Programação Funcional, 2022.2 — **Prova 3.1**

(Turma S do Thanos) — (points: 120; bonus: $0^\flat$; time: 90')

Nome:

2022-12-10

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 maneira facilmente legível.

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]

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.

(66) # F

(12) **F1.** Defina as typeclasses `Functor` e `Applicative`, e **enuncie** as leis de `Functor`.

(12) **F2.** Um applicative `m :: * -> *` consegue ser um Monad definindo uma *bind* ou uma *join*:

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

Mostre que as duas abordagens são equivalentes, definindo cada uma em termos da outra.
RESPOSTA. Não podes usar notação-do.

(24) **F3.** Instancie Maybe e List como Functors, Applicatives, e Monads[3].
Sobre o List: dê **duas** maneiras diferentes de instanciar como Applicatives...

SOBRE Maybe.

SOBRE List. (Dadas as funções da U1.)

(6) **F4.** ...e descreva **curtamente** a *interpretação computacional* de cada.[4]

(12) **F5.** Defina a *composição Kleisli* (denotada por $(\rightarrowtail)$ em Haskell), com tipo:

$$(\rightarrowtail) : \mathsf{Monad}\ m \Rightarrow (\alpha \to m\ \beta) \to (\beta \to m\ \gamma) \to (\alpha \to m\ \gamma)$$

RESPOSTA. (Podes usar notação-do, mas nesse caso a questão vale metade dos pontos.)

[3] Podes usar tanto a definição baseada no `bind` quanto no `join`.
[4] Deixe claro qual é qual!

(54) # T

```
data Tree α = Tip α | Fork (Tree α) (Tree α)
data Step   = L | R                                   type Path = [Step]
```

(36) **T1.** Levando em consideração os exemplos de uso no quadro, defina recursivamente as funções:

(4×) $\mathit{fmap} : ?$ $\qquad\qquad$ $\mathit{fold} : (\alpha \to \alpha \to \alpha) \to \mathsf{Tree}\ \alpha \to \alpha$

(4×) $\mathit{forks}, \mathit{depth}, \mathit{tips} : \mathsf{Tree}\ \alpha \to \mathsf{Nat}$ $\qquad\qquad$ $\mathit{flat} : \mathsf{Tree}\ \alpha \to \mathsf{List}\ \alpha$

(6×) $\mathit{search} : \alpha \to \mathsf{Tree}\ \alpha \to \mathsf{List\ Path}$ $\qquad$ $\mathit{fetch} : \mathsf{Path} \to \mathsf{Tree}\ \alpha \to \mathsf{Maybe}\ \alpha$

RESPOSTA. (**Não** repita as tipagens na resposta! (Ou seja, escreva apenas da *fmap*.))

(18) **T2.** Mostre como definir as *tips*, *depth*, *flat* como composições, em estilo point-free, usando ambas as *fold*, *fmap* para cada uma delas. Todas as definições **devem** começar com o nome da função, seguido por um '='. Podes usar `where` e λambdas se quiser—mas *sem trollagens!*[5]

Só isso mesmo.

[5] DEFINIÇÃO. Uma resposta é *trollagem* sse Thanos a acha trollagem.

Rascunho