Programação Funcional, 2019.2 **Prova 2.1**

(Turma T12 do Thanos) (points: 108; bonus: $0^\flat$; time: 90')

Nome:

27/11/2019

Regras:

I. Não vires esta página antes do começo da prova.

II. Nenhuma consulta de qualquer forma.

III. Nenhum aparelho ligado (por exemplo: celular, tablet, notebook, *etc.*).[1]

IV. Nenhuma comunicação de qualquer forma e para qualquer motivo.

V. $\forall x\big(\text{Colar}(x) \to \neg\text{Passar}(x, \text{FUN})\big)$.

VI. Use caneta para tuas respostas.

VII. Responda dentro das caixas indicadas.

VIII. Escreva teu nome em *cada* folha de rascunho extra *antes de usá-la*.

IX. Entregue *todas* as folhas de rascunho extra, juntas com tua prova.

X. Nenhuma prova será aceita depois do fim do tempo!

XI. Os pontos bônus são considerados apenas para quem consiga passar sem.[2]

XII. Responda em até 3 dos problemas.[3]

Boas provas!

[1] Ou seja, *desligue antes* da prova.

[2] Por exemplo, 25 pontos bônus podem aumentar uma nota de 5,2 para 7,7 ou de 9,2 para 10,0, mas de 4,9 nem para 7,4 nem para 5,0. A 4,9 ficaria 4,9 mesmo.

[3] Provas com respostas em mais que três problemas não serão corrigidas (tirarão 0 pontos).

(24) **A**

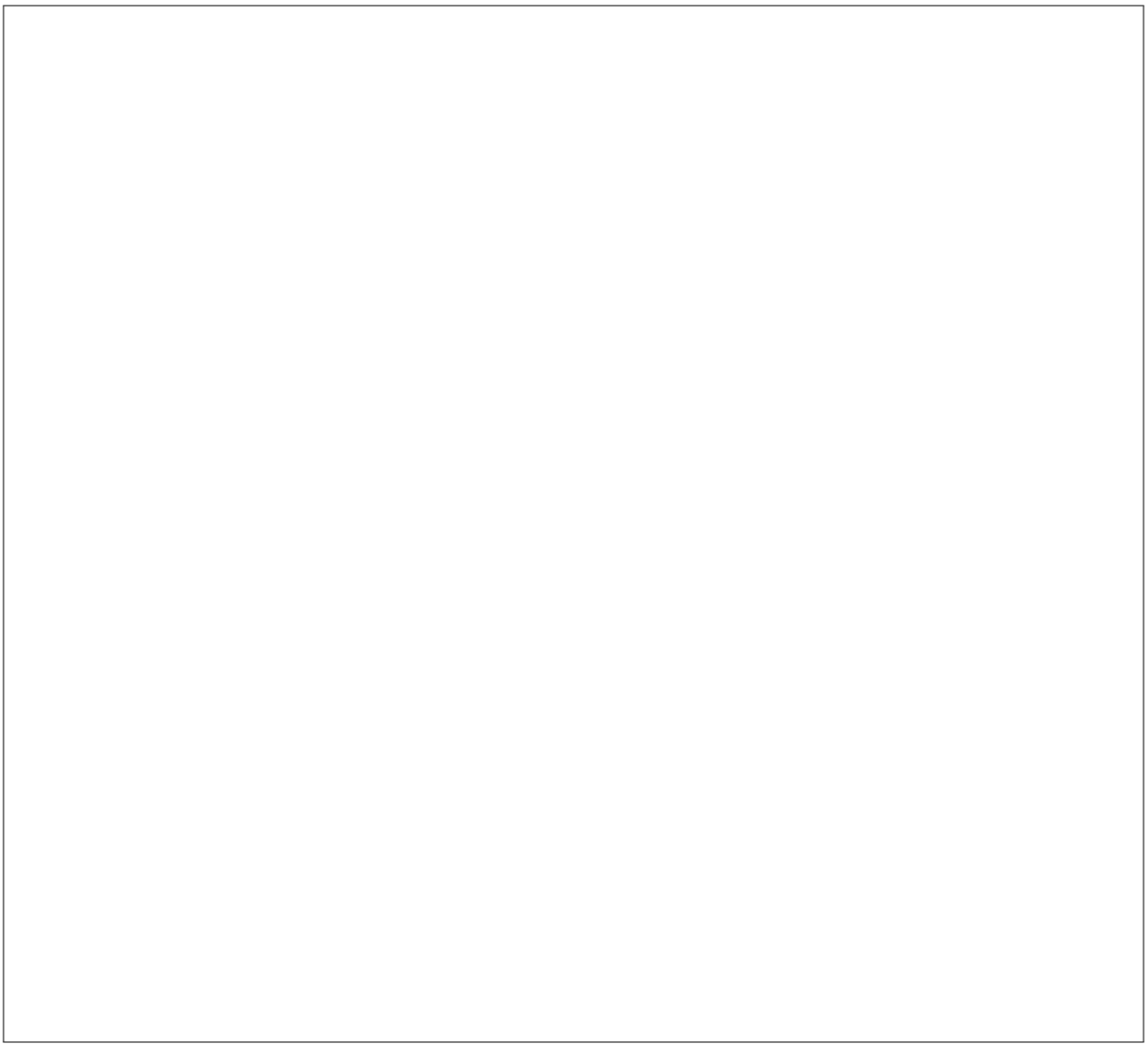
Defina os típos de dados:

```
Nat    :: *                 List   :: * -> *
Maybe  :: * -> *            Either :: * -> * -> *
```

e as funções

```
(+),(*)   :: Nat -> Nat -> Nat
len       :: List a -> Nat
drop      :: Nat -> List a -> List a
map       :: (a -> b) -> List a -> List b
filter    :: (a -> Bool) -> List a -> List a
head      :: List a -> Maybe a
(++)      :: List a -> List a -> List a
takeUntil :: (a -> Bool) -> List a -> List a
```

Definições.



(24) # B

Prove **exatamente um** dos dois teoremas.

(16) **B1.** `(+)` é associativa para todo número finito.

(24) **B2.** `rev` é uma involução quando restrita em listas finitas.

Prova da _____.

(32) # F

Um n00b de Haskell precisou definir as funções

```
sep1 ::      (a -> Bool) -> List a ->      (List a, List a)
sep  :: List (a -> Bool) -> List a -> List (List a, List a)
```

e entregou esse código lamentável:

```
sep1 p  xs = ( filter p xs , filter (not . p) xs )
sep  ps xs = undefined
```

Exemplos de uso desejado da sep:

```
sep  [ even
     , (> 8)
     , prime
     ]
     [ 0 .. 10 ]
   =
     [ ( [0,2,4,6,8,10] , [1,3,5,7,9] )
     , ( [9,10] , [0,1,2,3,4,5,6,7,8] )
     , ( [2,3,5,7] , [0,1,4,6,8,9,10] )
     ]
```

(24) **F1.** Humilhe o n00b definindo a `sep1` **como um fold**, definindo também a própria

```
foldr :: (a -> b -> b) -> b -> List a -> b
```

e explique curtamente qual é a vantagem disso.

HUMILHAÇÃO.

(8) **F2.** Como definarias a `sep`? DEFINIÇÃO.

(36) **L**

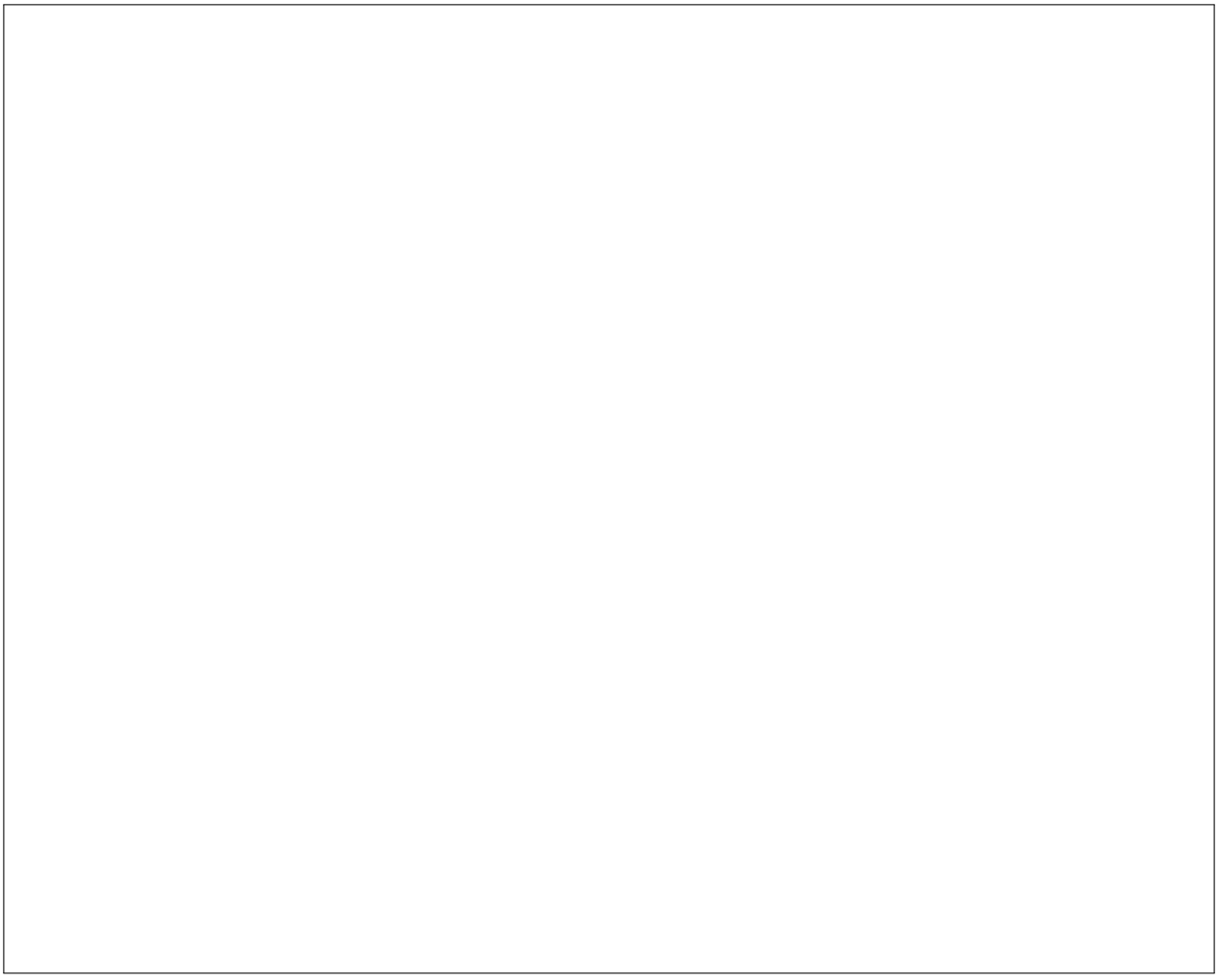
Considere o tipo de arvores seguinte

```
data Tree  =  Leaf Int | Node Tree Tree
```

Demonstre que o número de folhas é sempre um mais que o número de nodes.

Dica: *Comeca definindo funcões que contam o números de folhas e o número de nodes.*

Demonstração.



(36) # R

(4) **R1.** Defina as typeclasses `Functor` e `Applicative`
DEFINIÇÕES.

(12) **R2.** Instancie Listas, e Eithers como Functors e Applicatives. Sobre as Listas, dê duas maneiras diferentes de instanciar como Applicatives e descreva *curtamente* o uso de cada uma.
DEFINIÇÕES.

(12) **R3.** Demonstre as leis de functor para o Either ε.
DEMONSTRAÇÃO.

(8) **R4.** Defina as

```
($>) :: Functor f     => f a ->   b -> f b
(*>) :: Applicative f => f a -> f b -> f b
```

DEFINIÇÕES.

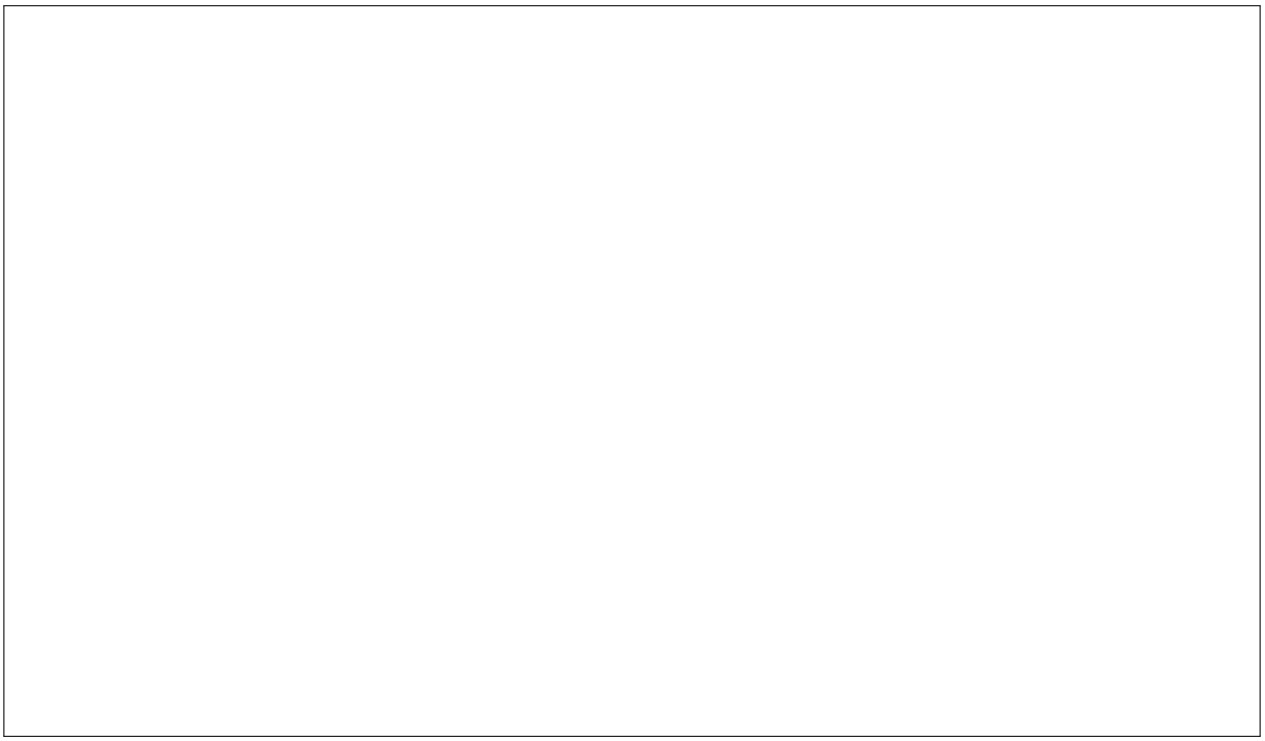
(36) # T

Considere o tipo de arvores

```
data Tree k a = Node k (Tree k a) (Tree k a)
              | Leaf a
```

(14) **T1.** Instancie como Functor:
DEFINIÇÃO.

(22) **T2.** Demonstre que as leis de Functor são satisfeitas.
DEMONSTRAÇÃO.



(36) # M

(20) **M1.** Um applicative `m :: * -> *` pode ser um Monad definindo um apropriado bind ou join:

```
return :: a -> m a
bind   :: m a -> (a -> m b) -> m b
join   :: m (m a) -> m a
```

Demonstre que as duas abordagens são equivalentes, definindo cada um em termos da outra.
DEMONSTRAÇÃO.

(16) **M2.** Instancie Maybes e Eithers como Monads (podes usar tanto a definição baseado no bind tanto no join).
DEFINIÇÕES.

Só isso mesmo.

Rascunho

Rascunho