

Cálculo de Programas

Algebra of Programming

UNIVERSIDADE DO MINHO
Lic. Ciências da Computação (2º ano)
Lic. em Engenharia Informática (3º ano)

2024/25 - Ficha (*Exercise sheet*) nr. 10

1. Considere o anamorfismo $r = \llbracket g \rrbracket$ em que

Consider the anamorphism $r = \llbracket g \rrbracket$ where

$$\begin{aligned} g [] &= i_1 () \\ g x &= i_2 (\text{last } x, \text{init } x) \end{aligned}$$

onde $\text{last } x$ dá o último elemento da lista x e $\text{init } x$ dá x sem esse último elemento. O que faz a função r ? Responda informalmente desenhando o diagrama de r .

in which $\text{last } x$ gives the last element of list x and $\text{init } x$ gives x without this last element. What does r do? Answer informally by drawing the diagram of r .

2. Considere a função:

Let function

$$x \ominus y = \text{if } x \leq y \text{ then } 0 \text{ else } 1 + x \ominus (y + 1)$$

Quais os valores das expressões $(3 \ominus 2) \ominus 3$ e $(3 \ominus 4) + 4$? Codifique $\widehat{\ominus} : \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{N}_0$ como um anamorfismo de naturais e faça o respectivo diagrama.

be given. Evaluate $(3 \ominus 2) \ominus 3$ and $(3 \ominus 4) + 4$ and encode $\widehat{\ominus} : \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{N}_0$ as an anamorphism over $\mathbb{N}_0$. Draw the corresponding diagram.

3. A função *concat*, extraída do *Prelude* do Haskell, é o catamorfismo de listas

The concat function, taken from the Haskell Prelude, is the list-catamorphism

$$\text{concat} = \llbracket [\text{nil}, \text{conc}] \rrbracket \tag{F1}$$

onde $\text{conc } (x, y) = x \mathbin{++} y$ e $\text{nil } _ = []$. Apresente justificações para a prova da propriedade

where $\text{conc } (x, y) = x \mathbin{++} y$ and $\text{nil } _ = []$. Provide justifications for proof of property

$$\text{length} \cdot \text{concat} = \text{sum} \cdot \text{map length} \tag{F2}$$

que a seguir se apresenta, onde é de esperar que as leis de *fusão-cata* e *absorção-cata* desempenhem um papel importante:

which is presented below, where the catafusion and cata-absorption laws are expected to play an important role:

$$\begin{aligned}
& \text{length} \cdot \text{concat} = \text{sum} \cdot \text{map length} \\
\equiv & \{ \dots \} \\
& \text{length} \cdot \llbracket [\text{nil}, \text{conc}] \rrbracket = \llbracket [0, \text{add}] \rrbracket \cdot \text{map length} \\
\equiv & \{ \dots \} \\
& \text{length} \cdot \llbracket [\text{nil}, \text{conc}] \rrbracket = \llbracket [0, \text{add}] \rrbracket \cdot (\text{id} + \text{length} \times \text{id}) \\
\Leftarrow & \{ \dots \} \\
& \text{length} \cdot [\text{nil}, \text{conc}] = [0, \text{add} \cdot (\text{length} \times \text{id})] \cdot (\text{id} + \text{id} \times \text{length}) \\
\equiv & \{ \dots \} \\
& \begin{cases} \text{length} \cdot \text{nil} = 0 \\ \text{length} \cdot \text{conc} = \text{add} \cdot (\text{length} \times \text{id}) \cdot (\text{id} \times \text{length}) \end{cases} \\
\equiv & \{ \dots \} \\
& \text{length} \cdot \text{conc} = \text{add} \cdot (\text{length} \times \text{length}) \\
\equiv & \{ \dots \} \\
& \text{true} \\
& \square
\end{aligned}$$

4. Recorra à lei da absorção-cata, entre outras, para verificar as seguintes propriedades sobre listas

Use the cata-absorption law, among others, to prove the following properties about lists

$$\text{length} = \text{sum} \cdot (\text{map } \underline{1}) \quad (\text{F3})$$

$$\text{length} = \text{length} \cdot (\text{map } f) \quad (\text{F4})$$

onde length , sum e map são catamorfismos de listas que conhece. (Recorda-se que o bifunctor de base para listas é $B(f, g) = \text{id} + f \times g$, de onde se deriva $F f = B(\text{id}, f) = \text{id} + \text{id} \times f$.)

where length, sum and map they are list-catamorphisms you know. (Remember that the basic bifunctor for lists is $B(f, g) = \text{id} + f \times g$, yielding $F f = B(\text{id}, f) = \text{id} + \text{id} \times f$.)

5. Seja dado o catamorfismo

Let catamorphism

$$\text{depth} = \llbracket [\text{one}, \text{succ} \cdot \text{umax}] \rrbracket \quad (\text{F5})$$

que dá a profundidade de árvores do tipo LTree , onde $\text{umax}(a, b) = \max a \ b$ e $\text{one} = \underline{1}$. Mostre, por absorção-cata, que a profundidade de uma árvore t não é alterada quando aplica uma função f a todas as suas folhas:

be given, which gives the depth of trees of type LTree , where $\text{umax}(a, b) = \max a \ b$ and $\text{one} = \underline{1}$. Show, by cata-absorption, that the depth of a tree t is not changed when you apply a function f to all its leaves:

$$\text{depth} \cdot \text{LTree } f = \text{depth} \quad (\text{F6})$$

6. O algoritmo “bubble-sort” é o ciclo-for

The “bubble-sort” algorithm is a for-loop:

```

bSort xs = for bubble xs (length xs) where
  bubble (x : y : xs)
    | x > y = y : bubble (x : xs)
    | otherwise = x : bubble (y : xs)
  bubble x = x

```

cujo corpo de ciclo é um hilomorfismo $\text{bubble} = \llbracket \text{conquer}, \text{divide} \rrbracket$. Recordando a hilo-factorização que se fez na aula teórica para a função *fib*, identifique os genes *divide* e *conquer* desse hilomorfismo.

Its loop-body is a hylomorphism $\text{bubble} = \llbracket \text{conquer}, \text{divide} \rrbracket$. Recalling the theory classes (cf. hylo-factorization of fib) identify the genes divide and conquer of this hylomorphism.

7. **Questão prática** — Este problema não irá ser abordado em sala de aula. Os alunos devem tentar resolvê-lo em casa e, querendo, publicarem a sua solução no canal **#geral** do Slack, com vista à sua discussão com colegas. Dão-se a seguir os requisitos do problema.

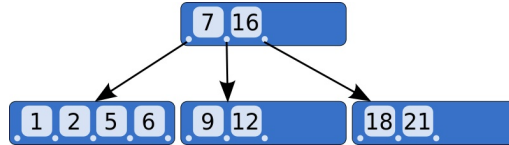
Open assignment — *This assignment will not be addressed in class. Students should try to solve it at home and, wishing so, publish their solutions in the #geral Slack channel, so as to trigger discussion among other colleagues. The requirements of the problem are given below.*

Problem requirements:

A “B-tree” is a generalization of the binary trees of the BTree module to more than two subtrees per node:

```
data B_tree a = Nil | Block { leftmost :: B_tree a, block :: [(a, B_tree a)] }
```

For instance, the B-tree



is represented by the data type above as follows:

```

t = Block {
  leftmost = Block {
    leftmost = Nil,
    block = [(1, Nil), (2, Nil), (5, Nil), (6, Nil)] },
  block = [
    (7, Block {
      leftmost = Nil,
      block = [(9, Nil), (12, Nil)] }),
    (16, Block {
      leftmost = Nil,
      block = [(18, Nil), (21, Nil)] })
  ]
}

```

Write a Haskell library for data type B_Tree following the same structure as the others already available, e.g. BTree.hs.

□