Cálculo de Programas

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

2025/26 - Ficha nr.º 4

1. Demonstre a igualdade

$$[\underline{k}, \underline{k}] = \underline{k} \tag{F1}$$

recorrendo à propriedade universal-+ e a uma lei que qualquer função constante \underline{k} satisfaz. (Consultar o formulário.)

2. Recorra às leis dos coprodutos para mostrar que a definição que conhece da função factorial,

$$\begin{cases} fac \ 0 = 1 \\ fac \ (n+1) = (n+1) * fac \ n \end{cases}$$
 (F2)

é equivalente à equação seguinte

$$fac \cdot [\underline{0}, \mathsf{succ}] = [\underline{1}, \mathsf{mul} \cdot \langle \mathsf{succ}, fac \rangle]$$

onde

$$\operatorname{succ} n = n + 1$$
$$\operatorname{mul} (a, b) = a * b$$

3. A função in $= [\underline{0}]$, succ] da questão anterior exprime, para succ n = n + 1, a forma como os números naturais (\mathbb{N}_0) são gerados a partir do número 0, de acordo com o diagrama seguinte:

$$1 \xrightarrow{i_1} 1 + \mathbb{N}_0 \xleftarrow{i_2} \mathbb{N}_0$$

$$\underbrace{0} \text{ in} = \underbrace{0}_{\mathbb{N}_0}, \text{succ}$$

$$\mathbb{N}_0$$
(F3)

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

resolvendo em ordem a out a equação

$$\operatorname{out} \cdot \operatorname{in} = id$$
 (F5)

e introduzindo variáveis.

4. Verifique no GHCi que a seguinte função

$$fac = [\underline{1}, \mathsf{mul}] \cdot (id + \langle \mathsf{succ}, fac \rangle) \cdot \mathsf{out}$$

a que corresponde o diagrama

calcula o factorial da sua entrada (F1), assumindo out (F4) e mul (a, b) = a * b já definidas.

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

data LTree
$$a = Leaf \ a \mid Fork \ (LTree \ a, LTree \ a)$$

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

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

$$in = [Leaf, Fork]$$
 (F6)

Desenhe um diagrama para esta função e calcule a sua inversa

out
$$(Leaf\ a)=i_1\ a$$

out $(Fork\ (x,y))=i_2\ (x,y)$

de novo resolvendo a equação out \cdot in =id em ordem a out, agora para o (F6).

Finalmente, faça testes em Haskell que involvam a composição in · out e tire conclusões.

6. Considere o isomorfismo

$$(A+B) + C \underset{\text{coassocl}}{\overset{\text{coassocr}}{\cong}} A + (B+C)$$

onde coassocr $= [id + i_1 \ , i_2 \cdot i_2]$. Calcule a sua conversa resolvendo em ordem a coassocl a equação,

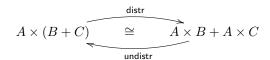
$$coassocl \cdot coassocr = id$$

isto é, a equação

$$\operatorname{coassocl} \cdot \underbrace{[id+i_1\;,i_2\cdot i_2]}_{\operatorname{coassocr}} = id$$

Finalmente, exprima coassocl sob a forma de um programa em Haskell *não recorra* ao combinador "either" e teste as duas versões no GHCi.

7. Os isomorfismos



estudados na aula teórica estão codificados na biblioteca Cp.hs. Supondo A = String, $B = \mathbb{B}$ e $C = \mathbb{Z}$, (a) aplique no GHCi undistr, alternativamente, aos pares ("CP", TRUE) ou ("LEI", 1); (b) verifique que (distr·undistr) x = x para essas (e quaisquer outras) situações que possa testar.

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

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

$$A \xrightarrow{i_1} A + B \xrightarrow{i_2} B$$

$$f \downarrow g$$

$$C \xleftarrow{\pi_1} C \times D \xrightarrow{\pi_2} D$$
(F7)

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

9. 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.

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,

$$((K + K') \times V)^* \underbrace{(K \times V)^* \times (K' \times V)^*}_{glue}$$

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 from the Haskell standard libraries.