Cálculo de Programas

2.º Ano de LEI (Universidade do Minho)
Ano Lectivo de 2010/11

Teste de frequência — 18 de Junho 2011
09h00
Salas 2202, 2203, 2204, 2205

Importante — *Ler antes de iniciar a prova:*

- *Esta prova consta de **10** questões que valem, cada uma, 2 valores. O tempo médio estimado para resolução de cada questão é de 15 min.*
- *Os alunos do **Método A** só devem responder às questões 7, 8, 9 e 10, devendo entregar o teste ao fim de uma hora.*
- *Os alunos do **Método B** devem responder a todas as questões. Destas, as primeiras 6 têm a nota mínima de 6 valores.*

PROVA SEM CONSULTA (2h30m)

Parte 1 — Método B apenas (6 valores de nota mínima)

Questão 1 Mostre que $[! \, , !] = !$. Daí deduza que $\langle [f \, , g], ! \rangle = [\langle f, ! \rangle \, , \langle g, ! \rangle]$.

Questão 2 Sabe-se que uma função polimórfica α satisfaz a propriedade natural ("grátis")

$$((f + g) \times k) \cdot \alpha = \alpha \cdot (k \times (g + f)) \tag{1}$$

Deduza o tipo de α e determine a sua definição sabendo que α é um isomorfismo.

RESOLUÇÃO: Primeiro constrói-se o diagrama de (1), tendo o cuidaddo de associar tipos diferentes às funções f, g e k:

$$\begin{tikzcd}
(E+D) \times A \arrow[d, "(f+g)\times k"'] & A \times (D+E) \arrow[l, "\alpha"'] \arrow[d, "k\times(g+f)"] \\
(E'+D') \times A' & A' \times (D'+E') \arrow[l, "\alpha"]
\end{tikzcd}$$

Conhecido assim o tipo de $\alpha :: A \times (D + E) \to (E + D) \times A$, repara-se que o seu tipo de saída é um produto. Logo faz sentido definir $\alpha = \langle f, g \rangle$ e procurar f e g com base no diagrama desse "split":

$$\begin{tikzcd}
E+D & (E+D) \times A \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & A \\
 & A \times (D+E) \arrow[ul, "f"] \arrow[u, "\langle f, g \rangle"] \arrow[ur, "g"'] &
\end{tikzcd}$$

De imediato se vê $g = \pi_1$. Quanto a f, basta lembrar que $\mathsf{coswap} :: D + E \to E + D$. Daí ter-se $f = \mathsf{coswap} \cdot \pi_2$ e, finalmente, $\alpha = \langle (\mathsf{coswap} \cdot \pi_2), \pi_1 \rangle$. $\square$

Questão 3 Apresente justificações para os passos que se seguem da inferência da definição em Haskell do combinador uncurry a partir da propriedade universal da exponenciação:

$$
\begin{array}{ll}
& k = \overline{f} \;\Leftrightarrow\; ap \cdot (k \times id) = f \\
\equiv & \qquad \{ \text{ Passo A } \} \\
& \widehat{k} = f \;\Leftrightarrow\; ap \cdot (k \times id) = f \\
\equiv & \qquad \{ \text{ Passo B } \} \\
& \widehat{k} = ap \cdot (k \times id) \\
\equiv & \qquad \{ \text{ Passo C } \} \\
& \mathsf{uncurry}\ k\ (a, b) = ap\ ((k \times id)\ (a, b)) \\
\equiv & \qquad \{ \text{ Passo D } \} \\
& \mathsf{uncurry}\ k\ (a, b) = ap\ (k\ a, b) \\
\equiv & \qquad \{ \text{ Passo E } \} \\
& \mathsf{uncurry}\ k\ (a, b) = k\ a\ b
\end{array}
$$

RESOLUÇÃO:

Passo A — sendo 'curry/uncurry' isomorfismos, tem-se $k = \overline{f} \;\Leftrightarrow\; \widehat{k} = f$ — ver (4.32), na página 119 do capítulo *Why Monads Matter*, a este propósito.

Passo B — duas coisas iguais a uma terceira são iguais entre si

Passo C — notação uncurry k para $\widehat{k}$; igualdade extensional (4) ; composição *ao ponto* (70)

Passo D — produto de funções *ao ponto* (76) ; $id\ x = x$

Passo E — aplicação de funções *ao ponto*: $ap\ (f, x) = f\ x$.

□

Questão 4 Um ciclo-for com corpo b e inicialização i, for b i, reduz-se à inicialização i sempre que o corpo "não faz nada" ($b = id$):

$$\mathsf{for}\ id\ i \;=\; \underline{i} \tag{2}$$

Demonstre (2) recordando que

$$\mathsf{for}\ b\ i \;=\; (\![[\underline{i}\ , b]]\!) \tag{3}$$

RESOLUÇÃO: A prova é simples, baseando-se nas propriedades de catamorfismos, alternativas e funções constantes:

$$
\begin{array}{ll}
& \mathsf{for}\ id\ i = \underline{i} \\
\equiv & \quad \{\ (3)\ \text{acima}\ \} \\
& (\![\underline{i}\ , id]\!) = \underline{i} \\
\equiv & \quad \{\ \text{universal-cata (36)}\ \} \\
& \underline{i} \cdot \mathsf{in} = [\underline{i}\ , id] \cdot (id + \underline{i}) \\
\equiv & \quad \{\ \text{função constante: } \underline{k} \cdot f = \underline{k}\ \text{; absorção-+ (20) ; natural-}id\ \text{(1) duas vezes}\ \} \\
& \underline{i} = [\underline{i}\ , \underline{i}] \\
\equiv & \quad \{\ \text{universal-+ (16)}\ \} \\
& \underline{i} \cdot i_1 = \underline{i}\ \wedge\ \underline{i} \cdot i_2 = \underline{i} \\
\equiv & \quad \{\ \text{função constante } (\underline{k} \cdot f = \underline{k})\ \text{duas vezes}\ \} \\
& \underline{i} = \underline{i}\ \wedge\ \underline{i} = \underline{i}
\end{array}
$$

□

Questão 5 Uma sequência numérica diz-se *íngreme* (*steep*) se cada elemento seu é maior que a soma dos seguintes:

$$
\begin{array}{l}
steep\ [\,] = True \\
steep\ (a : x) = a > sum\ x \wedge steep\ x
\end{array}
$$

Por exemplo, $[4, 2, 1]$ é íngreme mas já $[3, 2, 1]$ não o é. Sabendo-se que sum é o catamorfismo $sum = (\![zero\ , add]\!)$, para $zero\ \_ = 0$ e $add\ (x, y) = x + y$, e sendo fácil derivar

$$steep \cdot \mathsf{in} \quad = \quad [true\ , h] \cdot (id + id \times \langle sum, steep \rangle) \tag{4}$$

a partir da definição dada, para $true\ \_ = True$ e $h\ (a, (b, c)) = a > b \wedge c$, mostre que $steep$ se pode implementar da forma mais eficiente

$$steep = \pi_2 \cdot aux\ \mathbf{where}\ aux = (\![\langle zero, true \rangle\ , \langle (add \cdot (id \times \pi_1)), h \rangle]\!)$$

tendo-se ainda que $sum = \pi_1 \cdot aux$.

RESOLUÇÃO: De $sum = \pi_1 \cdot aux$ e $steep = \pi_2 \cdot aux$ deduz-se logo $aux = \langle sum, steep \rangle$ (justifique). Logo vamos poder usar a lei de recursividade múltipla para justificar aux:

$$
\begin{array}{ll}
& \langle sum, steep \rangle = (\![\langle zero, true \rangle\ , \langle (add \cdot (id \times \pi_1)), h \rangle]\!) \\
\equiv & \quad \{\ \text{lei da troca (27)}\ \} \\
& \langle sum, steep \rangle = (\!\langle [zero\ , add \cdot (id \times \pi_1)], [true\ , h] \rangle\!) \\
\equiv & \quad \{\ \text{recursividade múltipla (42)}\ \} \\
& \left\{ \begin{array}{l} sum \cdot \mathsf{in} = [zero\ , add \cdot (id \times \pi_1)] \cdot (id + id \times \langle sum, steep \rangle) \\ steep \cdot \mathsf{in} = [true\ , h] \cdot (id + id \times \langle sum, steep \rangle) \end{array} \right. \\
\equiv & \quad \{\ (4)\ \text{acima}\ \} \\
& sum \cdot \mathsf{in} = [zero\ , add \cdot (id \times \pi_1)] \cdot (id + id \times \langle sum, steep \rangle) \\
\equiv & \quad \{\ \text{exercício: justifique detalhadamente este passo}\ \}
\end{array}
$$

$$
\begin{aligned}
& \quad sum \cdot \mathsf{in} = [zero\ , add \cdot (id \times sum)] \\
\equiv & \qquad \{\ \text{universal-cata (36) entre outras leis — identifique quais}\ \} \\
& \quad sum = (\!|[zero\ , add]|\!) \\
\equiv & \qquad \{\ \text{enunciado}\ \} \\
& \quad \mathrm{TRUE}
\end{aligned}
$$

□

Questão 6 Eratóstenes de Cirene (c.276 AC-c.195 AC) foi um bibliotecário de Alexandria que, para além de ter conseguido medir o raio da Terra com uma precisão extraordinária para a altura, ficou célebre pelo seu algoritmo para determinar os números primos até n — o "*crivo* ($sieve$) de Eratóstenes" — algoritmo esse que se escreve em Haskell da forma seguinte:

$$
\begin{array}{l}
primes\ n = sieve\ [2\mathinner{.\,.}n] \\
\quad \textbf{where}\ sieve\ [\,] = [\,] \\
\qquad sieve\ (p : xs) = p : sieve\ [x \mid x \leftarrow xs, x\ \text{`}rem\text{`}\ p \not\equiv 0]
\end{array}
$$

Caracterize a função $sieve$ como o hilomorfismo $sieve = [\![conquer, divide]\!]$, determinando os respectivos genes $divide$ e $conquer$ e fazendo um diagrama ilustrativo.

RESOLUÇÃO: Do código dado infere-se (para listas)

$$
\begin{array}{l}
sieve \cdot nil = nil \\
sieve \cdot cons = cons \cdot (id \times sieve) \cdot aux
\end{array}
$$

onde

$$aux\ (p, xs) = (p, [x \mid x \leftarrow xs, x\ \text{`}rem\text{`}\ p \not\equiv 0]).$$

Por igualdade estrutural (26) as duas cláusulas juntas dão

$$
\begin{aligned}
& \quad sieve \cdot \mathsf{in} = \mathsf{in} \cdot (id + id \times sieve) \cdot (id + aux) \\
\equiv & \qquad \{\ \mathsf{out\ é\ o\ converso\ de\ in}\ \} \\
& \quad sieve = \mathsf{in} \cdot (id + id \times sieve) \cdot (id + aux) \cdot \mathsf{out} \\
\Leftarrow & \qquad \{\ \text{hilomorfismo } [\![g, h]\!] \text{ é solução da equação } x = g \cdot (\mathsf{F}\ x) \cdot h\ \} \\
& \quad sieve = [\![\mathsf{in}, ((id + aux) \cdot \mathsf{out})]\!]
\end{aligned}
$$

Logo $sieve$ pode ser vista como um hilomorfismo de listas em que $divide = (id + aux) \cdot \mathsf{out}$ e $conquer = \mathsf{in}$ (um anamorfismo, portanto). O diagrama correspondente

$$
\begin{tikzcd}
{[\mathbb{N}_0]} \arrow[r, "\mathsf{out}"] \arrow[rr, bend left, "divide"] \arrow[d, "\lbrack\!(divide)\!\rbrack"'] & 1 + \mathbb{N}_0 \times [\mathbb{N}_0] \arrow[r, "id+aux"] & 1 + \mathbb{N}_0 \times [\mathbb{N}_0] \arrow[d, "id+id\times\lbrack\!(divide)\!\rbrack"] \\
{[\mathbb{N}_0]} \arrow[rr, bend left, "\mathsf{out}"] \arrow[d, "(\!|\mathsf{in}|\!)"'] & \cong & 1 + \mathbb{N}_0 \times [\mathbb{N}_0] \arrow[ll, bend left, "\mathsf{in}"] \arrow[d, "id+id\times(\!|\mathsf{in}|\!)"] \\
{[\mathbb{N}_0]} & & 1 + \mathbb{N}_0 \times [\mathbb{N}_0] \arrow[ll, bend left, "conquer=\mathsf{in}"]
\end{tikzcd}
$$

mostra que neste hilomorfismo só estão listas em jogo: na entrada, no tipo intermédio e no tipo de saída: □

Parte 2 — Métodos A + B

Questão 7 Queremos contar quantas folhas de uma árvore de tipo $\mathsf{LTree}\ a$ satisfazem um dado predicado $p :: a \rightarrow Bool$. Para isso substitui-se cada folha por 1 ou 0, conforme satisfaz p ou não, e soma-se a árvore assim obtida,

$$hwmny\ p \quad = \quad sum \cdot \mathsf{LTree}\ (p \rightarrow \underline{1}, \underline{0}) \tag{5}$$

em que $sum = (\![[id\ , add]]\!)$ e $add\ (a, b) = a + b$. Mostre que $hwmny$ mais não é que a função que se segue, em Haskell:

$hwmny :: (Num\ n) \Rightarrow (a \rightarrow Bool) \rightarrow \mathsf{LTree}\ a \rightarrow n$
$hwmny\ p\ (Leaf\ a) \mid p\ a = 1 \mid otherwise = 0$
$hwmny\ p\ (Fork\ (x, y)) = (hwmny\ p\ x) + (hwmny\ p\ y)$

Questão 8 Muitas funções, como por exemplo $\mathsf{length} :: [\,a\,] \rightarrow \mathbb{N}_0$, podem definir-se simultaneamente como catamorfismos do seu tipo de entrada,

$$\mathsf{length} = (\![[zero\ , \mathsf{succ} \cdot \pi_2]]\!)$$

(onde $zero\ \_ = 0$ e $\mathsf{succ}\ n = n + 1$) e como anamorfismos do seu tipo de saída,

$$\mathsf{length} = [\!((id + \pi_2) \cdot \mathsf{out})\!]$$

(onde, para listas, out designa o isomorfismo converso de **in**). Mostre que, de facto

$$\begin{array}{ll} & \mathsf{length} = (\![[zero\ , \mathsf{succ} \cdot \pi_2]]\!) \\ \equiv & \quad \{ \text{ ... vários passos recorrendo, entre outras, às propiedades universal-cata e universal-ana ... } \} \\ & \mathsf{length} = [\!((id + \pi_2) \cdot \mathsf{out})\!] \end{array}$$

Questão 9 Pretende-se mostrar que a função seguinte, extraída do *Prelude* do Haskell,

$concat :: [[\,a\,]] \rightarrow [\,a\,]$
$concat = foldr\ (+\!\!+)\ [\,]$

é a **multipicação** do mónade das listas. Para isso, vai ser preciso provar, entre outras, a propriedade

$$concat \cdot concat = concat \cdot (\mathsf{map}\ concat) \tag{6}$$

Converta $concat$ num catamorfismo e demonstre (6) com base nas leis de catamorfismos que conhece, assumindo ainda que

$$concat\ (x +\!\!+ y) = (concat\ x) +\!\!+ (concat\ y) \tag{7}$$

se verifica.

Questão 10 Defina a função

$$mfor :: (Monad\ m, Integral\ t) \Rightarrow (a \rightarrow m\ a) \rightarrow a \rightarrow t \rightarrow m\ a$$

como resultado da "monadificação" do combinador ciclo-for sobre naturais,

$$\begin{array}{l} \mathsf{for}\ b\ i\ 0 = i \\ \mathsf{for}\ b\ i\ (n+1) = b\ (\mathsf{for}\ b\ i\ n) \end{array}$$

explicitando os passos da construção dessa função monádica.
