Primeira Prova de PLC (IF686)

Data: 22 de abril de 2010. **Tempo disponível: 2h00m.** **Prof.:** Fernando Castor

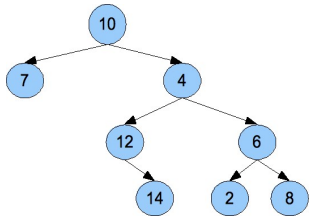
1. (2,0) Responda as perguntas abaixo.

(a) (1,0 ptos.) Quais são as diferenças entre tipificação estática e dinâmica em linguagens de programação? Quais são as vantagens e desvantagens de cada abordagem?

(b) (0,5 ptos.) Que tipo de tipificação Haskell usa? Estática ou dinâmica? E Java? Por que em Haskell, normalmente, não é necessário especificar o tipo de variáveis e funções?

(c) (0,5 ptos.) Por que o uso de produtos cartesianos não é suficiente para definir classes em linguagens de programação orientadas a objetos?

2. (3,5 ptos.) Defina um tipo de dados para árvores binárias (`Tree`) polimórficas, ou seja, cada nó da árvore guarda um valor de um tipo arbitrário (embora todos os nós guardem valores do mesmo tipo). Defina também uma função chamada `dfs` que, dada uma árvore, a percorre em profundidade (*depth-first*) imprimindo o conteúdo de cada nó ao passar por ele. Note que a função deve ser capaz de imprimir árvores para qualquer tipo de valor "imprimível" (não apenas inteiros, reais, etc.). A raiz de cada sub-árvore deve ser impressa **antes** dos nós da sub-árvore à esquerda. Ou seja, dada a árvore



uma chamada a `dfs` passando tal árvore como argumento **deve imprimir** o seguinte (incluindo as vírgulas e o ponto): `10, 7, 4, 12, 14, 6, 2, 8.`

3. (2,5 ptos.) Quais são os tipos das funções abaixo? Mostre como você obteve esses tipos. Se for necessário, identifique as classes dos parâmetros polimórficos. Caso não seja possível determinar o tipo de alguma das funções, explique o porquê.

(a) (1,0 pto.) `map.map.foldr`

(b) (1,5 ptos.) `map.((.) (foldr (++) (foldr (++) [] [[1], [2], [4,5,6], [3]])))`

4. (3,0 ptos.) Defina, usando as construções de Haskell que você aprendeu até agora, um tipo de dados chamado Grafo, que implementa um grafo direcionado cujos nós contém números inteiros (os rótulos dos nós). Defina em seguida uma função chamada `mapEdges` cuja assinatura é a seguinte:

```
mapEdges::(Int->Int->Int)->(Grafo Int)->(Int->[Int])
```

A função `mapEdges` retorna uma nova função que, quando invocada com o valor guardado em um nó `N` do grafo, devolve uma lista cujos elementos são o resultado de aplicar a função recebida como argumento por `mapEdges` (do tipo `Int->Int->Int`) aos valores armazenados nos nós das arestas que partem de `N`. Por exemplo, usando a árvore (`arvore`) da questão 2 como um grafo direcionado, uma chamada `mapEdges (+) arvore` produziria uma função que, ao ser chamada com o argumento `10` (o valor armazenado na raiz de `arvo re`), produziria como resultado a lista `[17, 14]` (resultado de aplicar `(+)` a 10 e 7 e a 10 e 4).