Cálculo de Programas
Algebra of Programming

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

2023/24 - Ficha *(Exercise sheet)* nr. 12 – última *(last)*

1. Um mónade é um functor T equipado com duas funções μ e u,

   A monad is a functor T equipped with two functions μ and u

$$A \xrightarrow{u} \mathsf{T}\ A \xleftarrow{\mu} \mathsf{T}\ (\mathsf{T}\ A) \tag{F1}$$

que satisfazem as propriedades

satisfying

$$\mu \cdot u = id = \mu \cdot \mathsf{T}\ u \tag{F2}$$
$$\mu \cdot \mu = \mu \cdot \mathsf{T}\ \mu \tag{F3}$$

para além das respectivas propriedades "grátis", onde $\mathsf{T}^2\ f$ abrevia $\mathsf{T}\ (\mathsf{T}\ f)$:

in addition to their "free" properties, where $\mathsf{T}^2\ f$ abbreviates $\mathsf{T}\ (\mathsf{T}\ f)$:

$$\mathsf{T}\ f \cdot u = u \cdot f \tag{F4}$$
$$\mathsf{T}\ f \cdot \mu = \mu \cdot \mathsf{T}^2\ f \tag{F5}$$

Partindo da definição

Starting from the definition of monadic composition,

$$f \bullet g = \mu \cdot \mathsf{T}\ f \cdot g \tag{F6}$$

de *composição monádica*, demonstre os factos seguintes:

prove the following facts:

$$\mu = id \bullet id \tag{F7}$$
$$f \bullet u = f \quad \wedge \quad f = u \bullet f \tag{F8}$$
$$(f \cdot g) \bullet h = f \bullet (\mathsf{T}\ g \cdot h) \tag{F9}$$
$$\mathsf{T}\ f = (u \cdot f) \bullet id \tag{F10}$$

---

2. Demonstre a seguinte propriedade da composição monádica:

   Prove the following property of monadic composition:

$$f \bullet [g\ , h] = [f \bullet g\ , f \bullet h] \tag{F11}$$

---

3. Considere a função

Consider

$$discollect : (A \times B^*)^* \to (A \times B)^*$$
$$discollect = lstr \bullet id$$

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

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

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

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

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

---

4. 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

$$\mathsf{T}\ X = X \times \mathbb{R} \tag{F12}$$

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:

$$\mathsf{T}\ f = f \times id \tag{F13}$$
$$u\ x = (x, 0) \tag{F14}$$
$$\mu\ ((x, t_1), t_2) = (x, t_1 + t_2) \tag{F15}$$

Vê-se bem como μ faz a adição dos tempos de execução. Contudo, para T ser um mónade terá de satisfizer as leis (F3) e (F2). 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 (F3) and (F2). Prove that it so happens.

---

5. 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

$$\begin{cases} \mu = (\!|\, [id\ , \mathsf{in} \cdot i_2]\, |\!) \\ u = \mathsf{in} \cdot i_1 \end{cases} \tag{F16}$$

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

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

---

6. (a) Alguns mónades conhecidos resultam de (F16). 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 (F16)? 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 (F16). 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 (F16)? And in the case where* $\mathsf{F}\ Y = O \times Y^*$*, where the type* O *is considered fixed at the outset?*

---

7. 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:

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

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.*