Cálculo de Programas
Resolução - Ficha 10

Eduardo Freitas Fernandes

2025

Exercício 1

$$
\begin{tikzcd}
A^* \arrow[rr, bend left, "out"] & & 1 + A \times A^* \arrow[ll, bend left, "in"] \\
A^* \arrow[u, "[\![g]\!]"] \arrow[rr, "g"'] & & 1 + A \times A^* \arrow[u, "id + id \times [\![g]\!]"']
\end{tikzcd}
$$

O anaformismo de g inverte uma lista, ou seja, é a função `reverse`.

Exercício 2

$$
\begin{tikzcd}
\mathbb{N}_0 \arrow[rr, bend left, "out"] & & 1 + \mathbb{N}_0 \arrow[ll, bend left, "in"] \\
\mathbb{N}_0 \times \mathbb{N}_0 \arrow[u, "\hat{\theta}"] \arrow[rr, "g"'] & & 1 + \mathbb{N}_0 \times \mathbb{N}_0 \arrow[u, "id + \hat{\theta}"']
\end{tikzcd}
$$

$$g = (\hat{\leq}) \rightarrow i_1 \cdot (!), i_2 \cdot (id \times succ)$$

Exercício 3

$$
\begin{aligned}
& \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\equiv\ & \{\text{Def. concat, Def. sum}\} \\
& \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\equiv\ & \{\text{Absorção-cata}\} \\
& \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\Longleftarrow\ & \{\text{Absorção-+, Fusão-cata}\} \\
& \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\equiv\ & \{\text{Fusão-+, Absorção-+, Eq-+}\} \\
& \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\equiv\ & \{\text{Functor-}\times\} \\
& \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\equiv\ & \{?????\} \\
& true
\end{aligned}
$$

Exercício 4

$$\begin{aligned}
& length = sum \cdot (map\ \underline{1}) \\
\equiv & \{\text{Def. sum}\} \\
& length = (\!|\, [\underline{0}, add] \,|\!) \cdot map\ \underline{1} \\
\equiv & \{\text{Absorção-cata}\} \\
& length = (\!|\, [\underline{0}, add] \cdot B\ (\underline{1}, id) \,|\!) \\
\equiv & \{\text{Functor Base de listas}\} \\
& length = (\!|\, [\underline{0}, add] \cdot (id + \underline{1} \times id) \,|\!) \\
\equiv & \{\text{Absorção-+}\} \\
& length = (\!|\, [\underline{0}, add \cdot (\underline{1} \times id)] \,|\!) \\
\equiv & \{\ add \cdot (\underline{1} \times id) = succ \cdot \pi_2\ \} \\
& length = (\!|\, [\underline{0}, succ \cdot \pi_2] \,|\!)
\end{aligned}$$

Podemos verificar que $add \cdot (\underline{1} \times id)$ é equivalente a $succ \cdot \pi_2$, pois esta recebe um par de valores, destroi o primeiro (transformando no valor 1) e mantém o segundo, de seguida soma o segundo a 1, obtendo assim o valor equivalente ao seu sucessor.

$$\begin{aligned}
& length = length \cdot (map\ f) \\
\equiv & \{\text{Def. length}\} \\
& length = (\!|\, [\underline{0}, succ \cdot \pi_2] \,|\!) \cdot (map\ f) \\
\equiv & \{\text{Absorção-cata}\} \\
& length = (\!|\, [\underline{0}, succ \cdot \pi_2] \cdot B\ (f, id) \,|\!) \\
\equiv & \{\text{Functor Base de listas}\} \\
& length = (\!|\, [\underline{0}, succ \cdot \pi_2] \cdot (id + f \times id) \,|\!) \\
\equiv & \{\text{Absorção-+, Natural-id}\} \\
& length = (\!|\, [\underline{0}, succ \cdot \pi_2 \cdot (f \times id)] \,|\!) \\
\equiv & \{\text{Natural-}\pi_2\} \\
& length = (\!|\, [\underline{0}, succ \cdot id \cdot \pi_2] \,|\!) \\
\equiv & \{\text{Natural-id}\} \\
& length = (\!|\, [\underline{0}, succ \cdot \pi_2] \,|\!)
\end{aligned}$$

Exercício 5

$$\begin{aligned}
& depth \cdot LTree\ f = depth \\
\equiv & \{\text{Def. depth}\} \\
& (\!|\, [one, succ \cdot umax] \,|\!) \cdot T_{LTree}\ f = depth \\
\equiv & \{\text{Absorção-cata}\} \\
& (\!|\, [one, succ \cdot umax] \cdot B_{LTree}\ (f, id) \,|\!) = depth \\
\equiv & \{\text{Functor Base de LTree}\} \\
& (\!|\, [one, succ \cdot umax] \cdot (f + (id \times id)) \,|\!) = depth \\
\equiv & \{\text{Functor-id-}\times\text{, Absorção-+}\} \\
& (\!|\, [one \cdot f, succ \cdot umax \cdot id] \,|\!) = depth \\
\equiv & \{\text{Fusão-const, Natural-id}\} \\
& (\!|\, [one, succ \cdot umax] \,|\!) = depth
\end{aligned}$$

Exercício 6

$$\begin{aligned}
&bubble\ (x : y : xs)\\
&\quad \mid x > y = y : bubble\ (x : xs)\\
&\quad \mid otherwise = x : bubble\ (y : xs)\\
&bubble\ x = x
\end{aligned}$$

$$\Longrightarrow$$

$$\begin{aligned}
&divide\ (x : y : xs)\\
&\quad \mid x > y = y\ ...\ (x : xs)\\
&\quad \mid otherwise = x\ ...\ (y : xs)\\
&divide\ x = x
\end{aligned}$$

$$\Longrightarrow$$

$$\begin{aligned}
&divide\ (x : y : xs)\\
&\quad \mid x > y = i_2\ (y, (x : xs))\\
&\quad \mid otherwise = i_2\ (x, (y : xs))\\
&divide\ x = i_1\ x
\end{aligned}$$

Podemos então inferir o tipo da função `divide`:

$$divide : A^* \rightarrow A^* + A \times A^*$$

Verificamos que o Functor Base necessário para formar este Hilomorfismo será o das **Listas com Sentinela**:

```
data SList a b = Stl b | Cons a (SList a b)
```

$$\mathbf{B}\,(Z, X, Y) = Z + X \times Y$$

Pelo tipo de `divide`, podemos inferir o tipo de `conquer`:

$$\begin{aligned}
&conquer : A^* + A \times A^* \rightarrow A^*\\
&conquer = [id, cons]
\end{aligned}$$

Exercício 7

$$Point : a \rightarrow a \rightarrow a \rightarrow Point\ a$$

$$\Downarrow$$

$$uncurry\ Point : (a, a) \rightarrow a \rightarrow Point\ a$$

$$\Downarrow$$

$$uncurry\ \$\ uncurry\ Point : ((a, a), a) \rightarrow Point\ a$$

$$\Downarrow$$

$$in = uncurry\ \$\ uncurry\ Point$$

Exercício 8

```haskell
module B_Tree where

import Cp

-- (1) Datatype definition

data B_Tree a = Nil | Block {leftmost :: B_Tree a, block :: [(a, B_Tree a)]}
    deriving (Show)

inB_Tree :: Either () (B_Tree a, [(a, B_Tree a)]) -> B_Tree a
inB_Tree = either (const Nil) (uncurry Block)

outB_Tree :: B_Tree a -> Either () (B_Tree a, [(a, B_Tree a)])
outB_Tree Nil = i1 ()
outB_Tree (Block l b) = i2 (l, b)

baseB_Tree g f = id -|- (f >< (map (g >< f)))

-- (2) Ana + cata + hylo

recB_Tree f = baseB_Tree id f

cataB_Tree g = g . (recB_Tree (cataB_Tree g)) . outB_Tree

anaB_Tree g = inB_Tree . (recB_Tree (anaB_Tree g)) . g

hyloB_Tree f g = cataB_Tree f . anaB_Tree g

-- (3) Map

instance Functor B_Tree where
fmap f = cataB_Tree (inB_Tree . baseB_Tree f id)

-- (4) Examples

-- (4.1) Count and depth

countB_Tree = cataB_Tree (either zero
                                 (add . (id >< sum . map (succ . p2))))

depthB_Tree = cataB_Tree (either zero
                                 (succ . umax . (id >< maximum . map p2)))

-- (4.2) Serialization

-- in-order traversal
inordtB_Tree = cataB_Tree (either nil (conc . (id >< (concat . map cons))))
```

```haskell
-- pre-order traversal
preordtB_Tree = cataB_Tree (either nil aux)
    where
        aux (l, []) = l
        aux (l, (h,a):t) = (h : l) ++ a ++ (concat (map cons t))

-- post-order traversal
postordtB_Tree = cataB_Tree (either nil (conc . (id >< (concat . map aux))))
    where
        aux (x, l) = l ++ [x]
```