

Cálculo de Programas

3.º Ano de LEI+MiEI (Universidade do Minho)
Ano Lectivo de 2022/23

Teste — 13 de Janeiro de 2023, 14h00–16h00
Salas E1-0.04 + E1-0.08

-
- Esta prova consta de 8 questões que valem, cada uma, 2.5 valores. O tempo médio estimado para resolução de cada questão é de 15 min.
 - Recomenda-se que os alunos leiam a prova antes de decidirem por que ordem querem responder às questões que são colocadas.

PROVA PRESENCIAL INDIVIDUAL SEM CONSULTA (2h)

Questão 1 Recordando da biblioteca `Cp.hs` o isomorfismo $\text{undistl} = [i_1 \times \text{id}, i_2 \times \text{id}]$, use diagramas para:

- descrever o tipo de `undistl` ;
- inferir a propriedade *natural* (ie. “grátis”) da função `distl` que é inversa de `undistl`. (**NB:** tem de formular essa propriedade mas não se pede para a provar analiticamente.)

Questão 2 Sabendo que a igualdade

$$(p? + p?) \cdot p? = (i_1 + i_2) \cdot p? \tag{E1}$$

se verifica, demonstre a seguinte propriedade do condicional de McCarthy:

$$p \rightarrow (p \rightarrow a, b), (p \rightarrow c, d) = p \rightarrow a, d \tag{E2}$$

Questão 3 Considere-se a função

$$h = \text{for loop } (0, 1) \tag{E3}$$

onde $\text{loop } (a, b) = (b, a + b)$. Sabendo que

$$\text{for } g \ i = \llbracket [i, g] \rrbracket \tag{E4}$$

e recorrendo à lei de recursividade mútua, deduza as definições *pointwise* das funções f e g tal que $h = \langle f, g \rangle$.

Questão 4 Mostre que o anamorismo que calcula os sufixos de uma lista

$$\text{suffixes} = \llbracket (g) \rrbracket \text{ where } g = (\text{id} + \langle \text{cons}, \pi_2 \rangle) \cdot \text{out}$$

é a função:

$$\begin{aligned} \text{suffixes } [] &= [] \\ \text{suffixes } (h : t) &= (h : t) : \text{suffixes } t \end{aligned}$$

Questão 5 Recorde o *problema do telemóvel antigo* que foi abordado na primeira ficha das aulas práticas desta disciplina:

(...) For each **list of calls** stored in the mobile phone (eg. numbers dialled, SMS messages, lost calls), the **store** operation should work in a way such that **(a)** the more recently a **call** is made the more accessible it is; **(b)** no number appears twice in a list; **(c)** only the most recent 10 entries in each list are stored.

Tendo-se pedido ao CHATGPT uma solução **pointfree** para estes requisitos, a resposta foi esta, para $\text{store} :: \text{Eq } a \Rightarrow a \rightarrow [a] \rightarrow [a]$:



Can you express the same in point-free Haskell?



Certainly! Here is the point-free version of the `store` function:



Copy code

```
import Data.List (nub)

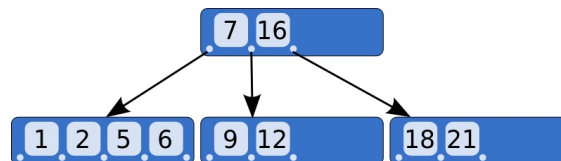
store = (take 10 . nub) . (:)
```

Apesar de impressionante, a resposta tem um erro (apenas!). Identifique-o e diga como se pode corrigir.¹

Questão 6 Uma “B-tree” é uma generalização das árvores binárias do módulo BTree a mais do que duas sub-árvores por nó:

```
data B_tree a = Nil | Block { leftmost :: B_tree a, block :: [(a, B_tree a)] }
```

Por exemplo, a B-tree²



é representada no tipo acima por:

```
t = Block {
  leftmost = Block {
    leftmost = Nil,
    block = [(1, Nil), (2, Nil), (5, Nil), (6, Nil)] }
```

¹CHATGPT usa a função `nub`, para a qual dá a seguinte explicação: “In Haskell, the `nub` function is used to remove duplicate elements from a list. It returns a new list containing only the unique elements from the original list, in the order in which they first appear. For example, `nub [1, 2, 3, 2, 1]` would return `[1, 2, 3]`”.

²Créditos: figura extraída de <https://en.wikipedia.org/wiki/B-tree>.

```

block = [
  (7, Block {
    leftmost = Nil,
    block = [(9, Nil), (12, Nil)]}),
  (16, Block {
    leftmost = Nil,
    block = [(18, Nil), (21, Nil)]})
]

```

Identifique, justificando, o functor de base

$$\begin{cases} B(X, Y) = \dots \\ B(f, g) = \dots \end{cases}$$

que capta o padrão de recursividade da declaração de `B_tree` dada acima, em Haskell, bem como o isomorfismo:

$$\text{in} : B(A, B_tree A) \rightarrow B_tree A.$$

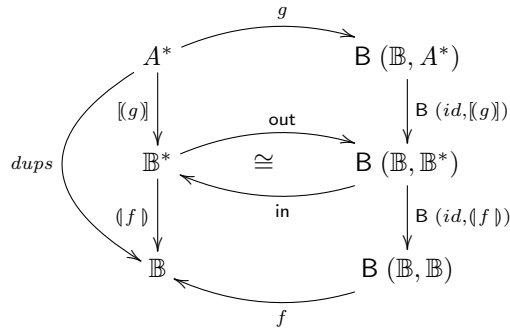
Questão 7 Considere a seguinte definição

```

dups :: (Eq a) => [a] -> B
dups [] = FALSE
dups (h : t) = h ∈ t ∨ (dups t)

```

de uma função que testa se uma lista contém elementos repetidos. Defina-a como um hilomorfismo identificando B e os genes f e g do diagrama seguinte:



Questão 8 Pode mostrar-se que a seguinte variante do tipo “rose tree”

```
data Rose a = L a | R [Rose a]
```

que tem por base $B(f, g) = f + \text{map } g$, forma um mónade

$$X \xrightarrow{u} \text{Rose } X \xleftarrow{\mu} \text{Rose } (\text{Rose } X)$$

onde

$$u = L \tag{E5}$$

$$\mu = \llbracket [id, \text{in} \cdot i_2] \rrbracket \tag{E6}$$

Construa as funções `in` / `out` para este tipo e desenhe o diagrama dos seus catamorfismos. Com base nesse diagrama,

- Converta para Haskell com variáveis a componente μ do referido mónade.
- Mostre que a lei monádica $\mu \cdot u = id$ se verifica.