

Cálculo de Programas

Algebra of Programming

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

2026/27 — Ficha (Worksheet) nr.º 4

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

Show by coproduct laws that the usual definition of the factorial function,

$$\begin{cases} fac\ 0 = 1 \\ fac\ (n + 1) = (n + 1) * fac\ n \end{cases} \quad (F1)$$

é equivalente à equação seguinte

is equivalent the following equation,

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

onde

where

$$\begin{aligned} succ\ n &= n + 1 \\ mul\ (a, b) &= a * b \end{aligned}$$

2. 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:

Function $in = [\underline{0}, succ]$ in the previous question (where $succ\ n = n + 1$) expresses the way natural numbers ($\mathbb{N}_0$) are generated from the number 0, according to the diagram below:

$$\begin{array}{ccccc} 1 & \xrightarrow{i_1} & 1 + \mathbb{N}_0 & \xleftarrow{i_2} & \mathbb{N}_0 \\ & \searrow \underline{0} & \downarrow in = [\underline{0}, succ] & \swarrow succ & \\ & & \mathbb{N}_0 & & \end{array} \quad (F2)$$

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,

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

$$\begin{cases} out\ 0 = i_1\ () \\ out\ (n + 1) = i_2\ n \end{cases} \quad (F3)$$

resolvendo em ordem a out a equação

by solving the equation

$$\text{out} \cdot \text{in} = \text{id} \quad (\text{F4})$$

e introduzindo variáveis.

for out and adding variables.

3. Verifique no GHCi que a seguinte função

On GHCi check that function

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

a que corresponde o diagrama

captured by diagram

$$\begin{array}{ccc} \mathbb{N}_0 & \xrightarrow{\text{out}} & 1 + \mathbb{N}_0 \\ \text{fac} \downarrow & & \downarrow \text{id} + \langle \text{succ}, \text{fac} \rangle \\ \mathbb{N}_0 & \xleftarrow{[\underline{1}, \text{mul}]} & 1 + \mathbb{N}_0 \times \mathbb{N}_0 \end{array}$$

calcula o factorial da sua entrada (F1), assumindo out (F3) e mul $(a, b) = a * b$ já definidas.

computes de factorial of its input, assuming out (F3) and mul $(a, b) = a * b$ already defined.

4. Considere o isomorfismo

Consider the isomorphism

$$(A + B) + C \begin{array}{c} \xrightarrow{\text{coassocr}} \\ \cong \\ \xleftarrow{\text{coassocl}} \end{array} A + (B + C)$$

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

where $\text{coassocr} = [\text{id} + i_1, i_2 \cdot i_2]$. Find its converse coassocl by solving the equation,

$$\text{coassocl} \cdot \text{coassocr} = \text{id}$$

isto é, a equação

that is, the equation

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

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

Finally express coassocl in pointwise Haskell code not using the “either” combinator and test both versions in GHCi.

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

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

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

$$\begin{array}{ccccc} A & \xrightarrow{i_1} & A + B & \xleftarrow{i_2} & B \\ f \downarrow & & \downarrow h & & \downarrow k \\ C & \xleftarrow{\pi_1} & C \times D & \xrightarrow{\pi_2} & D \end{array} \quad (\text{F5})$$

Demonstre esta lei recorrendo às propriedades (e.g. universais) dos produtos e dos coprodutos.	<i>Prove this law using the (e.g. universal) properties of products and co-products.</i>
6. Deduza o tipo mais geral da função $\alpha = (id + \pi_1) \cdot i_2 \cdot \pi_2$ e infira a respectiva propriedade grátis (natural) através de um diagrama.	<i>Infer the most general type of function $\alpha = (id + \pi_1) \cdot i_2 \cdot \pi_2$ and then its free (natural) property using a diagram.</i>
7. Considere as seguintes funções elementares que respectivamente juntam ou duplicam informação:	<i>Let the following basic functions be given that, respectively, gather or duplicate information:</i>
	$join = [id, id]$ (F6)
	$dup = \langle id, id \rangle$ (F7)
Calcule (justificando) a propriedade grátis da função $\alpha = dup \cdot join$ e indique por que razão não pode calcular essa propriedade para $join \cdot dup$.	<i>Calculate (justifying) the free property of the function $\alpha = dup \cdot join$ and indicate why you cannot calculate this property for $join \cdot dup$.</i>
8. Seja dada uma função α cuja propriedade grátis é:	<i>Let α be a function with free property:</i>
	$(f + h) \cdot \alpha = \alpha \cdot (f + g \times h)$ (F8)
Será esta propriedade suficiente para deduzir a definição de α ? Justifique analiticamente.	<i>Can a definition of α be inferred from (F8)? Justify.</i>
9. No cálculo de programas, as definições condicionais do tipo	<i>Conditional expressions of pattern</i>
	$h\ x = \text{if } p\ x \text{ then } f\ x \text{ else } g\ x$
são escritas usando o combinador ternário	<i>are expressed in the algebra of programming by the ternary combinator</i>
	$p \rightarrow f, g$
conhecido pelo nome de <i>condicional de McCarthy</i> , cuja definição	<i>known as the McCarthy conditional, whose definition</i>
	$p \rightarrow f, g = [f, g] \cdot p?$ (F9)
vem no formulário. Baseie-se em leis desse formulário para demonstrar a chamada 2ª-lei de fusão do condicional:	<i>can be found in reference sheet. Use this reference sheet to prove the so-called 2nd fusion-law of conditionals:</i>
	$(p \rightarrow f, g) \cdot h = (p \cdot h) \rightarrow (f \cdot h), (g \cdot h)$

10. Sabendo que as igualdades

Assuming

$$p \rightarrow k, k = k \quad (\text{F10})$$

$$(p? + p?) \cdot p? = (i_1 + i_2) \cdot p? \quad (\text{F11})$$

se verificam, demonstre as seguintes propriedades do mesmo combinador:

prove the following laws of the McCarthy conditional:

$$\langle (p \rightarrow f, h), (p \rightarrow g, i) \rangle = p \rightarrow \langle f, g \rangle, \langle h, i \rangle \quad (\text{F12})$$

$$\langle f, (p \rightarrow g, h) \rangle = p \rightarrow \langle f, g \rangle, \langle f, h \rangle \quad (\text{F13})$$

$$p \rightarrow (p \rightarrow a, b), (p \rightarrow c, d) = p \rightarrow a, d \quad (\text{F14})$$

11. **Questão prática** — Este problema não irá ser abordado em sala de aula. Os alunos devem tentar resolvê-lo em casa e, querendo, publicar 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,

$$\begin{array}{ccc} & \xrightarrow{\text{unglue}} & \\ ((K + K') \times V)^* & & (K \times V)^* \times (K' \times V)^* \\ & \xleftarrow{\text{glue}} & \end{array}$$

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.

□