de uma lista com eficiência semelhante à do algoritmo "quick sort", *i.é* do hilomorfismo

```
qSort = hyloBTree inord qsep
--    = (cataBTree inord) . (anaBTree qsep)
```

da biblioteca `BTree.hs`.

Qual a componente do hilomorfismo a modificar por forma a converter `qSort` na função pretendida? Justifique a sua resposta explicitando essa modificação.

Questão 6 Na programação funcional é vulgar a ocorrência de funções parciais, *i.é*, funções indefinidas para algum dos seus argumentos. Por exemplo, a divisão é parcial pois $n/0$ é um valor indefinido, ou *excepção*. A excepções são vulgarmente assinaladas através de mensagens de erro, estendendo-se o codomínio da função por forma a fornecer 'strings' explicativos. Em HASKELL, por exemplo,

```
(/) :: Double -> Double -> Double
```

pode ser estendida a

```
dv :: (Double,Double) -> Error Double
dv(n,0) = Err "Nem pense em dividir por 0!"
dv(n,m) = Ok (n / m)
```

onde

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

Defina `Error` como instância da classes `Functor` e `Monad`, isto é, preencha as reticências em

```
instance Functor Error where

    fmap f ........................................................

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

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

instance Monad Error where

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

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

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

Questão 7 Considere a seguinte definição de uma função em HASKELL:

```
parte :: [Int] -> ([Int],[Int])
parte [] = ([],[])
parte (x:xs)
      | odd x = let (l,r) = parte xs
                in (x:l,r)
      | otherwise = ([],x:xs)
```

Adoptando a metodologia sugerida nas aulas teórico-práticas, defina a função `parte` como um hilomorfismo, tornando explícito o tipo da estrutura de dados virtual.
