Cálculo de Programas
Algebra of Programming

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

2024/25 - Ficha *(Exercise sheet)* nr. 12 – última *(last)*

1. Demonstre a seguinte propriedade da composição monádica:

Prove the following property of monadic composition:

$$f \bullet [g\ , h] = [f \bullet g\ , f \bullet h] \tag{F1}$$

---

2. Considere a função

Consider

$$\begin{array}{l} \mathit{discollect} : (A \times B^*)^* \to (A \times B)^* \\ \mathit{discollect} = \mathit{lstr} \bullet \mathit{id} \end{array}$$

onde $\mathit{lstr}\ (a,x) = [(a,b) \mid b \leftarrow x]$, no mónade das listas:

where $\mathit{lstr}\ (a,x) = [(a,b) \mid b \leftarrow x]$*, in the list-monad:*

$$A \xrightarrow{\mathsf{singl}} A^* \xleftarrow{\mathit{concat}} (A^*)^*$$

Recordando $\mathit{concat} = (\!|\,[\mathsf{nil}\ , \mathsf{conc}]\,|\!)$ e a lei de absorção-cata (para listas), derive uma definição recursiva para $\mathit{discollect}$ que não use nenhum dos combinadores 'point-free' estudados nesta disciplina.

Recalling $\mathit{concat} = (\!|\,[\mathsf{nil}\ , \mathsf{conc}]\,|\!)$ *and cata-absorption (for lists), derive a recursive definition for* $\mathit{discollect}$ *that uses none of the 'point-free' combinators studied in this course.*

---

3. Pretende-se um mónade que consiga calcular o tempo de execução de programas funcionais de forma composicional. Para isso, define-se

The aim is to create a monad that can calculate the execution time of functional programs in a compositional way. To do this, define

$$\mathsf{T}\ X = X \times \mathbb{R} \tag{F2}$$

onde cada par (x,t) de $\mathsf{T}\ X$ regista o facto de o valor x ter sido obtido à custa de t unidades de tempo (e.g. milisegundos). De seguida, define-se o mónade $X \xrightarrow{u} \mathsf{T}X \xleftarrow{\mu} \mathsf{T}^2X$:

where each pair (x,t) *of* $\mathsf{T}\ X$ *records the fact that the value* x *to have been obtained at the expense of* t *units of time (e.g. milliseconds). Next, the monad* $X \xrightarrow{u} \mathsf{T}X \xleftarrow{\mu} \mathsf{T}^2X$ *is defined:*

$$\mathsf{T}\ f = f \times id \tag{F3}$$

$$u\ x = (x, 0) \tag{F4}$$

$$\mu\ ((x,t_1),t_2) = (x, t_1 + t_2) \tag{F5}$$

Vê-se bem como μ faz a adição dos tempos de execução. Contudo, para T ser um mónade terá de satisfizer as leis (62) e (63) do formulário. Prove que assim acontece.

It can be seen as μ adds execution times. However, for T to be a monad it must satisfy the laws (62) and (63) of the reference sheet. Prove that it so happens.

---

4. Suponha um tipo indutivo $\mathsf{T}\ X$ cuja base é o bifunctor

Let an inductive type $\mathsf{T}\ X$ be given whose base is the bifunctor

$$\mathsf{B}\ (X, Y) = X + \mathsf{G}\ Y$$
$$\mathsf{B}\ (f, g) = f + \mathsf{G}\ g$$

onde G é um outro qualquer functor. Mostre que $\mathsf{T}\ X$ é um mónade em que

where G is any other functor. Show that $\mathsf{T}\ X$ is a monad in which

$$\begin{cases} \mu = (\!|\, [id\ , \mathsf{in} \cdot i_2]\, |\!) \\ u = \mathsf{in} \cdot i_1 \end{cases} \tag{F6}$$

onde $\mathsf{in} : \mathsf{B}\ (X, \mathsf{T}\ X) \to \mathsf{T}\ X$.

where $\mathsf{in} : \mathsf{B}\ (X, \mathsf{T}\ X) \to \mathsf{T}\ X$.

---

5. (a) Alguns mónades conhecidos resultam de (F6). Mostre que é o caso de LTree — identifique G para esse caso; (b) Para $\mathsf{G}\ Y = 1$ (i.é $\mathsf{G}\ f = id$) qual é o mónade que se obtém por (F6)? E no caso em que $\mathsf{G}\ Y = O \times Y^*$, onde o tipo O se considera fixo à partida?

(a) Some known monads result from (F6). Show that LTree does so (for which G?) (b) For $\mathsf{G}\ Y = 1$ (ie $\mathsf{G}\ f = id$) what is the monad obtained by (F6)? And in the case where $\mathsf{F}\ Y = O \times Y^*$, where the type O is considered fixed at the outset?

---

6. Seja M um monad e T um functor. Em Haskell, a instância para listas ($\mathsf{T}\ X = X^*$) da função monádica

Let M be a monad and T a functor. In Haskell, the instance for lists ($\mathsf{T}\ X = X^*$) of the monadic function

$$\mathsf{sequence} : \mathsf{T}\ (\mathsf{M}\ X) \to \mathsf{M}\ (\mathsf{T}\ X)$$

é o catamorfismo

is the catamorphism

$$\begin{array}{l} \mathsf{sequence} = (\!|\, g\, |\!)\ \mathbf{where} \\ \quad g = [\mathsf{return}\ , id] \cdot (\mathsf{nil} + \lfloor \mathsf{cons} \rfloor) \\ \quad \lfloor f \rfloor\ (x, y) = \mathbf{do}\ \{ a \leftarrow x; b \leftarrow y; \mathsf{return}\ (f\ (a, b)) \} \end{array}$$

tal como se mostra neste diagrama:

as in the following diagram:

$$\begin{tikzcd}
(\mathsf{M}\ X)^* \arrow[d, "\mathsf{sequence}"'] & & 1 + \mathsf{M}\ X \times (\mathsf{M}\ X)^* \arrow[ll, "\mathsf{in}"'] \arrow[d, "id + id \times \mathsf{sequence}"] \\
\mathsf{M}\ (X^*) & & 1 + \mathsf{M}\ X \times \mathsf{M}\ (X^*) \arrow[ll, dotted, "g"'] \arrow[d, "\mathsf{nil} + \lfloor \mathsf{cons} \rfloor"] \\
& & X^* + \mathsf{M}\ (X^*) \arrow[llu, dotted, "{[\mathsf{return}\ , id]}"]
\end{tikzcd}$$

Partindo da propriedade universal-cata, derive uma versão de $\mathsf{sequence}$ em Haskell com variáveis que não recorra à composição de funções.

Starting from the universal-cata property, derive a version of $\mathsf{sequence}$ in Haskell with variables that doesn't resort to function composition.