

Cálculo de Programas

Algebra of Programming

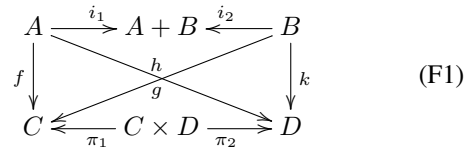
Lic./Mest.Int. em Engenharia Informática (3º ano)
 Lic. Ciências da Computação (2º ano)
 UNIVERSIDADE DO MINHO

2023/24 - Ficha nr.º 4

1. A *lei da troca* (identifique-a no formulário) permite-nos exprimir determinadas funções sob duas formas alternativas, conforme desenhado no respectivo diagrama:

$$[\langle f, g \rangle, \langle h, k \rangle] = \langle [f, h], [g, k] \rangle$$

The exchange law (check this law in the reference sheet) allows one to express certain functions in two alternative forms, as given below:



Demonstre esta lei recorrendo às propriedades (e.g. universais) dos produtos e dos coprodutos.

Prove this law using the (e.g. universal) properties of products and co-products.

2. Use a lei da troca (F1) para exprimir o isomorfismo

$$\text{undistl} = [i_1 \times id, i_2 \times id]$$

sob a forma de um ‘split’ de alternativas.

Use (F1) to express the isomorphism

in the form of a ‘split’ of alternatives.

3. Sabendo que as igualdades

$$p \rightarrow k, k = k \tag{F2}$$

$$(p? + p?) \cdot p? = (i_1 + i_2) \cdot p? \tag{F3}$$

se verificam, demonstre as seguintes propriedades do mesmo combinador:

prove the following laws of the McCarthy conditional:

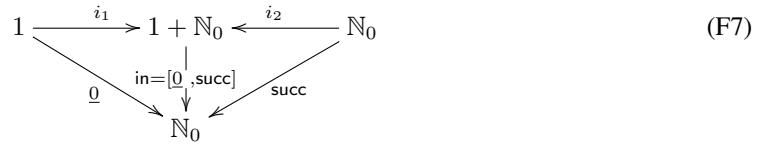
$$\langle (p \rightarrow f, h), (p \rightarrow g, i) \rangle = p \rightarrow \langle f, g \rangle, \langle h, i \rangle \tag{F4}$$

$$\langle f, (p \rightarrow g, h) \rangle = p \rightarrow \langle f, g \rangle, \langle f, h \rangle \tag{F5}$$

$$p \rightarrow (p \rightarrow a, b), (p \rightarrow c, d) = p \rightarrow a, d \tag{F6}$$

4. Considere a função $\text{in} = [\underline{0}, \text{succ}]$ (onde $\text{succ } n = n + 1$) que exprime a forma como os números naturais (\mathbb{N}_0) são gerados a partir do número 0, de acordo com o diagrama seguinte:

Let function $\text{in} = [\underline{0}, \text{succ}]$ be given (where $\text{succ } n = n + 1$) expressing the way natural numbers (\mathbb{N}_0) are generated from the number 0, according to the diagram below:



Sabendo que o tipo 1 coincide com o tipo $()$ em Haskell e é habitado por um único elemento, também designado por $()$, calcule a inversa de in ,

Knowing that type 1 matches type $()$ in Haskell and is inhabited by a single element, also denoted by $()$, find the inverse of in ,

$$\begin{aligned}
 \text{out } 0 &= i_1 () \\
 \text{out } (n + 1) &= i_2 n
 \end{aligned}$$

resolvendo em ordem a out a equação

by solving the equation

$$\text{out} \cdot \text{in} = \text{id} \tag{F8}$$

e introduzindo variáveis.

for out and adding variables.

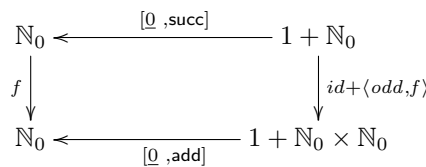
5. Na sequência da questão anterior, considere a equação em f

As follow up to the previous question, consider the equation in f

$$f \cdot [\underline{0}, \text{succ}] = [\underline{0}, \text{add}] \cdot (\text{id} + \langle \text{odd}, f \rangle) \tag{F9}$$

a que corresponde o diagrama

captured by diagram



onde $\text{add } (x, y) = x + y$ e $\text{odd } n = 2n + 1$. Use as leis do cálculo de programas para mostrar que a única solução para essa equação é a função:

where $\text{add } (x, y) = x + y$ and $\text{odd } n = 2n + 1$. Show that the solution to this equation is the function (in Haskell syntax):

$$\begin{cases} f 0 = 0 \\ f (n + 1) = (2n + 1) + f n \end{cases}$$

“Corra” mentalmente a função f para vários casos, eg. $f 0, f 1, f 2$ e responda: o que faz (ou parece fazer) a função f ? (A sua resposta, informal para já, será formalmente validada mais para a frente.)

“Run” function f mentally for various inputs, e.g. $f 0, f 1, f 2$ and answer: what does function f do (or seems to do)? (Your answer, informal for now, will be formally validated later on.)

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
```

faz sentido definir a função que mostra como construir árvores deste tipo:

$$\text{in} = [\text{Leaf}, \text{Fork}] \tag{F10}$$

Desenhe um diagrama semelhante a (F7) para esta função e calcule a sua inversa

$$\begin{aligned} \text{out} (\text{Leaf } a) &= i_1 a \\ \text{out} (\text{Fork } (x, y)) &= i_2 (x, y) \end{aligned}$$

de novo resolvendo a equação $\text{out} \cdot \text{in} = \text{id}$ em ordem a *out*, agora para o (F10). Finalmente, faça testes em Haskell que envolvam a composição $\text{in} \cdot \text{out}$ e tire conclusões.

Consider the following definition in Haskell of a particular type of binary tree:

By querying the types of constructors *Leaf* and *Fork* in GHCi, for example,

one can define

capturing how data of this type are built. Draw a diagram for this function similar to (F7) and find its inverse,

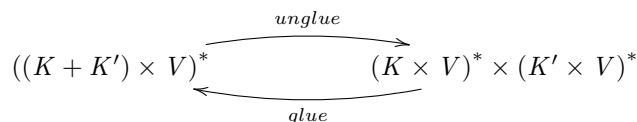
again solving the equation $\text{out} \cdot \text{in} = \text{id}$ for *out*, but now with respect to (F10). Finally, run tests in Haskell involving the composition $\text{in} \cdot \text{out}$ and draw conclusions.

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.

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.

Problem requirements:

Well-known services such as Google Maps, Google Analytics, YouTube, MapReduce etc. run on top of Bigtable or successors thereof. Such data systems rely on the so-called key-value NoSQL data model, which is widely adopted because of its efficiency and flexibility. Key-value stores can be regarded abstractly as lists of pairs $(K \times V)^*$ in which *K* is a datatype of keys and *V* is a type of data values. Keys uniquely identify values. Key-value stores with the same type *V* of values can be glued together as the diagram suggests,



where *unglue* performs the action opposite to *glue*.

Define *glue* and *unglue* in Haskell structured along the functional combinators $(f \cdot g, \langle f, g \rangle, f \times g)$ and *so on* studied in this course and available from library *Cp.hs*. Use **diagrams** to plan your solutions, in which you should avoid re-inventing functions over lists already available in the Haskell standard libraries.

□