Paradigmas de Linguagens Computacionais
Exame Escrito

Paulo Borba
Centro de Informática
Universidade Federal de Pernambuco

5 de fevereiro de 2013

Questão 1 Defina em Haskell as seguintes funções que trabalham com listas que representam mapeamentos de elementos de um tipo (como chave) em elementos de outro tipo (como valor):

1. `add`, que recebe um par `(chave,valor)` e uma lista de pares `m`, retornando uma lista que representa o mapeamento que contém o par recebido mais os pares de `m` que não têm `chave` como primeiro elemento;
2. `domain`, que recebe uma lista de pares `m` e retorna uma lista com todos os primeiros elementos dos pares em `m`;
3. `image`, que recebe uma lista de pares `m` e retorna uma lista com todos os segundos elementos dos pares em `m`;
4. `apply`, que recebe um elemento `chave` e uma lista de pares `m`, retornando `Nothing` caso `chave` não seja mapeada por `m`, e `Just valor` caso `m` mapeie `chave` em `valor`.

Assuma a declaração do seguinte tipo: `data Maybe a = Nothing | Just a`. Pode também assumir que as listas recebidas não mapeiam a mesma chave em mais de um valor. Use compreensão de listas para definir `domain` e `image`. Indique e explique o tipo das funções definidas. □

```
data Chave t = Chave t | Nil deriving (Eq, Show)

add :: (Num u, Eq t) => (Chave t, u) -> [(Chave t, u)] -> [(Chave t, u)]
add _ [] = []
add par m = (geraChave par ([(Nil, v1) | (Nil, v1) <- m])) : ([(Chave c2, v2) | (Chave c2, v2) <- m])

geraChave :: (Num u) => (Chave t, u) -> [(Chave t, u)] -> (Chave t, u)
geraChave (chave, valor) [] = (chave, valor)
geraChave (chave, valor) ((c, v):as) = geraChave (chave, valor + v) as

domain :: [(Chave t, u)] -> [t]
domain pares = [c | (Chave c, v) <- pares ]

image :: [(Chave t, u)] -> [u]
image pares = [v | (Chave c, v) <- pares ]

apply ::  Eq t => (Chave t) -> [(Chave t, u)] -> Maybe u
apply (chave) [] = Nothing
apply (chave) ((c, v):as)
   | (c == chave) = Just v
   | otherwise = apply chave as
```

Questão 2 Defina em Haskell o tipo de dados `Listas`, que representa uma linguagem de manipulação de listas de inteiros. Por exemplo, a interpretação do programa

```
X:=[2,4,9,1]; Y:=X; Z:=[ e * 2 | e <- Y, e < 3]; Z
```

dá como resultado a lista `[4,2]`. Como ilustrado, os elementos de `Listas` podem ser listas constantes como `[2,4,9,1]`, atribuições representadas pelo símbolo `:=` e envolvendo uma *string* e um elemento de `Listas`, variáveis como o `Z` do final do programa e o `X` usado na segunda atribuição, composições sequenciais representadas pelo símbolo `;` e envolvendo dois elementos de `Listas`, e compreensões que envolvem um elemento de `Expressão` (como `e * 2`), um elemento de `Predicado` (como `e < 3`), um elemento de `Listas` (no exemplo a variável `Y`, podendo ser qualquer outro elemento), e uma *string* representando cada elemento da lista sendo manipulada (no exemplo `e`). Assuma que os tipos `Expressão` e `Predicado` estão definidos. □