Cálculo de Programas

2.º Ano da LCC (8504N1)
Ano Lectivo de 2006/07

Exame (1.ª chamada da época normal) — 26 de Junho 2007
14h00
Salas 2303, 2304

NB: *Esta prova consta de* ***8*** *alíneas que valem, cada uma, 2.5 valores. Utilize folhas de resposta diferentes para cada grupo.*

PROVA SEM CONSULTA (2 horas)

GRUPO I

Questão 1 Sintetize, justificando, o isomorfismo i em

$$\begin{tikzcd} (A+B)^2 \arrow[r, bend left] \arrow[r, phantom, "\cong"] & (A^2+B^2)+2\times(A\times B) \arrow[l, bend left, "i"] \end{tikzcd} \qquad (1)$$

onde X^2 abrevia $X \times X$. **Sugestão**: Recorde os isomorfismos seguintes:

- $2 \times A \cong A + A$
- $A \times (B \times C) \cong (A \times B) \times C$
- $A \times (B + C) \cong (A \times B) + (A \times C)$

Questão 2 Demonstre a seguinte propriedade do combinador condicional de McCarthy

$$\langle f, (p \rightarrow g\ ,\ h)\rangle \quad = \quad p \rightarrow \langle f, g\rangle\ ,\ \langle f, h\rangle \qquad (2)$$

sabendo que

$$p \rightarrow id\ ,\ id \quad = \quad id \qquad (3)$$

se verifica.

Questão 3 Faça o diagrama de um hilomorfismo com as seguintes características:

- a sua estrutura de dados virtual (intermédia) é a de uma árvore com folhas (`LTree`)
- o catamorfismo que ele inclui soma as folhas do seu argumento
- o anamorfismo que ele inclui é o mesmo do algoritmo "merge sort".

Explicite os genes do hilomorfismo que desenhou e diga, sumariamente, que função é que ele implementa.

GRUPO II

Questão 4 Na biblioteca de listas fornecida no material pedagógico desta disciplina é o dado como exemplo de hilomorfismo a função que calcula n^2 somando os n primeiros ímpares consecutivos, $sq = [\![\, summing, odds \,]\!]$, em que $summing = [\underline{0}\ , (+)]$ e $odds$ é a função

$$\begin{array}{rcl} odds\,0 & = & i_1\,\textsc{Nil} \\ odds(n+1) & = & i_2(2n+1, n) \end{array}$$

Mostre que a transformada-PF desta última função é $odds \cdot in = ! + \langle f, id\rangle$, onde $in = [\underline{0}\ , succ]$ é o isomorfismo de construção de números naturais e $f\ n = 2n + 1$.

Questão 5 Como a adição de inteiros é comutativa ($x + y = y + x$), se se somarem todos os inteiros de uma árvore binária de tipo `LTree` usando o catamorfismo $add = (\![[id\ , (+)]]\!)$ obter-se-á o mesmo resultado que se se somarem os inteiros da mesma árvore "espelhada" por $mirror = (\![in \cdot (id + swap)]\!)$, isto é, verifica-se a propriedade:

$$add \cdot mirror = add$$

Complete a seguinte prova desse facto:

$add \cdot mirror = add$

$\equiv$ { À esq. de = : ....
À dir. de = : .... }

$add \cdot (\![in \cdot (id + swap)]\!) = (\![[id\ , (+)]]\!)$

$\Leftarrow$ { .... }

$add \cdot in \cdot (id + swap) = [id\ , (+)] \cdot (id + add \times add)$

$\equiv$ { À esq. de = : ....
À dir. de = : .... }

$[id\ , (+)] \cdot (id + add \times add) \cdot (id + swap) = [id\ , (+) \cdot (add \times add)]$

$\equiv$ { À esq. de = : ....
À dir. de = : .... }

$[id\ , (+)] \cdot (id + swap \cdot (add \times add)) = [id\ , (+) \cdot (add \times add)]$

$\equiv$ { À esq. de = : ....
À dir. de = : .... }

$[id\ , (+) \cdot swap \cdot (add \times add)] = [id\ , (+) \cdot (add \times add)]$

$\equiv$ { À esq. de = : ....
À dir. de = : .... }

$[id\ , (+) \cdot (add \times add)] = [id\ , (+) \cdot (add \times add)]$

$\equiv$ { .... }

$\textsc{True}$

Questão 6 As estruturas de dados recursivas (vulg. árvores) que linguagens como o Haskell admitem são traduzidas para estruturas em memória RAM usando *heaps*. Um *heap* é um *array* asssociativo: em cada célula de memória que ocupa, associa a um endereço um nó da estrutura de dados que está a armazenar, expressa em termos de endereços (vulg. *apontadores*). Assim, basta ter um endereço e um *heap* para ser possível reconstituir a árvore que ele representa, navegando de apontador em apontador.

Por exemplo, um *heap* para árvores

```
data LTree a = Leaf a | Fork (LTree a, LTree a)
```

poderá ser de tipo

```
data Heap a k = Heap [(k,(Either a (k, k)))] k
```

se se representar por uma lista de pares a associação entre apontadores (k) e nós ($a + k \times k$) envolvendo apontadores. Assim, a árvore

```
t = Fork(Leaf "azul",Fork(Leaf "verde",Leaf "amarelo"))
```

por exemplo, poderá representada pelo *heap*

```
h = Heap [(1,Right (2,3)),
          (2,Left "azul"),
          (3,Right (5,7)),
          (5,Left "verde"),
          (7,Left "amarelo")]
         1
```

Assumindo o tipo de dados `Heap` instanciado para a classe `BiFunctor` da forma que se segue,

```
instance BiFunctor Heap
         where bmap g f (Heap h k') =
                   Heap [ (f k) |-> (g -|- (f >< f)) x | (k,x) <- h ] (f k')
```

onde se usa a abreviatura infixa

```
x |-> y = (x,y)
```

defina o catamorfismo de tipo $LTree$ que converte árvores em heaps, de acordo com o plano seguinte: as folhas são transformadas em *heaps* singulares, com a referência 1, por exemplo `Leaf 5` é convertida em `Heap [(1 |-> Left 5)] 1`; pares de árvores deverão ser convertidas cada uma no seu *heap*, sendo depois todas as referências do da esquerda multiplicadas por 2, e todas as do da direita multiplicadas por 2 e adicionadas de uma unidade, por forma a que possam ser juntos esses dois *heaps* num só, por concatenação.

GRUPO III

Questão 7 Considere a definição do operador

$$str\ a\ t \stackrel{\rm def}{=} do\ \{\ b \leftarrow t\ ;\ return(a, b)\ \} \tag{4}$$

no contexto de uma mónade arbitrária. Qual o tipo de str? E o que faz esta função? Justifique a sua resposta indicando exemplos da sua aplicação a habitantes dos tipos monádicos $[a]$ e $Maybe\ a$.

Questão 8 Nesta disciplina foram estudadas três mónades, a mónade $Maybe$, a mónade das listas e a mónade de estado. A mónade das listas definiu-se fazendo $\mu = concat = (\![\underline{[\,]}\ , cat]\!)$ — onde $cat(x, y) = x + y$ é a função que concatena duas listas — e $u = singl$ (a função que constrói listas singulares).

É sabido que, para o par (μ, u) ser um mónade é necessário que determinadas propriedades se verifiquem, entre elas a da *unidade* que, no caso das listas, inclui a verificação de

$$concat \cdot (map\ singl) = id \tag{5}$$

Demonstre esta igualdade por reflexão e fusão-cata.
Importante: é sabido que, em Haskell, $[a] + l = a : l$, o que, sem variáveis, é a propriedade

$$cat \cdot (singl \times id) = cons \tag{6}$$

onde $cons(a, l) = a : l$. Pode recorrer a esta propriedade na sua demonstração.
