

Cálculo de Programas

Algebra of Programming

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

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

- | | |
|---|---|
| 1. Demonstre a seguinte propriedade da composição monádica: | <i>Prove the following property of monadic composition:</i> |
|---|---|

$$f \bullet [g, h] = [f \bullet g, f \bullet h] \quad (\text{F1})$$

2. Considere a função

Consider

$$\begin{aligned} \text{discollect} &: (A \times B^*)^* \rightarrow (A \times B)^* \\ \text{discollect} &= \text{lstr} \bullet \text{id} \end{aligned}$$

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

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

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

Recordando $\text{concat} = \llbracket [\text{nil}, \text{conc}] \rrbracket$ 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 $\text{concat} = \llbracket [\text{nil}, \text{conc}] \rrbracket$ and *cata-absorption* (for lists), derive a recursive definition for *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

$$\mathbb{T} X = X \times \mathbb{R} \quad (\text{F2})$$

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

where each pair (x, t) of $\mathbb{T} X$ 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 $X \xrightarrow{u} \mathbb{T} X \xleftarrow{\mu} \mathbb{T}^2 X$ is defined:

$$\mathbb{T} f = f \times \text{id} \quad (\text{F3})$$

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

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

Vê-se bem como μ faz a adição dos tempos de execução. Contudo, para T ser um mónade terá de satisfazer 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 $T\ X$ cuja base é o bifunctor

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

$$B\ (X, Y) = X + G\ Y$$

$$B\ (f, g) = f + G\ g$$

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

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

$$\begin{cases} \mu = \llbracket [id, in \cdot i_2] \rrbracket \\ u = in \cdot i_1 \end{cases} \quad (F6)$$

onde $in : B\ (X, T\ X) \rightarrow T\ X$.

where $in : B\ (X, T\ X) \rightarrow T\ X$.

5. (a) Alguns mónades conhecidos resultam de (F6). Mostre que é o caso de $LTree$ — identifique G para esse caso; (b) Para $G\ Y = 1$ (i.e. $G\ f = id$) qual é o mónade que se obtém por (F6)? E no caso em que $G\ Y = O \times Y^*$, 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 $G\ Y = 1$ (ie $G\ f = id$) what is the monad obtained by (F6)? And in the case where $G\ Y = O \times Y^$, 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 ($T\ X = X^*$) da função monádica

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

$$sequence : T\ (M\ X) \rightarrow M\ (T\ X)$$

é o catamorfismo

is the catamorphism

$$\begin{aligned} sequence &= \llbracket g \rrbracket \text{ where} \\ g &= [return, id] \cdot (nil + [cons]) \\ [f] \ (x, y) &= \text{do } \{ a \leftarrow x; b \leftarrow y; \text{return } (f\ (a, b)) \} \end{aligned}$$

tal como se mostra neste diagrama:

as in the following diagram:

$$\begin{array}{ccc} (M\ X)^* & \xleftarrow{in} & 1 + M\ X \times (M\ X)^* \\ \downarrow sequence & & \downarrow id + id \times sequence \\ M\ (X^*) & \xleftarrow{g} & 1 + M\ X \times M\ (X^*) \\ & \nwarrow [return, id] & \downarrow nil + [cons] \\ & & X^* + M\ (X^*) \end{array}$$

Partindo da propriedade universal-cata, derive uma versão de `sequence` em Haskell com variáveis que não recorra à composição de funções.

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