Métodos de Programação I

2.º Ano da LESI (5303O7) + LMCC (7003N5, Plano antigo)
Ano Lectivo de 2006/07

Exame (época de recurso) — 12 de Fevereiro 2007
14h00
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 Dada a função

$$iso = \langle ! + !, [id\ , id] \rangle \tag{1}$$

identifique o isomorfismo que esta função estabelece, desenhando o diagrama respectivo. Derive — aplicando a lei da troca — a versão *pointwise* de iso.

Questão 2 Partindo da propriedade natural do converso de iso,

$$(f + f) \cdot iso^{\circ} = iso^{\circ} \cdot (id \times f) \tag{2}$$

e da definição

$$p? = iso^{\circ} \cdot \langle p, id \rangle \tag{3}$$

demonstre a lei de fusão de predicado guardado que conhece.

Questão 3 Considere o seguinte tipo de dados.

```
data Tree a = Node a [Tree a]
```

Defina as funções `inTree`, `outTree`, `cataTree` e `anaTree` para este tipo de dados. Acompanhe as duas últimas definições com os diagramas respectivos.

GRUPO II

Questão 4 Suponha que o par de inteiros positivos (v, p) designa o número de bolas vermelhas (v) e pretas (p) que se encontram dentro de um saco, bolas essas que se vão tirando à sorte, sucessivamente, até o saco ficar vazio.

A função em Haskell que se segue

```
gera :: (Int,Int) -> Tree (Int,Int)
gera(0,0) = Node (0,0) []
gera(v,0) = Node (v,0) [gera(v-1,0)]
gera(0,p) = Node (0,p) [gera(0,p-1)]
gera(v,p) = Node (v,p) [gera(v-1,p),gera(v,p-1)]
```

representa sob a forma de uma árvore do tipo `Tree` da questão 3 todas as possíveis configurações do saco ao longo dessas experiências.

Codifique a função `gera` como um anamorfismo, desenhando o diagrama respectivo.

Questão 5 Como sabe, o operador monádico de *binding* é dado, para qualquer mónada F, pela definição

$$x >\!\!>\!= f \stackrel{\mathrm{def}}{=} (\mu \cdot \mathsf{F}\, f)x \qquad (4)$$

que, no caso da mónada das listas, instancia em

$$x >\!\!>\!= f \stackrel{\mathrm{def}}{=} (concat \cdot map\ f)x \qquad (5)$$

que, em notação pré-fixa, se pode escrever

$$(>\!\!>\!= f)x \stackrel{\mathrm{def}}{=} (concat \cdot map\ f)x$$

Sabendo que $concat$ é o catamorfismo de listas $(\![[nil\ , cat]]\!)$, onde $nil = \underline{[\,]}$ e $cat(x, y) = x +\!\!+\, y$, calcule a versão *pointwise* de $>\!\!>\!=$ que aparece na bilioteca `Control.Monad.hs`:

```
instance Monad [ ] where
    (x:xs) >>= f = f x ++ (xs >>= f)
    []     >>= f = []
```

Sugestão: Use a lei de absorção

$$(\![f]\!) \cdot map\ g = (\![f \cdot (id + g \times id)]\!) \qquad (6)$$

Recorda-se ainda que, para o tipo de dados em causa, o isomorfismo in é $[nil\ , cons]$, onde $cons(x, xs) = x : xs$.

Questão 6 No contexto da questão anterior, demonstre a lei

$$concat \cdot concat = concat \cdot (map\ concat) \qquad (7)$$

sabendo que

$$concat \cdot nil = nil \qquad (8)$$

$$concat \cdot cat = cat \cdot (concat \times concat) \qquad (9)$$

GRUPO III

Questão 7 Considere a seguinte declaração do tipo *LTree* como instância da class `Monad`:

```
instance Monad LTree where
   return = Leaf
   t >>= g = (join . fmap g) t
```

(a) Defina a função `join`; (b) Defina a função `decora :: a -> LTree b -> LTree (a, b)` que etiqueta cada folha da árvore com o valor dado como primeiro argumento usando notação-do ou *bind*, isto é, sem usar recursividade explícita.

Questão 8 Complete a seguinte demonstração de lei da associatividade da composição monádica:

$$f \bullet (g \bullet h) = (f \bullet g) \bullet h$$

$\equiv$ { .... }

$$f \bullet (\mu \cdot \mathsf{F}\, g \cdot h) = (\mu \cdot \mathsf{F}\, f \cdot g) \bullet h$$

$\equiv$ { .... }

⋮
