

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 3

1. Recorde a propriedade universal do combinador $[f, g]$,

Recall the universal property of the $[f, g]$ combinator,

$$k = [f, g] \equiv \begin{cases} k \cdot i_1 = f \\ k \cdot i_2 = g \end{cases}$$

da qual, como sabe, podem ser derivadas todas as outras que aparecem no respectivo grupo, no formulário.

from which, as you know, all the others appearing in the corresponding group of the reference sheet can be derived.

Use esta lei para demonstrar a lei

Use this property to prove the law

$$[i_1, i_2] = id$$

conhecida por *reflexão*-+.

known as +-reflexion.

-
2. Demonstre a igualdade

Prove the equality

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

recorrendo à propriedade universal acima e a uma lei que qualquer função constante k satisfaz. (Ver no formulário.)

using the universal property given above and a law that any constant function k satisfies. (Check the reference sheet.)

-
3. Seja dada a função $\text{coswap} = [i_2, i_1]$. Faça um diagrama que explique o tipo de coswap e mostre que $\text{coswap} \cdot \text{coswap} = id$.

Let function $\text{coswap} = [i_2, i_1]$ be given. Draw a diagram explaining the type of coswap and show that $\text{coswap} \cdot \text{coswap} = id$ holds.

-
4. Considere a função

Let function

$$\alpha = [\langle \text{FALSE}, id \rangle, \langle \text{TRUE}, id \rangle] \tag{F2}$$

Determine o tipo de α e mostre, usando a propriedade *universal*-+, que α se pode escrever em Haskell da forma seguinte:

be given. Infer the type of α and show, using the +-universal law, that α can be written in pointwise Haskell as follows:

$$\alpha (\text{Left } a) = (\text{FALSE}, a)$$

$$\alpha (\text{Right } a) = (\text{TRUE}, a)$$

5. O combinador funcional *soma* define-se por: $f + g = [i_1 \cdot f, i_2 \cdot g]$. Identifique no formulário os nomes das propriedades que se seguem e demonstre-as usando o cálculo de programas.

The sum of two functions f and g is defined by $f + g = [i_1 \cdot f, i_2 \cdot g]$. Check the names of the three laws that are given below in the reference sheet and prove them using the algebra of programming.

$$id + id = id \tag{F3}$$

$$(f + g) \cdot i_1 = i_1 \cdot f \tag{F4}$$

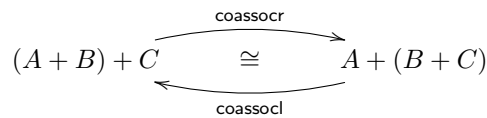
$$(f + g) \cdot i_2 = i_2 \cdot g \tag{F5}$$

6. Deduza o tipo mais geral da função $\alpha = (id + \pi_1) \cdot i_2 \cdot \pi_2$ e represente-o através de um diagrama.

Infer the most general type of function $\alpha = (id + \pi_1) \cdot i_2 \cdot \pi_2$ and draw it in a diagram of compositions.

7. Considere o isomorfismo

Consider the isomorphism



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

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

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

isto é, a equação

that is, the equation

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

Finalmente, exprima coassocl sob a forma de um programa em Haskell *não recorra* ao combinador “either”.

Finally express coassocl in pointwise Haskell code not using the “either” combinator.

8. 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,

$$fac\ 0 = 1$$

$$fac\ (n + 1) = (n + 1) * fac\ n$$

é equivalente à equação seguinte

is equivalent the following equation,

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

onde

where

$$succ \ n = n + 1$$

$$mul \ (a, b) = a * b$$

-
9. No cálculo de programas, as definições condicionais do tipo

Conditional expressions of pattern

$$h \ x = \mathbf{if} \ p \ x \ \mathbf{then} \ f \ x \ \mathbf{else} \ g \ x$$

são escritas usando o combinador ternário

are expressed in the algebra of programming by the ternary combinator

$$p \rightarrow f, g$$

conhecido pelo nome de *condicional de McCarthy*, cuja definição

known as the McCarthy conditional, whose definition

$$p \rightarrow f, g = [f, g] \cdot p? \tag{F6}$$

vem no formulário. Baseie-se em leis desse formulário para demonstrar a chamada 2ª-lei de fusão do condicional:

can be found in reference sheet. Use this reference sheet to prove the so-called 2nd fusion-law of conditionals:

$$(p \rightarrow f, g) \cdot h = (p \cdot h) \rightarrow (f \cdot h), (g \cdot h)$$

-
10. **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: *The solution given for a previous problem,*

$$store \ c = take \ 10 \cdot nub \cdot (c:) \tag{F7}$$

calls the standard function

$$nub \ (Eq \ a) \Rightarrow [a] \rightarrow [a]$$

available from the Data.List library in Haskell.

After inspecting the standard implementation of this function, define f so that

$$nub = [nil, cons] \cdot f.$$

is an alternative to the standard definition, where $nil \ _ = []$ and $cons \ (h, t) = h : t$. Check that $store \ c$ (F7) works properly once the standard nub is replaced by yours.

Important: *Structure your solution across the $f \cdot g$, $\langle f, g \rangle$, $f \times g$, $[f, g]$ and $f + g$ combinators available from library $Cp.hs$. Use **diagrams** to plan your solution, in which you should avoid re-inventing functions over lists already available in the Haskell standard libraries.*

□