Métodos de Programação I

2.º Ano da Lmcc (701055) + Lesi (531316)
Ano Lectivo de 2000/2001

Exame (época normal. 2.ª chamada) − 30/1/2001
14:30
Salas 2201, 2202, 2209, 2210

Grupo I

1. Calcule (caso exista) o tipo da função $\langle i_1, \pi_1 \rangle$
2. Determine, usando as funções swap :: $X \times Y \to Y \times X$ e distr :: $X \times (Y + Z) \to X \times Y + X \times Z$, um isomorfismo entre $(A \times B + C) \times D$ e $A \times (D \times B) + D \times C$. Apresente ainda uma codificação dessa função em Haskell.
3. Considere as seguintes definições:
   - $x = (f \circ \pi_2) \circ swap$
   - $y = \pi_1 \circ (f \times (id^A \circ ap))$

   (a) Para as funções x e y acima, apresente um diagrama correspondente às suas definições.
   (b) Mostre que para qualquer função f, $x = y$.

Grupo II

Considere o tipo de dados

```
data FList a b = Unit a | Add b (FList a b)
```

1. Defina as operações `cata`/`ana`/`hilo` para esse tipo de dados. Acompanhe essas definições com os diagramas respectivos.
2. O tipo apresentado é isomorfo ao tipo primitivo do *Haskell* `(a,[b])`. Defina as funções que testemunham esse isomorfismo respectivamente como um *catamorfismo* e um *anamorfismo* do tipo `FList`.
3. Considere a função

   ```
   span :: (a->bool) -> [a] -> ([a],[a])
   span pred [] = []
   span pred (x:xs) | pred x = let (l,r) = span pred xs in (x:l,r)
                    | otherwise = ([],x:xs)
   ```

   Adoptando a metodologia sugerida nas aulas teórico-práticas deste curso, mostre como podemos definir `span` como um *hilomorfismo* do tipo `FList`.

Grupo III

Considere as seguintes definições

```
data LTree a = Leaf a | Fork (LTree a) (LTree a)

unfold :: (a -> Maybe (b,a)) -> a ->  [b]
unfold f x = case f x of
             Nothing ->  []
             Just (h,y)  -> h : (unfold f y)
```

1. Defina a função `unfold` como um anamorfismo de listas, i.e., complete a seguinte definição

   ```
   unfold f = anaLList (g f)
        where g  ... = ...
   ```

2. Defina em Haskell um isomorfismo entre os tipos `Maybe a` e `Either () a`.
3. Por *ana*logia com a função da primeira al}ínea, defina `unfoldLTree`

   ```
   unfoldLTree :: ...
   unfoldLTree f = anaLTree (h f)
        where h ... = ...
   ```