Cálculo de Programas / Métodos de Programação I

2.º Ano de LCC (8504N1) / LESI (5303O7)
Ano Lectivo de 2007/08

Exame de recurso — 14 de Julho 2008
14h30
Salas 1303, 1304

PROVA SEM CONSULTA (2 horas)

NB*: Deverá resolver 8 das alíneas desta prova, de acordo com as seguintes instruções:*

- *Das questões* **1** *e* **2** *resolva apenas* **uma***, à sua escolha*
- *Das questões* **3** *e* **4** *resolva apenas* **uma***, à sua escolha*
- *Das questões* **7***,* **8** *e* **9** *resolva* **duas***, à sua escolha.*

As restantes **4** *alíneas (grupo II) são todas obrigatórias.*

GRUPO I

Questão 1 Considere o tipo seguinte que define árvores binárias completas (isto é, com folhas — um misto de $BTree$ e de $LTree$):

$$\mathbf{data}\ FTree\ a\ c = Unit\ c \mid Comp\ a\ (FTree\ a\ c, FTree\ a\ c)$$

Usando o algoritmo de Hindley-Milner para inferência de tipos polimórficos estudado nas aulas práticas, deduza o tipo principal (ie. mais geral) da função

$$mapFTree\ f\ g\ (Unit\ c) = Unit\ (g\ c)$$
$$mapFTree\ f\ g\ (Comp\ a\ (l, r)) = Comp\ (f\ a)\ (mapFTree\ f\ g\ l, mapFTree\ f\ g\ r)$$

Sugestão: comece por abreviar $mapFTree\ f\ g$ em k, infira o tipo de k e deduza o de $mapFTree$ a partir deste.

Questão 2 Demonstre a seguinte igualdade:

$$[\langle id, i_1 \cdot ! \rangle, (id \times i_2) \cdot swap] \quad = \quad \langle [id, \pi_2], ! + \pi_1 \rangle$$

Qual o isomorfismo que esta função estabelece? Justifique através de um diagrama ilustrativo.

Questão 3 Quantos quadrados se podem desenhar numa folha de papel quadriculado com $n \times n$ quadrículas? A resposta é dada pela função $nq\ n = \sum_{i=1,n} i^2$ isto é, em Haskell,

$$nq\ 0 = 0$$
$$nq\ (n + 1) = (n + 1) \uparrow 2 + nq\ n$$

É fácil de ver que nq é bastante ineficiente, pois cada iteração sua envolve o hilomorfismo sq. Uma hipótese para melhorarmos a sua eficiência é inventarmos a função $bnm\ n \stackrel{\mathrm{def}}{=} (n + 1) \uparrow 2$ e calcularmos para esta última as cláusulas (óbvias)

$$bnm\ 0 = 1$$
$$bnm\ (n + 1) = 2 * n + 3 + bnm\ n$$

na esperança de podermos combinar nq e bnm segundo a lei de recursividade mútua.

Contudo, o mesmo problema recorre em bnm, que agora depende do termo $2 * n + 3$. Temos pois que repetir o processo e inventar $lin\ n = 2 * n + 3$, a que correspondem as cláusulas

$$
\begin{aligned}
&lin\ 0 = 3\\
&lin\ (n+1) = 2 + lin\ n
\end{aligned}
$$

Redefinindo

$$
\begin{aligned}
&bnm'\ 0 = 1\\
&bnm'\ (n+1) = lin\ n + bnm'\ n\\
&nq'\ 0 = 0\\
&nq'\ (n+1) = bnm'\ n + nq'\ n
\end{aligned}
$$

ficamos assim com três funções — nq', bnm' e lin — mutuamente recursivas sobre a base polinomial $\mathsf{F}\ g = id + g$ dos naturais.

Mostre que, por aplicação da referida lei estendida a três funções,

$$
\left\{\begin{array}{l}
f \cdot in = h \cdot \mathsf{F}\ \langle f, \langle g, j\rangle\rangle\\
g \cdot in = k \cdot \mathsf{F}\ \langle f, \langle g, j\rangle\rangle\\
j \cdot in = l \cdot \mathsf{F}\ \langle f, \langle g, j\rangle\rangle
\end{array}\right.
\quad\equiv\quad \langle f, \langle g, j\rangle\rangle = (\!|\langle h, \langle k, l\rangle\rangle|\!) \tag{1}
$$

se obtém a versão linear de nq que se segue:

$$
\begin{aligned}
&nq''\ n = \mathbf{let}\ (a,b,c) = aux\ n\ \mathbf{in}\ a\\
&\quad \mathbf{where}\\
&\qquad aux\ 0 = (0,1,3)\\
&\qquad aux\ (n+1) = \mathbf{let}\ (a,b,c) = aux\ n\ \mathbf{in}\ (a+b, b+c, 2+c)
\end{aligned}
$$

Questão 4 Demonstre que a função que separa listas de pares em pares de listas,

$$
\begin{aligned}
&unzip :: [(a,b)] \to ([a],[b])\\
&unzip\ [\,] = ([\,],[\,])\\
&unzip\ ((x,y):t) = \mathbf{let}\ (l,r) = unzip\ t\ \mathbf{in}\ (x:l, y:r)
\end{aligned}
$$

pode ser definida em notação *pointfree* como $unzip = \langle map\ \pi_1, map\ \pi_2\rangle$. **Sugestão:** recorra à lei de "banana-split".

GRUPO II

Questão 5 A função nq da questão 3 é um exemplo de *paramorfismo* de naturais. Paramorfismos são variantes de catamorfismos cujos genes precisam de um parâmetro adicional para terem acesso, para além do resultado da chamada recursiva, ao respectivo argumento.

No caso geral, dado um tipo indutivo $T \cong \mathsf{F}\,T$, o paramorfismo de g relativamente ao functor F, designado por $\langle\!|g|\!\rangle$ é tal que

$$
\begin{tikzcd}
T \arrow[d, "\langle\!|\beta|\!\rangle"'] & \mathsf{F}\,T \arrow[l, "in"'] \arrow[d, "\mathsf{F}\ \langle id, \langle\!|\beta|\!\rangle\rangle"] \\
C & \mathsf{F}\,(T \times C) \arrow[l, "\beta"]
\end{tikzcd} \tag{2}
$$

a que corresponde a seguinte propriedade universal:

$$
k = \langle\!|\beta|\!\rangle \quad\Leftrightarrow\quad k \cdot in = \beta \cdot \mathsf{F}\ \langle id, k\rangle \tag{3}
$$

1. Complete a seguinte prova da lei de *fusão-para*, desenhando **também** o respectivo diagrama:

$$
\begin{array}{lll}
f \cdot \langle\!|\alpha|\!\rangle = \langle\!|\beta|\!\rangle & \Leftrightarrow & f \cdot \langle\!|\alpha|\!\rangle \cdot in = \beta \cdot \mathsf{F}\ \langle id, f \cdot \langle\!|\alpha|\!\rangle\rangle\\
& \Leftrightarrow & \qquad \{\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ \}\\
& & f \cdot (\langle\!|\alpha|\!\rangle \cdot in) = \beta \cdot \mathsf{F}\,(id \times f) \cdot (\mathsf{F}\ \langle id, \langle\!|\alpha|\!\rangle\rangle)\\
& \Leftrightarrow & \qquad \{\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ \}\\
& & f \cdot \alpha \cdot \mathsf{F}\ \langle id, \langle\!|\alpha|\!\rangle\rangle = \beta \cdot \mathsf{F}\,(id \times f) \cdot (\mathsf{F}\ \langle id, \langle\!|\alpha|\!\rangle\rangle)\\
& \Leftarrow & \qquad \{\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ \}\\
& & f \cdot \alpha = \beta \cdot \mathsf{F}\,(id \times f)
\end{array}
$$

2. Mostre que a seguinte função que calcula o número de palavras num *string*,

$$nw :: (Num\ a) \Rightarrow String \rightarrow a$$
$$nw\ [\,] = 0$$
$$nw\ (c : l) = \mathbf{if}\ \neg\ (sep\ c) \wedge lookahead\_sep\ l\ \mathbf{then}\ nw\ l + 1\ \mathbf{else}\ nw\ l$$

onde

$$lookahead\_sep\ [\,] = True$$
$$lookahead\_sep\ (c : l) = sep\ c$$

e

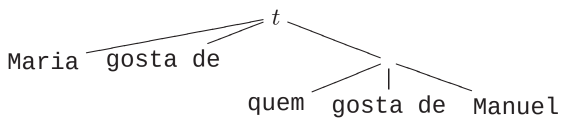
$$sep\ c = (c \equiv \texttt{' '} \vee c \equiv \texttt{'\textbackslash n'} \vee c \equiv \texttt{'\textbackslash t'})$$

é um paramorfismo de listas, instanciando para ela o diagrama (2), com identificação do gene β.

Questão 6 Considere o tipo de dados seguinte, definido em Haskell:

$$\mathbf{data}\ Nest\ a = Tip\ a \mid Nest\ [Nest\ a]\ \mathbf{deriving}\ Show$$

que nos permite definir árvores com informação aninhada, como por exemplo



cf.

$$t = Nest\ [Tip\ \texttt{"Maria"}, Tip\ \texttt{"gosta de"}, Nest\ [Tip\ \texttt{"quem"}, Tip\ \texttt{"gosta de"}, Tip\ \texttt{"Manuel"}]]$$

1. Defina as funções $inNest$, $outNest$, $baseNest$ e $cataNest$ que fazem parte da biblioteca a construir à volta do tipo $Nest$ e use-as para completar a seguinte declaração desse tipo como instância da classe $Functor$:

   $$\mathbf{instance}\ Functor\ Nest\ \mathbf{where}\ fmap\ g = cataNest\ (......................................)$$

2. Considere o catamorfismo deste tipo de dados

   $$invNest = cataNest\ (inNest \cdot (id + reverse))$$

   (onde $reverse$ é a função que conhece, importada de $GHC.List$) que inverte uma árvore de tipo $Nest$, por exemplo convertendo t dada acima em

   $$t'' = Nest\ [Nest\ [Tip\ \texttt{"Manuel"}, Tip\ \texttt{"gosta de"}, Tip\ \texttt{"quem"}], Tip\ \texttt{"gosta de"}, Tip\ \texttt{"Maria"}]$$

   Apresente justificações e complete a seguinte prova (apenas iniciada) de que $invNest$ é inversa de si própria, onde se retira o sufixo $Nest$ dos nomes das funções para melhor leitura:

   $$\begin{array}{ll} & inv \cdot inv = id \\ \equiv & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\ & inv \cdot (\!|in \cdot (id + reverse)|\!) = (\!|in|\!) \\ \Leftarrow & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\ & inv \cdot in \cdot (id + reverse) = in \cdot (\mathsf{F}\, inv) \\ \equiv & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\ & in \cdot (id + reverse) \cdot \mathsf{F}\, inv \cdot (id + reverse) = in \cdot \mathsf{F}\, inv \\ \equiv & \quad \{ \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ \} \\ & \cdots \text{(completar com os passos e justificações que faltam)} \cdots \end{array}$$

   NB: assuma propriedades óbvias sobre a função $reverse$ como, por exemplo, $reverse \cdot reverse = id$, a sua propriedade "grátis" $(map\ f) \cdot reverse = reverse \cdot (map\ f)$, etc.

GRUPO III

Questão 7 Na programação funcional é vulgar a ocorrência de funções parciais, *i.é*, funções indefinidas para algum dos seus argumentos. Por exemplo, a divisão é parcial pois $n\ /\ 0$ é um valor indefinido, ou *excepção*. No sentido de se assinalarem as excepções por mensagens de erro, estende-se o codomínio da função por forma a fornecer 'strings' explicativos. Em Haskell, por exemplo,

```
(/) :: Double -> Double -> Double
```

pode ser estendida a

$$
\begin{array}{l}
dv :: (Double, Double) \rightarrow Error\ Double \\
dv\ (n, 0) = Err\ \texttt{"Nem pense em dividir por 0!"} \\
dv\ (n, m) = Ok\ (n\ /\ m)
\end{array}
$$

onde Err e Ok são construtores do tipo

$$\mathbf{data}\ Error\ a = Err\ String \mid Ok\ a\ \mathbf{deriving}\ Show$$

cf.

$$
\begin{tikzcd}
Error\ a \arrow[r, bend left, "outE"] \arrow[r, phantom, "\cong"] & String + a \arrow[l, bend left, "inE"]
\end{tikzcd}
$$

que se promove a functor definindo

$$\mathbf{instance}\ Functor\ Error\ \mathbf{where}\ fmap\ f = inE \cdot (id + f) \cdot outE$$

e a mónade fazendo $return = Ok$ (unidade) e $join = [Err\ , id] \cdot outE$ (multiplicação μ). Calcule a definição *pointwise* de $\gg=$ para este mónade.

Questão 8 No módulo $St.hs$ define-se uma versão do mónade de estado com base no tipo de dados

$$\mathbf{data}\ St\ s\ a = St\{st :: (s \rightarrow (a, s))\}$$

que se mostra capaz de instanciar a classe $Monad$,

$$
\begin{array}{l}
\mathbf{instance}\ Monad\ (St\ s)\ \mathbf{where} \\
\quad return = St \cdot \overline{id} \\
\quad (St\ x) \gg= g = St\ (\widehat{(st \cdot g)} \cdot x)
\end{array}
$$

e depois a classe $Functor$:

$$
\begin{array}{l}
\mathbf{instance}\ Functor\ (St\ s)\ \mathbf{where} \\
\quad fmap\ f\ t = \mathbf{do}\ \{a \leftarrow t; return\ (f\ a)\}
\end{array}
$$

Mostre que esta forma de declarar functores a partir de mónades está correcta.

Questão 9 Defina o combinador $foldBTree :: t \rightarrow (a \rightarrow t \rightarrow t \rightarrow t) \rightarrow BTree\ a \rightarrow t$ para o tipo

$$\mathbf{data}\ BTree\ a = Empty \mid Node\ (a, (BTree\ a, BTree\ a))\ \mathbf{deriving}\ Show$$

que conhece da biblioteca $BTree.hs$ e, a partir desse, a sua variante monádica de tipo $foldBTreeM :: (Monad\ m) \Rightarrow t \rightarrow (a \rightarrow t \rightarrow t \rightarrow t) \rightarrow BTree\ a \rightarrow m\ t$.
