(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
Data Nat        Nat : Type
  0 : Nat
  S : Nat → Nat

Data List α     List : Type → Type
  Nil : List α
  Cons : α → List α → List α

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

```
len : List α → Nat
len [] = 0
len (x :: xs) = S (len xs)

(++) : List α → List α → List α
[] ++ xs = xs
xs ++ ys = x :: (xs ++ ys)

sum : List Nat → Nat
sum [] = 0
sum (x :: xs) = x + (sum xs)
```

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = d·n + d·m
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs · product ys
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = sum ns + k · length ns
```

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

(36) P

Escolha **até duas** das equações de G para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE length (xs ++ ys) = length xs + length ys ✓.

Sejam xs, ys : List α ✓

Por indução em xs. ✓

BASE -- len([] ++ ys) = len [] + len ys ✓

Calculamos:

len ([] ++ ys)

= len ys [(++).1] ✓

len [] + len ys

= 0 + len ys [(len.1)] ✓

= len ys [(+).com, (+).id] ✓

PASSO INDUTIVO

Suponha len (xs ++ ys) = len xs + len ys (H.I.) ✓

Calculamos:

len ((x :: xs) ++ ys) ✓

= len (x :: (xs ++ ys)) [(++).2] ✓

= S (len (xs ++ ys)) [(len).2] ✓

= S (len xs ++ len ys) [H.I.] ✓

len (x :: xs) + len ys

= S (len xs) + len ys [(len).2] ✓

= S (len xs + len ys) ✓ [(.ass)] X

■

DEMONSTRAÇÃO DE map f (xs ++ ys) = map f xs ++ map f ys ✓

Sejam xs, ys : List α e f : α → β ✓

Por indução em xs. ✓

BASE -- map f ([] ++ ys) = map f [] ++ map f ys ✓

Calculamos:

map f ([] ++ ys)

= map f ys [(++).1] ✓

map f [] ++ map f ys

= [] ++ map f ys [(map).1] ✓

= map f ys [(++).1] ✓

PASSO INDUTIVO ✓

Suponha map f (xs ++ ys) = map f xs ++ map f ys (H.I.)

Calculamos:

map f (x :: xs ++ ys)

= map f (x :: (xs ++ ys)) [(++).2] ✓

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

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

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

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

■ ✓

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree $\alpha\beta$
RESPOSTA.

$$\frac{\varphi(0) \Longrightarrow \varphi(S0) \quad \varphi(p)}{(\forall p)\ \varphi(p) \Longrightarrow \varphi(Sp)}\ \mathrm{IND}^{\mathrm{Tree}\ \alpha\beta}_{\varphi}$$

X

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

```
height : Tree → Nat
height Tree a b = b - 1
height Tree _ 0 = 0
height Tree 0 b = b - 1
```

X

```
tips : Tree → Nat
tips Tree a b = a
tips Tree 0 b = 0
tips Tree a 0 = 0
```

X

Só isso mesmo.

(24) A

filter ℓ x::xs =!

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
data Nat
  O : Nat
  S : Nat → Nat

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

(++)

filter : (α → Bool) → L α → L α
filter _ [] = []
filter ρ [x::xs] =        nunca assim?
  | ρ x == True
  | otherwise

Len : L Nat → Nat
Len [] = O
Len [x::xs] = S(Len [xs])

(++) : L α → L α → L α
[] ++ ys = ys        o que soriu esse pattern-matching?
[x::xs] ++ [y::ys] = x :: [xs] ++ [y::ys]

data List α
[] : List α --Nil
X : [] : α X → List α → List α        ??
  = x :: filter ρ xs
  = filter ρ xs
```

(++)

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

mul_add
d · (n + m) = dn + dm ✓

xs = x::xs'

- map f (xs ++ ys) = f x :: map f (xs' ++ ys) ✗
  product (xs ++ ys) = x * product (xs' ++ ys) ✗
  reverse (xs ++ ys) =
- length (xs ++ ys) = length xs + length ys ✓
  length (map f xs) = length xs ✓
  sum (map (+ k) ns) =

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

length [x::L] + length ys

(36) P S (length [L]) +

(∀x : Nat) (∀l : List nat)

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

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

⊢ length (xs ++ ys) = length xs + length ys

Por indução em xs.

Caso []: -- ALVO: length ([] ++ ys) = length [] + length ys

Calculamos:

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

5k · n + 5k · m
n · 5k + m · 5k
0 (n · k) + n + (m · k) + m
(n · k) + (m · k) + n + m

nomes ruins

Caso [illegible] L: -- (∀x : Nat) (∀L : List Nat) [P(L) ⇒ P(x::L)]
-- H.I.: length (L ++ ys) = length L + length ys
-- ALVO: length ([x::L] ++ ys) = length [x::L] + length ys

Calculamos:

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

length [x::L] + length ys
= S (length L) + length ys [length.2]
= length ys + S (length L) [(+).com]?
= S (length ys + length L) [(+).2]
= S (length L + length ys) [(+).com]

Imediato

DEMONSTRAÇÃO DE _____.

⊢ d · (n + m) = d n + d m

Por indução em d.

Caso 0: -- 0 · (n + m) = 0 · n + 0 · m

Calculamos:

0 · (n + m)
= (n + m) · 0 [(·)-com]
= 0 [(·).1]

0 · n + 0 · m
= n · 0 + m · 0 [(·)-com]
= 0 + 0 [(·).1]
= 0 [(+).1]

cadê?

Caso S k: -- H.I.: k · (n + m) = k n + k m

Calculamos:

S k · (n + m)
= (n + m) · S k [(·)-com]
= [illegible] (n + m) · k + (n + m)
= (k · (n + m)) + (n + m) [(·)-com]
= (k n + k m) + (n + m) [Hip.]
? = k n + k m + n + m [(+)-Assoc]
? = (k n + n) + (k m + m) [(+)-com]
= n · S k + m · S k [(·).2]
Imediato = S k · n + S k · m [(·)-com]

S k · (n + m) =

(19) **I**

Para qualquer α : Type, definimos o tipo

P(Nil)

P(cons)
Se eu receber coisas legais, vou construir coisas legais

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.

type error

$$\frac{P(Tip) \qquad \forall k : tree\ a\ b.\ [P(k) \Rightarrow P(Fork\ P)]}{(\forall t :: tree\ a\ b)[P(t)]}\ Ind_\varphi$$

??

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
Nat : Type
0 : Nat
S : Nat -> Nat
```

```
sum : (Numeral a) => List a -> a
sum [] = 0
sum (x :: xs) = {sum xs} + x
```

```
map : (a -> b) -> List a -> List b
map f [] = []
map f (x :: xs) = f x :: {map xs}
```

```
List : Type -> Type
Nil : List
Cons : a -> List a -> List a
```

```
Len : Integral i -> List a -> i
Len [] = 0
Len (x :: xs) = {Len xs} + 1
```

```
(++) : List a -> List a -> List a
[] ++ ys = ys
(x :: xs) ++ ys = x :: (xs ++ ys)
```

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = (d n) + (d m)
map f (xs ++ ys) = f ((map xs) ++ (map ys))
product (xs ++ ys) = (product xs) + (product ys)
reverse (xs ++ ys) = (Rev xs) ++ (Rev ys)
length (xs ++ ys) = (len xs) + (len ys)
length (map f xs) = f (len (map xs))
sum (map (+ k) ns) = (+ k) (sum (map ns))
```

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

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE $d \cdot (n+m) = dn + dm$.

Indução: ?

Base:

calcule:

$0 \cdot (n+m)$

$= 0n + 0m$ [(0) - com] ??

$= 0 + 0$ [(0) - 1]

$= 0$ [(+) - 1]

Caso $c = Sc$

calculamos

$(Sc) \cdot n + (Sc) \cdot (Sm)$

$= ((c \cdot n) + c) + ((c \cdot m) + c + m)$

$= ?$

Passo Indutivo:

Sejam n, m : Nat

Seja c : Nat

Suponha H.I. [redacted] $\rightarrow c \cdot S(n+m) = (c \cdot n + c \cdot (Sm))$

APP H.I. em c para obter $c \cdot (n+m) = cn + cm$

[redacted]: caso $c = 0$

calculamos:

$0 \cdot n + 0 \cdot (Sm)$

$= 0 + 0$ [(·) - 1]

$= 0$ [(+) - 1]

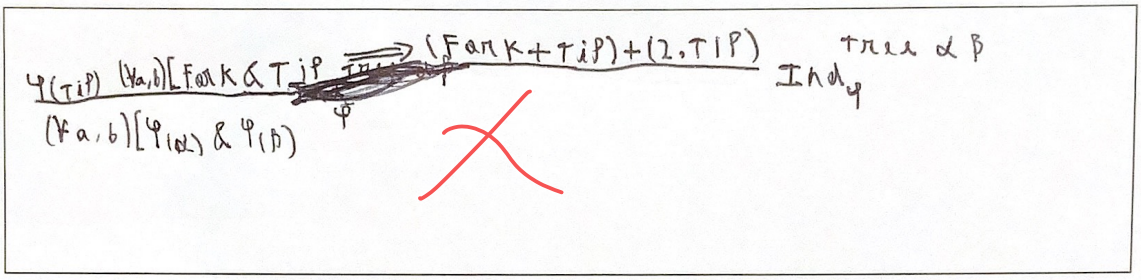
DEMONSTRAÇÃO DE $len\,xs + +\, ys = (len\,xs + (len\,ys)$

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α. β
RESPOSTA.



(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

Só isso mesmo.

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
Data Nat where.
|O = Nat
|S O = Nat → Nat

Data List a where
|Nil = List a
|Cons = a → List a → List a

map :: (a → b) → [a] → [b]
map f [] = []
map f (x:xs) = f x : map f xs

(++) :: [a] → [a] → [a]
[] ++ ys = ys
(x : xs) ++ ys = x : (xs ++ ys)

sum :: [a] → n
sum [] = 0
sum (x : xs) = x + sum xs

len :: [a] → n
len [] = 0
len (x : xs) = S (len xs)
```

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = dn + dm
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs · product ys
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) =
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes. $(\forall xs, ys)$

DEMONSTRAÇÃO DE $[length\ (xs +\!\!+\ ys) = length\ xs + length\ ys]$

Sejam xs, ys : List a ✓
Por indução em xs. ✓
Caso []: ✓
  Calculamos:
    length ([] ++ ys) ✓
    = length ys [(++).1] ✓
    = 0 + length ys [IdL-(+)] ✓
    = length [] + length ys [length.1] ✓
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] ✓
    = (S length xs) + length ys [(+).2] ✗

= len (x : xs) + len ys ✓
-- ↑ faltou espaço :c

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

Seja f : a → β
Sejam xs, ys : List a
Por indução em xs. ✓
Caso []: ✓
  Calculamos:
    map f ([] ++ ys) ✓
    = map f ys [(++).1] ✓
    = [] ++ map f ys [(++).1] ✓
    = map f [] ++ map f ys [map.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 [++.I] ✓
    = map f (x : xs) ++ map f ys [map.2] ✓

✓

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α. β
RESPOSTA.

$$\frac{(\forall\, \alpha)[\varphi(Tip\ \alpha)] \qquad (\forall \beta)[\varphi(Fork\ \beta)]}{(\forall \alpha, \beta)[\,Tree\ \alpha\ \beta \Leftarrow Tip\ \beta \wedge Fork\ \alpha\,]}\ Ind\ \varphi^{\,Tree\ \alpha\ \beta}$$

X

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

```
tips 1 = 2
tips n = 2 + tips (n-1)
```

X

Só isso mesmo.

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

Nat :: type (∀a)[a ≥ 0] a = nat ✗	List :: list nat → type (∀f : list) [f] = list ✗
Sum :: nat → nat → nat (∀x, y, z : nat) x + 0 = x x + y = z ✗	Product nat → nat → nat (∀x, y, z : nat) x · y = z x · 0 = 0 x · 1 = x ✗
Length : List → nat (∀ L : list)(∀ n : nat) [L] = n ✗	(++) :: list nat → list nat → list nat (∀n, i, j : list) [n] + [i] = [j] ✗

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = (d·n) + (d·m) ✓
map f (xs ++ ys) = (map f (xs)) ++ (map f (ys)) ✗
product (xs ++ ys) = (product (xs)) + (product (ys)) ✗
reverse (xs ++ ys) = (reverse (xs)) + (reverse (ys)) ✗
length (xs ++ ys) = (length (xs)) + (length (ys)) ✓
length (map f xs) = length((map f) xs) ✗
sum (map (+ k) ns) = sum(map(ns)) + (map(k)) ✗
```

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

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE $d \cdot (n+m) = d \cdot n + d \cdot m$

$(\forall a, b, c : \text{INT})$ a Seja $c = sc'$ ← essa linha não faz sentido

Indução

Base $((a \cdot (b+0)) = (a \cdot b) + (a \cdot 0))$ [hip]

$a \cdot b = (a \cdot b) + 0$ [(+).1 e (·).1]

$a \cdot b = (a \cdot b)$ [(+).1]

HI $((a \cdot (b + sc')) = a \cdot b + a \cdot sc')$ [hip]

$a \cdot s(b + c') = a \cdot b + a \cdot sc'$ [(+).2]

$a \cdot (b+c') + a = a \cdot b + a \cdot c' + a$ [(·).2]

$a \cdot (b+c') + a = a \cdot (b + c') + a$ [(·)-ass]

Imediato

essas são umas proposições soltas. (isso não "compila").

inferir algo trivial como $a \cdot b = a \cdot b$ ou $0 = 0$ não é notícia boa. Já sabíamos essas por refl.

DEMONSTRAÇÃO DE ~~length (xs ++ ys) = length (xs) + length (ys)~~

Prof length (xs ++ ys) = length (xs) + length (ys)

$(\forall xs, ys : \text{list})$ a seja $ys = sys'$

Base

length (xs ++ 0) = length (xs) + length (0) [hip]

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

HI

length (xs ++ s(ys')) = length (xs) + length (s(ys')) [hip]

length (s(xs ++ ys')) = length (xs) + length (s(ys')) [(+).2]

length (s(xs ++ ys')) = length (s(xs ++ ys')) [(+)-ass]

Imediato

mesma coisa

X

(19) **I**

β : Type

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.

Para qualquer α:Type e β:Type
Base
  Tree α 0 [hiβ]
  Tree α [(+).1]
HI
  Tree α sb'
  Tree α sb' [(+).2]

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

(∀ α,b :Type)
lenght (forK:α)==1 → height = 0
lenght (forK:α)==m height ≤ m

Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
data Nat
  O : Nat
  S : Nat → Nat

data List α
  Nil : List α
  Cons : α → List α → List α

data len : Lα → Nat
  Len [] = O
  Len (α :: xs) = ?

data (++) : Lα → Lα → Lα
  [] ++ Lα = : Lα
  (x :: xs) ++ (y :: ys) = ?

data map : (α → β) → Lα → Lβ

data rev : Lα → Lα
  rev [] = : []
```

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = d·n + d·m
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs ++ product ys
reverse (xs ++ ys) = Rev(Rev(xs ++ ys))
length (xs ++ ys) = len xs + len ys
length (map f xs) =
sum (map (+ k) ns) =
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE $d \cdot (n + m) = d \cdot n + d \cdot m$.

Sejam $d, n : Nat$
Por indução ✓
Caso base: — isso é o que tu quer demonstrar
Logo $d \cdot (n + 0) = d \cdot n + d \cdot 0$ [pela escolha de n]
Logo $d \cdot n = d \cdot n + d \cdot 0$ [(+).1] ✗
Logo $d \cdot n = d \cdot n$ [(·).1]
Imediato
↳ toda essa tinta para chegar em $A = A$. (Que ninguém duvidou, ou se importou.)
Passo indutivo:
Seja $k : Nat$ tal que $k = S n'$.
Logo $d \cdot (n + S n') = d \cdot n + d \cdot S n'$ [HI]
Logo $d \cdot (S (n + n')) = d \cdot n + d \cdot S n'$ [(+).2]

DEMONSTRAÇÃO DE $len\ (xs ++ ys) = len\ xs + len\ ys$.

Seja $xs : List\ a$
Por indução
Caso base:
Logo $len\ (xs ++ [\,]) = len\ xs + len\ [\,]$ [pela escolha de ys]
Logo $len\ xs = len\ xs + 0$
Logo $len\ xs = len\ xs$ [(+).1] ✗
Imediato.
Passo indutivo:
Seja $ks : La$ tal que $ks =: (k :: ks)$
Logo $len\ (xs ++$

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
data Nat
  | O : Nat
  | S_ : Nat -> Nat
```

```
data List a
  | Nil : List a
  | Cons x xs : a -> List a -> List a
```

```
filter :: (a -> Bool) -> [a] -> [a]
filter _ [] = []
filter f (x:xs)
  | f x = x : filter f xs
  | otherwise = filter f xs
```

```
map :: (a -> b) -> [a] -> [b]
map _ [] = []
map f (x:xs) = f x : map f xs
```

```
length :: [a] -> Nat
length [] = O
length (x:xs) = S (length xs)
```

```
reverse :: [a] -> [a]
reverse [] = []
reverse (x:xs) = reverse xs ++ [x]
```

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = (d · n) + (d · m)
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs · product ys
reverse (xs ++ ys) = reverse xs ++ reverse ys
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = sum ns + ?
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

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

Sejam xs, ys : List Nat
por indução em xs.
caso []:
calculamos:

len ([] ++ ys) = len ys [(++).1]
len [] + len ys = 0 + len ys [len.1]
= len ys + 0 [(+)-com]
= len ys [(+).1] ✓

caso (x':xs'):
seja xs' : List Nat tal que len (xs' ++ ys) = len xs' + len ys.
calculamos:

len ((x':xs') ++ ys) = len (x' : xs' ++ ys) [(++).2]
= S (len (xs' ++ ys)) [len.2]
= S (len xs' + len ys) [HI]
= S (len ys + len xs') [(+)-com]
= len ys + S (len xs') [(+).2 ←]
= len ys + len (x':xs') [len.2 ←]
= len (x':xs') + len ys [(+)-com]

Imediato.

DEMONSTRAÇÃO DE Reverse (xs ++ ys) = reverse xs ++ reverse ys

Sejam xs, ys : List Nat.
Por indução em xs. ✓
caso []: ✓
calculamos:

rev ([] ++ ys) = rev ys [(++).1] ✓
rev [] ++ rev ys = [] ++ rev ys [rev.1] ✗
= rev ys + [] [(+)-com] ✗
= rev ys [(+).1]

caso (x':xs'): ✓
seja xs' tal que rev (xs' ++ ys) = rev xs' ++ rev ys. ✓
calculamos:

rev (x':xs') ++ ys) = rev (x' : (xs' ++ ys)) [(++).2] ✓
= rev (xs' ++ ys) ++ [x'] [rev.2] ✓
= (rev xs' ++ rev ys) ++ [x'] [IH] ✓
= (rev xs' ++ [x']) ++ rev ys [(++)-assoc] ✗ ✗
= rev (x':xs') ++ rev ys [rev.2 ←]

Imediato.

LEMMATA

lemma 1

[∀ xs, ys, zs : List α) [(xs ++ ys) ++ zs = xs ++ (ys ++ zs)]

Sejam xs, ys, zs : List Nat.

Por indução em xs.

Caso []:

Calculamos:

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

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

✓

Caso (x' : xs'):

Seja xs' tal que (xs' ++ ys) ++ zs = xs' ++ (ys ++ zs)

Calculamos:

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

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

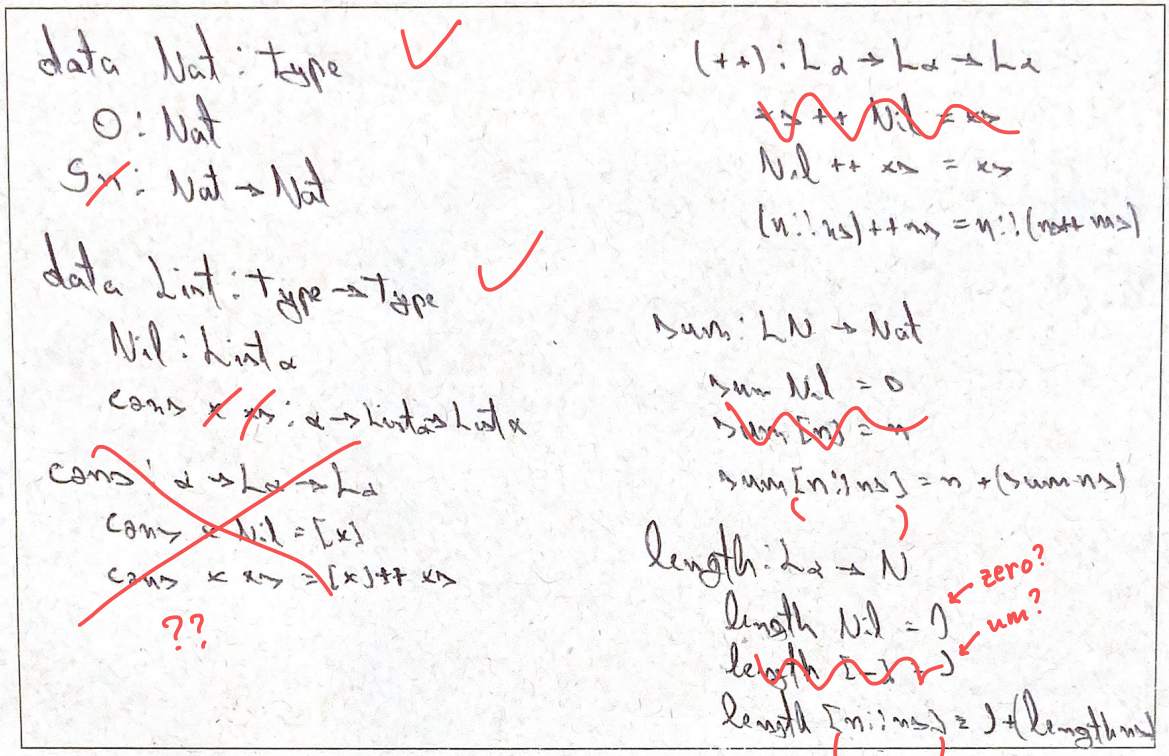
= x' : (xs' ++ (ys ++ zs)) [HI]

= (x' : xs') ++ (ys ++ zs) [(++).2 ⇐]

✓

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.



(21) G

Complete ~~até 8~~ 7 das equações seguintes com algo interessante:[4]

```
d · (n + m) = dn + dm
map f (xs ++ ys) = (map f xs) ++ (map f ys)
product (xs ++ ys) = (product xs) · (product ys)
reverse (xs ++ ys) = (reverse ys) ++ (reverse xs)
length (xs ++ ys) = (length xs) + (length ys)
length (map f xs) = length xs
sum (map (+ k) ns) = (sum ns) + (length ns · k)
```

(2)

[1, 2, 3] k = 3
[4, 5, 6]

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

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE $(\forall xs, ys)[\ product(xs ++ ys) = product\ xs \cdot product\ ys\ ]$.

Sejam xs, ys : [N] Passo Indutivo

Por indução em ys ✗

Base ??

Calculemos ✗

product (xs ++ Nil)

= product xs

Calculemos

product xs · product Nil

= product xs . 1

= product xs

Logo product (xs ++ Nil) = product xs · product Nil

Imediato

DEMONSTRAÇÃO DE $(\forall d, n, m)[\ d(n+m) = dn + dm\ ]$.

Sejam d, n, m : Nat

Por indução em m

Base

Calculemos

d(n + 0)

= d n ✗

= dn ?

Calculemos

dn + d 0

= dn + 0

= dn

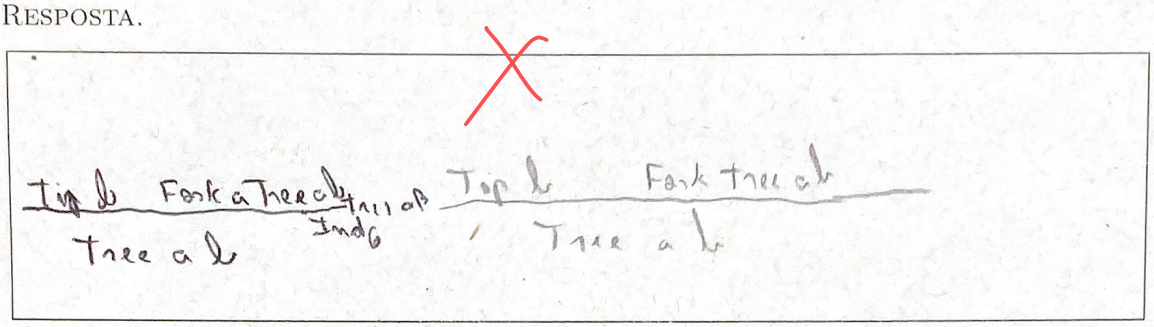
Logo ?

(19) **I**

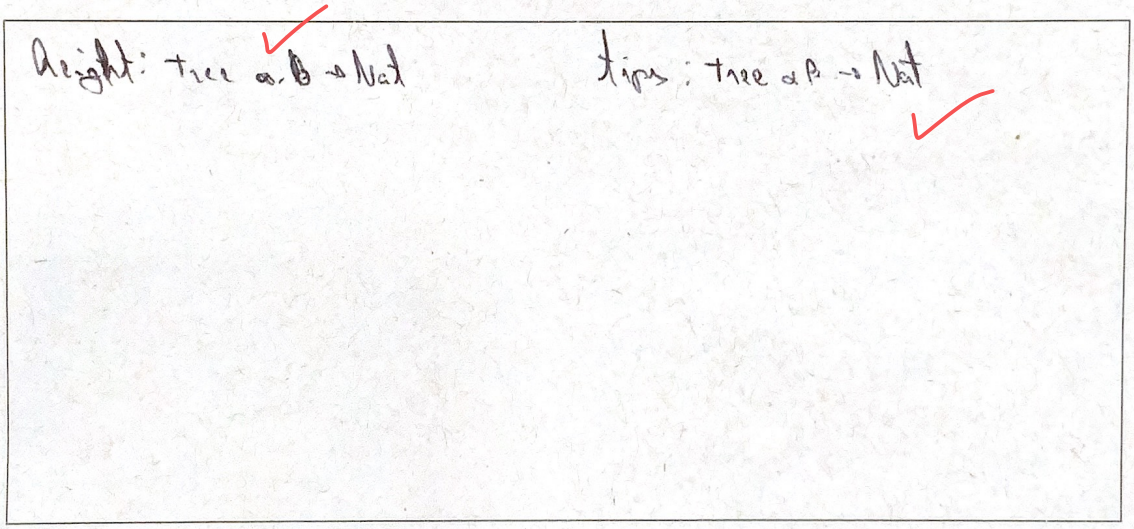
Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.



(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.



Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

é uma lista com um membro

i) sum :: List α → Nat
sum [_] = ~~error "empty list"~~ 0
sum (x::xs) = x + sum xs

ii) product :: List α → Nat
prod [_] = SO
prod (x::xs) = x * prod xs

iii) Nat :: type
O : Nat
SO : Nat
X

iv) length :: List α → Nat
length [_] = O
length (x:xs) = SO + length xs

v) Concat (++) :: List α × List α → List α
[_] ++ (x:xs) = (x:xs)
~~(y:ys) ++ [_] = (y:ys)~~
(y:ys) ++ (x:xs) = [y] ++ [ys ++ [x] ++ xs]
X

VI) ~~map :: a → [~~
reverse :: List α → List α
rev [_] = [_]
~~rev [x] = [x]~~
rev (x:xs) = (reverse xs) ++ [x]

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = d·n + d·m                          ✓
map f (xs ++ ys) = (map f xs) ++ (map f ys)      ✓
product (xs ++ ys) = (prod xs) ++ (prod ys)      X
reverse (xs ++ ys) = (reverse ys) ++ (reverse xs) ✓
length (xs ++ ys) = (length xs) + (length ys)    ✓
length (map f xs) = length xs                    ✓
sum (map (+ k) ns) = sum ns + k*(length ns)      ✓
```

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

(36) P

Escolha **até duas** das equações de G para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE $d \cdot (n+m) = d \cdot n + d \cdot m$.

~~Seja~~ Sejam P, a e b, Vamos demonstrar $P(a+b) = P \cdot a + P \cdot b$

Por indução:

↑ por quê??

- ~~Caso~~ base:

~~0~~ Calculamos:

$0(a+b) = 0 \cdot a + 0 \cdot b$

$= 0$ ((·)-1) ✗

- Passo Indutivo:

H: $P(a+b) = P \cdot a + P \cdot b$

? T: $Sp(a+b) = Sp \cdot a + Sp \cdot b$

Calculamos:

? $= (a+b) \cdot Sp$ ((·)-com)

$= [(a+b) \cdot P] + (a+b)((\cdot)-2)$ Cadê?

$= (P \cdot a + P \cdot b) + (a+b)$ [(·)-com] (H)

$= P \cdot a + a + P \cdot b + b$ [((+)-ass P a P b a),

$= \cancel{S}P(a+b) Sp \cdot a + Sp \cdot b$ (← (·)-2) ((+)com)]

DEMONSTRAÇÃO DE length (xs ++ ys) = length xs + length ys ▨ ?

Calculamos:

length xs + length ys =

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.

Fork α Tip β / Tree $\alpha\beta$ Ind / Tree $\alpha\beta$ φ

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

```
height :: Tree α β → Nat
height tree 0,0 = 0          → height tree 1,0 = 1
height tree α β = S β S (fork
height tree α anyInt = error "Impossible to form"
height tree anyOdd β = error "Impossible to form"
                                          β = 2·q+1
height tree α β = B -( B * height q+1 , (q,1)- euclid (β,2)
```

Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
(++) : La → La → La
[] ++ ys = ys
(x::xs) ++ ys = x::(xs ++ ys)

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

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

```
Nat : type
0 : Nat
S : Nat → Nat

Lista : α → Lista α
Nil : Lα
Cons : α → Lα → Lα

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

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = dn + dm
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs + product ys
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = sum ns + length ns · k
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes

DEMONSTRAÇÃO DE `map f (xs ++ ys) = map xs ++ map ys`

Seja ys : La
Por indução.
Base:
  Seja f : a → β
  Calculamos:
    map f ([] ++ ys) = map f (ys) [(++).1 ys]
                     = [] ++ map f (ys) [(++).1 map f ys]
                     = map f [] ++ map f ys [map.1 f]
    imediato.
Passo indutivo:
  Seja f : a → β
  Seja ks : La
  Suponha map f (ks ++ ys) = map ks ++ map ys : (h.i)
  Seja k : a
  Calculamos:
    map f ((k :: ks) ++ ys) = map f (k :: (ks ++ ys)) [(++).2 k ks ys]
                            = f k :: map f (ks ++ ys) [map.2 f (k ++ ys)]
                            = f k :: (map f ks ++ map f ys) [h.i]
                            = (f k :: map f ks) ++ map f ys [(++).2 (f k) map f ks map f ys]
                            = map f (k :: ks) ++ map f ys [map.2 f k ks : map f ks].
  imediato.

DEMONSTRAÇÃO DE

Seja ys : La
Por indução.
Base:
  Calculamos:
    len ([] ++ ys) = len (ys) [(++).1 ys]
                   = len (ys) + 0 [0-id-r (+) len ys]
                   = 0 + len (ys) [(+)-comm 0 len ys]
                   = len [] + len (ys) [len.1].
  imediato.
Passo indutivo:
  Seja ks : La
  Suponha len (ks ++ ys) = len ks + len ys : (h.i)
  Seja k : a
  Calculamos:
    len ((k :: ks) ++ ys) = len (k :: (ks ++ ys)) [(++).2 k ks ys]
                          = S (len (ks ++ ys)) [len.2 (ks ++ ys)]
                          = S (len ks + len ys) [h.i]
                          = S (len ys + len xs) [(+)-comm len ks len ys]
                          = len ys + S (len ks) [(+).2 len ks len ys]
                          = S (len ks) + len ys [(+)-comm S (len ks) len ys]
                          = len (k :: ks) + len ys. [len.2 k ks].

Lema!

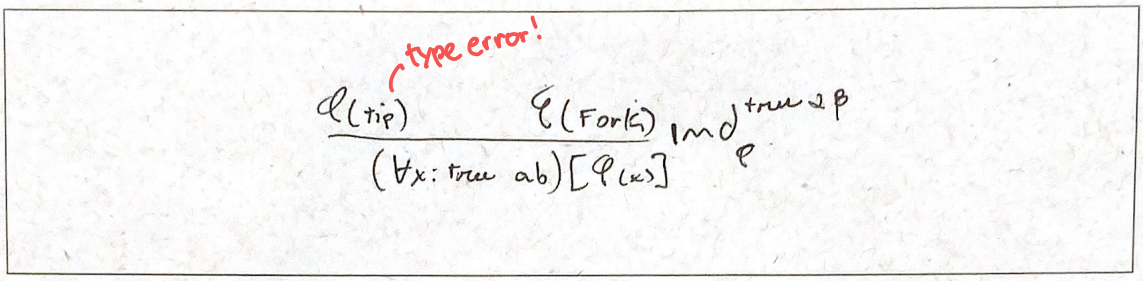
imediato.

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.



(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

Só isso mesmo.

(24) A

escreva infixo!

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
Data Nat where
  S:Nat |
  SO: Nat -> Nat

(++) :: [a] -> [a] -> [a]
(++) [] ys = ys
(++) (x:xs) ys = x:(xs ++ ys)

List a where
  Nil : List a
  Cons : a -> List a -> List a

length : [a] -> [a] -> [a]
length [] = x
length (x:xs) = S + length xs

Sum :: Num a => [a] -> a
Sum [] = 0
Sum (x:xs) = x + sum xs

Product :: Num a => [a] -> a
Product [] = 1
Product (x:xs) = x * Product xs
```

(21) G

Complete até 8 das equações seguintes com algo interessante:[4]

```
d · (n + m) = n
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs ++ product ys
reverse (xs ++ ys) = (++) (reverse ys) xs
length (xs ++ ys) = length xs ++ length ys
length (map f xs) = S length (map f xs)
sum (map (+ k) ns) = Sum (map K + map ns)
```

Grr..

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE `map f (xs ++ ys) = map f xs ++ map f ys`

Sejam xs, ys Tq. [map f (xs ++ ys)
Por indução no xs: -- CB: [] Acho: mesmo Trocando xs por []
Seja f ??
Calculamos
map f [] ys
= map f ys [(++) . 1 ys]
= map f ++ [] [(++) . 1]
= map f [] ys ++ map f []
Passo indutivo -- CB: (x : xs) Acho: xs nova x : xs
Seja f
Calculamos:
map f ((x : xs) ++ ys)
= xs' ++ ys [(++) . 2]
...

nada mudou!
cadê ??
map . 1

DEMONSTRAÇÃO DE `Product (xs ++ ys) = Product xs ++ Product ys`

Sejam xs, ys Tq. [Product (xs ++ ys = Product xs ++ Product ys]
Por indução no xs: CB: []
Calculamos:
Product ([] ++ ys)
= Product [] ++ ys = ys [(++) . 1
= Product ++ [] [(++) . 1]
= Product ? = 1 [Product . 1]
Passo indutivo
Calculamos
Product x : xs ++ ys
= Product x : (xs ++ ys) [(++) . 2]
...

??

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

Data Nat where 0 : Nat S : Nat → Nat ✓	length : List α → Int len [] = 0 len (x : xs) = length xs + 1 ✓
Data List α where Nil : List α Cons : α → List α → List α ✓	(++) : List α → List α → List α ~~xs ++ [] = xs~~ [] ++ xs = xs (x : xs) ++ ys = x : (xs ++ ys) ✓
map : (α → β) → List α → List β map f [] = [] map f (x : xs) = map x :: map xs ✓	Reverse : List α → List α Rev [] = [] Rev (x : xs) = Rev xs ++ [x] ✓

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = d · n + d · m
map f (xs ++ ys) = map f xs ++ map f ys ✓
product (xs ++ ys) = Prod xs · Prod ys ✓
reverse (xs ++ ys) = Rev ys ++ Rev xs ✓
length (xs ++ ys) = len xs + len ys ✓
length (map f xs) = len xs ✓
sum (map (+ k) ns) =
```

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

(36) P

Escolha **até duas** das equações de G para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE len (xs ++ ys) = len xs + len ys

Sejam xs, ys : List α. ✓
Por indução em xs. ✓
Caso []: ✓
Calculamos:
len ([] ++ ys) ✓
= len ys [(++).1] ✓
len [] + len ys ✓
= 0 + len ys [(len).1] ✓
= len ys + 0 [(+) com] ✓
= len ys [(+)-idR] ✓
Caso x :: xs'
Seja xs' : List α tal que IH: len ((x :: xs') ++ ys) = len (x :: xs) + len ys ✓
Calculamos:
len ((x :: xs') ++ ys) ✓
= len (x :: (xs' ++ ys)) [(++).2] ✓
= [illegible] len (xs' ++ ys) + 1 ✓
len (x :: xs') + len ys ✓
= 1 + len xs' + len ys [(len).2] ✗ ← olhe no teu len.2.
= [illegible] len (xs' ++ ys) + 1 [IH]

DEMONSTRAÇÃO DE d · (n + m) = d · n + d · m

Sejam d, n, m : nat ✓
Por indução em m. ✓
Caso 0:
Calculamos:
d · (n + 0) ✓
= d · n [(+)-idR]
d · n + d · 0
= d · n + 0 [(·).1]
= d · n [(+)-idR] ✓

Caso S m' : ~~Seja m' : nat tal que d · (n + S m') = d · n + d · S m'~~
Seja m' : nat tal que d · (n + S m') = d · n + d · S m' ✗
→ já tá no escopo!
Calculamos:
d · (n + S m')
= d · S (n + m') [(+).2]
= (d · (n + m)) + d [(·).2]
[illegible]
d · n + d · S m'
[illegible] ✗
= [illegible] d · (n + S m') [(+)]
= (d · (n + m)) + d [(·).2]

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
DATA NAT
  O : NAT
  S : NAT → NAT

DATA LIST α
  Nil : LIST α
  Cons : α → LIST α → LIST α

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

(++) : L α → L α → L α
[] ++ ys = ys
(x : xs) ++ ys = x : (xs ++ ys)

Reverse : L α → L α
Reverse [] = []
Reverse (x : xs) = reverse xs ++ [x]

length : L α → Nat
length [] = O
length (_ : xs) = S (length xs)
```

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = dn + dm
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs · product ys
reverse (xs ++ ys) = rev ys ++ rev xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = (sum ns) . k
```

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

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

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

Seja xs, ys : L a ✓
Por INDUÇÃO NO ys ✓
CASO [] ✓
calculemos:
len ([] ++ ys)
= len ys [(++).1 ys := ys]
len [] + len ys
= 0 + len ys [len.1]
= len ys + 0 [(+)-comm o len ys]
= len ys [(+).1 n := len ys]
imediato ✓
CASO (x : xs') ✓
seja x : a
seja xs' t.q len (xs' ++ ys) = len xs' + len ys ✓
calculamos: len (x : xs') + len ys
= S (length xs') + len ys [len.2] ✓
= len ys + S (len xs') [(+)-comm] ✓
= S (len ys + len xs') [(+).2] ✓
= S (len (xs ++ ys)) [H.I] ✓

len ((x : xs') ++ ys)
= len (x : (xs ++ ys)) [(++).2]
= S (len (xs ++ ys)) [len.2] ✓
= S (len (xs ++ ys)) (H.I)
imediato ✓

DEMONSTRAÇÃO DE d (n+m) = dn + dm .

Seja n, m : NAT
Por INDUÇÃO
BASE
Calculemos:
0.(n+m)
= (n+m).0 [(·)-com 0 (n+m)]
= 0 [(·).1 n := (n+m)]
0n + 0m
= n0 + m0 [(·)-com 0 n ; (·)-com 0 m] Cadê?
= 0 + 0 [(·).1 n := m ; (·).1 n := m]
= 0
imediato ✓

P.I
Seja K t.q K(n+m) = Kn + Km
Calculemos:
SK.(n+m)
= (n+m).SK [(·)-comm]
= ((n+m).K) + (n+m) [(·).2]
= (K.(n+m)) + (n+m) [(·)-comm]
= Kn + Km + (n+m) (H.I)

(SK.n) + (SK.m)
= (n.SK) + (m.SK) [(·)-com SKn ; (·)-com SKm]
= ((n.K) + n) + ((m.K) + m) [(·).2 n SK ; (·).2 m SK]
= (n.K) + n + (m.K) + m [(·)-ASSOC]
= (n.K) + (m.K) + n + m [(+)-comm]
= Kn + Km + n + m [(·)-com NK ; (·)-com mK]
= Kn + Km + (n+m) [(+)-ASSOC n m Km]
imediato ✓

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.

$$\frac{\mathcal{E}(Tip) \quad (\forall x : Tree\ \alpha\ \beta)[\mathcal{E}(Tip) \Rightarrow \mathcal{E}(Fork\ Tip)]}{(\forall x : Tree\ \alpha\ \beta)[\mathcal{E}(x)]}$$

IND Tree α β

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

height : Tree α β → NAT
height Tip = 0
height Fork : S (height Fork)

tips : Tree α β → ?
tips Tip : 1
tips Fork : S S (tips For-1)

Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
data Nat where O | S Nat
data List α where Nil | Cons α (List α)

map : (α → β) → [α] → [β]
map _ [] = []
map f (a:as) = (f a) :: (map f as)

length : List α → Nat
length [] = O
length (a::as) = S (length as)

filter : (α → Bool) → [α] → [α]
filter _ [] = []
filter P (a:as) = if (P a) then (a :: (filter P as)) else (filter P as)

sum : [Nat] → Nat
sum [] = O
sum (a::as) = a + (sum as)
```

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = dn + dm
map f (xs ++ ys) = (map f xs) ++ (map f ys)
product (xs ++ ys) = Prod xs · Prod ys
reverse (xs ++ ys) = rev ys ++ rev xs
length (xs ++ ys) = len xs + len ys
length (map f xs) = len xs
sum (map (+ k) ns) = (Sum ns) + (len ns · k)
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE $d\cdot(n+m) = dn + dm$.

Ind. no m. ✓
Caso O:
Calc $d\cdot(n+O)$
$= d\cdot n$ ✓ [(+).1 n O]
$dn + dO$
$= dn + O$ [(·).1 d O]
$= dn$ [(+).1 (dn) O]
Imediato. ✓

Caso Sm: -- $d\cdot(n+m) = dn+dm$ ← H.I
Calc $d\cdot(n+Sm)$
$= d\cdot S(n+m))$ ✓ [(+).2 n m]
$= d\cdot(n+m) + d$ ✓ [(·).2 d (S(n+m))]
$= (dn + dm) + d$ ✓ [H.I]
$dn + d(Sm)$
$= dn + (dm + d)$ ✓ [(·).2 d (Sm)]
Imediato.
↳ faltou (+)-ass.

DEMONSTRAÇÃO DE $\text{len}\,(\text{map}\,f\,xs) = \text{len}\,xs$.

Ind. no xs
Caso []:
Calc: len(map f []))
$=$ len [] [map.1 f []]
$= O$. [len.1 []]
len []
$= O$. [len.1 []]
Imediato.

Caso (x::xs):
Calc: len(map f (x::xs))
$=$ len ((f x)::map f xs) [map.2 f (x::xs)]
$= S$ (len(map f xs)) [len.2 (f x) (map f xs)]
$= S$ [len xs). [H.I] ✓
len (x::xs)
$= S$(len xs), [len.2 x xs] ✓
Imediato.

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree $\alpha\beta$
RESPOSTA.

$$\frac{\forall x : \beta\ [\varphi(\text{Tip}\ x)] \quad (\forall t_1, t_2 : \text{Tree}\ \alpha\ \beta)\ [\varphi(t_1) \,\&\, \varphi(t_2) \to (\forall x : \alpha)\ [\varphi(\text{fork}\ x\ t_1\ t_2)]]}{(\forall t : \text{Tree}\ \alpha\ \beta)\ [\varphi(t)]}\ \text{Ind}^{\text{Tree}\ \alpha\beta}_{\varphi}$$

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

```
height :: Tree α β → Nat
height (Tip _) = 0
height (fork a t1 t2) = S (max (height t1) (height t2))

max :: Nat → Nat → Nat
max 0 m = m
max n 0 = n
max (S n) (S m) = max n m

tips :: Tree α β → Nat
tips (Tip _) = S 0
tips (fork a t1 t2) = (tips t1) + (tips t2)
```

Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
data Nat
  O :: Nat
  S x :: Nat -> Nat

data List α
  [] :: List α
  (:) :: α -> List α -> List α

(++) :: Lα -> Lα -> Lα
[] ++ ys = ys
(x:xs) ++ ys = x : (xs ++ ys)
infixr 3 ++

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

sum :: List α -> Nat
sum [] = 0
sum (x:xs) = x + sum xs

product :: List α -> Nat
product [] = 1
product (x:xs) =
    x . product xs
```

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = (d.n) + (d.m)
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs . product ys
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = sum ns . k
```

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

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α. B
RESPOSTA.

$$\frac{\varphi(\text{Tip}) \qquad (\forall\ \text{Tree}\ a\ b)\ [\varphi(a) \Rightarrow \varphi(\text{Tree}\ a\ b)]}{\varphi\ (\text{Tree}\ a\ b)}\ \text{IND}^{\text{Tree}\ \alpha\ \beta}_{\varphi}$$

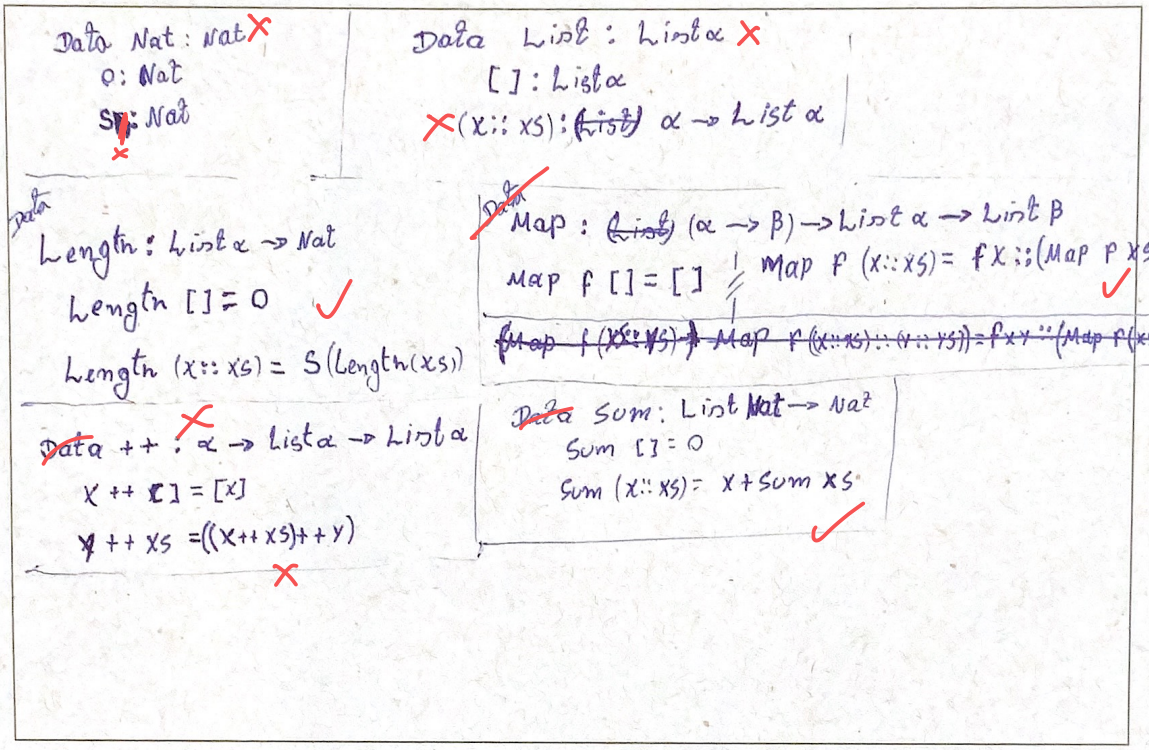
(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

```
tips :: Tree α β → Nat
tips (b) = 1.
tips (T cd) = 2 + tips (T cd)

height :: T α β → Nat
height (b) = 1
height T cd = 1 + height T cd
```

Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.



(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = dn + dm
map f (xs ++ ys) = (f x y :: (Map f (xs' :: ys'))) Map f xs :: (Map f ys)
product (xs ++ ys) = x · y :: (Product (xs' :: ys'))
reverse (xs ++ ys) = xs :: (Reverse ys)
length (xs ++ ys) = S (Length ys)
length (map f xs) = S (Length (Map f xs'))
sum (map (+ k) ns) = ((n + k) :: (Map (+ k) ns')) (n + k) + Sum (map (+ k) ns')
```

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

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE $d \cdot (n + m) = dn + dm$.

Sejam d, n, m
Calculamos:
$d \cdot (n + m)$
$= dn + dm$ [(·) - distR -- (·) - distR, o "R" ficou com muita tinta.

↑ ALVO

□.

DEMONSTRAÇÃO DE length (xs ++ ys) = S (length ys).

Seja xs
Seja ys
Calculamos:
length (xs ++ ys)

= ~~(xs)~~ S (length ys) [length.2 com x = xs, xs = ys]

□.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

DATA NAT:
  0 : NAT
  S : NAT -> NAT

DATA LIST α :
  [] : LIST α
  _ :: _ : α -> LIST α -> LIST α

LENGTH : LIST α -> NAT
  LENGTH [] = 0
  LENGTH (X :: XS) = S0 + LENGTH XS

SUM : LIST NAT -> NAT
  SUM [] = 0
  SUM (X :: XS) = X + SUM XS

MAP : (α -> β) -> LIST α -> LIST β
  MAP [] Lα = Lα
  MAP F (A :: Lα) = F(A) :: MAP F Lα

PRODUCT : LIST NAT -> NAT
  PROD [] = 0 X = 1
  PROD (X :: []) = X
  PROD (X :: XS) = X · PROD XS

— — 0 = 0
— — S0 = 1

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = 0·N + 0·M
map f (xs ++ ys) = (MAP F XS) ++ (MAP F YS)
product (xs ++ ys) = X · PRODUCT (XS ++ YS)
reverse (xs ++ ys) = REV YS ++ REV XS
length (xs ++ ys) = LENGTH XS + LENGTH YS
length (map f xs) = LENGTH XS
sum (map (+ k) ns) = SUM NS + K · LENGTH NS
```

[4] DEFINIÇÃO. Chamamos algo de *interessante* ssc Thanos acha tal algo interessante.

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE LENGTH (MAP F XS) = LENGTH XS .

-- F SÓ MUDOU CADA COISA DE XS, MAS NÃO O TAMANHO DE XS

X

DEMONSTRAÇÃO DE SUM (MAP (+ K) NS) = SUM NS + K · LENGTH NS

-- ADICIONAMOS K A CADA ELEMENTO DE NS

X

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.

$$\frac{(\forall \beta)[P(\text{Tip}(\beta))] \quad (\forall \alpha)[P(\text{Fork}(\alpha))]}{(\forall \alpha, \beta)[P(\alpha) \ \& \ P(\beta)]} \text{ IND}^{\text{Tree}\ \alpha\ \beta}_{P}$$

X

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
Data Nat where
  Nat : Type
  O : Nat
  S : Nat -> Nat

Length : List α -> Nat
Length [] = O
Length (x :: xs) = S (Length xs)

Data List where
  List : Type -> Type
  [] : List α
  Cons : α -> List α -> List α

(++) : List α -> List α -> List α
[] ++ ys = ys
(x :: xs) ++ ys = x :: (xs ++ ys)

Sum : List α -> Int
Sum [] = 0
Sum (x :: xs) = x + Sum xs

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

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = d·n + d·m
map f (xs ++ ys) = Map f xs ++ Map f ys
product (xs ++ ys) = Product xs · Product ys
reverse (xs ++ ys) = Reverse ys ++ Reverse xs
length (xs ++ ys) = Length xs + Length ys
length (map f xs) = Length xs
sum (map (+ k) ns) = Sum ns + k · length ns
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE $Map\ f\ (xs ++ ys) = Map\ f\ xs ++ Map\ f\ ys$

Sejam $f : \alpha \to \beta, ys : List\ \alpha$ ✓ — assim tu chamou de ys isso.

Por indução

Base: -- ?

Calculamos: $Map\ f\ ([] ++ ys)$ ✓

$= Map\ f\ ys$ [(++).1] ✓

$= Map\ f\ ys ++ []$ [← (++).1] ✗

$= Map\ f\ ys ++ Map\ f\ []$ [← Map.1]

Imediato

Passo indutivo: -- ?

Calculamos $Map\ f\ ((x :: xs) ++ ys)$

$= Map\ f\ (x :: (xs ++ ys))$ [(++).2]

$= f\ x :: Map\ f\ (xs ++ ys)$ [Map.2]

$= f\ x :: (Map\ f\ xs ++ Map\ f\ ys)$ [H.I]

$= \ldots$

DEMONSTRAÇÃO DE $Length\ (xs ++ ys) = Length\ xs + Length\ ys$

Seja $ys : List\ \alpha$

Por indução

Base:

Calculamos: $Length\ ([] ++ ys)$ ✓

$= Length\ ys$ [(++).1] ✓

$= Length\ ys + 0$ [(+).1] ✓

$= Length\ ys + Length\ []$ [← Length.1] ✓

Imediato

Passo indutivo:

Calculamos: $Length\ ((x :: xs) ++ ys)$

$= Length\ (x :: (xs ++ ys))$ [(++).2]

$= S\ (Length\ (xs ++ ys))$ [Length.2]

$= S\ (Length\ xs + Length\ ys)$ [H.I]

$= S\ (Length\ ys + Length\ xs)$ [+ Com]

$= S\ (Length\ ys) + Length\ xs$ [(+).2] ✗

$= Length\ (x :: xs) + Length\ ys$ ✗

Imediato

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.

$$\frac{\varphi(Tip) \qquad (\forall t, t')[\varphi(t) \,\&\, \varphi(t') \Rightarrow \varphi(t)]}{(\forall t : Tree)[\varphi(t)]}\ Ind^{Tree\ a\ B}_{\varphi}$$

quem é?

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

se fosse assim no Nat seria:

$(\forall k)[\varphi(k) \Rightarrow \varphi(n)]$!

Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
Nat a -> nat ??
0 : 0
n : sn                                  X

List list a -> List a
List [] = []
List xs = (x :: xs)                     X

map list nat -> list nat                X
map []                                  X
map (x :: xs) = x :: (map xs)
X

(++) : ??
[] ++ ys = ys
(x :: xs) ++ ys = x :: (xs ++ ys)       ✓

length : List a -> nat
length [] = 0                           ✓
length (x :: xs) = S (length xs)

product : List a -> nat
product [] = S 0                        ✓
product (x :: xs) = x . product xs
```

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) =
map f (xs ++ ys) = f x :: (map f (xs ++ ys))    X
product (xs ++ ys) = product (xs) . product (ys)    ✓
reverse (xs ++ ys) =
length (xs ++ ys) = length (xs) + length (ys)    ✓
length (map f xs) = S (length f x :: (map f xs))    X
sum (map (+ k) ns) =
```

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

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE Lenght (xs ++ ys) ?

lenght (xs ++ ys) = $Lenght (xs) + lenght (ys)

Seja lenght (xs ++ ys) ??

Indução no xs

Caso []

Calculamos: Lenght ([] ++ xs]

= lenght ys [++]. ys = ys

Caso (x::xs):

Calculamos: Lenght (Oxs ++ ys]

?

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

DEMONSTRAÇÃO DE _____.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

① data Nat where
  0 : Nat
  S : Nat → Nat

② data List α where
  [] : List α
  (::) : α → List α → List α

③ map : (α → β) → [α] → [β]
  map _ [] = []
  map f (x :: xs) = f x :: map f xs

④ (++) : [α] → [α] → [α]
  [] ++ ys = ys
  (x :: xs) ++ ys = x :: (xs ++ ys)

⑤ length : [α] → Nat
  length [] = 0
  length (x :: xs) = S (length xs)

⑥ sum : Integral α ⇒ [α] → α
  sum [] = 0
  sum (x :: xs) = x + sum xs

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = dn + dm
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs * product ys
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = sum ns + k * length ns
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

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

Sejam xs, ys : List α ~~, tq map f (xs~~

Por indução no xs :

Caso [] :

Calculamos:

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

Calculamos:

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

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

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

Caso x::xs: Suponha map f (xs ++ ys) = map f xs ++ map f ys (HI)

Calculamos:

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

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 ~~[map.2]~~ [(++).2]

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

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

Sejam xs, ys : List α.

Por indução no xs :

Caso [] :

Calculamos:

length ([] ++ ys)

= length ys

Calculamos.

length [] + length ys [length.1]

= 0 + length ys

~~length~~ = S (0 + length ys)

= length ys.

Caso x::xs:

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

Calculamos:

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

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

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

= S (len xs + len ys)

= len xs + S (len ys).

Calculamos:

~~len xs + len ys~~

len (x::xs) + len ys

S (len x) + len ys.

...e...?

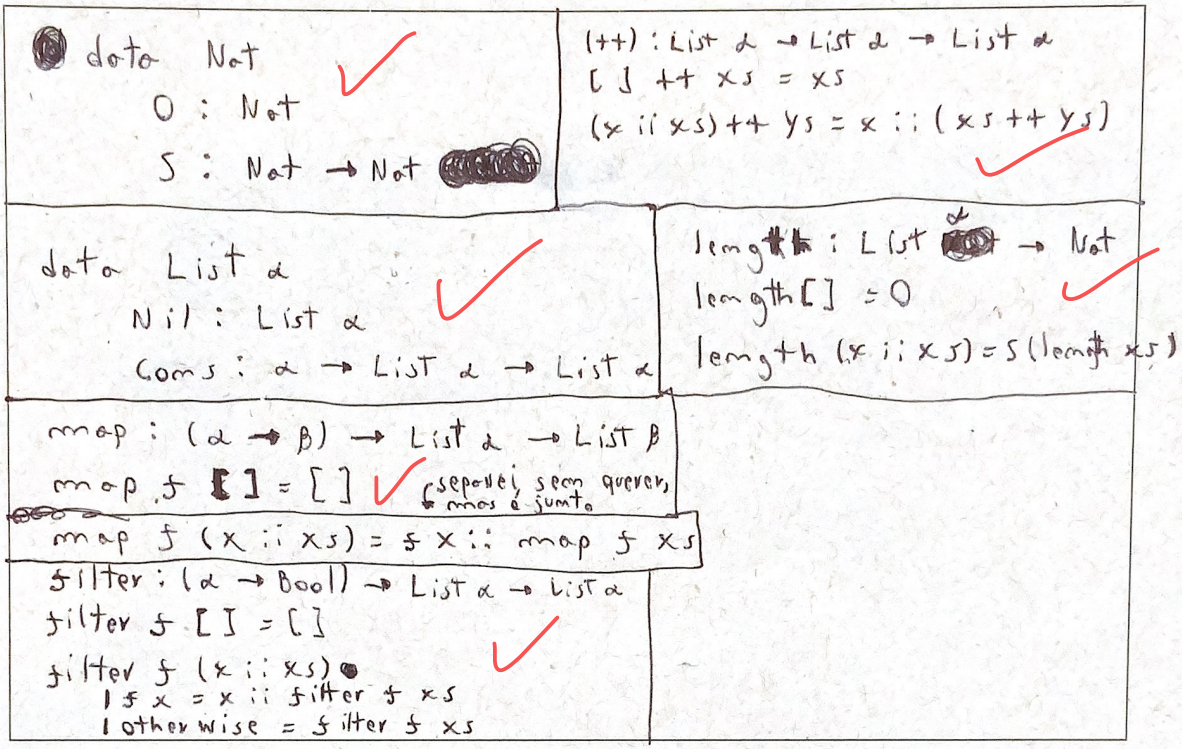
length (x::xs) + length ys

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

=

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.



(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = d · n + d · m
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs · product ys
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = map (+ k) (sum ns)
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

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

Sejam xs, ys : List α.
Por indução em xs.
Caso []:
  Calculamos:
    len ([] ++ ys)
      = len ys [(++).1] ✓
    len [] + len ys
      = 0 + len ys [len.1]
      = len ys + 0 [(+).comm]
      = len ys [(+).1] ✓
Caso (x' :: xs'):
  Calculamos:
    len ((x' :: xs') ++ ys)
      = len (x' :: (xs' ++ ys)) [(++).2]
      = S (len (xs' ++ ys)) [len.2] ✓
      = S (len xs' + len ys) [HI]
      = S (len ys + len xs') [(+).comm] ✓
      = len ys + S (len xs') [(+).2] ✓
      = len ys + len (x' :: xs') [len.2] ✓
      = len (x' :: xs') + len ys [(+).comm] ✓ ?
■

DEMONSTRAÇÃO DE d · (m + m) = d · m + d · m .

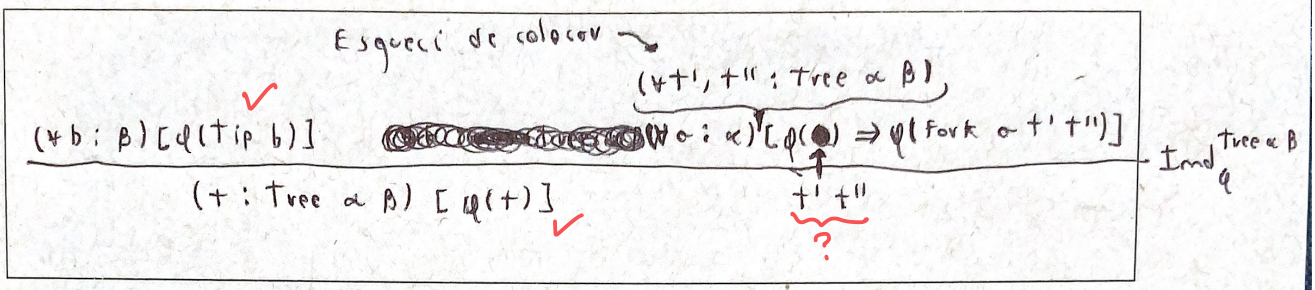
Sejam d, m, m : Nat.
Por indução em ~~m~~.
Caso 0:
  Calculamos:
    d · (m + 0)
      = d · m [(+).1]
    d · m + d · 0
      = d · m + 0 [(·).1]
      = d · m [(+).1] ✓
Caso S m':
  Calculamos:
    d · (m + S m')
      = d · S (m + m') [(+).2]
      = d · (m + m') + d [(·).2]
      = (d · m + d · m') + d [HI]
      = d · m + (d · m' + d) [(+).ass]
      = d · m + d · S m' [(·).2]
■ ✓

(19) **I**

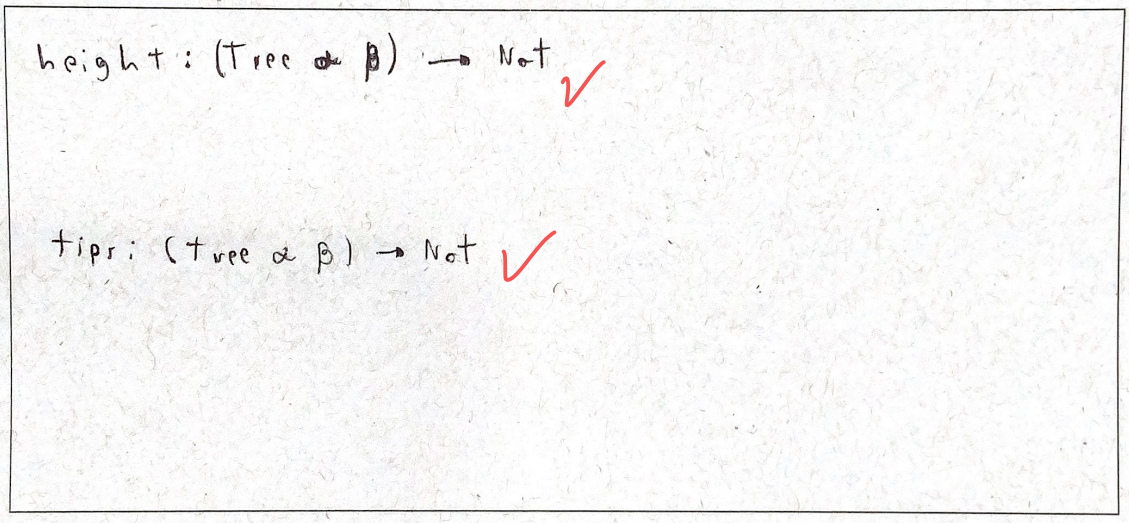
Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.



(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.



Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

Nat é um type.

Nat n = { 0
        | S0 = 1
        | SS0 = 2
        | SSS0 = 3
        | ⋮

indutivamente, temos que se n é um nat, então n+1 é um nat.

Product : Nat → Nat → Nat

(·) : { CASO 0 : 0 · S0 = 0
      | 0 · Sn = 0
      | CASO n : n · Sn = n+n+n+... + (Sn vezes)

length : Nat → Nat

sum : Nat → Nat → Nat

(+) : 0 + S0 = S0

S0 + S0 = SS0

S(0+S0) = SS0 -- CASO BASE

⋮

então, para todo n : -- PASSO INDUTIVO

S(n + Sn) = SSn

List : Set α → [ ] [ ] [set α] ??

([ ]) : { [ ] , set α = ∅
        | [set α] , set α ≠ ∅

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = d·n + d·m
map f (xs ++ ys) = map f (xs) ++ map f (ys)
product (xs ++ ys) = product (xs) ++ product (ys)
reverse (xs ++ ys) = rev (xs) ++ rev (ys)
length (xs ++ ys) = l(xs) ++ l(ys)
length (map f xs) = map (length f xs)
sum (map (+ k) ns) = map (sum (+k) ns)
```

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

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE d. (n+m) .

d. (n+m) = d.n + d.m [ℤM - distr - R]

X

DEMONSTRAÇÃO DE length (xs ++ ys) .

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

X

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree $\alpha\beta$
RESPOSTA.

Tree α β

a : α b : β X
———————— IND Tree α β
(a → tree a b → tree a b → tree a b) → b → tree a b

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

height : ~~tree~~ length (Fork) → tip → Nat

tips : ~~a'→~~ tip → Nat X

Só isso mesmo.

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
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 p [] = []
Filter p (x:xs) = case p x of
  |True → x : filter p xs
  |false → filter p xs

Length : List α → Nat
Length [] = 0
Length (x:xs) = S(length xs)

(++) : List α → List α → List α
[] ++ ys = ys
(x:xs) ++ ys = x : (xs ++ ys)

sum : Numb α ⇒ List α → α
sum [] = 0
sum (x:xs) = x + sum xs

product : Numb α ⇒ List α → α
product [] = 1
product (x:xs) = x · product xs
```

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = d·n + d·m
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs · product ys
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = (length ns)·k + sum ns
```

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

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

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

Indução no xs

Caso []:

length ([] ++ ys)
= length ys [(++).1]
= 0 + length ys [(+) idL ←]
= length [] + length ys [length.1 ←]

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] Lemma?
= length ys + S (length xs) [(+).2 ←]
= length ys + length (x:xs) [length.2 ←]
= length (x:xs) + length ys [(+) com]

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

Indução no xs

Caso []:

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

Caso (x:xs):

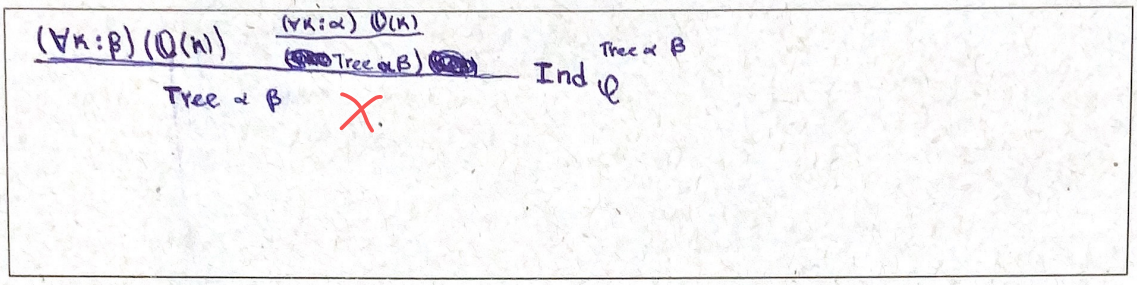
map f ((x:xs) ++ ys)
= map f (x : (xs ++ ys)) [(++).2]
~~= f x : map f (xs ++ ys) [map.2]~~
~~= f x : map f xs ++ map f ys~~
~~= map f ((x:xs) ++ ys) [~~
= f x : map f (xs ++ ys) [map.2]
= f x : ((map f xs) ++ (map f ys)) [H.I]
= (f x : map f xs) ++ map f ys [(++).2 ←]
= map f (x:xs) ++ map f ys [map.2 ←]

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.



(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

Só isso mesmo.

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
data Nat
  0 : Nat
  S : Nat → Nat

data List α
  Nil  : List α
  Cons : α → List α → List α

reverse List α → List α
  reverse [] = []
  reverse (x : xs) = reverse xs ++ x

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

sum : List Nat → Nat
  sum [] = 0
  sum (x : xs) = x + sum xs

product : List Nat → Nat
  product [] = 1
  product (x : xs) = x · product xs
```

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = dn + dm
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs · product xs
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = sum ns + k · length ns
```

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

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

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

Usando a recursividade em xs ✗ ?!?

Caso []
Calculamos
length ([] ++ ys)
= length ys ✓ [?]

length [] + length ys
= 0 + length ys
= length ys + 0 [?]
= length ys

Caso (x : xs)
Calculamos
length ((x : xs) ++ ys)
= length (x : (xs ++ ys)) ✓
= S (length (xs ++ ys)) ✓
= S (length xs + length ys) ✓
= S (length ys + length xs) ✓
= length ys + S (length xs) ✓
= length ys + length (x : xs) ✓
= length (x : xs) + length ys ✓

[? .]

DEMONSTRAÇÃO DE product (xs ++ ys) = product xs · product ys .

Usando a recursividade em xs

Caso []
Calculamos
product ([] ++ ys) ✓
= product ys

product [] · product ys
= 1 · product ys ✓
= product ys

Caso (x : xs)
Calculamos
product ((x : xs) ++ ys) ✓
= product (x : (xs ++ ys)) ✓
= x · product (xs ++ ys) ✓
= x · (product xs · product ys) ✓
= product (x : xs) · product ys ✗
aff.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

type error

```
map : (A → B) → LA → LB
map f [] = []
map f (x::xs) = f x :: map f xs

filter : (X → Bool) → LX → LX
filter f [] = []
filter f (x::xs) = if f x then x :: filter f xs
                   else filter f xs

(++) : LX → LX
[] ++ xs = xs
xs ++ (y::ys) = y ++ (xs ++ ys)

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

Product : List X → Nat
pred [] = []
pred (x::xs) = x + prod xs

List : type → type
List [] = []
List (x::xs) = x ++ List xs
```

(21) **G**

Complete ~~até 8~~ Todos das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = d.n + d.m
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs + product ys
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = id
```

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

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

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

Por indução em xs
caso []
Seja ys
calc: len ([]++ys) = len ys [len 1] ✓
= []++len ys [len 1] ~
= len [] ++ len ys [len 1] ✓
imediato.

caso (x::xs)
seja ys
~~Passo indutivo~~
calc: len ((x::xs)++ys) = S(len xs)+ len ys [len 2] ✗
= len (x::xs)+ len ys [len 2]
imediato.

-- len (xs++ys) = len xs + len ys ??
-- => len ((x::xs)++ys) = len (1::xs)
-- + len ys

DEMONSTRAÇÃO DE map f (xs++ys) = map f xs ++ map f ys .

indução em xs
caso []
Seja ys
calc: map f ([]++ys) = map f ys [map 1].
= []++map f ys [map 1]
= map f [] ++ map f ys [map 1]
imediato.

caso (x::xs)
Seja xs
~~Passo indutivo~~
calc: map f ((x::xs)++ys) =
= f x :: map f xs ++ map f ys [map 2] ✗
= map f (x::xs) ++ map f ys [map 2]
imediato.

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

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
data Nat
  O :: Nat
  S :: Nat → Nat

data List a
  Nil :: List a
  Cons :: L a → a

map :: (a → b) → [a] → [b]
map _ [] = []
map f (x : xs) = f x : (map f xs)

filter :: (a → Bool) → [a] → [a]
filter _ [] = []
filter f (X : xs) = if f x
  then x : filter f xs
  else filter f xs

length :: [a] → Nat
length [] = 0
length (_ : xs) = S (length xs)

sum :: [Nat] → Nat
sum [] = 0
sum (x : xs) = x + (sum xs)
```

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = (d · n) + (d · m)
map f (xs ++ ys) = (map f xs) ++ (map f ys)
product (xs ++ ys) = (product xs) · (product ys)
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = (sum ns) + (K · length ns)
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE $d \cdot (n + m) = (d \cdot n) + (d \cdot m)$.

Sejam d, n, m : Nat
Por indução em d:
Caso Base:
calculemos:
$0 \cdot (n+m) = (n+m) \cdot 0$ [(·)-comm]
$= 0$ [(·).1]
calculemos:
$0 \cdot n + 0 \cdot m = 0$ [(·)₀.1 n ; (·).1 m]
Logo $0 \cdot (n+m) = 0 \cdot n + 0 \cdot m$ [(=)-trans]
Imediato.
Passo indutivo:
Seja k.
Suponha hi: $k \cdot (n+m) = kn + km$
calculemos:
$(Sk) \cdot (n+m) = (n+m) \cdot (Sk)$ [(·)-comm]
$= ((n+m) \cdot k) + (n+m)$ [(·).2]
$= (kn + km) + (n+m)$ [hi]
calculemos:
$(Sk) \cdot n + (Sk) \cdot m = n \cdot (Sk) + m \cdot (Sk)$ [(·)-comm]
$= ((n \cdot k) + n) + ((m \cdot k) + m)$
$= (n \cdot k) + (n + ((m \cdot k) + m))$ [(+)-assoc]
$= (n \cdot k) + ((m \cdot k) + n + m)$ [(+)-comm]
$= (n \cdot k) + ((m \cdot k) + (n + m))$ [(+)-assoc]
$= (k \cdot n) + (k \cdot m) + (n + m)$ [(·)-comm]
Logo $(Sk) \cdot (n+m) = (Sk) \cdot n + (Sk) \cdot m$ [(=)-trans] ~
Imediato.

DEMONSTRAÇÃO DE $length\ (xs ++ ys) = len\ xs + len\ ys$

Sejam xs, ys
Por indução no xs
Caso Base:
Logo length ([] ++ ys) = length ys [(++).1]
Logo len [] + len ys = 0 + len ys [len.1]
Logo len [] + len ys = len ys [(+)-comm ; (+).1]
Imediato.
Passo indutivo:

```
(++) :: [a] -> [a] -> [a]
[] ++ ys = ys
(x : xs) ++ ys = x : (xs ++ ys)
```

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.

$$\frac{(\forall x)[\varphi(t) \to \varphi(\text{Tip } x)]}{\varphi(t : \text{Tree } a\ b)}$$ ✗

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

eu sei que o typeclass Num me permite usar operações math, mas acho que ele não me deixam retornar números em si, então escolhi integral que deixa

↳ confundiu

```
Data Nat
  O : Nat
  S : Nat -> Nat

Data List a :
  Nill : [] a
  Cons : a -> list a -> list a

filter :: (a -> Bool) -> [a] -> [a]
fil _ [] = []
fil con (x:xs) =
  | con x = x : fil con xs
  | otherwise = fil con xs

length :: [a] -> infNat
len [] = 0
len (x:xs) = 1 + len xs
             S( )

product :: integral a => [a] -> a
product [] = 1
product (x:xs) = x * product xs

map :: (a -> b) -> [a] -> [b]
map _ [] = []
map fun (x:xs) = fun x : map fun xs
```

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d * (n + m)          = d*n + d*m
map f (xs ++ ys)     = map f (xs++[]) = map f xs
product (xs ++ ys)   = product (xs++[]) = product xs
reverse (xs ++ ys)   = reverse (xs++[]) =
length (xs ++ ys)    = length (xs++[]) = length xs
length (map f xs)    = length (ma f []) = 0
sum (map (+ k) ns)   = sum (map (+k) []) = 0
```

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

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
data Nat = O | S Nat

filter :: (a -> Bool) -> [a] -> [a]
filter _ [] = []
filter f (x:xs) =
  if fx
  then x : filter f xs
  else filter f xs
length :: [a] -> Nat
length [] = O
length (x:xs) = S (length xs)
sum :: [Nat] -> Nat
sum [] = O
sum (x:xs) = x + sum xs

product :: [Nat] -> Nat
product [] = 1
product (x:xs) = x . product xs
(++) :: [a] -> [a] -> [a]
[] ++ ys = ys
(x:xs) ++ ys = x : (xs ++ ys)
```

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = d·n + d·m
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs · product ys
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = sum ns + (length ns) · k
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE $d \cdot (n + m) = d \cdot n + d \cdot m$

$(\forall m)(\forall d, n)[d \cdot n + m = d \cdot n + d \cdot m]$
Por indução no m -- Alvo: $(\forall d, n)[d \cdot (n + m) = d \cdot n + d \cdot m]$
Caso $m =: 0$
Sejam $d, n : Nat$ -- Alvo: $d \cdot (n + 0) = d \cdot n + d \cdot 0$
Calculamos:
$d \cdot (n + 0)$
$= d \cdot n$ [ι+).1 com $m := n$]
$d \cdot n + d \cdot 0$
$= d \cdot n + 0$ [ι·).1 com $n := d$]
$= d \cdot n$ [ι+).1 com $n := d \cdot n$]
Logo $d \cdot (n + 0) = d \cdot n + d \cdot 0$
Trivial
Caso $m =: Sm'$ -- H.1 : $d \cdot (n + m') = d \cdot n + d \cdot m'$
Suponha $d \cdot (n + m') = d \cdot n + d \cdot m'$
Calculamos:
$d \cdot (n + Sm')$
$= d \cdot S(n + m')$ [ι+).2 com $n := n, m := m'$]
$= d \cdot (n + m') + d$ [ι·).2 com $n := d, m := n + m'$]
$d \cdot n + d \cdot Sm'$
$= d \cdot n + d \cdot m' + d$ [ι·).2]
$= d \cdot (n + m') + d$ [H.1] → Logo $d \cdot (n + Sm') = d \cdot n + d \cdot Sm'$
Trivial

DEMONSTRAÇÃO DE $ILength(xs ++ ys) = ILength(xs) + ILength(ys)$

Por indução no xs
Caso $xs =: []$
Sejam $ys : List$ -- Alvo: $ILength([] ++ ys) = ILength[] + ILength\ ys$
Calculamos:
$ILength([] ++ ys)$
$= ILength(ys)$ [ι++).2]
$ILength[] + ILength\ ys$
$= 0 + ILength\ ys$ [$ILength.1$ com $[a] := []$]
$= ILength\ ys + 0$ [ι+).1 com $n := 0, m := ILength\ ys$]
$= ILength\ ys$ [ι+).1 com $n := ILength\ ys$]
Logo $ILength([] ++ ys) = ILength[] + ILength\ ys$
Trivial

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

```
height :: Tree α β → Nat
```

```
tips :: Tree α β → Nat
tips Tip = 1
tips (Fork a) = 1 + tips (Fork a-1)
```

Só isso mesmo.

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
Nat : Type
data Nat
  0 : Nat
  S : Nat → Nat
List : Type → Type
data List α
  [] : List α
  (::) : α → List α → List α

Map : (α → β) → List α → List β
Map _ [] = []
Map f (x :: xs) = f x :: Map f xs

Length : List α → Nat
Length [] = 0
Length (x :: xs) = S (Length xs)

(++) : List α → List α → List α
[] ++ ys = ys
(x :: xs) ++ ys = x :: (xs ++ ys)

Sum : List Nat → Nat
Sum [] = undefined
Sum [x] = x
Sum (x :: xs) = x + Sum xs
```

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = dm + dm
map f (xs ++ ys) = (map f xs) ++ (map f ys)
product (xs ++ ys) = (product xs) + (product ys)
reverse (xs ++ ys) = (reverse ys) ++ (reverse xs)
length (xs ++ ys) = (length xs) + (length ys)
length (map f xs) = length xs
sum (map (+ k) ns) = (sum ns) + (k · (length ns))
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE length (map f Xs) = length Xs .

Seja Xs : List α.
Por indução em Xs.
Base:
| Seja f : α → β
| Calc:
| | len (map f []) ✓
| | = len [] [map.1 f]
P.I.:
| Seja Ks : List α t.q. (∀f)[len (map f Ks) = len Ks], (hi)
| Seja K : α.
| Seja f : α → β.
| Calc:
| | len (map f (K :: Ks))
| | = len (f K :: map f Ks) [map.2 f K Ks]
| | = S (len (map f Ks)) [length.2 (f K) (map f Ks)]
| | = S (len Ks) [hi f]
| | = len (K :: Ks) ■ [length.2 K Ks]

DEMONSTRAÇÃO DE map f (Xs ++ Ys) = (map f Xs) ++ (map f Ys)

Seja Xs : List α.
Por indução em Xs.
Base:
| Seja Ys : List α.
| Seja f : α → β.
| Calc:
| | map f ([] ++ Ys)
| | = map f Ys [(++).1 Ys]
| | = [] ++ (map f Ys) [(++).1 (map f Ys)]
| | = (map f []) ++ (map f Ys) [map.1 f]

| | = (f K) :: (map f (Ks ++ Ys)) [map.2 f K (Ks ++ Ys)]
| | = (f K) :: ((map f Ks) ++ (map f Ys)) [hi Ys f]
| | = map f (K :: Ks) ++ map f Ys [map.2 f K Ks]
| | ■

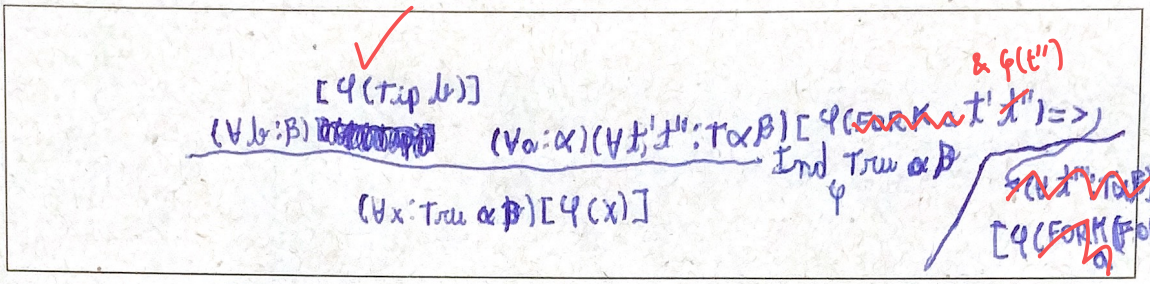
P.I.:
| Seja Ks : List α t.q. (∀Ys)(∀f)[map f (Ks ++ Ys) = (map f Ks) ++ (map f Ys)], (hi)
| Seja K : α.
| Seja Ys : List α.
| Seja f : α → β.
| Calc:
| | map f (K :: Ks ++ Ys)
| | = map f (K :: (Ks ++ Ys)) [(++).2 K Ks Ys]

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.



(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
map : (α → β) → Lista → Listβ
map _ [] = []
map f (x::xs) = f x :: map f xs

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

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

```
(++) : Lα → Lα → Lα
l ++ [] = l
[] ++ l = l
(x::xs) ++ ys = x::(xs ++ ys)

**
sum : LNat → Nat
sum [] = 0
sum (x::xs) = x + sum xs

***
product : LNat → Nat
product [] = S 0
product (x::xs) = x * product xs
```

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

?? Assim sempre será considerado errado

```
d · (n + m) = d·n + m·d
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = prod xs * prod ys
reverse (xs ++ ys) = rev ys ++ rev xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = (length ns) · k + sum ns
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE length (map f xs) = length xs .

Por indução.

Caso []:
| Calculamos:
| | length (map f []) = length [] (map.1) ✓

Caso (k::ks):
| Seja ks: LNat tal que length (map f ks) = length ks. ✓
| Seja k: Nat.
| Calculamos:
| | length (map f (k::ks)) = length (f k :: map f ks) (map.2)
| | = S (length (map f ks)) (length.2)
| | = S (length ks) (HI)
| | = length (k::ks) ✓ (length.2)

■

DEMONSTRAÇÃO DE sum (map (+k) ns) = (length ns) · k + sum ns.

Por indução.

Base: -- ns = []
| Calculamos:
| | sum (map (+k) []) = sum [] ✓ (map.1)

PI:
| Seja Rs: LNat tal que sum (map (+k) Rs) = (length Rs) · k + sum Rs.
| Seja R: Nat.
| Calculamos:
| | sum (map (+k) (R::Rs)) = sum ((R+k) :: map (+k) Rs) ✓ (map.2)
| | = (R+k) + sum (map (+k) Rs) ✓ (sum.2)
| | = (R+k) + (length Rs) · k + sum Rs ✓ (HI)
| | = R + (S (length Rs)) · k + R + sum Rs (distr.(·) sobre (+)) ×
| | = (length (R::Rs)) · k + sum (R::Rs) (sum.1) ×

(19) **I**

Para qualquer $\alpha\beta$: Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree $\alpha\beta$
RESPOSTA.

$$\frac{\varphi(Tip) \qquad \varphi(T') \;\&\; \varphi(T'') \Rightarrow \varphi(T)}{(\forall T : Tree)[\varphi(T)]}$$

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

```
height : T αβ → Nat
height Tip = 0
height (Fork f b) = S(max(height f) (height b))

Qual o tipo de Tip?

tips : T αβ → Nat
tips Tip = 1
tips Fork(f, b) = tips f + tips b
```

Só isso mesmo.

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

~~Data List~~
~~Nil : Data List~~

```
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) =
  | f x = x :: (filter f xs)
  | otherwise = filter f xs

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

```
rev : List α -> List α
rev [] = []
rev (x::xs) = (rev xs) ++ [x]

(++): List α -> List α -> List α
(++) [] ys = ys
(++) (x::xs) ys = x :: (xs ++ ys)

Sum : List α -> α
Sum [] = 0
Sum (x::xs) = x + (Sum xs)

-- α é um número no sum
Num α ⇒ ...
```

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = dn + dm
map f (xs ++ ys) = (map f xs) ++ map f ys
product (xs ++ ys) = (product xs) + product ys
reverse (xs ++ ys) = (reverse ys) ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = Sum ns + (K · (len ns))
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE $(\forall xs, ys : Lista\ \alpha)\ [len\ (xs ++ ys) = (len\ xs) + len\ ys]$

Sejam xs, ys : List α
Indução no xs.
caso []:
calculamos:
len([] ++ ys)
= len ys [(++).1]
len [] + len ys
= ~~[] + len~~
= 0 + len ys [len.1]
= len ys + 0 [(+)-comm]
= len ys [(+).1]

caso (x::xs):
calculamos
len ((x::xs) ++ ys)
= len (x :: (xs ++ ys)) [(++).2]
= S (len (xs ++ ys)) [len.2]
= S (len xs + len ys) [HI]
= S (len ys + len xs) [(+)-comm]
= len ys + S (len xs) [← (+).2]
= len ys + len (x::xs) [← len.2]
= len (x::xs) + len ys [(+)-comm]

∎

DEMONSTRAÇÃO DE $map\ f\ (xs ++ ys) = (map\ f\ xs) ++ (map\ f\ ys)$

Seja f : α → β
Sejam xs, ys : List α
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) [HI]
= map f (x::xs) ++ map f ys [← map.2]

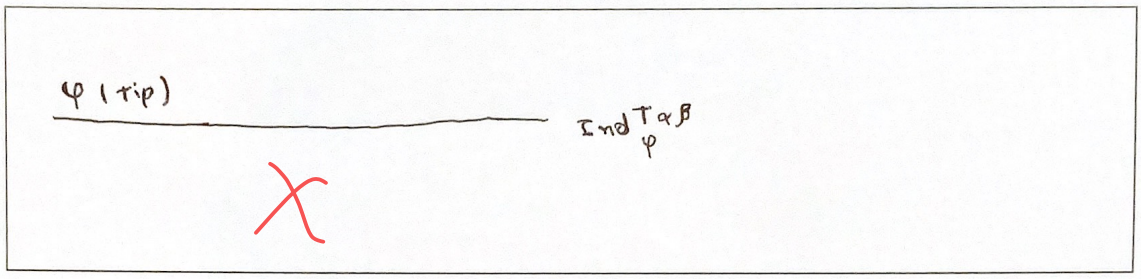
∎

(19) **I**

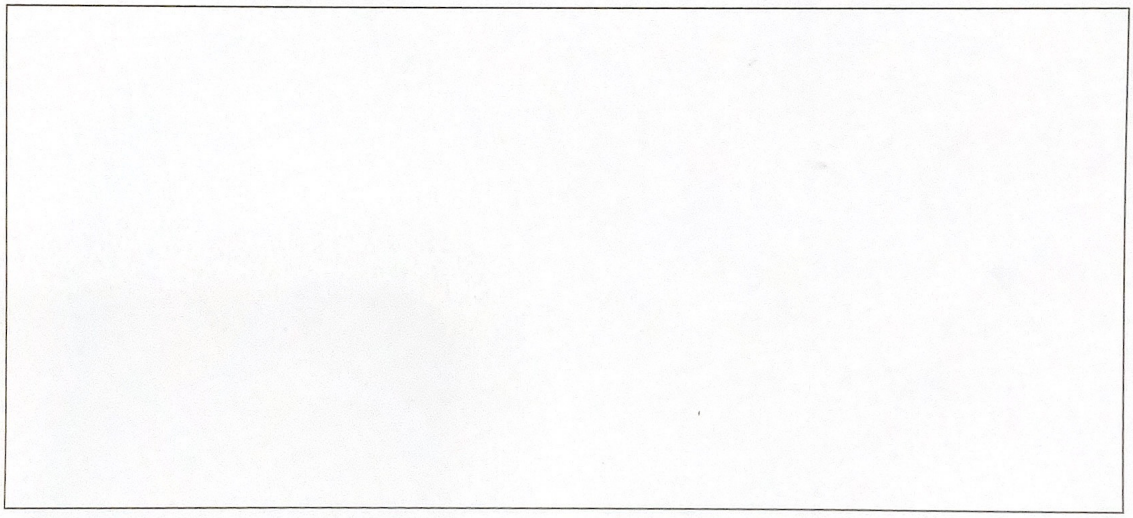
Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.



(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.



Só isso mesmo.

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES. 1 2 3 4 5 6

```
Nat: α → α
0 = 0
Sn = α                                   X

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

map f [] = []
map f (x:xs) = map f x ++ map f xs        X

(++) : List α → List α → List α
(++) 0 0 = 0
(++) List n List ns = List (n:ns)        X
```

```
length: List α → List α → Nat
length [] = 0                            X
length (x:xs) = length x + length xs

Sum: List α → α                          X
Sum [] = 0
Sum (x:xs) = x + Sum xs                  ✓
```

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = (d · n) + (d · m)                          ✓
map f (xs ++ ys) = map f xs ++ map f ys                  ✓
product (xs ++ ys) = product xs · product ys             ✓
reverse (xs ++ ys) = reverse xs + reverse ys             X
length (xs ++ ys) = length xs + length ys                ✓
length (map f xs) = length xs                            ✓
sum (map (+ k) ns) = (k · ns) + sum ns                   X
```

type error

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE map f (xs ++ ys) = map f xs ++ map f ys .

Por indução

Base: [map f ([] ++ ys)]

Calculemos:

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

~~= map~~

= [] ++ map f ys

~~PASSO INDUTIVO:~~ PASSO INDUTIVO:

(∀K)[map f (Ks ++ ys)] = map f Ks ++ map f ys

DEMONSTRAÇÃO DE length (map f xs) = length xs .

Por indução

Base:

Calculemos:

~~length (map f []~~

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

= 0 (length.1) ? ~

Passo indutivo:

(∀K)[length (map f Ks)] = length [Ks] (map.1)

= length Ks (length.2)

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.

Por indução:
Base: C
Calculemos:
Fork = 0

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

height:
l (fork == 0) = 0
l (fork == n) = n

tips:
l (fork == 0)
l tip = 0
l tip = 1
l (fork · n) / 2

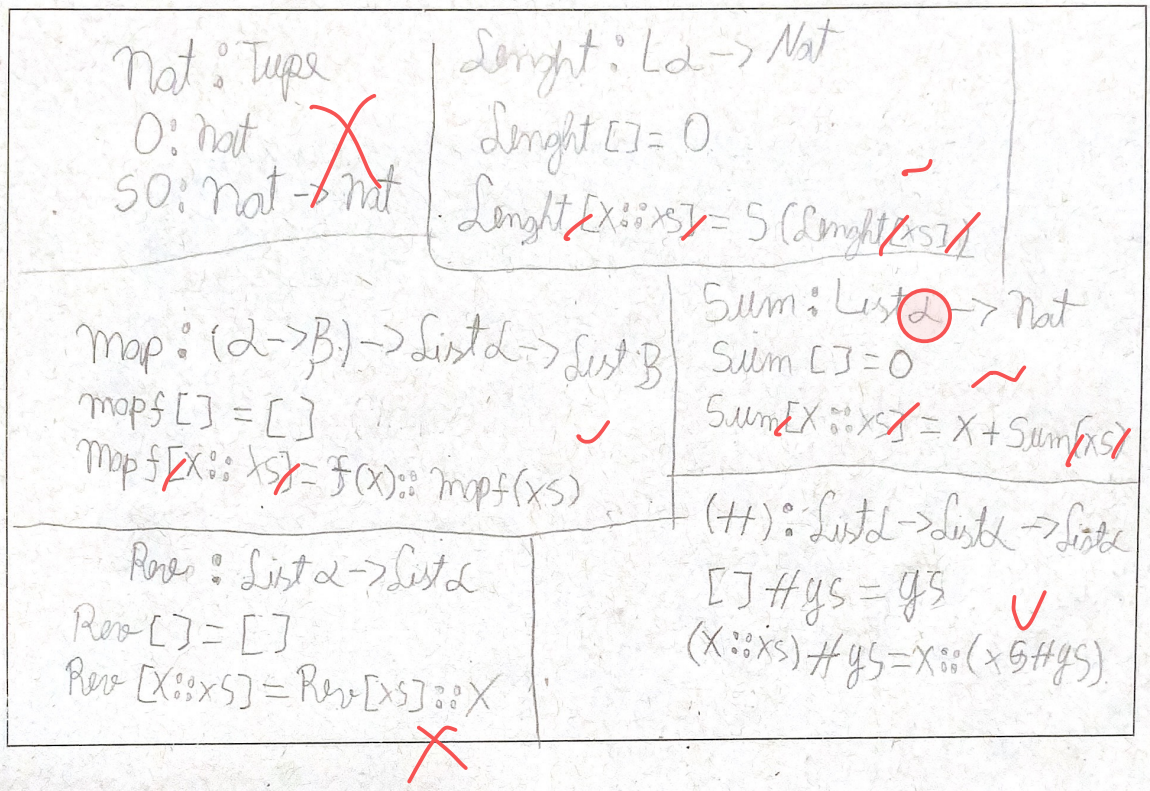
tips:
l (fork == 0)
l tip = 0
l tip = 1
l (fork · n) / 2
l tip = 0
l tip = n
l tip = 2n

Só isso mesmo.

(++):

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.



(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

1 `d · (n + m) =` (d·n) + (d·m)
2 `map f (xs ++ ys) =` map f (xs) ++ map f (ys)
3 `product (xs ++ ys) =` Product xs . Product ys
4 `reverse (xs ++ ys) =` Rev ys ++ Rev xs
5 `length (xs ++ ys) =` Lenght xs + Lenght ys
6 `length (map f xs) =` Lenght (Rev (map f xs))
7 `sum (map (+ k) ns) =` Sum (Rev (map (+k) ns))

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE $d.(n+m) = (d.n)+(d.m)$ (3)

Seja d: Nat
Por indução em d

Caso d = 0 -- $0.(n+m) = (0.n)+(0.m)$

Calculamos
$0.(n+m)$
$= (n+m).0$ [(.) com $0\ (n+m)$]
$= 0$ [(.).1]

$(0.n)+(0.m)$
$= (n.0)+(0.m)$ [(.) com $0\ n$]
$= (n.0)+(m.0)$ [(.) com $0\ m$]
$= 0+(m.0)$ [(.).1 $n := n$]
$= 0+0$ [(.).1 $n := m$]
$= 0$ [(+) id] Imediato

Caso $d = Sd'$
Seja d' t.q $d'.(n+m) = (d'.n)+(d'.m)$
Calculamos
$Sd'.(n+m)$
$= (n+m).Sd'$ [(.) com]
$= ((n+m).d')+(n+m)$ [(.).2]
$= (d'.(n+m))+(n+m)$ [(.) com $(n+m)\ d'$]
$= (d'.n)+(d'.m)+(n+m)$ [H.I]
$(Sd'.n)+(Sd'.m)$
$= (n.Sd')+(m.Sd')$ [(.) com $Sd'\ n$, (.) com $Sd'\ m$]

Caso $d = Sd'$
Seja d' t.q $d'.(n+m) = (d'.n)+(d'.m)$
Calculamos
$Sd'.(n+m)$
$= (n+m).Sd'$ [(.) com]
$= ((n+m).d')+(n+m)$ [(.).2]
$= (d'.(n+m))+(n+m)$ [(.) com $(n+m)\ d'$]
$= (d'.n)+(d'.m)+(n+m)$ [H.I]
$(Sd'.n)+(Sd'.m)$
$= (n.Sd')+(m.Sd')$ [(.) com $Sd'\ n$, (.) com $Sd'\ m$]

Imediato

DEMONSTRAÇÃO DE $Rev(xs\#ys) = Rev\,ys \# Rev\,xs$ (4)

Seja xs: List A
Por indução em xs

Caso [] --
Calculamos
$Rev([]\#ys)$
$= Rev(ys)$ [(#).1]

$Rev\,ys \# Rev\,[]$
$= Rev\,ys \# []$ (Rev.1)
$= []\#Rev\,ys$ [[] id]
$= Rev\,ys$ [(#).1]
Imediato

Caso $xs' = x :: xs'$
Seja xs' t.q $Rev(xs'\#ys) = Rev\,ys \# Rev\,xs'$
Calculamos
$Rev((x::xs')\#ys)$
$= Rev(x::(xs'\#ys))$ [(#).2]
$= Rev(xs'\#ys) :: x$ [Rev.2]
$= (Rev\,ys \# Rev\,xs') :: x$ [H.I]

$Rev\,ys \# Rev(x::xs')$
$= Rev\,ys \# (Rev\,xs' :: x)$ [Rev.2]

Imediato

LEMMA iRi

Q: $[] \# xs = xs \# []$ ([] ≈ xs JDE)

Seja xs: List A

Por indução em xs

Caso []

Calculamos

[] # [] = [] ((#).1) refl!

[] # [] Imediato

Caso x :: xs'

Seja xs' t.q [] # xs' = xs' # []

calculamos

[] # (x :: xs') = (x :: xs') [(#).1] ✓

(x :: xs') # [] = x :: (xs' # []) [(#).2] ✓

= x :: ([] # xs') [H.i] ✓

= x :: xs' [(#).1] Imediato ✓

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
Data Nat Where
    0 : Nat
    S : Nat -> Nat

Data List α Where
    Nil : List α
    Cons : α -> List α -> List α

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

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

(++) : List α -> List α -> List α
[] ++ xs = xs
(x:xs) ++ ys = x : (xs ++ ys)

sum : List α -> Nat
sum [] = 0
sum (x:xs) = x + sum xs
```

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) =          dn + dm
map f (xs ++ ys) =     map f xs ++ map f ys
product (xs ++ ys) =   product xs · product ys
reverse (xs ++ ys) =   reverse ys ++ reverse xs
length (xs ++ ys) =    length xs + length ys
length (map f xs) =    length xs
sum (map (+ k) ns) =   k + sum ns
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

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

Sejam xs, ys : list α.
Por indução.
Caso []: --- xs = []
calculamos (para os dois lados):

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

?!

Caso (x : xs):
Seja xs : list α tal que length (xs ++ ys) = length xs + length ys. (HI)
calculamos (para os dois lados):

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

length (x : xs) + length ys = S (length xs) + length ys [len.2]
= length ys + S (length xs) [(+).com]
= S (length xs + length ys) [(+).2]
= S (length xs + length ys) [(+).com]
= S (length (xs ++ ys)) [(H.I)]

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

Sejam xs, ys : list α.
Por indução.
Caso []: --- xs = []
calculamos (para ambos os lados):

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

Caso (x : xs):
Seja xs tal que map f (xs ++ ys) = map f xs ++ map f ys
calculamos (para ambos os lados):

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

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

ass!

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.

$$\frac{(\forall x)[P(\mathrm{Tip}\ x)] \qquad (\forall l : \mathrm{Tree}\ a\ b)\,[P(l) \wedge P(r) \Rightarrow (\forall y)[P(\mathrm{Fork}\ y\ l\ r)]}{(\forall t)[P(\mathrm{Tree}\ a\ b)]}$$

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
data Nat where:
  O :: Nat
  S :: Nat -> Nat

sum :: List Nat -> Nat
sum Nil = 0
sum (x : xs) = x + sum xs

data List a where:
  Nil :: List a
  Cons :: a -> List a -> List a

product :: List Nat -> Nat
product Nil = 1
product (x : xs) = x * product xs

map :: (a -> b) -> List a -> List b
map _ Nil = Nil
map f (x : xs) = f x : map f xs

filter :: (a -> Bool) -> List a -> List a
filter _ Nil = Nil
filter p (x : xs) = if then else (p x) (x : filter p xs) (filter p xs)
```

(21) **G**

Complete ~~até 8~~ das equações seguintes **com algo interessante:**[4]

```
d . (n + m) = d . n + d . m
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs * product ys
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = sum ns + k * length ns
```

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

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE $\delta \cdot (n+m) = \delta n + \delta m$.

Por indução no m

Base $m = 0$

Calculamos $\delta n + \delta \cdot 0$

$= \delta n + 0$ [(·).1]

$= \delta n$ [(+).1]

Calculamos $\delta (n+0)$

$= \delta \cdot n$ [(+).1]

Imediata

Passo Indutivo

Seja k t.q. $\delta (n+k) = \delta n + \delta k$ (H.I.)

Calculamos $\delta (n+Sk)$

$= \delta \cdot S(n+k)$ ✓ [(+).2 $n := n$ $m := k$]

$= \delta \cdot (n+k) + \delta$ ✓ [(·).2 $n := \delta$ $m := n+k$]

$= (\delta n + \delta k) + \delta$ [H.I.]

-- Continua em (2)

-(2)

Calculamos $\delta n + \delta Sk$

$= \delta n + (\delta k + \delta)$ [(·).2 $n := \delta$ $m := k$]

Imediata ..e..?

DEMONSTRAÇÃO DE _____ .

(19) **I**

Para qualquer α : Type, definimos o tipo

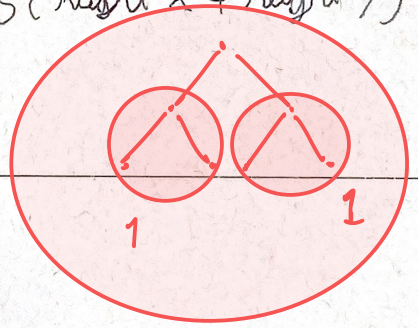
```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

```
tips :: Tree a b → Nat
tips (Tip _) = 1
tips (Fork x y) = tips x + tips y
```

```
height :: Tree a b → Nat
height Tip _ = 0
height Fork x y = S (height x + height y)
```



$2 \neq 1+1+1$

Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
data Nat where
  O : Nat
  S : Nat → Nat

data List α where
  [] : List α
  (::) : α → List α → List α

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

length : List α → Nat
length [] = O
length (x::xs) = S (length xs)

reverse : List α → List α
reverse [] = []
reverse (x::xs) = reverse xs ++ [x]

(++) : List α → List α → List α
[] ++ xs = xs
(x::xs) ++ ys = x :: (xs ++ ys)

sum : List Nat → Nat
sum [] = O
sum (x::xs) = x + sum xs

product : List Nat → Nat
product [] = 1
product (x::xs) = x · product xs
```

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = d·n + d·m
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = (product xs) · (product ys)
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = k · (length ns) + sum ns
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palàvras diferentes.

DEMONSTRAÇÃO DE $len\ (xs ++ ys) = len\ xs + len\ ys$.

```
-- (∀xs, ys : La)[len (xs ++ ys) = len xs + len ys]
Sejam xs, ys : La.
Por indução no xs.
Caso []:
  calculamos:
    len ([] ++ ys)
      = len ys      [(++).1]
    len [] + len ys
      = 0 + len ys  [(len).1]
      = len ys      [(+).1]
Caso (x :: xs'):
  calculamos:
    len ((x :: xs') ++ ys)
      = len (x :: (xs' ++ ys))   [(++).2 x := x, xs := xs', ys := ys]
      = S (len (xs' ++ ys))      [(len).2 x := x, xs := xs']
      = S (len xs' + len ys)     [H.I]
      > S (len xs + len xs')     [(+)-com]
      = len ys + S (len xs')     [(+).2]
```

```
len (x :: xs') + len ys
  = S (len xs' | + len ys   [(len).2]
  = len ys + S (len xs')    [(+)-com]
```

DEMONSTRAÇÃO DE $map\ f\ (xs ++ ys) = map\ f\ xs ++ map\ f\ ys$

```
-- (∀xs, ys : La)[map f (xs ++ ys) = map f xs ++ map f xs]
Sejam xs, ys : La.
Por 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)  [H.I]
```

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

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.

$$\frac{(\forall b : \beta)[\varphi(\mathit{Tip}\ b)] \qquad (\forall a : \alpha)(\forall t, t' : \mathit{Tree}\ \alpha\ \beta)[\varphi(t) \wedge \varphi(t') \Rightarrow \varphi(\mathit{Fork}\ a\ t\ t')]}{(\forall t : \mathit{Tree}\ \alpha\ \beta)[\varphi(t)]}\ \mathit{Ind}^{\mathit{Tree}}_{\varphi}$$

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

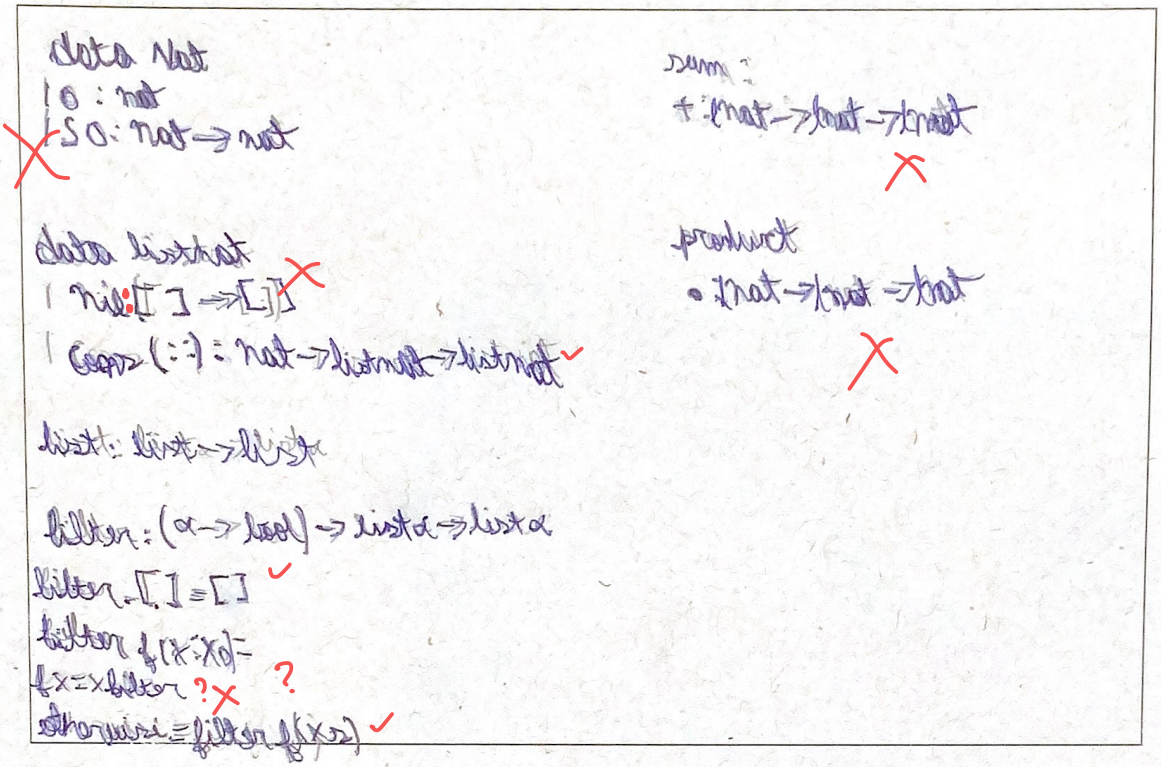
```
height : Tree α β → Nat
height (Tip _) = 1
height t =

tips : Tree α β → Nat
tips (Tip _) = 1
tips t =
```

Só isso mesmo.

(24) A

Não é legível caneta em cima de lapis!

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.



(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

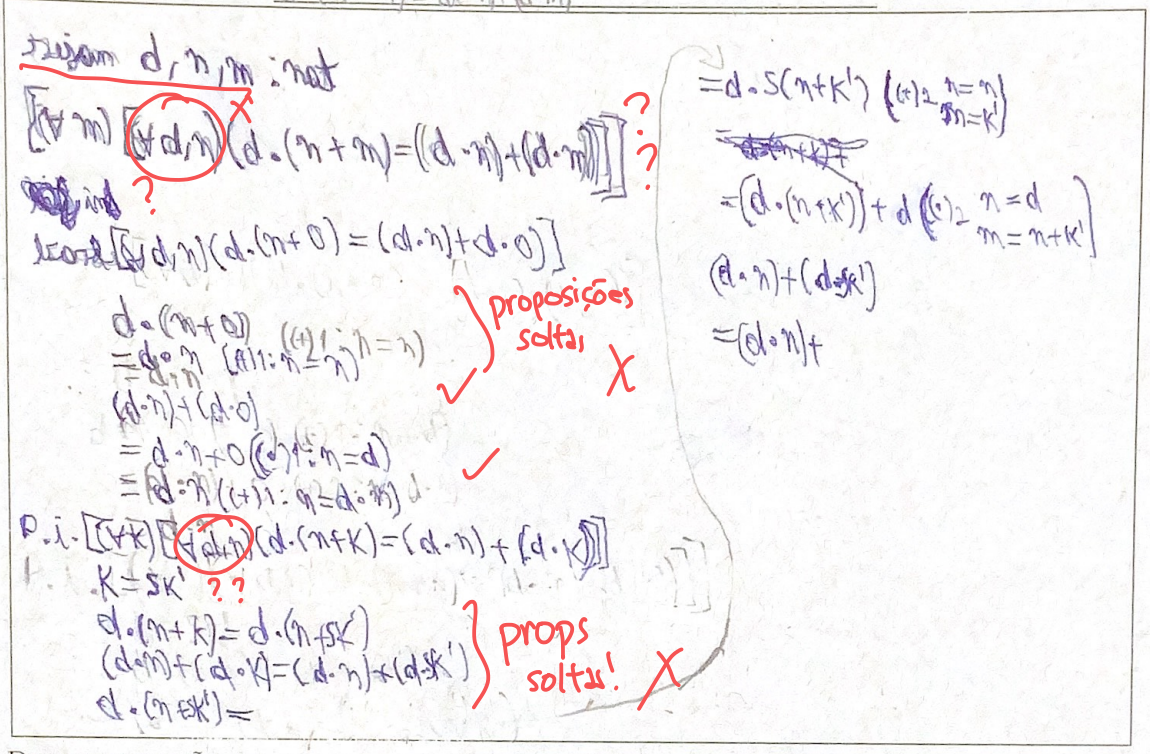
```
d · (n + m) = (d · n) + (d · m)
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = xs · ys
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = k · ns
```

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

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE $d \cdot (n + m) = (d \cdot n) + (d \cdot m)$



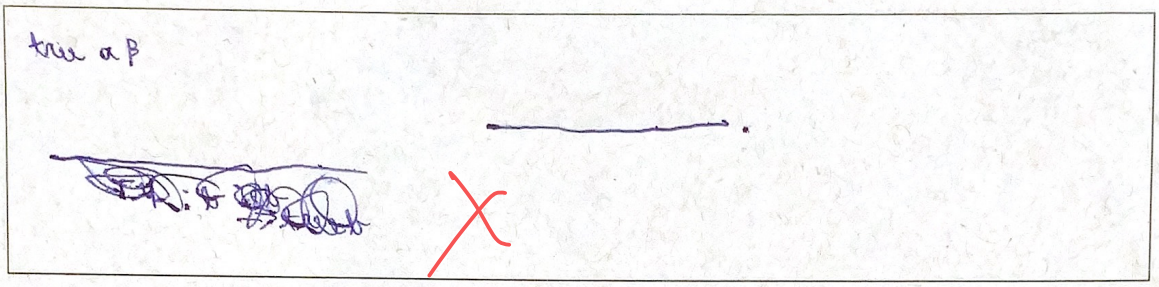
DEMONSTRAÇÃO DE _____

(19) **I**

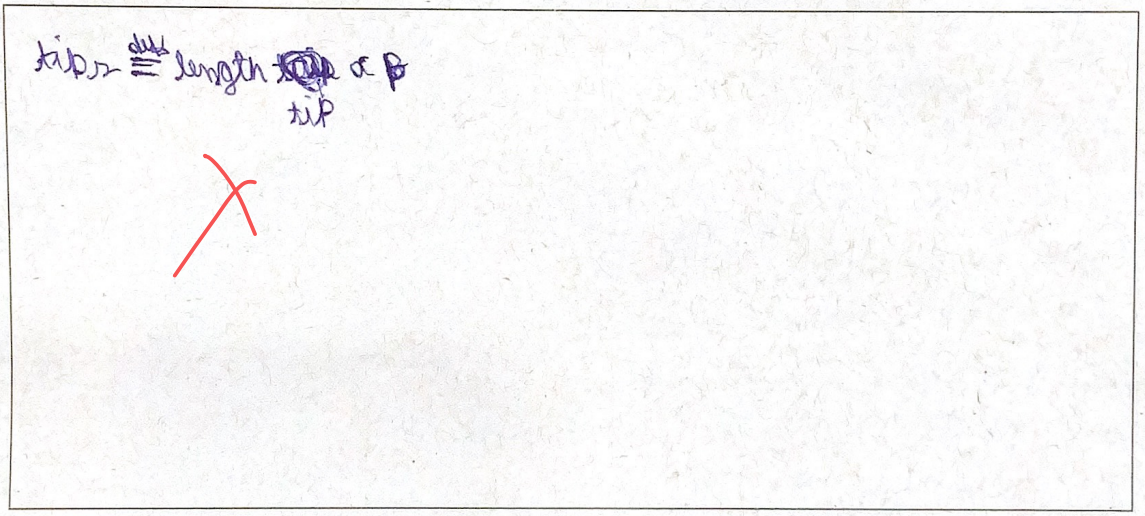
Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.



(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.



Só isso mesmo.

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

List: ?

(NAT)
Data Nat : type
0 : Nat
S : Nat -> Nat ✓

(list)
data List α : type
Nil : List α
Cons : α -> List α -> List α ✓

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

(filter)
filter : (α -> bool) -> L α -> L α
filter _ [] = []
filter f (x :: xs) = ✓
| f(x) = x : filter f xs
| otherwise = filter f xs

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

(++) : L α -> L α -> L α
(++) xs [] = xs ✓
(++) (x :: xs) ys =
x :: ((++) xs ys)

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = ~~d*n + d*m~~ ~~(n+m)~~ + ~~(d · (n + pred m))~~ (n+pred m)+(d.(n+pred m))
map f (xs ++ ys) = map f xs ++ map f ys ✓
product (xs ++ ys) = product xs (*) product ys ~
reverse (xs ++ ys) = reverse ys (++) reverse xs ~
length (xs ++ ys) = length xs + length ys ~
length (map f xs) = length xs ✓
sum (map (+ k) ns) = sum ns + (length ns) (*) k) ~
```

não inventa sintaxe!.

[4] DEFINIÇÃO. Chamamos algo de *interessante* ssc Thanos acha tal algo interessante.

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE d . (n + m) = (n + pred m) + (d . (n + pred m)

pred : Nat → Nat
pred 0 = 0
pred S n = n

Demonstração
??? -- pred m --
basta d.(n+m) ser igual a (n + pred m) + (d . (n + pred m))
Calculamos
d . (n+m)
≡ d . (S (n + pred m)) [(+).2 com n := n, m := pred m]
= d (n + pred m) + (d . (n + pred m)) [(*).2 com n := d, m := n + pred m)

X

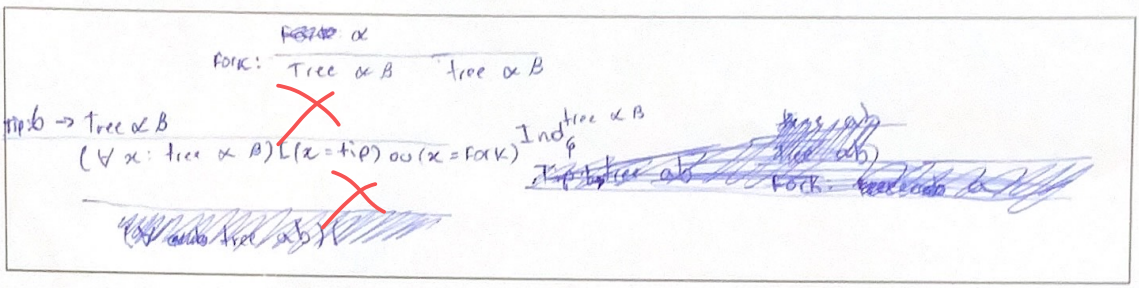
DEMONSTRAÇÃO DE _____ .

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.



(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
data Nat where
  O : Nat
  S : Nat -> Nat

length : List α -> Nat
length [] = O
length (x :: xs) = S (length xs)

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

data List α where
  Nil : List α
  Cons : α -> List α -> List α

reverse : List α -> List α
reverse [] = []
reverse (x :: xs) = reverse xs ++ [x]

(++) : List α -> List α -> List α
[] ++ ys = ys
(x :: xs) ++ ys = x :: (xs ++ ys)

sum : Num α => List α -> α
sum [] = 0
sum (x :: xs) = x + sum xs
```

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = d·n + d·m
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs · product ys
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = sum ns + k · length ns
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE $length(xs ++ ys) = length\ xs + length\ ys$

$(\forall xs, ys)[length(xs ++ ys) = length\ xs + length\ ys]$
Sejam xs e ys listas.
Por indução no xs
Caso xs = [ ]
length ([ ] ++ ys) = length ys [(++).1]
= length ys + 0 [← (+).1]
= 0 + length ys [(+) com]
= length [ ] + length ys [← length.1]
Caso xs = (z :: zs)
length ((z :: zs) ++ ys) = length (z :: (zs ++ ys)) [(++).2]
= S(length (zs ++ ys)) [length.2]
= S(length zs + length ys) [H.I]
= S(length ys + length zs) [(+) com]
= length ys + S(length zs) [← (+).2]
= S(length zs) + length ys [(+) com]
= length (z :: zs) + length ys [← length.2]

DEMONSTRAÇÃO DE $Sum(map(+k)\ ns) = Sum\ ns + k \cdot length\ ns$

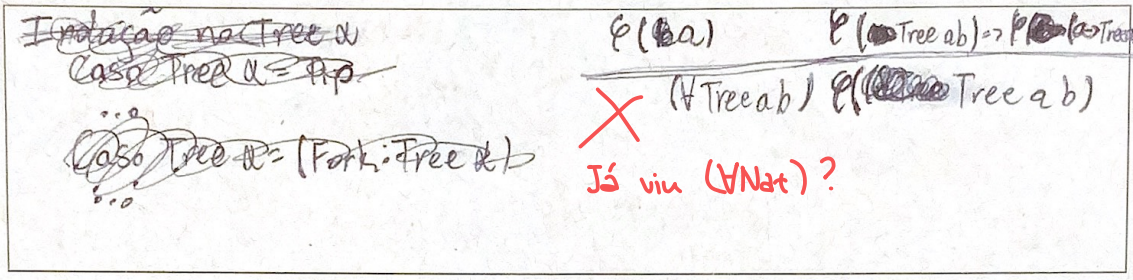
$(\forall ns)(\forall k)[sum(map(+k)\ ns) = sum\ ns + k \cdot length\ ns]$
Seja ns uma lista
Seja K um natural
Por indução no ns
Caso ns = [ ]
sum (map (+k) [ ]) = sum [ ] [map.1]
= sum [ ] + 0 [← (+).1]
= sum [ ] + k · 0 [← (·).1 n=k]
= sum [ ] + k · length [ ] [← length.1]
Caso ns = (x :: xs)
sum (map (+k) (x :: xs)) = sum ((x+k) :: map (+k) xs) [map.2]
= (x+k) + sum (map (+k) xs) [sum.2]
= (x+k) + (sum xs + k · length xs) [H.I]
= x + k + sum xs + k · length xs [(+)-ass]
= x + sum xs + k · length xs + k [(+)-com] (quantos?)
= sum (x :: xs) + k · length xs + k [← sum.2]
= sum (x :: xs) + k · S(length xs) [← (·).2]
= sum (x :: xs) + k · length (x :: xs) [← length.2]

(19) **I**

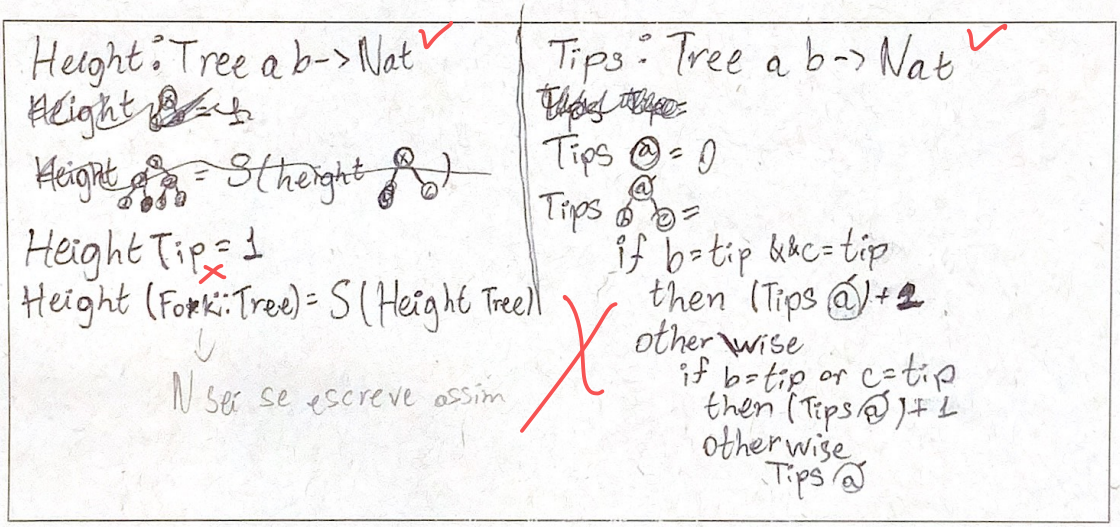
Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.



(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.



Só isso mesmo.

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

Data Nat
O : Nat ✓
S : Nat → Nat

Data List
Nil : [ ]
Cons : (x :: 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 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) = S (Length xs) ✓

(++) : List α → List α → List α
[ ] ++ ys = ys
(x :: xs) ++ ys = x :: (xs ++ ys) ✓

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = dn + dm ✓
map f (xs ++ ys) = map f xs ++ map f ys ✓
product (xs ++ ys) = product xs · product ys ✓
reverse (xs ++ ys) = reverse xs ++ reverse ys X
length (xs ++ ys) = length xs + length ys ✓
length (map f xs) = length xs ✓
sum (map (+ k) ns) = k + sum ns X
```

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

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE $len(xs \# ys) = len\ xs + len\ ys$.

Seja ys.
Por indução em xs ✓
Base:
calculamos:
$len([] \# ys)$
$= len\ ys\ [(\#).1\ ys]$ ✓
$len\ [] + len\ ys$
$= 0 + len\ ys\ [len.1]$
$= len\ ys + 0\ [(+)\text{-comm}\ 0\ len\ ys]$
$= len\ ys\ [(+).1\ len\ ys]$ ✓
Imediato.

Passo indutivo:
Suponha $len(ks \# ys) = len\ ks + len\ ys$. ✓
Seja K.
calculamos:
$len\ [(K::ks) \# ys)$
$= len(K::(ks \# ys))\ [(\#).2]$ ✓
$= S(len(ks \# ys))\ [(len).2]$ ✓
$= S(len\ ks + len\ ys)$ ~ [H.I]
$= S(len(K::ks) + len\ ys)$ X
$= len\ K::ks + len\ ys\ [(len).2]$
Imediato.

DEMONSTRAÇÃO DE $map\ f\ (xs \# ys) = map\ f\ xs \# map\ f\ ys$

Seja f.
Seja ys.
Por indução em xs: ✓
Base:
calculamos:
$map\ f\ ([] \# ys)$
$= map\ f\ ys\ [(\#).1]$ ✓
$map\ f\ [] \# map\ f\ ys$
$= map\ f\ ys\ [map.1]$ ✓
Imediato.

Passo indutivo:
Suponha $map\ f\ (ks \# ys) = map\ f\ ks \# map\ f\ ys$
Seja K

calculamos:
$map\ f\ (K::ks) \# map\ f\ ys$
$= map\ f\ ks \# map\ f\ ys\ [(\#).2]$??
$= map\ f\ (ks \# ys)\ [H.I]$
$= map\ f\ (K::ks \# ys)\ [(\#).2]$
Imediato.

? ?

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
data Nat                    data List α
  O : Nat                     [] : List α
  S : Nat -> Nat              (:) : α -> List α -> List α

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

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

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

(++) :: [α] -> [α] -> [α]
[] ++ ys = ys
(x:xs) ++ ys = x : (xs ++ ys)
```

(21) G

Complete até 8 das equações seguintes com algo interessante:[4]

```
d · (n + m) = d·n + d·m
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs · product ys
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = sum ns + K.length ns
```

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

(36) P

Escolha **até duas** das equações de G para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

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

Por indução em xs.

Base:

Calculamos:

length ([] ++ ys) = len ys [++.1] ✓

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

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

Passo Indutivo:

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

Calculamos:

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

= S (length (xs ++ ys)) [len.2] ✓

= S (len xs + len ys) [H.1] ✓

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

= len ys + S (len xs) [(+)-com] ✓

= S (len ys + len xs) [(+).2] ✓

= S (len xs + len ys) [(+)-com] ✓

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

Por indução em xs.

Base:

Calculamos:

map f ([] ++ ys) = map f ys [++.1] ✓

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

= map f ys [++.1] ✓

Passo indutivo:

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

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) [H.1] ✓

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

..e?

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.

Seja φ : Tree a b → Prop

$$\frac{(\forall x : b)[\varphi(\text{Tip } x)] \qquad (\forall x : a)(\forall l, r : \text{Tree } a\ b)[\varphi(l) \wedge \varphi(r) \Rightarrow \varphi(\text{Fork } x\ l\ r)]}{(\forall t : \text{Tree } a\ b)[\varphi(t)]}$$

☺

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

```
~~heigth :: ~~
heigth :: Tree a b → Nat
heigth (Tip n) = ~~S~~0
heigth (Fork n l r) = n
```

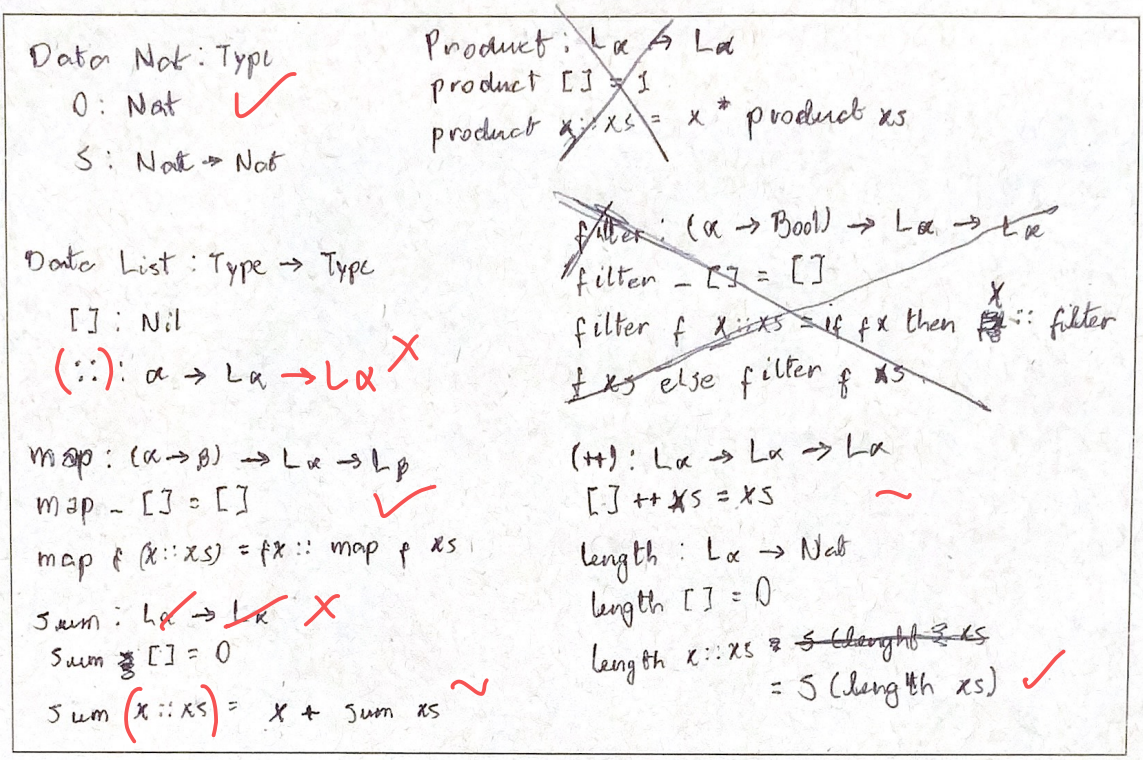
TYPE ERROR

```
tips :: Tree a b → Nat
tips (Tip n) = SSO
tips (Fork n l r) = SSO * n
```

Só isso mesmo.

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.



(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = d·n + d·m
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = product xs + product ys
reverse (xs ++ ys) = rev xs ++ rev ys
length (xs ++ ys) = len xs + len ys
length (map f xs) = len xs
sum (map (+ k) ns) = Sum ns + len * K
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE `map f (xs ++ ys) = map f xs ++ map f ys`.

Por indução em xs.

Caso []:

| Calculamos:

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

= [] ++ map f ys [(++).1 map f ys] ✓

= map f [] ++ map f ys [(map).3 f []] ✓

Caso x::xs:

| Suponha map f (xs ++ ys) = map f xs ++ map f ys : h.i.

| Calculamos:

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

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

= map f x::xs ++ map f ys [x]

∎

DEMONSTRAÇÃO DE `length (map f xs) = lenght xs`.

Por ind. em xs.

Caso []:

| Calculamos:

| length (map f []) = len [] [map.3 f] ✓

Caso x::xs:

| Suponha length (map f xs) = len xs : h.i

Calculamos:

| Len (map f (x::xs)) = Len (fx :: map f xs) [map.2 ✓

= S (Len (map f xs)) [length.2 ✓

= S (len xs) [h.i] ✓

= Len (x::xs) [length.2 ✓

∎

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
Data Nat : Type
     0 : Nat
     S : Nat → Nat

Data List α : Type
     Nil : List α
     Cons : α → List α → List α

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

Length : List α → Nat
     length [] = 0
     length (x :: xs) = S (length xs)

(++) : List α → List α → List α
     (++) xs [] = xs
     (++) xs ys = x :: (xs ++ (y :: ys))

reverse : List α → List α
     reverse [] = []
     reverse [x] = [x]
     reverse (x :: xs) = reverse xs :: x
```

por que prefixo??

List : ?

type error

(21) **G**

Complete até 8 das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = dm + dm
map f (xs ++ ys) = f x :: map (xs ++ (y :: ys))
product (xs ++ ys) =
reverse (xs ++ ys) = reverse (xs ++ (y :: ys)) :: x
length (xs ++ ys) = length xs + length ys
length (map f xs) = S (length xs)
sum (map (+ k) ns) =
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE length (map f xs) = S (length xs) .

Seja xs : List α

Calculamos:

length (map f xs)

= length (fx :: map xs) [map . 1]

= S (length xs) [length . 2]

DEMONSTRAÇÃO DE d (n + m) = dn + dm .

Sejam n e m : Nat

Por indução.

Base:

Calculamos:

0 · n + 0 · m

= 0 + 0 · m [(·) . 1]

= 0 + 0 [(·) . 1]

= 0 [(+) . 1]

0 · (n + m)

= 0 [(·) . 1]

Passo indutivo:

Seja k tal que k (n + m) = kn + km.

Calculamos:

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

map $\overset{def}{\Leftrightarrow} \forall(xs, ys)[\psi(xs) \Rightarrow \varphi(ys)]$
filter $\overset{def}{\Leftrightarrow} \forall(xs)[\psi(xs) \Rightarrow x]$

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = (d · n) + (d · m)
map f (xs ++ ys) = map(f xs -> ys)
product (xs ++ ys) = Prod(xs -> ys)
reverse (xs ++ ys) = rev(xs -> ys)
length (xs ++ ys) = L(xs -> ys)
length (map f xs) = L(map(f xs))
sum (map (+ k) ns) = Sum(map(_+k) -> ns)
```

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

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE Map f (xs ++ ys) .

Map : [Nat] -> [Nat]

- []
- xs -> ys

DEMONSTRAÇÃO DE _____ .

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree $\alpha.\beta$.
RESPOSTA.

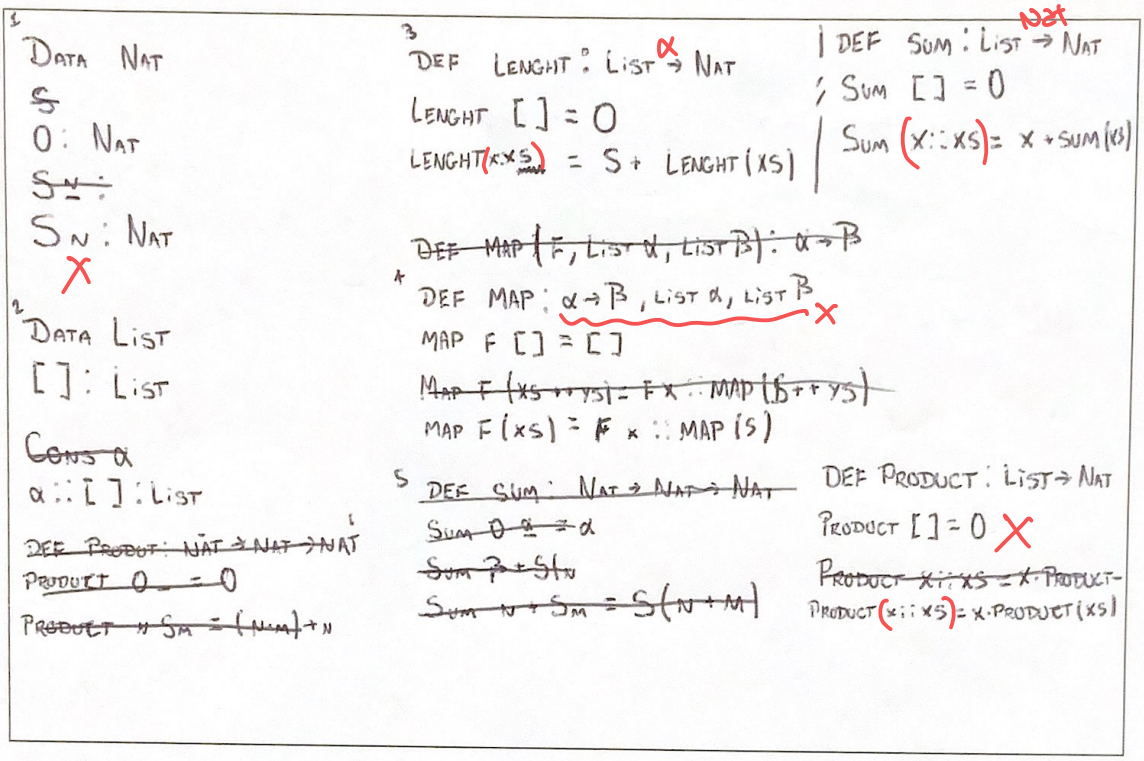
$$\frac{\varphi(x) \quad \varphi(\beta)}{\text{Tree } \alpha\ \beta}\ \text{IND}_{\varphi}^{\text{Tree } \alpha\ \beta}$$

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

Só isso mesmo.

(24) **Aviso:** matemática é case sensitive: (b ≠ B)
(...e até font-sensitive: a ≠ a)

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.



(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = D · N + D · M
map f (xs ++ ys) = F x :: MAP F (S ++ S)
product (xs ++ ys) = PRODUCT x S :: PRODUCT (S++S)
reverse (xs ++ ys) = REVERSE x Y :: REVERSE (S++S)
length (xs ++ ys) = SS + LENGTH (S++S)
length (map f xs) = LENGTH ( F x :: MAP F S)
sum (map (+ k) ns) = SUM (+ K) N + MAP (+K) S
```

Confundiu demais a notação:
xs é um nome só; não é x·s

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

(36) P

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE D · (N + M) = DN + DM .

SEJA D, N, M : ~~NAT T.0~~. NAT

CALCULAMOS:

D(N+M) =

= D·N + D·M (·).COM

IMEDIATO. X

DEMONSTRAÇÃO DE MAP F (XS ++ YS) = FX∀. MAP F (S ++ S) X

SEJA F: α → β

SEJA XS: LIST α

SEJA YS: LIST β

CALCULAMOS:

XS = X::S (LIST.2)

~~CALCULAMOS:~~

APLICO F EM x PARA GANHAR F x

CALCULAMOS:

~~MAP~~ F YS = Y::S (LIST.2)

CALCULAMOS

MAP F (XS ++ YS) =

MAP F (X::S ++ Y::S) = (LIST.2) X

~~MA~~ F x MAP (S++S) (MAP.2)

IMEDIATO.

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree $\alpha.\beta$
RESPOSTA.

TIP: B → TREE α β FORK: A → TREE A B → TREE A B → TREE A B

TREE : ~~TREE~~ ~~NAT~~ (α) → ~~NAT~~ (β) → TREE

IND TREE α β
φ

X

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

```
DEF HEIGHT : TREE α β → NAT
HEIGHT TREE 0 _ = 0
HEIGHT TREE A _ = 1 + HEIGHT TREE (A-1) _
```

type errors!

```
DEF TIPS : TREE α β → NAT
TIPS TREE _ 0 = 0
TIPS TREE _ A = ?
```

Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
data Nat where
O : Nat
S : Nat → Nat

length : list α → Nat
len [] = 0
len (m :: m2) = S (len m2)

data list where
| Fil : list α
| Cons2 : Nat → list α → list α

Sum : L α → Nat
Sum [] = 0
Sum (m :: m2)
= m + Sum m2

(++) : l α → l α → l α
xs ++ [] = xs
xs ++ ys = S (x (xs :: ys))

map : (α → β) → [α] → [β]
map _ [] = []
map f (x :: x2) = f x :: (map f x2)
```

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = (Prod d . Prod m) + (Prod d . Prod m)
map f (xs ++ ys) = map F X2 ++ map F Y2
product (xs ++ ys) = Prod X2 . Prod Y2
reverse (xs ++ ys) = rev X2 ++ rev Y2
length (xs ++ ys) = len X2 + len Y2
length (map f xs) = len X2
sum (map (+ k) ns) = Sum m2
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE map f (x2 ++ y2) = map f x2 ++ map f y2

depois x2, y2 : l α ✓

indução no x2 ✓

caso [ ]

tá justificando o quê?!

Cale:

map f ([ ] ++ y2) [hip(1)] ✓

= map f (y2) [(++) . 1] ✓

map f [ ] ++ map f y2

= [ ] ++ map y2 [map.(1)]

= map + y2 [(++) . 1] ✓

caso (x :: x2)

Cale

map f ((x :: x2) ++ y2)

= map f (x :: (x2 ++ y2)) [?]

[map.2)

= f x :: (map f x2 ++ ✓

map f y2) [Hi]

= ~~f x :: map f~~

= (f x :: map f x2)

++ map f y2 [(++) . 2]

X ?

DEMONSTRAÇÃO DE X = ? . map f (x :: x2 ++

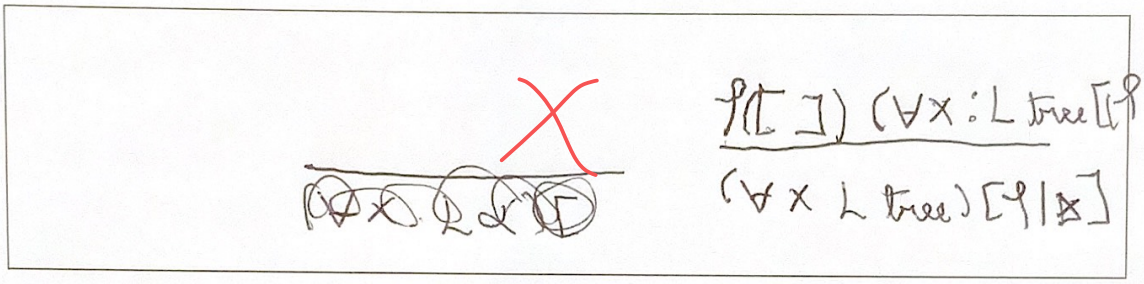
map f x y [map.(2)]

(19) **I**

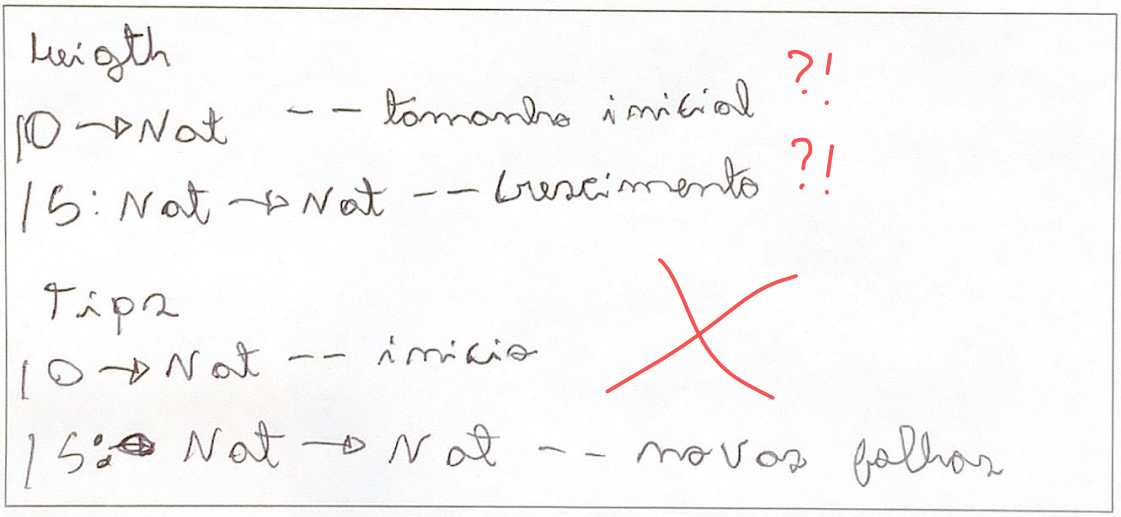
Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.



(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.



Só isso mesmo.

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

data Nat
  O : Nat
  S : Nat → Nat
	Sum : L α → α
	Sum [ ] = 0
	Sum (x : xs) = x + Sum xs

data list α
  Nil : list α
  Cons : α → Lα → Lα
	Product : L α → α
	Product [ ] = 1
	Product (x : xs) = x . Product xs

map : (α → β) → Lα → Lβ map _ [ ] = [ ] map f (x : xs) = f x : map f xs	Length : Lα → α Length [ ] = 0 Length (x : xs) = S (Length xs)

(21) G

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = d·n + d·m
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = Product xs ++ Product ys
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = k + (sum ns)
```

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

(36) P

Escolha **até duas** das equações de G para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

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

Seja f : α → β
Sejam xs, ys : List α
Indução em xs
Caso [ ] ✓
Calculemos:

isso não é um cálculo

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

Caso ~~indutivo~~ :

↳ concluiu algo trivial que já sabíamos (refl).

DEMONSTRAÇÃO DE sum (map (+K) ns) = K + (sum ns)

Seja K : α → β
Seja ns list α
Calculemos:
sum (map (+K) ns) =
✗ sum (K : ns) [(map).2]
K + (sum ns) [(sum).2]

(++) : L α → L α → L α
[ ] ++ ys = ys
(x : xs) ++ ys = x : (xs ++ ys)

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.

$$\frac{P(0) \qquad (\forall \alpha)[\exists K \Rightarrow P\alpha]}{(\forall \alpha)[P\alpha]}$$

(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

1) Data nat : type
   O : Nat
   S : Nat → Nat

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

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

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

5) Sum : [α] → Int
   Sum [] = error "empty list" e daí??
   ~~Sum [x] = x~~
   Sum (x::xs) = x + Sum xs

6) ~~Data List~~
   Para qualquer α : type
   data List α :
   [] : List α
   Cons : α → List α → List α

List : ?

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

d · (n + m) = dn + dm
map f (xs ++ ys) = (map f xs) ++ (map f ys)
product (xs ++ ys) = product xs • product ys
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = length xs + length ys
length (map f xs) = length xs
sum (map (+ k) ns) = Sum ns + (k • length ns)

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE length (map f xs) = length xs .

Por indução.

Base:

Seja f : (α → β)

Calculamos:

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

Imediato ■

Passo indutivo:

Seja ks : Lα t.q HI: (∀f)[length map f ks = length ks]

Seja k : α

Seja f : (α → β)

Calculamos:

length (map f (k::ks))

= length (f k :: map f ks) [(map).2]

= S (length map f ks) [length.2]

= S (length ks) [HI]

= length k::ks [length.2 k]

Imediato ■

DEMONSTRAÇÃO DE ~~length (xs ++ ys) = length xs + length ys~~ map f (xs ++ ys) = (map f xs) ++ (map f ys)

Por indução.

Base:

Seja ys : Lα

Seja f : (α → β)

Calculamos:

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

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

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

Imediato □

Passo indutivo:

Seja ks : Lα t.q HI : (∀f)[map f (ks ++ ys) = (map f ks) ++ (map f ys)]

Calculamos

map f ((k::ks) ++ ys)

= map f (k::(ks++ys)) [(++).2]

= f k :: map (ks ++ ys) [map.2]

= f k :: ((map f ks) ++ (map f ys)) [HI]

= (f k :: map f ks) ++ (map f ys) [(++).2]

= map f (k::ks) ++ map f ys [map.2]

Imediato ■

(++) : [α] → [α] → [α]

[] ++ xs = xs

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

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

Dados ??

0 : Nat

S : Nat → Nat

(21) G

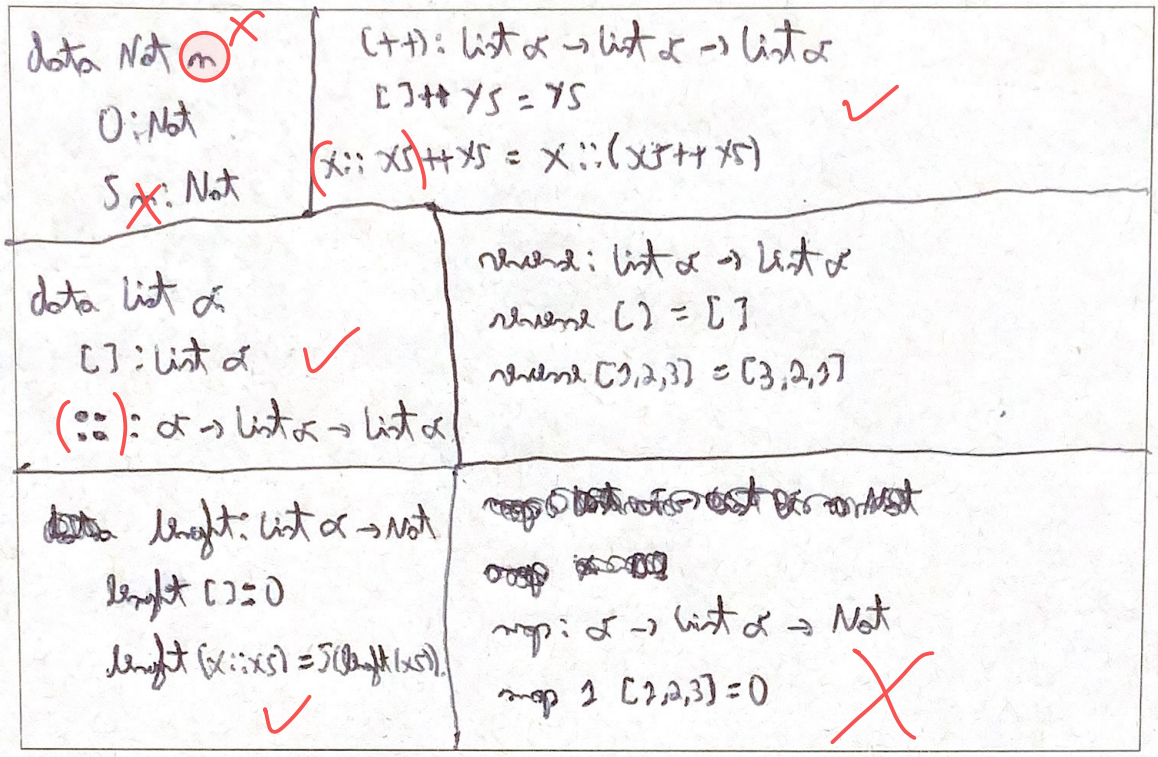
Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = d.n + d.m ✓
map f (xs ++ ys) = f xs ++ f ys ✗
product (xs ++ ys) = product xs * product ys ✓
reverse (xs ++ ys) = reverse ys ++ reverse xs ✓
length (xs ++ ys) = length xs + length ys ✓
length (map f xs) = length xs ✓
sum (map (+ k) ns) = k + (sum ns) ✗
```

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

(24) A

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.



(21) G

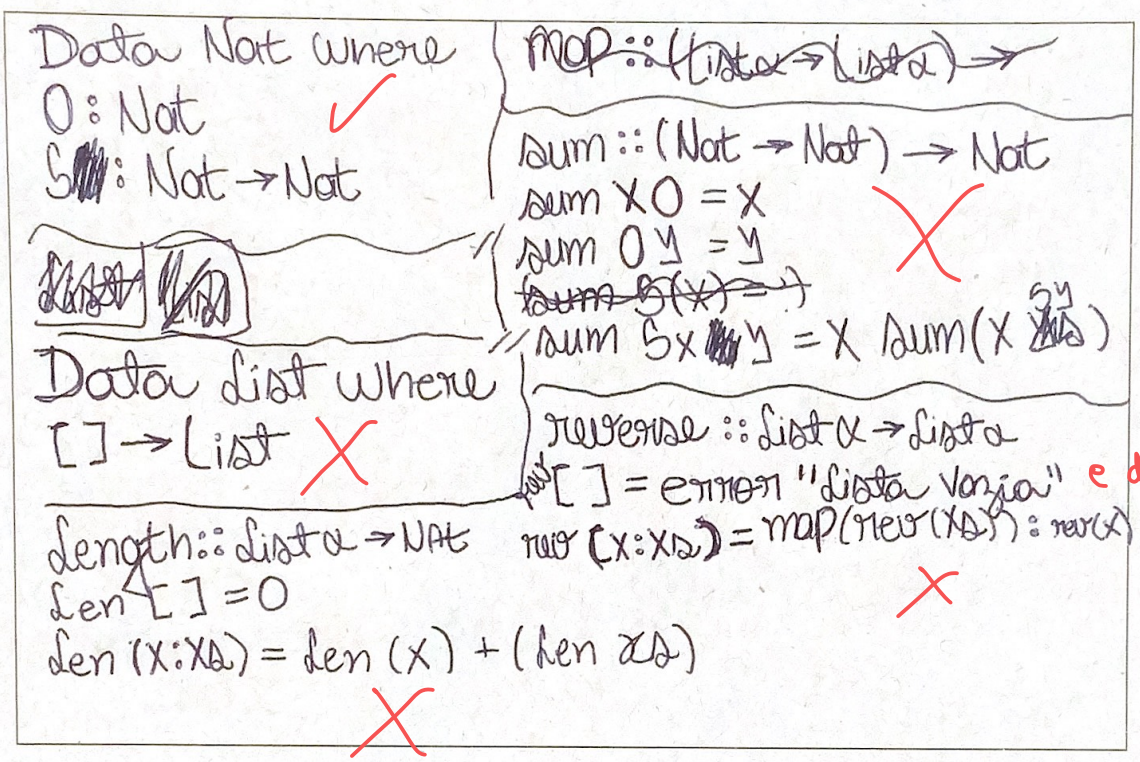
Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = (d · n) + (d · m)
map f (xs ++ ys) =
product (xs ++ ys) =
reverse (xs ++ ys) = (reverse ys) ++ (reverse xs)
length (xs ++ ys) = (length xs) + (length ys)
length (map f xs) = 1
sum (map (+ k) ns) =
```

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

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.



(21) **G**

Complete **até 8** das equações seguintes **com algo interessante**:[4]

```
d · (n + m) = dn + dm
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = Product xs · Product ys
reverse (xs ++ ys) = reverse ys ++ reverse xs
length (xs ++ ys) = (length (xs + ys)) Length xs + Length ys
length (map f xs) = Length (map f xs) + Length (map f ys)
sum (map (+ k) ns) =
```

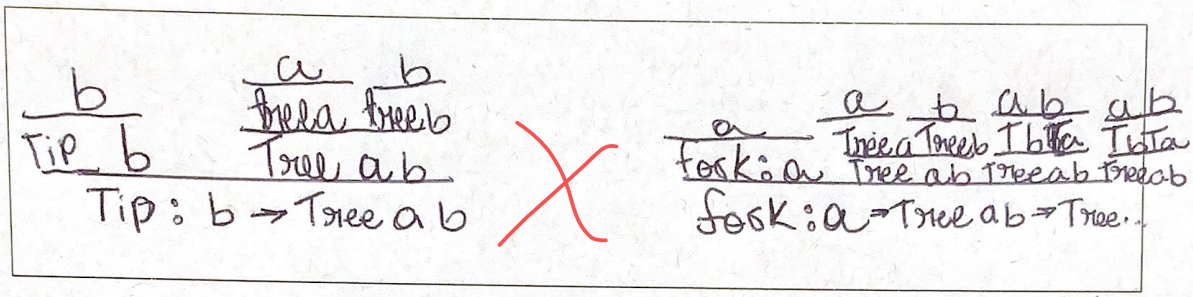
[4] DEFINIÇÃO. Chamamos algo de *interessante* sse Thanos acha tal algo interessante.

(19) **I**

Para qualquer α : Type, definimos o tipo

```
data Tree a b
  Tip  : b → Tree a b
  Fork : a → Tree a b → Tree a b → Tree a b
```

(9) **I1.** Escreva como regra de inferência o princípio da indução para o tipo Tree α.
RESPOSTA.



(10) **I2.** Defina a função *height* que calcula a altura de uma arvore e a função *tips* que conta quantas folhas tem numa arvore.
RESPOSTA.

```
height :: floos → Not
??
```

Só isso mesmo.

(24) **A**

Defina 6 dos: Nat, List, *map*, *filter*, *length*, *reverse*, (++), *sum*, *product*.
DEFINIÇÕES.

```
data Nat where
  O : Nat
  S : Nat → Nat
```

```
data List α where
  Nil : List α
  Cons : α → List α → List α
```

```
map : (α → β) → L α → L β
map _ [] = []
map f (x :: xs) = f x :: map f xs
```

```
filter : (α → Bool) → List α → List α
filter _ [] = []
filter p (x :: xs)
  | p x = x :: filter p xs
  | otherwise = filter p xs
```

```
length : L α → Nat
len [] = O
len (x :: xs) = S (len xs)
```

```
reverse : L α → L α
rev [] = []
rev (x :: xs) = rev xs ++ [x]
```

Type error!

(21) **G**

Complete **até 8** das equações seguintes **com algo interessante:**[4]

```
d · (n + m) = d · n + d · m
map f (xs ++ ys) = map f xs ++ map f ys
product (xs ++ ys) = prod xs · prod ys
reverse (xs ++ ys) = rev ys ++ rev xs
length (xs ++ ys) = len xs + len ys
length (map f xs) = len xs
sum (map (+ k) ns) =
```

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

(36) **P**

Escolha **até duas** das equações de **G** para demonstrar. As duas equações escolhidas precisam começar com palavras diferentes.

DEMONSTRAÇÃO DE map f (xs ++ ys) = map f xs ++ map f ys.

Sejam f : α → β. ✓
Sejam xs, ys : List α. ✓
Por indução em xs.
Caso xs = []:
Calculamos:
map f ([] ++ ys)
= map f ys [(++).1] ✓
map f [] ++ map f ys
= [] ++ map f ys [(map).1]
= map f ys ✓ [(++).1]

Caso xs = (x :: xs'):
Suponha map f (xs' ++ ys) = map f xs' ++ map f ys
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) [H.I.]
map (x :: xs') ++ map f ys
= (f x :: map f xs') ++ map f ys [(map)2] ✗
dss. !

(++) : L α → L α → L α
[] ++ ys = ys
(x :: xs) ++ ys = x :: (xs ++ ys)

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

Sejam xs, ys : List α.
Por indução em xs. ✓
Caso xs = []:
Calculamos:
len ([] ++ ys)
= len ys [(++).1] ✓
len [] + len ys
= 0 + len ys [(len).1]
= len ys + 0 [(+)-com]
= len ys [(+).1] ✓

Caso xs = (x :: xs')
Suponha len (xs' ++ ys) = len xs' + len ys.
Calculamos:
len ((x :: xs') ++ ys)
= len (x :: (xs' ++ ys)) ✓ [(++).2]
= S (len (xs' ++ ys)) ✓ [(len).2]
= S (len xs' + len ys) ✓ [H.I.]
len (x :: xs') + len ys
= S (len xs') + len ys ✓ [(len).2]
= len ys + S (len xs') ✓ [(+)-com]
= S (len ys + len xs') ✓ [(+).2]
= S (len xs' + len ys) [(+)-com] ✓