Trabalho Prático nº3

Métodos de Programação I

2004/2005

Resumo

O objectivo deste trabalho é consolidar os conhecimentos sobre padrões de recursividade universais de árvores e listas.

1 Preâmbulo

Este trabalho deve ser realizado por grupos de três alunos, e deve ser submetido de acordo com as instruções divulgadas na página da disciplina, antes da data limite aí mencionada.

2 Introdução

Será conveniente relembrar previamente o enunciado do trabalho n.1, em particular para os conceitos sobre expressões aritméticas e sua compilação, a utilizar na secção 3.2.

3 Trabalho a Realizar

3.1 Manipulação de Árvores Binárias

Considere os seguintes tipo de dados:

```
data FBTree b a = Leaf a | FNode b (FBTree b a) (FBTree b a)
data BTree a = Empty | Node a (BTree a) (BTree a)
```

1. Defina a função `cataFBTree` que gera os catamorfismos do tipo `FBTree`.
2. Utilizando a função anterior, escreva a função de mapeamento

```
mapFBTree :: (b -> c) -> (a -> d) -> FBTree b a -> FBTree c d
```

3. Defina as funções `cataBTree` e `anaBTree` que geram respectivamente os catamorfismos e os anamorfismos do tipo `BTree`.

4. Escreva os catamorfismos de travessia *preorder* e *postorder* `BTree a -> [a]` para o tipo `BTree`.

5. Escreva um catamorfismo de conversão de árvores do tipo `FBTree a a` para o tipo `BTree a`.

6. Escreva de novo a função da alínea anterior, agora como anamorfismo do tipo `BTree`.

3.2 Árvores de Expressão e Máquinas de Stack

Recorde o enunciado do Trabalho Prático n.1 do presente ano, em que se representava expressões aritméticas por árvores binárias, que eram depois compiladas para programas (sequências de instruções) de uma máquina de *stack*. Os seguintes tipos são apropriados para o problema descrito:

```
data OP = SOMA | SUB | PROD | DIV
type Expressao = FBTree OP Int

data Instr = Push Int | Op OP
type Prog = [Instr]
```

Nesta parte do trabalho deverá escrever todas as funções pedidas como combinações de funções definidas na secção 3.1.

1. Escreva a função de avaliação directa `avalia :: Expressao -> Int` como um catamorfismo.

2. Pretende-se em seguida efectuar a compilação de expressões para programas; para isso utiliza-se o seguinte tipo intermédio, de árvores de instruções:

   ```
   type ArvOp = FBTree Instr Instr
   ```

   (a) Escreva a função de conversão `convExprArvOp :: Expressao -> ArvOp`

   (b) Escreva agora uma função `compilePN :: Expressao -> Prog` que produz o programa que se obtém efectuando uma travessia *preorder* da árvore de instruções obtida a partir de uma expressão (diz-se um programa em *Polish Notation*).

   (c) Repita a alínea anterior para uma função `compileRPN :: Expressao -> Prog` que efectua uma travessia *postorder* da mesma árvore (resultando um programa em *Reverse Polish Notation*).

3. Finalmente, é necessário escrever-se as funções de execução dos programas.

(a) Escreva a função `execPN :: Prog -> [Int]`, que executa um programa a partir de uma *stack* vazia e devolve a *stack* final resultante da execução. Utilize um padrão de recursividade de listas apropriado.

(b) Complete a definição

```
execRPN :: Prog -> [Int]
execRPN p = execRPNt p []
  where execRPNt = ...
```

definindo `execRPN :: Prog -> [Int] -> [Int]` como um catamorfismo de listas (de ordem superior).

4. Em geral não é possível recuperar uma árvore a partir de uma lista resultante de uma sua travessia. No entanto, no caso concreto em questão, é possível converter um programa numa árvore de expressão. Escreva duas funções de conversão de programas PN e RPN em árvores de expressão.

3.3 Generalização para Árvores n-árias

Considere agora os tipos

```
data FTree b a = Leaf a | FNode b [FTree b a]
data Tree a = Node a [Tree a]
```

Repita os pontos da secção 3.1, com as alterações necessárias para estes tipos. Utilize-os para generalizar a definição de expressão aritmética, por forma a admitir operadores de aridade diferente de dois. Repita depois também os pontos da secção 3.2.

4 Valorização

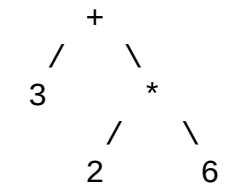
1. Implemente as funções de execução de programas PN/RPN com a utilização de um *State Monad*.

2. Utilize o *Monad IO* para visualizar os estados intermédios da *stack* durante a execução.

3. Complemente o seu trabalho com um *parser* de expressões aritméticas.

5 Diversos

Exemplo de uma árvore de expressão e dos programas PN e RPN dela resultantes:



[Op SOMA,Push 3,Op PROD,Push 2,Push 6] (PN)

[Push 3,Push 2,Push 6,Op PROD,Op SOMA] (RPN)