Programação Funcional, 2024.2 — **Prova 2**

(Turma T12 do Thanos) — (points: 118; bonus: $0^\flat$; time: 84')

Nome:

2025-01-08

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. Escreva em forma facilmente legível.

VII. Responda dentro das caixas indicadas.

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

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

X. Responda em até 2 dos problemas.[2]

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

[2] Provas violando esta regra não serão corrigidas (tirarão 0 pontos).

(22) # G

(10) **G1.** Complete as equações seguintes **com algo interessante:**[3]

```
map f (flatten t) =
sum (replicate n x) =
sum (map (+ k) ns) =
sum (map (· k) ns) =
sorted (map (+ k) ns) =
```

(12) **G2.** Escolha **uma** das equações de **G1** para demonstrar. Precisas definir (corretamente!) todas as funções envolvidas, exceto se foram definidas em outras questões desta prova. Definições.

Demonstração de _____ .

[3] Definição. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(70) # F

Para qualquer α : Type, definimos

```
data Tree a                              data Dir        type Path = List Dir
  Tip  : a → Tree a                        L : Dir
  Fork : Tree a → Tree a → Tree a          R : Dir
```

(12) **F1.** Defina as funções:

```
depth   : Tree a → Nat          flatten : Tree a → List a
ntips   : Tree a → Nat          mirror  : Tree a → Tree a
nforks  : Tree a → Nat          fetch   : Tree a → Path → Maybe a
```

Definições.

(12) **F2.** Defina uma versão eficiente de *flatten* usando uma função auxiliar *flatten′*.

Dica: *Use indução.*

Definição.

(4) **F3.** Enuncie uma equação interessante sobre *flatten* e *mirror*.
Resposta.

(16) **F4.** Enuncie e demonstre uma equação interessante sobre *ntips* e *nforks*.
RESPOSTA.

(10) **F5.** Desenhe os diagramas comutativos que correspondem aos teoremas **F3** e **F4**.
RESPOSTA.

(8) **F6.** Modifique o Tree para representar arvores que carregam informações não apenas nas folhas, mas nos nós também, mantendo a natureza binária.
RESPOSTA.

(8) **F7.** Qual seria um tipo razoável para a *flatten* desse tipo de arvores?
Defina tal *flatten* (não se preocupe com eficiência).
RESPOSTA.

(48) # E

Considere o tipo de expressões aritméticas envolvendo apenas números e divisões inteiras:

```
data Expr
  Val : Int → Expr
  Div : Expr → Expr → Expr
```

(4) **E1.** Dada a função *div* que retorna o quociente explique qual o problema da função seguinte:

```
eval : Expr → Int
eval (Val n) = n
eval (Div u v) = div (eval u) (eval v)
```

RESPOSTA.

(8) **E2.** Defina uma versão segura (total) de *div* (chame de *safediv*).
DEFINIÇÃO.

(24) **E3.** Usando tua função *safediv* defina uma melhor *eval* : Expr → ?.
DEFINIÇÃO.

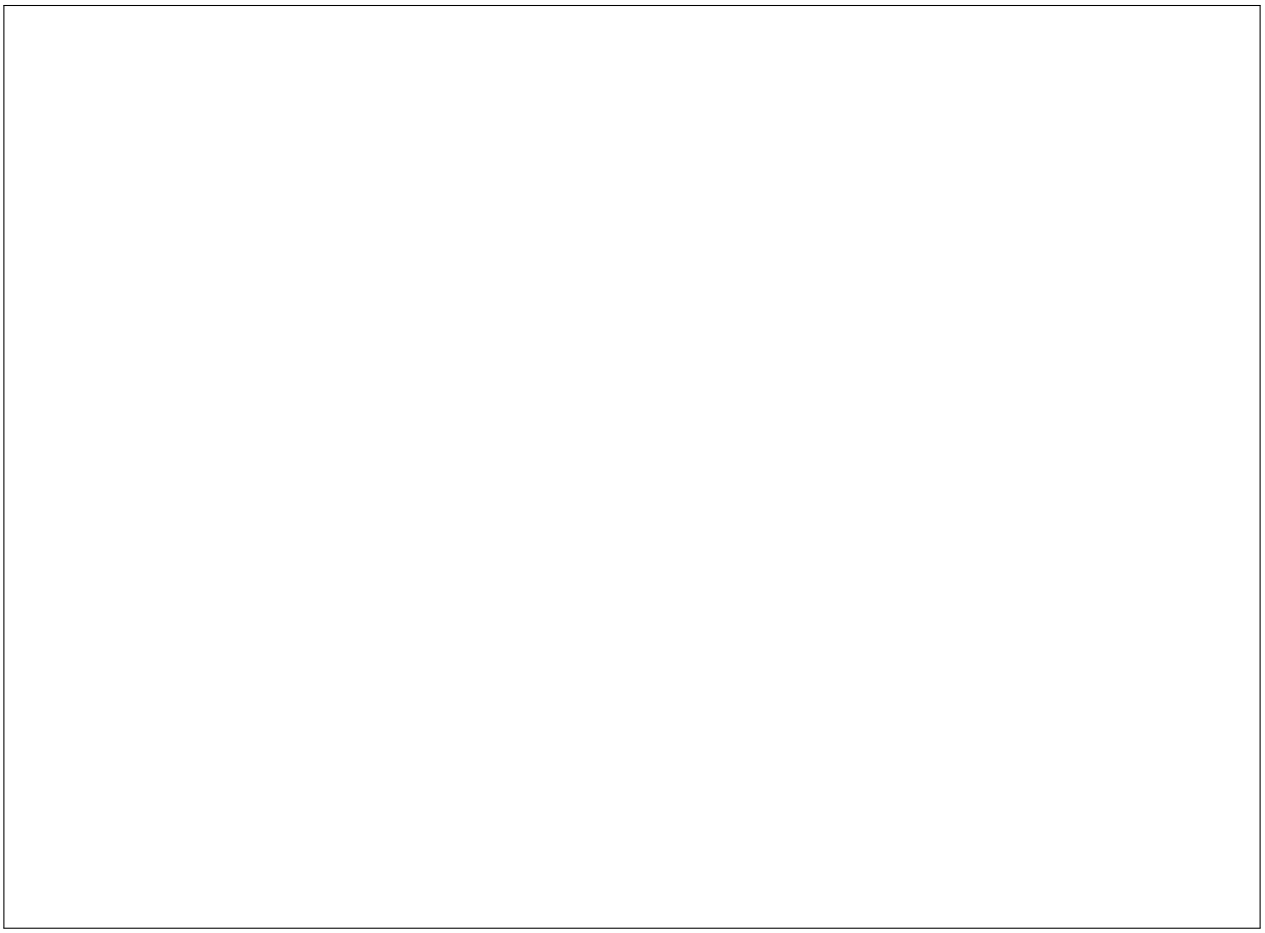
(12) **E4.** Um aluno—talvez não o melhor da turma—respondeu na **E3** assim:

```
eval : Expr → Maybe Int
eval (Val n)   = pure n
eval (Div u v) = pure safediv <*> eval u <*> eval v
```

Compila? Se não, explique o porquê. Se sim, explique porque seria melhor evitar esta definição.
RESPOSTA.

Só isso mesmo.

Lemmata (até 2)



Rascunho