Exercícios sobre Polimorfismo de Tipos

prof. André Rauber Du Bois

Universidade Federal de Pelotas
dubois@inf.ufpel.edu.br

1 Questionário

1. Qual o tipo mais geral (polimórfico) das seguintes funções:

```
head (a:x) = a

tail (a:x) = x

fst (t,u) = t

shift ((a,b),c) = (a,(b,c))
```

2. Qual o tipo mais geral da função `concatena`:

```
concatena [] = []
concatena (x:xs) = x ++ concatena xs
```

3. Qual o tipo mais geral da função `inverte`:

```
inverte [] = []
inverte (x:xs) = inverte xs ++ [x]
```

4. Qual o tipo mais geral da função zipp3:

```
zipp3 [] x y = []
zipp3 x [] y = []
zipp3 x y [] = []
zipp3 (x:xs) (y:ys) (z:zs) = (x,y,z) : zipp3 xs ys zs
```

5. Qual o tipo mais geral da função `mapMaisUm`:

```
mapMaisUm f [] = []
mapMaisUm f (x:xs) = f (x+1) : mapMaisUm f xs
```

6. `foldr` é uma função de alta ordem disponível em Haskell que permite aplicar um operador (ou uma função de dois argumentos) entre os elementos de uma lista. Os argumentos dessa função são: um operador, um valor para devolver no caso de uma lista vazia e uma lista. Por exemplo, a chamada

```
foldr (+) 0 [1,2,3,4]
```

calcula a seguinte expressão:

```
1 + 2 + 3 + 4 + 0
```

A função `foldr` é implementada da seguinte forma:

```
foldr f v []     = v
foldr f v (x:xs) = f x (foldr f v xs)
```

No exemplo apresentado anteriormente (`foldr (+) 0 [1,2,3,4]`) usamos o `foldr` com o seguinte tipo:

```
foldr :: (Int -> Int -> Int) -> Int -> [Int] -> Int
```

o tipo polimórfico do `foldr` é

```
foldr :: (a -> b -> b) -> b -> [a] -> b
```

Você consegue explicar o porquê dessa função possuir esse tipo? Você consegue fazer um exemplo que use tipos diferentes para `a` e `b`?