Cálculo de Programas

2.° ano
Lic. Ciências da Computação e Mestrado Integrado em Engenharia Informática
UNIVERSIDADE DO MINHO

2017/18 - Ficha nr.° 12

1. O facto de $\mathsf{length}\colon A^* \to \mathbb{N}_0$ poder ser definida tanto como *catamorfismo* de listas como *anamorfismo* de naturais (que foi assunto de uma questão de uma ficha anterior) pode generalizar-se da forma seguinte: sejam dados dois tipos indutivos

$$\begin{tikzcd} \mathsf{T}_1 \arrow[r, bend left, "\mathsf{out}_1"] \arrow[r, phantom, "\cong"] & \mathsf{F}\ \mathsf{T}_1 \arrow[l, bend left, "\mathsf{in}_1"] \end{tikzcd} \qquad \begin{tikzcd} \mathsf{T}_2 \arrow[r, bend left, "\mathsf{out}_2"] \arrow[r, phantom, "\cong"] & \mathsf{G}\ \mathsf{T}_2 \arrow[l, bend left, "\mathsf{in}_2"] \end{tikzcd}$$

e $\alpha : \mathsf{F}\ X \to \mathsf{G}\ X$, isto é, α satisfaz a propriedade *grátis*

$$\mathsf{G}\ f \cdot \alpha = \alpha \cdot \mathsf{F}\ f \tag{F1}$$

Então $(\!|\mathsf{in}_2 \cdot \alpha|\!) = [\!(\alpha \cdot \mathsf{out}_1)\!]$, como se mostra a seguir (complete as justificações):

$$\begin{array}{ll} & k = (\!|\mathsf{in}_2 \cdot \alpha|\!) \\ \equiv & \quad \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\ & k \cdot \mathsf{in}_1 = \mathsf{in}_2 \cdot \alpha \cdot \mathsf{F}\ k \\ \equiv & \quad \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\ & \mathsf{out}_2 \cdot k = \mathsf{G}\ k \cdot \alpha \cdot \mathsf{out}_1 \\ \equiv & \quad \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\ & k = [\!(\alpha \cdot \mathsf{out}_1)\!] \\ \Box & \end{array}$$

Identifique T_1, T_2 e α no caso de length.

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

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

que satisfazem (para além das naturais, ie. "grátis") as propriedades

$$\begin{array}{l} \mu \cdot u = id = \mu \cdot \mathsf{T}\ u \\ \mu \cdot \mu = \mu \cdot \mathsf{T}\ \mu \end{array}$$

— identifique-as no formulário — com base nas quais se pode definir a *composição monádica*:

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

(Identifique-a também no formulário.) Demonstre os factos seguintes:

$$\begin{array}{rcll} \mu & = & id \bullet id & \text{(F2)} \\ f \bullet u = f & \wedge & f = u \bullet f & \text{(F3)} \\ (f \cdot g) \bullet h & = & f \bullet (\mathsf{T}\ g \cdot h) & \text{(F4)} \\ \mathsf{T}\ f & = & (u \cdot f) \bullet id & \text{(F5)} \end{array}$$

3. O tipo $A^* = 1 + A$ ("apontador" para A) foi o primeiro exemplo de mónade apresentado nesta disciplina, em que $\mu = [i_1\ , id]$ e $u = i_2$.

   (a) Mostre que tais μ e u satisfazem as duas propriedades (F2) e (F2) que caracterizam um mónade, neste caso:

   $$\mu \cdot u = id = \mu \cdot (id + u)$$
   $$\mu \cdot \mu = \mu \cdot (id + \mu)$$

   (b) O mónade $A^* = 1 + A$ pode ser implementado em Haskell por $Maybe\ a = \mathsf{Nothing} \mid \mathsf{Just}\ a$ pois $Maybe\ a \xleftarrow{inSeq = [\underline{\mathsf{Nothing}}\ , \mathsf{Just}]} 1 + a$ é um isomorfismo, tendo-se:

   $$\begin{tikzcd} Maybe\ (Maybe\ a) \arrow[d, "\mu"'] & 1 + Maybe\ a \arrow[l, "inSeq"'] \arrow[d, "! + id"] \\ Maybe\ a & 1 + Maybe\ a \arrow[l, "{[inSeq \cdot i_1\ , id]}"'] \end{tikzcd} \qquad \mu \cdot inSeq = [inSeq \cdot i_1\ , id] \cdot (! + id)$$

   Derive deste diagrama a seguinte implementação *pointwise* de μ:

   $$\mu\ \mathsf{Nothing} = \mathsf{Nothing}$$
   $$\mu\ (\mathsf{Just}\ a) = a$$

4. A função $discollect : (A \times B^*)^* \to (A \times B)^*$ que apareceu (sem ser definida) numa questão da ficha nº2 não é mais do que

   $$discollect = lstr \bullet id \qquad \text{(F6)}$$

   — onde $lstr\ (a, x) = [(a, b) \mid b \leftarrow x]$ — no mónade das listas, $\mathsf{T}\ A = A^*$,

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

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

5. Em Haskell, um mónade declara-se instanciando a classe $Monad$, onde se define a unidade u (que aí se designa por return) e uma operação $x \gg\!= f$, conhecida como aplicação monádica, ou *"binding"* de f a x, que é tal que

   $$x \gg\!= f = (f \bullet id)\ x = (\mu \cdot \mathsf{T}\ f)\ x \qquad \text{(F7)}$$

   Mostre que:

   $$\mu = (\gg\!= id) \qquad \text{(F8)}$$
   $$g \bullet f = (\gg\!= g) \cdot f \qquad \text{(F9)}$$
   $$x \gg\!= (f \bullet g) = (x \gg\!= g) \gg\!= f \qquad \text{(F10)}$$