Cálculo de Programas Algebra of Programming

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

2023/24 - Ficha (Exercise sheet) nr. 10

1. 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

$$concat = ([nil, conc])$$
 (F1)

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

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

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

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:

2. O diagrama genérico de um catamorfismo de gene g sobre o tipo paramétrico T $X \cong$ B (X, T X) cuja base é o bifunctor B, bem como a sua propriedade universal, são representados a seguir:

The generic diagram of a catamorphism with gene g over the parametric type T $X \cong$ B (X, T X) with base B, as well as its universal property, are represented below:

$$k = (g) \equiv k \cdot \text{in} = g \cdot \underbrace{B(id, k)}_{F k}$$

De seguida, apresenta-se uma revisão do inventário de tipos indutivos da questão 6 da ficha anterior, recorrendo agora aos seus functores de base:

Next, a review of the inventory of inductive types of question 6 of the previous exercise sheet is given, now using its base-functors:

(a) Árvores com informação de tipo A nas folhas (Trees with data in their leaves):

$$\mathsf{T} = \mathsf{LTree}\ A \qquad \left\{ \begin{array}{l} \mathsf{B}\ (X,Y) = X + Y^2 \\ \mathsf{B}\ (g,f) = g + f^2 \end{array} \right. \text{ in } = [\mathit{Leaf}\ , \mathit{Fork}]$$

$$\mathsf{Haskell:}\ \mathbf{data}\ \mathsf{LTree}\ a = \mathit{Leaf}\ a \mid \mathit{Fork}\ (\mathsf{LTree}\ a, \mathsf{LTree}\ a)$$

(b) Árvores com informação de tipo A nos nós (Trees whose data of type A are stored in their nodes):

(c) Árvores com informação nos nós e nas folhas (Full trees — data in both leaves and nodes):

T = FTree
$$B$$
 A
$$\begin{cases} \mathbf{B} \ (Z,X,Y) = Z + X \times Y^2 \\ \mathbf{B} \ (h,g,f) = h + g \times f^2 \end{cases} \text{ in } = [\mathit{Unit} \ , \mathit{Comp}]$$
 Haskell: $\mathbf{data} \ \mathsf{FTree} \ b \ a = \mathit{Unit} \ b \ | \mathit{Comp} \ (a,(\mathsf{FTree} \ b \ a,\mathsf{FTree} \ b \ a))$

(d) Árvores de expressão (Expression trees):

T = Expr
$$V$$
 O
$$\begin{cases} \mathbf{B} \ (Z,X,Y) = Z + X \times Y^* \\ \mathbf{B} \ (h,g,f) = h + g \times \mathsf{map} \ f \end{cases} \mathsf{in} = [\mathit{Var} \ , \mathit{Term}]$$
 Haskell: $\mathbf{data} \ \mathsf{Expr} \ v \ o = \mathit{Var} \ v \ | \mathit{Term} \ (o, [\mathsf{Expr} \ v \ o])$

Partindo da definição genérica de map associado ao tipo T,

Starting from the generic definition of map associated with the type T,

$$\mathsf{T}\,f = (\!(\mathsf{in} \cdot \mathsf{B}\,(f,id)\!)\!)$$

calcule $fmap \ f = T \ f$ para T := BTree, entregando o resultado em Haskell sem combinadores pointfree. (Repare-se que se tem sempre F k = B (id, k).

derive $fmap \ f = T \ f \ for \ T := BTree, \ de$ livering the result in Haskell without pointfree combinators. (Note that we always have F k = B (id, k).

3. 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

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

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

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

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

4. Seja dado o catamorfismo

Let catamorphism

$$depth = ([one, succ \cdot umax])$$

que dá a profundidade de árvores do tipo LTree, onde umax (a,b) = max a b. 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 $umax(a,b) = max \ a \ b$. Show, by cata-absorption, that the depth of a tree t is not changed when you apply a function f to all its leaves:

$$depth \cdot \mathsf{LTree}\ f = depth$$
 (F5)

5. Um anamorfismo é um "catamorfismo ao contrário", i.é uma função $k:A\to T$ tal que

An anamorphism is a "reverse catamorphism", i.e. a function $k: A \to T$ such that

$$k = \mathsf{in} \cdot \mathsf{F} \ k \cdot g \tag{F6}$$

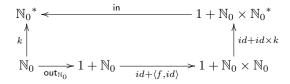
escrevendo-se $k = [\![g]\!]$. Mostre que o anamorfismo de listas

One writes k = [g]. Show that the list-anamorphism

$$k = [(id + \langle f, id \rangle) \cdot \mathsf{out}_{\mathbb{N}_0}] \tag{F7}$$

descrito pelo diagrama

depicted in diagram



é a função

is the function

$$k \ 0 = []$$

 $k \ (n+1) = (2 \ n+1) : k \ n$

para $f \ n=2 \ n+1$. (Que faz esta função?)

for f n = 2 n + 1. (What does this function do?)

6. Mostre que o anamorfismo que calcula os sufixos de uma lista

Show that the anamorphism that computes the suffixes of a list

$$suffixes = \llbracket (g) \rrbracket$$
 where $g = (id + \langle \mathsf{cons}, \pi_2 \rangle) \cdot \mathsf{out}$

é a função:

is the function:

$$suffixes[] = []$$

 $suffixes(h:t) = (h:t): suffixes t$

7. O formulário desta disciplina apresenta duas definições alternativas para o functor T f, uma como *catamorfismo* e outra como *anamorfismo*. Identifique-as e acrescente justificações à seguinte prova de que essas definições são equivalentes:

The reference sheet of this course presents two alternative definitions for functor T f, one as a catamorphism and another as an anamorphism. Identify them and fill in justifications in the following proof that such definitions are equivalent:

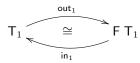
$$\begin{array}{lll} \mathsf{T}\,f = \{ (\mathsf{in} \cdot \mathsf{B}\,(f,id)) \} \\ & \equiv & \{ & & & \} \\ & \mathsf{T}\,f \cdot \mathsf{in} = \mathsf{in} \cdot \mathsf{B}\,(f,id) \cdot \mathsf{F}\,(\mathsf{T}\,f) \\ & \equiv & \{ & & & \} \\ & \mathsf{T}\,f \cdot \mathsf{in} = \mathsf{in} \cdot \mathsf{B}\,(id,\mathsf{T}\,f) \cdot \mathsf{B}\,(f,id) \\ & \equiv & \{ & & & \} \\ & \mathsf{out} \cdot \mathsf{T}\,f = \mathsf{F}\,(\mathsf{T}\,f) \cdot \mathsf{B}\,(f,id) \cdot \mathsf{out} \\ & \equiv & \{ & & & \} \\ & \mathsf{T}\,f = [\![\mathsf{B}\,(f,id) \cdot \mathsf{out}]\!] \end{array}$$

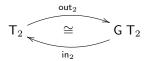
8. Mostre que o catamorfismo de listas length = $([zero, succ \cdot \pi_2])$ é a mesma função que o anamorfismo de naturais $[(id + \pi_2) \cdot out_{List}]$.

Show that the list catamorphism length = $\{ [\text{zero }, \text{succ} \cdot \pi_2] \}$ is the same function as the \mathbb{N}_0 -anamorphism $\{ (id + \pi_2) \cdot \text{out}_{\mathsf{List}} \}$.

9. O facto de length : $A^* \to \mathbb{N}_0$ poder ser definida tanto como *catamorfismo* de listas como *anamorfismo* de naturais (questão 8) pode generalizar-se da forma seguinte: sejam dados dois tipos indutivos

The fact that length: $A^* \to \mathbb{N}_0$ is both a list-catamorphism and a \mathbb{N}_0 -anamorphism (question 8) generalizes as follows: let two inductive types be given





e $\alpha: \mathsf{F} \ X \to \mathsf{G} \ X$, isto é, α satisfaz a propriedade $\mathit{grátis}$:

and $\alpha: \mathsf{F} \ X \to \mathsf{G} \ X$, that is, α satisfying the free property:

$$\mathsf{G}\,f\cdot\alpha=\alpha\cdot\mathsf{F}\,f\tag{F8}$$

Então $(in_2 \cdot \alpha) = [(\alpha \cdot out_1)]$, como se mostra a seguir (complete as justificações):

Then $\{\inf_2 \cdot \alpha\} = [\alpha \cdot \mathsf{out}_1], \text{ as shown below (complete the justifications):}$

$$\begin{array}{lll} k = (\!\!\lceil \operatorname{in}_2 \cdot \alpha \,) \\ & & & \\ k \cdot \operatorname{in}_1 = \operatorname{in}_2 \cdot \alpha \cdot \mathsf{F} \, k \\ \\ & & \\ & & \\ \operatorname{out}_2 \cdot k = \mathsf{G} \, k \cdot \alpha \cdot \operatorname{out}_1 \\ \\ & & \\ k = (\!\!\lceil \alpha \cdot \operatorname{out}_1 \!\!\rceil) \end{array} \right]$$

Como aplicação imediata deste resultado, redefina

Use the result above to redefine

defina

$$\left\{ \begin{array}{l} \textit{mirror} :: \mathsf{LTree}\ a \to \mathsf{LTree}\ a \\ \textit{mirror} = \{\!\!\{ \mathsf{in} \cdot (id + \mathsf{swap}) \}\!\!\} \end{array} \right. \tag{F9}$$

como um anamorfismo.

as an anamorphism.