Teoria de Categorias & Programação Funcional
2022.2

Lista de Exercícios 2[†]

Entregar as soluções das questões assinaladas com *
até **18 de outubro no começo da aula**.

A entrega deve ser feita em arquivo(s) `.hs` por email para
`hugonobrega@ic.ufrj.br`

As questões podem ser resolvidas em dupla, mas as duplas não podem ser repetidas da Lista 1.

Questão 1. Na aula 8 fizemos a seguinte definição:

```
data Natural = Zero | Suc Natural
  deriving (Eq, Show)
```

o que faz com que o seguinte aconteça no REPL:

```
> Zero == Zero
True
> Zero == Suc Zero
False
> Suc . Suc . Suc . Suc $ Zero
Suc (Suc (Suc (Suc Zero)))
```

a. Modifique a definição de `Natural` e faça as novas definições necessárias para que o seguinte passe a acontecer no REPL:

```
> Zero

> Suc Zero
#
> Suc . Suc . Suc . Suc $ Zero
####
> -- etc!
```

Dica: lembre-se de que você pode digitar

```
> :i Show
```

[†]Publicada em 3/10

no REPL para obter mais informações sobre a classe de tipos `Show`.

*** b.** Modifique a definição de `Natural` e faça as novas definições necessárias para que o seguinte passe a acontecer no REPL:

```
> Zero
0
> Suc Zero
1
> Suc . Suc . Suc . Suc $ Zero
4
> -- etc!
```

*** c.** Usando o construtor de tipos `Maybe` para implementar a ideia de "operação parcial", faça a definição da operação de subtração entre naturais em Haskell.

d. Faz sentido tentarmos definir uma instância de `Semigroup` para `Natural`? Se sim, defina uma. Se não, justifique por quê.

*** e.** Faz sentido tentarmos definir uma instância de `Functor` para `Natural`? Se sim, defina uma. Se não, justifique por quê.

Questão 2. Em matemática, um *anel* é uma estrutura com assinatura:

- um universo
- duas operações binárias (que aqui denotaremos $\#$ e $\star$)
- uma operação unária (que aqui denotaremos $\underset{\#}{inv}$)
- duas constantes (que aqui denotaremos z e u)

satisfazendo os seguintes axiomas:

1. "$\#$ é associativa"
2. "$\#$ é comutativa", ou seja:

$$\forall x, y \ \ (x \# y = y \# x)$$

3. "z é um elemento neutro para $\#$", ou seja:

$$\forall x \ \ (x \# z = x = z \# x)$$

4. "a operação $\underset{\#}{inv}$ é inversa em relação a $\#$" no seguinte sentido:

$$\forall x \ \ (x \# \underset{\#}{inv}(x) = z = (\underset{\#}{inv}(x)) \# x)$$

5. "$\star$ é associativa"
6. "u é um elemento neutro para $\star$", ou seja:

$$\forall x \ \ (x \star u = x = u \star x)$$

7. "$\star$ é distributiva em relação a $\#$", ou seja:

$$\forall x, y, z \ \ [x \star (y \# z) = (x \star y) \# (x \star z)]$$
$$\forall x, y, z \ \ [(y \# z) \star x = (y \star x) \# (z \star x)]$$

*** a.** Dê uma definição para a classe de tipos (*type class*) dos anéis em Haskell e defina instâncias de `Natural` (da aula 8) e `Integer` para essa classe. Você consegue satisfazer todos os axiomas com as suas instâncias?

Um *corpo* é uma estrutura com assinatura obtida da assinatura dos anéis adicionando-se uma operação parcial unária (que aqui denotaremos $\underset{\star}{inv}$), satisfazendo os seguintes axiomas:

1.–7. todos os axiomas de anéis

8. "$\star$ é comutativa", ou seja:

$$\forall x, y \ \ (x \star y = y \star x)$$

9. $\forall x \ \ (x \neq z \implies x \in \mathrm{dom}(\underset{\star}{inv}))$

10. "$\underset{\star}{inv}$ é uma operação inversa de $\star$", ou seja:

$$\forall x \ \ (x \neq z \quad \implies \quad x \star \underset{\star}{inv}(x) = u = \underset{\star}{inv}(x) \star x)$$

b. Novamente usando `Maybe` para implementar a noção de "operação parcial", dê uma definição para a classe de tipos dos corpos em Haskell e defina instâncias de `Bool` e `Float` para essa classe. Você consegue satisfazer todos os axiomas com as suas instâncias?

***Questão 3.** Implemente o algoritmo de Euclides em Haskell, com assinatura `Integer -> Integer -> Integer`. (Vamos estipular que o mdc de 0 e 0 é 0).

Questão 4. Nesta questão, use a nossa definição de `ArvBin` das aulas 7 e 8.

*** a.** Faça uma função de tipo `ArvBin a -> [a]` que produza uma lista contendo exatamente os valores dos nós não-folha da árvore dada (em qualquer ordem).

*** b.** Defina uma instância de `Foldable` para `ArvBin a`.

c. Faça uma função de tipo `ArvBin a -> Integer`, que indica quantos nós não-folha tem a árvore dada na entrada.

d. Faça uma função de tipo `a -> ArvBin a -> Bool`, que indica se o valor dado está em algum nó da árvore dada ou não.

*** e.** Faça uma função de tipo `a -> ArvBin a -> ArvBin a -> ArvBin a`, que "junte" árvores com uma nova raiz especificada, no seguinte sentido mais preciso: para entradas `x s t`, o retorno é uma árvore que tem como nós exatamente `x` e os nós de `s` e `t`.

*** f.** Expresse a especificação do "sentido mais preciso" do item (e) acima usando expressões do tipo "o valor bla é um nó da árvore ble", os conectivos lógicos usuais e a função especificada no item (d) acima (você não precisar fazer o item (d) para fazer o item atual, pode apenas "citar" o que a função do item (d faz como uma "caixa preta")). [Essa não é uma questão de Haskell!]

*** g.** Vamos chamar de "arrumada" uma árvore binária na qual, para qualquer valor em qualquer nó da árvore, tenhamos que esse valor é

- maior ou igual a todos os valores na subárvore filha da esquerda do nó; e
- menor ou igual a todos os valores na subárvore filha da direita do nó.

Faça uma função que indique (com um `Bool`) se a árvore de tipo `ArvBin a` dada na entrada é "arrumada".