

Cálculo de Programas

2.º ano

Lic. Ciências da Computação e Mestrado Integrado em Engenharia Informática
UNIVERSIDADE DO MINHO

2016/17 - Ficha nr.º 3

1. Considere as funções seguintes:

$$\begin{aligned} f &= \langle \pi_1 \cdot \pi_1, \pi_2 \times id \rangle \\ g &= \langle id \times \pi_1, \pi_2 \cdot \pi_2 \rangle \end{aligned}$$

- (a) Identifique os tipos de f e g . Justifique a sua resposta construindo os respectivos diagramas.
(b) Use as leis to cálculo de programas para mostrar que $f \cdot g = id$ se verifica.

2. Considere uma função d da qual apenas conhece duas propriedades:

$$\begin{cases} \pi_1 \cdot d = id \\ \pi_2 \cdot d = id \end{cases} \quad (\text{F1})$$

Mostre (recorrendo directamente à propriedade universal correspondente) que essa função é, necessariamente, a mesma que em Programação Funcional escreveria da forma seguinte, em Haskell:

$$\begin{aligned} d &:: a \rightarrow (a, a) \\ d \ a &= (a, a) \end{aligned}$$

(Esta função, que duplica um valor, designa-se habitualmente por função *diagonal*.)

3. Identifique os tipos das expressões $\langle \pi_1, \langle id, \pi_2 \rangle \rangle$ e $\langle id, \langle \pi_1, \pi_2 \rangle \rangle$. Como compara este último com o tipo da função d da alínea anterior?
4. Considere a função

$$\alpha = [\langle \text{False}, id \rangle, \langle \text{True}, id \rangle]$$

Determine o tipo de α e mostre, usando a propriedade *Universal+*, que α se pode escrever em Haskell da forma seguinte:

$$\begin{aligned} \alpha \ (\text{Left } a) &= (\text{False}, a) \\ \alpha \ (\text{Right } a) &= (\text{True}, a) \end{aligned}$$

5. Recorra à lei Eq+ (entre várias outras) para mostrar que a definição que conhece da função factorial,

$$\begin{aligned} fac \ 0 &= 1 \\ fac \ (n + 1) &= (n + 1) * fac \ n \end{aligned}$$

é equivalente à equação seguinte

$$fac \cdot [0, succ] = [1, mul \cdot \langle succ, fac \rangle].$$

onde $succ \ n = n + 1$ e $mul \ (a, b) = a * b$.

6. Considere a seguinte declaração de um tipo de *árvores binárias*, em Haskell:

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

Indagando os tipos dos construtores Leaf e Fork, por exemplo no GHCi,

```
*LTree> :t Fork
Fork :: (LTree a, LTree a) -> LTree a
*LTree> :t Leaf
Leaf :: a -> LTree a
```

é fácil desenhar o diagrama que explica a construção da função

$$inLTree = [Leaf, Fork]$$

Desenhe-o e calcule a sua inversa

```
outLTree :: LTree a -> Either a (LTree a, LTree a)
outLTree (Leaf a) = i1 a
outLTree (Fork (x, y)) = i2 (x, y)
```

resolvendo a equação

$$outLTree \cdot inLTree = id$$

em ordem a *outLTree*.

7. No Cálculo de Programas, as definições condicionais do tipo $h\ x = \text{if } p\ x \text{ then } f\ x \text{ else } g\ x$ ou

```
h x
| p x = f x
| otherwise = g x
```

são escritas usando o combinador ternário

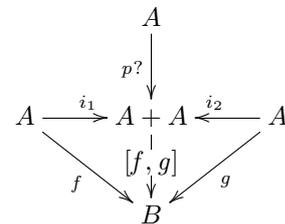
$$p \rightarrow f, g$$

conhecido pelo nome de *condicional de McCarthy*, cuja definição é (ver formulário e diagrama em baixo)

$$p \rightarrow f, g = [f, g] \cdot p?$$

onde

$$p? x = \begin{cases} i_1\ x & \text{if } p\ x \\ i_2\ x & \text{if } \neg p\ x \end{cases}$$



Demonstre a seguinte lei de fusão deste combinador condicional (identifique-a no formulário):

$$(p \rightarrow f, g) \cdot h = (p \cdot h) \rightarrow (f \cdot h), (g \cdot h)$$