Cálculo de Programas
Trabalho Teórico-Prático
LCC — Ano Lectivo de 2007/08

J.N. Oliveira & O.M. Pacheco
Departamento de Informtica
Universidade do Minho

Junho de 2008

Conteúdo

1 Preâmbulo

A disciplina de Cálculo de Programas tem como objectivo principal ensinar a programação de computadores como uma disciplina científica. Para isso parte-se de uma álgebra da programação (conjunto de leis universais e seus corolários) e aplica-se essa álgebra a problemas de programação por resolução de equações cujas incógnitas são os próprios programas que se querem obter.

Por razões de ordem pedagógica, restringe-se a aplicação desta ciência ao desenvolvimento de programas funcionais e usa-se a linguagem HASKELL como veículo de programação [1].

O presente trabalho tem por objectivo concretizar na prática os objectivos da disciplina acima enunciados: (a) por um lado, colocar os alunos perante problemas de programação que deverão ser resolvidos em HASKELL; (b) por outro lado, convidar os alunos a documentarem esses mesmos programas com os cálculos que precederam a sua codificação. Há ainda um terceiro objectivo: o de ensinar a produzir textos técnico-científicos de qualidade.

Para cumprir de forma integrada e simples os objectivos acima vamos recorrer a uma técnica de programação dita literária [4], cujo princípio base é o seguinte: *a documentação de um programa deve coincidir com ele próprio*. Por outras palavras, o código fonte e a sua documentação deverão constar do mesmo documento (ficheiro). O texto que está a ler é um exemplo de *programação literária*, como adiante se referirá.

2 Como realizar o trabalho

Este trabalho teórico-prático deve ser realizado por grupos com um máximo de três alunos. Os detalhes da avaliação (datas para submissão do relatório e sua defesa oral) são os que constam da página da disciplina na *internet*, onde também se encontra o ficheiro original do presente texto. Recomenda-se uma abordagem equilibrada e participativa dos membros do grupo de trabalho por forma a poderem responder às questões que serão colocadas na defesa oral do relatório.

Em que consiste, então, o *relatório* a que se refere o parágrafo anterior? Deverá ser um texto de programação literária [4] que responda aos requisitos e questões de programação que constam deste enunciado, mais adiante.

3 Como documentar código HASKELL

Na programação científica, como é aquela que se ensina nesta disciplina, é tão importante produzir um bom programa, ou pacote de programas (vulg. *aplicação de 'software'*) como produzir evidência sobre a forma como foi concebido (decisões tomadas, etc). Este aspecto faz com que a *documentação* sobre um programa seja tanto ou mais valiosa que o programa propriamente dito. (Essa documentação é, afinal, o mais precioso e duradouro bem de que uma empresa dispõe sempre que um autor de um produto de 'software' seu a abandona.)

Um erro clássico dos hábitos tradicionais de programação é o de se escrever a documentação só depois do respectivo programa *estar pronto*. Por via de regra, a documentação escrita nessa fase tardia é omissa e de má qualidade. Isto porque muitos detalhes do desenvolvimento já estão esquecidos nessa altura, sendo muitas vezes tudo escrito *a correr*.

Todos são unânimes em recomendar que a documentação seja escrita *ao mesmo tempo que* o código, e para isso nada melhor do que usar o *mesmo ficheiro* de texto para registar código e documentação. Como já se disse acima, chama-se a esta estratégia programação literária, sendo o HASKELL uma das linguagens em que tal estratégia é prática comum.

[1] A álgebra da programação estende-se à lógica em geral, encarada como liguagem para especificar problemas e construir soluções. Esta continuação desta disciplina aborda-se na unidade curricular *Métodos Formais em Engenharia de Software* do Mestado em Informática desta Universidade, cf.

`http://www.di.uminho.pt/ensino/formacao-avancada/metodos-formais-em-engenharia-de-software`.

É o que se passa com o texto (PDF) que neste momento está a ler: foi gerado a partir de um ficheiro (`cp0708t.lhs`) usando o processador de texto LaTeX [3]; mas, se submeter o mesmo ficheiro ao seu interpretador de HASKELL preferido, verá que o código nele incluído será seleccionado, carregado e executado.

Exemplo: vamos declarar de seguida os isomorfismos in e out associados à definição de listas em HASKELL tal como foram definidos nas aulas desta disciplina. Primeiro vamos dar um nome ao módulo que estamos a definir

> **module** *Cp0708t* **where**
> **import** *Cp*
> **import** *BTree*
> **import** *LTree*
> **import** *Ratio*
> **import** *HugsList*
> **import** *System.Time*

aproveitando para indicar que ele depende de funções disponíveis nos módulos *Cp*, *BTree*, etc. Basta, então escrevermos:

$$inl = [\underline{[]}\ , \widehat{(:)}]$$
$$out\ [\,] = i_1\ ()$$
$$out\ (a : x) = i_2\ (a, x)$$

Se se inspeccionar o ficheiro `cp0708t.lhs` vê-se que todo o texto indentado acima está entre dois delimitadores, `\begin{code}` e `\end{code}`. É nisso que consiste o protocolo que o LaTeX e o HASKELL estabelecem entre si: quando o LaTeX encontra blocos `\begin{code} ... \end{code}`, limita-se a mostrá-los em formato *pretty-printed*. Por seu lado, o HASKELL está preparado para ignorar tudo o que é texto LaTeX *e só prestar atenção* ao que está dentro desses blocos.

Por exemplo, se se submeter `cp0708t.lhs` ao GHCi [2], ter-se-á evidência de que tudo correu bem,

```
   ___         ___ _
  / _ \ /\  /\/ __(_)
 / /_\// /_/ / /  | |      GHC Interactive, version 6.4.1, for Haskell 98.
/ /_\\/ __  / /___| |      http://www.haskell.org/ghc/
\____/\/ /_/\____/|_|      Type :? for help.

Loading package base-1.0 ... linking ... done.
Compiling Cp                ( ./Cp.hs, interpreted )
Compiling LTree             ( ./LTree.hs, interpreted )
Compiling HugsList          ( ./HugsList.hs, interpreted )
Compiling BTree             ( ./BTree.hs, interpreted )
Compiling Cp0708t           ( cp0708t.lhs, interpreted )
Ok, modules loaded: Cp0708t, BTree, HugsList, LTree, Cp.
```

podendo-se de imediato questionar o interpretador.

Por exemplo, pode pedir-se para este calcular *out* `["a","b","c"]`, devendo-se obter

```
*Cp0708t> out ["a","b","c"]
Right ("a",["b","c"])
```

etc.

Continuando, vamos agora declarar o combinador $(\!|\_|\!)$ que nos permite construir *catamorfismos* de listas:

[2] Note que tem de ter acessível os ficheiros `Cp.hs`, etc que são importados. Note ainda que deve invocar o GHCi com as opções de comando `-fglasgow-exts -fallow-undecidable-instances -fallow-overlapping-instances`.

$$cata\ g = g \cdot rec\ (cata\ g) \cdot out$$

onde

$$rec\ f = id + id \times f$$

Voltando de novo ao GHCi, podemos inquirir qual o tipo de $cata$,

```
*Cp0708t> :t cata
cata :: forall c a. (Either () (a, c) -> c) -> [a] -> c
```

qual o resultado da aplicação do catamorfismo

$$(\!|[\underline{0}\ , (1+) \cdot \pi_2]|\!) \tag{1}$$

à lista acima,

```
*Cp0708t> cata (either (const 0) ((1+) . p2)) ["a","b","c"]
3
```

etc.

4 Como documentar cálculos de programas

Nas aulas desta disciplina, sempre que se pretende derivar um programa em HASKELL, ou raciocinar sobre ele, não se usa a notação HASKELL directamente, mas sim uma notação mais compacta, de feição matemática, que permite abreviar texto a agilizar os cálculos. A expressão (1) acima, por exemplo, é a que é usada no cálculo de catamorfismos e não a expressão HASKELL que acima se passou ao GHCi. Se quisermos, podemos ver (1) como o "pretty-printing" do referido texto em sintaxe HASKELL.

Assim, não vamos usar *programação literária* para documentar os cálculos, mas sim o próprio LaTeX, que tem uma natural afinidade com a matemática. De facto, a produção de texto em LaTeX é feita declarando funções (designadas "macros") cujo resultado é o texto a imprimir. Veja-se o exemplo do operador unário $cata$, que em LaTeX é a macro unária do mesmo nome declarada no ficheiro auxiliar `cp0708t.sty`.

A inspecção deste ficheiro revelará ao leitor a vantagem de se reproduzir na linguagem do processador de texto a estrutura da linguagem matemática que estamos a usar [3]. Isso revela-se em particular no que diz respeito à apresentação dos cálculos, tal como o que se segue, extraído "verbatim" dos apontamentos teóricos da disciplina [5]:

$$\begin{array}{ll} & swap \cdot swap \\ = & \quad \{\ \text{by definition } swap \stackrel{\text{def}}{=} \langle \pi_2, \pi_1 \rangle\ \} \\ & \langle \pi_2, \pi_1 \rangle \cdot swap \\ = & \quad \{\ \text{by } \times\text{-fusion (12)}\ \} \\ & \langle \pi_2 \cdot swap, \pi_1 \cdot swap \rangle \\ = & \quad \{\ \text{definition of } swap \text{ twice}\ \} \\ & \langle \pi_2 \cdot \langle \pi_2, \pi_1 \rangle, \pi_1 \cdot \langle \pi_2, \pi_1 \rangle \rangle \\ = & \quad \{\ \text{by } \times\text{-cancellation (10)}\ \} \\ & \langle \pi_1, \pi_2 \rangle \\ = & \quad \{\ \text{by } \times\text{-reflexion (11)}\ \} \\ & id \end{array}$$

[3] Em particular, a secção *Importing/modifying isolatin1* mostra como usar caracteres especiais para esse efeito.

A inspecção de filecp0708t.lhs revelará uma estrutura

```
\begin{calculation}
%
     swap \comp swap
%
\just={ .................. }
%
        ......
%
\just={ .................. }
%
        id
%
\end{calculation}
```

cujo 'layout' se aproxima bastante do que é produzido e onde desempenha papel muito relevante a *macro* binária `just` (de *"justificação"*) cujo primeiro argumento é o símbolo a justificar (igualdade de funções, neste caso) e o segundo é o texto justificativo.

5 O que se pretende

O anexo A deste texto contém um enunciado de questões várias sobre a matéria da disciplina, umas de natureza teórica e outras de natureza prática, devendo resultar das primeiras cálculos de programas em estilo *point-free* e das últimas programas em HASKELL que deverão poder ser testados num interpretador de HASKELL. Os alunos devem responder a cada pergunta escrevendo o texto `lhs` *("literate haskell")* que entenderem e como entenderem, ora escrevendo os seus cálculos como acima se explicou, ora embebendo código HASKELL no seu texto. Para imprimir o relatório basta usar a sequência de comandos

```
lhs2TeX -o cp0708t.tex cp0708t.lhs
pdflatex cp0708t
```

(onde se usa `cp0708t.lhs` como exemplo) em que o papel essencial é desempenhado pelo pre-processador `lhs2TeX` [4].

Sugere-se o aproveitamento da fonte LaTeX deste documento (`cp0708t.lhs`) como ponto de partida. O ficheiro fonte do relatório deverá ser enviado à equipa docente seguindo as instruções que serão publicadas na página da disciplina.

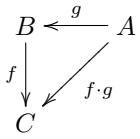
Os alunos deverão ainda aproveitar este texto para mostrar que sabem manipular biliografias (em BibTeX) e índices/glossários (usando `makeindex`). Para se conhecer a vastíssima documentação disponível sobre LaTeX, BibTeX, etc, etc, basta escrever `"latex"`, `"bibtex"`, `"makeindex"` etc num motor de busca e começar a navegar. Ou então visitar o TUG (*TeX Users Group*) em `www.tug.org/`.

6 Desenho de diagramas

Há muitos *pacotes LaTeX* para desenhar diagramas e outros objectos gráficos [1]. Sugere-se, entre estes, o XY-PIC [5], pela sua simplicidade e pelo facto de constar das principais distribuições de LaTeX. Como exemplo,

[4] Ver `www.informatik.uni-bonn.de/~loeh/lhs2tex`.
[5] Ver `http://www.tug.org/applications/Xy-pic/`.

dá-se o diagrama,



extraído de [5]. Bastará inspeccionar o ficheiro `cp0708t.lhs` para ver quão simples é desenhar diagramas como este.

A Exercícios a resolver

Como orientação geral para a resolução das questões que se seguem, considerem que as respostas (sejam elas cálculos, provas, código HASKELL, etc) serão tanto mais valorizadas quanto mais simples forem. (No caso de código HASKELL, procurem usar combinadores que reduzam o número de linhas de código.)

O número mínimo de alíneas a resolver é 8, das quais 4 são obrigatórias (marcadas com o sinal "† "). As restantes 4 são à escolha de cada grupo. As valorizações de questões ou alíneas obrigatórias, quando existam, são naturalmente opcionais.

Exercício 1. † Demonstre a lei de fusão da exponenciação, cf. (33) no Anexo B, pág. 14, desenhando o diagrama respectivo.

Exercício 2. A função exponencial $e^x : \mathbb{R} \to \mathbb{R}$ (onde "e" designa o número de Euler) pode ser definida de várias maneiras, entre elas aquela que toma a forma de uma série de Taylor:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \tag{2}$$

A seguinte função, em HASKELL,

$$\begin{aligned} &euler :: Double \to Integer \to Double \\ &euler\ x\ 0 = 1 \\ &euler\ x\ (n+1) = x\ .\hat{}\ (n+1)\ /\ fac\ (n+1) + (euler\ x\ n) \end{aligned}$$

calcula uma aproximação a e^x, onde o parâmetro adicional indica o número de parcelas da série de Taylor que estão a ser tomadas em consideração. Por exemplo, enquanto que $euler\ 1\ 1 = 2.0$, já $euler\ 1\ 10$ dá o resultado 2.7182818011463845.

Recorde das aulas práticas que a exponenciação

$$\begin{aligned} &a\ .\hat{}\ 0 = 1 \\ &a\ .\hat{}\ (n+1) = (a\ .\hat{}\ n) * a \end{aligned}$$

é um catamorfismo de naturais, e que a função factorial também se pode exprimir através de um catamorfismo do mesmo tipo:

$$\begin{aligned} &fac = \pi_2 \cdot fcata\ \mathbf{where} \\ &\quad fcata\ 0 = (1,1) \\ &\quad fcata\ (n+1) = \mathbf{let}\ (a,b) = fcata\ n\ \mathbf{in}\ (a+1, a*b) \end{aligned}$$

Contudo, a função $euler$ não é muito eficiente. Se recorrer às funções disponíveis na package $System.Time$ para contagem de tempo de CPU verifica que, enquanto o cálculo de $euler\ 1\ 100$ é quase instantâneo, o de $euler\ 1\ 1000$ leva cerca de 9 segundos, o de $euler\ 1\ 2000$ 33 segundos, etc

1. † Apresente em detalhe todos os cálculos que deverá fazer para, recorrendo a leis como a de recursividade múltipla e/ou o seu corolário *banana-split*, obter uma versão mais eficiente de $euler\ x\ n$.

2. Apresente resultados de testes de execução das duas funções (variando o valor de n) por forma a documentar o ganho em eficiência. Para esse efeito, transforme-as em funções sobre o mónade de IO e recorra, nesse contexto, a funções como $getClockTime$, $diffClockTimes$ etc. da package $System.Time$ que o ajudem nessa tarefa.

Exercício 3. Foi estudada nas aulas práticas da disciplina a generalização da lei distributiva

$$a * (b + c) \quad = \quad a * b + a * c \tag{3}$$

ao caso de n parcelas,

$$a * \sum_{i=1}^{n} b_i \quad = \quad \sum_{i=1}^{n} (a * b_i) \tag{4}$$

que se pode traduzir em HASKELL da forma seguinte:

$$distLaw\ a = [(a*) \cdot summation\ id, summation\ (a*)]$$

onde

$$summation\ f = cata\ [\underline{0}\ , \widehat{(+)} \cdot (f \times id)]$$

Mostre que este resultado pode ser generalizado a catamorfismos de outras estruturas como, por exemplo $LTree$ ou $BTree$.
NB: recorde das aulas a versão *pointfree* de (3):

$$(a*) \cdot \widehat{+} \quad = \quad \widehat{+} \cdot ((a*) \times (a*)) \tag{5}$$

Valorização: poderá optar por provar o resultado para o caso geral envolvendo catamorfismos de um tipo genérico $\mathsf{T}\ A \cong \mathsf{B}(A, \mathsf{T}\ A)$.

Exercício 4. Defina catamorfismos de naturais que construam a série $\{a_n\}$ e calculem o somatório do problema 3/4/19 do Round 4 proposto pela USAMTS (`http://www.usamts.org/Problems/U_Problems.php`).

Exercício 5. Se inspeccionar, através do código disponível em `demos.hs` do material pedagógico da disciplina, as estruturas intermédias (virtuais) dos hilomorfismos *hanoi* e *qSort*, de tipo $BTree$, apercebe-se de propriedades que essas estruturas têm, quaisquer que sejam os argumentos que sejam passados àquelas funções.

Propriedades desta natureza são habitualmente designadas *invariantes*. Codifique o teste dessas propriedades sobre a forma de hilomorfismos booleanos (predicados) sobre $BTree$.

Exercício 6. Use a lei de fusão-cata instanciada para $LTree$ na demonstração da propriedade seguinte,

$$countLTree \cdot invLTree \quad = \quad countLTree \tag{6}$$

que nos diz que $countLTree$ é invariante de $invLTree$ (ie., o número de folhas de uma $LTree$ não se altera com o seu espelhamento).

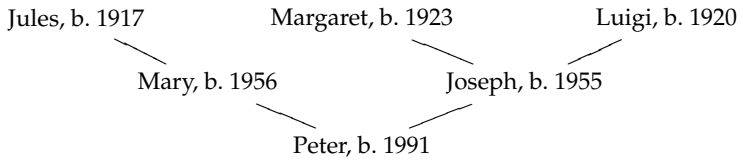
Exercício 7. O matemático francês Édouard Lucas (1842-1891), a quem se atribui a formulação do problema das Torres de Hanói, generalizou a série de números de Fibonacci às chamadas *sequências de Lucas* (ver por exemplo `http://en.wikipedia.org/wiki/Lucas_sequence`).

Recorra às bibliotecas estudadas nesta disciplina para definir funções que implementem o cálculo de tais sequências.

Exercício 8. Seja dado o tipo de dados, em HASKELL

$$\begin{array}{l}\textbf{data}\ PTree\ a\ b = Ind\{\\ \quad name :: a,\\ \quad birth\ :: b,\\ \quad mother :: Maybe\ (PTree\ a\ b),\\ \quad father :: Maybe\ (PTree\ a\ b)\\ \quad \}\ \textbf{deriving}\ (Show, Eq)\end{array}$$

para registo de árvores genealógicas. Por exemplo, a árvore



(7)

é representada pela $PTree$

$$\begin{array}{l}t = Ind\{\, name = \texttt{"Peter"},\\ \quad birth = 1991,\\ \quad mother = Just\ (Ind\{\, name = \texttt{"Mary"},\\ \quad\quad birth = 1956,\\ \quad\quad mother = Nothing,\\ \quad\quad father = Just\ (Ind\{\, name = \texttt{"Jules"},\\ \quad\quad\quad birth = 1917,\\ \quad\quad\quad mother = Nothing,\\ \quad\quad\quad father = Nothing\,\})\,\}),\\ \quad father = Just\ (Ind\{\, name = \texttt{"Joseph"},\\ \quad birth = 1955,\\ \quad mother = Just\ (Ind\{\, name = \texttt{"Margaret"},\\ \quad\quad birth = 1923,\\ \quad\quad mother = Nothing,\\ \quad\quad father = Nothing\,\}),\\ \quad father = Just\ (Ind\{\, name = \texttt{"Luigi"},\\ \quad\quad birth = 1920,\\ \quad\quad mother = Nothing,\\ \quad\quad father = Nothing\,\})\,\})\,\}\end{array}$$

1. † Defina a álgebra $inPTree :: ((a, b), Maybe\ (PTree\ a\ b), Maybe\ (PTree\ a\ b)) \rightarrow PTree\ a\ b$, a sua conversa $outPTree$ e ainda a triologia $anaPTree$, $cataPTree$, $hyloPTree$. Defina também $PTree$ como instância da classe $BiFunctor$. Use esses combinadores na formulação *pointfree* das seguintes operações:

   - contagem dos indivíduos de uma $PTree$
   - transformação de uma dada $PTree$ numa outra em que cada data de nascimento dá lugar à idade actual do respectivo indivíduo, mantendo-se o resto da informação.

   Valorização: definir a operação que representa graficamente uma dada $PTree$; pode gerar essa visualização usando uma *xymatrix* — tal como em (7) — ou recorrendo a um pacote gráfico do seu agrado, eg. a biblioteca `SOEGraphics` [2].

2. Uma outra forma de representar árvores genealógicas consiste em associar identificadores únicos a cada indivíduo ("BI"s) e usá-los como referências para os seus progenitores. Uma representação da árvore (7) neste tipo de formato poderá ser, por exemplo, a estrutura

$$
\begin{array}{l}
h1 = Heap\ [\\
\quad 1 \mapsto (("\texttt{Peter}", 1991), Just\ 2, Just\ 3), \\
\quad 2 \mapsto (("\texttt{Mary}", 1956), Nothing, Just\ 6), \\
\quad 6 \mapsto (("\texttt{Jules}", 1917), Nothing, Nothing), \\
\quad 3 \mapsto (("\texttt{Joseph}", 1955), Just\ 5, Just\ 7), \\
\quad 5 \mapsto (("\texttt{Margaret}", 1923), Nothing, Nothing), \\
\quad 7 \mapsto (("\texttt{Luigi}", 1920), Nothing, Nothing) \\
\quad\ ]\ 1
\end{array}
$$

   que habita o tipo

$$\textbf{data}\ Heap\ a\ k = Heap\ [(k, (a, Maybe\ k, Maybe\ k))]\ k\ \textbf{deriving}\ Show$$

   e onde se usa a notação

$$x \mapsto y = (x, y)$$

   para indicar a associação de informação a cada BI. Note que o construtor $Heap$ é binário: o seu primeiro argumento é a lista associativa a que acima se referiu. O segundo é o BI do indivíduo que é a raiz da árvore (1, no caso acima).

   Para se ver que $h1$ representa de facto t basta calcular

$$teste = absf\ h1 \equiv t$$

   e verificar que $teste = True$, onde $absf$ é a função (parcial) que converte $Heap$ em $PTree$:

$$
\begin{array}{l}
absf :: (Eq\ k) \Rightarrow Heap\ (a, b)\ k \rightarrow PTree\ a\ b \\
absf\ (Heap\ h\ k) = \textbf{let}\ Just\ (a, x, y) = lookup\ k\ h \\
\quad \textbf{in}\ Ind\ (\pi_1\ a)\ (\pi_2\ a) \\
\qquad\quad (fmap\ (absf \cdot Heap\ h)\ x) \\
\qquad\quad (fmap\ (absf \cdot Heap\ h)\ y)
\end{array}
$$

   Contudo, nem todo o $Heap$ representa uma árvore de tipo $PTree$ — como se pode ver aplicando $absf$ aos dois $Heap$s que se seguem,

$$
\begin{array}{l}
h2 = Heap\ [\\
\quad 1 \mapsto (("\texttt{Peter}", 1991), Just\ 2, Just\ 3),
\end{array}
$$

$$
\begin{array}{l}
\quad 2 \mapsto ((\texttt{"Mary"}, 1956), \mathit{Nothing}, \mathit{Just}\ 9),\\
\quad 6 \mapsto ((\texttt{"Jules"}, 1917), \mathit{Nothing}, \mathit{Nothing}),\\
\quad 3 \mapsto ((\texttt{"Joseph"}, 1955), \mathit{Just}\ 5, \mathit{Just}\ 7),\\
\quad 5 \mapsto ((\texttt{"Margaret"}, 1923), \mathit{Nothing}, \mathit{Nothing}),\\
\quad 7 \mapsto ((\texttt{"Luigi"}, 1920), \mathit{Nothing}, \mathit{Nothing})\\
\quad\ ]\ 1
\end{array}
$$

e

$$
\begin{array}{l}
h3 = \mathit{Heap}\ [\\
\quad 1 \mapsto ((\texttt{"Peter"}, 1991), \mathit{Just}\ 2, \mathit{Just}\ 3),\\
\quad 2 \mapsto ((\texttt{"Mary"}, 1956), \mathit{Nothing}, \mathit{Just}\ 6),\\
\quad 6 \mapsto ((\texttt{"Jules"}, 1917), \mathit{Nothing}, \mathit{Nothing}),\\
\quad 3 \mapsto ((\texttt{"Joseph"}, 1955), \mathit{Just}\ 5, \mathit{Just}\ 1),\\
\quad 5 \mapsto ((\texttt{"Margaret"}, 1923), \mathit{Nothing}, \mathit{Nothing}),\\
\quad 7 \mapsto ((\texttt{"Luigi"}, 1920), \mathit{Nothing}, \mathit{Nothing})\\
\quad\ ]\ 1
\end{array}
$$

Identifique quais são os problemas em cada caso e re-escreva $absf$ por forma a colmatar o seu comportamento deficiente.

Valorização: investigue como definir a conversa de $absf$ — isto é, a função $repf :: PTree\ a\ b \to Heap\ (a, b)\ Integer$ que representa árvores em *heaps* — sob a forma de um catamorfismo, isto é, encontre o gene g tal que $repf = cataPTree\ g$.

3. Declare $Heap$ com instância da classe $BiFunctor$ por forma a que, em $bmap\ g\ f$, f seja a função que altera BIs. Com base nessa instância, investigue propriedades que f deva satisfazer para que a árvore representada por um dado $Heap\ h$ seja a mesma que é representada por $bmap\ id\ g\ h$, isto é, tal que

$$absf\ (bmap\ id\ g\ h) \equiv absf\ h$$

Exercício 9. No seu livro *The Haskell School of Expression* [2], Paul Hudak dá a seguinte sintaxe para a notação musical:

```
data Music = Note Pitch Dur    -- a note (atomic object)
  | Rest Dur                   -- a rest (atomic object)
  | Music :+: Music            -- sequential composition
  | Music :=: Music            -- parallel composition
  | Tempo (Ratio Int) Music    -- scale the tempo
  | Trans  Int Music           -- transposition
  | Instr  IName Music         -- instrument label
  deriving (Show, Eq)
```

onde

```
type Pitch    = (PitchClass, Octave)
data PitchClass = Cf | C | Cs | Df | D | Ds | Ef | E | Es | Ff | F | Fs
                | Gf | G | Gs | Af | A | As | Bf | B | Bs
  deriving (Eq, Show)
```

$\mathbf{type}\ Octave = Int$
$\mathbf{type}\ Dur = Ratio\ Int$ -- in whole notes
$\mathbf{type}\ IName = String$ -- instrument name, eg. piano, flute etc

Apesar da sua simplicidade, esta sintaxe (abstracta) é suficiente para grafar música polifónica a várias vozes e instrumentos, e para gerar ficheiros audíveis em formato MIDI arbitrariamente complexos [6].

1. O isomorfismo

$$\begin{aligned}
&outMusic :: Music \\
&\quad \to (Maybe\ Pitch, Dur) + ((Una\ (Ratio\ Int)\ Int\ IName, Music) + (Bin, (Music, Music))) \\
&outMusic\ (Note\ p\ d) = i_1\ (Just\ p, d) \\
&outMusic\ (Rest\ d) \quad = i_1\ (Nothing, d) \\
&outMusic\ (m1 :+: m2) = i_2\ (i_2\ (Seq, (m1, m2))) \\
&outMusic\ (m1 :=: m2) \ = i_2\ (i_2\ (Par, (m1, m2))) \\
&outMusic\ (Tempo\ t\ m) = i_2\ (i_1\ (Temp\ t, m)) \\
&outMusic\ (Trans\ t\ m) = \ i_2\ (i_1\ (Tra\ t, m)) \\
&outMusic\ (Instr\ t\ m) = \ \ i_2\ (i_1\ (Ins\ \ t, m))
\end{aligned}$$

onde

$$\begin{aligned}
&\mathbf{data}\ Bin = Seq \mid Par \\
&\mathbf{data}\ Una\ t\ t'\ i = Temp\ t \mid Tra\ t' \mid Ins\ i
\end{aligned}$$

é testemunha de que $Music$ é um tipo polinomial do 2.º grau, com base

$$\mathsf{B}(A, B, C, X) \quad = \quad C + (B \times X + (A \times X^2)) \tag{8}$$

isto é, tem-se

$$recMusic\ f = id + (id \times f + (id \times (f \times f)))$$

e

$$cataMusic\ f = f \cdot (recMusic\ (cataMusic\ f)) \cdot outMusic$$

Considere o catamorfismo

$$h = cataMusic\ [maybe\ 0\ \underline{1} \cdot \pi_1\ , [id\ , \widehat{(+)}] \cdot (\pi_2 + \pi_2)]$$

O que faz esta função? Apresente exemplos informativos. Escreva ainda como catamorfismo a função que calcula a duração de um dado trecho musical.

2. Complete a definição da biblioteca associada a $Music$ (parametrizando-a) e desenhe o diagrama dos respectivos hilomorfismos.

Exercício 10. Na secção (5) da biblioteca `LTree.hs` declara-se o tipo $LTree\ a$ como instância da classe $Monad$. Apresente cálculos que provem que as propriedades (38) e (39) são de facto válidas, para este

[6] Ver `http://www.haskell.org/haskore`. Por exemplo, o ficheiro `cp0708t.mid` que tem disponível no material pedagógico foi gerado a partir da conversão para MIDI do trecho cuja sintaxe encontra na variável s. (Para não sobrecarregar o presente módulo, essa variável está comentada neste documento; terá que "descomentar" o respectivo texto para a ter acessível em LaTeX e em HASKELL.)

mónade.
Valorização: poderá optar por provar que, em geral, qualquer tipo cuja base é $\mathsf{B}(f,g) = f + \mathsf{F}\, g$ (onde F é um functor arbitrário) forma um mónade, para $\mu = (\!|[id\,, in \cdot i_2]|\!)$ e $u = in \cdot i_1$. Já agora: que outros mónades estudou que caem nesta classe?

Exercício 11. † Com base no exemplo que vem dado no ficheiro `mobile.hs`, escreva em HASKELL um simulador de uma máquina de calcular que inclua um registo principal (correspondente ao visor da máquina) e memória. Simule o pressionar de teclas (eg. MC, M+, M-, etc) por comandos de IO, aos quais o simulador deve reagir mostrando a actualização do visor (registo principal).
Valorização: combine os mónades de estado e de IO com o de tratamento de erro por forma a lidar com operações matematicamente erradas, por exemplo, a sequência "1", "/", "0", "=".

B Algumas leis do cálculo funcional

B.1 Produto

$$\textbf{Universal-}\times \quad k = \langle f, g\rangle \quad \equiv \quad \begin{cases} \pi_1 \cdot k = f \\ \pi_2 \cdot k = g \end{cases} \tag{9}$$

$$\textbf{Cancelamento-}\times \quad \pi_1 \cdot \langle f, g\rangle = f \quad , \quad \pi_2 \cdot \langle f, g\rangle = g \tag{10}$$

$$\textbf{Reflexão-}\times \quad \langle \pi_1, \pi_2\rangle \quad = \quad id_{A\times B} \tag{11}$$

$$\textbf{Fusão-}\times \quad \langle g, h\rangle \cdot f = \langle g \cdot f, h \cdot f\rangle \tag{12}$$

$$\textbf{Absorção-}\times \quad (i \times j) \cdot \langle g, h\rangle = \langle i \cdot g, j \cdot h\rangle \tag{13}$$

$$\textbf{Def-}\times \quad f \times g = \langle f \cdot \pi_1, g \cdot \pi_2\rangle \tag{14}$$

$$\textbf{Natural-}\pi_1 \quad \pi_1 \cdot (f \times g) \quad = \quad f \cdot \pi_1 \tag{15}$$

$$\textbf{Natural-}\pi_2 \quad \pi_2 \cdot (f \times g) \quad = \quad g \cdot \pi_2 \tag{16}$$

$$\textbf{Functor-}\times \quad (g \cdot h) \times (i \cdot j) \quad = \quad (g \times i) \cdot (h \times j) \tag{17}$$

$$\textbf{Functor-id-}\times \quad id_A \times id_B \quad = \quad id_{A\times B} \tag{18}$$

$$\textbf{Eq-}\times \quad \langle f, g\rangle = \langle h, k\rangle \ \equiv \ f = h \wedge g = k \tag{19}$$

B.2 Coproduto

$$\textbf{Universal-}+ \quad k = [f\ , g] \quad \equiv \quad \begin{cases} k \cdot i_1 = f \\ k \cdot i_2 = g \end{cases} \tag{20}$$

$$\textbf{Cancelamento-}+ \quad [g\ , h] \cdot i_1 = g \quad , \quad [g\ , h] \cdot i_2 = h \tag{21}$$

$$\textbf{Reflexão-}+ \quad [i_1\ , i_2] \quad = \quad id_{A+B} \tag{22}$$

$$\textbf{Fusão-}+ \quad f \cdot [g\ , h] = [f \cdot g\ , f \cdot h] \tag{23}$$

$$\textbf{Absorção-}+ \quad [g\ , h] \cdot (i + j) = [g \cdot i\ , h \cdot j] \tag{24}$$

$$\textbf{Def-}+ \quad f + g = [i_1 \cdot f\ , i_2 \cdot g] \tag{25}$$

$$\textbf{Functor-}+ \quad (g \cdot h) + (i \cdot j) = (g + i) \cdot (h + j) \tag{26}$$

$$\textbf{Functor-id-}+ \quad id_A + id_B = id_{A+B} \tag{27}$$

$$\textbf{Eq-}+ \quad [f\ , g] = [h\ , k] \ \equiv \ f = h \ \wedge \ g = k \tag{28}$$

$$\textbf{Lei da troca} \quad [\langle f, g\rangle\ , \langle h, k\rangle] \quad = \quad \langle [f\ , h], [g\ , k]\rangle \tag{29}$$

B.3 Exponenciação

$$\textbf{Universal-exp} \quad k = \overline{f} \quad \equiv \quad f = ap \cdot (k \times id) \tag{30}$$

$$\textbf{Cancelamento-exp} \quad f = ap \cdot (\overline{f} \times id) \tag{31}$$

$$\textbf{Reflexão-exp} \quad \overline{ap} = id_{B^A} \tag{32}$$

$$\textbf{Fusão-exp} \quad \overline{g \cdot (f \times id)} = \overline{g} \cdot f \tag{33}$$

$$\textbf{Absorção-exp} \quad f^A \cdot \overline{g} = \overline{f \cdot g} \tag{34}$$

$$\textbf{Def-exp} \quad f^A = \overline{f \cdot ap} \tag{35}$$

$$\textbf{Functor-exp} \quad (g \cdot h)^A = g^A \cdot h^A \tag{36}$$

$$\textbf{Functor-id-exp} \quad id^A = id \tag{37}$$

B.4 Mónadas

Multiplicação	$\mu \cdot \mu = \mu \cdot \mathsf{F}\,\mu$	(38)
Unidade	$\mu \cdot u = \mu \cdot \mathsf{F}\,u = id$	(39)

C Fonte de `cp0708t.mid`

Omitida sob a forma de um comentário LaTeX.

Referências

[1] Michel Goossens, Sebastian Rahtz, and Frank Mittelbach. *The LaTeX Graphics Companion*. Addison-Wesley, 1997. ISBN 0-201-85469-4.

[2] P. Hudak. *The Haskell School of Expression - Learning Functional Programming Through Multimedia*. Cambridge University Press, 1st edition, 2000. ISBN 0-521-64408-9.

[3] D.E. Knuth. *The TEXbook*. Addison-Wesley Publishing Company, 7th edition, 1986.

[4] D.E. Knuth. *Literate Programming*. CSLI Lecture Notes Number 27. Stanford University Center for the Study of Language and Information, Stanford, CA, USA, 1992.

[5] J.N. Oliveira. *Program Design by Calculation* . Draft of textbook in preparation (since 2005). Departamento de Informática, Universidade do Minho.

Índice