Cálculo de Programas
Exame Especial

22 de Setembro de 2008, 9h30
Sala CP2 305

O exame tem a duração de 2h00. Todas as questões valem 3 pontos, com excepção da última que vale apenas 2.

Questão 1 Considere a seguinte função.

$$\mathsf{xyz} = (\mathsf{id} \triangledown \mathsf{id}) \vartriangle (\mathsf{id} \triangledown \mathsf{id})$$

Identifique o seu tipo desenhando o diagrama respectivo. Defina xyz em Haskell no estilo *point-wise*. Será que esta função testemunha um isomorfismo? Justifique a sua resposta.

Questão 2 Considere o seguinte tipo de dados para representar árvores binárias sem conteúdo.

```
data Tree = Tip | Fork Tree Tree
```

Atendendo ao isomorfismo $\mathsf{Tree} \cong 1 + \mathsf{Tree} \times \mathsf{Tree}$, defina as funções out em Haskell no estilo *point-wise* e in no estilo *point-free*. Desenhe o diagrama dos catamorfismos para este tipo, e identifique a respectiva lei universal. Codifique o catamorfismo em Haskell no estilo *point-wise*.

Questão 3 É trivial definir uma função recursiva que conta o número de folhas destas árvores.

```
nfolhas :: Tree → Int
nfolhas Tip        = 1
nfolhas (Fork l r) = nfolhas l + nfolhas r
```

Demonstre que nfolhas pode ser definida em *point-free* usando o catamorfismo

$$\mathsf{nfolhas} = (\!|\underline{1} \triangledown \mathsf{plus}|\!)$$

onde plus = uncurry (+). Justifique todos os cálculos que efectuar e desenhe o diagrama respectivo.

Questão 4 Considere a função de *Fibonacci*.

```
fib :: Int → Int
fib 0 = 1
fib 1 = 1
fib n = fib (n − 1) + fib (n − 2)
```

É bem conhecido que esta função pode ser escrita como a composição da função nfolhas após um *unfold* que gera uma árvore com a estrutura das chamadas recursivas originais.

$$\mathsf{fib} = \mathsf{nfolhas} \circ \mathsf{gerat}$$

Defina o padrão de recursividade *unfold* para o tipo Tree e implemente gerat usando um *unfold*. Desenhe também o diagrama respectivo.

Questão 5 Demonstre que nfolhas ∘ mirror = nfolhas, onde mirror é a função que espelha uma árvore e que pode ser implementada com o seguinte catamorfismo.

$$\mathsf{mirror} = (\!|\mathsf{in} \circ (\mathsf{id} + \mathsf{swap})|\!)$$

Justifique todos os cálculos que efectuar. Para efectuar esta prova necessita da propriedade comutativa da adição. Demonstre que essa propriedade pode ser definida em *point-free* como plus ∘ swap = plus.

Questão 6 A parte curricular do mestrado de Informática consiste em duas Unidades Curriculares de Especialização (UCEs). Quando um aluno se candidata ao mestrado apresenta uma lista com as UCEs que pretende frequentar.

```
data UCE = MFES | CSSI | ...
type Numero = Int
type Aluno = (Numero, [UCE])
```

As candidaturas são seriadas de acordo com as médias dos alunos, sendo produzida uma lista onde os alunos com melhor média aparecem à cabeça.

```
type Candidaturas = [Aluno]
```

Depois de seriados os alunos tem que ser alocados por ordem por forma a obter uma lista de colocações nas UCEs.

```
type Colocacoes = [(UCE, [Numero])]
```

Assumindo que se encontra definida, usando o *monad* estado, a função que coloca um aluno

```
colocaUm :: Aluno → State Colocacoes ()
```

defina a função que coloca todos os alunos respeitando a seriação.

```
colocaTodos :: Candidaturas → Colocacoes
```

Questão 7 Suponha que se pretende implementar um *monad* que contém sempre um e um só resultado, mas que também contabiliza o número de *binds* que são feitos. É possível implementar este *monad* com o seguinte tipo de dados.

```
data Conta a = Conta a Int
```

Defina a instância da classe Monad por forma a obter o comportamento descrito.