Métodos de Programação I

2.º Ano da LMCC (7003N5) + LESI (5303O7)
Ano Lectivo de 2003/04

Exame (época de recurso) — 21 de Fevereiro 2004
09h30
Salas 2201 a 2204

NB*: Esta prova consta de **8** alíneas que valem, cada uma, 2.5 valores.*

PROVA SEM CONSULTA (2 horas)

Questão 1 Considere as definições seguintes de um tipo de árvores binárias cujos nodos são etiquetados com números inteiros, e de uma função sobre essas árvores.

$$\mathsf{data\ IntBTree} = \mathsf{Empty} \mid \mathsf{Node}\ (\mathsf{Int}, (\mathsf{IntBTree}, \mathsf{IntBTree}))$$

$$\mathsf{heap2list} = (\!|[\underline{[\,]}, \mathsf{cons} \cdot (\mathsf{id} \times \mathsf{merge})]|\!)_{\mathsf{IntBTree}}$$

em que $\mathsf{cons} = \mathsf{uncurry}\ (:)$ e $\mathsf{merge} : [\mathsf{Int}] \times [\mathsf{Int}] \to [\mathsf{Int}]$ é a função usual de fusão de listas ordenadas crescentemente.

1. Escreva uma definição *pointwise* de heap2list, justificando todos os passos para a sua obtenção.
2. Seja countNodes a função assim definida:

   ```
   countNodes Empty = 0
   countNodes (Node(x,(e,d)) = succ ((countNodes e) + (countNodes d))
   ```

   Prove, justificando todos os passos, a igualdade

   $$\mathsf{countNodes} = \mathsf{length} \cdot \mathsf{heap2list}$$

   N.B. Deverá aplicar a lei de fusão dos catamorfismos. Assuma como válida a equivalência $\mathsf{length} \cdot \mathsf{merge} = (\mathsf{uncurry}(+)) \cdot (\mathsf{length} \times \mathsf{lengt}$
3. A ideia subjacente à função heap2list é que quando aplicada a determinadas árvores, ela permite obter listas ordenadas de forma crescente. As árvores em questão (usualmente designadas por *heaps*) caracterizam-se pelo facto de o conteúdo de cada nó ser inferior ou igual aos conteúdos de todos os nós seus descendentes.

   Tendo isto em conta, é possível escrever a função `heap2list` como um anamorfismo de listas. Escreva essa definição.

   Sugestão: comece por definir uma função que permita combinar duas *heaps*.

Questão 2 A seguinte função em Haskell

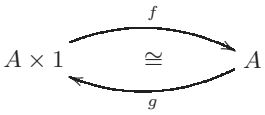
```
sqrt' p x = loop p x 1
            where loop p x r = let r'= (r + x / r) / 2
                               in if abs(r - r') < p
                                  then r' else loop p x r'
```

calcula a raíz quadrada de um número x com erro p. Por exemplo,

```
> sqrt' 0.01 2
1.41422 :: Double
>
```

1. Represente $loop\ p\ x$ como um hilomorfismo e faça um diagrama explicativo.
2. Defina em Haskell o tipo de dados indutivo intermédio deste hilomorfismo (aquele que é saída do anamorfismo e entrada do catamorfismo) e represente o valor dessa estrutura para a situação em que $loop$ é invocado de $sqrt'\ 1\ 2$.

Questão 3 Considere o seguinte diagrama:



1. Identifique ou defina as funções f e g que testemunham o isomorfismo.
2. Sintetize, justificando, o isomorfismo

$$A \times (1 + X)^2 \xrightarrow{v} A + A \times X + A \times X + A \times X^2$$

(**Sugestão**: Recorde os isomorfismos seguintes:

- $A^2 \cong A \times A$
- $2 \times A \cong A + A$
- $A \times (B \times C) \cong (A \times B) \times C$
- $A \times (B + C) \cong (A \times B) + (A \times C)$

)

Questão 4 A definição em Haskell que se segue

```
mfold k f [] = k
mfold k f (h:t) = do { b <- mfold k f t ; f h b }
```

estende o combinador `foldr` no contexto de uma mónade arbitrária. Qual o tipo mais geral de $mfold$? Complemente a sua resposta indicando instâncias da sua aplicação a habitantes dos tipos monádicos $[a]$ e $Maybe\ a$.
