(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs +\!\!+\ ys)$ = length xs + length ys

map id = id

map $f \circ$ map g = map (f∘g)

filter $p \circ$ map f = map f ∘ filter (p∘f)

map f $(xs +\!\!+\ ys)$ = map f xs ++ map f ys

product ∘ map $(n\,\hat{}\,)$ = (n^) ∘ sum

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.
DEFINIÇÕES.

length : Lα → Nat length [ ] = 0 length (x :: xs) = S (length xs)	map : (α → β) → Lα → Lβ map f [ ] = [ ] map f (x :: xs) = f x :: (map f xs)	filter : (α → Bool) → Lα → Lα filter p [ ] = [ ] filter p (x :: xs) = if p x then x :: filter p xs else filter p xs

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.
DEMONSTRAÇÃO DA map f (xs ++ ys).

Sejam xs, ys : Lα
Indução no xs.
Caso [ ]:
Calculamos:
map f ([ ] ++ ys)
= map f ys [(++).1]
map f [ ] ++ map f ys
= [ ] ++ map f ys [map.1]
= map f ys [(++).1]

Caso (x :: xs):
Calculamos:
map f ((x :: xs) ++ ys)
= map f (x :: (xs ++ ys)) [(++).2]
= f x :: map f (xs ++ ys) [map.2]
= f x :: (map f xs ++ map f ys) [hi]
= (f x :: map f xs) ++ map f ys) [(++).2 ←]
= map f (x :: xs) ++ map f ys [map.2 ←]

[6] DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

$\text{length}\ (xs +\!\!+\ ys) =$ length xs + length ys

$\text{map}\ f \circ \text{map}\ g =$ map (f ∘ g)

$\text{map}\ f\ (xs +\!\!+\ ys) =$ ~~map f xs~~ (map f xs) ++ (map f ys)

$\text{map id} =$ id

$\text{filter}\ p \circ \text{map}\ f =$ map f ∘ filter (p ∘ f)

$\text{product} \circ \text{map}\ (n\,\hat{}\,) =$ map (n^) ∘ sum

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.

Definições.

length :: List α -> Nat
length Nil = 0
length (Cons x xs) = S(length xs)
map :: (α -> β) -> List α -> List β
map _ Nil = Nil
map f (Cons x xs) = Cons (~~f(x)~~ f x) (map f xs)

filter :: (α -> Bool) -> List α -> List α
filter _ Nil = Nil
filter f (Cons x xs) = if (f x) then (Cons x (filter f xs)) else (filter f xs)
fold :: α -> (α -> α -> α) -> List α -> α
fold x _ Nil = x
fold _ f (Cons x xs) = f x (fold x xs)

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

Demonstração da (∀l : List α)[(map f ∘ map g) l = map (f∘g) l]

Seja l : List α
indução em l:
Caso l = []
Calc:
(map f ∘ map g) [] = map f (map g []) (∘.1)
= map f ([]) (map.1)
= [] (map.1)

map (f∘g) [] = [] (map.1)
imediato

Caso l = x :: xs.
Calc:
(map f ∘ map g) (x :: xs) = map f (map g (x :: xs)) (∘.1)
= map f ((g x) :: (map g xs)) (map.2)
= f(g x) :: map f (map g xs) (map.2)
= (f∘g) x :: map f (map g xs) (∘.1)
= (f∘g) x :: ((map f ∘ map g) xs) (∘.1)
= (f∘g) x :: map (f∘g) xs (HI)
= map (f∘g) (x :: xs) (map.2)

Imediato

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

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs ++ ys)$ = length xs + length ys ✓

map $f \circ$ map g = map (f ∘ g) ✓

map f $(xs ++ ys)$ = map f xs ++ map f ys ✓

map id = id ✓

filter $p \circ$ map f = map f (filter (p ∘ f)) ✓ ✗

product ∘ map $(n\,\hat{}\,)$ = ✗ (n^) product

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold. ✓ ✓ ✓

Definições.

map : (α → β) → Lα → Lβ
map f [ ] = [ ]
map f (x :: xs) = f x :: map f xs

filter : (α → β) → Lα → Lβ
filter f [ ] = [ ]
filter f (x :: xs) = if f x then x :: filter f xs
else filter f xs

length : Lα → Nat
length [ ] = 0
length (x :: xs) = S (length xs)

fold : (α → β → β) → β → Lα → β
fold f b [ ] = b
fold f b (x :: xs) = f (fold f b x) (fold f b xs)

typeerror

(18) **L3.** Escolha **exatamente** uma da **primeira coluna** do **L1** para demonstrar.

Demonstração da ?

Seja l : Lα.
Indução no l.
Caso [ ]:
Vale: map (f ∘ g) [ ]
✗ = map (f (g [ ]))
= map f [ ] [map.1]
= [ ] [map.1]
(map f ∘ map g) [ ]
= map f (map g [ ])
= map f [ ] [map.1]
= [ ] [map.1]

Caso (x :: xs)
Seja ls : Lα. ?
Suponho map (f ∘ g) l = (map f ∘ map g) l [(H.I)] ✓
Vale: (map f ∘ map g)(l :: ls)
= map f (map g (l :: ls)) [(∘).2]
= map f (g l :: map g ls) [map.2]
= (f ∘ g) l :: map f (map g ls) [map.2]

F(gl)

map (f ∘ g)(l :: ls)
✗ = map (f (g (l :: ls))) [(∘).2]
= f (g l) :: map (f ∘ g) ls [map.2]
= (f ∘ g) l :: (map f ∘ map g) ls [(H.I)]
= (f ∘ g) l :: map f (map g ls) [(∘).2]

parece que tu trata o map f l como se fosse map (f l) mas na verdade é (map f) l

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

(66) **T**

(8) **T1.** Escreva a regra de inferência que corresponde à indução do tipo LTree $\alpha\ \beta$.

$$\frac{(\forall x{:}\alpha)[\varphi(\text{tip}\,\alpha)] \qquad (\forall l,r{:}\text{Ltree}\,\alpha\,\beta)[\varphi(l)\wedge\varphi(r)\Rightarrow(\forall y{:}\beta)[\varphi(\text{fork}\,y\,l\,r)]]}{(\forall l{:}\text{Ltree}\,\alpha\,\beta)[\varphi(l)]}$$

(12) **T2.** Defina o que precisa para o GTree _ virar um Functor.[7]
DEFINIÇÃO.

fmap: $(\alpha \to \alpha) \to (\text{Gtree}\,\alpha \to \text{Gtree}\,\alpha)$
fmap f (Node x []) = Node (f x) []
fmap f (Node x (t:ts)) = Node (f x) ((fmap f t):(fmap f ts))

β β

parece 7 (sete)

type error

necessárias!

não improvisem tanto nas letras!

(14) **T3.** Levando em consideração os exemplos de uso no quadro, defina recursivamente as funções:

(2×) forks, tips : LTree $\alpha\ \beta \to$ Nat

(3×) join, meet : $(\alpha \to \alpha \to \alpha) \to$ LTree $\alpha\ \alpha \to$ LTree $\alpha\ \alpha \to$ LTree $\alpha\ \alpha$

(4×) balanced : LTree Nat Nat $\to$ Bool

RESPOSTA. **Não** repita as tipagens na resposta!

forks:
forks (tip _) = 0
forks (fork _ l r) = S (forks l + forks r)

tips:
tips (tip _) = S0
tips (fork _ l r) = tips l + tips r

balanced (tip x) = X
balanced (fork x l r) = if ((x > bal l) ∧ (x > bal r)) then true else false

! !

GRRR...

«if b then True else False»

é uma maneira muito complicada de dizer «b».

[7]**Com** tipagem; e **sem** demonstrar as leis necessárias!

(20) **T4.** Demonstre: tips = S ∘ forks. l r   tips l = S(forks l)
" r   "   " r

Podes considerar dados quaisquer dos teoremas da **L1**.

Demonstração.

Seja t : Ltree.
Indução no t. ✓

(∘) ?

Caso tip x:

tips (tip x) = S∘ ✓   [(tips).1]
= S(forks (tip x))   [(forks).1]

Caso (fork l r):

tips (fork_ l r) = tips l + tips r   [(tips).2]
$\boxed{=}$ S(forks l) + S(forks r) ✓   [(HI)]
= S(S(forks l + forks r)   [(Sum).2]
~~=~~ S(tips l + tips r)   [??]
~~=~~ S(fork_ l r)   [(tips).2]

forks

(12) **T5.** Defina funções eval e step para o ArEx.
Demonstração.

eval : ArEx → Nat
eval (tip x) ≝ x ✓
eval (plus l r) ≝ eval l + eval r ✓
eval (times l r) ≝ eval l * eval r ✓

Step: ArEx → ArEx
Step (tip x) = x   — Type Error
Step (plus l r) =

Só isso mesmo.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs +\!\!+\ ys) =$ length xs + length ys

map $f \circ$ map $g =$ map $(f \circ g)$

map $f\ (xs +\!\!+\ ys) =$ map f xs ++ map f ys

map id $=$ id

filter $p \circ$ map $f =$ map f ∘ (filter (p ∘ f))

product $\circ$ map $(n\,\hat{}\,) =$ (n^) ∘ product

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.

DEFINIÇÕES.

```
length : Lα → Nat
length [] = 0
length (x::xs) = S (length xs)
map : (α → β) → Lα → Lβ
map f [] = []
map f (x::xs) = f x :: map f xs

filter (α → Bool) → Lα → Lα
filter _ [] = []
filter f (x::xs) = if f x then x :: filter xs
                   else filter xs
```

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

DEMONSTRAÇÃO DA length (xs ++ ys) = length xs + length ys

Indução no xs.

Caso []:

Calc: length ([] ++ ys)
= length ys (++.1)

length [] + length ys
= 0 + length ys (length.1)
= length ys ((+)-id)

Caso x::xs:

length (x::xs ++ ys)
= length (x :: (xs ++ ys)) (++.2)
= S (length (xs ++ ys)) (length.2)
= S (length xs + length ys) (H.I)
= S (length ys + length xs) ((+)-com)
= length ys + S (length xs) (Nat.2)
* = S (length xs) + length ys ((+)-com)

~~length (x::xs ++ ys)~~
* =
= length (x::xs) + length ys (len.2)

[6] DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(66) **T**

(8) **T1.** Escreva a regra de inferência que corresponde à indução do tipo LTree α β.

$$\frac{(\forall a:\alpha)[\varphi(\mathsf{Tip}\ a)] \qquad (\forall l,r:\mathsf{LTree}\ \alpha\ \beta)[\varphi(l)\ \&\ \varphi(r) \Rightarrow (\forall b:\beta)[\varphi(\mathsf{Fork}\ b\ l\ r)]]}{(\forall t:\mathsf{LTree}\ \alpha\ \beta)[\varphi(t)]}\ \mathrm{IND}^{\varphi}_{\mathsf{LTree}\ \alpha\ \beta}$$

$\varphi : \mathsf{LTree}\ \alpha\ \beta \to \mathsf{Prop}$

(12) **T2.** Defina o que precisa para o GTree _ virar um Functor.[7]
Definição.

```
fmap : (α → β) → (GTree α) → GTree β)
fmap f (Node x [ ]) = Node (f x) [ ]
fmap f (Node x ~~y:ys~~ ts) = Node (f x) (fmap f y)::(map (fmap f) ys)
```

map (fmap f) ts

aqui tu tá reimplementando a própria map_{List}

nomes ruins, são vizinhos de x

(14) **T3.** Levando em consideração os exemplos de uso no quadro, defina recursivamente as funções:

(2×) forks, tips : LTree α β $\to$ Nat

(3×) join, meet : $(\alpha \to \alpha \to \alpha) \to$ LTree α α $\to$ LTree α α $\to$ LTree α α

(4×) balanced : LTree Nat Nat $\to$ Bool

Resposta. **Não** repita as tipagens na resposta!

```
forks (Tip _) = 0
forks (Fork _ l r) = S (forks l + forks r)

tips (Tip _) = S0
tips (Fork _ l r) = tips l + tips r

join f (Tip a) (Fork b l r) = Fork (f a b) l r
join f (Fork a l r) (Tip b) = Fork (f a b) l r
join f (Fork a l r) (Fork b l' r') = Fork (f a b) (join f l l')(join f r r')
join f (Tip a) (Tip b) = Tip (f a b)

meet f (Tip a) (Tip b) = Tip (f a b)
meet f (Tip a) (Fork b l r) = Tip (f a b)
meet f (Fork a _ _) (Tip b) = Tip (f a b)
meet f (Fork a l r) (Fork b l' r') = Fork (f a b) (meet f l l')(meet f r r')

balanced (Tip _) = True
balanced (Fork a l r) = f.
```

[7]**Com** tipagem; e **sem** demonstrar as leis necessárias!

(20) **T4.** Demonstre: tips = S ∘ forks.

Podes considerar dados quaisquer dos teoremas da **L1**.

DEMONSTRAÇÃO.

Seja $t : LTree\ \alpha\ \beta$.

Indução no t.

Caso (t = Tip _):

Calculamos:

tips (Tip _) = S O. [tips.1]

(S ∘ forks)(Tip _) = S (forks (Tip _)) [(∘).1]

já tem!

= S O. [forks.1]

Caso (t = Fork _ ℓ r): Sejam ℓ, r : LTree α β.

(Suponha tips ℓ = (S ∘ forks) ℓ & tips r = (S ∘ forks) r. (HI))

Calculamos: t

tips (Fork _ ℓ r) = tips ℓ + tips r [tips.2]

= (S ∘ forks) ℓ + tips r [(HI).1]

= (S ∘ forks) ℓ + (S ∘ forks) r [(HI).2]

já que ocorre, melhor usar nome mesmo

= S (forks ℓ) + (S ∘ forks) r [(∘).1]

= S (forks ℓ) + S (forks r) [(∘).1]

= S (S (forks ℓ) + (forks r)) [(+).2]

= S (S (forks ℓ + forks r)) [succ-add]

= S (forks (Fork _ ℓ r)) [forks.2]

= (S ∘ forks)(Fork _ ℓ r). [(∘).1]

(12) **T5.** Defina funções eval e step para o ArEx.

DEMONSTRAÇÃO.

eval : ArEx → Nat

eval (Tip a) = a

eval (Plus ℓ r) = eval ℓ + eval r

eval (Times ℓ r) = eval ℓ · eval r

~~step : ArEx → ArEx~~

~~step (Plus (Tip a) (Tip b)) = Tip (a + b)~~

~~step (Times (Tip a) (Tip b)) = Tip (a · b)~~

~~step : ArEx → ArEx~~

~~step (Plus ℓ r)~~

~~ℓ = (Tip a) && r = (Tip b) = Tip (a + b)~~

~~ℓ = (Plus ℓ' r') && r = ... = step ℓ~~

~~ℓ = (Times ℓ' r') && r = ... = step ℓ~~

~~(Tip a) && r ... = step r~~

otherwise

~~step (Times ℓ r)~~

~~ℓ = (Tip a) && r = (Tip b) = Tip (a · b)~~

~~ℓ =~~

step : ArEx → ArEx

step (Plus (Tip a) (Tip b)) = Tip (a + b)

step (Plus (Plus ℓ r) _) = step (Plus ℓ r)

step (Plus (Times ℓ r) _) = step (Times ℓ r)

step (Plus _ r) = step r

step (Times (Tip a) (Tip b)) = Tip (a · b)

step (Times (Plus ℓ r) _) = step (Plus ℓ r)

step (Times (Times ℓ r) _) = step (Times ℓ r)

step (Times _ r) = step r

step (Tip a) = (Tip a)

x

x

Só isso mesmo.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs +\!\!+\, ys) =$ length xs + lenght ys	map id $=$ id
map $f \circ$ map $g =$ map(f∘g)	filter $p \circ$ map $f =$ map f ∘ filter (p∘f)
map f $(xs +\!\!+\, ys) =$ map f xs ++ map f ys	product ∘ map $(n\,\hat{}\,) =$ (n^) ∘ product

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.

DEFINIÇÕES.

lenght : Lα → Nat lenght [] = 0 lenght (x::xs) = S (lenght xs)	filter : (α → Bool) → Lα → Lα filter f [] = [] filter f (x::xs) = if (f x) then x :: (filter f xs) else filter f xs	fold : (α → β → β) → Lα → Lβ fold f [] = [] fold f (x:xs) = :(
map : (α → β) → Lα → Lβ map f [] = [] map f (x::xs) = f x :: (map f xs)		

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

DEMONSTRAÇÃO DA length (xs ++ ys) = lenght xs + lenght ys

Seja ys
Indução no xs
Caso []
Calc length (xs ++ ys)
= length ([] ++ ys)
= length ys [++.1]
Calc length xs + length ys
= length [] + length ys
= 0 + length y [length.1]
= length ys ["+.1"]

Caso Cons

Seja ns t.q. length (ns ++ ys) = length ns + length ys (Hi)
Calc length ((n :: ns) ++ ys)
= length (n :: (ns ++ ys)) [++.2]
= S(length (ns ++ ys)) [length.2]
Calc length (n :: ns) + length ys
= S(length ns) + length ys [length.2]
= S (length ns + lenght ys) ["+.2"]
= S (length (ns ++ ys)) [Hi]
∎

[6]DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(42) **L**

(12) **L1.** Complete as igualdades seguintes com algo interessante:[6]

length $(xs ++ ys)$ = (length xs) + (length ys)

map $f \circ$ map g = map $(f \circ g)$

map f $(xs ++ ys)$ = (map f xs) ++ (map f ys)

map id = id

filter $p \circ$ map f = map $f \circ$ filter $(p \circ f)$

product $\circ$ map $(n\,\hat{}\,)$ =

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.

opcional parênteses.

Definições.

length : L α → Nat length [] = 0 length x::xs = S (length xs)	map : $(\alpha \to \beta) \to (L\alpha \to L\beta)$ map _ [] = [] map f x::xs = $(f\ x)$::(map f xs)
filter : $(\alpha \to$ Bool$) \to L\alpha \to L\alpha$ filter _ [] = [] filter f x::xs = | $f\ x$ = True = x::(filter f xs) | otherwise = filter f xs	fold : $(\alpha \to \beta \to \beta) \to \beta \to (L\alpha \to \beta)$ fold _ b [] = b fold f b x::xs = f x (fold f b xs)

« b = True »

apenas « b »!

espaços de aplicação. (Ficou apertado, sry) rlx

(18) **L3.** Escolha exatamente uma $\boxed{\textbf{da primeira coluna}}$ do **L1** para demonstrar.

Demonstração da map id = id.

é negrito, até

:(

Seja ls : Lα. Por indução no ls.

Caso ls = []. Calculamos:

map id [] = [] [(map.1)]

= id [] [(id).1]

Caso ls = k::ks. Calculamos:

map id (k::ks) = (id k)::(map id ks) [(map).2]

= (id k)::(id ks) [(HI)]

= k::(id ks) [(id)]

= k::ks [(id)]

= id (k::ks). [id]

∎

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

(66) **T**

(8) **T1.** Escreva a regra de inferência que corresponde à indução do tipo LTree α β.

$$\frac{(\forall a:\alpha)[\varphi(\mathsf{Tip}\ a)] \qquad (\forall l, r : \mathsf{LT}\ \alpha\ \beta)[\varphi(l) \land \varphi(r) \Rightarrow (\forall b:\beta)[\varphi(\mathsf{Fork}\ b\ l\ r)]]}{(\forall t : \mathsf{LT}\ \alpha\ \beta)[\varphi(t)]}\ \mathsf{LT}\alpha\beta\ \mathsf{IND}\ \varphi$$

(12) **T2.** Defina o que precisa para o GTree _ virar um Functor.[7]

a [ ]

a (x::xs

DEFINIÇÃO.

use map

```
mapt : (α → β) → (GT α → GT β)
mapt f (Leaf a []) = Leaf (f a) []
mapt f (Leaf a (x::xs)) = Leaf (f a) [mapt f x :: map (mapt f) xs]
```

Node!

(14) **T3.** Levando em consideração os exemplos de uso no quadro, defina recursivamente as funções:

(2×) forks, tips : LTree α β $\to$ Nat

(3×) join, meet : $(\alpha \to \alpha \to \alpha) \to$ LTree α α $\to$ LTree α α $\to$ LTree α α

(4×) balanced : LTree Nat Nat $\to$ Bool

RESPOSTA. **Não** repita as tipagens na resposta!

```
forks (Tip _) = 0
forks (Fork _ l r) = S(forks l + forks r)

tips (Tip _) = S0
tips (Fork _ l r) = tips l + tips r

join f (Tip a) (Tip b) = Tip (f a b)
join f (Fork a l r) (Fork b s t) = Fork (f a b) (join f l s) (join f r t)
join f (Tip a) (Fork b r t) = Fork (f a b) r t
join f (Fork a l r) (Tip b) = Fork (f a b) l r

meet f (Tip a) (Fork b _ _) = Tip (f a b)
meet f (Fork a _ _) (Tip b) = Tip (f a b)
meet f (Tip a) (Tip b) = Tip (f a b)
meet f (Fork a l r) (Fork b s t) = Fork (f a b) (meet f l r) (meet f s t)
```

[7] **Com** tipagem; e **sem** demonstrar as leis necessárias!

(20) **T4.** Demonstre: $\text{tips} = \mathsf{S} \circ \text{forks}$.

Podes considerar dados quaisquer dos teoremas da **L1**.

DEMONSTRAÇÃO.

Seja $t : LT\,\alpha\,\beta$. -- α, β : Type , $t : LT\,\alpha\,\beta \vdash$ tips t = (S o forks) t

Indução no t.

caso (Tip a):

calculamos:

tips (Tip a)

= S0

(S o forks)(Tip a)

= S (forks (Tip a))

= S0

✓

caso (Fork b l r)

calculamos:

tips (Fork b l r)

= tips l + tips r

(S o forks) (Fork b l r)

= S(forks (Fork b l r))

= SS(forks l + forks r)

= S(forks l) + S(forks r)

= (S o forks) l + (S o forks) r

= tips l + (S o forks) r   [(hi)]

= tips l + tips r

justificativas!

(12) **T5.** Defina funções eval e step para o ArEx.
DEMONSTRAÇÃO.

eval (Tip a) = a

eval (Plus l r) = eval l + eval r

eval (Times l r) = eval l · eval r

✓

Type error!

step (Tip _) = 0

step (Plus l r) = S(step l + step r)

step (Times l r) = S(step l + step r)

X

Só isso mesmo.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs +\!\!+\ ys)$ = length xs + length ys

map id = id

map $f \circ$ map g = map(f∘g)

filter $p \circ$ map f = filter p (map f)

map f $(xs +\!\!+\ ys)$ = (map f xs) ++ (map f ys)

product $\circ$ map $(n\,\hat{}\,)$ = (n^)∘product

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.
DEFINIÇÕES.

length : L α → Nat
length [ ] = 0
length (x::xs) = S (length xs)
map : (α → β) → L α → L β
map f [ ] = [ ]
map f (x::xs) = (f x) :: (map f xs)

filter : (α → Bool) → L α → L α
filter f [ ] = [ ]
filter f (x::xs) = if (f x) then (x :: filter xs) else (filter xs)
fold : (α → β → β) → β → (L α → β)
fold _ b [ ] = b
fold f b (x::xs) = f x (fold b xs)

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.
DEMONSTRAÇÃO DA map f ∘ map g = map (f∘g)

Seja α : Type.
Indução no l.
caso l = [ ]
Calc:
(map f ∘ map g) [ ] = map f (map g [ ]) [∘.1]
= map f [ ] [map.1]
= [ ] [map.1]
= map (f∘g) [ ] [map.1]

Caso l = (x :: xs)
Suponha (map f ∘ map g) xs = map (f∘g) xs. (H.I.)
Calc: (map f ∘ map g) (x::xs) = map f (map g (x::xs)) [∘.1]
= map f (g x :: (map g xs)) [map.2]
= ((f (g x)) :: (map f (map g xs))) [map.2]

Calc: map (f∘g) (x::xs) = ((f∘g) x) :: (map (f∘g) xs) [map.2]
= ((f (g x)) :: (map (f∘g) xs)) [∘.1]
= ((f (g x)) :: ((map f ∘ map g) xs)) [H.I.]
= ((f (g x)) :: (map f (map g xs))) [∘.1]

∎

[6]DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(66) T

solto s

(8) **T1.** Escreva a regra de inferência que corresponde à indução do tipo LTree α β.

$$\frac{(\forall a : \alpha)[\varphi(\mathrm{Tip}\ \alpha\ \beta)] \qquad (\forall \beta)[\varphi(l)\ \&\ \varphi(r) \Rightarrow \varphi(\mathrm{Fork}\ \beta\ l\ r)]}{(\forall t)[\varphi(t)]}$$

melhor

(12) **T2.** Defina o que precisa para o GTree _ virar um Functor.[7]
DEFINIÇÃO.

nope!

Precisa (de um α) e uma mapGT. $[GT\alpha] \to GT\alpha$ TYPE $\to$ TYPE	mapGT :: $(\alpha \to \beta) \to GT\alpha \to GT\beta$ mapGT f (Node x []) = Node (f x) [] mapGT f (Node x xs) = Node (f x) (mapList(*) (mapGT f) xs)

(14) **T3.** Levando em consideração os exemplos de uso no quadro, defina recursivamente as funções:

(2×) forks, tips : LTree α β $\to$ Nat
(3×) join, meet : $(\alpha \to \alpha \to \alpha) \to$ LTree α α $\to$ LTree α α $\to$ LTree α α
(4×) balanced : LTree Nat Nat $\to$ Bool

RESPOSTA. **Não** repita as tipagens na resposta!

Forks:
Forks Tip _ = 0
Forks Fork l r = S (Forks l + Forks r)

Tips:
Tips Tip _ = S0
Tips Fork l r = Tips l + tips r

Join:
Join f (Fork x l_1 r_1) (Fork y l_2 r_2) = Fork (f x y) (Join f l_1 l_2) (Join f r_1 r_2)
Join f (Tip x) (Fork y l r) = Fork (f x y) l r
Join f (Fork x l r) (Tip y) = Fork (f x y) l r

meet:
meet f (Fork x l_1 r_1) (Fork y l_2 r_2) = Fork (f x y) (meet f l_1 l_2) (meet f r_1 r_2)
meet f (Tip x) (Fork y _ _) = Tip (f x y)
meet f (Fork x _ _) (Tip y) = Tip (f x y)

Balanced:
balanced Fork x (Fork y l_1 r_1) (Fork z l_2 r_2)
| x ≥ y & x ≤ z = ...
| otherwise = False

:P

T2(*) mapList :: $(\alpha \to \beta) \to [\alpha] \to [\beta]$
mapList f [] = []
mapList f x:xs = (f x) : (mapList f xs)

(veja Zulip!)

[7] **Com** tipagem; e **sem** demonstrar as leis necessárias!

(20) **T4.** Demonstre: tips = S ∘ forks.

Podes considerar dados quaisquer dos teoremas da **L1.**

DEMONSTRAÇÃO.

Seja lt : LTree. ✓
Indução no lt.
Caso lt = Tip x:
    Logo Tips lt = S0.
    Logo Forks lt = 0.
    Logo Tips lt = S(Forks lt).

?!
calcule!!

Caso lt = Fork x l r:
    Suponha que Tips l = S(Forks l) & Tips r = S(Forks r)
    Logo Forks lt = S(Forks l + Forks r)
    Calculemos:
        Tips lt = Tips l + Tips r
                = S(Forks l) + S(Forks r)
                = S(S(Forks l + Forks r)
                = S(~~Forks~~ forks x l r))
                = S(Forks lt).

justifique!

✓

(12) **T5.** Defina funções eval e step para o ArEx.
DEMONSTRAÇÃO.

```
eval :: ArEx -> Nat
eval (Tip x) = x
eval (Plus x y) = (eval x + eval y)
eval (Times x y) = (eval x · eval y)

Step :: ArEx -> ArEx
Step (Plus (Tip x) (Tip y)) = Tip (x+y)
Step (Times (Tip x) (Tip y)) = Tip (x·y)
Step (Tip x) = Tip x
Step (Plus a b) = Plus (Step a) (Step b)
Step (Times a b) = Times (Step a) (Step b)
```

✓ ✓ ✓ ✗ ✗

assim não vai respeitar sua spec. de efetuar 1 passo só!

Só isso mesmo.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs \mathbin{+\!\!+} ys) =$ length xs + length ys ✓

map id = id ✓

map $f \circ$ map $g =$ map $(f \circ g)$ ✓

filter $p \circ$ map $f =$ map $f \circ$ filter $(p \circ f)$ ✓

map f $(xs \mathbin{+\!\!+} ys) =$ map f xs ++ map f ys ✓

product $\circ$ map $(n\,\hat{}\,) =$ $(n\hat{}) \circ$ product ✗

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold. ✓ ! ✓

Definições.

length : $L\alpha \to$ Nat

length [] = 0

length (x::xs) = S + length xs

map : $(\alpha \to \beta) \to L\alpha \to L\beta$

map f [] = []

map f (x::xs) = []

✗ !

filter : $(\alpha \to$ Bool$) \to L\alpha \to L\alpha$

filter p [] = []

filter p (x::xs) = if p x then x::(filter p xs)

else filter (p xs)

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

Demonstração da map f (xs ++ ys) = map f xs ++ map f ys .

Indução no xs.

Caso []:

map f ([] ++ ys) = map f ys [(++).1]

map f [] ++ map f ys = [] ++ map f ys [(map).1]

= map f ys [(++).1]

Caso (x::xs):

map f ((x::xs) ++ ys) = map f (x::(xs ++ ys)) [(++).2]

= f x :: map f (xs ++ ys) [(map).2] veja tua map.2 !

= f x :: (map f xs ++ map f ys) [H.I]

= (f x :: map f xs) ++ map f ys [(++).2]

= map f (x::xs) ++ map f ys [(map).2] ✓

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

onde precisou?

$xs ++ (ys ++ zs) = (xs ++ ys) ++ zs$

Indução no ys. escolha má

Caso []:

$xs ++ ([] ++ zs) = xs ++ zs$ [(++).1]

$(xs ++ []) ++ zs = xs ++ zs$ [(++).1]

Caso (y :: ys):

$xs ++ ((y :: ys) ++ zs) = xs ++ (y :: (ys ++ zs))$ [(++).2]

$= xs ++ ((y :: ys) ++ zs)$ [H.I].

$= (xs ++ (y :: ys)) ++ zs$??

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs \mathbin{+\!\!+} ys) =$ (lenght xs) + (lenght ys)

map id = id_{Ld}

map $f \circ$ map $g =$ Map(F∘g)

filter $p \circ$ map $f =$ Map F ∘ Filter (P∘F)

map f $(xs \mathbin{+\!\!+} ys) =$ (Map F xs) ++ (Map F ys)

product $\circ$ map $(n\,\hat{}\,) =$ (n^) ∘ sum

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.

DEFINIÇÕES.

Lengnt: List α → Nat
  Lenght [ ] = 0
  Lengnt x::xs = S(lenght xs)
Map: (α → β) → List α → List β
  Map F [ ] = [ ]
  Map F x::xs = F x :: Map F xs

Filter: (α → Bool) → List α → List α
  Filter F [ ] = [ ]
  Filter F x::xs = IF (F x) then (x :: Filter F xs) else (Filter F xs)
Fold: β → (α → β) → List α → β ✗
  Fold N C [ ] = N
  Fold N C x::xs = C x :: Fold N C xs ✗

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

DEMONSTRAÇÃO DA Map F ∘ Map g = Map(F ∘ g)

Seja xs : List α
Por indução em xs: obrigatórias!
Caso [ ]:
  Calculamos: (Map F ∘ Map g)[ ] = Map F(Map g [ ])   [(∘₁)]
  = Map F [ ]   [(Map₁) g]
  = [ ]   [(Map₁) F]
  Calculamos: Map (F∘g) [ ] = [ ]   [(Map₁)(F∘g)]

Caso x::xs:
  Calculamos: (Map F ∘ Map g)(x::xs) = Map F (Map g (x::xs))   [(∘₁)]
  = Map F (g x :: Map g xs)   [(Map₂)]
  = F (g x) :: Map F (Map g xs)   [(Map₂)]
  = f(g x) :: (Map F ∘ Map g) xs   [(∘₁)]
  = f(g x) :: Map (F∘g) xs   [H.I]
  = Map (F∘g) x::xs   [(Map₂)]

miracle step!

[6]DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

Fold [ ] = N
Fold x xs = x

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs +\!\!+\, ys) =$ length xs + length ys

map $f \circ$ map $g =$ map (f ∘ g)

map $f\ (xs +\!\!+\, ys) =$ map f (xs) ++ map f (ys)

map id = id

filter $p \circ$ map $f =$ map f ∘ filter (p ∘ f)

product ∘ map $(n\,\hat{}\,) =$

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.

DEFINIÇÕES.

map : (α → β) → List α → List β

map f [] = []

map f x::xs = f x :: map xs

length : List α → Nat

length [] = 0

length x::xs = S (length xs)

filter : (α → Bool) → List α → List α

filter f [] = []

~~filter f [] =~~

filter f x::xs = if f x then x :: filter xs

else filter xs

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

DEMONSTRAÇÃO DA length (xs ++ ys) = length xs + length ys

Sejam xs, ys : List α

~~Indução no xs~~ indução no ys

~~Base (caso xs = []):~~

~~length ([] ++ ys) = length ys (++).1~~

~~Base (caso ys = []):~~

~~length (xs ++ []~~

Base (caso xs = []):

length ([] ++ ys) = length ys (++).1

= 0 + length ys (+ id L)

= length xs + length ys [?]

Passo indutivo [caso xs = x::xs]:

[6]DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs \mathbin{+\!\!+} ys) =$ length xs + length ys

map $f \circ$ map $g =$ map $(f \circ g)$

map f $(xs \mathbin{+\!\!+} ys) =$ map f xs ++ map f ys

map id $=$ id

filter $p \circ$ map $f =$ map f ∘ filter (p∘f)

product $\circ$ map $(n\,\hat{}\,) =$ map (m^) ∘ product

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.

DEFINIÇÕES.

length : List α → Nat
length [] = 0
length (x::xs) = S (length xs)
filter : (α → Bool) → List α → List α
filter p [] = []
filter p (x::xs) = if p x then x :: filter xs else filter xs

map : (α → α) → List α → List α
map f [] = []
map f (x::xs) = f x :: map f xs

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

DEMONSTRAÇÃO DA map f (xs ++ ys) = map f xs ++ map f ys

Seja xs, ys : List α.
Indução no xs.
Caso []:
calc: map f ([] ++ ys)
= map f ys [(++).1]
map f [] ++ map f ys
= [] ++ map f ys [(map).1]
= map f ys [(++).1]

Caso (x::xs):
Seja xs : List α t.q. map f (xs ++ ys) = map f xs ++ map f ys. (h.I)
Calc:
map f ((x::xs) ++ ys)
= map f (x :: (xs ++ ys)) [(++).2]
= f x :: map f (xs ++ ys) [(map).2]
= f x :: (map f xs ++ map f ys) [(h.I)]
= (f x :: map f xs) ++ map f ys [(++).2]
= map f (x:xs) ++ map f ys [(map).2]

■

[6]DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs +\!\!+\ ys)$ = length xs + length ys

map $f \circ$ map g = map (f∘g)

map f $(xs +\!\!+\ ys)$ = map f xs ++ map f ys

map id = id

filter $p \circ$ map f = filter p ∘ map (p∘f)

product ∘ map $(n\,\hat{}\,)$ =

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.
DEFINIÇÕES.

```
length :: Lα → Nat
length [] = 0
length (x:xs) = S(length xs)
filter :: (α → Bool) → Lα → Lα
filter _ [] = []
filter p (x:xs) = if p x then x : filter xs else filter xs

map :: (α → β) → Lα → Lβ
map _ [] = []
map f (x:xs) = f x : map xs

fold :: (β → α → β) → β → Lα → β
fold _ p [] = p
fold f p (x:xs) = f p x (fold f p xs)
```

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.
DEMONSTRAÇÃO DA map id = id.

X

grr...

Seja $\ell : L\alpha$.
Indução no ℓ.
Caso $\ell = []$:
  map id []
    = []
  id []
    = []
Caso $\ell = (x : xs)$.
  Seja $x : \alpha$.
  Calc:
    map id (x:xs)
      = id x : (map id xs)
      = x : (id xs)   [(h.i)]
      = x : xs
    id (x : xs)
      = x : xs
■

[6]DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs +\!\!+ ys)$ = length xs + length ys

map id = id

map $f \circ$ map g = map $(f \circ g)$

filter $p \circ$ map f = filter p (map f)

map f $(xs +\!\!+ ys)$ = map f xs ++ map f ys

product $\circ$ map $(n\,\hat{}\,)$ =

??

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.
DEFINIÇÕES.

length : L α -> Nat
length [] = 0
length (x:xs) = S(length xs)

map : (α -> β) -> L α -> L β
map f [] = []
map f (x:xs) = f x :: map f xs

filter : (α -> Bool) -> L α -> L α
filter p [] = []
filter p (x:xs) = | p x = x :: filter p xs
| otherwise = filter p xs

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.
DEMONSTRAÇÃO DA map f o map g = map (f o g).

Seja l : L α.
Indução no l.

Caso []:

Calc: (map f ∘ map g) [] = map f (map g []) [∘.1]
= map f [] [map.1]
= [] [map.1]

map (f∘g) [] = [] [map.1].

Caso x:xs:

Calc:
(map f ∘ map g)(x:xs) = map f (map g (x:xs)) [∘.1]
= map f (g x : map g xs) [map.2]
= f(g x) : map f (map g xs) [map.2]
= (f∘g) x : map f (map g xs) [∘.1]
(map f ∘ map g) xs [∘.1]
pulou = (f∘g) x : map (f∘g) xs [H.I]
= map (f∘g) (x:xs) [map.2]

---

[6] DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs ++ ys)$ = length xs + length ys

map id = id

map $f \circ$ map g = map (f ∘ g)

filter $p \circ$ map f = map f (filter f ∘ g)

map f $(xs ++ ys)$ = map f xs ++ map f ys

product ∘ map $(n\,\hat{}\,)$ = (n^) map ∘ product.

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.

DEFINIÇÕES.

length : List α → Nat length [ ] = 0 length x::xs = S(length xs)	map : (α → β) → List α → List β map f [ ] = [ ] map f(x::xs) = f x :: map f xs	filter : (α → β) → List α → List α filter f [ ] = [ ] filter f(x::xs) = if f x then x :: filter f xs else filter f xs

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

DEMONSTRAÇÃO DA length(xs ++ ys) = length xs + length ys

⊙ (∀xs, ys : List Nat)[length(xs ++ ys) = length xs + length ys]

Sejam xs, ys : List Nat.

Indução no xs.

Caso [ ]:

Calculamos:

length([ ] ++ ys) = length ys [(++).1]

length [ ] + length ys = 0 + length ys [length.1]

= length ys. [id-Add]

Caso x :: xs:

Calculamos:

length ((x :: xs) ++ ys) = length (x :: (xs ++ ys)) [(++).2]

= S(length(xs ++ ys)) [length.2]

= S(length xs + length ys) [H.I]

length x :: xs + length ys = S(length xs) + length ys [length.2]

= S(length xs + length ys). (?)

∎

[6] DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(42) **L**

type error

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs \mathbin{+\!\!+} ys) =$ lenght xs ++ length ys	map id = id
map $f \circ$ map $g =$ map (f∘g)	filter $p \circ$ map $f =$ filter map p∘f
map f $(xs \mathbin{+\!\!+} ys) =$ map f xs ++ map f ys	product $\circ$ map $(n\,\hat{}\,) =$ map (n^) ∘ product (n^)

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.

Definições.

lenght : L α → α lenght [] = 0 lenght (x::xs) = S(lenght xs)	filter α → Lα → Lα filter _ [] = [] filter _(x::xs) = if _ = x , x ? otherwise filter _ xs
map : α → Lα → Lα map f [] = [] map f (x::xs) = x : map f xs	

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

Demonstração da map f (xs ++ ys) = map f xs ++ map f ys

Vou demonstrar map f(xs ++ ys) = map f xs ++ map ys.

Indução no xs:

Caso []:

map f ([] ++ ys) = map f (ys)

= map ys ??

map f [] ++ map f ys = [] ++ map f ys

= map f ys

Caso (x::xs) (++ 2)

map f((x::xs) ++ ys) = map f (x:(xs ++ ys))

= map f x : map f (xs ++ ys) (map 2)

? = x : map f (xs ++ ys)

= x : map f xs ++ map f ys

= map f (x:xs) ++ map f ys

map f (x:xs) ++ map ys = x : map f xs ++ map f ys

= x : map f (xs ++ ys)

? = map f x : map f (xs ++ ys)

= map f (x : (xs ++ ys))

= map f((x:xs) ++ ys)

---

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

(42) **L**

type error!

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs +\!\!+\ ys)$ = length xs + length ys

map id = id

* map $f \circ$ map g = map(f ∘ g)

filter $p \circ$ map f = map p ∘ filter (p∘f)

map f $(xs +\!\!+\ ys)$ = map f xs ++ map f ys

product ∘ map $(n\,\hat{}\,)$ = (n^) ∘ product

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.

Definições.

```
map : (α → β) → Lα → Lβ
map f [] = []
map f (x::xs) = f x :: map f xs

filter : (α → Bool) → Lα → Lα
filter p [] = []
filter p (x::xs) = if px then x::(filter p xs)
                   else filter p xs

fold : (α → β) → β → Lα → β
fold f n [] = β
fold f n (x::xs) = fx(fold f n xs)

length : Lα → Nat
length [] = 0
length (x::xs) = S(length xs)
```

(χ → β)

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

Demonstração da *.

Indução no ℓ.

Caso []:

```
(map f ∘ map g) []  = map f (map g [])      (∘)
                    = map f []              (map.1)
                    = []                    (map.1)

map (f ∘ g) []      = []                    (map.1)
```

Caso (x::xs):

```
(map f ∘ map g) (x::xs)
  = map f (map g (x::xs))                   (∘)
  = map f (gx :: map g xs)                  (map.2)
  = f(gx) :: map f (map g xs)               (map.2)
2 = (f ∘ g) x :: (map f ∘ map g) xs         (∘) ×2
  = (f ∘ g) x :: map (f ∘ g) xs             (hi)
  = map (f ∘ g) (x::xs)                     (map.2)
```

---

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

(42) L

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs +\!\!+\ ys)$ = length xs + length ys	map id = id
map $f \circ$ map g = map (f∘g)	filter $p \circ$ map f = map f ∘ filter (p∘f)
map f $(xs +\!\!+\ ys)$ = map f xs ++ map f ys	product $\circ$ map $(n\,\hat{}\,)$ = (n^) ∘ product

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.

DEFINIÇÕES.

length : List α → Nat length [] = 0 length (x::xs) = S (length xs)	fold : (α → β → β) → β → Lα → β fold f x [] = x fold f x (y::ys) = f y (fold f x ys)
map : (α → β) → List α → List β map f [] = [] map f (x::xs) = (f x) :: (map f xs)	filter : (α → bool) → L α → L α filter f [] = [] filter f (x::xs) = if (f x) then x::(filter f xs) else filter f xs

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

DEMONSTRAÇÃO DA length (xs++ys) = length xs + length ys .

Seja ys : List α.
Indução no xs.
Caso [].
Calc: length ([]++ys) = length ys [(++).1]
length [] + length ys = 0 + length ys [(length).1]
= length ys [NA-Id]
~~Logo length([]++ys) = length [] + length ys.~~
Caso (x::xs).
(Seja xs tq length(xs++ys) = length xs + length ys. (HI))
Calc: length ((x::xs)++ys) = length (x :: (xs++ys)) [(++).2]
= S(length (xs++ys)) [(length).2]
= S(length xs + length ys) [HI]
length (x::xs) + length ys = S(length xs) + length ys [(length).2]
= length ys + S(length xs) [NA-Com]
= S(length ys + length xs) [(+).2]
= S(length xs + length ys) [NA-Com]
~~Logo length((x::xs)++ys) = length(x::xs) + length ys.~~

■

[6] DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs +\!\!+ ys)$ = length xs ++ lenght ys

map id = id

map $f \circ$ map g = map (f ∘ g)

filter $p \circ$ map f = map f ∘ (filter p)

map f $(xs +\!\!+ ys)$ = map f xs ++ map f ys

product $\circ$ map $(n\,\hat{}\,)$ = map ((n^) ∘ product)

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.

DEFINIÇÕES.

map : (α → β) → Lα → Lβ
map [] = []
map xs [] = xs
map (x::xs) (y::ys) = map (Lxs ++ Lys)
? ?

length : Lα → α
length [] = 0
length xs = (x::xs)

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

DEMONSTRAÇÃO DA map id.

(Sei uma é simples, mas não sei) :(

[6]DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs \mathbin{+\!\!+} ys)$ = length xs + length ys if xs ≠ ys. ✗ map id = id ✓ ✗

map $f \circ$ map g = MAP (f∘g) ✓ filter $p \circ$ map f ≠ filter (MAP p∘f)

map f $(xs \mathbin{+\!\!+} ys)$ = MAP (f xs) ++ MAP (f ys) ✓ product $\circ$ map $(n\,\hat{}\,)$ ≠ (n^) (product n^)

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold. ✓ ✓

DEFINIÇÕES.

MAP : (α → β) → L α → L β MAP _ [ ] = [ ] MAP f (x xs) = f x (MAP f xs)	filter : (α → Bool) → L α → L β filter _ [ ] = [ ] filter f (x xs) : if f x then x : filter f xs else filter f xs ✓
lenght L α → NAT. lenght [ ] = 0 lenght x xs = S0 + lenght xs S (len xs) !	fold : (Nat → a → a) → a → L α → L β fold _ _ [ ] = [ ] fold 0 _ _ = S0 ✗

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

DEMONSTRAÇÃO DA MAP f ∘ MAP g = MAP (f∘g).

(MAP f ∘ MAP g) l $\stackrel{?}{=}$ MAP (f∘g) l.

INDUÇÃO NO l.

CASO [ ]:

calc.

(MAP f ∘ MAP g) [ ] $\boxed{=}$ [ ] ?? ✗

= MAP (f∘g) [ ] ✓

CASO x xs

(MAP f ∘ MAP g) (x::xs) = f x MAP f xs ∘ g x MAP g xs ?? ✗

= f x g x MAP f xs ∘ MAP g xs ✗

= (f∘g) x MAP f xs ∘ MAP g xs (∘ - 1)

= (f∘g) x MAP (f∘g) xs [HI]

= MAP (f∘g) x xs (2.MAP)

[6] DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs +\!\!+\, ys)$ = LENGTH XS $\boxed{+\!\!+}$ LENGTH YS

map id = id

map $f \circ$ map g = MAP(f o g)

filter $p \circ$ map f = MAP f (FILTER P)

map f $(xs +\!\!+\, ys)$ = MAP f XS ++ MAP f YS

product $\circ$ map $(n$ ^ $)$ = N ^ MAP f (N)

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.
DEFINIÇÕES.

LENGTH :: LIST α → NAT LENGTH [ ] = 0 LENGTH X:XS = S (LENGTH XS)	FILTER :: LIST α → IF ELSE → LIST β FILTER P [ ] = [ ] FILTER P X:XS = IF PX, X ++ FILTER XS ELSE FILTER XS
MAP f :: (α → β) → LIST α → LIST β MAP f [ ] = [ ] MAP f (X:XS) = fX (MAP f XS)	FOLD ::

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.
DEMONSTRAÇÃO DA MAP f (XS ++ YS) = MAP f XS ++ MAP f YS .

SEJA YS LIST α E XS LIST α

INDUÇÃO EM XS:

newlines/indentação!

PARTE MAP f ([ ] ++ YS). CALC. MAP f ([ ] ++ YS) =

= MAP f (YS) [++.1]

PARTE MAP f [ ] ++ MAP f YS.

CALC. MAP f [ ] ++ MAP f YS =

= [ ] ++ MAP f (YS) [MAP.1]

= MAP f (YS) [++.1]

PASSO INDUTIVO.

SEJAM XS, YS : LIST α T.Q. MAP f (XS ++ YS) = MAP f XS ++ MAP f YS

~~PARTE MAP f (X:XS ++ YS)~~:

CALC. MAP f (X:XS ++ YS) =

= MAP f (X:(XS ++ YS)) [++.2]

= fX (MAP f (XS ++ YS)) [MAP.2]

= fX (MAP f XS ++ MAP f YS) [HI]

= MAP f X:XS ++ MAP f YS [MAP.2]

~~PARTE MAP f X:XS ++ MAP f YS~~.

CALC MAP f X:XS ++ MAP f YS =

= fX (MAP f XS) ++ MAP f YS [MAP.2]

= fX (MAP f (XS ++ YS)) [HIP]

= MAP f X:XS ++ YS [MAP.2]

∎

---

[6] DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

$\text{length}\ (xs +\!\!+\ ys) = $ length xs + length ys

$\text{map}\ f \circ \text{map}\ g = $ map (f∘g)

$\text{map}\ f\ (xs +\!\!+\ ys) = $ map f xs ++ map f ys

$\text{map id} = $ Id map

$\text{filter}\ p \circ \text{map}\ f = $ filter ? (map f)

$\text{product} \circ \text{map}\ (n\,^\wedge\,) = $ map(S(n)^)

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.
DEFINIÇÕES.

length : List α → Nat length [] = 0 length x::xs = S(length xs)	map : (α→β) → List α → List β map f [] = [] map f (x::xs) = f x :: map f xs	filter : (α→Bool) → List α → List α filter f [] = [] filter f x::xs = if $\boxed{f x = true}$ grrr... then x :: filter f xs else filter f xs

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.
DEMONSTRAÇÃO DA length(xs ++ ys) = length xs + length ys.

Indução na xs.

BASE:

Calc.:

length([] ++ ys) = length ys

= length ys + 0

= length ys + length []

(justifique!)

??

PASSO INDUTIVO:

Suponha length(xs++ys) = length xs + length ys (hi)

Calc.:

length((x::xs) ++ ys) = length (x::(xs++ys))

= S(length(xs++ys))

= S(length xs + length ys) [hi]

⋮

?

[6]DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs +\!\!+\; ys)$ = length xs + length ys

map $f \circ$ map g = map $(f \circ g)$

map f $(xs +\!\!+\; ys)$ = map f xs ++ map f ys

map id = id

filter $p \circ$ map f = map $f \circ$ filter $(p \circ f)$

product $\circ$ map $(n\,\hat{}\,)$ = $(n\hat{})\circ$ product

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.

DEFINIÇÕES.

map: (α → β) → List α → List β
map f [] = []
map f (x::xs) = (f x)::(map f xs)
filter α → List α → List α
filter p [] = []
filter p(x::xs) = if p x then (x:: filter p xs) else filter p xs

length: List α → α
length [] = 0
length (x::xs) = x + length xs
fold: List α → List α
fold [] = []
fold (x::xs) = x :: fold xs

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

DEMONSTRAÇÃO DA (map f ∘ map g) l = map (f∘g) l

Seja l
indução no l.
Base:
calculamos:

$(\text{map } f \circ \text{map } g)[\,] = \text{map } f(\text{map } g\,[\,])$
$= \text{map } f\,[\,]$
$= [\,]$

$\text{map }(f\circ g)[\,] = (f\circ g)[\,]$
$= f(g[\,])$
$= f[\,]$
$= [\,]$

(just.)

Passo indutivo:
Calculamos:

$(\text{map } f\circ \text{map } g)(x::xs)$
$= \text{map } f(\text{map } g\,(x::xs))$
$= \text{map } f(g\,x :: \text{map } g\,xs)$
$= f(g\,x) :: \text{map } f(\text{map } g\,xs)$
$= (f\circ g)\,x :: (\text{map } f\circ g)\,xs$
$= \text{map }(f\circ g)(x::xs)$

pulou demais

[6]DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs \mathbin{+\!\!+} ys)$ = length xs + length ys

map id = Id

map $f \circ$ map g = map (f∘g)

filter $p \circ$ map f = filter p ++ map(p∘f)

map f $(xs \mathbin{+\!\!+} ys)$ = map f xs ++ map f ys

product $\circ$ map $(n\,\hat{}\,)$ = map(n^) product

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.

DEFINIÇÕES.

lenght : List α → Nat lenght [] = 0 lenght (x::xs) = S(lenght xs)	map : (α → B) → List α → List B map f [] = [] map f (x::xs) = f x :: map f xs	filter : (α → Bool) → List α → List bool filter f [] = [] filter f (x::xs) | f(x) = x :: filter xs | otherwise = filter (xs)

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

DEMONSTRAÇÃO DA map f ∘ map g .

(∀l : List α) [map f ∘ map g = map(f∘g)]

Indução no l

Base:

Seja l = []

Calculamos:

solto

(map f ∘ map g) = map f (map g []) [(∘).1]

= map f [] [(map).1]

= [] [(map).1]

Como map (f∘g) [] ≐ [] pela map.1 Logo map f ∘ map g = map (f∘g) □

o que significa esse (=)?

Caso indutivo:

Suponha map f ∘ map g xs = map (f∘g) xs (H.1)

Calculamos: (map f ∘ map g)(x::xs) = map f (map g (x::xs)) [(∘).1]

= map f (g x :: map g xs) [(map).2]

= f(g x) :: map f (map g xs) [(map).2]

= f(g x) :: map (f∘g) xs [(∘).1]

= f∘g x :: map (f∘g) xs [(∘).1]

= map (f∘g) (x::xs) [(map).2] ∎

[6]DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(66) **T**

(8) **T1.** Escreva a regra de inferência que corresponde à indução do tipo LTree α β

$$\frac{(\forall a:\alpha)[P(\mathsf{Tip}\ a)] \qquad (\forall b:\beta)(\forall l, r: LT\alpha\beta)[P(\mathsf{Fork}\ b\ l\ r)]}{(\forall T: LT\alpha\beta)[P(T)]}\ \mathrm{Ind}_{LT\alpha\beta}$$

(12) **T2.** Defina o que precisa para o GTree _ virar um Functor.[7]
DEFINIÇÃO.

(14) **T3.** Levando em consideração os exemplos de uso no quadro, defina recursivamente as funções:

(2×) forks, tips : LTree α β $\to$ Nat
(3×) join, meet : $(\alpha \to \alpha \to \alpha) \to$ LTree α α $\to$ LTree α α $\to$ LTree α α
(4×) balanced : LTree Nat Nat $\to$ Bool

RESPOSTA. **Não** repita as tipagens na resposta!

```
forks Tip _ = 0
forks Fork b l r = S(forks l + forks r)
tips Tip _ = S0
tips Fork _ l r = tips l + tips r
join φ Tip a Tip a' = Tip (φ a a')          -- desculpa o riscado
join φ Tip a Fork a' l r = Fork (φ a a') l r
join φ Fork a l r Fork a' l' r' = Fork (φ a a') (join φ l l') (join φ r r')
meet φ Tip a Tip a' = Tip (φ a a')
meet φ Tip a Fork a' _ _ = Tip (φ a a')
meet φ Fork a _ _ Tip a' = Tip (φ a a')
meet φ Fork a l r Fork a' l' r' = Fork (φ a a') (meet φ l l') (meet φ r r')

join φ Fork a l r Tip a'
= Fork (φ a a') l r
-- foi mal :P
```

[7]**Com** tipagem; e **sem** demonstrar as leis necessárias!

(20) **T4.** Demonstre: tips = $\boxed{S \circ}$ forks.

Podes considerar dados quaisquer dos teoremas da **L1.**

DEMONSTRAÇÃO.

Seja T : LT α β.

Indução no T.

Caso Tip _ :

Calculamos:

tips Tip _ = S0 [tips.1]

= S(forks ~~Tip~~ Tip _) [forks.1]

Caso Fork _ l r :

Calculamos:

tips (Fork _ l r) = tips l + tips r [tips.2]

= S(forks l) + S(forks r) [hi-l & hi-r]

= S(forks l + forks r) ? [(S)-dis]

X?!

forks (Fork _ l r) = S(forks l + forks r) [forks.2]

cadê o $\boxed{(S\circ)}$?

1+(2+2) = 6 ?

(12) **T5.** Defina funções eval e step para o ArEx.

DEMONSTRAÇÃO.

eval : ArEx -> Nat

~~eval x = if (step x == x) then~~

eval Tip x = x

eval Z = eval (step Z) ✓

step : ArEx -> ArEx

step Tip x = Tip x

~~step Plus x~~

step Plus Tip x Tip x' = Tip (x+x')

step Times Tip x Tip x' = Tip (x · x')

-- ultimo caso :P

step _ = boom

Só isso mesmo.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

length $(xs +\!\!+ ys) =$ xy2 | map id = map

map $f \circ$ map $g =$ map f∘g | filter $p \circ$ map $f =$ p ++ map f

map $f\ (xs +\!\!+ ys) =$ map f xs ++ map f ys | product $\circ$ map $(n^\wedge) =$ map(n^n)

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold.

DEFINIÇÕES.

~~length: retorna o tamanho da List.~~
~~mp: cria uma lista com duas dimensões.~~
~~filter: retira ou separa da List o que for solicitado~~
~~fold: corta a list pela metade~~

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

DEMONSTRAÇÃO DA map f ∘ map g = map f∘g

~~Ex: map X ∘ map X = map ∘ map & X∘X~~
~~= map X~~

~~Ex2: map X ∘ map Y = map ∘ map & X∘Y~~
~~= map X∘Y~~

~~Logo:~~
~~X = f~~
~~Y = g~~
~~map f ∘ map g = map ∘ map & f∘g~~
~~= map f∘g~~

[6]DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(42) **L**

(12) **L1.** Complete as igualdades seguintes **com algo interessante:**[6]

$\text{length}\ (xs +\!\!+\ ys) = $ length xs + length ys

$\text{map id} = $ id

$\text{map}\ f \circ \text{map}\ g = $ map (f∘g)

$\text{filter}\ p \circ \text{map}\ f = $ filter (p∘f)

$\text{map}\ f\ (xs +\!\!+\ ys) = $ (map f xs) ++ (map f ys)

$\text{product} \circ \text{map}\ (n\,\hat{}\,) = $ (n^) ∘ sum

(12) **L2.** Defina recursivamente as funções: length, map, filter, fold. ~

DEFINIÇÕES. tipos!!!

```
length [] = 0
length (x:xs) = S(length xs)

map f [] = []
map f (x:xs) = fx : map f xs

filter p [] = []
filter p (x:xs) = if px
                  then x : filter p xs
                  else filter p xs

fold op n [] = n
fold op n (x:xs) = op (fold op n xs) x
```

(18) **L3.** Escolha **exatamente uma da primeira coluna** do **L1** para demonstrar.

DEMONSTRAÇÃO DA len (xs ++ ys) = len xs + len ys .

Indução em xs.

Caso ([]):

Calculamos len ([] ++ ys) = len ys [(++).1]

= 0 + len ys [(+)-id]

= len [] + len ys [len.1] ← parou!

~~= len xs + len ys [escolha de xs]~~

Caso (x:xs):

(Suponha len(xs ++ ys) = len xs + len ys, como HI.)

Calculamos len ((x:xs) ++ ys) = len (x : (xs ++ ys)) [(++).2]

= 1 + len (xs ++ ys) [len.2]

= 1 + (len xs + len ys) [HI]

pulou (= len (x:xs) + len ys [len.2]

[6] DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.