Cálculo de Programas
Exame

12 Julho de 2008, 9h30

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

Questão 1 Identifique o isomorfismo que a seguinte função testemunha, desenhando o diagrama respectivo.

$$\mathsf{iso} = (\mathsf{inl} \times \mathsf{id}) \triangledown (\mathsf{inr} \times \mathsf{id})$$

Defina esta função em Haskell no estilo *point-wise*. Demonstre a seguinte propriedade da função iso, justificando todos os cálculos que efectuar.

$$\mathsf{fst} \circ \mathsf{iso} = \mathsf{fst} + \mathsf{fst}$$

Questão 2 Considere o seguinte tipo de dados para representar listas onde não só é possível inserir elementos à cabeça (Cons), mas também na cauda da lista (Snoc).

```
data List a = Nil | Cons a (List a) | Snoc a (List a)
```

Atendendo ao isomorfismo $\mathsf{List}\ a \cong 1 + (a \times \mathsf{List}\ a + a \times \mathsf{List}\ a)$, 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 Uma das vantagens destas listas é que é possível definir a seguinte função de inversão, que executa em tempo linear.

```
rev :: List a → List a
rev Nil = Nil
rev (Cons x l) = Snoc x (rev l)
rev (Snoc x l) = Cons x (rev l)
```

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

$$\mathsf{rev} = (\!|\mathsf{in} \circ (\mathsf{id} + \mathsf{coswap})|\!)$$

onde $\mathsf{coswap} = \mathsf{inr} \triangledown \mathsf{inl}$. Justifique todos os cálculos que efectuar.

Questão 4 Pretende-se implementar a função $\mathsf{divide} :: (a \to \mathsf{Bool}) \to [a] \to \mathsf{List}\ a$ que, dado um predicado p e uma lista normal do Haskell l, cria uma lista do tipo List a onde todos os elementos de l que satisfazem p aparecem antes dos elementos que não satisfazem p. Não é necessário manter a ordem original dos elementos. Defina o padrão de recursividade *unfold* para o tipo List a e implemente divide p usando um *unfold*.

Questão 5 Demonstre que rev∘rev = id. Relembre que a função coswap satisfaz as seguintes propriedades:

$$\mathsf{coswap} \circ \mathsf{coswap} = \mathsf{id}$$
$$(f + g) \circ \mathsf{coswap} = \mathsf{coswap} \circ (g + f)$$

Justifique todos os cálculos que efectuar.

Questão 6 Considere o seguinte tipo para árvores generalizadas.

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

Usando o *monad* estado implemente a função decora :: Tree $a \to$ Tree (a, Int) que decora cada nó da árvore com a posição em que esse nó aparece numa travessia *preorder*.

Questão 7 Considere o seguinte tipo Haskell, que permite armazenar um resultado do tipo a juntamente com uma lista de mensagens que, de alguma forma, descrevem a forma como o resultado foi obtido.

```
data Log a = Log a [String] deriving Show
```

É possível definir uma instância da classe Monad para este tipo, onde o operador *bind* acumula todas as mensagens obtidas até um dado momento. Considerando tal instância e assumindo que existe uma função log :: String $\to$ Log () que cria uma nova mensagem, seria possível, por exemplo, escrever a seguinte função de avaliação de expressões.

```
data Exp = Const Int | Sum Exp Exp
instance Show Exp where ...
eval :: Exp → Log Int
eval (Const x)     = return x
eval a@(Sum l r) = do x ← eval l
                      y ← eval r
                      log (show a ++ " = " ++ show (x + y))
                      return (x + y)
```

O resultado de avaliar a expressão Sum (Sum (Const 3) (Const 4)) (Const 2) seria

```
Log 9 ["(3+4) = 7","((3+4)+2) = 9"]
```

Implemente a instância da classe Monad e a função log por forma a obter o comportamento descrito.