Cálculo de Programas

2.º Ano de LCC+MiEI (Universidade do Minho)
Ano Lectivo de 2018/19

Exame da época especial — 24 de Julho de 2019
14h00–16h00
E1-2.17/2.18/2.19

- *Este teste consta de **8** questões que valem, cada uma, 2.5 valores. O tempo médio estimado para resolução de cada questão é de 15 min.*
- *Os alunos devem ler a prova antes de decidirem por que ordem responder às questões colocadas.*

PROVA SEM CONSULTA (2h)

Questão 1 Determine o tipo mais geral da função $\alpha = \langle i_1, \pi_1 \rangle$ e, a partir dele, a propriedade grátis de α. Justifique convenientemente a sua resposta.

Questão 2 Demonstre a 2ª lei de fusão do condicional de McCarthy,

$$(p \to f, g) \cdot h \quad = \quad (p \cdot h) \to (f \cdot h), (g \cdot h)$$

Questão 3 Uma das primeiras linguagens de programação funcional foi o LISP (1958). Em LISP há apenas um único suporte para representação de dados, designado por *expressão-S* — abreviatura de "expressão simbólica". Uma *expressão-S* ou é uma valor atómico ou é uma sequência (possivelmente vazia) de *expressões-S*. Considera-se um *átomo* toda a unidade de informação indivisível, não-estruturada (*i.é* "atómica").

Por exemplo, são átomos os inteiros e os "strings" alfanuméricos, *e.g.* 10, -5, $a12$, bca. Dão-se a seguir exemplos de *expressões-S* não atómicas, escritas na própria sintaxe concreta do LISP:

$()$
(1)
$(1\ um\ 2\ dois)$
$(1\ (2\ (3\ (4))))$

Seja

$$\mathbf{data}\ SExp\ a = At\ a \mid Ex\ [SExp\ a]$$

a declaração de um tipo de dados em Haskell para descrever *expressões-S*.

Desenhe o diagrama dos catamorfismos deste tipo e exprima a operação que conta o número de átomos presentes numa *expressão-S* como um desses catamorfismos.

Questão 4 Considere a função que realiza a partição de uma lista em duas outras listas que recolhem, respectivamente, os elementos que verificam e os elementos que não verificam determinado predicado p:

$$\mathsf{partition}\ p = \langle \mathsf{filter}\ p, \mathsf{filter}\ (\neg \cdot p) \rangle \tag{E1}$$

Mostre que partition $p = (\!|g\ p|\!)$, determinando $g\ p$. **Sugestão**: comece por descrever filter p como um catamorfismo, por forma a poder depois usar a lei de "banana-split". **NB**: não se pede para calcular a versão *pointwise* the partition p.

Questão 5 Utilizando a lei de fusão-cata, a propriedade da comutatividade da soma (que em notação *pointfree* pode ser expressa por $\mathsf{add} \cdot \mathsf{swap} = \mathsf{add}$) e outras do cálculo estudado nesta disciplina, demonstre o facto

$$\mathsf{nfolhas} \cdot \mathsf{mirror} = \mathsf{nfolhas}$$

onde

$$\mathsf{nfolhas} = (\![\underline{1}, \mathsf{add}]\!) \tag{E2}$$

$$\mathsf{mirror} = (\!|\mathsf{in} \cdot (id + \mathsf{swap})|\!) \tag{E3}$$

são catamorfismos do tipo LTree.

Questão 6 A lei de recursividade mútua tem uma versão dual que envolve alternativas e anamorfismos em vez de *splits* e catamorfismos:

$$\left\{ \begin{array}{l} f = \mathsf{in} \cdot \mathsf{F}\ [f\,, g] \cdot h \\ g = \mathsf{in} \cdot \mathsf{F}\ [f\,, g] \cdot k \end{array} \right. \equiv [f\,, g] = [\!(\,[h\,, k]\,)\!] \tag{E4}$$

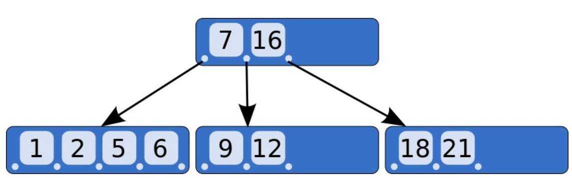
Complete a demonstração de (E6) que se segue:

$$\begin{array}{ll}
 & [f\,, g] = [\!(\,[h\,, k]\,)\!] \\
\equiv & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
 & \mathsf{out} \cdot [f\,, g] = \mathsf{F}\ [f\,, g] \cdot [h\,, k] \\
\equiv & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
 & \vdots \\
\Box &
\end{array}$$

Questão 7 Uma "B-tree" é uma generalização das árvores binárias do módulo BTree a mais do que duas sub-árvores por nó:

$$\mathbf{data}\ \mathsf{B\_tree}\ a = Nil \mid Block\ \{ leftmost :: \mathsf{B\_tree}\ a, block :: [(a, \mathsf{B\_tree}\ a)] \}$$

Por exemplo, a B-tree[1]



[1] Créditos: figura extraída de `https://en.wikipedia.org/wiki/B-tree`.

é representada no tipo acima por:

$$
\begin{array}{l}
t = \mathit{Block}\ \{ \\
\quad \mathit{leftmost} = \mathit{Block}\ \{ \\
\quad\quad \mathit{leftmost} = \mathit{Nil}, \\
\quad\quad \mathit{block} = [(1, \mathit{Nil}), (2, \mathit{Nil}), (5, \mathit{Nil}), (6, \mathit{Nil})]\}, \\
\quad \mathit{block} = [\\
\quad\quad (7, \mathit{Block}\ \{ \\
\quad\quad\quad \mathit{leftmost} = \mathit{Nil}, \\
\quad\quad\quad \mathit{block} = [(9, \mathit{Nil}), (12, \mathit{Nil})]\}), \\
\quad\quad (16, \mathit{Block}\ \{ \\
\quad\quad\quad \mathit{leftmost} = \mathit{Nil}, \\
\quad\quad\quad \mathit{block} = [(18, \mathit{Nil}), (21, \mathit{Nil})]\}) \\
\quad\quad]\}
\end{array}
$$

Identifique, justificando, o functor de base

$$
\left\{
\begin{array}{l}
\mathsf{B}\ (X, Y) = \ldots \\
\mathsf{B}\ (f, g) = ...
\end{array}
\right.
$$

que capta o padrão de recursividade da declaração de $\mathsf{B\_tree}$ dada acima, em Haskell, bem como o isomorfismo:

$$\mathsf{in} : \mathsf{B}\ (A, \mathsf{B\_tree}\ A) \rightarrow \mathsf{B\_tree}\ A.$$

Questão 8 Em Haskell, a instância para listas da função monádica $\mathsf{sequence} :: (\mathit{Monad}\ \ m, \mathit{Traversable}\ \ t) \Rightarrow t\ (m\ \ a) \rightarrow m\ (t\ \ a)$ é o catamorfismo

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

tal como se mostra neste diagrama:

$$
\begin{tikzcd}
(\mathsf{T}\ X)^* \arrow[d, "\mathsf{sequence}"'] & 1 + \mathsf{T}\ X \times (\mathsf{T}\ X)^* \arrow[l, "\mathsf{in}"'] \arrow[d, "id + id \times \mathsf{sequence}"] \\
\mathsf{T}\ (X^*) & 1 + \mathsf{T}\ X \times \mathsf{T}\ (X^*) \arrow[l, dotted, "g"'] \arrow[d, "\mathsf{nil} + \mathsf{cons}^\flat"] \\
& X^* + \mathsf{T}\ (X^*) \arrow[ul, dotted, "{[\mathsf{return}\,, id]}"]
\end{tikzcd}
$$

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

ANEXO — Catálogo de alguns tipos de dados estudados na disciplina.

1. Números naturais:

$$\mathsf{T} = \mathbb{N}_0 \qquad \begin{cases} \mathsf{F}\ X = 1 + X \\ \mathsf{F}\ f = id + f \end{cases} \qquad \mathsf{in} = [\underline{0}\,, \mathsf{succ}] \tag{E5}$$

   Haskell: Int inclui $\mathbb{N}_0$.

2. Listas de elementos em A:

$$\mathsf{T} = A^* \qquad \begin{cases} \mathsf{F}\ X = 1 + A \times X \\ \mathsf{F}\ f = id + id \times f \end{cases} \qquad \mathsf{in} = [\mathsf{nil}\,, \mathsf{cons}] \tag{E6}$$

   Haskell: $[\,a\,]$.

3. Árvores com informação de tipo A nos nós:

$$\mathsf{T} = \mathsf{BTree}\ A \qquad \begin{cases} \mathsf{F}\ X = 1 + A \times X^2 \\ \mathsf{F}\ f = id + id \times f^2 \end{cases} \qquad \mathsf{in} = [\underline{Empty}\,, Node] \tag{E7}$$

   Haskell: $\mathbf{data}\ \mathsf{BTree}\ a = Empty \mid Node\ (a, (\mathsf{BTree}\ a, \mathsf{BTree}\ a))$.

4. Árvores com informação de tipo A nas folhas:

$$\mathsf{T} = \mathsf{LTree}\ A \qquad \begin{cases} \mathsf{F}\ X = A + X^2 \\ \mathsf{F}\ f = id + f^2 \end{cases} \qquad \mathsf{in} = [Leaf\,, Fork] \tag{E8}$$

   Haskell: $\mathbf{data}\ \mathsf{LTree}\ a = Leaf\ \ a \mid Fork\ (\mathsf{LTree}\ a, \mathsf{LTree}\ a)$.

5. Árvores com informação nos nós e nas folhas:

$$\mathsf{T} = \mathsf{FTree}\ B\ \ A \qquad \begin{cases} \mathsf{F}\ X = B + A \times X^2 \\ \mathsf{F}\ f = id + id \times f^2 \end{cases} \qquad \mathsf{in} = [\,Unit\,, Comp] \tag{E9}$$

   Haskell: $\mathbf{data}\ \mathsf{FTree}\ b\ \ a = Unit\ \ b \mid Comp\ (a, (\mathsf{FTree}\ b\ \ a, \mathsf{FTree}\ b\ \ a))$.

6. Árvores de expressão:

$$\mathsf{T} = Expr\ \ V\ \ O \qquad \begin{cases} \mathsf{F}\ X = V + O \times X^* \\ \mathsf{F}\ f = id + id \times \mathsf{map}\ f \end{cases} \qquad \mathsf{in} = [\,Var\,, Op] \tag{E10}$$

   Haskell: $\mathbf{data}\ Expr\ \ v\ \ o = Var\ \ v \mid Op\ (o, [\,Expr\ \ v\ \ o\,])$